POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

STRUMENTI PER LA GESTIONE DELL'INCERTEZZA PER L'ANALISI DI PRESTAZIONI DEL SOFTWARE

Relatore: Prof.ssa Raffaela Mirandola

Co-relatore: Dott. Diego Perez-Palacin

Tesi di Laurea di:

Stefano Solimito, matricola 786975

Anno Accademico 2013 - 2014

Sommario

Questa tesi nasce dall'esigenza di affrontare il problema dell'incertezza nell'ambito dei modelli a rete di code. Per svolgere le analisi ho utilizzato un algoritmo chiamato MUSE [7], il quale svolge il rulo di guida nel processo di individuazione e gestione del fenomeno. Durante la fase di ricerca dell'incertezza ho utilizzato una tassonomia che ne descrive una possibile classificazione, grazie alla quale ho potuto fornire dei test mirati a verificare le singole tipologie di incertezza al fine di classificare il fenomeno rilevato. Successivamente ho poi proposto una serie di tecniche utili alla gestione del problema, mediante la realizzazione di modelli delle richieste inviate dai client al server del sistema considerato. Al fine di applicare i metodi descritti, ho realizzato un software in grado di mettere a disposizione tutte le funzionalità necessarie ad individuare il tipo di incertezza presente nei dati e successivamente gestirla mediante la computazione di modelli. I dati utilizzati per queste analisi automatizzate sono i dati sull'utilizzo del *webmail* server dell'università di Saragozza e dei server di Wikipedia. Grazie ai modelli realizzati mediante il programma è stato possibile ottenere previsioni sul tempo di risposta del server considerato, utilizzando formule appartenenti alla teoria delle code. Questi valori sono stati poi confrontati con quelli forniti da un metodo simulativo, più preciso ed affidabile ma computazionalmente più dispendioso.
Durante lo svolgimento del progetto ho inoltre proposto un approccio alternativo al problema cercando di utilizzare delle reti neurali come modello, volendo con esse realizzare predizioni del tempo di risposta di un ipotetico server.

Indice

Capitolo 1

Introduzione

Questo progetto di tesi va a trattare il problema dell'incertezza nell'ambito dell'ingegneria del software per quanto concerne l'analisi delle prestazioni. Il software al giorno d'oggi ha una importanza fondamentale sia nella vita quotidiana che in aree più critiche, dove implementazioni inaccurate possono provocare conseguenze impreviste e spesso dannose. Volendo evitare effetti indesiderati è quindi un obiettivo primario, durante la fase di sviluppo di una applicazione software, poterne verificare la qualità. Parlando di qualità di un software non ci si limita a considerare solo il suo corretto funzionamento ma anche il rispettare determinati vincoli sulla sua struttura interna e sulle prestazioni desiderate. Esistono vari indicatori, detti parametri di qualità, che consentono di valutare la bontà del software realizzato. Questi valori vengono misurati utilizzando diverse tecniche e tra più utilizzati per l'analisi delle prestazioni di un programma troviamo: correttezza, affidabilità, robustezza, efficienza e usabilità. Esistono inoltre ulteriori indicatori quali: ecocompatibilità, scalabilità, verificabilità, manutenibilità, riparabilità, evolvibilità, riusabilità e portabilità. L'importanza dei singoli parametri può essere definita in base alla tipologia di software che si sta realizzando e all'ambiente in cui esso andrà ad operare, ma alcuni risultano generalmente imprescindibili.

Purtroppo non sempre è possibile rispettare i vincoli imposti su questi parametri o comprendere quale sia effettivamente la qualità del software durante la fase di sviluppo. Per risolvere questo problema è possibile ricorrere ad un approccio *model driven.* Seguendo questa metodologia vengono fornite una serie di linee guida che consentono di proseguire nel processo di sviluppo del software focalizzandosi sulla creazione di modelli. Operare in questo modo significa fornire una maggiore astrazione al processo di realizzazione del software, portando in secondo piano la parte di computazione e quella algoritmica.
I modelli che vengono costruiti grazie a questo approccio, se sufficientemente accurati, potranno dare una stima del comportamento che il software avrà una volta inserito nell'ambiente in cui si desidera farlo operare. Con il *model driven* è quindi possibile, durante la fase di progettazione, comprendere meglio le qualità del programma effettuando analisi che altrimenti andrebbero realizzate a posteriori.
Frequentemente però, durante la fase di sviluppo di una applicazione, le informazioni in possesso dei programmatori, per quanto riguarda l'ambiente in cui essa andrà ad operare, non risultano sufficientemente precise o complete. La mancanza di conoscenze esaustive, o l'imprecisione di quelle possedute in fase di sviluppo, possono portare alla creazione di modelli meno significativi e con una precisione inferiore rispetto alle aspettative.

Mediante l'uso di modelli lo sviluppatore desidera raccogliere tutti gli aspetti fondamentali della realtà in cui andrà ad operare il programma, filtrando quelli ritenuti poco rilevanti. Se tra gli elementi utilizzati per creare il modello si dovessero trovare informazioni errate o incomplete questo porterà, inevitabilmente, a produrre astrazioni in cui è presente il problema dell'incertezza.
Questo fenomeno di incertezza nei modelli viene affrontato già ampiamente in altri settori come filosofia, fisica, statistica, economia, finanza, psicologia o sociologia, dando origine in ognuno di questi a risultati imprevisti e in alcuni casi difficilmente gestibili. Nel campo dell'ingegneria del software il problema dell'incertezza, nonostante i grossi disturbi che può causare, non è stato ancora affrontato in modo esaustivo e non sono presenti ampi studi a riguardo.
Avere modelli in cui è presente il fenomeno dell'incertezza può portare a realizzare previsioni errate, le quali possono avere un impatto considerevolmente dannoso su un progetto in corso di sviluppo. Andare a creare un software non rispettando le specifiche di qualità, definite in fase di raccolta dei requisiti, può portare una azienda a sforare il tetto di budget preventivato per riuscire ad ottenere i risultati desiderati. È quindi possibile, con pochi errori di valutazione, trasformare un progetto potenzialmente redditizio in un problema da risolvere in cui sarà necessario investire tempo ed energie non previste. Generare modelli affidabili, le cui predizioni abbiano il grado di accuratezza desiderato, svolge un ruolo chiave in questo campo, in modo da poter utilizzare i risultati ottenuti come guida durante il processo di sviluppo. Per fare ciò bisogna poter garantire la qualità delle predizioni effettuate e di conseguenza fronteggiare il problema dell'incertezza identificandolo e gestendolo correttamente.

Questo elaborato di tesi si concentra su una categoria specifica di qualità del sistema, le prestazioni e una categoria specifica di modelli, i modelli a reti di code, che ben si prestano a rappresentare le situazioni reali che ho affrontato durante il progetto. I dati che vado ad analizzare sono riferiti all'utilizzo del *webmail* server dell'università di Saragozza e dei server di Wikipedia. Nei capitoli successivi introduco il concetto di incertezza, prima più in generale e successivamente all'interno di casi di studio più specifici. Mi concentro poi sulla descrizione dei modelli a reti di code, spiegandone il funzionamento e come essi possano essere usati per rappresentare le situazioni reali di mio interesse. Questi modelli, nella loro forma più semplice, con un solo servente ed un solo client, vengono utilizzati per realizzare predizioni del tempo di risposta del server, partendo dai dati in mio possesso e modellizzando il traffico delle richieste.
Una volta definito il concetto di incertezza e spiegato il funzionamento di un modello a rete di code vado a mostrare alcune tecniche utili all'individuazione dell'incertezza e ad una sua classificazione, mediante l'utilizzo di una opportuna tassonomia. Successivamente, nel corso di questo elaborato, descrivo alcune tecniche in grado di generare modelli del traffico delle richieste, creati con un approccio di tipo statistico. Ognuno di questi modelli va ad utilizzare indicatori statistici, quali media e varianza campionaria, per rappresentare le informazioni

in mio possesso.
Tramite questi indicatori sono in grado di realizzare previsioni del tempo di risposta del server più affidabili, mediante l'applicazione di formule appartenenti alla teoria delle code, riuscendo cosi a gestire l'imprecisione comportata dalla presenza dell'incertezza. Al fine di fornire dei risultati reali derivanti dall'applicazione di queste tecniche vado a mostrare in funzione un *tool,* da me realizzato sulla base dei due casi di studio reali affrontati. Questo programma mette a disposizione tutte le funzionalità utili ad eseguire in modo automatizzato i metodi di individuazione e gestione dell'incertezza.
Le previsioni ottenute dai modelli vengono comparate con valori vicini a quelli reali, ricavati tramite la simulazione di una rete di code, controllabile dal software. Il confronto permette di valutare quale dei modelli è in grado di fornire predizioni di qualità media migliore, rendendo possibile individuare il metodo di generazione più efficace. Per migliorare ulteriormente la capacità predittiva delle singole tecniche viene applicata un'ulteriore tecnica a valle dei risultati ottenuti, con la quale è possibile avvicinare ulteriormente le predizioni realizzate ai valori reali attesi.
Oltre a questo approccio di tipo statistico propongo inoltre l'uso di una rete neurale come modello. Tramite questa struttura dati, eseguendo un corretto addestramento, potrei essere in grado di ottenere previsioni attendibili del tempo di risposta del server, riuscendo cosi a gestire la presenza di incertezza.

Capitolo 2

L'incertezza

2.1 Definizione

Nel capitolo introduttivo ho utilizzato più volte il termine incertezza senza però averne precisato l'effettivo significato. Ritengo quindi opportuno fornire una definizione di incertezza in modo da poter meglio comprendere i concetti che verranno introdotti successivamente.
Frank Hyneman Knight, nel 1921, descrisse l'incertezza come:

"Uncertainty must be taken in a sense radically distinct from the familiar notion of risk, from which it has never been properly separated.... The essential fact is that 'risk' means in some cases a quantity susceptible of measurement, while at other times it is something distinctly not of this character; and there are far-reaching and crucial differences in the bearings of the phenomena depending on which of the two is really present and operating.... It will appear that a measurable uncertainty, or 'risk' proper, as we shall use the term, is so far different from an unmeasurable one that it is not in effect an uncertainty at all."[1]

Questa definizione però potrebbe risultare un po' troppo generale rispetto al contesto a cui è limitato questo lavoro di tesi, occorre quindi trovare una definizione del fenomeno che sia generale ed al tempo stesso limitata al campo del *modelling*.
Nell'articolo [2] viene fornita una seconda definizione per l'incertezza che potrebbe rivelarsi adatta al contesto in cui questo progetto si svolge:

"any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system"

Si potrebbe anche affermare, in modo meno formale, che l'incertezza è esprimibile come uno stato in cui si possiede una conoscenza limitata e dove è impossibile descrivere in modo esatto tutti gli aspetti del mondo ritenuti rilevanti. Conseguentemente l'esito o gli esiti futuri della situazione osservata risultano di difficile previsione.

2.2 Tassonomia

Per poter classificare nel modo più opportuno le differenti tipologie di incertezza ho scelto di utilizzare la tassonomia proposta nell'articolo [3].
Secondo questa classificazione è possibile descrivere l'incertezza individuata mediante l'uso di tre dimensioni: localizzazione, natura e livello. Ciascuna di queste ha la possibilità di assumere differenti valori e caratterizza il fenomeno osservandolo da una prospettiva differente.
La dimensione localizzazione è riferita alla fonte alla quale è possibile attribuire l'incertezza, per essa sono definiti i valori: parametri di input, struttura del modello e contesto.

- **Incertezza del contesto:** Connessa ai limiti del modello, questa incertezza riguarda le informazioni che vanno modellate e quindi la completezza del modello rispetto alla parte del mondo reale che si desidera rappresentare.
- **Incertezza nella struttura del modello:** Riferita alla forma del modello, ossia quanto accuratamente la struttura del modello rappresenta il sottoinsieme di mondo reale che si desidera rappresentare e se vi sono fenomeni che sono stati erroneamente trascurati.
- **Incertezza sui parametri di input:** Un tipo di incertezza statistica riferita ai valori di input del modello e ai metodi usati per calibrarne i parametri.

La prospettiva Natura si riferisce al fatto che l'incertezza possa dipendere da imperfezioni sulla conoscenza acquisita o sulla variabilità del fenomeno descritto, definendone appunto la natura.
Questa dimensione può assumere due valori: aleatorio o epistemico.

- **Incertezza aleatoria:** Dovuta ad una variabilità innata di alcune parti del modello considerate o alla casualità degli eventi.
- **Incertezza epistemica:** Dovuta a mancanza di dati sufficienti alla costruzione di conoscenza affidabile, imperfezioni nella fase di acquisizione dei dati o imperfezioni nel processo di costruzione della conoscenza.

Infine, la dimensione livello va a caratterizzare l'intensità della incertezza osservata. Nell'articolo [4] è possibile vedere come questa prospettiva sia caratterizzata da una scala in cui è possibile collocare l'incertezza osservata ed i cui estremi sono compresi tra conoscenza deterministica e totale ignoranza. All'interno di questa vengono definiti differenti valori, oguno dei quali indica quanta conoscenza manca per essere in grado di studiare deterministicamente il sistema.

Riporto di seguito la scala utilizzata:

- **Zero ordine di incertezza:** Mancanza di qualsiasi forma di incertezza
- **Primo ordine di incertezza:** Mancanza di conoscenza. Il soggetto manca di conoscenza riguardo ad un determinato elemento ma si è a conoscenza di tale mancanza.
- **Secondo ordine di incertezza:** Mancanza di conoscenza e di consapevolezza. In questo caso non vi è consapevolezza di quello che non si conosce.
- **Terzo ordine di incertezza:** Mancanza di un processo per scoprire la mancanza di consapevolezza. A meno di sapere dell'esistenza dell'incertezza il soggetto non ha modo di variare questa situazione di mancanza di conoscenza.
- **Quarto ordine di incertezza:** In questo caso si parla di meta incertezza. Si ha incertezza riguardo agli ordini stessi di incertezza.

Grazie alla tassonomia appena presentata, durante la fase di realizzazione del software, sono stato in grado di implementare dei test per individuare le diverse tipologie di incertezza in modo da comprendere meglio quale fosse il problema presente nei dati e come poterne effettuare una corretta gestione.

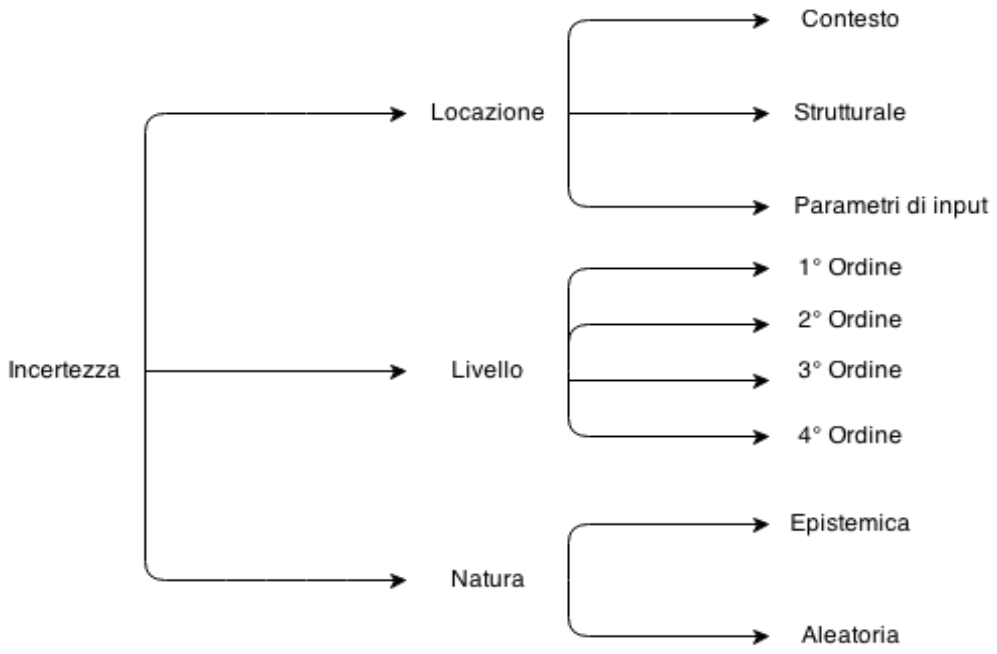

Figura 2.1 Tassonomia dell'incertezza utilizzata.

Capitolo 3

Test per individuare le tipologie di incertezza

Per poter gestire correttamente l'incertezza è necessario prima comprenderla, la tassonomia descritta nel capitolo 2 consente di classificare il fenomeno secondo determinate caratteristiche. In questo capitolo esporrò una serie di test orientati ad individuare le varie forme di incertezza conosciute. Una volta identificata la tipologia di incertezza osservata sarà possibile scegliere la tecnica più opportuna per gestirla arginando i problemi da essa comportati. Prima di passare ad esporre nel dettaglio i metodi è necessario comprendere meglio la situazione che si desidera trattare con essi. Sia dato un server a cui viene collegato un unico client o centro di servizio il quale effettua richieste mediante il canale di connessione. Il numero di richieste inviate viene osservato dal centro di servizio collegato al server e viene registrato all'interno di un log. Allo stesso modo, osservando la situazione dal punto di vista del server, viene tenuta traccia del carico su di esso conservando le rilevazioni su di un altro file. La situazione descritta può essere rappresentata mediante l'uso di un modello a rete di code caratterizzato da un unico centro di servizio e da un unico servente riuscendo cosi a preservare gli aspetti ritenuti rilevanti per le analisi.
Questa classe di modelli è stata proposta per la prima volta nell'ambito della teoria delle code, la quale è principalmente attribuita allo statistico ed ingegnere Agner Krarup Erlang[5]**.** Erlang, nel 1909, pubblicò "*The theory of probability and telephone conversations*", un articolo riguardante le attese nelle chiamate telefoniche relativamente al centralino di Copenhagen.
Successivamente, nel 1953, Kendall propose un sistema di notazione, visibile in [6], che è divenuto oggi lo standard per descrivere i nodi di una rete di code.
Un nodo è definito da fattori A,S e C, dove A denota il tempo trascorso tra un arrivo e l'altro in coda, S la dimensione dei *jobs* e C il numero di serventi al nodo. Vari sono i valori che possono assumere questi attributi, la configurazione più semplice è però la rete detta M/M/1 nella quale vi è un singolo servente, gli arrivi sono determinati da un processo di Poisson ed i tempi di servizio hanno distribuzione esponenziale.

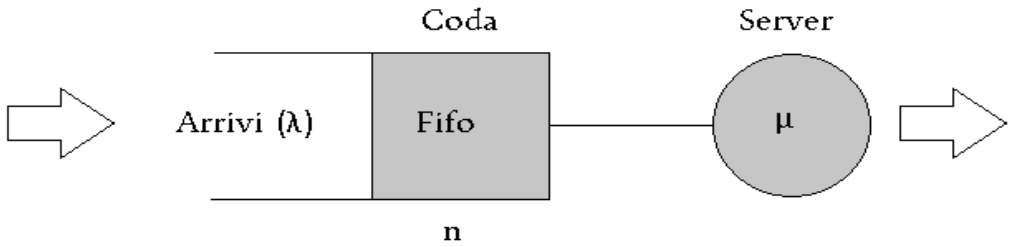

Figura 3.1 Singolo nodo di una rete di code di tipo M/M/1

In figura 3.1 è possibile vedere un nodo di una rete M/M/1 in cui gli arrivi giungono con una frequenza caratterizzata dal parametro λ, la coda invece, che frequentemente viene assunta come infinita, viene gestita con una logica FIFO (*First In First Out*) ed il tempo di servizio è caratterizzato da un parametro μ.
Il punto di forza di questa teoria è la possibilità, risolvendo alcune semplici equazioni, di ottenere alcune misure rilevanti per il modello.
Un esempio di queste misure sono:

- Utilizzo: La proporzione di tempo in cui il server è occupato.
- Lunghezza della coda: Il numero medio di clienti in attesa di essere serviti o che stanno venendo serviti in quel momento.
- Tempo di risposta: Il tempo medio speso in coda per essere serviti sommato al tempo di servizio.
- Frequenza: Il tasso di transito dei clienti nel centro di servizio.

Il modello a rete di code che utilizzerò per esporre i metodi di test ha l'obiettivo di essere semplice ed al tempo stesso esplicativo della situazione descritta precedentemente. Caratteristiche principali del modello proposto:

- Un server, il cui tempo di servizio è esprimibile come una distribuzione esponenziale di parametro caratteristico μ. Tale tempo di servizio non rientra nelle informazioni a disposizione per l'applicazione dei test.
- Un unico centro di servizio, collegato al servente di cui è nota la frequenza di arrivo delle richieste, esprimibile come una distribuzione esponenziale di parametro λ. Le frequenze di arrivo sono calcolate mediante il log delle richieste a disposizione. Per ogni istante di campionamento viene dunque associato una certa frequenza λ_i.
- Il carico registrato sul servente, contrassegnato come N, viene registrato su un log in cui per ogni istante di campionamento è associato un valore di carico.

Per prima cosa, dopo aver definito il modello, occorre comprendere se sia presente una qualche forma di incertezza su di esso in modo da non eseguire una serie di test che sarebbero altrimenti superflui. Per verificare ciò basta andare a rilevare la presenza di inconsistenze tra i dati misurati e quelli ottenibili mediante calcolo. Come primo passo nel processo di identificazione è necessario andare a computare i vari parametri caratterizzanti il tempo di servizio nei diversi istanti di campionamento, per fare ciò è possibile utilizzare la relazione $\mu_i = N_i / \lambda_i$ nota dalla teoria delle code. L'insieme dei valori μ_i può poi essere rappresentato mediante il calcolo della sua media campionaria, $\overline{\mu}$.
In assenza di una qualsiasi forma di incertezza mi aspetto che tale media consenta di risalire nuovamente ai valori di carico N_i registrati nel log del server, in caso contrario potrebbe essere presente una qualche forma di incertezza. Utilizzando la relazione $N_i = \mu_i/\lambda_i$ vado a calcolare tutti i valori di carico mediante l'uso della teoria delle code in modo da poter effettuare un confronto con quelli registrati.

Se la differenza tra le varie coppie di valori, [$N_{i\ calcolato}$, $N_{i\ misurato}$] dovesse rivelarsi troppo elevata tale discrepanza evidenzierebbe la presenza di una forma di incertezza. Nel caso in cui venga riscontrata la presenza di incertezza sui dati occorre utilizzare delle opportune analisi per classificare il fenomeno secondo la tassonomia descritta nel capitolo 2.
Per svolgere tale compito ho scelto di utilizzare un approccio proposto nell'articolo [7] che prende il nome di MUSE (*Managing Uncertainties in Software modEls*) il quale partendo da un modello dato è in grado di fare da guida nell'individuare l'esistenza di incertezza e nella sua gestione.

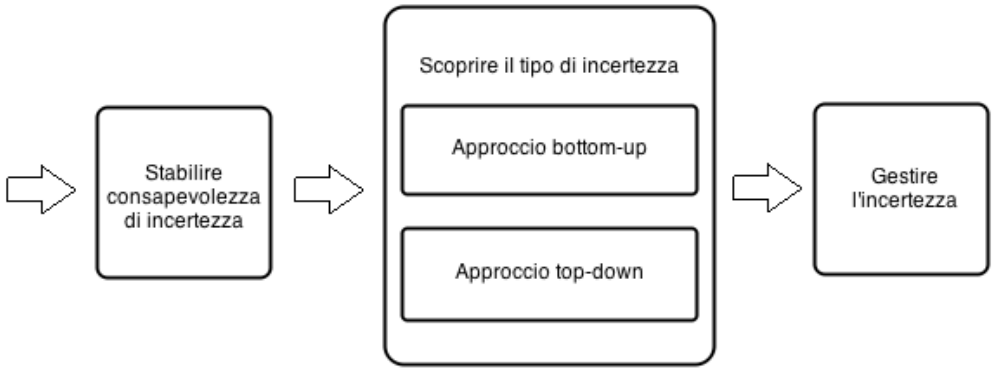

Figura 3.2. Panoramica del MUSE

Nella fase di identificazione dell'incertezza, secondo blocco in figura 3.2, il MUSE ha la possibilità di usare due diversi tipi di approccio, l'uno di tipo *bottom-up* e l'altro *top-down*, i quali vengono mostrati in figura 3.3.

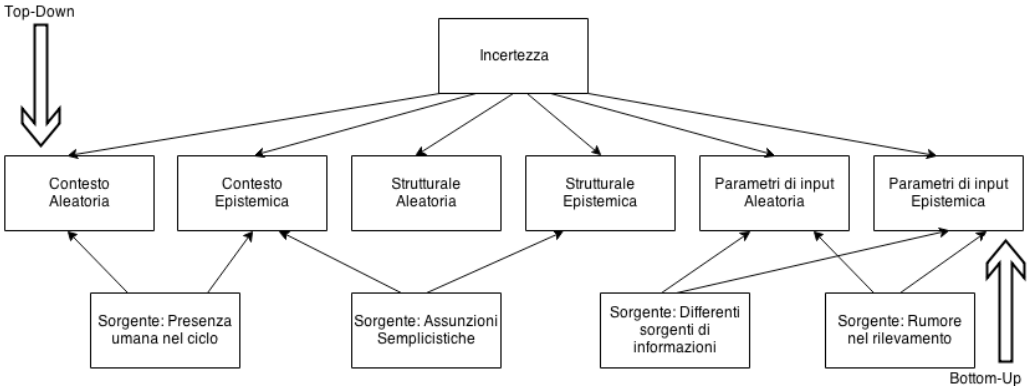

Figura 3.3 Due metodi per identificare l'incertezza

Il metodo da me scelto eseguirà, con logica *top-down*, i test per l'individuazione della tipologia di incertezza nel seguente ordine:

- Test per la ricerca di incertezza di tipo aleatorio sui parametri di input.
- Test per la ricerca di incertezza di tipo epistemico sui parametri di input.
- Test per la ricerca di incertezza strutturale.

Ognuno di questi test può essere visto come un modulo indipedente con il compito di verificare la presenza di una singola tipologia di incertezza. Questi moduli ricevono in ingresso il modello e forniscono in uscita un output di tipo booleano venendo cosi trattati dall'algoritmo guida come delle *black box*. Nel caso in cui uno dei test eseguiti dal MUSE identifichi la presenza di incertezza del tipo ricercato il processo verrà arrestato. L'ordine in cui i test vengono eseguiti non è casuale, il MUSE utilizza inizalmente quelli che vanno a verificare la presenza di tipologie di incertezza più facilmente gestibili e solo in presenza di riscontri negativi seleziona i test più complessi.
Vi è quindi da tenere presente che non è garantito che con questo metodo l'incertezza rilevata venga classificata nella sua reale classe di appartenenza infatti, verranno prima eseguiti i test per verificare l'incertezza sui parametri di input, essendo questo fenomeno il più facile da individuare. Nel caso in cui i primi due test diano esito negativo si procederà alla ricerca di incertezza nella struttura del modello utilizzato. Se essi dovessero però fornire un esito positivo potrebbe verificarsi il caso in cui una incertezza dovuta al contesto venga classificata come causata dai parametri di input perchè quest'ultima tipologia, più semplice ,è in grado di spiegarla sufficientemente bene.
Per i casi di studio analizzati non è stato necessario spingersi oltre l'analisi dell'incertezza di tipo strutturale ma, nel caso in cui anche l'ultimo test fallisca, sarà necessario ampliare la ricerca del fenomeno con la realizzazione di metodi per l'individuazione di incertezza dovuta al contesto.

3.1 Incertezza aleatoria sui parametri di input

In questa sezione andrò a descrivere il primo test suggerito dal MUSE mediante il quale si desidera andare a verificare la presenza di incertezza di tipo aleatorio sui dati di input. Questa classe di incertezza, come già detto, è tra le più semplici da gestire e risulta quindi conveniente andare ad analizzarla per prima.
Come modello di riferimento per l'applicazione del test andrò ad utilizzare quello mostrato in figura 3.1, composto da un solo servente ed un solo servizio ad esso collegato di cui sono noti unicamente le richieste inviate al servente ed il carico misurato su di esso. L'algoritmo va ad analizzare il comportamento dei valori μ_i, rappresentanti i parametri caratteristici delle distribuzioni esponenziali dalle quali è possibile estrarre i tempi di servizio nei vari istanti di campionamento.
L'insieme dei parametri μ da utilizzare per il test viene calcolato utilizzando la relazione $\mu_i = N_i / \lambda_i$, nota dalla teoria delle code. Se la varianza campionaria su tale insieme dovesse risultare eccessivamente elevata, andando ad indicare valori troppo sparsi, il fenomeno di incertezza registrato sarà difficilmente classificabile come di tipo aleatorio. Infatti, affinchè l'incertezza sia rappresentabile con la tipologia osservata l'algoritmo si aspetta di calcolare un valore di servizio quasi costante o con una varianza molto contenuta e ritenuta tollerabile per essere rappresentata con un fenomeno aleatorio. In presenza di un

fenomeno di questo tipo infatti il parametro caratterizzante il tempo di servizio non dovrebbe fluttuare eccessivamente.
Basandosi quindi sul valore calcolato della varianza campionaria il test fornirà un output di tipo booleano. Nel caso in cui venga restituito il valore falso sarà necessario proseguire con i test successivi, suggeriti dall'algoritmo principale che guida l'analisi. Se il test dovesse fornire in output il valore vero sarà possibile interrompere il processo ed affermare che l'incertezza riscontrata è di tipo aleatorio sui parametri di input o quanto meno ben spiegabile da questa classe.

3.2 Incertezza epistemica sui parametri di input

Questo metodo di analisi va a verificare se l'incertezza rilevata può essere associata ai dati di input e classificabile non come un errore di tipo aleatorio ma dovuto alla mancanza di una conoscenza approfondita su qualche aspetto della realtà che si vuole rappresentare. Il modello che utilizzerò per spiegare il processo di test è quello visibile in figura 3.1, composto da un solo servente ed un solo servizio ad esso collegato di cui sono noti unicamente il log degli *arrival rate* del servizio ed il log del carico sul servente.
Quello che si desidera andare a verificare mediante l'ausilio di questo metodo è la presenza di una qualsiasi correlazione in grado di spiegare la discrepanza tra i dati che sono stati misurati e quelli che possono essere calcolati, mediante l'applicazione di alcune semplici equazioni appartenenti alla teoria delle code. Partendo dall'assunzione che una relazione tra il carico sul servente e le richieste inviate dal servizio esista, essa potrebbe non risultare evidente per la presenza di un qualche errore strumentale che potrebbe aver condizionato la registrazione dei log. Una ipotesi ragionevole, per attribuire una causa a tale errore di registrazione, è la possibilità che l'orario di sistema del server e l'orario di sistema del servizio ad esso collegato differiscano per via di posizioni geografiche, impostazioni diverse o di scelte degli amministratori differenti.
Nel caso in cui questa ipotesi dovesse rivelarsi reale dovrebbe comunque esistere una relazione tra i due log misurati, individuabile incrociando tutte le possibili coppie di elementi. Il test quindi andrà a calcolare tutte le svariate combinazioni di coppie spostando l'orario di registrazione di uno dei due log avanti o indietro nel tempo di un certo numero di ore.

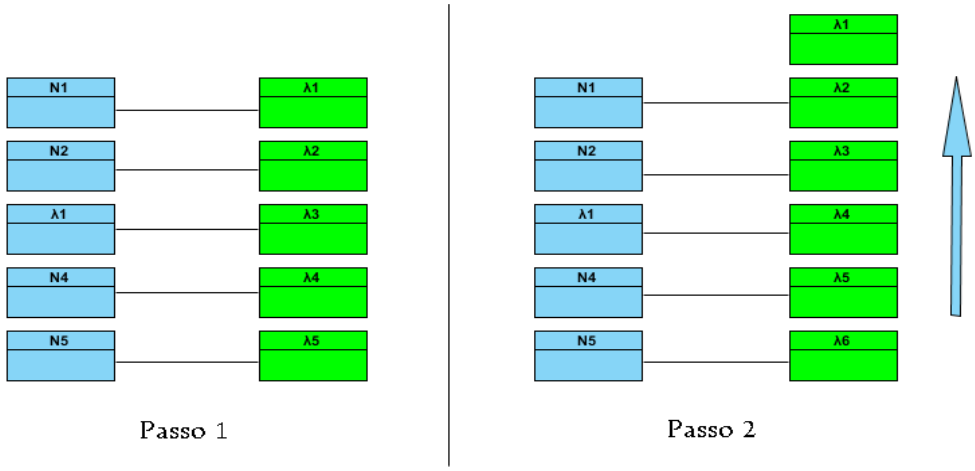

Figura 3.4. Rappresentazione del processo di ricerca delle diverse combinazioni tra le coppie appartenenti ai due log disponibili.

Nel caso in cui tale relazione esista e spostando in avanti o indietro nel tempo l'orario di registrazione degli arrival rate del servizio le coppie create di N_i e λ_i dovessero risultare coerenti il test restituirà in output il valore booleano vero andando quindi a classificare l'incertezza rilevata come di tipo epistemico sui parametri di input. Se il valore di output dovesse essere falso sarà invece necessario proseguire con ulteriori test che andranno a prendere in considerazione tipologie di incertezza più complesse da gestire, non essendo il fenomeno rilevato spiegabile in modo più semplice.

3.3 Incertezza strutturale

Questo test ha l'obiettivo di andare a verificare la presenza di incertezza strutturale. Ricercando questo tipo di fenomeno si va ad ipotizzare che il modello mostrato in figura 3.1 possa essere strutturalmente inesatto, non tenendo esso conto di una qualche componente rilevante ai fini delle analisi. Una ipotesi ragionevole da cui partire è che esistano un certo numero di richieste ignote giunte al servente e non registrate nel log che vanno ad influire sul suo carico di lavoro (figura 3.5). Queste richieste non comparendo all'interno del log, registrato osservando il sistema dal centro di servizio, devono necessariamente provenire da un secondo client collegato al server e non visibile in fase di raccolta dati. Se ciò dovesse rivelarsi vero il non aver registrato alcune richieste spiegherebbe la discrepanza tra le coppie $[N_{i\ calcolato}, N_{i\ misurato}]$ calcolate all'inizio del capitolo. Per risolvere il problema potrebbe essere sufficiente aggiungere un certo quantitativo di richieste in opportuni istanti temporali del log registrato (figura 3.6) in modo che i valori di carico calcolati sul server vadano a crescere avvicinandosi a quelli registrati. Essi infatti sono influenzati direttamente dalle richieste giunte, come è possibile vedere dalle formule appartenenti alla teoria delle code applicate.

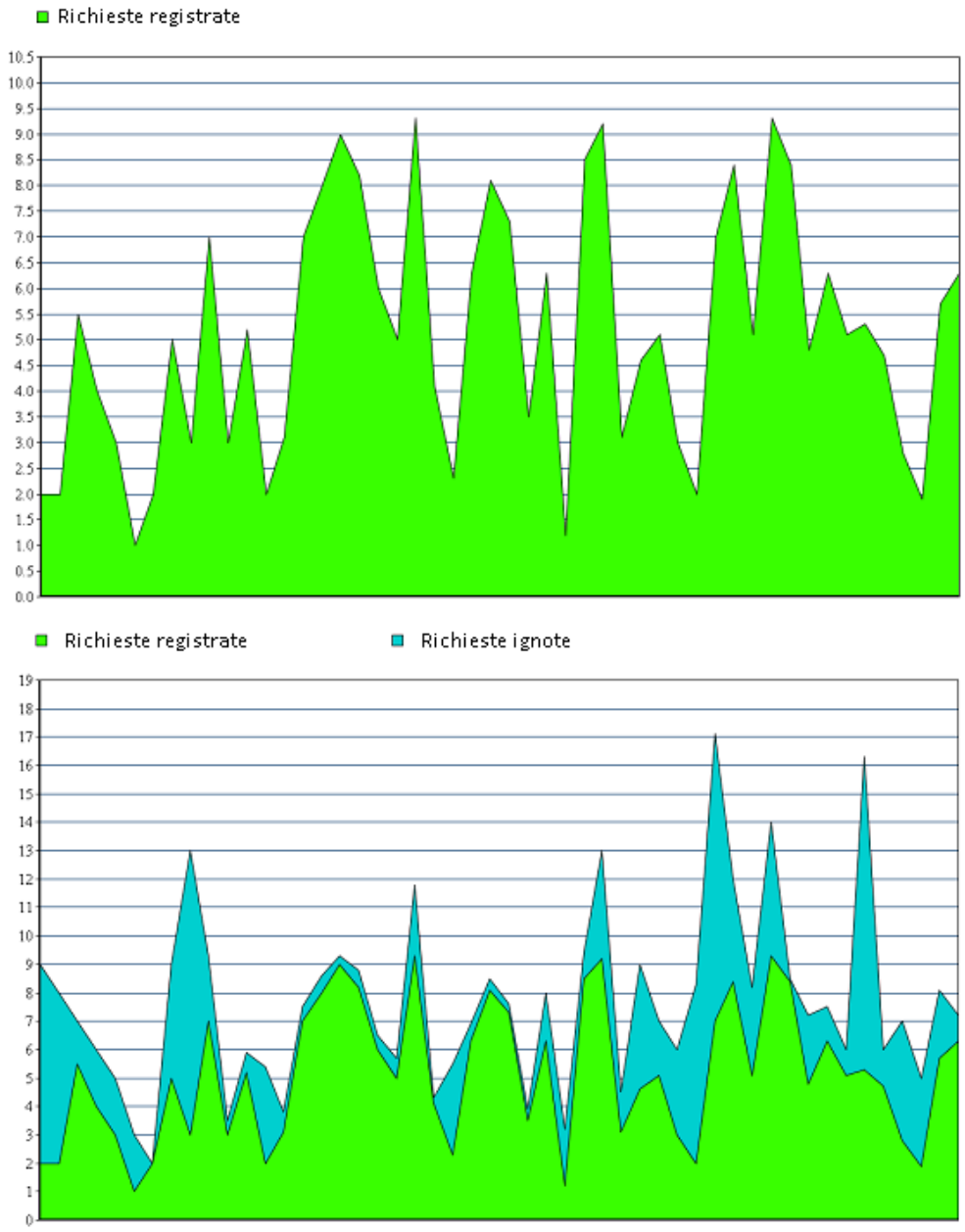

Figura 3.5 Log di richieste registrate di esempio (parte superiore) a cui viene sommato il contributo di una serie di richieste ignote (parte inferiore).

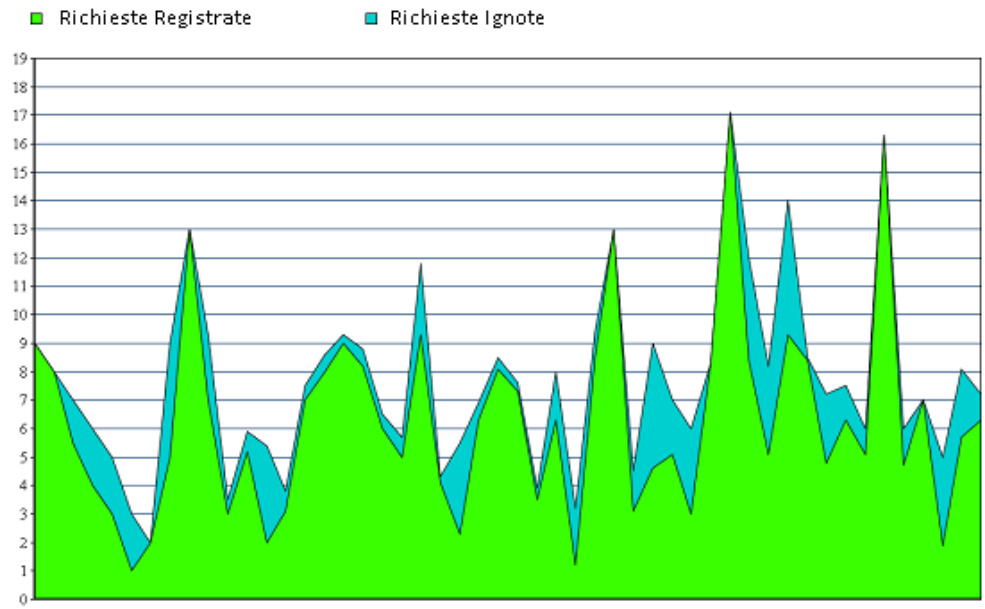

Figura 3.6 Valori delle richieste registrate d'esempio modificate per avvicinarsi maggiormente al valore che si avrebbe conoscendo le richieste ignote giunte al server nei vari istanti temporali.

L'idea è quindi quella di fare alcune piccole modifiche mirate al log delle richieste in modo tale da avere un *matching* accettabile tra i dati misurati e quelli calcolati riducendo cosi l'incertezza. Occorre però selezionare in modo opportuno quali istanti temporali modificare nel log per ridurre nel modo migliore possibile l'area tra le due funzioni. Per riuscire ad automatizzare il processo di selezione delle modifiche da effettuare sui valori contenuti nel log è possibile avvalersi di un algoritmo di tipo genetico.
L'algoritmo che ho deciso di utilizzare per la modifica del log delle richieste è l'*Hill Climbing* [8] il quale opera cambiando iterativamente piccole parti del modello di partenza nel tentativo di massimizzare (o minimizzare) una certa funzione obiettivo f(x). La scelta di tale funzione risulta molto importante, essa deve essere in grado di fornire una stima accurata del miglioramento del modello in modo da poter guidare l'algoritmo nel modo ottimale. Per il caso in esame il miglioramento sarà dato da quanto i profili di carico, dati dalle coppie $[N_{i\ calcolato}, N_{i\ misurato}]$, si sono avvicinati dopo aver apportato l'ultima modifica. Ad ogni iterazione l'*Hill Climbing* va ad effettuare una singola mutazione al problema che sta trattando (variazione di un valore nel log delle richieste) e successivamente va a calcolare come tale cambiamento influisce sul valore della funzione obiettivo. Applicando questo metodo qualsiasi modifica che migliora il risultato, stimato mediante la funzione f(x), è accettato e l'algoritmo prosegue fino a quando non è possibile progredire ulteriormente nella riduzione della funzione obiettivo. L'*Hill Climbing*, per come opera, fornisce la garanzia di ottenere solo un ottimo locale, questo problema può però essere aggirato mediante l'utilizzo di diversi stratagemmi. Ad esempio è possibile avviare l'algoritmo più volte, utilizzando per ognuna di queste punti di partenza differenti, selezionati in modo casuale o utilizzando un metodo apposito per la determinazione del punto di partenza.

Ad ogni iterazione dell'algoritmo esistono però differenti modifiche applicabili che portano ad una riduzione della funzione obiettivo, ad esempio nel log riportato in figura 3.5 come prima scelta è possibile selezionare diversi dati in grado di ridurre l'area tra le due funzioni. Per ovviare a questo problema ho scelto di far guidare il processo di scelta da una euristica. La funzione selezionata consente di ottenere il miglior risultato possibile senza dover tenere in considerazione molteplici mutazioni, facendo cosi decrescere il peso computazionale e semplificando il processo di scelta. Anche in termini di memoria richiesta l'algoritmo diventa meno esigente non essendo necessario andare a tener traccia delle differenti versioni del modello generate per ogni iterazione.
Sempre con l'idea di ridurre il lavoro computazionale e garantire la validità del test ho ritenuto inoltre importante impostare un numero massimo di mutazioni consentite in modo da non cambiare eccessivamente il modello prodotto. Rimarranno quindi, molto probabilmente, alcune discrepanze tra le due funzioni, come mostrato in figura 3.6, rendendo necessaria una valutazione di quanto l'algoritmo ha effettivamente consentito di avvicinare i valori di carico misurato e calcolato e se i cambiamenti attuati hanno effettivamente portato ad un sensibile miglioramento del modello. Riporto di seguito uno schema riassuntivo del funzionamento del metodo di *Hill Climbing* applicato:

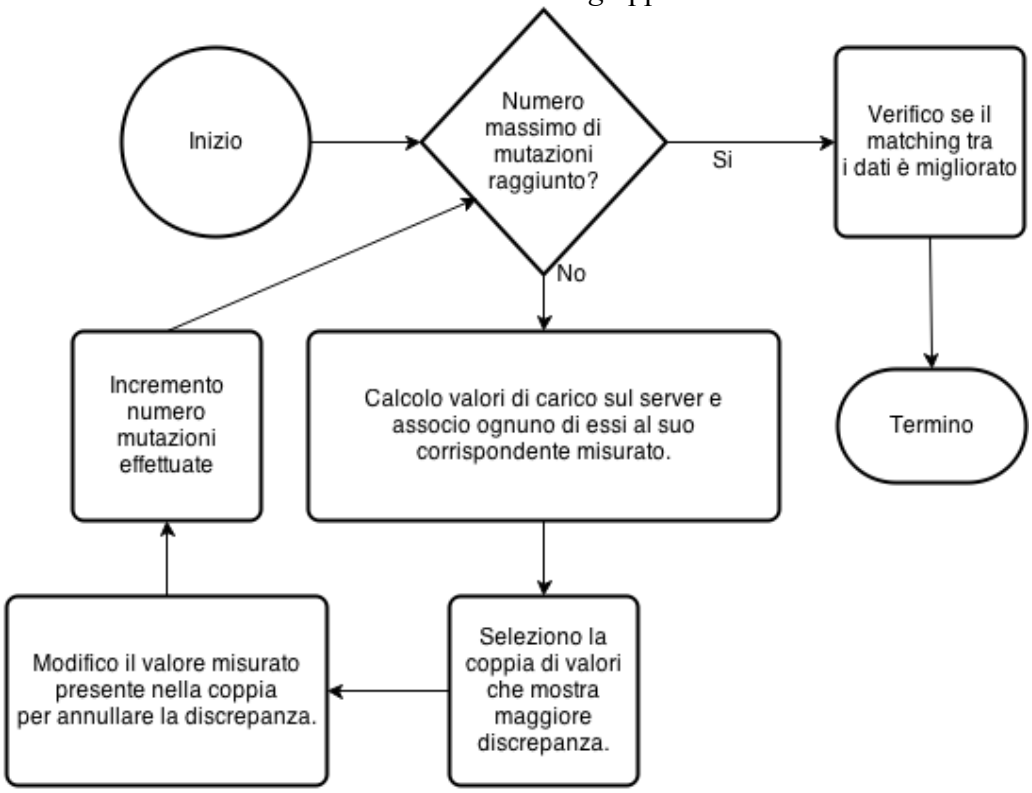

Figura 3.7 Funzionamento dell'algoritmo di *Hill Climbing* per il modello proposto.

Il test, dopo aver calcolato i valori di carico sul servente, mediante l'uso delle relazioni note dalla teoria delle code, va ad effettuarne la comparazione con i valori misurati. Ad ogni passo dell'algoritmo, grazie all'euristica scelta, viene selezionata una coppia di elementi [$N_{i\ calcolato}$, $N_{i\ misurato}$] che mostra la maggiore differenza, tale coppia viene quindi utilizzata come candidata alla modifica.
Il valore della coppia misurato viene modificato in modo tale che la differenza tra esso ed il valore corrispondente calcolato sia nullo. Operare questo cambiamento equivale ad aggiungere un certo numero di richieste nell'istante temporale selezionato, riducendo cosi l'area tra le due funzioni in figura 3.5, in modo tale che il carico calcolato sul server in quel momento aumenti fino al valore di quello misurato. Se, dopo aver eseguito il numero massimo di mutazioni consentite, andando a calcolare il carico sul servente, mediante i nuovi valori di richieste computati, il profilo prodotto è sufficientemente simile al profilo di carico misurato, allora il test classificherà l'incertezza rilevata come di tipo strutturale terminando il processo.
Questo test rappresenta l'ultimo dei metodi concepiti per l'identificazione della tipologia di incertezza, secondo la tassonomia descritta nel capitolo 2 esistono altre classi di incertezza ma non sono stati realizzati test per la loro identificazione in quanto nei casi di studio affrontati sono sempre riuscito a ricondurre i fenomeni osservati ad una delle categorie precedentemente esposte. Vi è inoltre da tener conto che una incertezza di tipo contestuale potrebbe essere benissimo classificata con un'altra classe più semplice da gestire. Nel momento in cui l'algoritmo utilizzato va a rilevare che una tipologia di incertezza riesce a spiegare sufficientemente bene il fenomeno riscontrato risulta inutile tentare di classificarlo con una tipologia più complessa da gestire.

Capitolo 4

Metodi proposti per gestire l'incertezza

Dopo aver fornito una definizione per il concetto di incertezza ed una tassonomia utile per la sua classificazione, in questo capitolo, usando come riferimento il semplice modello presentato nel capitolo 3, definirò una serie di metodi utili alla gestione del problema comportato dal fenomeno dell'incertezza. Tramite le tecniche presentate sarà possibile migliorare le prestazioni dei modelli ed ottenere predizioni di migliore qualità. Attuando una corretta gestione di questo problema si riesce dunque a rappresentare in modo migliore la porzione di realtà d'interesse e nel caso del modello proposto riuscire a definire in modo migliore il funzionamento dell'intero sistema utilizzato come riferimento. I metodi che andrò successivamente ad elencare possono essere suddivisi a grandi linee in due categorie:

- Metodi che utilizzano un approccio statistico
- Uso di reti neurali come modello

Tramite l'uso di queste tecniche si desidera riuscire a rappresentare nel modo più sintetico ed affidabile possibile le informazioni e fornire previsioni di misure rilevanti quali ad esempio il tempo di risposta di una certa macchina a cui è collegato il servizio. La bontà di un modello viene infatti generalmente definita da quanto esso rappresenta bene gli aspetti della realtà che si desidera tenere in considerazione, dal quanto risulta utile nel effettuare predizioni e dalla sua complessità.
Preso come riferimento il modello a rete di code proposto nel capitolo 3, composto da un solo servente a cui è associato un unico centro di servizio, è possibile applicare su di esso metodi di gestione dell'incertezza, che verranno proposti nel corso di questo capitolo, al fine di riuscire a modellizzare nel modo più preciso possibile il traffico di richieste proveniente dal servizio ed indirizzate al server. Il valore del tempo di risposta del servente è computabile utilizzando formule ricavate dalla teoria delle code, partendo dalla registrazione delle richieste ed avendo a disposizione ulteriori informazioni sullo stato del sistema. La teoria delle code ha però dei limiti, talvolta infatti potrebbe non risultare possibile calcolare i valori a cui si è interessati. Uno dei casi in cui questo metodo risulta non applicabile è quello in cui il server non dispone di sufficienti risorse per soddisfare il numero delle richieste giunte in un dato istante temporale; il tempo di risposta, in questa situazione, non è quindi ottenibile mediante una analisi statica ma è richiesto di utilizzare un altro approccio per risalire al suo valore. Una valida possibilità è quella di effettuare una simulazione della rete di code, eventualmente ripetendola più volte al fine di

avere dei risultati il più possibile statisticamente attendibili.
Utilizzare una simulazione può però comportare anche degli svantaggi, sebbene i risultati ottenuti mediante essa possano risultare un ottimo metro di paragone per verificare le prestazioni dei modelli, il metodo simulativo risulta eccessivamente dispendioso dal punto di vista computazionale, specie se sono richieste molteplici esecuzioni. In termini di tempo e di risorse utilizzate quindi tale metodo non è paragonabile all'uso di modelli per il calcolo di misure ritenute rilevanti, quali il tempo di risposta di un servente. L'approccio simulativo risulta quindi sconsigliabile nel caso esistano alternative valide ed applicabili a minor costo.
È importante trovare un modo per essere in grado di applicare le semplici equazioni note ad un insieme più ampio di casi. Per aumentare l'applicabilità di tale tecnica potrebbe essere utile rappresentare le richieste di servizio in modo differente, utilizzando ad esempio alcuni indici statistici quali media e varianza campionarie. Tramite tali indici è possibile rappresentare in modo più compatto le informazioni contenute nei log ed allo stesso tempo, a meno di casi limite, sarà possibile utilizzare formule note della teoria delle code per effettuare le previsioni. Esprimendo in questo modo un log si introducono però delle approssimazioni che potrebbero rendere il risultato meno preciso di quanto si possa desiderare, le tecniche che verranno esposte hanno lo scopo di mantenere la possibilità di computare un risultato in modo agevole e riuscire a gestire l'incertezza affinchè le previsioni realizzate mediante modelli siano valide.
Alternativamente si potrebbe realizzare un modello differente dal precedente utilizzando una rete neurale addestrata opportunatamente con apprendimento supervisionato. Tramite tale rete potrebbe essere possibile, fornendo il giusto input e con la giusta configurazione, ottenere l'output atteso effettuando cosi delle predizioni per quanto riguarda il tempo di risposta di un server.
Nei capitoli successivi presenterò una serie di tecniche volte a consentire di realizzare previsioni del tempo di risposta di un server mediante la costruzione di modelli del traffico di richieste, nonostante sia stata rilevata in precedenza una forma di incertezza strutturale ed i limiti di applicabilità delle formule apppartenenti alla teoria delle code. Proverò inoltre a variare il modello andando ad utilizzare una rete neurale per effettuare previsioni. L'accuratezza dei risultati ottenuti verrà valutata con i valori ricavati mediante tecniche simulative.

4.1 DBSCAN

Il primo metodo che vado a presentare è il DBSCAN [9] (*Density-Based Spatial Clustering of Applications with Noise*). Questa tecnica venne proposta per la prima volta nel 1996 da Martin Ester, Hans-Peter Kriegel, Jörg Sander e Xiaowei Xu e rientra nell'ambito delle tecniche di *clustering*.
Grazie al DBSCAN è possibile connettere differenti regioni di punti usando come indicatore la densità, aree con densità simile vengono connesse, permettendo cosi di individuare ed isolare gli *outlier* presenti nell'insieme dei dati.

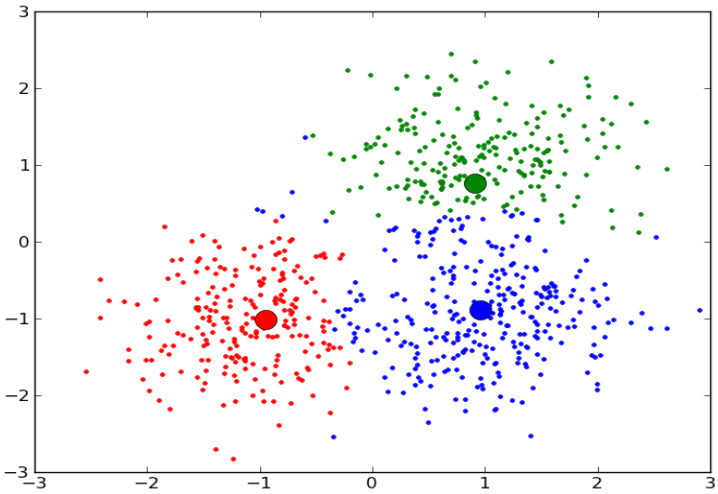

Figura 4.1 Applicazione del DBSCAN su un insieme di dati.

In figura 4.1 è possibile vedere come in un insieme di punti vengano identificate dall'algoritmo tre aree a densità diverse e vengano creati tre differenti *cluster*.
Usando sempre come riferimento il modello proposto nel capitolo 3 si può capire perchè gli *outlier* in un determinato log possano risultare molto rilevanti. Andando ad esaminare le richieste di servizio, ad esempio, individuare punti anomali in aree a bassa densità equivale, nella maggior parte dei casi, ad identificare richieste di servizio eccessivamente alte che in alcuni casi potrebbero rendere impossibile l'uso dell'analisi mediante la teoria delle code. Vi è anche da tenere presente che la funzione che caratterizza l'andamento del tempo di risposta in funzione dell'utilizzo (figura 4.2) non ha un andamento lineare ed il contributo di questi picchi di richieste è quindi molto maggiore rispetto a quello che si avrebbe utilizzando una funzione lineare.

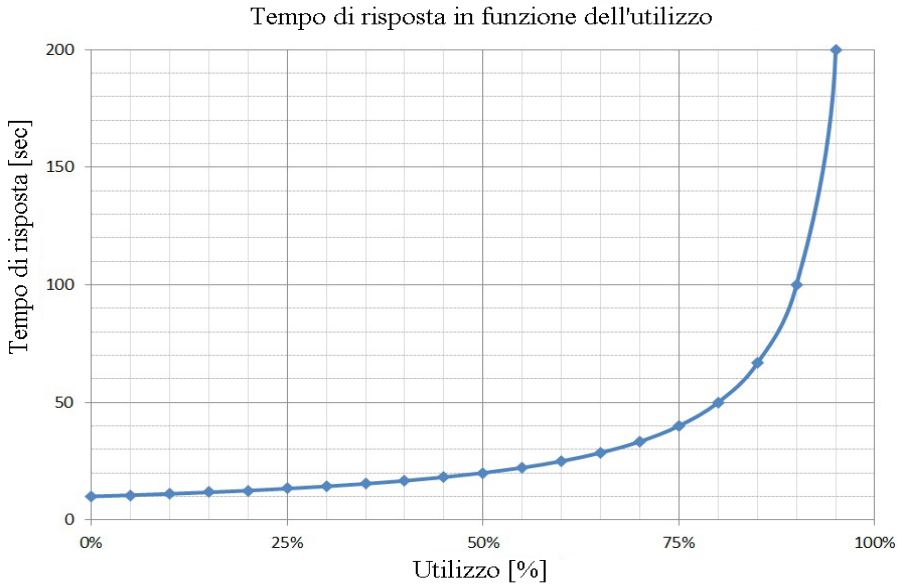

Figura 4.2. Andamento del tempo di risposta di un servente in funzione del suo utilizzo.

Individuare e trattare questi valori separatamente potrebbe quindi rappresentare una soluzione per non perdere tali informazioni ed allo stesso tempo riuscire ad utilizzare semplici equazioni per attuare predizioni su valori ritenuti rilevanti nel funzionamento del sistema analizzato. Il numero di *cluster* utilizzabili con questa tecnica è variabile, nell'implementazione realizzata per questo progetto si è scelto di utilizzarne uno solo e considerare tutti i dati esterni ad esso come punti anomali, interpretandoli quindi come picchi di richieste, probabilmente non gestibili immediatamente con la capacità di servizio associata al server.
Per capire il funzionamento di questo algoritmo basato sulla densità per prima cosa è necessario introdurre una serie di concetti che risulteranno poi utili per la sua descrizione:

- **Direttamente raggiungibile in densità:** Siano q e p due punti, q viene detto direttamente raggiungibile da p se essi non sono distanti più di una certa quantità ε, tale quantità è definita generalmente dall'utente che sta utilizzando l'algoritmo ed è utile ad indicare l'area entro cui cercare punti vicini. A seconda del numero di dimensioni utilizzate la quantità ε può quindi indicare una distanza o una superficie o un volume. Assegnando ad ε un valore ritenuto grande l'algoritmo utilizzato sarà impostato in un modo più permissivo, ossia andrà a ricercare tutti i punti adiacenti a quello di partenza prendendo in considerazione un intorno più ampio.
- **Raggiungibile in densità:** Siano q e p due punti, q si dice raggiungibile in densità p se esiste una sequenza $p_1 \ldots p_n$ di punti con $p_1 = p$ e $p_n = q$ dove ognuno di essi è direttamente raggiungibile dal suo predecessore.
- **Densamente connesso:** Due punti p e q sono connessi in densità se esiste un punto o tale che sia o-p che o-q siano raggiungibili in densità.

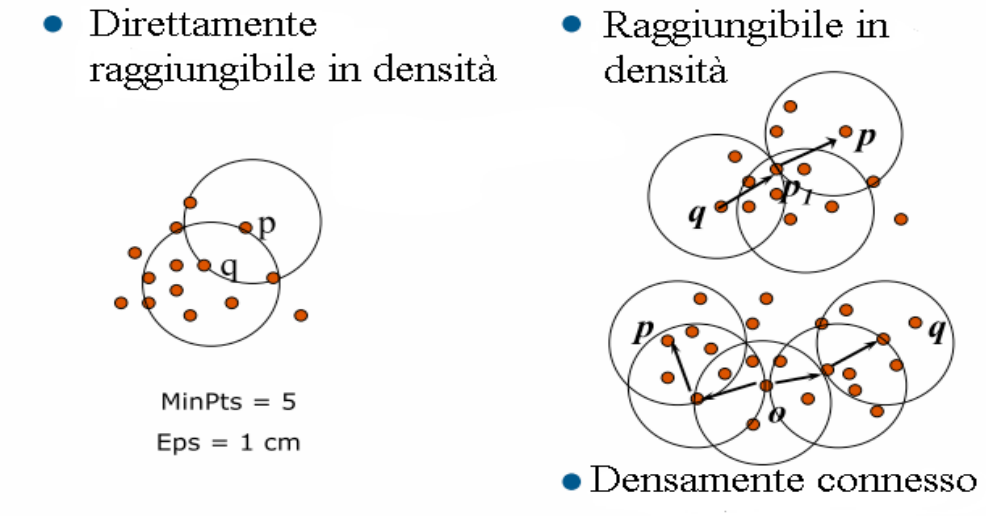

Figura 4.2. Rappresentazione dei casi di direttamente raggiungibile in densità, densamente raggiungibile e densamente connesso per coppie di punti.

Partendo dai concetti esposti precedentemente ho ora la possibilità di introdurre la definizione di *cluster* per l'algoritmo DBSCAN. Un *cluster* può essere definito come un insieme in cui tutti i punti al suo interno sono mutualmente connessi in densità. Inoltre, se un punto è connesso in densità ad un altro punto del *cluster* allora anch'esso ne è parte.
Per poter spiegare l'algoritmo sono inoltre necessari i tre seguenti concetti:

- **Core point:** Punto all'interno di un *cluster*, da esso è possibile raggiungere in densità un numero di punti maggiore del numero minimo definito per poter creare un *cluster* (minpts).
- **Border point:** Punto posto sul bordo di un *cluster*, esso è raggiungibile in densità da un *core point* ma, partendo da un *border point*, non è più possibile espandere ulteriormente il *cluster*.
- **Noise point:** Rappresenta un *outlier*, l'algoritmo non è in grado di collocarlo in alcun *cluster* con i parametri ε e minpts forniti.

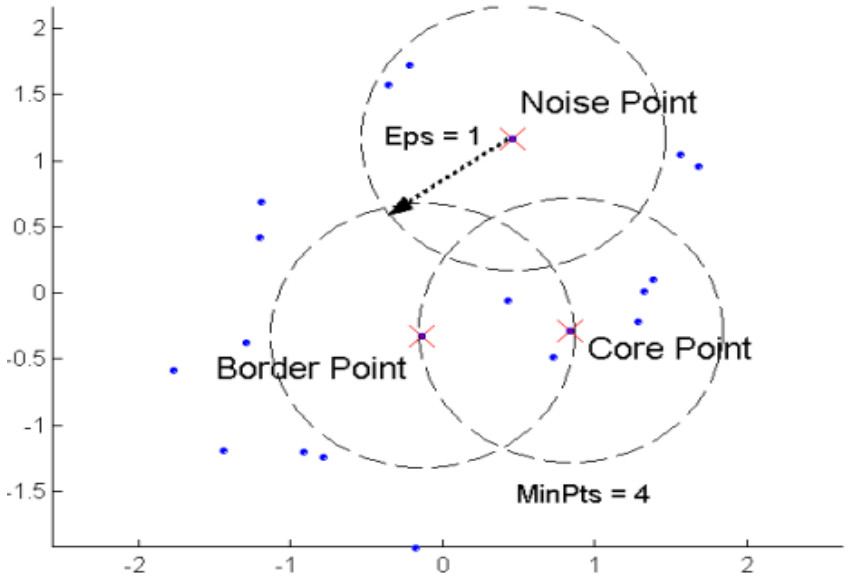

Figura 4.3. Rappresentazione di esempi di *noise point*, *border point* e *core point*.

Preso in esame il modello di riferimento proposto nel capitolo 3 è possibile applicare l'algoritmo di DBSCAN alle richieste inviate dal servizio andando a creare un modello rappresentativo ma più compatto.

L’algoritmo è composto dai seguenti passi che vengono scanditi iterativamente:

- Viene scelto un punto p dall’insieme dei punti
- Si calcolano tutti i punti raggiungibili in densità dal punto p selezionato dati i parametri ε e Minpts
- Se p è un *core point* allora si può procedere alla formazione del suo *cluster*, il quale verrà iterativamente espanso.
- Se p è un *border point*, non vi sono sufficienti punti raggiungibili in densità da p e l’algoritmo di DBSCAN visita il punto successivo dell'insieme dei dati
- Il processo continua fino a quando tutti i punti presenti nell’elenco iniziale non sono terminati.

Una volta terminato il processo di creazione degli insiemi di dati essi possono essere caratterizzati calcolando per ognuno degli indici statistici significativi sul campione rappresentato da ogni *cluster*. Alcuni indici, che vado ad utilizzare per caratterizzare il campione, sono ad esempio la media e la varianza campionarie, è inoltre necessario tenere sempre presente il peso che l’insieme osservato rappresenta rispetto alla totalità dei dati. Preservare le informazioni contenute negli *outlier* rimossi dal *cluster* principale risulta molto importante. Benchè essi siano difficili da gestire singolarmente il loro insieme fornisce informazioni importanti che se aggregate possono migliorare il potere predittivo di un modello generato solo con l'insieme di dati principale.
Una volta terminata la fase di computazione dei due modelli si procede a collegare il traffico di richieste prodotto dal servizio, rappresentato dai due modelli generati, ad un server caratterizzato da un certo tempo di servizio rappresentato dal parametro μ.
Le predizioni verranno quindi effettuate mediante i valori di tempo di risposta calcolati su tale macchina. Il server utilizzato può sia essere un server reale di cui si conosce come si comporta il tempo di servizio che uno fittizio. Prendendo come riferimento il tempo di risposta si otterranno due distinte predizioni dai due differenti modelli. Tali predizioni saranno difficilmente precise per via del fatto che, in molti casi, verranno ottenute mediante approssimazioni dovute all’impossibilità di utilizzare le formule appartenenti alla teoria delle code in alcuni casi limite. Andando però ad aggregare i risultati ottenuti in un unico valore la precisione della predizione aumenterà sensibilmente. Quanto essa risulterà precisa dipenderà dal confronto con il valore atteso ottenuto mediante l’uso di un metodo simulativo sulla stessa macchina e con gli stessi dati in ingresso. Per aggregare i due valori ottenuti ho scelto di utilizzare tecniche di *model averaging* che veranno approfondite più nel dettaglio nei capitoli successivi. Per ora basti sapere che le due predizioni vengono unite computando la media dei due valori pesandoli sulla base della porzione di dati che vanno a rappresentare. Il modello caratterizzante gli *outlier* sarà quindi generalmente associato ad un peso molto basso ma fornirà un contributo al tempo di rispsota elevato. Il risultato finale ottenuto rappresenta il tempo di risposta aggregato dei due modelli, il cui valore risulterà molto più vicino al valore ottenuto mediante

la simulazione rispetto a quello calcolato partendo dalla media di tutti i valori presenti nel log. Questa differenza di prestazioni può trovare spiegazione nella forma della funzione caratterizzante il tempo di risposta (figura 4.2), essa non è lineare ma bensì esponenziale con una curva sempre più accentuata all'aumentare del carico sul server. Questo giustifica il contributo elevato fornito dalla parte di *outlier* nonostante il suo peso sia minimo.

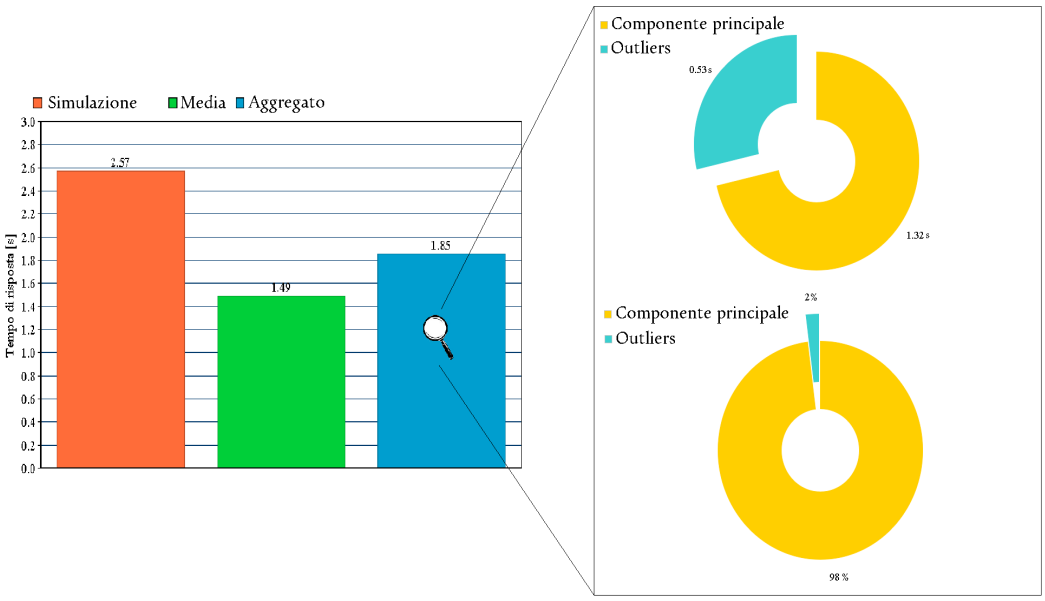

Figura 4.4. Rappresentazione di un valore di tempo di risposta calcolato mediante approccio simulativo, media dei valori e aggregando la componente principale e gli *outlier* dopo aver calcolato separatamente il tempo di risposta.

4.2 Suddivisione dei dati mediante la computazione di limiti

Con questo metodo, così come per il DBSCAN esposto nella sezione 4.1, si parte dall'idea di andare a rimuovere i valori eccessivamente elevati dai dati, senza però perdere le informazioni in essi contenute, al fine di riuscire a rappresentare in modo più preciso l'insieme dei campioni a disposizione. Per tenere conto delle informazioni rimosse per generare il secondo modello, cosi come per il DBSCAN, si utilizzerà un metodo di aggregazione che verrà presentato più nel dettaglio successivamente.
Preso ancora una volta come riferimento il modello descritto nel capitolo 3 si desidera andare ad effettuare una classificazione binaria [10] dei dati da modellizzare. Volendo utilizzare il log contenente le richieste inviate dal servizio l'algoritmo va ad avvalersi di una tecnica volta alla computazione di un limite che verrà utilizzato per attuare una separazione lineare tra l'insieme dei dati e quello degli *outlier*. Per calcolare tale limite ho scelto di utilizzare come punto di partenza la mediana dei dati, essa infatti, a differenza della media, fornisce un valore meno suscettibile ai valori anomali e quindi generalmente più affidabile.

Con la tecnica proposta si va ad utilizzare, assieme alla mediana computata, un termine additivo il cui valore viene determinato moltiplicando la varianza campionaria con una costante K positiva scelta opportunamente. Selezionando correttamente tale costante il limite che si andrà a calcolare fungerà da separatore tra i valori appartenenti all'insieme principale e quelli troppo elevati da considerare separatamente.

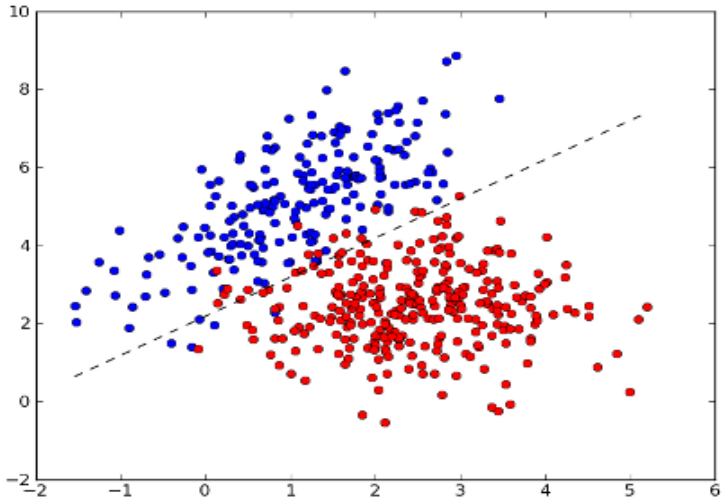

Figura 4.5. Ripartizione di un insieme di punti mediante l'uso di un classificatore binario.

In figura 4.5 è possibile vedere un generico classificatore binario che opera nel medesimo modo del metodo proposto. Nel caso in esame però il limite viene posto orizzontalmente rispetto al grafico rappresentate i dati e si desidera classificare i valori in modo tale che la maggior parte di essi ricadano nell'insieme principale e solo un certo numero di valori eccessivamente elevati venga classificato nell'insieme secondario.
Tutti i campioni che andranno a trovarsi al di sopra del limite computato verranno considerati come non appartenenti alla classe principale e quindi rimossi durante la prima fase di calcolo, quelli che invece si troveranno al di sotto verranno classificati come appartenenti al modello principale.
Dopo aver assegnato ogni campione al proprio insieme di appartenenza, mediante il sistema di classificazione binaria proposto, sarà possibile andare a calcolare separatamente tutti gli indici statistici che caratterizzano i due insiemi, cosi come è stato fatto nel capitolo precedente per gli insiemi generati mediante l'uso della tecnica DBSCAN.
Partendo dalle due medie campionarie associate ad i due insiemi è possibile realizzare predizioni su valori significativi, quali ad esempio il tempo di risposta di un ipotetico server caratterizzato da un certo parametro di servizio.
Come visibile in figura 4.2 la funzione caratterizzante il tempo di risposta in funzione dell'utilizzo di una macchina non ha un andamento lineare ma esponenziale e computare separatamente i tempi di risposta dei due insiemi per poi aggregarli evita di andare a sottostimarne il valore, fornendo una predizione

più precisa. Un risultato di livello inferiore si avrebbe invece andando ad utilizzare come valore di partenza la media campionaria ottenuta considerando tutti i dati come se fossero in un unico insieme. Il risultato finale ottenuto mediante questa tecnica sarà un unico valore di tempo di risposta o di una qualsiasi misura rappresentativa del funzionamento del sistema che si desiderava predire, rappresentate il contributo dei due insiemi generati.

4.3 Ricerca di pattern

La tecnica di ricerca di *pattern* [11] che vado a presentare in questo capitolo, a differenza di quelle esposte nei capitoli 4.1 e 4.2, non è focalizzata sulla individuazione e rimozione di una piccola porzione dei dati a disposizione ma sull'individuare comportamenti diversi delle richieste di servizio. Usando questo metodo si teorizza che in periodi temporali differenti possano esistere profili di traffico diversi. Prendendo ancora come riferimento il modello a rete di code proposto nel capitolo 3 è possibile andare ad applicare questa tecnica su uno dei log dei dati a disposizione. Presa ad esempio la registrazione delle richieste inviate dal servizio al servente e conoscendo la data ed ora di inizio di campionamento, oltre al tempo che intercorre tra la registrazione di un valore e l'altro, è possibile collocare l'intero log su di un asse temporale arricchendo quindi ogni campione registrato con una informazione aggiuntiva. Disponendo ora della dimensione temporale si può procedere ad applicare tecniche di selezione dei dati differenti da quelle esposte in precedenza. Conoscendo l'istante in cui è stato registrato ogni campione è infatti possibile suddividere i dati secondo diversi criteri ottenendo insiemi i cui i valori in esso contenuti potrebbero evidenziare modalità diverse di funzionamento. I *pattern* utilizzabili per la suddivisione dei campioni sono molteplici, ed in alcuni casi potrebbero richiedere di ripartire i dati in un numero elevato di insiemi. Ad esempio un analista potrebbe decidere di voler studiare separatamente il traffico tra server e servizio di ogni giorno della settimana, utilizzando quindi sette insiemi diversi. Ipotizzando inoltre un uso notturno molto simile del servizio, a parte nel periodo del fine settimana, durante il quale potrebbero essere state programmate operazioni automatizzate, si potrebbe decidere di raffinare ulteriormente il criterio di ricerca includendo un ottavo insieme per tutti i valori registrati in ore notturne ed un nono per accogliere i campioni registrati durante la notte del fine settimana. Il criterio di ricerca da utilizzare dipende molto da quale si pensa possa caratterizzare profili di funzionamento differenti e gli unici limiti sono dettati dal buon senso. Di seguito riporto una serie di *pattern* comunemente ricercati all'interno dei dati, in cui intuitivamente potrebbe verificarsi un quantitativo di richieste differente proveniente dal servizio.

- Periodo diurno - Periodo notturno
- Settimana - Fine settimana
- Periodo diurno settimanale - Periodo diurno fine settimana - Periodo notturno settimanale - Periodo notturno fine settimana

Scegliendo uno dei *pattern* più comuni qui presentati o utilizzandone uno concepito appositamente si otterrà un numero variabile di insiemi. Da ognuno dei gruppi di campioni ottenuti sarà possibile andare a computare degli indici statistici rappresentativi, solitamente media e varianza campionaria.
Nel caso si stia usando il log delle richieste presente nel modello proposto nel capitolo 3 calcolandone la media compionaria si otterrà il numero di richieste medio stimato. Partendo dai valori ottenuti sarà possibile, utilizzando un server il cui tempo di servizio è caratterizzato da un certo parametro μ ed applicando alcune semplici relazioni della teoria delle code, risalire ai tempi di risposta associati. Tali valori andranno poi aggregati fornendo un risultato più preciso rispetto a quello che si otterrebbe andando a calcolare il tempo di risposta del servente partendo dalla media di tutti i dati posti in un unico insieme. Questo perchè la funzione caratterizzante il tempo di risposta di una macchina al variare delle risorse disponibili non ha un andamento lineare ma bensì esponenziale ed è quindi opportuno computare separatamente i due valori.

4.4 Uso di reti neurali come modello

Andare ad utilizzare una rete neurale come modello potrebbe rappresentare una alternativa rispetto ai metodi caratterizzati da un approccio statistico proposti nei capitoli precedenti. Utilizzando sempre come riferimento il modello illustrato nel capitolo 3, se in precedenza l'idea alla base era quella di andare a modellizzare le informazioni contenute nei log mediante l'uso di indicatori statistici, quali media e varianza campionaria, con questo metodo ci si propone di cambiare completamente il modello ed utilizzare una rete neurale. Tale struttura, ricevendo in ingresso una serie di esempi che la guidino nell'apprendimento e generati mediante l'uso delle informazioni contenute nei log a disposizione, dopo un corretto addestramento potrebbe riuscire a fornire come risultato le predizioni desiderate. Nel caso del modello usato essa dovrà fornire, in modo sufficientemente preciso, il tempo di risposta per una serie di campioni ricevuti in ingresso. Questo tempo di risposta dovrà essere sufficientemente vicino a quello computato mediante l'uso di un metodo simulativo applicato ad un ipotetico sistema. Tale simulazione verrà eseguita con i medesimi dati in ingresso e con un tempo di servizio caratterizzato da un parametro μ definito opportunamente.
Nei paragrafi successivi cercherò di rendere maggiormente chiaro cosa si intende con rete neurale partendo dalla più semplice possibile, il percettrone, fino a giungere alle reti multistrato, più complesse e con maggiore capacità di apprendimento. Presenterò inoltre un metodo utile per l'addestramento di queste ultime ed una tecnica volta ad arginare il problema del sovraddamento.

4.4.1 Definizione di rete neurale e percettrone

Una rete neurale è un modello computazionale parallelo, costituito da numerose unità elaborative omogenee, dette neuroni, fortemente interconnesse mediante collegamenti di varia intensità. Esse sono ispirate al sistema nervoso centrale animale, più precisamente al cervello. Nella maggior parte dei casi una rete neurale artificiale è un sistema adattivo che cambia la sua struttura basandosi su informazioni esterne o interne che scorrono attraverso la rete stessa durante la fase di apprendimento. Una classe di modelli statistici può essere chiamata neurale se possiede le seguenti caratteristiche:

- Consiste in un insieme di pesi adattativi che vengono modificati mediante un algoritmo di apprendimento.
- E' in grado di approssimare funzioni non lineari dei suoi input.

Queste strutture sono state sviluppate sin dai primi anni '40. Nel 1943 Warren McCulloch e Walter Pitts crearono un primo modello computazionale basato su algoritmi matematici con il nome di *Threshold logic* [12]. Sempre negli anni '40 lo psicologo Donald Hebb creò una teoria di apprendimento basata sul meccanismo di plasticità neuronale conosciuta anche come *Hebbian Learning* [13], considerata come un esempio tipico di apprendimento non supervisionato. Successivamente, nel 1954, Farley and Wesley A. Clark usarono i calcolatori dell'MIT per simulare una *Hebbian Network* [14] . Nel 1958, infine, Frank Rosenblatt introdusse il concetto di Percettrone [15], un caso particolare di rete neurale usato attualmente per la classificazione lineare di *pattern*.

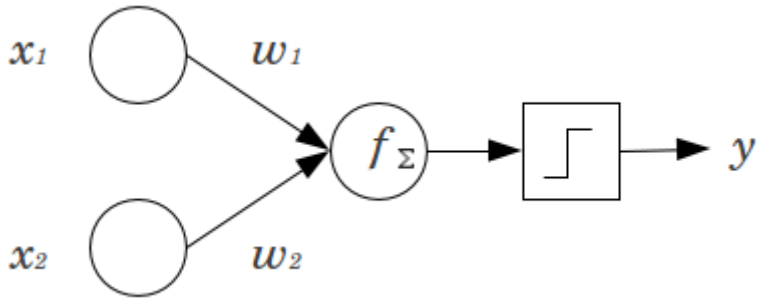

Figura 4.6. Rappresentazione di un percettrone.

L'unità chiamata percettrone, visibile in figura 4.6, consiste in un singolo neurone caratterizzato da una propria funzione di attivazione, da dei pesi sinaptici ed una soglia modificabili.

Il suo oputput può essere computato come segue:

$$y = 1 \text{ se } \sum_{i=0}^{n} w_i x_i > 0 \quad \text{altrimenti - 1} \qquad (4.1)$$

Negli anni ‘50 e ‘60 si visse, nel campo della ricerca sulle reti neurali, sull’onda di un grande entusiasmo, che fu però smorzata nel 1969 da Marvin Minsky e Seymour Papert i quali mostrarono alla comunità di ricercatori i limiti di un singolo strato di percettroni [16]. Un percettrone infatti non è in grado di apprendere funzioni non linearmente separabili quali ad esempio lo XOR. Questa scoperta ridimensionò di molto le aspettative sulle capacità delle reti neurali raffreddando il grande entusiasmo che aveva contraddistinto la ricerca in questo campo nei primi anni del suo sviluppo.

4.4.2 Reti multistrato e tecniche di addestramento

Dopo aver appreso i limiti del singolo percettrone vi fu una battuta d’arresto nello sviluppo delle reti neurali che durò alcuni anni. Successivamente si iniziarono a sperimentare altre topologie di rete al fine di poter risolvere problemi più complessi di quelli solo linearmente separabili, nel tentativo di superare i limiti evidenziati in precedenza. Connettendo più strati di percettroni si realizzanono le prime reti multistrato *feedforward* in grado di trattare problemi prima difficilmente gestibili e con capacità di apprendimento superiori a quelle del singolo percettrone.

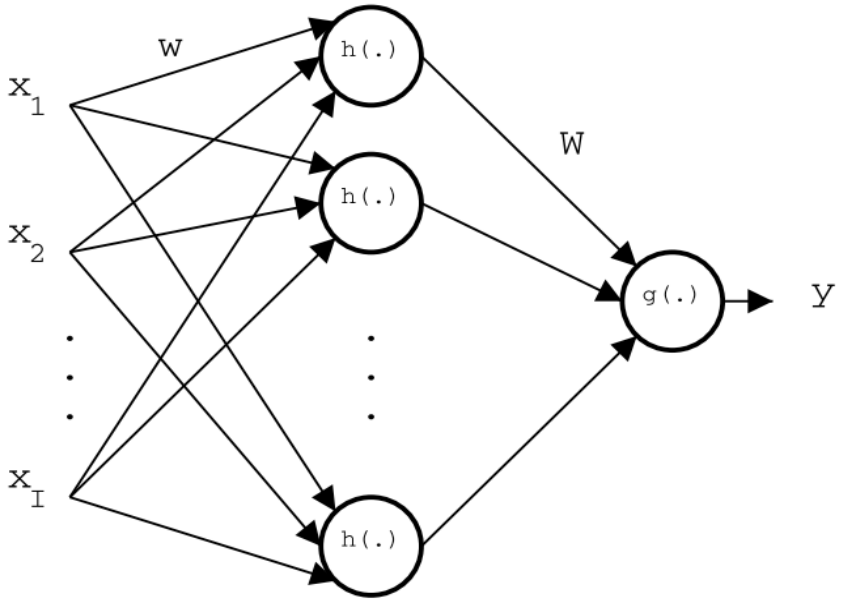

Figura 4.7. Rete neurale multistrato.

In una rete neurale multistrato (figura 4.7) ogni neurone di ogni strato è connesso ai neuroni dello strato successivo creando cosi una serie di livelli che partendo dallo strato di ingresso guidano il flusso verso lo strato di uscita. Gli strati interni vengono generalmente chiamati strati nascosti e per essi, solitamente, si sceglie come funzione di attivazione una sigmoide o una tangente iperbolica, al fine di rimuovere la relazione di linearità tra ingresso ed uscita e riuscire a mantenere i valori che scorrono all'interno della rete limitati.
La formula utilizzata per andare a computare la funzione di output è la seguente:

$$Y = g\left(\sum_{j}^{J} h\left(\sum_{i}^{I} w_{ji} x_i\right)\right) \qquad (4.2)$$

Mentre per avere una stima dell'errore commesso:

$$E = \sum_{n}^{N} \left(t_n - y_n\right) \qquad (4.3)$$

con t_n n-esimo output atteso.
Per questa classe di reti neurali è possibile utilizzare svariate tecniche di apprendimento, ma la più nota è sicuramente la *backpropagation*. Mediante questo metodo i valori forniti in uscita vengono confrontati con quelli attesi computando l'errore commesso tramite una apposita funzione, scelta in precedenza, detta funzione di errore. Con la *backpropagation* si vuole riuscire a minimizzare l'errore commesso sfruttando come stima di esso il risultato fornito dalla funzione di errore. Dopo aver determinato il suo ammontare esso viene propagato all'indietro lungo la rete partendo dallo strato di uscita in direzione di quello di ingresso, aggiornando iterativamente i pesi posti sugli archi di connessione dei neuroni lungo il percorso. Questo aggiornamento viene effettuato mediante una tecnica di ottimizzazione non lineare chiamata gradiente discendente ed è importante notare come sia necessario utilizzare funzioni di attivazione per i nodi della rete derivabili, affinchè essa risulti applicabile.
In riferimento alla figura 4.7 riporto di seguito le formule per l'aggiornamento dei pesi della rete neurale multistrato rappresentata secondo la tecnica di propagazione all'indietro dell'errore:

$$W_j^{t+1} = W_j^t + 2\eta \sum_{n}^{N} \left(t - g(A)\right) g'(A) b_j \qquad (4.4)$$

$$w_{ji}^{t+1} = w_{ji}^t + 2\eta \sum_{n}^{N} \left(t - g(A)\right) g'(A) W_j h'(a_j) x_i \qquad (4.5)$$

Con $a_j = \sum_{i}^{I} W_j b_j$ detto valore di attivazione, $b_j = h(a_j)$ valore di uscita dello j-esimo nodo dello strato nascosto, η coefficiente di apprendimento e $A = \sum_{j}^{J} W_j b_j$.

Andando a minimizzare progressivamente l'errore si otterrà una configurazione dei pesi per la rete che potrà essere ritenuta sufficientemente buona per l'apprendimento della funzione desiderata. Per giungere a tale configurazione potrebbe risultare necessario iterare il processo di aggiornamento dei pesi anche per molti cicli fino a quando la rete non avrà raggiunto uno stato ritenuto accettabile, affermando cosi che essa ha appreso correttamente la funzione desiderata.

4.4.3 Arginare il problema dell'overfitting

L'apprendimento effettuato mediante reti neurali rientra nella categoria dell'apprendimento supervisionato, pertanto, per riuscire ad addestrare in modo efficace una rete è necessario disporre di un insieme di addestramento. Per insieme di addestramento si intende un insieme di combinazioni ingresso-uscita, in grado di rappresentare in modo sufficientemente preciso la funzione che si desidera apprendere. L'insieme di campioni selezionati, per essere soddisfacente, deve avere una dimensione sufficientemente ampia tale per cui al suo interno vi sia un numero di esempi minimo per rappresentare la funzione correttamente. Oltre al numero di esempi è rilevante anche la qualità degli stessi, avere molteplici esempi che vanno a coprire solo alcuni casi possibili renderà difficile apprendere correttamente la funzione, è necessario avere una corretta stratificazione dei campioni in modo tale che tutti i possibili casi abbiano un proprio rappresentante tra di essi. Il modo migliore per riassumere tutti i requisiti che deve avere un insieme di addestramento è quello di dire che esso deve essere uno specchio della realtà che si desidera modellizzare.
Al termine del processo di addestramento si può procedere ad una fase di test nella quale si va ad utilizzare un secondo insieme di esempi, sconosciuto alla rete neurale e correttamente stratificato, chiamato insieme di test. Ai fini della valutazione della rete risulta molto importante riuscire ad avere un buon risultato sugli esempi presenti in esso non incorrendo cosi nel problema del sovradattamento.
In statistica ed in informatica, si parla di sovradattamento (*overfitting*) quando un modello statistico si adatta ai campioni osservati usando un numero eccessivo di parametri e perdendo cosi di generalità. Nel caso delle reti neurali si può parlare di sovradattamento quando una rete si adatta eccessivamente all'insieme di addestramento ottenendo buoni risultati su di esso ma fornendo cattive prestazioni in fase di test. Un modello assurdo e sbagliato può adattarsi perfettamente, se è abbastanza complesso rispetto alla quantità di dati disponibili nell'insieme di addestramento, ma non per questo si può dire che esso è in grado di rappresentare correttamente gli aspetti che si desiderano raccogliere della realtà. Esistono svariate tecniche per arginare il problema dell'overfitting nell'ambito delle reti neurali, consentendo cosi di migliorare la generalizzazione del modello. Tra i metodi più celebri troviamo l'*Early Stopping* e il *Weight Decay*.

Il metodo dell'*Early Stopping* si fonda sull'idea di interrompere anticipatamente il processo di apprendimento sapendo che da una certa iterazione dell'algoritmo di aggiornamento dei pesi in poi, sebbene l'errore sull'insieme di addestramento tenda a decrescere, l'errore sull'insieme di test inizierà ad aumentare.

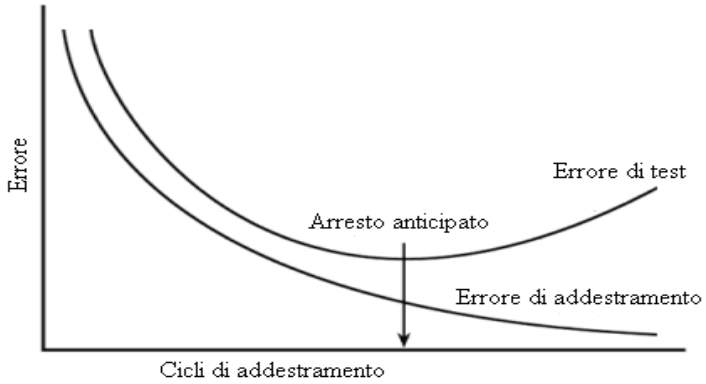

Figura 4.6. Andamento degli errori commessi sul l'insieme di addestramento e su quello di test e scelta del punto ottimale per arrestare l'addestramento di una rete.

Arrestando anticipatamente il processo di apprendimento ed andando a scegliere il passo di addestramento più appropriato, come mostrato in figura 4.6, si cerca quindi di ottenere il miglior risultato possibile sull'insieme di test ignorando il miglioramento delle prestazioni su quello di addestramento che non porterebbe benefici a livello generale.

"For valid generalization, the size of the weights is more important than the size of the network"[17]

Questo è il concetto di fondo del *Weight decay*, partendo quindi da questa idea si cerca di spingere la rete verso configurazioni caratterizzate da pesi minori che dovrebbero quindi favorire una maggiore generalizzazione del problema.
Con questa tecnica viene sommato un termine additivo alla funzione di errore influenzando la configurazione dei pesi per la rete. Usualmente questo termine viene computato come segue:

$$\hat{w} = argmin_w \sum_n^N (t_n - y)^2 + \gamma \sum_m^M w^2 \qquad (4.6)$$

Con γ coefficiente di decadimento opportunamente scelta.
Differenti valori della stessa costante determineranno comportamenti diversi della rete durante l'addestramento e la conseguente scelta di configurazioni diverse. Con una selezione opportuna il termine additivo contribuirà dunque a penalizzare configurazioni di pesi con valori elevati e favorire quelle reputate più semplici. Utilizzando questo metodo si andranno cosi a creare reti meno complesse riuscendo ad arginare il problema del sovradattamento.

Capitolo 5

Presentazione di casi di studio reali

In questo capitolo presenterò due casi di studio reali i cui dati sono stati utilizzati, durante il progetto di tesi, per l'applicazione delle tecniche, esposte nei capitoli precedenti, volte ad identificare e gestire l'incertezza.
La fase di acquisizione delle informazioni è stata realizzata mediante l'uso di *script* sviluppati appositamente per la tipologia di dato he si desiderava trattare. Per il caso di studio Unizar, che verrà presentato nella sezione 5.1, i dati sono stati forniti all'inizio del progetto e non è stata necessaria alcuna acquisizione.
I diversi sistemi reali sono stati rappresentati mediante l'uso di modelli a rete di code atti a raccogliere gli aspetti dela realtà ritenuti importanti per le analisi. Partendo da essi, grazie all'ausilio del software realizzato, è stato possibile, analizzando i log registrati, procedere all'esecuzione delle tecniche volte ad identificare e gestire l'incertezza sui dati a disposizione. Modellizzando il traffico delle richieste intercorse tra il server e i centri di servizio è stato possibile realizzare previsioni sul tempo di risposta del servente rappresentato. I risultati ottenuti sono stati poi confrontati con quelli computati mediante metodi simulativi, più affidabili e computazionalmente dispendiosi, al fine di valutare la bontà dei modelli realizzati e quindi dei metodi proposti per la loro costruzione.

5.1 Unizar

Il primo caso di studio che vado a presentare è quello chiamato Unizar, riferito a dati acquisiti dall'Università di Saragozza. Questo istituto pubblico è situato nell'omonima città di Saragozza nella regione di Aragon della spagna Settentrionale. La struttura nacque nel settimo secolo come scuola a gestione eclesiastica per poi divenire nel 1400, un centro di insegnamento delle arti libere e successivamente nel 1542 fu ufficialmente riconosciuta con il titolo di università. Attualmente, nella sua interezza, l'università di Saragozza conta quasi 37000 studenti ed uno staff di circa 6000 collaboratori ripartiti su diversi campus nelle città di Jaca, Huesca, La almunia de Dona Godina, Teruel e Saragozza [18]. Tutti i dipendenti e gli studenti dell'università si avvalgono di un servizio di posta elettronica per la gestione della propria corrispondenza, la cui architettura, di tipo centralizzata, è visibile in figura 5.1.

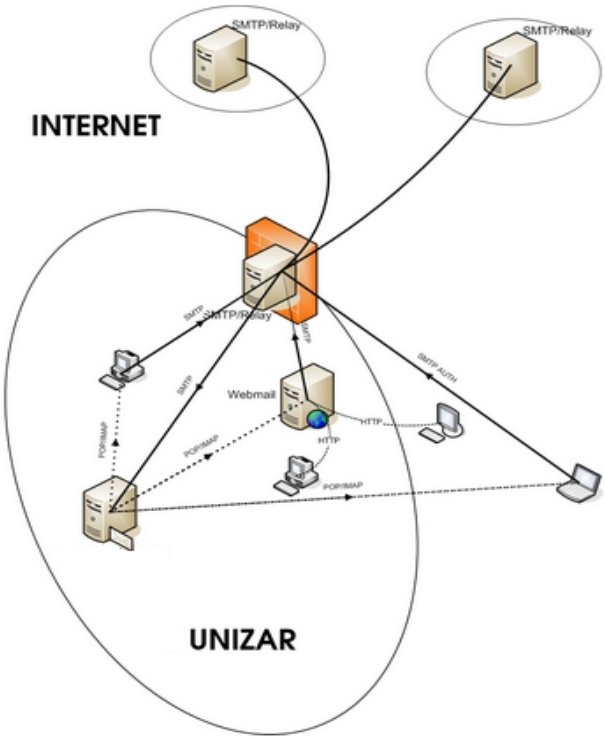

Figura 5.1. Architettura di sistema della mail dell'università di Saragozza.

Il traffico SMTP (*Simple Mail Transfer Protocol*) tra l'università ed internet e tra i sistemi interni è gestito da *proxy* che sono responsabili dell'instradamento di tutti i messaggi, di filtrare i contenuti e rilevare lo *spam* agendo come veri e proprio *firewall* a livello applicativo. Di conseguenza solo le macchine che fungono da *proxy* di primo livello sono autorizzate a stabilire canali SMTP con l'esterno e nessun sistema al di fuori dalla rete è in grado di instaurare una connessione SMTP con qualsiasi macchina presente sulla rete interna senza dover passare per i filtri. Un'architettura centralizzata del tipo mostrato in figura 5.1 comporta un alto rischio di interruzione nel flusso di messaggi causabile da picchi di lavoro o malfunzionamenti delle macchine che fungono da nodi primari; è inoltre da tenere in considerazione l'elevata vulnerabilità ad attacchi di tipo "*denial of service*" (DoS). Per ridurre al minimo questo rischio, l'università di Saragozza ha scelto di utilizzare sistemi ad alta disponibilità e bilanciamento del carico al fine di garantire continuità del servizio ed evitare quindi ritardi nella consegna dei messaggi di posta.
In figura 5.2 è possibile vedere la schermata di controllo del servizio di posta dell'università in cui compaiono tutti i server appartenenti al sistema.

Servidores POP3/IMAP			
piedra.unizar.es	-91	The system reject mailMan web access	Mon Nov 24 04:59:15 2014
mesa.unizar.es			Thu Jan 1 01:00:00 1970
gallego.unizar.es			Thu Jan 1 01:00:00 1970
posta1.unizar.es	0	The system status is normal	Fri Oct 24 08:35:17 2014
posta2.unizar.es	0	The system status is normal	Fri Oct 24 08:35:17 2014
posta3.unizar.es	0	The system status is normal	Fri Oct 24 08:35:17 2014
posta4.unizar.es	0	The system status is normal	Fri Oct 24 08:35:17 2014
celes1.unizar.es	0	The system status is normal	Fri Oct 24 08:35:17 2014
grio.unizar.es	-35	"Too many messages in the queue mqueue: 11086"	Thu Nov 20 20:40:10 2014
queiles.unizar.es	10	"The file system '/home/celes1' exceeds the 97% (99%)"	Sun Nov 23 06:29:03 2014

Servicio Webmail	*0*	*The system status is normal*	*Tue Oct 7 10:33:45 2014*
piedra.unizar.es	-91	The system reject mailMan web access	Mon Nov 24 04:59:15 2014
mesa.unizar.es			Thu Jan 1 01:00:00 1970
gallego.unizar.es			Thu Jan 1 01:00:00 1970
tranquera.unizar.es	-1		

Figura 5.2 Schermata controllo servizio di posta università di Saragozza

Per motivi di studio il mio interesse è ricaduto sul servizio webmail composto dalle macchine Piedra, Mesa, Gallego e Tranquera, dal quale sono stati registrati i dati utilizzati nel corso delle analisi e forniti all'inizio della fase di progetto. L'acquisizione delle informazioni è avvenuta durante un periodo di osservazione compreso tra il 1 ed il 31 Luglio 2013 e tra il 12 Novembre ed il 31 Dicembre 2013. La frequenza di campionamento adottata per la registrazione è di 60 secondi/campione, generando cosi circa 120.000 valori. Gli elementi interessati del sistema, durante l'acquisizione delle informazioni, sono stati il server, di cui si è registrato il carico, ed il client, di cui si è tenuta traccia delle richieste inviate. I dati raccolti durante tale periodo hanno prodotto i seguenti log:

- Log di carico sul server: Carico registrato sul server nei vari istanti di campionamento.
- Log delle richieste inviate dal servizio webmail: Numero di richieste di servizio provenienti dal servizio webmail verso il server nei vari istanti di campionamento, dal quale è possibile ricavare la frequenza media di invio delle richieste.

Per modellizzare il caso Unizar, la cui struttura del sistema è molto simile a quella proposta nella situazione teorica esposta nei capitoli precedenti, ho scelto di utilizzare un modello a rete di code in grado di raccogliere gli aspetti rilevanti della realtà ritenuti importanti per le analisi.

Questo modello è caratterizzato da un solo servente, con un tempo di servizio sconosciuto ma ricavabile dai dati, a cui è collegato un unico client. Il servizio collegato è rappresentato dal servizio webmail, di cui è stata tenuta traccia delle richieste inviate.
In figura 5.3 è possibile vedere uno schema riassuntivo dei dati raccolti e di come essi siano legati al server ed al servizio webmail.

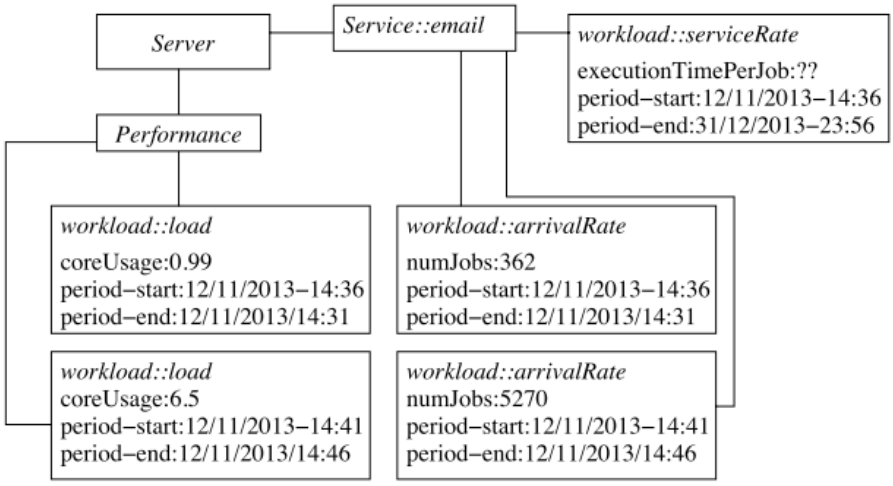

Figura 5.3. Modello Server- Servizio mail caratterizzante il caso di studio reale Unizar.

Le analisi per l'identificazione ed i metodi per la gestione dell'incertezza sono state applicate al modello a rete di code, descritto precedentemente, ed ai dati ad esso associati. In questo modo è stato possibile valutare il software realizzato al fine di automatizzare tale processo. Mediante il *tool* implementato sono stato in grado di computare modelli del traffico, presente nel log, al fine di ottenere predizioni del tempo di risposta del server in diverse condizioni di funzionamento. La valutazione della qualità delle predizioni, e di conseguenza dei metodi utilizzati, è stata effettuata confrontando i valori predetti con quelli ottenuti mediante la simulazione di un sistema a rete di code implementata dal software.

5.2 Wikipedia

In questa sezione presenterò il caso di studio Wikipedia i cui dati sono stati utilizzati al fine di avere risultati quanto più possibile affidabili. Questa organizzazione ha infatti reso pubblici il numero di richieste giunte alle sue varie pagine, dette *pagecounts* [19] , fornendo una serie di informazioni che, se estratte e trattate correttamente, avrebbero potuto ben adattarsi alle analisi che desideravo attuare.

Wikipedia è una enciclopedia libera che trova editori in tutto il mondo ed ha introdotto un nuovo metodo collaborativo per accedere ed aggiornare le informazioni. L'organizzazione ha iniziato a prestare i suoi servizi nel 2001 avendo a disposizione risorse molto limitate, l'intera struttura infatti veniva eseguita su di un unico server ed era composta da uno script Perl. Negli anni però Wikipedia è cresciuta enormemente, nel 2008 Domas Mituzas che lavorava nel supporto MySQL per Sun Microsystems, avendo la possibilità di accedere ad alcune informazioni riguardo i server di Wikipedia, ha condiviso alcune metriche interessanti [20] riguardo le operazioni svolte sulle sue pagine che danno bene l'idea di quanto in pochi anni l'organizzazione sia cresciuta:

- 50.000 richieste http al secondo
- 800.000 *queries* SQL al secondo
- 1.5 terabyte di dati compressi
- 200 server applicativi, 20 server di database e 70 server dedicati a *Squid cache*

Attualmente l'organizzazione fornisce i suoi servizi in 287 lingue, per un totale di oltre 32 milioni di articoli e 47 milioni di utenti registrati [21], i suoi server sono inoltre dislocati in molteplici aree geografiche al fine di fornire un servizio il quanto più possibile efficiente e stabile. Vista l'enorme mole di dati a disposizione e la complessità dell'organizzazione da studiare ho scelto di operare una selezione su di essi, al fine di utilizzare per il progetto quelle che potevano essere le aree linguistiche maggiormente rilevanti o più interessanti da analizzare. Anche il periodo temporale preso in considerazione è stato ridotto in modo da non dover gestire un eccessivo volume di dati ma allo stesso tempo avere un insieme di campioni sufficientemente ampio per poter condurre le analisi. Le aree linguistiche selezionate sono le seguenti:

- Italiana
- Inglese
- Tedesca
- Spagnola
- Francese
- Olandese
- Russa
- Svedese
- Catalana
- Lombarda

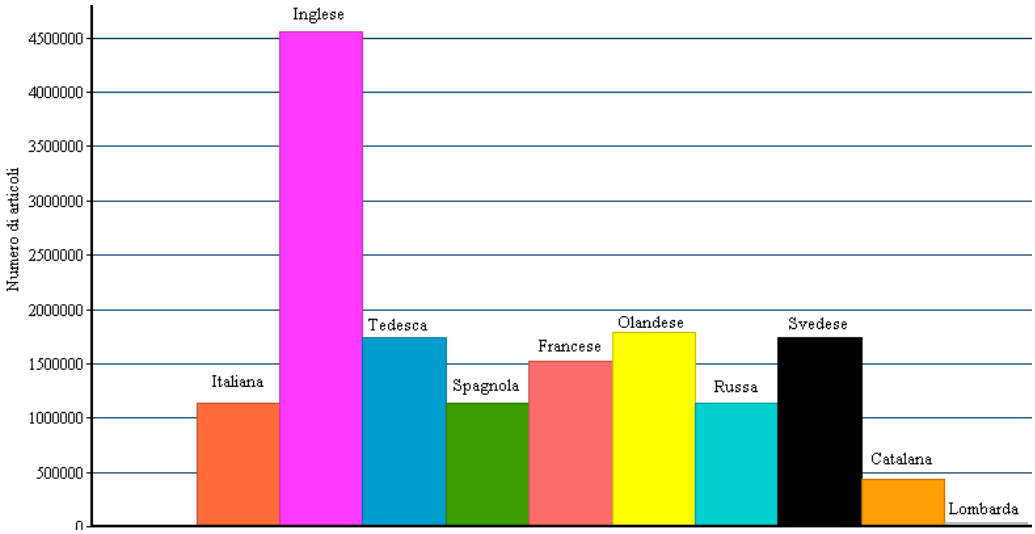

Figura 5.4 Numero di articoli presenti su Wikipedia in data 31 Ottobre 2014 per le dieci aree linguistiche selezionate per le analisi.

Come è possibile vedere in figura 5.4 le aree linguistiche selezionate sono caratterizzate da un numero di articoli realizzati anche molto differente e di conseguenza è ragionevole aspettarsi che anche il volume delle richieste alle pagine varierà in funzione di ciò.
I dati scaricati dai server di Wikipedia durante la fase di acquisizione coprono un periodo tra l'1 Gennaio 2013 ed il 30 Giugno 2013 e per ogni area linguistica sono stati registrati un totale di circa 4500 campioni. Il modello che ho scelto di utilizzare per andare a rappresentare in modo semplice il complesso sistema di Wikipedia, anche in questo caso, è una rete di code costituta da un unico servente, il cui tempo di servizio non è noto. Al server è collegato un unico centro di servizio da cui giungono la somma totale delle richieste per l'area linguistica osservata. I dati in mio possesso sono stati estratti dai log resi pubblici da Wikipedia eseguendo delle operazioni sulle informazioni in essi contenute per generare dei file che avessero un formato compatibile con il software realizzato. Volendo automatizzare il processo di estrazione, filtraggio ed aggregazione dei dati è stato realizzato uno *script* in ambiente .NET con interfaccia utente implementata mediante l'uso dei *Windows Forms* e programmazione ad eventi. Tale *script* si è occupato di scaricare sequenzialmente i file contenenti le informazioni richieste, per un volume di dati di circa 300gb. Per ogni ora di funzionamento Wikipedia ha messo a disposizione un archivio compresso contenente tutte le richieste a tutte le pagine ricevute negli ultimi sessanta minuti. L'intervallo di campionamento quindi per questo caso di studio varia ed è pari ad un ora, a differenza di quanto rilevato per Unizar dove si usava un intervallo di campionamento pari ad un minuto.

Listing 5.1: *Stralcio di log fornito da Wikipedia*

```
1    fr.b Special:Recherche/Achille_Baraguey_d%5C%27Hilliers 1 624
2    fr.b Special:Recherche/Acteurs_et_actrices_N 1 739
3    fr.b Special:Recherche/Agrippa_d/%27Aubign%C3%A9 1 743
4    fr.b Special:Recherche/All_Mixed_Up 1 730
5    fr.b Special:Recherche/Andr%C3%A9_Gazut.html 1 737
```

In 5.1, rappresentante uno stralcio di log fornito da Wikipedia, è possibile visionare la sintassi utilizzata dall'organizzazione per registrare le visite alla proprie pagine. Viene inoltre mostrato il traffico di dati che tali visite comportano, anche se questa informazione non è risultata rilevante ai fini delle analisi. Andando ad utilizzare una breve guida, messa a disposizione da Wikipedia, sono stato in grado di interpretare la sintassi utilizzata potendo cosi selezionare le informazioni di mio interesse. A seguire riporto l'interpretazione della porzione di log mostrata in 5.1:

- Il primo campo, fr.b fa riferimento alla sezione francese di Wikibooks e quindi all'area linguistica considerata.
- Il secondo campo si riferisce al nome della pagina richiesta da uno o più utenti nel corso dell'ora analizzata.
- Il terzo campo indica il numero di visite, non uniche, effettuate alla pagina di riferimento durante l'ora considerata.
- Il quarto ed ultimo campo fa riferimento al numero di byte trasferiti per soddisfare le richieste. Tale informazione non è risultata interessante per le analisi che si desideravano effettuare ed è stata filtrata dallo script in fase di acquisizione.

Lo *script* che ho realizzato per l'estrazione delle informazioni rilevanti contenute nei vari file scaricati ha dovuto necessariamente avvalersi di espressioni regolari per eseguire il processo. Con il termine espressione regolare si intende una sequenza di simboli che identifica un insieme di stringhe, essa può essere considerata come una funzione che prende in ingresso una stringa da analizzare e fornisce in uscita un valore di tipo booleano a seconda che la stringa segua o meno un certo pattern. Una volta scaricato un file lo *script* va ad operare una selezione sulle informazioni ritenute importanti. Per ogni area linguistica viene generato, dal log sotto esame, un valore, computato aggregando tutte le richieste presenti giunte da tale area. Questo valore viene poi salvato dallo *script* in un nuovo file associato all'area linguistica che l'ha generato. Nei log creati dal programma sono quindi presenti tante righe quanti sono stati gli istanti di campionamento, ed in ognuna di esse è possibile trovare un valore contenente le richieste totali giunte nell'istante rappresentato.

Listing 5.2: *Esempio log realizzato dai dati provenienti da Wikipedia per la regione linguistica catalana*

```
1       00 01 01 Jan 2013 Tuesday
2       39376
3       18565
4       18244
5       18334
6       17082
7       17162
8       16695
9       19355
```

In 5.2 è posibile vedere una porzione di uno dei file generati dal *tool*, nella quale sono state registrate le richieste giunte ai server di Wikipedia dall'area linguistica catalana. La prima riga indica l'istante di inizio del campionamento, nel caso specifico a mezzanotte ed un minuto del primo gennaio 2013. Tutte le righe successive contengono invece il numero di richieste giunte ad ogni ora a partire da tale istante. L'operazione di scaricamento e raffinamento dei dati provenienti dai server di Wikipedia, al fine di generare log analizzabili dai software realizzati, è risultata la più onerosa dal punto di vista temporale e ha richiesto quasi due mesi perchè fosse portata a completamento. Dopo tale periodo è stato possibile utilizzare i valori associati alle diverse aree linguistiche per la generazione dei modelli e le conseguenti analisi.

Capitolo 6

Il programma realizzato per lo studio dei dati

In questo capitolo andrò ad illustrare più nel dettaglio il *tool* realizzato e le funzionalità in esso implementate. Tale software è stato sviluppato con l'idea di andare a verificare le tecniche esposte nei capitoli precedenti, sia per quanto riguarda l'identificazione che per quanto riguarda la gestione dell'incertezza. La fase di sviluppo si è svolta in ambiente .NET andando ad utilizzare l'IDE Visual Studio nella sua versione Visual Studio 2013 Ultimate. Come linguaggio di programmazione è stato scelto C#. L'interfaccia utente è stata realizzata nel medesimo linguaggio e sfruttando le funzionalità messe a disposizione dalle librerie per l'utilizzo dei Windows Forms.
Nei capitoli successivi verranno mostrate delle schermate del programma in funzione mentre utilizza dati tratti dai casi di studio reali, esposti nel capitolo 5, al fine di far comprendere nel miglior modo possibile come funziona l'interfaccia grafica del *tool* e come è possibile avvalersi delle funzionalità implementate. Il programma, come già detto, è stato utilizzato con i dati acquisiti dall'Università di Saragozza e dai server di Wikipedia con l'idea di riuscire a proporre modelli utili ad effettuare previsioni per quanto riguarda il tempo di risposta del server per i casi di studio in esame.

6.1 La struttura utilizzata

La struttura utilizzata per il *tool* è stata influenzata dall'aver scelto di utilizzare i *Windows Forms* per la sua realizzazione. Le funzioni da implementare sono state distribuite su più pagine in modo da facilitare l'interazione tra l'utente ed il programma, designando una schermata principale come punto di partenza per effettuare tutte le analisi. L'utente ha quindi la possibilità di navigare all'interno dell'applicazione fino a raggiungere la funzione da lui desiderata, come mostrato dai collegamenti tra le pagine evidenziati in figura 6.1.

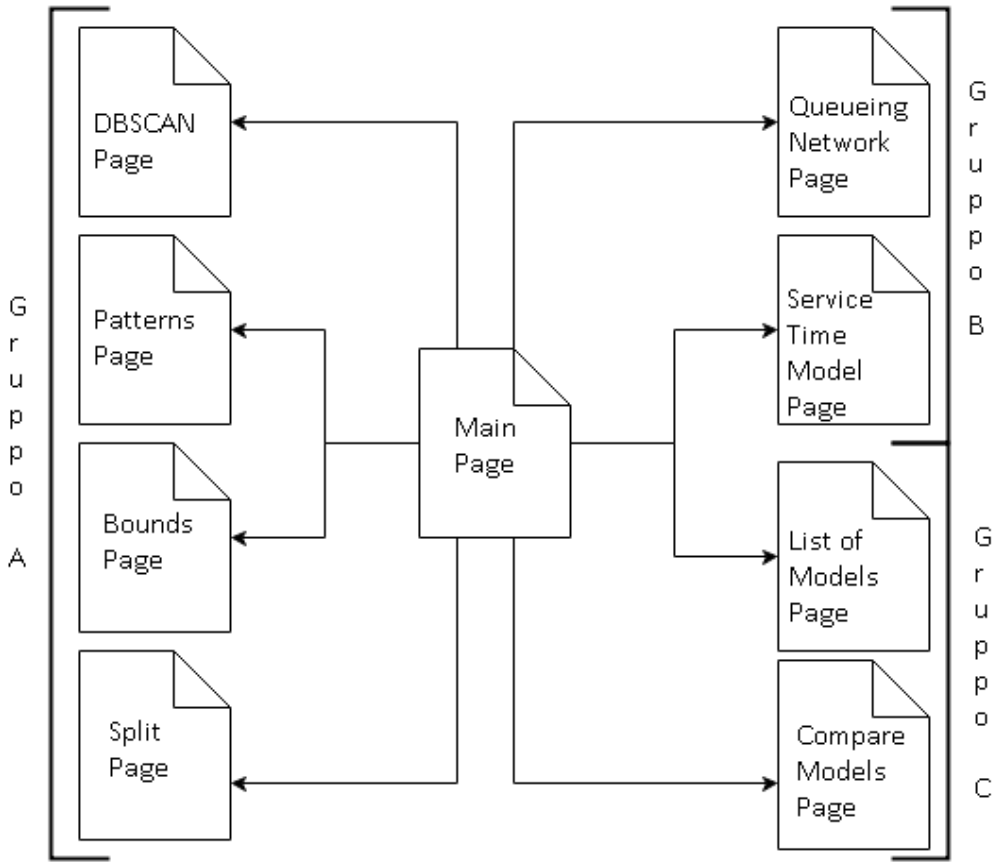

Figura 6.1. Architettura parte *front-end* del software realizzato

Come già detto, il processo di navigazione utilizza come punto di partenza la *Main Page*, rappresentante il cuore del programma. Le pagine ad essa collegate possono essere divise in gruppi secondo le funzionalità che vanno ad implementare. Al termine di ogni analisi eseguita l'utente viene ricondotto alla pagina principale da cui può riprendere il processo di analisi. Nel gruppo A, il più a sinistra in figura 6.1, sono presenti quattro pagine, le quali raccolgono tutte le implementazioni dei metodi per la creazione di modelli delle richieste utili per la gestione dell'incertezza. A questo gruppo appartengono:

- **DBSCAN Page**: In essa è implementato il metodo di DBSCAN presentato nella sezione 4.1.
- **Patterns Page**: Contenente l'implementazione del metodo di ricerca dei *pattern* presentato nella sezione 4.3.
- **Bounds Page**: In essa è implementato il metodo che mediante la computazione di limiti a partire da media e varianza ripartisce i dati orizzontalmente in differenti gruppi, tale metodo è stato presentato nella sezione 4.2.
- **Split Page**: Contenente un metodo che opera una divisione verticale del log caricato suddividendolo in differenti sezioni temporali.

La navigazione tra le varie pagine del programma realizzato viene gestita con la logica della programmazione ad eventi andando ad individuare le interazioni che avvengono tra l'utente e l'interfaccia mediante rilevamento delle azioni di mouse e tastiera sull'interfaccia grafica.

6.2 Main Page e sue funzionalità

La *Main Page*, come già detto, è il cuore del programma, tramite essa può accedere a molteplici opzioni per eseguire tutte le analisi implementate nel software. In questo capitolo descriverò le funzionalità immediatamente accessibili in tale pagina come esse possono essere utilizzate. All'avvio del programma all'utente verrà mostrato, nella parte superiore della finestra attiva, un menu contenente diverse opzioni.

Figura 6.2. Schermata pagina iniziale.

Inizialmente l'unica voce del menu selezionabile consente di indicare un file di log di cui si desidera effettuare il caricamento. I dati contenuti nel log verranno successivamente utilizzati per le diverse analisi realizzabili. I file di log, per essere riconoscibili dal programma devono rispettare una formattazione che consenta al software di interpretarli correttamente e trasformarli in un insieme di dati gestibile.

Le caratteristiche fondamentali perchè un file di log sia utilizzabile sono:

- La prima riga del file deve indicare, con il formato scelto, la data ed il tempo di inizio del campionamento.
- Ogni riga deve contenere un unico valore.
- Non devono essere presenti linee vuote all'interno o valori testuali estranei.

Listing 6.1: *Stralcio di log contenente le richieste registrate dall'università di Saragozza nel periodo compreso tra il 12 novembre 2013 ed il 31 dicembre 2013.*

```
1   14 36 12 Nov 2013 Tuesday
2   0.0
3   93.0
4   111.0
5   83.0
6   74.0
7   0.0
8   548.0
9   0.0
10  0.0
11  1513.0
12  0.0
13  1558.0
14  750.0
15  161.0
```

Come visibile in 6.1 la prima riga ha la funzione di andare ad indicare al programma che i dati, di cui sta per effettuare la lettura, sono stati registrati a partire da un certo istante temporale. I campi presenti in tale riga rispettano un ordine fissato e non mutabile:Ora, Minuto, Data, Mese, Anno e Giorno. In questo caso specifico la registrazione è iniziata alle ore 14:36 di martedì 12 novembre 2013. Tale informazione risulterà fondamentale nell'applicazione della tecnica di ricerca di *pattern* esposta nella sezione 4.3.
I formati riconosciuti dal programma come log di dati sono due:

- File con estensione .1min: In questo tipo di log il campionamento dei dati è stato effettuato ogni minuto.
- File con estensione .1hour: In questo tipo di log il campionamento dei dati è stato effettuato ogni ora.

Una volta caricati i dati il *tool* va a mostrare all'utente una serie di informazioni utili riguardo l'insieme di campioni, quali ad esempio media campionaria, varianza campionaria e la dimensione della popolazione dei dati. Vengono inoltre proposti dei test utili per diagnosticare se vi sia incertezza sui dati ed eventualmente identificare la tipologia che meglio ne spiega l'origine.
In figura 6.3 è mostrata la schermata principale del software e le informazioni fornite all'utente riguardo i dati che vengono utilizzati.

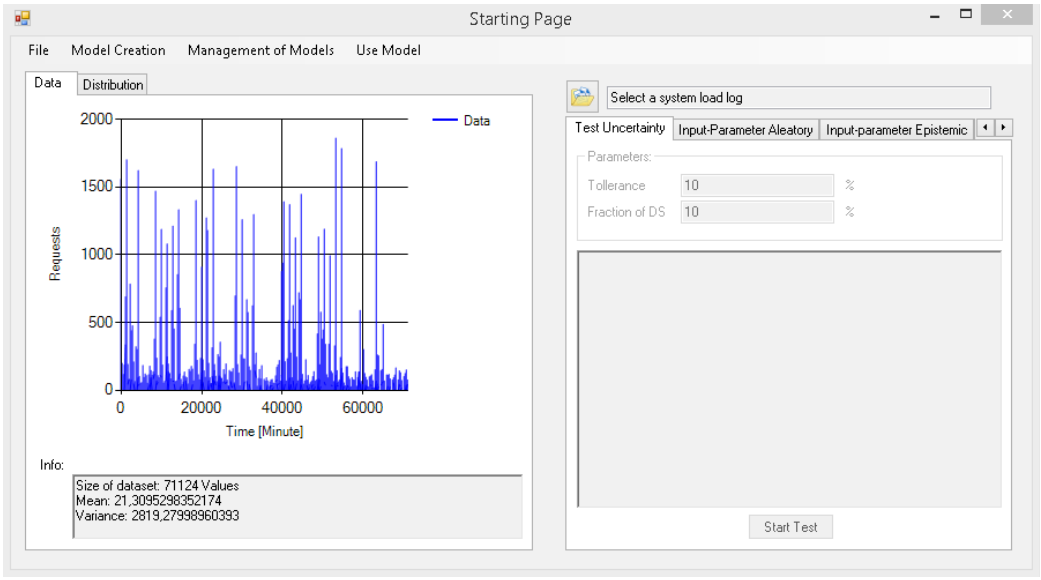

Figura 6.3. Schermata pagina principale con dati Unizar.

Nella parte sinistra della finestra (figura 6.3) è visibile un grafico, in colore blu, rappresentante l'andamento dei valori acquisiti dall'università di Saragozza. In tale grafico l'asse delle ascisse indica l'istante temporale osservato e l'asse delle ordinate il numero di richieste ad esso associato; a tale scopo sono state predisposte delle etichette che facilitano l'identificazione del periodo di campionamento e di cosa si sta rappresentando. Continuando ad analizzare la schermata principale, nella finestra testuale, posta immediatamente sotto il grafico dei dati, si possono trovare le informazioni riassuntive del *dataset*, precedentemente citate. In figura 6.4 è invece possibile vedere la schermata principale del software con un diverso campione di dati caricato. In questo caso viene mostrato l'andamento dei dati acquisiti da Wikipedia per l'area italiana.

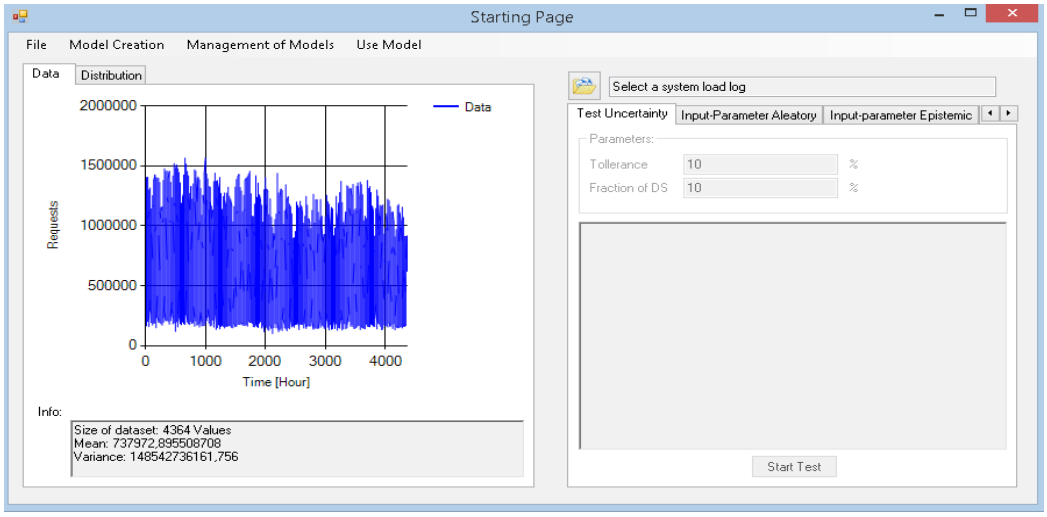

Figura 6.4. Schermata pagina principale dati Wikipedia area linguistica italiana

Osservando il grafico in figura 6.4 è possibile vedere come l'intervallo di campionamento utilizzato per i dati provenienti dai server di Wikipedia sia maggiore ed impostato ad un ora, inoltre le dimensioni dell'insieme dei campioni variano considerevolmente, passando da 71.000 campioni del caso Unizar a circa 4300 registrati per Wikipedia.
Il software, oltre a fornire tutte le informazioni riguardo al campione osservato descritte precedentemente, mette a disposizione ulteriori funzionalità per le analisi dell'utente. Nella parte sinistra della finestra attiva, andando a selezionare la scheda etichettata come *"Distribution"*, è possibile visionare il grafico della distribuzione dei dati. Questa informazione aggiuntiva in alcuni casi potrebbe rivelarsi utile per la realizzazione di modelli ma in altri si è rivelata fuorviante. Osservando la figura 6.5, è possibile vedere la distribuzione dei valori di carico sul server dell'Università di Saragozza nel periodo compreso tra il 12 novembre 2013 ed il 31 dicembre 2013.

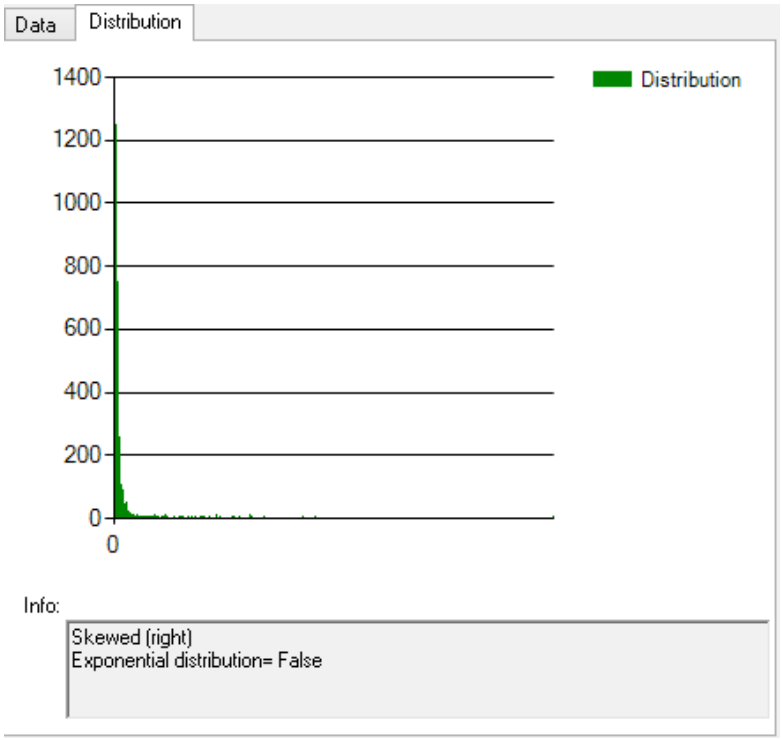

Figura 6.5. Distribuzione di dati rappresentanti il carico sul server appartenente all'Università di Saragozza.

La forma della funzione mostrata, inizialmente, mi ha portato a pensare che essa potesse essere una distribuzione esponenziale, convincendomi di poter rappresentare il carico sul server in modo sufficientemente preciso avvalendomi di questo tipo di distribuzione statistica. In realtà, effettuando alcune verifiche sulla varianza misurata e su quella calcolata partendo dalla media, utilizzando le relazioni conosciute per tale distribuzione, ho successivamente notato che i risultati trovati non coincidevano, portandomi cosi a scartare l'ipotesi iniziale.
Al fine di comprendere se si potesse trattare di una distribuzione esponenziale

ho implementato un test basato sulla seguente formula [22], la quale va a misurare l'asimmetria della distribuzione dei dati basandosi sul dispersione del campione:

$$Y = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{(\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2)^{\frac{3}{2}}} \qquad (6.1)$$

Con n numero di dati presenti nel *dataset*, x_i campione di indice i estratto dal *dataset* e $\overline{x}$ media aritmetica dei valori. Nel caso in cui Y risulti maggiore di 0 si potrà affermare che la distribuzione dei dati è asimmetrica a destra, caratteristica di una distribuzione esponenziale. Nel caso contrario, con un valore minore di 0 la distribuzione risulterà asimmetrica a sinistra. Se l'asimmetria rilevata dovesse essere corretta (destra) si può procedere a confrontare la differenza tra la varianza misurata e quella calcolata partendo dalla media, utilizzando relazioni note leganti media e varianza nella distribuzione esponenziale[23]

$$\text{Media} = \frac{1}{\lambda} \qquad (6.2)$$

$$\text{Varianza} = \frac{1}{\lambda^2} \qquad (6.3)$$

Nel caso in cui il modulo della differenza tra il valore calcolato e quello misurato risulti maggiore di una certa soglia sarà possibile affermare che non ci si trova di fronte ad una distribuzione esponenziale, stessa cosa si può dire nel caso di una asimmetria sinistra della funzione. Nel caso in cui, invece, la differenza tra le due medie dovesse essere tollerabile sarà possibile procedere con le analisi modellizzando il carico del server con tale distribuzione.

Tornando a descrivere le funzionalità implementate nella pagina principale del software, nella parte destra della schermata principale sono accessibili una serie di test utili a diagnosticare la presenza di incertezza ed individuarne la tipologia. Per poter usufruire di tali test è però necessario effettuare il caricamento di un log di dati aggiuntivo, contenente la registrazione del carico sul server per lo stesso periodo preso in considerazione dal log delle richieste che si sta utilizzando. Questo log risulta necessario al fine di poter ricavare l'utilizzo del server nei vari istanti temporali, in modo da poter utilizzare questa informazione durante l'esecuzione dei test, come mostrato nei metodi esposti nel capitolo 3.

6.2.1 Test per individuare la presenza di incertezza

Tutti i test realizzati, che verranno mostrati nel corso dei capitoli successivi, si basano su metodi presentati nel capitolo 3. Questi metodi vanno ad incrociare i valori contenuti nel log delle richieste con quelli contenuti nel log associato al carico registrato sul servente, in modo da poterne ricavare l'utilizzo nei vari istanti temporali. Per questo motivo è richiesto di caricare un secondo insieme di dati per l'esecuzione delle analisi. Risulta quindi necessaria una fase preliminare durante la quale l'utente dovrà fornire in ingresso al programma un log contenente il carico sul server registrato nel medesimo periodo temporale interessato dal log caricato in precedenza e contenente le richieste. Una volta fornito al programma tutti i dati necessari l'utente ha la possibilità di eseguire quattro differenti test, ognuno contenuto in una scheda dedicata. Queste tecniche possono essere utilizzate in un qualsiasi ordine desiderato ma è preferibile rispettare la sequenza con cui sono state progettate. Ogni metodo rappresenta un modulo totalmente indipendente dagli altri e questo garantisce la possibilità di espandere le capacità del software, nel caso fosse richiesto in futuro di eseguire differenti tipi di test. Risulta opportuno iniziare le analisi partendo dal primo test, concepito al fine di determinare la presenza o meno di incertezza sui dati in possesso dell'utente. Dopo aver ottenuto il responso da tale metodo, nel caso in cui esso sia positivo, è possibile procedere andando a verificare singolarmente le varie forme di incertezza, partendo dalla più facile da individuare e gestire fino alle più complesse, come descritto nel MUSE esposto nel capitolo 3.
I test attualmente resi disponibili dal *tool* sono i seguenti:

- Test per individuare se vi è incertezza sui dati
- Test per individuare se l'incertezza rilevata è riconducibile al tipo aleatorio sui parametri di input
- Test per individuare se l'incertezza rilevata è riconducibile al tipo epistemico sui parametri di input
- Test per individuare se l'incertezza rilevata è riconducibile al tipo strutturale

Il primo test che andrò quindi a descrivere in questo capitolo è quello che si pone l'obiettivo di fornire all'utente informazioni riguardo alla presenza di una generica forma di incertezza sui dati. Se la tecnica applicata dovesse fornire un esito negativo non avrebbe alcun senso andare ad utilizzare ulteriori metodi, vista l'assenza di una qualunque forma di incertezza che si potrebbe voler gestire. Per accedere a questo tipo di analisi è necessario selezionare la scheda contrassegnata con il nome "*Test Uncertainty*", posizionata nella parte destra della schermata principale. Mediante questa sezione del software l'utente ha la possibilità di impostare due parametri in modo da calibrare l'algoritmo.

I parametri selezionabili sono i seguenti:

- Tolleranza
- Frazione dell'insieme dei dati

I valori assegnati ai due parametri in ingresso influenzano direttamente i risultati prodotti in uscita dal test, rendendolo più o meno tollerante nelle analisi effettuate. In figura 6.6 è possibile vedere i risultati ottenuti per il caso di studio reale inerente al servizio webmail dell'università di Saragozza presentato nel capitolo 5. I valori mostrati sono riferiti al log registrato nel periodo compreso tra il 12 novembre 2013 ed il 31 dicembre 2013.

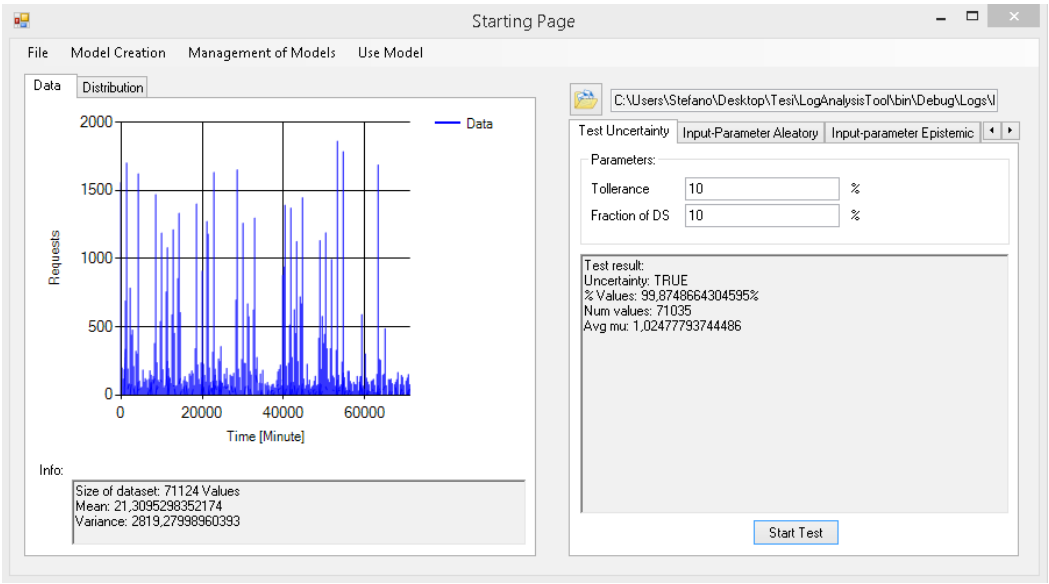

Figura 6.6 Output fornito dal test adibito a rilevare la presenza di incertezza per i dati provenienti da Unizar.

Come visibile in figura 6.6 nei dati di Unizar il test rileva la presenza di incertezza segnalando che per gran parte dei valori calcolati non vi è riscontro con i valori misurati presenti nel log del carico del servente. Visto il risultato ottenuto è quindi necessario andare ad indagare su quale possa essere la forma di incertezza insita nei dati al fine di poterla gestire successivamente. Per quanto riguarda invece i dati forniti da Wikipedia non è stato possibile utilizzare tale test vista l'assenza di un secondo log che indicasse il carico del server nei vari istanti temporali. Anche i test successivi risultano purtroppo inapplicabili in assenza di un secondo log e pertanto i dati appartenenti ai server di Wikipedia non verranno trattati nel corso di questa sezione.

6.2.2 Test incertezza aleatoria sui parametri di input

La seconda scheda, selezionabile nella sezione dei test posta nella parte destra della pagina principale, contiene il metodo per individuare la presenza di incertezza aleatoria sui parametri di input. Tale metodo è stato concepito modularmente ed è quindi indipendente da tutti gli altri test che si potrebbe desiderare svolgere. Questa logica è stata applicata nella realizzazione di ogni metodo, risulta però consigliabile utilizzare le tecniche proposte nell'ordine con cui sono state concepite essendo esse ordinate in base alla difficoltà di gestione dell'incertezza della classe che si cerca di individuare. Il metodo implementato nella scheda contrassegnata con il nome "*input-parameter aleatory uncertainty*", esposto precedentemente nel capitolo 3 , ha l'obiettivo di andare a verificare quanto siano sparsi i parametri μ caratterizzanti il tempo di servizio del server osservato. Per avere una misura della dispersione dei valori nel campione analizzato ho scelto di utilizzare come stimatore la varianza campionaria, calcolabile come [24]:

$$S_{n-1}^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1} \qquad (6.4)$$

Dove n è la dimensione del dataset, x_i è il campione i-esimo e $\bar{x}$ la media campionaria. Nel caso in cui l'incertezza presente nei dati sia determinata da un fenomeno aleatorio sui parametri di ingresso i valori del tempo di servizio non dovrebbero risultare eccessivamente sparsi e quindi la loro varianza contenuta. Se invece ci si trovasse in presenza di qualche altro tipo di fenomeno difficilmente si computerà un parametro μ quasi costante e si avrebbero dei valori largamente distribuiti e difficilmente riconducibili a questo tipo di incertezza.
Analizzando i dati caricati, appartenenti al webmail server dell'università di Saragozza, il software ha prodotto i risultati visibili in figura 6.7. Per questo caso di studio l'algoritmo ha prodotto valori μ eccessivamente sparsi. Essendo tali valori associati al tempo di risposta del server, rappresentando il parametro caratteristico della distribuzione esponenziale da cui è possibile estrarre il tempo di servizio, è possibile vedere come i risultati trovati siano tutt'altro che costanti. Inoltre i punti di minimo e di massimo, visibili nel grafico della distribuzione, risultano eccessivamente distanti perchè si possa parlare di fenomeno aleatorio. Risulta quindi necessario procedere con ulteriori test al fine di verificare quale tipologia di incertezza sia riconducibile a quella riscontrata ed andando ad analizzare tipi più complessi.

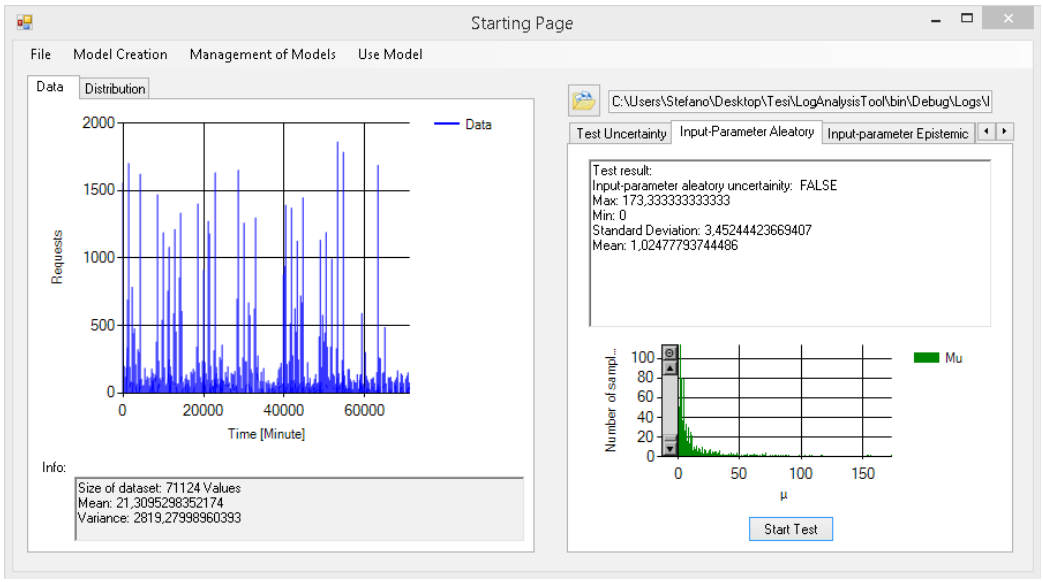

Figura 6.7. Output fornito dal test adibito a rilevare incertezza di tipo aleatorio sui parametri di input.

6.2.3 Test incertezza epistemica sui parametri di input

Nella terza scheda, posta nella parte sinistra della pagina principale, è inserito il test per l'individuazione dell'incertezza epistemica sui parametri di input. Il metodo utilizzato per realizzare tale test è già stato esposto ampiamente nella sezione 3.2. Il punto focale dell'algoritmo di test è quello di andare a trovare un qualche tipo di errore strumentale o di interpretazione dei dati che possa giustificare l'incertezza rilevata al fine di riuscirla ad inserire in questa classe.
Andando ad ipotizzare una errata registrazione dell'orario, dal lato del server o del servizio, l'algoritmo implementato va a provare differenti combinazioni dei valori contenuti nel log, spostando in avanti o indietro l'orario di uno dei due. Si cerca quindi di trovare una combinazione per la quale il *matching* delle coppie di dati contenute nei due log migliori visibilmente. Se tale combinazione dovesse esistere, difficilmente si potrebbe parlare di coincidenza visto il numero di valori solitamente trattati e si potrebbe pensare effettivamente ad un errore di tipo strumentale.

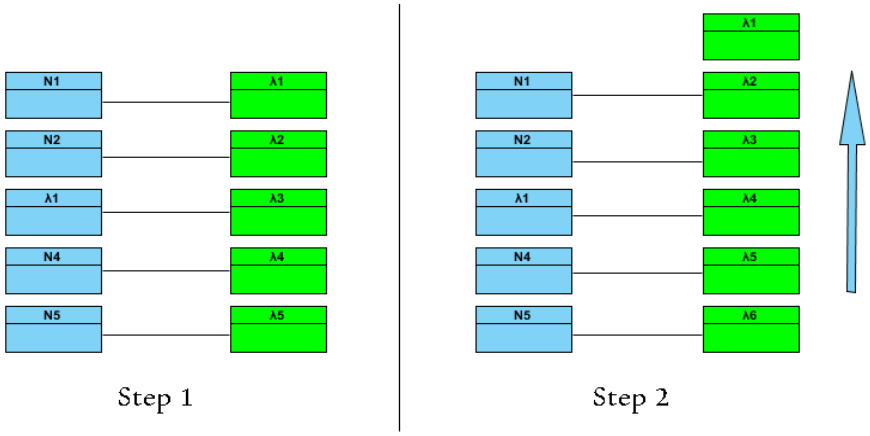

Figura 6.8. Rappresentazione del processo di ricerca delle diverse combinazioni tra le coppie appartenenti ai due log disponibili.

Come visibile in figura 6.9 il test implementato consente all'utente l'inserimento di un unico parametro di controllo, utile per la calibrazione dell'algoritmo. Questo parametro, identificato dall'etichetta *"tollerance"*, influenza appunto la tolleranza del metodo definendo la frazione di dati per la quale è accettabile avere un *matching* insoddisfacente tra i valori.

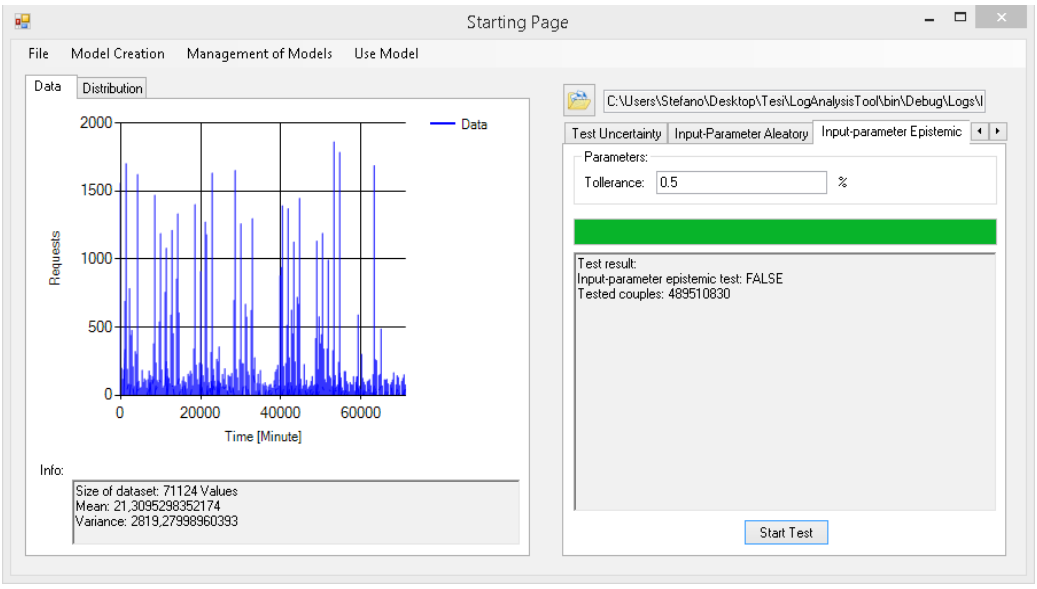

Figura 6.9. Risultati forniti dal test adibito alla ricerca di incertezza di tipo epistemico sui parametri di input.

Sempre in figura 6.9 è possibile vedere i risultati ottenuti con il test descritto sui dati provenienti dal servizio webmail dell'università di Saragozza. Il metodo implementato fornisce in uscita il valore booleano falso, andando ad evidenziare l'impossibilità di far ricadere l'incertezza rilevata in questa classe.

Secondo la tassonomia, esposta nella sezione 2.1, con l'esecuzione di questo secondo test è possibile escludere i parametri di input come causa dell'anomalia ed in questo caso specifico qualsiasi tipo di errore significativo nella registrazione dei dati. Occorre quindi procedere con ulteriori test volti alla ricerca di forme di incertezza più complesse, cosi come suggerito dal MUSE nel capitolo 3, proseguendo quindi con l'approccio *top-down* scelto.

6.2.4 Test incertezza strutturale

Il metodo che verrà esposto in questo capitolo è l'ultimo a disposizione per la diagnosi della tipologia di incertezza ed è stato precedentemente descritto nel capitolo 3. Il test implementato va a ricercare l'incertezza di tipo strutturale verificando se con piccole variazioni scelte opportunamente per il modello sia possibile ottenere un sensibile miglioramento della discrepanza rilevata tra i dati calcolati e quelli misurati. Le variazioni, come già spiegato nella sezione 3.3, verranno attuate sui valori registrati delle richieste al fine di consentire una riduzione della discrepanza misurata tra i valori appartenenti alle coppie $[N_{icalcolato}, N_{imisurato}]$. Per attuare in modo automatizzato queste modifiche il metodo si avvale di un algoritmo genetico chiamato *Hill Climbing* guidato da una opportuna euristica.
La funzione di test è accessibile nella parte sinistra della pagina principale del *tool* selezionando la la scheda contrassegnata dal nome "*structural uncertainty*". Nella schermata adibita all'analisi dei dati l'utente ha la possibilità di selezionare il numero massimo di modifiche che desidera consentire al modello. L'algoritmo, dopo aver effettuato tutte le modifiche necessarie, avvia una fase di valutazione del nuovo profilo di traffico ottenuto, i cui risultati sono visibili nella casella testuale posta nella parte inferiore della schermata di test.
Nel caso dei dati forniti dall'università di Saragozza i risultati ottenuti sono visibili in figura 6.10 dalla quale è possibile vedere come venga identificata una forma di incertezza di tipo strutturale sui dati a disposizione, nonostante il fenomeno rilevato non sia necessariamente appartenente a questa classe. L'incertezza strutturale risulta comunque la classe che meglio consente di spiegare il problema riscontrato tra quelle analizzate, ed è quindi consigliabile il suo utilizzo al fine di non doverlo andare a trattare con tipologie di incertezza più complesse. È quindi possibile interrompere la fase di test per i dati registrati da Unizar, avendo classificato in modo sufficientemente preciso il fenomeno identificato nella fase precedente del MUSE.

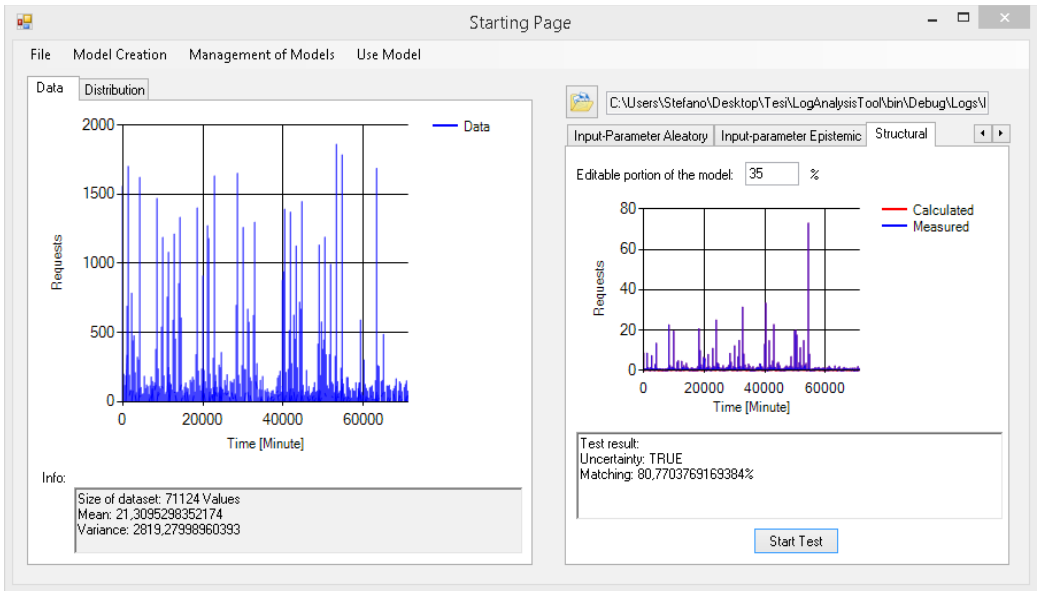

Figura 6.10. Output del test adibito alla ricerca di incertezza di tipo strutturale.

6.3 Metodi per la creazione dei modelli

Nelle prossime sezioni andrò a mostrare le differenti tecniche utilizzate dal software per la generazione dei modelli delle richieste utilizzati per la gestione dell'incertezza rilevata. Ognuno dei metodi implementati ha ottenuto risultati diversi in termini di prestazioni, intese come precisione media delle predizioni realizzate, rendendo le tecniche applicate più o meno utili, rispetto ai casi di studio osservati. Le schermate che compariranno nelle sezioni successive mostreranno il *tool* in funzione su diversi insiemi di dati, registrati sia dall'università di Saragozza che dai server di Wikipedia. È possibile trovare una descrizione più ampia di questi dati e di come essi sono stati acquisiti nel capitolo 5. Per realizzare previsioni mediante i modelli generati mi sono avvalso di relazioni appartenenti alla teoria delle code con le quali mi è stato possibile ricavare il tempo di risposta del server che desideravo utilizzare. Mediante il software è poi possibile confrontare tali valori con quelli ottenuti con metodi più affidabili, ma anche computazionalmente più onerosi, quali ad esempio le simulazioni, dando così una idea di quanto le previsioni generate con i modelli differiscano dai risultati reali.

6.3.1 Metodo di Split

Il primo metodo che ho deciso di prendere in considerazione è quello denominato con l'etichetta "S*plit"*. Per accedere a questa tecnica l'utente, dalla pagina principale del *tool*, va a selezionare mediante il menu a tendina, posto nella parte superiore della finestra attiva, la voce *"Model creation"*. Dopo la comparsa delle diverse opzioni disponibili è necessario poi selezionare la voce *"Clean data"* e successivamente il metodo di *split*, che verrà descritto in questa sezione. L'algoritmo utilizzato rientra nella categoria dei metodi di gestione dell'incertezza che si basano sull'idea di mantenere solo una parte dei dati, ritenuta rilevante, ed eliminare o elaborare separatamente i dati anomali rimanenti. Una volta caricata la pagina del *software* adibita ad utilizzare le funzioni del metodo, mediante l'interfaccia grafica, l'utente avrà la possibilità di impostare e mandare in esecuzione il processo di *split*, dopo aver definito alcuni parametri necessari alla calibrazione dell'algoritmo. Operando una selezione da interfaccia grafica è possibile evidenziare la parte del log che si desidera analizzare e creare un modello di essa rappresentato da una serie di indici statistici.

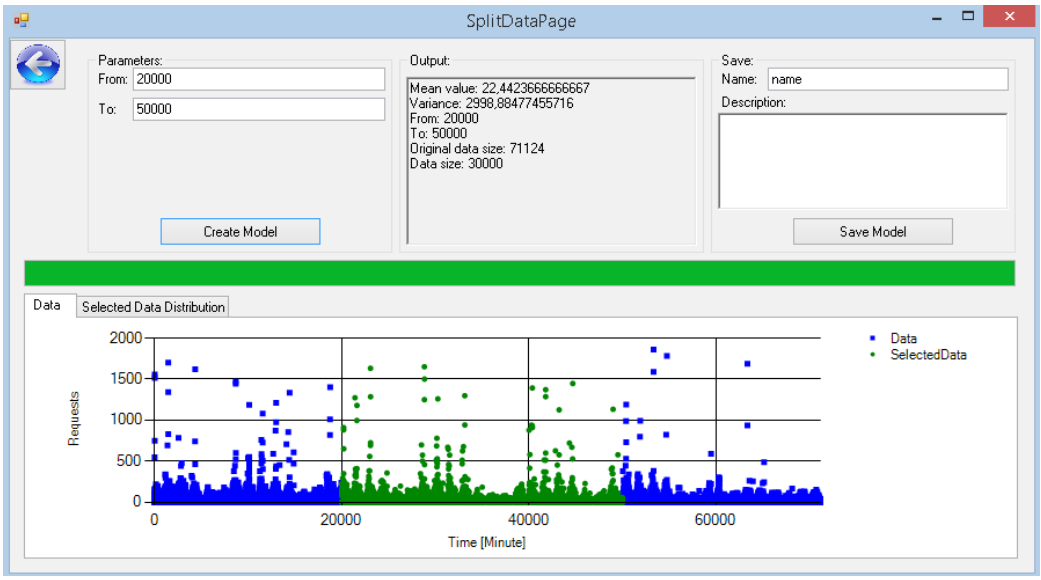

Figura 6.11 Metodo di *split* applicato ai dati provenienti dal caso di studio Unizar.

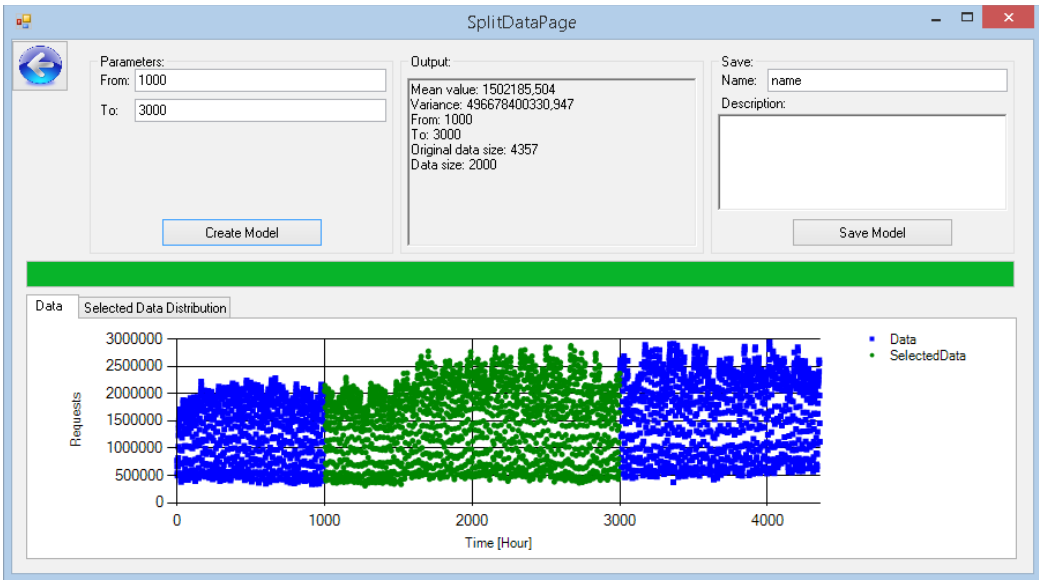

Figura 6.12 Metodo di *split* applicato ai dati provenienti dal caso di studio Wikipedia per l'area linguistica russa.

Nelle figure 6.11 e 6.12 è possibile vedere la tecnica di *split* in azione sugli insiemi di dati acquisiti rispettivamente dai server di Unizar e di Wikipedia.
Per creare un modello delle richieste, nella tabella etichettata con il nome "*parameters*", è necessario inserire, sotto forma di valore numerico, i due campioni limite da considerare. Il parametro denominato "*From*" rappresenta il campione di partenza della sezione che si desidera modellizzare del log, mentre il parametro denominato "*To*" rappresenta il campione finale di tale sezione. Se i valori inseriti risultano corretti il processo di *split* viene avviato dal programma e tutti i dati compresi tra il campione iniziale e quello finale vengono selezionati. Una volta completata questa operazione gli elementi di interesse dell'utente vengono rappresentati sul grafico, posto nella parte sottostante della finestra attiva ed evidenziati in colore verde, al fine di fornire una anteprima grafica della selezione effettuata. Se la porzione di log scelta pare adatta allo scopo, l'utente ha la possibilità di creare un nuovo modello e successivamente salvarlo. È inoltre consentito caratterizzare il file salvato con una breve descrizione testuale, da inserire nel campo predisposto nella parte sinistra della pagina. Premendo il tasto "*Save Model*" l'utente va ad avviare il processo di serializzazione generando cosi un file XML contenente il modello.

Il metodo di *split,* dalle prove effettuate e rispetto alle altre tecniche implementate, non fornisce risultati stabili. Le previsioni della misura del tempo di risposta generate, partendo da modelli delle richieste realizzati con tale metodo, risultano di qualità variabile e non forniscono grande affidabilità. Questa mancanza di stabilità può essere probabilmente estesa ad ogni caso di studio affrontabile con tale tecnica, essa infatti si basa sull'idea di utilizzare solamente la porzione di dati selezionata, come se essa fosse in grado di rappresentare tutte le informazioni in possesso dell'utente. Tale idea non sembra

poter portare ad effettuare previsioni utilizzabili in termini di tempo di risposta rendendo questo metodo non particolarmente utile a fini predittivi. La tecnica di *split* fornisce però un primo approccio, non particolarmente raffinato o utile, ma comunque un punto di partenza. Le predizioni ottenute mediamente con questo metodo possono essere viste come un limite sotto il quale la qualità media delle previsioni realizzate non dovrebbe mai scendere. Tutte le altre tecniche che andrò a presentare quindi, per dimostrarsi quantomeno utili, dovranno fornire modelli le cui previsioni abbiano una qualità media superiore a quella ottenibile con questo metodo.

6.3.2 DBSCAN

Il DBSCAN, come gli altri metodi di gestione dell'incertezza, è contenuto in una pagina dedicata del *software* realizzato. Per accedere alla schermata da cui è possibile gestire questo metodo di *clustering* è necessario, partendo dalla pagina principale del programma, selezionare dal menu a tendina posto nella parte superiore l'opzione "*Model creation*". Una volta fattò ciò dal sottomenu a comparsa bisogna scegliere il metodo DBSCAN posto all'interno della categoria "*Clean Data*". Nella sezione adibita alla gestione del metodo di *clustering* l'utente ha la possibilità di andare ad impostare due parametri al fine di calibrare nel modo ottimale l'algoritmo. Il parametro ε caratterizza la distanza entro cui il metodo di *clustering* deve andare a cercare i punti vicini a quello considerato, più esso è grande più viene permessa la creazione di *cluster* con densità minore. Il parametro etichettato con il nome "*minpts"* caratterizza invece il numero minimo di punti necessari per poter costituire un *cluster*, più esso è grande e più l'algoritmo richiede che vi sia una popolazione maggiore di punti all'interno di un *cluster* per poterlo considerare tale. Oltre a ciò la pagina fornisce tutte le funzioni per caratterizzare il modello già presentate precedentemente, quali la possibilità di assegnarvi un nome ed una breve descrizione.
La tecnica di DBSCAN da me implementata opera monodimensionalmente e valuta solo i valori dei campioni presenti nel *dataset* ignorando la dimensione temporale. Il numero di *cluster* impostati normalmente per la generazione è uno, tutti gli altri punti fuori da esso vengono automaticamente considerati *outlier* ed isolati. Per *outlier* si intende un valore eccessivamente sparso, rispetto alla distribuzione di tutti gli altri dati presenti nel dataset osservato, e quindi atipico.
I risultati ottenibili con tale algoritmo variano a seconda dei parametri scelti dall'utente, i quali vanno ad influenzare la tolleranza applicata dalla tecnica. Inserendo un parametro ε eccessivamente piccolo la ricerca dei vicini potrebbe non spingersi sufficientemente lontano da riuscire a creare *cluster* che invece, con altri valori del medesimo parametro, sarebbero altrimenti possibili.
Lo stesso discorso è applicabile ad un valore del parametro "*minpts"* troppo grande che potrebbe andare a richiedere dimensioni per i *cluster* eccessive rispetto alla densità dei dati contenuti nei log a disposizione.
In figura 6.13, dove vengono utilizzati i dati provenienti dal servizio webmail dell'università di Saragozza, è possibile vedere come parametri errati causino la

segnalazione di un numero eccessivo di *outlier*. Risulta inoltre possibile il caso contrario in cui il numero di *outlier* rilevato sia insufficiente a seguito di una calibrazione troppo permessiva dell'algoritmo.

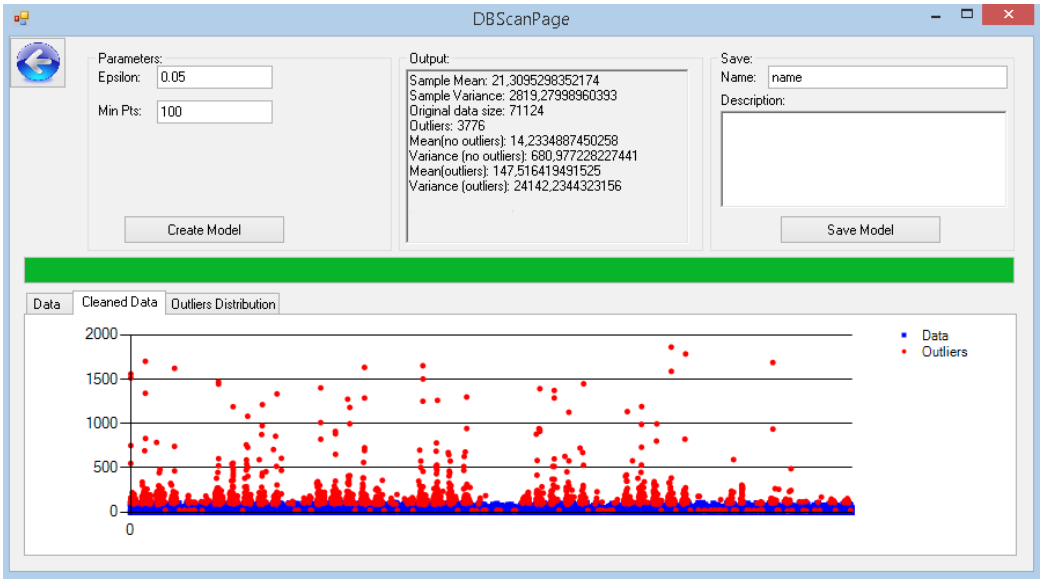

Figura 6.13. Schermata metodo DBSCAN con errata calibrazione dell'algoritmo di *clustering* per i dati provenienti dall'università di Saragozza.

Nelle figure 6.14 e 6.15 è possibile vedere il caso in cui i parametri siano invece impostati in modo corretto rendendo quindi utile il salvataggio dei modelli computati Grazie ad una calibrazione corretta dell'algoritmo risulta possibile evidenziare solo gli *outlier* più significativi riuscendo a separare i dati a più alta densità da essi in modo più efficace. Nel caso mostrato, utilizzando i dati di Unizar e Wikipedia, solo una piccola parte degli *outlier* viene evidenziata al fine di essere considerata separatamente, essendo difficilmente trattabile.

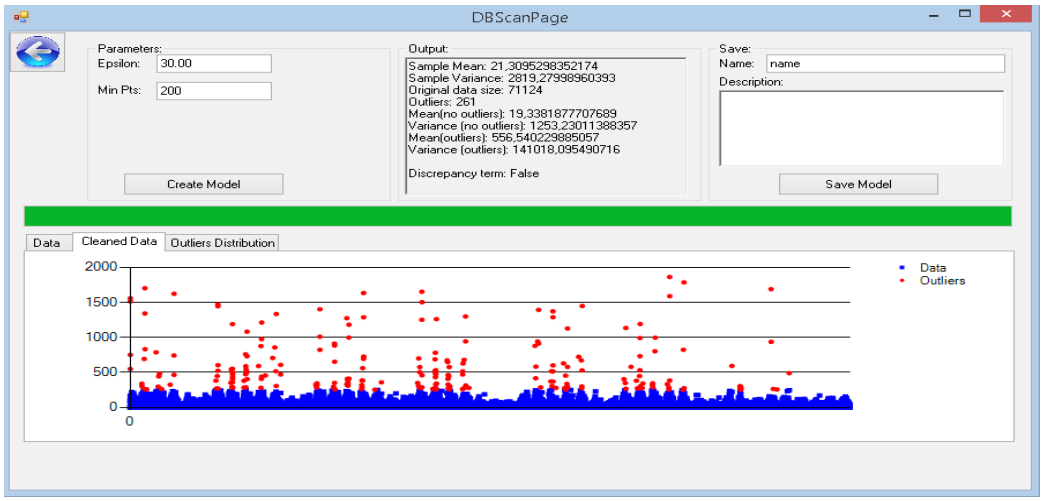

Figura 6.14. Schermata DBSCAN con corretta calibrazione dell'algoritmo di *clustering* per il caso di studio Unizar.

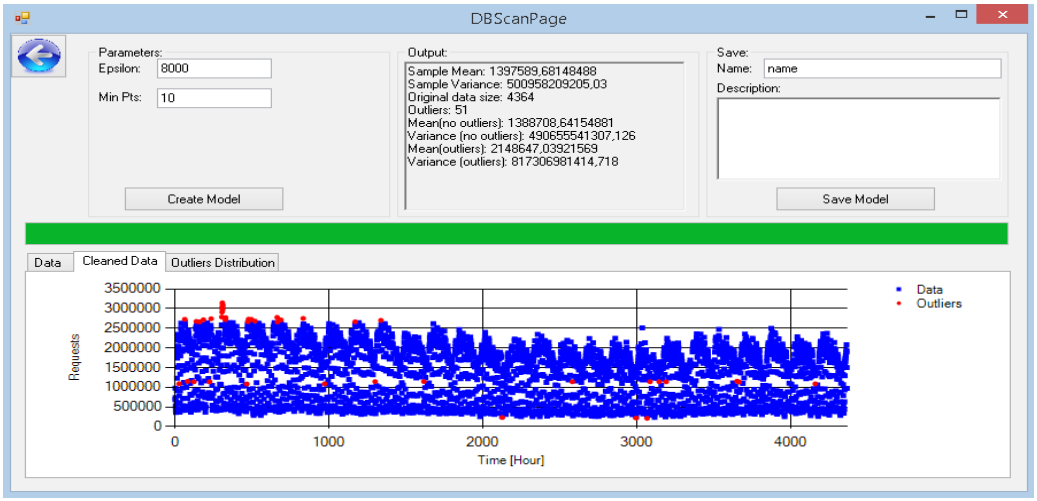

Figura 6.15 Schermata DBSCAN con corretta calibrazione dell'algoritmo di *clustering* per i dati acquisiti da Wikipedia ed inerenti all'area linguistica tedesca.

Dopo aver trovato le impostazioni del metodo che più soddisfano l'utente il software realizzato fornisce la possibilità di salvare il modello appena creato in formato XML. Tale modello, generato con il metodo di DBSCAN, a differenza di quello computato con il metodo di *split*, illustrato nella sezione 6.3.1, è però composto da un numero maggiore di componenti:

- Componente base: Vi sono inserite informazioni riguardo media e varianza campionaria dei dati selezionati oltre alla porzione di *dataset* che il componente rappresenta.
- Componente *outlier*: Vi sono inserite informazioni riguardo media e varianza campionaria degli *outlier* rilevati oltre alla porzione di *dataset* che il componente rappresenta.

Usando questa struttura le informazioni riguardanti gli *outlier* non vengono perdute ma vengono salvate separatamente per consentire, successivamente al calcolo del tempo di risposta, di applicare tecniche di aggregazione sui risultati al fine di migliorare la precisione delle predizioni.
Dopo aver applicato il DBSCAN su differenti insiemi di dati, con differenti calibrazioni, i modelli ottenuti hanno fornito previsioni più stabili di quelle realizzate mediante la tecnica di *split* presentata nel capitolo precedente. Il *clustering* basato sulla densità è infatti naturalmente predisposto all'individuazione di elementi anomali in un insieme di dati, svolgendo in modo molto preciso la ripartizione tra *outlier* e dati comuni.

6.3.3 Suddivisione dei dati mediante la computazione di limiti

Questo metodo per la suddivisizione dei dati mediante la computazione dei limiti è l'ultima tecnica per la creazione di modelli presente nella categoria che utilizza la rimozione degli *outlier*. Questi metodi prevedono il salvataggio separato delle informazioni legate ai punti anomali al fine di poter applicare tecniche di aggregazione dei risultati in una fase successiva.
Per accedere alla pagina del software dedicata a questo metodo, come per quelle esposte nei capitoli precedenti, è necessario selezionare dal menu principale la sezione "*Clean data*" e successivamente la tecnica in questione. Una volta aperta la schermata corretta l'utente avrà la possibilità, come in precedenza, di andare ad impostare alcuni parametri. Nel caso specifico sarà possibile selezionare il valore di un solo attributo per la calibrazione dell'algoritmo, sono inoltre disponibili tutte le opzioni il salvataggio del modello esposte per i metodi precedenti. L'unico parametro influenzabile dall'utente è denominato con l'etichetta "K"e definisce quanto la fascia di dati che si vuole estrarre dal log dovrà essere ampia. Andando ad utilizzare un valore per K basso si tenderà ad escludere un maggior numero di dati, mentre per un parametro K più elevato l'algoritmo selezionerà solo i picchi di richieste più alti considerandoli come *outlier*. Questo limite viene calcolato sommando al valore della mediana dei dati la varianza campionaria, moltiplicata per la costante K fornita dall'utente. In figura 6.16 è possibile vedere la schermata del software adibita alla gestione di questo metodo. L'utente, mediante una anteprima grafica, è in grado di comprendere quali campioni verranno considerati *outlier*, basandosi sul limite calcolato. Tale limite viene tracciato in colore rosso sul grafico posto nella parte inferiore della schermata. Sono presenti inoltre ulteriori informazioni poste in una finestra testuale contenente per ogni componente la media e la varianza campionaria, oltre alla porzione di *dataset* da essa occupata.
Le componenti descritte in tale finestra sono due, una contenente i dati e l'altra gli *outlier* selezionati. Oltre al grafico precedentemente citato vi sono due ulteriori anteprime grafiche poste nella medesima posizione e selezionabili cambiando la scheda in evidenza. Il primo di questi grafici contiene i dati filtrati e privi degli *outlier*, il secondo permette di visualizzare informazioni riguardo alla distribuzione degli elementi anomali. Grazie a queste informazioni aggiuntive l'utente è in grado di decidere in modo più chiaro se il modello da lui realizzato possa risultare utile per effettuare delle previsioni sul tempo di risposta o altre grandezze significative e se esso possa ritenersi rappresentativo dell'insieme di dati caricato.

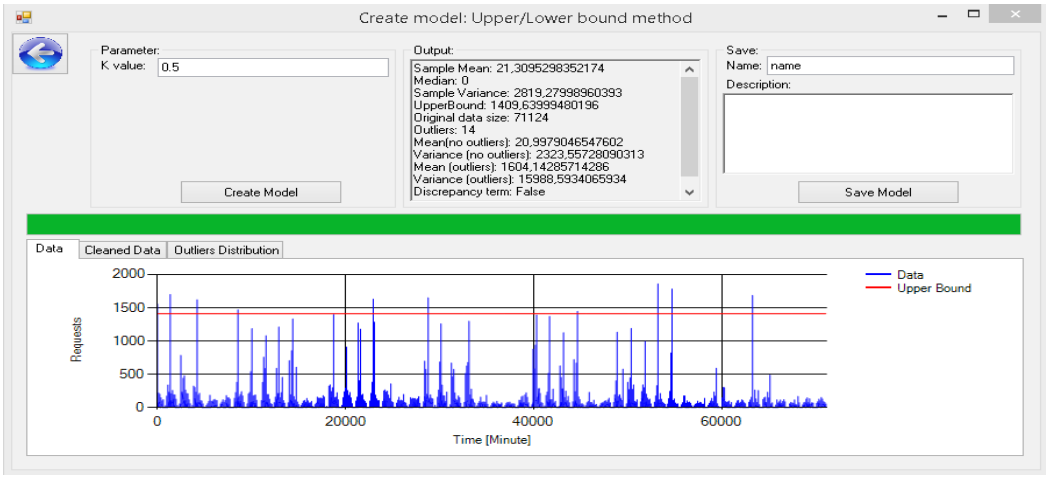

Figura 6.16. Schermata metodo di ripartizione dei dati mediante l'uso di limiti applicato ai dati provenienti dall'università di Saragozza.

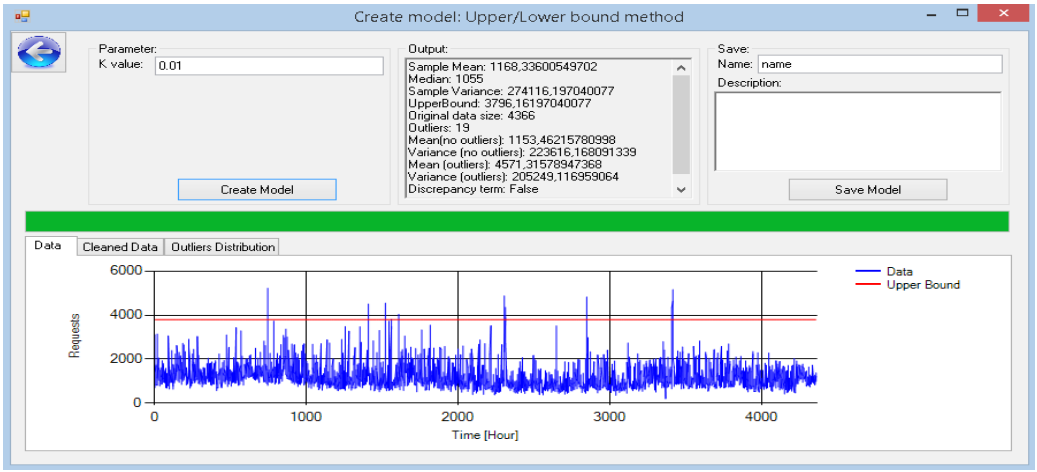

Figura 6.17 Schermata metodo di ripartizione dei dati mediante l'uso di limiti applicato ai dati provenienti dai server di Wikipedia per l'area linguistica lombarda.

Nelle figure 6.16 e 6.17 è possibile vedere il metodo applicato a due diversi insiemi di dati e con calibrazioni diverse dell'algoritmo al fine di riuscire a selezionare in modo opportuno i campioni anomali desiderati. Sia per i dati provenienti da Unizar che per quelli associati a diverse aree linguistiche di Wikipedia è stato possibile selezionare solo una piccola porzione di picchi di richieste riuscendo così a mantenere, come desiderato, una componente principale sufficientemente grande per le analisi.
Il metodo presentato si è dimostrato, in seguito alla creazione di diversi modelli, in grado di fornire risultati mediamente di buona qualità a patto che la costante K fosse stata impostata correttamente dall'utente. Le previsioni ottenute hanno inoltre subito un sensibile miglioramento mediante la fase di aggregazione, effettuata successivamente alla computazione del tempo di risposta, aumentando il potere predittivo dei modelli realizzati con questo metodo.

Posso quindi affermare che questa tecnica può collocarsi, a livello di potere predittivo e stabilità, sopra l'algoritmo di *split,* illustrato nella sezione 6.3.1, e i risultati ottenuti mediante essa sono paragonabili a quelli ottenuti con il DBSCAN, presentato nella sezione 6.3.2.

6.3.4 Ricerca di pattern

Vado ora a descrivere la sezione del software realizzata per utilizzare il metodo di ricerca dei *pattern*. Per accedere alla schermata da cui è possibile utilizzare questa tecnica, partendo dalla finestra principale del *tool* sviluppato, è necessario selezionare la voce del menu "*Model creation*" e successivamente l'opzione "Find patterns".Questo algoritmo, a differenza degli altri esposti in precedenza, non si basa sull'idea di andare a rimuovere i dati classificati come *outlier* ma sul cercare *pattern* di funzionamento differenti per il log caricato. La figura 6.18 va a mostrare la pagina del *tool* dedicata alla generazione di modelli con questa tecnica.

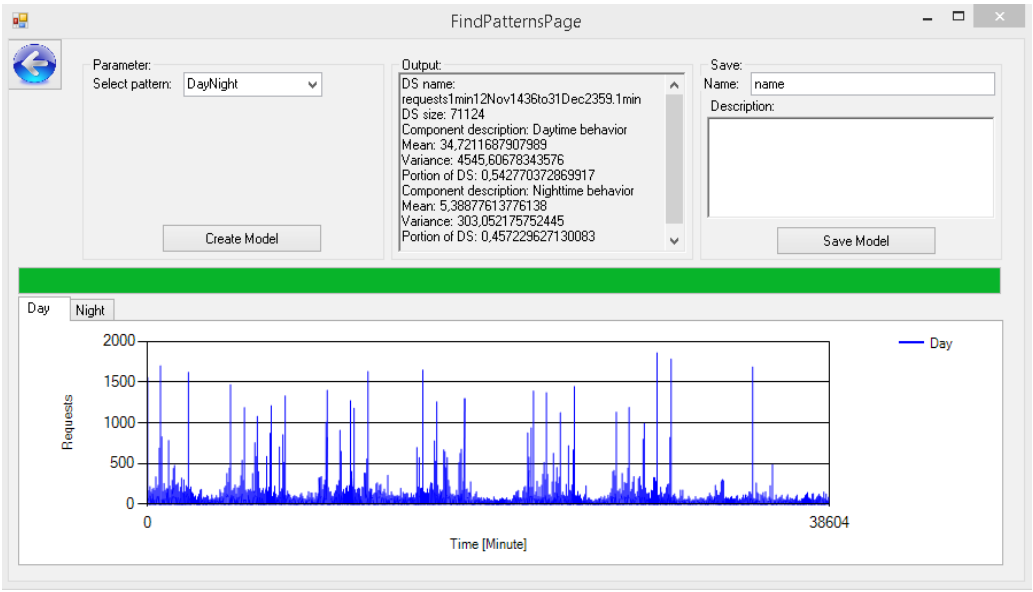

Figura 6.18. Metodo di ricerca dei *patern* utilizzando dati provenienti dall'università di Saragozza.

I *pattern* che l'utente può andare a ricercare sono stati preprogrammati e non è previsto l'inserimento di alcun parametro da parte sua. L'unica scelta attuabile dall'interfaccia grafica, mediante il menu a tendina posto nella parte in alto a sinistra della finestra, è riferita al criterio di ricerca che si vuole utilizzare.
Tutti i *pattern* che sono stati programmati si basano sulla suddivisione temporale dei dati ed ipotizzano quindi diversi valori di utilizzo per differenti periodi. Per operare tale divisione è stato necessario, in fase di lettura del log, inserire per ogni valore l'esatto istante di campionamento. Tale valore può essere calcolato

per ogni campione partendo dal momento in cui il campionamento è iniziato e conoscendo l'intervallo di tempo che intercorre tra la registrazione di un dato e l'altro Una volta completato l'arricchimento dei campioni con le informazioni temporali necessarie, il programma procede al caricamento della pagina per l'utilizzo del metodo vero e proprio. Al termine di questa operazione l'utente ha la possibilità di creare il modello selezionando il *pattern* da lui desiderato. Nella versione attuale del software il menu a tendina per la scelta del criterio di ricerca contiene le tre seguenti etichette:

- *"DayNight"*
- *"Week&Weekend"*
- *"Mix"*

Ognuno dei quali è associato ad uno dei *pattern* discussi nella sezione 4.3 e prevede di ricercare all'interno dei dati diverse modalità di funzionamento. Questi diversi profili di traffico possono essere ricercati tra il giorno e la notte, durante la settimana ed il fine settimana oppure all'interno di una combinazione delle due opzioni precedenti.
Una volta selezionato il criterio desiderato tutti i dati verranno scansionati e collocati nel loro insieme di riferimento ed il software, nella parte inferiore della finestra attiva, provvederà a rappresentare ogni *pattern* su un grafico differente.
Per i criteri di ricerca "*DayNight*" e "*WeekWeekend*" verranno generati due grafici, selezionabili dall'utente mediante le schede dedicate. Per il pattern "*Mix*" i componenti del modello saranno quattro e di conseguenza anche il numero di grafici e schede per contenerli varierà.

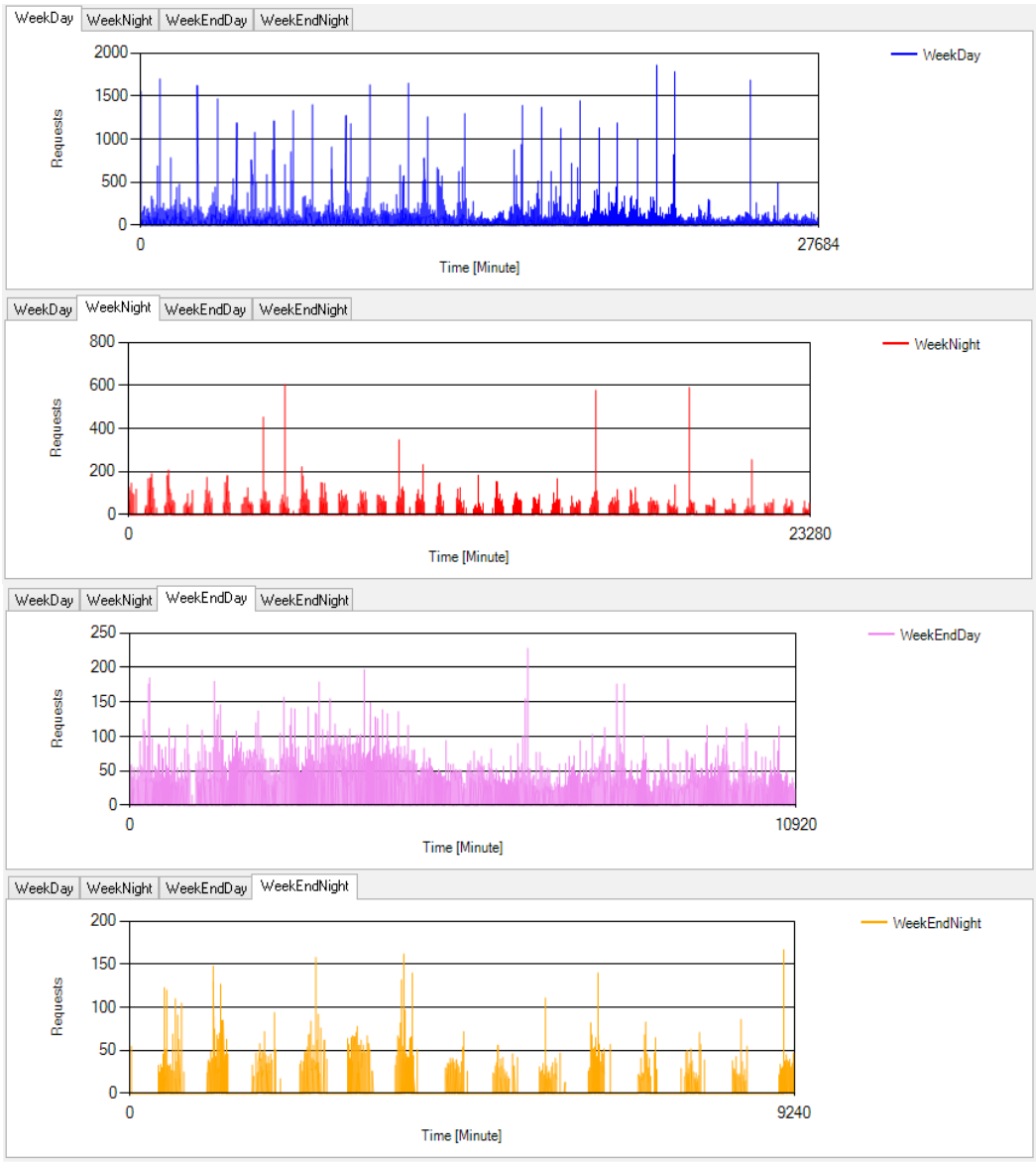

Figura 6.19. Grafici rappresentanti i pattern *WeekDay, WeekNight, WeekEndDay* e *WeekEndNight* messi a confronto per i dati provenienti dall'università di Saragozza.

Grazie all'anteprima, presente nella pagina di gestione del metodo, figura 6.18, ed alle informazioni contenute nella casella testuale, posta nella parte superiore della finestra, l'utente ha la possibilità di leggere e visualizzare una serie di informazioni importanti riguardanti il modello generato. Nella casella testuale sono disponibili diversi indici statistici, quali media e varianza campionaria delle aree di dati selezionate; inoltre è possibile farsi una idea qualitativa di come il carico delle richieste vada a variare nei diversi periodi temporali.
Sebbene questo metodo parta dall'idea che a periodi temporali differenti si possano associare profili di funzionamento di versi ciò non è sempre vero. In figura 6.20 è possibile vedere come, per i dati registrati dai server di Wikipedia e

provenienti dall'area linguistica inglese, non vi sia una sostanziale variazione del traffico, gli andamenti dei vari grafici risultano molto simili e il vantaggio nell'applicazione del metodo minimo. Tale risultato può trovare spiegazione nel grande bacino di utenza coperto da questa area linguistica, distribuito su diversi fusi orari, a cui risulta difficile associare picchi o flessioni nel volume delle richieste. In figura 6.19 invece è possibile vedere come, per i dati registrati da Unizar, vi siano invece considerevoli flessioni nel volume del traffico al variare del periodo temporale in cui si sta osservando il sistema.

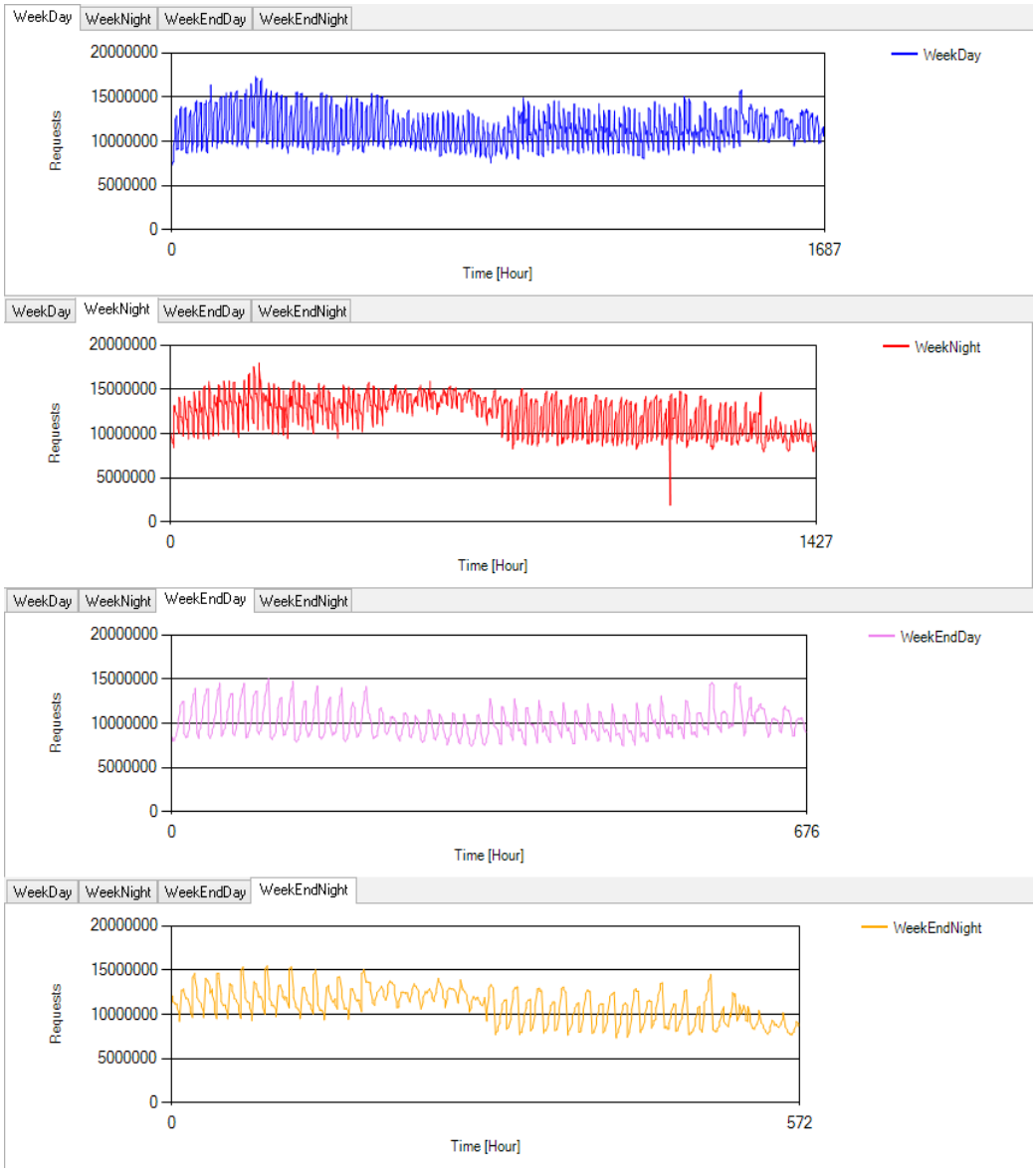

Figura 6.20 Grafici rappresentanti i pattern *WeekDay*, *WeekNight*, *WeekEndDay* e *WeekEndNight* messi a confronto per i dati provenienti da Wikipedia per l'area linguistica inglese.

Dopo la fase di computazione del modello, come per i metodi esposti precedentemente, vi è la possibilità di salvare i risultati ottenuti in formato XML, assegnando un nome al file ed inserendo eventualmente una breve descrizione di esso.
Questo metodo, dopo una serie di test effettuata sugli insiemi di dati a disposizione, è risultato stabile per quanto riguarda la qualità delle previsioni effettuate. La ricerca di *pattern* ha il vantaggio di non richiedere alcuna calibrazione da parte dell'utente, sebbene durante la parte di programmazione del software io abbia dovuto scegliere quali, secondo la mia personale opinione, potessero essere i criteri di ricerca più interessanti per i dati in esame. La parte di impostazione dell'algoritmo è quindi stata spostata dall'utente al programmatore, il quale ha il compito di andare a scegliere i criteri di ricerca. Vi è inoltre da tenere presente che i *pattern* scelti potrebbero non essere utili per tutti i tipi di dato analizzabili con il software realizzato, come per il caso mostrato in figura 6.20, fortunatamente però il metodo implementato si presta ad eventuali espansioni, fornendo cosi la possibilità di introdurre nuove tecniche ricerca.

6.3.5 Creazione di modelli del tempo di servizio

Per poter simulare una rete di code con un determinato tempo di servizio, in modo da essere in grado di calcolare il tempo di risposta in presenza di un flusso di richieste, è necessario disporre di determinate informazioni.
Il flusso di richieste nel caso di studio Unizar, descritto nel capitolo 5, è contenuto in un log del quale si conosce l'intervallo di tempo che intercorre tra la registrazione di un campione e l'altro. Risulta quindi possibile farsi una idea del numero di richieste che giungono mediamente in ogni istante temporale campionato. Per riuscire a determinare il tempo di servizio della rete di code simulata invece è necessario assegnare un valore al parametro μ o creare un modello che faccia variare il valore di tale parametro in modo coerente con i dati registrati. La tecnica che andrò ad esporre in questa sezione consente la generazione di un modello rappresentante il tempo di servizio ed applicabile alla simulazione. Per poter procedere alla creazione di questo modello l'utente, partendo dalla pagina principale del software, deve andare a selezionare, nel menu posto nella parte superiore, la voce "*Model creation*" e successivamente "*Create μ model*". Prima di effettuare questa selezione è però necessario caricare un log contenente il carico del server nei vari istanti di campionamento. Per quanto riguarda il caso di studio Wikipedia non è quindi possibile disporre di tale funzione, non avendo registrato alcun log del carico del server. Risulta quindi necessario, per questo caso, in fase di creazione della simulazione, determinare un parametro μ fisso ed appropriato al log delle richieste utilizzato.

Nel caso di studio Unizar invece il metodo risulta applicabile e durante il caricamento della pagina il software provvede alla generazione dell'insieme dei parametri μ, in cui ogni valore è ottenuto mediante l'applicazione della formula $\mu_i = \frac{\lambda_i}{N_i}$ dove λ_i rappresenta il valore delle richieste di servizio nell'istante di campionamento i-esimo ed N_i il carico registrato sul servente nel medesimo istante. I valori μ calcolati vengono poi rappresentati sul grafico principale visibile in figura 6.21.

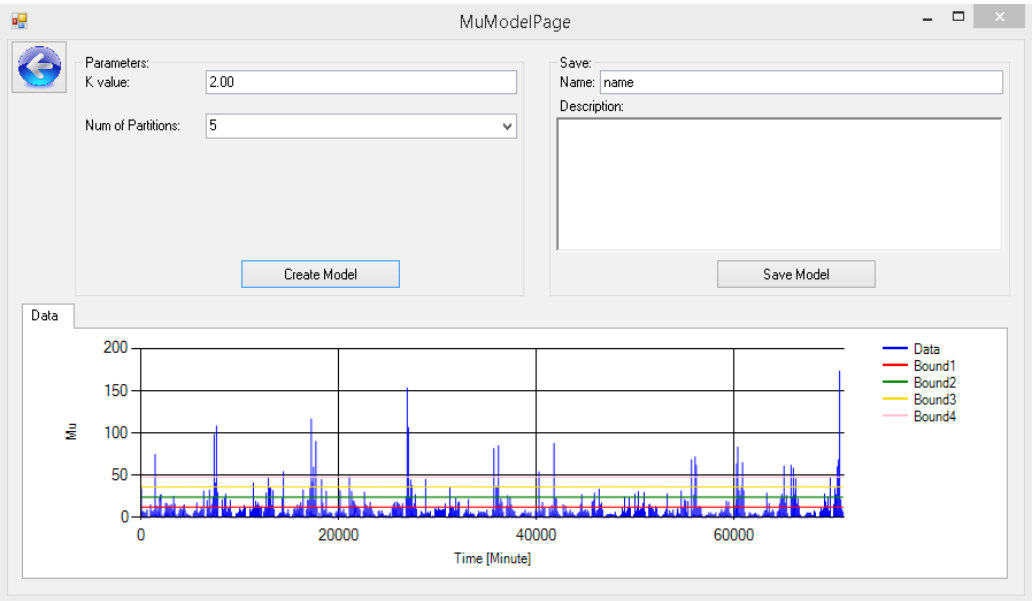

Figura 6.21. Schermata per la creazione di un modello del tempo di servizio.

E' possibile inoltre osservare l'andamento dei parametri μ che si desidera modellizzare in modo da poter farsi una idea di come la velocità nel soddisfare le richieste del server cambi. L'idea alla base di questo metodo è quella di partizionare orizzontalmente il grafico andando a dividere i valori dei parametri μ in fasce che verranno poi rappresentate con l'uso di indici statistici. Per eseguire questa operazione di divisione dei dati l'utente ha la possibilità di selezionare un valore da assegnare al parametro K. Tale valore, insieme a quello della varianza campionaria, va a determinare la dimensione di ogni singola partizione creata, ossia l'ampiezza delle diverse fasce di valori. Vi è inoltre la possibilità di selezionare il numero di partizioni che si desidera creare. Una volta computati i limiti, mediante la formula:

$$\text{Valore Limite} = \text{Mediana} + Z \cdot K \cdot \text{Varianza} \qquad (6.5)$$

dove K è la costante definita in precedenza dall'utente e Z un valore crescente definito dal programma, l'algoritmo procede con l'analisi di tutti i dati. Una volta terminato il processo, nel caso in cui l'utente sia soddisfatto del

partizionamento effettuato, vi è la possibilità di salvare il modello in formato XML al fine di poterlo utilizzare successivamente durante la fase di simulazione.
Per ogni componente contenuto nel modello creato viene memorizzato un valore $\overline{\mu}$ rappresentante la media campionaria dei parametri caratterizzanti il tempo di servizio del server contenuti nella partizione. Viene inoltre registrata la porzione di dati rappresentata dal valore salvato. Nel momento in cui si deciderà di utilizzare il modello in una simulazione il parametro μ_i, caratteristico del tempo di servizio per quell'istante della simulazione, sarà estratto con probabilità pari alla porzione di dati rappresentata da ogni componente del modello.
Ipotizzando quindi di avere un modello composto da tre componenti strutturati come segue:

- Componente 1: $\mu = 100$ [r/s], Porzione dei dati rappresentata = 20%
- Componente 2: $\mu = 200$ [r/s], Porzione dei dati rappresentata = 50%
- Componente 3: $\mu = 500$ [r/s], Porzione dei dati rappresentata = 30%

Durante la fase di simulazione il valore μ, se utilizzato per un campione sufficientemente ampio di richieste, assumerà il valore 100 per circa il 20% delle volte, 200 per il 50% e 500 per il restante 30%, determinando cosi un tempo di servizio medio molto vicino ai valori reali.

6.4 La simulazione

Al fine di poter utilizzare i modelli, realizzati con i metodi presentati nei capitoli precedenti, il software mette a disposizione una simulazione di una rete di code in grado di calcolare il tempo di risposta del sistema, descritto nel capitolo 3, che si desidera rappresentare. I valori ottenuti in uscita dalla simulazione verranno poi confrontati con quelli ottenuti mediante le predizioni in modo da poter valutare la qualità dei modelli realizzati e di conseguenza delle tecniche utilizzate. Avendo la possibilità di ottenere dei risultati più precisi effettuando una simulazione ci si potrebbe chiedere perchè non limitarsi ad essa, evitando cosi l'uso di altre tecniche, la risposta sta nel peso computazionale della stessa. Eseguire una simulazione di una rete di code infatti, per un log di richieste molto lungo e ricco di valori, può richiedere diverso tempo e memoria impegnate; applicare una semplice formula ad un modello invece è un processo molto più leggero e rapido. Per accedere alla pagina adibita alla simulazione l'utente, dalla schermata principale del software sviluppato, deve andare a selezionare dal menu, posto nella parte superiore della stessa, la voce *"use model"* e successivamente *"queueing network"*.

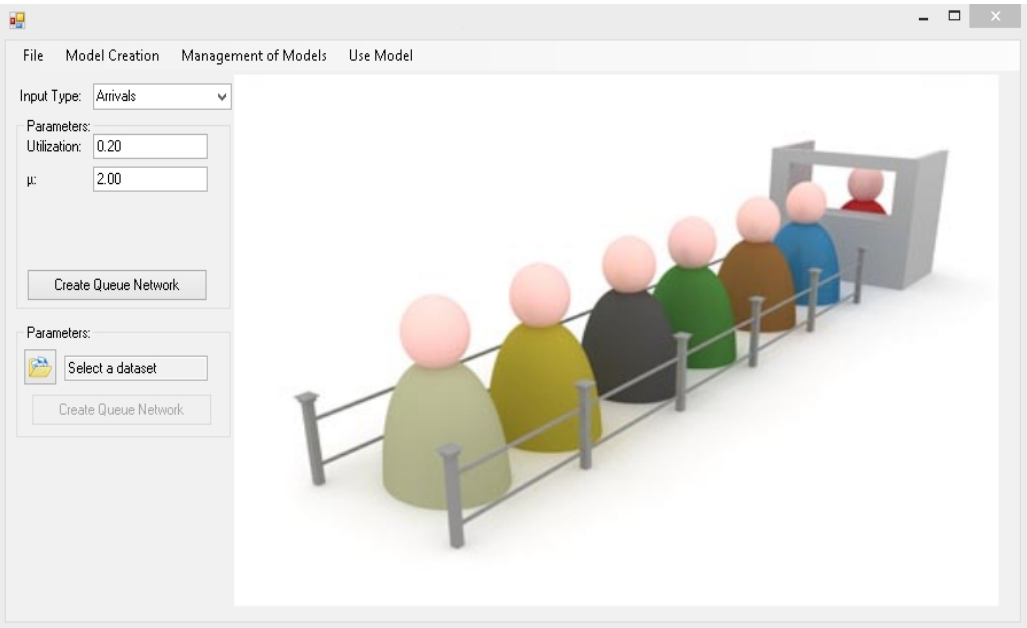

Figura 6.22. Schermata per l'impostazione della rete di code.

In figura 6.22 è visibile la pagina per la gestione della rete nella quale l'utente ha la possibilità di inserire diversi parametri, prima di avviare la simulazione, al fine di generare la rete che più si adatta alle sue esigenze. Nella casella di selezione etichettata *"Input type"* è richiesto di decidere il tipo di log caricato nel software, sono ammessi sia log delle richieste che log di carico sul server.
Vi è poi da precisare come la rete si comporta durante la fase di servizio. L'utente può caricare un modello del tempo di servizio, il cui processo di realizzazione è stato spiegato nella sezione 6.3.5, oppure impostare un valore μ fissato. Se si decide di utilizzare questa seconda opzione allora tale parametro andrà selezionato valutando i valori presenti nei dati caricati, al fine di non essere eccessivamente basso impedendo cosi delle analisi accurate. Sempre per questo caso, è anche possibile precisare la percentuale di utilizzo di base della rete. Definire un valore pari al 20% di utilizzo va ad indicare che sulla rete si sta eseguendo in contemporanea un altro processo, le cui richieste impegnano il sistema mediamente il 20%, lasciando quindi meno capacità di gestire ulteriore carico. Una volta terminata l'operazione di configurazione della macchina è chiesto all'utente di premere il tasto contrassegnato con l'etichetta *"Create queueing network"* per costruire ed avviare la rete. La simulazione esegue un semplice algoritmo che genera iterativamente le richieste, con la frequenza desiderata, e calcola l'istante temporale in cui esse vengono servite secondo le caratteristiche selezionate per il server. I valori del tempo di risposta associati ad ogni singola richiesta vengono poi salvati in un file di simulazione.

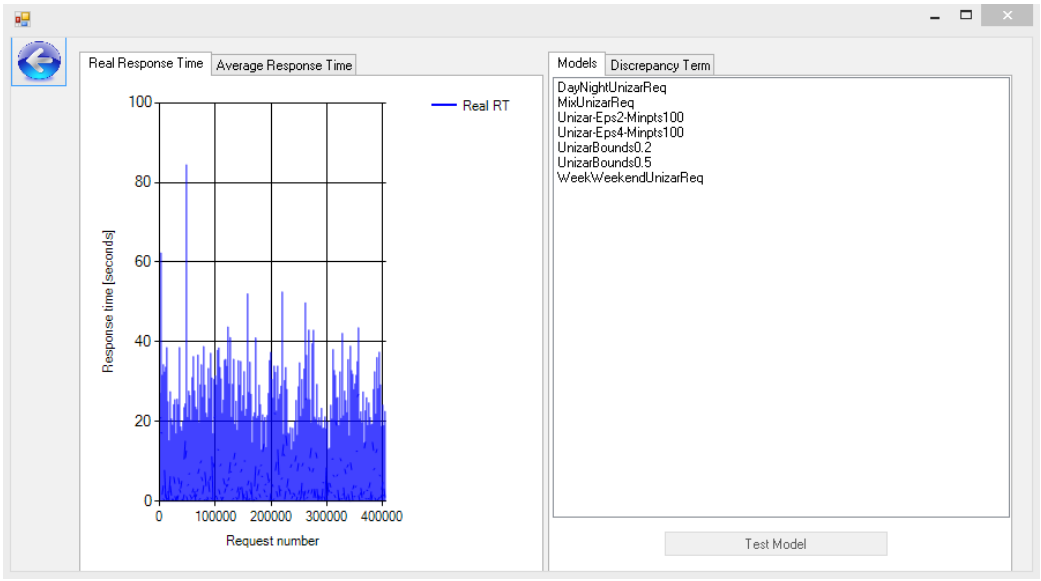

Figura 6.23. Schermata gestione rete di code con grafico del tempo di risposta reale per i dati provenienti dall'università di Saragozza.

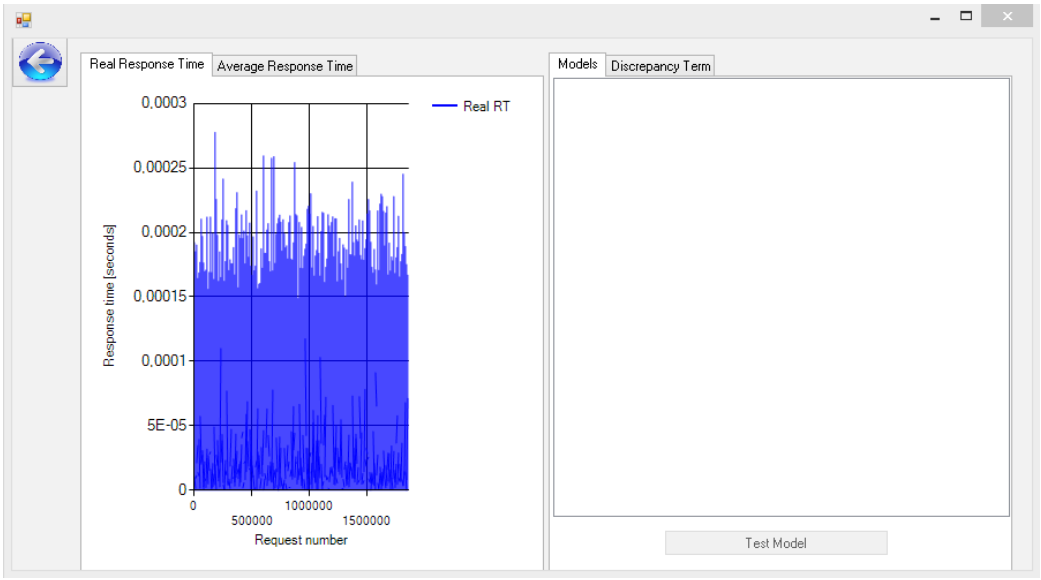

Figura 6.24 Schermata di gestione rete di code con grafico del tempo di risposta per i dati provenienti dai server di Wikipedia per l'area linguistica catalana.

Nel file di simulazione vengono inoltre inserite informazioni aggiuntive riguardo la configurazione della rete di code usata e del tipo di dato fornito in ingresso, rendendo possibile evitare di eseguire tutta la simulazione nei test successivi ma caricandone direttamente i risultati. Selezionando la scheda *"Average Response time"* sarà possibile accedere a diverse informazioni, quali la dimensione dell'insieme di dati utilizzato ed il tempo di risposta medio del server durante tutto il periodo in analisi. Tale valore viene utilizzato dai modelli come punto di riferimento, ed è ad esso che le loro predizioni devono avvicinarsi per poter essere considerate di buona qualità.

Dopo aver eseguito le prime simulazioni con questo metodo mi sono reso conto che il tempo di risposta medio prodotto con i dati provenienti da Unizar aveva dimensioni eccessivamente grandi, cosi grandi da rendere i risultati, forniti dalle previsioni dei modelli, totalmente inadeguate. Essendo la teoria delle code di facile applicazione ho inizialmente avuto dubbi riguardo al corretto funzionamento dell'algoritmo di simulazione ma, dopo aver eseguito una serie di test con log molto semplici di cui conoscevo il tempo di risposta atteso, mi sono reso conto che i risultati forniti erano corretti. Il problema che comportava un tempo di risposta eccessivamente elevato era la dimensione della coda. Stavo operando con una coda infinita. Nella realtà un server opera con una dimensione della coda finita e nel caso in cui sopraggiunga un eccessivo numero di richieste in un dato istante si limita a non accettarne più. Operare con una coda infinita invece consente di accettare qualsiasi numero di richieste ed in presenza di un picco di esse la dimensione della coda può crescere a tal punto da dilatare i tempi di servizio enormemente. Questa dilatazione non è però realistica ed è unicamente provocata dalle assunzioni sotto le quali ho scelto di svolgere la simulazione. Al fine di avere un tempo di risposta reale più affidabile possibile e non volendo imporre una dimensione della coda fissa, sia perchè la simulazione originariamente non è stata strutturata per poterlo fare, sia perchè non ho a disposizione tale caratteristica del server di Unizar ho scelto di operare solo con quei dati che non provocassero picchi di richieste tali da alterare enormente il tempo di risposta rispetto a quello reale. Per rimuovere questi dati ho scelto di utilizzare un percentile ritenuto affidabile, attorno al 90°, in modo tale da andare a simulare alcuni picchi di richieste ma non gli eventi eccessivamente anomali. Tale percentile è stato scelto empiricamente e potrebbe variare utilizzando differenti insiemi di dati, la variazione del tempo di risposta calcolato è stata però significativa esso è sceso di oltre 200 volte, per il caso di studio Unizar, rimuovendo solo una piccola porzione dei dati. Tale porzione inoltre andava ad influenzare la simulazione in modo negativo generando tempi di risposta improbabili che nella realtà non potrebbero mai verificarsi.
Per quanto riguarda i dati provenienti da Wikipedia, non avendo a disposizione informazioni riguardo al carico del server, questo accorgimento è risultato inutile se durante la fase di simulazione viene utilizato un server sufficientemente potente da non rendere i picchi di richieste ingestibili. Inoltre i dati registrati da questo caso di studio per la maggiorparte delle aree osservate non hanno evidenziato un grandissimo numero di anomalie e si sono rivelati, specie per aree linguistiche molto ampie, abbastanza costanti.

6.5 Uso dei modelli e loro raffinamento

Dopo aver ottenuto un valore di riferimento sufficientemente affidabile è il momento di andare a verificare i risultati generati dai modelli prodotti con i metodi per la gestione dell'incertezza. Rispetto all'esecuzione della simulazione questo processo, come già detto in precedenza, risulta molto più rapido, esso infatti si basa sull'utilizzo di alcune semplici formule derivanti dalla teoria delle code per il calcolo del tempo di risposta. Una volta caricato il modello desiderato esso viene esaminato dall'algoritmo ed ogni sua componente viene computata estraendo, separatamente dalle altre, il numero di richieste medie stimato e la porzione di dati rappresentata. Prendendo in esempio un ipotetico modello generato con il metodo della ricerca dei *pattern,* i dati contenuti avranno la seguente struttura per il *pattern Week / Weekend*:

- Componente 1: *Weekend*
  - Richieste medie giunte .
  - Porzione del dati rappresentata dalla componente.
- Componente 2: *Week*
  - Richieste medie giunte.
  - Porzione del dati rappresentata dalla componente.

L'algoritmo prima considera la componente 1 ed estrae le richieste medie giunte, dividendole per il periodo che intercorre tra la registrazione di un campione e l'altro, ottenendo un certo λ misurabile in richieste al secondo .
Nel caso in cui si sia deciso di eseguire la simulazione con un certo valore μ fissato opportunamente ed un valore di utilizzo base, U_{base}, è possibile andare a ricavare il tempo di risposta medio per la componente osservata del modello come segue.

L'utilizzo in una rete di code può essere calcolato come $U_{base} = \frac{\lambda_{base}}{\mu}$, partendo da tale relazione è possibile ricavare il numero medio di richieste al secondo che giungono dal processo base inserito sul server come $\lambda_{base} = U_{base} \cdot \mu$.
Tale quantità viene poi aggregata dall'algoritmo al numero di richieste per secondo contenute nella componente del modello andando ad ottenere $\lambda_{aggr} = \lambda_{base} + \lambda_{componente}$.
Partendo dal valore ottenuto è possibile determinare l'utilizzo complessivo della macchina da parte dei due processi come $U_{aggr} = \frac{\lambda_{aggr}}{\mu}$ dalla quale si può poi risalire al carico medio sul servente provocato dal contributo dei due processi mediante la relazione $N_{aggr} = \frac{U_{aggr}}{1 - U_{aggr}}$. Ottenuta tale quantità ricavare il tempo di risposta è immediato, il valore desiderato si ottiene come $R = \frac{N_{aggr}}{\lambda_{aggr}}$.

In alcuni casi però tale valore potrebbe non essere ricavabile, le relazioni provenienti dalla teoria delle code hanno un limite, esse risultano valide unicamente in presenza di una coda stabile e quindi di un utilizzo del server inferiore al 100%. Componenti di un modello rappresentanti un insieme di *outlier* potrebbero però ragionevolmente imporre un numero di richieste medio in arrivo maggiore di quelle che la macchina è in grado di soddisfare. Al fine di non perdere le informazioni contenute in questi componenti sono stato quindi costretto a ricorrere ad una approssimazione perchè essi potessero dare comunque il proprio contributo al calcolo del tempo di risposta.

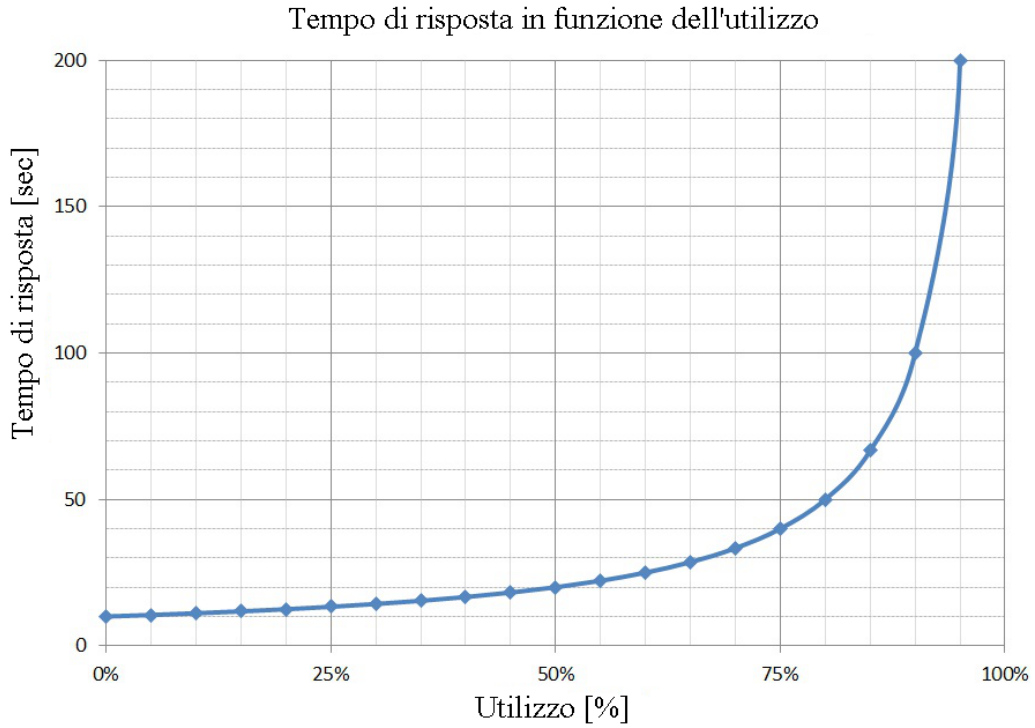

Figura 6.25. Andamento del tempo di risposta in funzione dell'utilizzo del server.

Ricordando che la funzione che caratterizza l'andamento del tempo di risposta in funzione dell'utilizzo del server non ha un andamento lineare (figura 6.25) ho scelto di utilizzare la curva dell'esponenziale per incrementare il valore calcolato. Il problema del non poter utilizzare la teoria delle code si verifica nel momento in cui il numero di richieste medie contenuto nel componente del modello è superiore al valore del parametro μ utilizzato. Ho scelto quindi di assegnare una quantità di richieste medie in arrivo tanto più vicino al 100% del valore μ quanto il valore contenuto nella componente è elevato.
Il valore inserito non va però mai a raggiungere il μ della macchina, mantenendo l'utilizzabilità della formula e si trova sempre nei punti dove la curva e più ripida e dove un piccolo cambiamento provoca una grande variazione del tempo di risposta calcolato. Utilizzando tale metodo è possibile estrarre un valore di tempo di risposta, anche molto alto, da componenti per le quali non sarebbe altrimenti stato possibile procedere al calcolo del loro contributo. Questo valore

però va a sottostimare quello reale che fornirebbe il server utilizzando una simulazione. Inoltre tale metodo richiede una calibrazione basata sui valori contenuti nei dati, al fine di capire a quanto ammontano i picchi massimi di richieste e riucire ad estrarre per essi valori più alti possibili, mantenendo così l'utilizzabilità delle formule.
Una volta terminato il processo e calcolati tutti i tempi di risposta viene avviata la fase di aggregazione che ha il compito di combinare i risultati ottenuti, migliorando la precisione delle predizioni. Questa fase di raffinamento dei risultati va ad effettuare un calcolo della media di tutti i valori trovati, pesandoli in base alla porzione di dati rappresentata dalla loro componente associata.
Operando in questo modo è possibile quidi tenere conto del contributo degli *outlier,* sebbene esso venga sottostimato. Anche dopo la fase di aggregazione sarà quindi necessario andare a raffinare le predizioni ottenute mediante i modelli, perchè troppo inferiori al valore del tempo di risposta ottenuto con l'uso della simulazione.

6.6 Il discrepancy term

Un ulteriore modo per migliorare i risultati ottenuti con i modelli generati è andare ad utilizzare metodi che fanno inferenza sulla discrepanza del modello, ossia la differenza tra il risultato atteso da esso e quello effettivamente ottenuto.
Un modello contenente incertezza può essere rappresentato da una funzione f con ingresso X tale per cui:

$$Y= f(X), \text{ con Y valore di uscita del modello.} \qquad (6.6)$$

Il valore di uscita Y sarà però difficilmente coincidente con quello atteso Z, ossia la previsione che si vorrebbe realizzare. Al fine di migliorare il potere predittivo di f e riuscire ad ottenere Z dal modello esaminato viene computato un termine additivo chiamato *discrepancy term* tale per cui:

$$Z = f(x) + \delta_z \quad (6.7)$$

Il *discrepancy term* δ_z è quindi un quantificatore dell'errore strutturale del modello, la differenza tra il valore di uscita effettivamente emesso e quello atteso. Questo tipo di approccio ha il vantaggio, rispetto alle tecniche di aggregazione, di prevedere la computazione di un solo modello e non di molteplici da combinare con pesi diversi. Vi è però anche una parte critica, riuscire a calcolare il termine additivo necessario perchè la precisione del modello riesca ad aumentare fino al livello desiderato [25]. Sebbene l'utilizzo del *discrepancy term* sia spesso posto come alternativa alle tecniche di aggregazione dei modelli, visti i risultati ottenuti, anche dopo aver combinato le diverse componenti calcolate, ho ritenuto opportuno dare la possibilità all'utente di migliorare le predizioni ottenute con l'uso di questa tecnica.
Per poter calibrare opportunamente un *discrepancy term* ho utilizzato come

valore di riferimento il tempo di risposta ottenuto mediante il processo di simulazione, essendo tale risultato con grande probabilità il più veritiero in mio possesso. Visto l'andamento della funzione mostrata in figura 6.25 e caratterizzata da una crescita sottolineare nel primo tratto ed esponenziale nel secondo, ho ipotizzato che le previsioni effettuate con i modelli, anche dopo la fase di aggregazione, siano di tipo ottimistico. Questa convinzione è anche dovuta al fatto che nel caso di componenti non processabili mediante l'utilizzo della teoria delle code ho scelto di inserire comunque il loro contributo ma approssimandolo in negativo. È quindi mia opinione che ogni previsione del tempo di risposta realizzata si assesti al di sotto del valore reale ottenuto mediante simulazione. Tale ipotesi ha trovato successivamente conferma negli innumerevoli modelli, generati con varie tecniche, e dai risultati da essi prodotti. Ognuno di questi ha fornito in uscita un valore di tempo di risposta, con diversi gradi di precisione nella predizione, sempre inferiore al valore ottenuto dalla simulazione. Una eccezione è rappresentata dal metodo di *split*, illustrato nella sezione 6.3.1, il quale per casi estremi e log particolari fornisce valori d'uscita maggiori del valore reale calcolato. Vista però l'inaffidabilità del metodo stesso non ho ritenuto significativo tale risultato.
Per migliorare i valori del tempo di risposta predetti è quindi necessario aggiungere ad essi un valore positivo che sia stato opportunamente calibrato. Questo valore deve compensare la perdita di precisione dovuta all'incertezza, ma allo stesso tempo garantire di non superare il tempo di risposta prodotto dalla simulazione. Ho scelto calcolare il *discrepancy term* in funzione degli istanti di campionamento in cui le richieste vanno a provocare un utilizzo del servente maggiore del 100%. Sono infatti questi picchi di funzionamento ad introdurre ulteriore difficoltà nelle previsioni alterando il valore reale del tempo di risposta anche in modo piuttosto elevato nel caso di un eccessivo numero di richieste persistente. Il *discrepancy term* che viene calcolato sarà quindi tanto più grande quanto più frequenti sono gli istanti di campionamento in cui le richieste eccedono la capacità della macchina. Nel programma da me realizzato l'utente ha la possibilità di introdurre un *discrepancy term* mediante la pagina contenente la simulazione della rete di code descritta nella sezione 6.4. Il processo può essere attuato dopo aver ottenuto i risultati della simulazione ed operato un confronto con il tempo di risposta fornito dai vari modelli. Nel caso in cui l'utente desideri migliorare la precisione sarà sufficiente, dopo aver selezionato la scheda *"Discrepancy term"*, evidenziare il modello designato e premere il tasto adibito all'aggiunta del termine. È quindi possibile attuare questo processo in modo selettivo sui modelli che lo richiedono.
In figura 6.26, per i dati acquisiti dall'università di Saragozza, viene mostrata la differenza di predizioni tra modelli che utilizzano il *discrepancy term,* nella parte superiore, e modelli che invece non ne fanno uso, nella parte inferiore.
Nella figura 6.27 invece si può vedere come il *discrepancy term* venga applicato ai risultati dei modelli realizzati per l'area linguistica Catalana di Wikipedia.
Ad ogni colonna mostrata è associato il tempo di risposta del server, predetto mediante uno specifico modello. Fa eccezione la colonna in colore blu, il cui risultato è stato ottenuto tramite un processo simulativo. Come è possibile

vedere in figura 6.27 i modelli realizzati con il metodo di *Split* (in colore nero e arancio) forniscono dei risultati instabili, mettendo cosi in dubbio l'affidabilità della tecnica. Le predizioni realizzate utilizzando la ricerca dei *pattern* hanno invece evidenziato risultati migliori. Nello specifico il criterio chiamato *Mix,* che va a suddividere i dati in quattro dgruppi, visibile in colore grigio in figura 6.27 e in colore azzurro in figura 6.26, ha registrato il risultato migliore.
Dopo l'applicazione del *discrepancy term* tutti i tempi di risposta predetti si sono avvicinati significativamente al tempo di risposta ottenuto mediante simulazione. L'affidabilità dei differenti modelli e delle predizioni realizzate mediante essi invece non è cambiata.

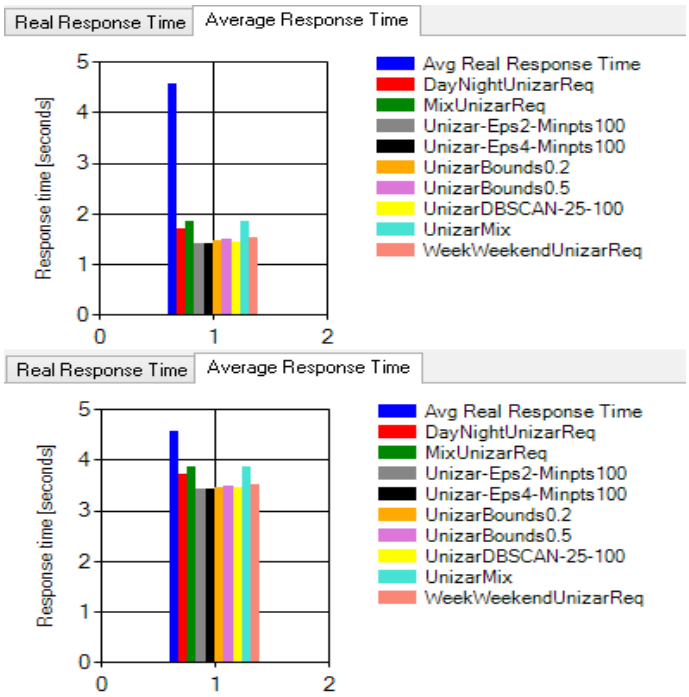

Figura 6.26. Confronto tra modelli privi di *discrepancy term* (sopra) e modelli che lo utilizzano (sotto) per i dati provenienti dall'università di Saragozza.

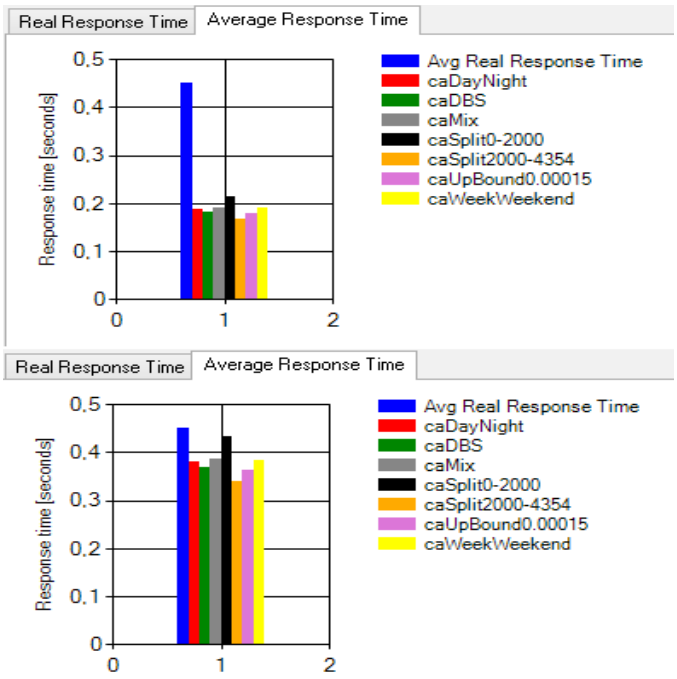

Figura 6.27 Confronto tra modelli privi di *discrepancy term* (sopra) e modelli che lo utilizzano (sotto) per i dati provenienti dall'area linguistica catalana.

I risultati ottenuti possono variare in funzione dell'insieme di dati e della tecnica utilizzata per realizzare il modello ma consentono frequentemente di avvicinarsi alle previsioni desiderate. Nel caso in cui l'insieme di dati dovesse variare o si decidesse di estendere il software con ulteriori tecniche di modellizzazione potrebbe essere necessario ripensare alla formula usata per il calcolo del termine additivo. Utilizzando infatti nuove tecniche, dotate di una maggiore precisione, bisognerà probabilmente ricalibrare il *discrepancy term* in modo tale da non superare il valore del tempo di risposta atteso con le predizioni dei nuovi modelli. Anche cambiando la formula però le basi su cui essa si fondano rimarranno molto probabilmente invariate, il *discrepancy term* tenderà a basarsi sul numero di picchi registrati nel log e sulla loro dimensione.

Capitolo 7

Uso di una rete neurale come modello

In questo capitolo andrò a descrivere la rete neurale, realizzata in ambiente .NET, con lo scopo di fornire predizioni di valori significativi, quale ad esempio il tempo di risposta di un server che si desidera simulare. L'approccio utilizzato in questo caso può essere considerato differente rispetto a quello mostrato in precedenza, che si avvaleva in prevalenza di metodi statistici. Tutte e due le tecniche hanno però l'obiettivo di riuscire a gestire il fenomeno dell'incertezza riscontrata durante la prima fase del MUSE, descritta nel capitolo 3.
L'implementazione della rete neurale è stata realizzata separatamente dal *tool* principale di analisi dei dati ed è composta da una libreria che mette a disposizione tutte le funzioni e le strutture dati necessarie al suo funzionamento. L'utente ha la possibilità di interagire con le funzionalità fornite mediante una interfaccia grafica, sviluppata in ambiente .NET sfruttando i Windows Forms e la programmazione ad eventi.. Nella sezione 6.6, parlando del *discrepancy term,* ho introdotto un ipotetico modello rappresentabile con una funzione f che riceve in ingresso un parametro X. Addestrando opportunamente una rete neurale, mediante una serie di esempi significativi, è possibile portarla ad apprendere una qualsiasi funzione f tale per cui ad un determinato valore di ingresso X essa sarà in grado di associare un valore di uscita vicino a quello atteso, Z. Si tratta quindi di un processo di apprendimento supervisionato strettamente dipendente dalla qualità degli esempi forniti. L'idea è quindi di andare a sfruttare le reti neurali per portarle ad apprendere quella funzione che lega i dati in mio possesso, carico sul server e numero di richieste inviate, con la predizione del tempo di risposta del server che si desidera simulare.

7.1 Struttura scelta per la rete, insiemi di addestramento e test

In questa sezione descriverò l'interfaccia utente, realizzata per utilizzare la libreria che implementa la rete neurale, grazie alla quale è possibile definire una topologia per la struttura che si desidera addestrare. Una volta avviato il programma l'utente ha accesso ad un menu, posto nella parte superiore della finestra principale, con il quale può caricare una rete salvata precedentemente o avviare la creazione di una nuova. In figura 7.1 viene mostrata la finestra del *tool* che guida l'utente nella creazione di una nuova rete. Per prima cosa viene richiesto di decidere il numero di strati che la struttura dovrà utilizzare, tale numero è comprensivo dello strato di ingresso e di quello di uscita. Una volta selezionata la topologia desiderata sarà necessario andare a descrivere in modo più dettagliato ogni strato inserito.

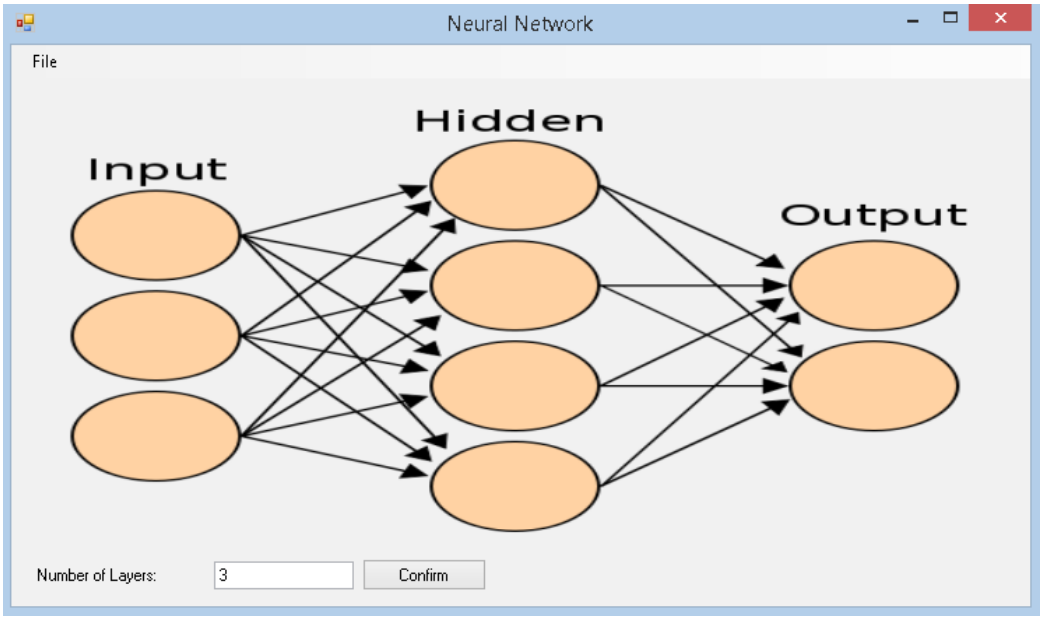

Figura 7.1. Schermata per la scelta del numero di strati della rete.

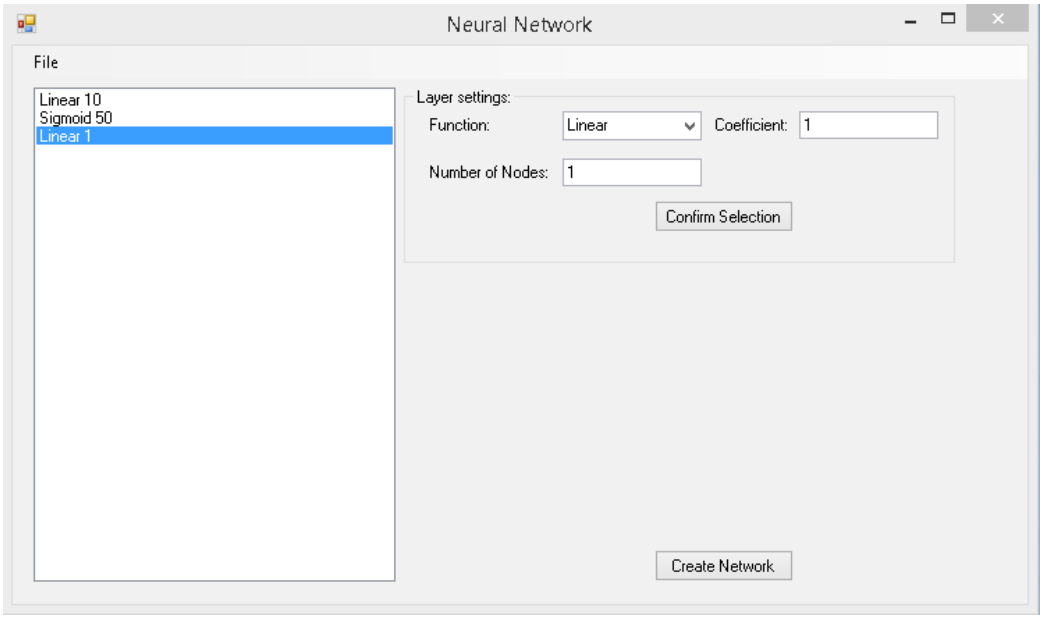

Figura 7.2. Schermata per la definizione delle caratteristiche di ogni strato.

Nella pagina del programma rappresentata dalla figura 7.2 è possibile vedere come l'interfaccia utente consenta di inserire svariate informazioni per ogni singolo strato. All'utente viene richiesto di selezionare la funzione matematica che i nodi dello strato selezionato andranno ad utilizzare per trattare i valori forniti in ingresso. Viene inoltre richiesto il numero di nodi di cui lo strato sarà composto.

Le funzioni matematiche definite ed utilizzabili in una rete neurale sono le seguenti:

- Lineare
- Tangente iperbolica
- Sigmoide

La funzione lineare viene generalmente utilizzata negli strati di ingresso e di uscita in modo da dimensionare i valori in ingresso secondo le esigenze dell'utente. La tangente iperbolica e la sigmoide vengono invece utilizzate negli strati più interni della rete al fine di rimuovere la relazione di linearità tra ingresso ed uscita. Questi tipi di funzione consentono inoltre di limitare la dimensione dei valori che scorrono all'interno di una rete. Terminate tutte le operazioni preliminari, volte a definire la topologia della nuova rete, il software consente di effettuare l'inizializzazione, rendendo disponibile il tasto *"Create Network"* posto nella parte inferiore della finestra attiva.

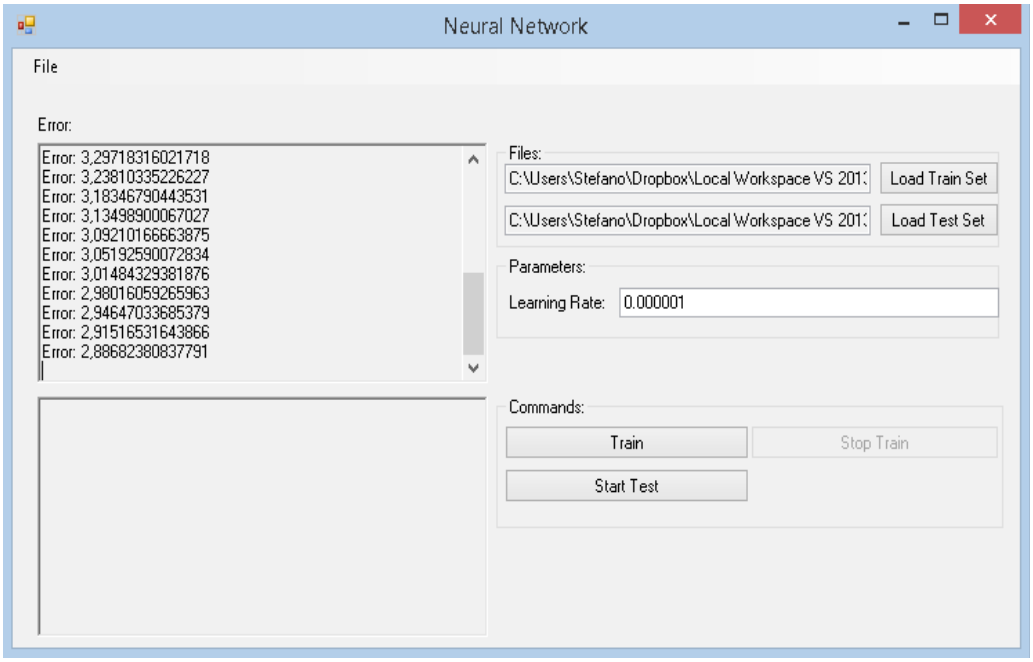

Figura 7.3. Processo di addestramento di una rete neurale.

Nella finestra che va ad aprirsi, visibile in figura 7.3, l'utente ha la possibilità di caricare un insieme di dati da utilizzare per l'addestramento, contenente una serie di esempi rappresentativi della funzione che si desidera far apprendere alla struttura. Inoltre è possibile definire un coefficiente di apprendimento, il quale va a determinare la rapidità con cui i pesi posti sugli archi vengono aggiornati durante la fase di addestramento. Il processo di minimizzazione dell'errore può essere avviato in qualsiasi momento premendo il tasto contrassegnato con l'etichetta *"Train"*, a patto di aver preventivamente caricato un insieme di esempi valido. Nella finestra testuale, posta nella parte sinistra della schermata

attiva, è possibile visualizzare il decadimento dell'errore medio durante l'esecuzione dei cicli d'addestramento. L'errore commesso sull'insieme di esempi scende tanto più velocemente quanto è più alto il coefficiente di apprendimento assegnato. Impostare valori troppo elevati può però causare problemi, la rete potrebbe effettuare variazioni eccessivamente brusche nell'aggiornamento dei pesi, diventando in alcuni casi instabile. È quindi fondamentale calibrare opportunamente il valore del coefficiente di apprendimento ed eventualmente modificarlo solo dopo un certo numero di iterazioni, nel caso in cui la variazione dell'errore non fosse soddisfacente.
Il processo di addestramento gestito dal *tool* può essere interrotto in qualsiasi momento, sia per salvare la struttura corrente che per procedere con la validazione tramite test. Per eseguire la fase di test, volta a comprendere se la rete è effettivamente riuscita ad apprendere la funzione desiderata, è necessario aver caricato un insieme di esempi da utilizzare come validazione. Questo insieme viene chiamato insieme di test e deve essere realizzato in modo tale che gli elementi al suo interno siano rappresentativi della porzione di realtà che si vuole rappresentare ed allo stesso tempo ignoti alla rete neurale.
Per poter realizzare un insieme di esempi utili a predire il tempo di risposta di un determinato server, partendo da un log contenente le richieste e conoscendo le caratteristiche di servizio della stessa, ho cercato un compromesso tra la precisione desiderata e la dimensione della rete da costruire. Fornire in ingresso alla rete per intero il log o grandi parti di esso mi avrebbe costretto a costruire una rete con un numero eccessivo di nodi nello strato di ingresso, inoltre il numero di esempi a disposizione sarebbe stato eccessivamente limitato.
Ho scelto quindi di strutturare gli insiemi di addestramento e di test con esempi numerosi ed il più compatti possibile. Ogni esempio è costituito da 7 elementi, il primo rappresentate il valore del parametro μ, caratterizzante il tempo di servizio di un server, i cinque successivi estratti sequenzialmente dal log delle richieste e l'ultimo, calcolato mediante la simulazione, equivalente al tempo di risposta del server che si desidera simulare per il tratto di log analizzato.
Al fine di avere il maggior numero possibile di esempi, ed allo stesso tempo porre l'enfasi su come variando un valore delle richieste possa cambiare anche drasticamente il tempo di risposta, i cinque valori di un esempio estratti dal log sono stati prelevati in modo tale da sovrapporsi parzialmente ai cinque valori prelevati per l'esempio precedente. Per chiarire maggiormente questo processo di creazione vado a riportare un esempio:

Siano a,b,c,d,e,f,g,h i valori contenuti in un log, μ il parametro di riferimento della macchina e z_i il tempo di risposta previsto per la porzione di log i-esima analizzata. Dati tre esempi inseriti nell'insieme di addestramento in sequenza il loro contenuto sarà:

- Esempio 1: μ, a, b, c, d, e, z_1
- Esempio 2: μ, b, c, d, e, f, z_2
- Esempio 3: μ, c, d, e, f, g, z_3

Gli esempi generati con tale criterio vengono salvati in appositi file di estensione .smpl e sono caricabili durante la fase di addestramento, a patto che i valori in essi contenuti siano compatibili con la topologia definita per la rete. Per compatibilità con la topologia si va ad intendere il prevedere lo stesso numero di valori di ingresso e di uscita per la rete che si desidera utilizzare.
Per utilizzare le reti neurali, realizzate con la struttura descritta precedentemente, ho usato i dati provenienti dal caso di studio Wikipedia, descritto nel capitolo 5. Le previsioni del tempo di risposta da inserire negli esempi sono invece state ottenute di volta in volta mediante l'uso di un metodo simulativo, in modo da ottenere valori il più possibile affidabili.
Nonostante diversi tentativi di addestrare differenti reti, ognuna con una propria struttura caratteristica, i risultati ottenuti non si sono però rivelati soddisfacenti. Utilizzando reti neurali con un solo strato nascosto e con un basso numero di nodi la durata di un passo di apprendimento è risultata molto breve. Per passo di apprendimento si intende il tempo che impiega l'algoritmo a computare il valore di uscita della rete, effettuare una stima dell'errore commesso e propagare questo errore a ritroso nella struttura al fine di aggiornare iterativamente i pesi, mediante la tecnica di *backpropagation* descritta nel capitolo 4.
Per reti di questo tipo il decadimento dell'errore è risultato molto rapido, nelle prime iterazioni dell'algoritmo, per poi fissare i valori emessi in uscita, senza progredire significativamente nella correzione dell'errore. Le quantità predette si sono quindi rivelate eccessivamente lontane dai valori desiderati, portandomi cosi a scartare questo tipo di reti. Nel caso di strutture più complesse, con due strati interni ed un maggior numero di nodi, la durata di un passo di addestramento si è dilatata molto rendendo il processo significativamente più lento. L'errore, in questo caso, ha continuato a decrescere ad ogni iterazione, ma non sono stato comunque in grado di ottenere risultati utilizzabili. Nel caso di reti troppo piccole, con un breve periodo di addestramento si giunge ad un punto in cui non si è più in grado di migliorare la previsione e i risultati forniti sono comunque inaccettabili. Nel caso di reti complesse la durata dell'addestramento diventa troppo elevata e la riduzione dell'errore troppo lenta e sempre meno significativa perchè si possa arrivare a previsioni utilizzabili in un tempo ragionevole. Il *tool* implementato si è però dimostrato in grado di apprendere funzioni più semplici come addizioni e moltiplicazioni con numeri naturali, mostrando che effettivamente il metodo utilizzato per l'implementazione della rete funziona. Nel caso affrontato il problema riscontrato nell'apprendere la funzione desiderata potrebbe identificarsi nella complessità della stessa e quindi nella necessità di disporre di una rete neurale di maggiori dimensioni, ma allo stesso tempo più veloce nell'eseguire il processo di addestramento. Si potrebbe quindi pensare di passare ad una architettura *multi thread* in modo da velocizzare il processo e capire quanto si possa ridurre effettivamente l'errore con reti grandi. Un'altra ipotesi è che gli insiemi di addestramento forniti non siano rappresentativi della funzione o non correttamente stratificati influenzando cosi negativamente i risultati ottenuti. Purtroppo non sono stato in grado di giungere a risultati accettabili seguendo questa strada e ho dovuto rinunciare ad effettuare previsioni mediante questo tipo di modelli.

Capitolo 8

Conclusioni

Durante la fase iniziale di questo elaborato di tesi ho analizzato il problema dello studio dell'incertezza. Per fare ciò, nel capitolo 2, ho descritto una tassonomia utile alla classificazione del fenomeno, la quale mi è stata poi utile durante il processo di identificazione. Questo processo, cosi come quello di gestione dell'incertezza, è stato guidato dal MUSE, un algoritmo descritto nel capitolo 3. Per la fase di identificazione, rispettando l'approccio *top-down* suggerito dal MUSE, ho descritto una serie di test utili a rilevare il fenomeno e riuscire a ricondurlo alla classe di incertezza più semplice e rappresentativa. Una volta compreso il tipo di incertezza da affrontare ho suggerito diversi metodi utili alla gestione del problema, limitatamente al campo dei modelli a rete di code. L'unione di queste fasi di analisi mi ha portato, in una fase successiva, a sviluppare un software in grado di fornire tutte le funzionalità necessarie ad individuare e gestire il fenomeno dell'incertezza su modelli a rete di code in modo automatizzato. Grazie a questo programma è possibile fornire predizioni del tempo di risposta di un server che si desidera simulare e valutare la qualità delle stesse. Nel capitolo 7 ho introdotto un differente approccio alla gestione dell'incertezza, presentando un *tool* adibito alla creazione di reti neurali. Tali reti sono state addestrate per cercare di ottenere previsioni utilizzabili riguardo al tempo di risposta di un server simulato.
I dati utilizzati, descritti nel capitolo 5, sono stati acquisiti dall'università di Saragozza e dai server di Wikipedia. I risultati ottenuti mediante i due programmi ed i due differenti approcci utilizzati sono stati significamente diversi. Con il primo software, utilizzando un approccio statistico per la realizzazione dei modelli delle richieste, è stato possibile ottenere previsioni del tempo di risposta di un server utilizzando le relazioni note dalla teoria delle code. Queste previsioni si sono rivelate più o meno accurate, a seconda della tecnica utilizzata per la creazione del modello, ma comunque utilizzabili e migliorabili con l'introduzione di un *discrepancy term*. L'utilizzo delle reti neurali come modello non ha invece portato a previsioni sufficientemente precise da essere tenute in considerazione, come evidenziato nel capitolo 7.
Per quanto riguarda il software pricipale, che utilizza un approccio statistico al problema, esso è riuscito ad identificare un fenomeno di incertezza di tipo strutturale nel caso di studio Unizar.

I metodi utilizzati per realizzare modelli del traffico delle richieste hanno fornito risultati di differente qualità:

- Tecnica di *Split*: Non può essere ritenuta affidabile, le predizioni ottenute mediante i modelli prodotti sono risultate troppo casuali e l'idea di partizionare i log dei dati con questo criterio non può essere una strada perseguibile a meno di alcuni casi estremi.
- DBSCAN: Il metodo di *clustering* applicato si è dimostrato naturalmente predisposto all'individuazione degli *outlier,* se calibrato opportunamente. Il trattare questi valori separatamente dai dati ritenuti non anomali, eseguendo successivamente tecniche di aggregazione, ha migliorato la qualità delle predizioni ottenendo risultati stabili. In alcuni casi è possibile che il metodo consideri alcuni valori centrali del log come *outlier* per via della presenza di aree a bassa densità, ciò risulta poco desiderabile volendo andare ad individuare i picchi di richieste. Le predizioni ottenute, anche in questi casi risultano comunque di un buon livello.
- Suddivisione dei dati in aree attraverso la computazione di limiti: Questa tecnica cosi come il DBSCAN ha ottenuto buoni risultati, grazie all'interfaccia grafica fornita è possibile per l'utente individuare ed isolare agevolmente i picchi delle richieste al fine di poterli trattare separatamente ed aggregare successivamente le predizioni. Se calibrato opportunamente questo metodo ha fornito risultati paragonabili a quelli del DBSCAN.
- Ricerca di *pattern*: Questa approccio ha fornito le predizioni di migliore qualità, dimostrandosi sempre stabile nei risultati prodotti. Il pattern che è risultato più preciso nella predizione del tempo di risposta è quello chiamato *Mix* che va a combinare la divisione dei dati in notturni e diurni con i valori di richieste registrate in settimana e nel fine settimana. Studiando l'area linguistica inglese dai dati provenienti dai server di Wikipedia nono sono state però evidenziate grandi differenze di funzionamento nei diversi periodi temporali.

Per tutte e quattro le tecniche utilizzate dal software principale è stato necessario introdurre un *discrepancy term,* descritto nella sezione 6.6, al fine di ridurre la discrepanza tra il valore di tempo di risposta desiderato a quelli predetti dai modelli. Questo termine additivo è stato computato sulla base della frequenza dei picchi di richieste rilevati nei vari insiemi di dati e sulla loro intensità. La formula utilizzata per il calcolo del termine è stata quindi ottenuta in modo empirico e potrebbe non essere utilizzabile su insiemi di dati differenti. L'introduzione del termine ha comunque significativamente migliorato le predizioni realizzate, avvicinandole al valore desiderato.

8.1 Sviluppi futuri

Per quanto riguarda l'uso delle reti neurali, non essendo stato in grado di ottenere risultati attendibili mediante il software realizzato, vi è da considerare la possibilità di modificare le librerie sviluppate. Operando un lavoro di ottimizzazione sulle funzioni di addestramento e rendendo l'intera libreria *thread safe* si potrebbe utilizzare reti neurali più complesse passando ad una architettura *multithread.* Gestendo più processi in parallelo sarà possibile sfruttare maggiormente i *core* della CPU accelerando il processo di apprendimento delle reti ed avendo cosi la possibilità di giungere a dei risultati conclusivi, positivi o negativi che siano. Si potrebbe inoltre pensare di implementare delle ottimizzazioni per quanto riguarda la scelta delle configurazioni dei pesi della rete. La libreria utilizzata attualmente fornisce un meccanismo di *Early Stopping,* ma forse utilizzare la tecnica di *Weight Decay* faciliterebbe maggiormente il processo di addestramento.
Per quato riguarda il software principale, che utilizza un approccio statistico, potrebbe essere utile implementare ulteriori criteri nella tecnica di ricerca dei *pattern*, la quale si è dimostrata la migliore come qualità delle predizioni realizzate mediante i suoi modelli. Inoltre, utilizzando altri insiemi di dati, si potrebbe cercare di giungere ad una formula applicabile a livello più generale per la computazione del *discrepancy term.* Quella utilizzata attualmente infatti è stata pensata sui due casi di studio affrontati ma sarà difficilmente applicabile a livello generale. Anche andando ad utilizzare diversi insiemi di dati ritengo sensato basare il calcolo del termine additivo sui picchi di richieste rilevati e sulla loro intensità.

BIBLIOGRAFIA

[1] Knight, F. H. Risk, Uncertainty e Profit. *Boston: Hart, Schaffner & Marx*. 1921

[2] W. Walker, P. Harremo´ Ss, J. Romans, J. van der Sluus, M. van Asselt, P. Janssen e M. Krauss. Defining uncertainty. a conceptual basis for uncertainty management in model-based decision support. In *Integrated Assessment*. 4:5-17. 2003.

[3] Diego Perez-Palacin e Raffaela Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: a Taxonomy and an Example of Availability Evaluation. In *Proc. of the 5th ACM/SPEC international conference on Performance engineering*. pp. 3-14. 2014.

[4] P. G. Armour. The five orders of ignorance. *Communications.* ACM. pp. 17-20. 2000.

[5] http://en.wikipedia.org/wiki/Agner_Krarup_Erlang. 10 Luglio 2014.

[6] Kendall, D. G. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics 24*. pp. 338-354. 1953.

[7] Diego Perez-Palacin e Raffaela Mirandola. Dealing with uncertainties in the performance modelling of software systems. In *Proc. of the 10th international ACM Sigsoft conference on Quality of software architectures*. pp. 33-42. 2014 .

[8] http://en.wikipedia.org/wiki/Hill_climbing. 2 Ottobre 2014.

[9] Ester, Martin, Kriegel, Hans-Peter, Sander, Jörg, Xu, Xiaowei. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Data Mining and Knowledge Discovery*. Vol 2 Issue2. pp.169-194. 1998.

[10] http://en.wikipedia.org/wiki/Binary_classification, 25 Novembre 2014

[11] Christopher M. Bishop. Pattern *Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York . 2006.

[12] McCulloch, Warren e Walter Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*. 52(1-2):99-115. 1943.

[13] Hebb e Donald. The Organization of Behavior: A Neuropsychological Theory. *New York: Wiley*. 1949.

[14] Farley, B.G. E W.A. Clark. Simulation of Self-Organizing Systems by Digital Computer. *IRE Transactions on Information Theory*. pp. 76-84. 1954.

[15] Rosenblatt, F. The Perceptron: A Probalistic Model For Information Storage And Organization In The brain. *Psychological Review.* Vol 65. pp. 386-408. 1958.

[16] Minsky M. L. e Papert S. A. *Perceptrons*. *Cambridge, MA: MIT Press*. 1969.

[17] Bartlett, P.L.,in Mozer, M.C., Jordan,M.I., e Petsche, T. Advances in Neural Information Processing Systems 9. *Cambridge, MA: The MIT Press*. 1997.

[18] http://en.wikipedia.org/wiki/University_of_Zaragoza. 20 Agosto 2014.

[19] http://dumps.wikimedia.org/other/pagecounts-raw/. 10 Luglio 2014.

[20] http://www.datacenterknowledge.com/archives/2008/06/24/a-look-inside-wikipedias-infrastructure/. 5 Novembre 2014.

[21] http://meta.wikimedia.org/wiki/List_of_Wikipedias. 11 Luglio 2014.

[22] https://support.gooddata.com/entries/38190507-Normality-Testing-Skewness-and-Kurtosis. 25 Agosto 2014.

[23] http://en.wikipedia.org/wiki/Exponential_distributio. 25 Agosto 2014.

[24] http://it.wikipedia.org/wiki/Varianza. 26 Agosto 2014.

[25] Mark Strong, Jeremy E. Oakley e Jim Chilcott. Managing structural

uncertainty in health economic decision models: a discrepancy approach. *Journal of the Royal Statistical Society: Series C.* Vol 61. pp. 25-45. 2012.

[26] http://msdn.microsoft.com/it-it/vstudio/aa496123.aspx. 11 Luglio 2014.

[27] http://www.visualstudio.com/. 11 Luglio 2014.

[28] http://www.w3.org/XML/. 5 Ottobre 2014.

[29] http://www.w3.org/. 5 Ottobre 2014.

[30] http://www.r-project.org/. 10 Luglio 2014.

[31] http://cm.bell-labs.com/cm/ms/departments/sia/S/index.html. 11 Luglio 2014.

[32] http://it.wikipedia.org/wiki/Bell_Laboratories. 11 Luglio 2014.

[33] http://msdn.microsoft.com/it-it/library/system.windows.forms.datavisualization.charting.chart. 24 Agosto 2014.

[34] http://www.centerspace.net/products/nmath, 5 Ottobre 2014.

[35] http://www.centerspace.net/doc/NMathSuite/ref/html/T_CenterSpace_NMath_Core_RandGenExponential.htm, 5 Ottobre 2014.

Appendice A: Tecnologie utilizzate

In questa appendice presenterò le principali tecnologie utilizzate durante la fase di progetto per la realizzazione del *tool* di analisi dei dati, degli *script* per la acquisizione dei dati e per l'implementazione della rete neurale. Cercherò inoltre di motivare le scelte effettuate al fine di rendere il più chiaro possibile il processo di sviluppo.

.NET

.NET [26] è una piattaforma di sviluppo software creata da Microsoft che include linguaggi di programmazione quali C#, Visual Basic, CLR ed una estesa libreria di classi. La prima versione di questa piattaforma è stata distribuita nel 2002 subendo da allora continue evoluzioni. Una delle caratteristiche peculiari di .NET è quella di essere indipendente dalla versione di Windows installata sulla macchina in uso oltre ad includere molte funzionalità progettate espressamente per ambiente internet; essa inoltre è strutturata per garantire il massimo grado di sicurezza ed integrità dei dati.

Figura A.1. Logo Visual Studio 2013.

L'IDE (*Integrated Development Enviroment*) associato alla piattaforma è Visual Studio [27] , il quale è stato utilizzato nell'ambito di questo progetto per la realizzazione del software di analisi dei dati, la creazione dei modelli e per gli *script* realizzati incaricati dell'acquisizione dei campioni. Più precisamente, per lo sviluppo, sono andato ad utilizzare la versione Visual Studio 2013 Ultimate fornita in licenza gratuita dal Politecnico di Milano. Per sviluppare sia il *tool* principale che tutti gli *script* ho scelto di avvalermi della programmazione ad oggetti con l'idea di realizzare librerie di classi e funzionalità che potessero essere eventualmente riciclate o espanse in futuro. Il linguaggio di programmazione designato per l'implementazione è stato C#, la scelta è ricaduta su tale linguaggio sia per le potenzialità che esso offre sia per la vasta documentazione a disposizione per le sue librerie standard. Sia Java che c++ sarebbero state probabilmente valide alternative ma vista la mia maggiore

dimestichezza con tale linguaggio ho preferito muovermi su un terreno maggiormente conosciuto.
L'interfaccia utente è stata sviluppata anch'essa in C#, sfruttando la programmazione ad eventi ed i Windows Form. Questa scelta è stata dettata dalla facilità con cui sia possibile generare una interfaccia utente, anche complessa, con poche azioni *drag and drop* dall'editor, autogenerando parte del codice. Un altro motivo che mi ha portato ad utilizzare questo linguaggio è il pieno supporto alla programmazione parallela mediante librerie standard e ben documentate. Questo tipo di programmazione si è rivelata utile in alcuni casi per garantire all'utente la possibilità di utilizzare controlli durante l'esecuzione delle analisi. Viste quindi le mie conoscenze pregresse riguardo l'ambiente .NET e l'IDE di sviluppo Visual Studio oltre alla funzionalità che avrebbero potuto essere richieste nel corso del progetto non ho avuto dubbi sull'orientarmi su questa tecnologia.

XML

XML [28] (*eXstensible Markup Language*) è un linguaggio marcatore derivato da SGML, esso definisce un insieme di regole per codificare documenti in un formato che sia leggibile sia da macchine che da umani. La prima specifica di questo linguaggio fu realizzata da W3C [29] (*World Wide Web Consortium*) alla fine degli anni '90 con lo scopo di formalizzare quali caratteristiche comuni dovessero avere le varie versioni introdotte nello standard HTML durante la celebre guerra dei browser tra Microsoft ed il suo Internet Explorer e Netscape. Successivamente ci si rese però conto che XML rappresentava uno strumento molto più potente, non solo limitato al contesto web ma in grado di rappresentare documenti, consentire lo scambio di informazioni tra sistemi differenti, rappresentare immagini o definire formati di dati.
Nell'ambito di questo progetto ho scelto di utilizzare XML come linguaggio di *markup* per i dati in quanto esso si integra perfettamente con l'ambiente .NET. La piattaforma di sviluppo mette infatti a disposizione librerie di classi che facilitano la gestione di dati in formato XML sia nella fase di scrittura che di lettura, rendendo semplici queste operazioni. L'ambiente .NET consente di serializzare intere classi o selezionare quali informazioni in esse contenute si desidera salvare su file. Questo formato è stato utilizzato durante la fase di salvataggio dei modelli prodotti in uscita dal *tool* oltre che per preservare i risultati dell'esito delle simulazioni realizzate.

R

R [30] è un GNU *Project* ed è sia un linguaggio che un ambiente utilizzato per analisi statistiche anche di tipo grafico, può essere considerato come una diversa implementazione del linguaggio S [31]. Quest'ultimo è stato sviluppato nei Bell Laboratories [32], formalmente dall'AT&T, ora Lucent Technologies, da John Chambers ed i suoi colleghi. Non vi sono molte differenze tra i due linguaggi e sostanzialmente la gran parte del codice scritta per S può essere eseguita su R senza necessitare alcuna modifica. R è stato progettato come un vero e proprio linguaggio di programmazione, esso consente agli utenti di aggiungere nuove funzioni e nel caso di compiti computazionalmente onerosi è possibile eseguire a *run time* del codice esterno scritto in C, C++ o Fortran. Inoltre è possibile utilizzare codice C per manipolare direttamente oggetti definiti in R. Le funzionalità messe a disposizione possono essere ulteriormente estese tramite l'uso di pacchetti aggiuntivi, scaricabili da una vasta lista sul sito principale del progetto. Tali pacchetti possono aggiungere nuove analisi statistiche e funzionalità ad un ambiente che è in costante evoluzione. Sebbene non abbia usato R direttamente nello sviluppo del progetto esso è risultato fondamentale in una fase iniziale fornendo spunti grazie alla grande varietà di analisi statistiche che questo *tool* consente. Un esempio sono gli innumerevoli algoritmi di *clustering* proposti nei vari pacchetti che mi hanno dato l'idea di sperimentare l'algoritmo di DBSCAN nella fase di gestione dell'incertezza per la creazione di modelli.

Appendice B: Implementazione

In questa appendice mi concentrerò sull'implementazione del *tool,* mostrato in funzione nel capitolo 6, ed utilizzato per l'applicazione di metodi volti ad individuare e gestire l'incertezza. Mostrerò inoltre l'implementazione dell'applicazione utilizzata per la realizzazione delle reti neurali, mostrata nel capitolo 7. Verranno commentati stralci di codice, ritenuti da me rilevanti, al fine di descrivere nel modo più chiaro possibile il funzionamento dei programmi e le scelte effettuate in fase di sviluppo.

Implementazione del tool per lo studio dei dati

Navigazione tra le pagine

La navigazione tra le varie pagine del programma viene gestita con la logica della programmazione ad eventi, andando ad individuare le interazioni che avvengono tra l'utente e l'interfaccia mediante rilevamento dei click del mouse.

Listing B.1: *Event handler mouse click per eseguire navigazione alla pagina di dbscan*

```
1        private void dbscanMenuItem_Click(object sender, EventArgs e)
2        {
3          DBScanPage dbscanPage = new DBScanPage(this);
4         dbscanPage.Show();
5         Hide();
6        }
```

Nel codice in B.1 vi è un esempio del processo di navigazione tra la *Main Page* e la pagina contenente il metodo utilizzato per li *clustering*, il DBSCAN. Il cambio di schermata avviene condizionatamente al click dell'utente sulla voce del menu adibita a tale compito. Una volta rilevato l'*input* il programma, raccogliendo l'evento di click del mouse, lancia il metodo *dbscanMenuItem_Click* il quale genera una nuova pagina di destinazione. Terminata la costruzione della pagina di arrivo il contesto corrente viene spostato, rendendo invisibile la schermata attuale ed andando ad attivare quella di destinazione appena creata.

Caricamento dei Log nel programma

Il codice mostrato in B.2 è inerente al metodo utilizzato per il caricamento dei log all'interno del programma e come i Windows Forms vengano utilizzati per guidare l'utente nella selezione. L'algoritmo utilizza un filtro in modo da consentire la selezione di solo i file la cui estensione è supportata. Un elenco dei tipi supportati ed una loro descrizione è disponibile nel capitolo 6.

Listing: B.2: *Stralcio di codice utilizzato per la selezione del log da caricare*

```
1       try
2       {
3         loadFile.InitialDirectory = Directory.GetCurrentDirectory() + @"\Logs\";
4         loadFile.Filter = "Logs (*.1min)|*1min|Logs (*.1hour) |*1hour";
5         loadFile.FilterIndex = 1;
6         loadFile.RestoreDirectory = true;
7         loadFile.ShowDialog();
8         selectedLog = loadFile.FileName;
9         LoadDataset();
10          ...
11      }
12      catch (Exception exc) {...}
```

Dopo aver operato la selezione di un file supportato la parte di caricamento vera e propria viene demandata al metodo *LoadDataset* visibile in B.3.

Listing B.3: *Metodo per il caricamento dei log inserito nella MainPage*

```
1       public void LoadDataset()
2       {
3         ...
4        start = new SamplingTime();
5         try
6       {
7         using (StreamReader sr = new StreamReader(selectedLog))
8         {
9           String temp = sr.ReadToEnd();
10          Char[] separator = { '\n' };
11          String[] content = temp.Split(separator);
12          data = new List<double>();
13          String[] startingDate = content[0].Split(' ', '\r');
14          start.Hour = int.Parse(startingDate[0]);
15          start.Minute = int.Parse(startingDate[1]);
16          start.Day = (DayOfWeek)Enum.Parse(typeof(DayOfWeek),
17          startingDate[5]);
18          for (int i = 1; i < content.Length; i++)
19          {
20            content[i]= content[i].Replace(',', '.');
21          }
22          for (int i = 1; i < content.Length; i++)
```

```
23          {
24            try
25            {
26              data.Add(double.Parse(content[i],
27               CultureInfo.InvariantCulture));
28            }
29              catch (Exception exc){...}
30            }
31          }
32        }
33        catch (Exception exc){...}
```

L'algoritmo utilizzato esegue la lettura dei log e successivamente l'estrazione e conversione dei dati contenuti. La prima riga del file viene trattata in modo differente dalle successive, in essa infatti sono contenute informazioni di tipo temporale sul quando la registrazione dei dati è iniziata.
I campi presenti in tale riga rispettano un ordine fissato e non mutabile: Ora, Minuto, Data, Mese, Anno, Giorno. Tali campi, separati dal carattere di spazio, vengono estratti singolarmente ed utilizzati per popolare la classe *SamplingTime*, visibile in B.3.

Listing B.3: *Classe SamplingTime*

```
1         public class SamplingTime
2        {
3          public double Minute
4          {
5            get;
6            set
7              {
8                if (value > 59 || value < 0) throw new ArgumentOutOfRangeException();
9                else this.Minute = value;
10             }
11         }
12         public double Hour
13        {
14           get;
15           set
16             {
17               if (value > 59 || value < 0) throw new ArgumentOutOfRangeException();
18               else this.Minute = value;
19             }
20         }
21         public DayOfWeek Day { get; set; }
22         public int DayNumber { get; set; }
23         public Month Month { get; set; }
24         public uint Year { get; set; }
25        }
```

Questa classe è stata definita appositamente per descrivere, con il livello di precisione desiderata, i singoli istanti di campionamento e viene largamente utilizzata in alcuni metodi di analisi. Durante la fase di caricamento l'unico istante che è effettivamente rilevante è quello iniziale, gli altri, se richiesto, possono essere calcolati in funzione di esso, conoscendo l'intervallo che intercorre tra la registrazione di un campione e l'altro. Le altre linee del log vengono trattate invece dall'algoritmo, mostrato in B.3, in modo differente. Dopo la lettura i dati estratti vengono registrati su di una lista di variabili in formato *double* pronte per essere utilizzate nel corso delle analisi successive. Nel caso in cui sia necessario, il programma ha inoltre la capacità di correggere la sintassi dei dati inseriti nel log fornito in ingresso. In questo modo l'utente ha la possibilità di utilizzare sia la virgola che il punto per segnalare i valori decimali.

Fase di disegno dei grafici

Per tracciare tutti i grafici presenti nelle varie schermate del programma il procedimento prevede l'uso di funzioni fornite nelle librerie *standard* .NET. Nel processo di disegno mi sono avvalso del componente *Chart* contenuto nella libreria *System.Windows.Form.DataVisualization* [33], grazie al quale è possibile, in poche righe di codice, riuscire a far apparire un grafico con l'aspetto desiderato.

Listing B.4: *Esempio di uso del componente Chart per il Plot del grafico dell'andamento dei dati nella Main page*

```
1       dataChart.Series["Data"].ChartType = SeriesChartType.FastLine;
2       dataChart.Series["Data"].Color = Color.Blue;
3       dataChart.ChartAreas["Default"].AxisY.Title = "Requests";
4       dataChart.ChartAreas["Default"].AxisX.ScaleView.Zoomable = true;
5       dataChart.ChartAreas["Default"].AxisY.ScaleView.Zoomable = true;
6       dataChart.ChartAreas["Default"].AxisX.ScrollBar.IsPositionedInside = true;
7       dataChart.ChartAreas["Default"].AxisY.ScrollBar.IsPositionedInside = true;
8       dataChart.ChartAreas["Default"].CursorX.AutoScroll = true;
9       dataChart.ChartAreas["Default"].CursorY.AutoScroll = true;
10      dataChart.ChartAreas["Default"].CursorX.IsUserSelectionEnabled = true;
11      dataChart.ChartAreas["Default"].CursorY.IsUserSelectionEnabled = true;
12      dataChart.Series["Data"].Points.Clear();
13      for (int i = 0; i < data.Count; i++)
14      {
15          dataChart.Series["Data"].Points.AddY(data[i]);
16      }
```

Guardando il codice in B.4 si può osservare come sia immediato l'utilizzo di questo componente. Per poter tracciare un grafico per prima cosa è necessario definire una nuova Serie di dati, ossia i punti che verranno visualizzati in esso. A tal fine è possibile utilizzare direttamente l'IDE di sviluppo Visual Studio

andando a modificare le proprietà dell'oggetto di tipo *Chart* disponibili mediante interfaccia grafica. Nel caso mostrato viene definita una unica serie Data, ma in generale è possibile tracciare più grafici sovrapposti inserendo molteplici serie. L'attributo *chartType* ha lo scopo di precisare il tipo di grafico scelto mentre quello *color* si riferisce appunto al colore da utilizzare per rappresentarlo. Oltre a queste caratteristiche vengono anche definite delle etichette associate agli assi del grafico, in modo da facilitare la lettura da parte dell'utente, ed alcuni attributi necessari per permettere le azioni di *zoom in* e *zoom out*. L'inserimento dei dati all'interno del grafico avviene mediante un processo iterativo, contenuto tra la riga 13 e la riga 16, al termine del quale la componente *Chart* si occuperà della rappresentazione.

Algoritmo per verificare se una distribuzione di dati è di tipo esponenziale

Il codice presente in B.5 rappresenta un algoritmo di test per andare a verificare una eventuale asimmetria della distribuzione dei dati, e la direzione in cui essa si verifica.

Listing B.5: *Algoritmo utilizzato per il test incaricato di verificare se la distribuzione dei dati è asimmetrica e mostrare le informazioni nella text box presente nella pagina principale.*

```
1        meanValue = 0;
2        meanValue = Utilities.computeMean(data);
3        variance = Utilities.computeVariance(data);
4        double num = 0;
5        double den = 0;
6        double skewedTestResult = 0;
7        for (int i = 0; i < data.Count; i++)
8        {
9           num += (double)Math.Pow(data[i] - meanValue, 3);
10       }
11       num = num / (data.Count - 1);
12       for (int i = 0; i < data.Count; i++)
13       {
14          den += (double)Math.Pow(data[i] - meanValue, 2);
15       }
16       den = (double)Math.Pow(den / (data.Count - 1), 3 / 2);
17       skewedTestResult = num / den;
18        if (skewedTestResult > 0)
19       {
20          distributionInfo.Text += "Skewed (right) \n";
21       }
22       else
23       {
24          distributionInfo.Text += "Skewed (left) \n";
25       }
```

Combinando il codice in B.5 e quello riportato in B.6 è possibile verificare se una distribuzione di dati è riconducibile ad una distribuzione esponenziale. Il primo metodo mostrato consente di escludere distribuzioni che hanno una asimmetria scorretta andando ad implementare la formula 6.2.

Listing B.6: *Test per verificare se la distribuzione dei dati è riconducibile ad una distribuzione esponenziale.*

```
1       if (Math.Abs(variance - (1 / Math.Pow(1 / meanValue, 2))) > threshold )
2           isExp= false;
3       else
4           isExp = true;
```

Il codice in B.6 viene invece utilizzato dal software per verificare se la distribuzione dei dati caricati in esso sia di tipo esponenziale, andando a sfruttare le relazioni tra media e varianza note per questo tipo di distribuzione. Il metodo mostrato viene però utilizzato dal software solo dopo aver effettuato una selezione sulle distribuzioni che palesemente non possono rientrare in questa categoria, mediante la funzione in B.5.

Algoritmo per individuare la presenza di incertezza sui dati

Questa analisi, che è già stata presentata in precedenza nel capitolo 3, desidera verificare la presenza di incertezza sui dati caricati all'interno del software. L'algoritmo viene applicato dal programma andando a leggere i parametri inseriti dall'utente tramite interfaccia grafica.

Listing B.7: *Algoritmo di test per individuare la presenza di incertezza sui dati posto nella MainPage*

```
1       double threshold = double.Parse(tolleranceTextBox.Text,
2       CultureInfo.InvariantCulture)/100;
3       double fraction = double.Parse(fractionTextBox.Text,
4       CultureInfo.InvariantCulture)/100;
5       List<double> mus = new List<double>();
6       if (data.Count != systemLoadData.Count) throw new ArgumentException();
7       for (int i = 0; i < data.Count; i++)
8       {
9         if (systemLoadData[i] != 0)
10                mus.Add(data[i]/systemLoadData[i]);
11        else mus.Add(0);
12      }
13      double avgmu = Utilities.computeMean(mus);
14      List<double> calcoulatedSystemLoadData = new List<double>();
15      for (int i = 0; i < systemLoadData.Count; i++)
16      {
17        calcoulatedSystemLoadData.Add(data[i] / avgmu);
```

```
18      }
19      int count = 0;
20      for (int i = 0;i < systemLoadData.Count; i++)
21      {
22        if (Math.Abs(calcoulatedSystemLoadData[i]-systemLoadData[i]) > threshold) count++;
24      }
```

Il parametro di tolleranza, fornito in ingresso al metodo, visibile nel codice in B.7 con il nome *threshold*, ha la funzione di andare a stabilire una soglia. Più sarà grande il valore assegnato alla variabile più l'algoritmo si dimostrerà permissivo nell'accettare valori calcolati differenti da quelli misurati. Il secondo parametro, *fraction* in riga 3, va invece ad indicare la frazione minima di dati, che devono essere classificati come differenti da quelli misurati, necessaria perchè il test rilevi una forma di incertezza. Nel ciclo *for*, riga 20, viene calcolata la quantità di valori che mostrano eccessiva discrepanza utilizzando la variabile *count* come contatore. Presa la lista dei valori calcolati e quella dei valori misurati viene estratta iterativamente una coppia di essi e se la loro differenza è sufficientemente grande il contatore viene incrementato. Al termine della fase di test l'algoritmo procede alla verifica della quantità di coppie diverse e se il numero registrato è sufficiente per superare la soglia impostata dall'utente.

Calcolo della varianza campionaria

Questo metodo si occupa di calcolare la varianza campionaria e viene largamente utilizzato dal programma, soprattutto nei test che vanno a determinare la dispersione dei dati, come quello mostrato nella sezione 6.2.2. La funzione viene fornita come metodo statico dalla libreria sviluppata appositamente per il *tool* al fine di essere accessibile a tutti i metodi che potrebbero desiderare utilizzarla.

Listing B.8: *Metodo statico posto nella classe Utilities utilizzato per il calcolo della varianza campionaria*

```
1       public static double computeVariance(List<double> data)
2       {
3         double variance = 0;
4         double meanValue = computeMean(data);
5         if (data.Count == 0) return 0;
6         for (int i = 0; i < data.Count; i++)
7         {
8                 variance += Math.Pow((data[i] - meanValue), 2);
9         }
10        variance = variance / (data.Count - 1);
11        return (double) variance;
12      }
```

Hill Climbing

L'algoritmo mostrato in B.9 viene utilizzato nel test per la verifica della presenza di incertezza strutturale ed è mostrato in funzione nella sezione 6.2.4.

Listing B.9: *Algoritmo di test per l'individuazione dell'incertezza strutturale posizionato nella MainPage*

```
1       double maxMutations = double.Parse(mutationsTextBox.Text,
2       CultureInfo.InvariantCulture) / 100;
3       List<double> mus = new List<double>();
4       if (data.Count != systemLoadData.Count) throw new ArgumentException();
5       for (int i = 0; i < data.Count; i++)
6       {
7         if (systemLoadData[i] != 0)
8         mus.Add(data[i] / systemLoadData[i]);
9        else mus.Add(0);
10      }
11      List<double> orderedMus = mus;
12      orderedMus.Sort();
13      double muPercentile = mus[(mus.Count / 100) * 90];
14      List<double> Ns = new List<double>();
15      for (int i = 0; i < data.Count; i++)
16      {
17        Ns.Add(data[i] / muPercentile);
18      }
19       int max = (int)(data.Count * maxMutations);
20      int currentMutationStep = 0;
21      List<double> N = data;
22      while(currentMutationStep < max)
23      {
24        int selectedIndex = 0;
25        double selectedDelta =0;
26        for(int i=0;i< Ns.Count;i++)
27        {
28                  if(Math.Abs(Ns[i] -N[i]) > selectedDelta)
29                  {
30                    selectedIndex= i;
31                    selectedDelta = Math.Abs(Ns[i] - N[i]);
32                  }
33       }
34        Ns[selectedIndex] = N[selectedIndex];
35        currentMutationStep++;
36      }
37      double NsSum = Ns.Sum();
38      double NSum = N.Sum();
39      if (Math.Abs(NsSum - NSum) < NSum * 0.1)
```

Il metodo, come primo passo, va a calcolare μ^*, ossia il novantesimo percentile dei valori μ_i, riga 13. Tale valore viene computato utilizzando la relazione $\mu_i = N_i / \lambda_i$. Dopo aver determinato μ^* ,il metodo, iterando sui dati nel ciclo *for* a riga 15, va a ricavare una serie di nuovi carichi per il servente, utilizzando il parametro μ^*. Terminato il calcolo di questi carichi viene avviato l'algoritmo di *Hill Climbing* vero e proprio. Fino a quando il numero massimo di mutazioni inserite dall'utente non è stato superato, riga 22, si effettua una mutazione alla volta sui valori N_i^* scegliendo tra tutte le coppie (N_i^* , N_i), mediante il ciclo *for* a riga 26, quella la cui differenza è maggiore. Tale coppia viene individuata trovando quella il cui valore è pari alla variabile *selectedDelta* nel momento in cui viene analizzata l'ultima coppia di dati. Dopo aver eseguito il numero di mutazioni massime consentite dall'utente il test avvia la fase di verifica, in modo da poter controllare se il modello è sensibilmente migliorato variandone solo una parte. Per fare ciò l'algoritmo va a confrontare il carico misurato sul sistema ed i carichi calcolati e modificati mediante l'algoritmo genetico, da riga 37 a 39.

La classe Model

La classe *Model*, visibile in B.10, è adibita alla rappresentazione della porzione di realtà che si desidera modellizzare. Essa raccoglie diverse informazioni rilevanti al fine di poter effettuare le predizioni desiderate. Dell'insieme di dati utilizzato va ad incorporare il nome del file ed il numero di campioni in esso contenuti, sono inoltre presenti una breve descrizione del modello ed il metodo utilizzato per la sua realizzazione, salvato in un campo enumerativo. Risulta quindi possibile estendere l'enumerazione con ulteriori valori in seguito all'introduzione di nuove tecniche.

Listing B.10: *Classe Model utilizzata per raccogliere gli aspetti chiave di ciò che si desidera rappresentare*

```
1       public class Model
2       {
3         public String Name { get; set; }
4         public String DataSetName { get; set; }
5         public Method method { get; set; }
6         public int DataSetSize { get; set; }
7         public List<ModelComponent> Components { get; set; }
8         public String Description { get; set; }
9       }
```

Ogni modello contiene al suo interno una serie di componenti, uno per ogni insieme di dati che si desidera considerare separatamente, i quali possono essere ritenuti a tutti gli effetti dei modelli rappresentanti però solo una parte dei dati a disposizione.

Listing B.11: *Classe Model Component*

```
1       public class ModelComponent
2       {
3         public double PortionOfDataset { get; set; }
4         public double Mean { get; set; }
5         public double Variance { get; set; }
6         public string Description { get; set; }
7       }
```

Ogni oggetto di tipo *ModelComponent*, visibile in B.11, contiene al suo interno una serie di indicatori statistici utili per rappresentare la porzione di dati considerata. Tra gli indicatori utilizzati si può trovare la media e la varianza campionaria dei dati rappresentati, viene inoltre salvata la porzione dell'insieme dei dati interessata da tale componente. Sempre nella classe *ModelComponent* è presente una breve descrizione testuale, al fine di facilitare l'identificazione del componente da parte dell'utente. In B.12 è possibile vedere come venga creato un nuovo modello, dal metodo di *Split*, e come esso venga salvato mediante serializzazione della classe *Model* in formato XML.

Listing B.12: *Metodo utilizzato per la creazione e salvataggio del modello in formato XML nella pagina di Split*

```
1       Model model = new Model();
2       double meanValue = Utilities.computeMean(selectedData);
3       double variance = Utilities.computeVariance(selectedData);
4       model.DataSetName = PreviusPage.DsName;
5       model.Description = modelDescriptionTextBox.Text;
6       ModelComponent baseComponent = new ModelComponent();
7       baseComponent.Mean = meanValue;
8       baseComponent.Variance = variance;
9       baseComponent.PortionOfDataset = (double)selectedData.Count /
10      (double)data.Count;
11      baseComponent.Description = "Base";
12      model.Components = new List<ModelComponent>();
13      model.Components.Add(baseComponent);
14      model.DataSetSize = selectedData.Count;
15      model.Name = modelNameTextBox.Text;
16      model.method = Method.Split;
```

Le informazioni che vengono inserite nel file sono le seguenti:

- Una descrizione fornita dall'utente per identificarlo agevolmente in fase di lettura, scorrendo le informazioni associate ad esso mediante l'interfaccia grafica del software realizzato.
- Il metodo utilizzato per la realizzazione, utilizzando un apposito valore enumerativo.
- L'insieme dei dati di riferimento su cui è stato computato.
- Un unico componente contenente media e varianza campionaria dei dati.

Algoritmo utilizzato nel metodo di Split

Il codice in B.13 è inerente all'algoritmo utilizzato per l'attuazione del metodo di *Split*, mostrato in funzione nella sezoine 6.3.1.

Listing B.13: *Porzione di codice adibita alla lettura dei parametri di input e controllo dei valori inseriti dall'utente nella pagina di Split*

```
1       try
2       {
3         fromBound = int.Parse(fromTextBox.Text);
4         toBound = int.Parse(toTextBox.Text);
5         if (fromBound < 0 || toBound >= data.Count) throw new
6                 IndexOutOfRangeException();
7         selectedData = new List<double>();
8         for (int i = fromBound; i < toBound; i++)
9         {
10                selectedData.Add(data[i]);
11        }
12      }
13      catch (Exception exc)
14      {
15        String message = exc.Message;
16        String caption = "Error";
17        MessageBox.Show(message, caption, MessageBoxButtons.OK,
18        MessageBoxIcon.Error);
19        modelInfoTextBox.Text = "Error";
20        return;
21      }
```

Come visibile in B.13 i due parametri inseriti dall'utente, *"From "* e *"To"*, vengono processati e letti come *integer*, il loro valore inoltre viene controllato per verficare che essi siano compresi all'interno dei limiti dell'*array* contenente i dati caricati, riga 5. Nel caso in cui venga generato un problema nella fase di lettura o il valori inseriti siano scorretti il programma provvede a sollevare una eccezione, mostrata all'utente mediante un messaggio in *popup*, contenente la tipologia di errore. Se i valori letti risultano corretti il processo di *Split* viene avviato e tutti i campioni compresi tra il dato iniziale e quello finale sono selezionati e prelevati iterativamente dall'*array* contenente i dati da analizzare, ciclo *for* a riga 8.

Algoritmo di DBSCAN

Listing B.14: *Metodi che implementano il DBSCAN*

```
1       static List<List<Point>> GetClusters(List<Point> points, double eps, int minPts)
2       {
3         try
4         {
5                 if (points == null) return null;
6                 List<List<.Point>> clusters = new List<List<Point>>();
7                 int clusterId = 1;
8                 for (int i = 0; i < points.Count; i++)
9                 {
10                  Point p = points[i];
11                  if (p.ClusterId ==Point.UNCLASSIFIED)
12                 {
13                        if (ExpandCluster(points, p, clusterId, eps, minPts, bar))
14                        clusterId++;
15                  }
16                }
17                 int maxClusterId = points.OrderBy(p => p.ClusterId).Last().ClusterId;
18                if (maxClusterId < 1) return clusters;
19                for (int i = 0; i < maxClusterId; i++) clusters.Add(new List<Point>());
20                foreach (Point p in points)
21                {
22                        if (p.ClusterId > 0) clusters[p.ClusterId - 1].Add(p);
23                }
24                return clusters;
25        }
26       catch (Exception exc)
27       {... }
28    }
29
30      static List<Point> GetRegion(List<Point> points,Point p, double eps)
31      {
32        List<Point> region = new List<Point>();
33        for (int i = 0; i < points.Count; i++)
34        {
35                double distSquared = Math.Abs(p.Y - points[i].Y);
36                if (distSquared <= eps) region.Add(points[i]);
37        }
38        return region;
39    }
40
41      static bool ExpandCluster(List<Point> points,Point p, int clusterId, double eps, int
40      minPts)
43      {
44        List<Point> seeds = GetRegion(points, p, eps);
45        if (seeds.Count < minPts) // no core point
46        {
47                p.ClusterId = Point.NOISE;
48                return false;
```

```
49          }
50          else // all points in seeds are density reachable from point 'p'
51          {
52                  for (int i = 0; i < seeds.Count; i++) seeds[i].ClusterId = clusterId;
53                  seeds.Remove(p);
54                  while (seeds.Count > 0)
55                  {
56                    Point currentP = seeds[0];
57                    List<Point> result = GetRegion(points, currentP, eps);
58                    if (result.Count >= minPts)
59                    {
60                            for (int i = 0; i < result.Count; i++)
61                            {
62                              Point resultP = result[i];
63                              if (resultP.ClusterId ==Point.UNCLASSIFIED ||
64                              resultP.ClusterId ==Point.NOISE)
65                              {
66                              if (resultP.ClusterId ==Point.UNCLASSIFIED)
67                              seeds.Add(resultP);
68                              resultP.ClusterId = clusterId;
69                            }
70                    }
71                    bar.PerformStep();
72                  }
73                  seeds.Remove(currentP);
74            }
75            return true;
76        }
77    }
```

Il codice in B.14 illustra i tre metodi utilizzati nell'algoritmo che implementa il DBSCAN, mostrato in funzione nella sezione 6.3.2. *GetClusters* è la funzione principale, essa si avvale di *ExpandCluster* per espandere iterativamente i vicini del punto considerato. *ExpandCluster* a sua volta, utilizza il metodo *GetRegion* per determinare i punti con distanza minore del parametro ε, inserito dall'utente, rispetto al punto osservato p.

Algoritmo per la suddivisione dei dati mediante la computazione di limiti

L'algoritmo presente nel codice in B.15 viene mostrato in funzione nella sezione 6.3.3. Esso utilizza i dati contenuti nel log e ne calcola la varianza mediante la funzione *Utilities.computeVariance* fornita dalle librerie esterne e mostrata precedentemente. Dopo tale operazione i dati vengono ordinati e si procede a determinarne la parità in modo da poter calcolare la mediana (operazioni eseguite da riga 2 a riga 8). Una volta ottenuta la mediana è possibile computare un limite, al di sopra del quale tutti i valori dei campioni osservati verranno considerati dall'algoritmo come *outlier* e separati dal'insieme principale.

Questo limite superiore viene calcolato mediante la formula mostrata a riga 10 utilizzando il parametro denominato "K" inserito dall'utente. L'algoritmo mostrato va dunque ad implementare un classificatore binario.

Listing B.15: *Metodo utilizzato per la computazione dei limiti per la generazione di un classificatore binario.*

```
1       variance = Utilities.computeVariance(data);
2       orderedData = new List<double>();
3       for (int i = 0; i < data.Count; i++)
4       {
5               orderedData.Add(data[i]);
6       }
7       orderedData.Sort();
8       median = orderedData[(orderedData.Count / 2)];
9       k = double.Parse(kValueTextBox.Text, CultureInfo.InvariantCulture);
10      upperBound = median + k * variance;
```

Algoritmo di ricerca dei pattern

Questo algoritmo viene mostrato in funzione nella sezione 6.3.4 e si avvale di una serie di strutture dati visibili in B.16.

Listing B.16: *Struttura dati utilizzata per l'individuazione dei patterns*

```
1       public class PatternValue
2       {
3               public double Value;
4               public SamplingTime Time;
5               public Pattern AssociatedPattern;
6       }
7
8       public enum Pattern
9       {
10              None=0,Day,Night,Weekend,Week, DayWeek, DayWeekend, NightWeek,
11              NightWeekend
12      }
13
14      public class SamplingTime
15      {
16              public double Minute
17              {
18                      get;
19                      set;
20              }
21              public double Hour
22              {
23                      get;
24                      set;
```

```
25              }
26              public DayOfWeek Day { get; set; }
27              public int DayNumber { get; set; }
28              public Month Month { get; set; }
29              public uint Year { get; set; }
30      }
```

La classe *Pattern*, in B.16, viene utilizzata per il salvataggio dei valori del log, essa consente di associare informazioni aggiuntive ad ogni dato, come l'istante temporale in cui il campione in esame è stato registrato, identificato dal campo *Time* di tipo *SamplingTime*. Inoltre è possibile salvare il *pattern* che si desidera associare a tale valore determinandolo mediante l'istante temporale registrato. I tipi di *pattern* programmati sono stati caratterizzati con l'introduzione di una enumerazione, visibile in riga 8. Tale scelta è stata attuata nell'ottica di dare la possibilità di espansione inserendo nuovi criteri di ricerca nel programma, a patto di prevedere la creazione di nuovi metodi per ogni valore enumerativo introdotto. Nel codice in B.17 è possibile vedere il metodo utilizzato in fase di caricamento della pagina adibita alla ricerca dei *pattern.* Questo metodo si occupa della preparazione dei dati prima di poter applicare l'algoritmo di ricerca principale.

Listing B.17: *Metodo avviato durante il load della pagina adibita alla ricerca dei pattern per eseguire la preparazione dei dati.*

```
1       Data = PreviusPage.data;
2       Start = PreviusPage.start;
3       PatternValues = new List<PatternValue>();
4       PatternValue temp = new PatternValue();
5       temp.Value = Data[0];
6       temp.Time = Start;
7       PatternValues.Add(temp);
8       for (int i = 1; i < Data.Count; i++)
9       {
10              temp = new PatternValue();
11              temp.Value = Data[i];
12              temp.Time = Utilities.nextSamplingTime(PatternValues[i - 1].Time,
13                uint interval);
14              PatternValues.Add(temp);
15      }
```

Ogni dato contenuto nel log in ingresso viene convertito nel formato mostrato in B.16, arricchendolo delle informazioni necessarie a consentire la ricerca dei *pattern* desiderati. Al fine di individuare per ogni valore l'esatto istante di campionamento l'algoritmo chiama iterativamente il metodo *Utilities.nextSamplingTime*, il quale ha il compito di restituire, ricevendo in ingresso l'ultimo istante letto e l'intervallo di campionamento, il momento in cui è stato registrato il valore osservato . La prima iterazione di tale metodo viene svolta partendo dal primo campione, di cui si conosce l'istante di registrazione perchè segnalato nella prima riga del log.

Listing B.18: *Codice eseguito durante la creazione del modello nella pagina Patterns per selezionare il metodo corretto da eseguire in base al pattern scelto dall'utente*

```
1       if (selectPatternComboBox.SelectedIndex != -1)
2       {
3         switch (selectPatternComboBox.SelectedIndex)
4         {
5                 case 0:
6                 createModelDayNight();
7                 plot();
8                 break;
9                 case 1:
10                createModelWeekWeekEnd();
11                plot();
12                break;
13                case 2:
14                createModelMix();
15                plot();
16                break;
17        }
18      }
19      else
20      { … }
```

In B.18 è possibile vedere l'algoritmo principale il quale, in base al valore selezionato nel menu a tendina, corrispondente al valore enumerativo del *pattern* d'interesse, richiama la funzione più appropriata per la generazione, mediante l'uso di un costrutto di *switch*. Ognuna delle funzioni *createModelDayNight*, *createModelWeekWeekEnd* e *createModelMix* va a suddividere i dati secondo un criterio differente e grazie al costrutto usato, aggiungendo dei nuovi casi allo *switch*, vi è la possibilità di espandere i metodi di ricerca.

Simulazione di una rete di code

In B.19 è possibile vedere il metodo di avvio di una simulazione di una rete di code, mostrato in funzione nella sezione 6.4.

Listing B.19: *Processo di simulazione di una rete di code*

```
1       public List<double> startSimulation(InputType input)
2       {
3         double serverTime = 0;
4         double lastRequestTime = 0;
5         avgResponseTimeList = new List<double>();
6         int currentStep = 0;
7         List<double> sortedRate = new List<double>();
```

```
8           foreach (double d in AggrAvgArrivalRateList)
9           {
10                  sortedRate.Add(d);
11          }
12          sortedRate.Sort();
13          double percentile = sortedRate[(sortedRate.Count / 100) * Constants.percentile];
14          List<double> cleanedAggrAvgArrivalRateList = new List<double>();
15          foreach (double d in AggrAvgArrivalRateList)
16          {
17                  if (d < percentile) cleanedAggrAvgArrivalRateList.Add(d);
18          }
19          double simulationTime = cleanedAggrAvgArrivalRateList.Count *
20          Constants.simulationStepDuration;//Durata della simulazione
21          while (lastRequestTime < simulationTime)
22          {
23                  if (lastRequestTime > (currentStep + 1) * Constants.simulationStepDuration)
24                    currentStep++;
25                  double requestTime = lastRequestTime +
26                  Utilities.getExponentialNumber(cleanedAggrAvgArrivalRateList[currentStep]);
27                  if (!double.IsInfinity(requestTime))
28                  {
29                    lastRequestTime = requestTime;
30                    double responseTime = Math.Max(serverTime, requestTime) +
31                    server.getNextServiceTime();
32                    avgResponseTimeList.Add(responseTime - requestTime);
33                    serverTime = responseTime;
34                  }
35                  else
36                  {
37                    lastRequestTime = lastRequestTime + Constants.simulationStepDuration;
38                  }
39          }
40          return avgResponseTimeList;
41      }
```

L'algoritmo legge la frequenza di arrivo delle richieste, per ogni intervallo di campionamento, simulandole una per una con la frequenza corretta. Esistono due timer, uno associato al server (*simulationTime*) e uno associato all'ultima richiesta giunta (*lastRequestTime*). A seconda dell'istante temporale in cui è stata generata l'ultima richiesta di servizio l'algoritmo utilizza una frequenza di arrivo differente, determinata dai parametri presenti nel log caricato. Tramite la frequenza selezionata viene estratto l'intervallo di tempo che è necessario far trascorrere prima che giunga la prossima richiesta. Tale intervallo è calcolato mediante il metodo statico *Utilities.getExponentialNumber*, mostrato in B.20, che estrae dalla corretta distribuzione esponenziale un valore appropriato. Dopo aver ottenuto l'intervallo di tempo il timer *lastRequestTime* viene incrementato, spostandolo avanti nel tempo di tale valore, riga 29. Il server provvede ora a consumare la prima richiesta presente nella sua coda, andando ad estrarre un tempo di servizio associato ad essa, utilizzando nuovamente la funzione *Utilities.getExponentialNumber*.

Il valore del tempo di servizio che viene generato dipende direttamente dal parametro μ inserito, se si sta utilizzando un valore fisso o da quello selezionato dal modello del tempo di servizio caricato. In questo secondo caso il valore varierà di volta in volta a seconda della probabilità che ha un determinato μ di essere selezionato. Una volta ottenuto il tempo di servizio per la richiesta considerata, essa viene consumata ed il timer del server viene portato in avanti di un valore pari al tempo impiegato per tale operazione. Il processo viene iterato fino all'esaurimento delle richieste presenti sul log.

Listing B.20: *Metodo statico posto nella classe Utilities per ottenere un numero casuale secondo una distribuzione esponenziale con un certo parametro caratteristico*

```
1       public static double getExponentialNumber(double param)
2       {
3         return (Math.Log(1 - randomNumberGenerator.NextDouble()) /(-param));
4       }
```

Durante lo svolgimento del progetto di tesi ho provato ad utilizzare anche librerie alternative che implementassero direttamente la funzione per estrarre un numero casuale da una distribuzione esponenziale di parametro λ, come ad esempio NMATH 6.0 [34]. I risultati ottenuti da queste librerie, nello specifico dalla classe *RandGenExponential* [35] presente nella libreria *CenterSpace.NMath.core*, mi sono sembrati molto simili a quelli forniti utilizzando la funzione in B.20 portando quindi la mia scelta su di essa.

Calcolo del discrepancy Term

Questo metodo viene utilizzato per calcolare il termine additivo necessario per avvicinare le previsioni realizzate da un modello a quelle attese. Il suo funzionamento è stato mostrato nella sezione 6.6.

Listing B.21: *Metodo per il calcolo del discrepancy term inserito nella pagina per la gestione della rete di code*

```
1       foreach (double d in cleanedData )
2       {
3         if ((d/Time) >= Mu && (d/Time)< percentileValue)
4         {
5                 peaksCount++;
6                 term += (d/Time)/Mu;
7         }
8       }
9       ResponseTime += Math.Sqrt(ResponseTime * (term / (cleanedData.Count -
10      peaksCount)));
```

Il codice in B.21 va a calcolare il numero di volte in cui le richieste medie, contenute nel log, superano la capacità della macchina di servirle. Per ognuna di esse l'algoritmo stima di quanto questa capacità sia stata superata, incrementando il valore della variabile *term*. Il *discrepancy term* viene poi computato in funzione del tempo di risposta calcolato, del numero di picchi e della somma della loro intensità. Questa formula, in riga 9, è stata ottenuta in modo empirico, analizzando il valore desiderato di *discrepancy term* su vari modelli a disposizione, al fine di avere una quantità che garantisse sufficiente precisione e contemporaneamente un valore predetto inferiore a quello atteso.

Implementazione delle reti neurali

La libreria realizzata per la rete neurale è composta da diverse classi che implementano tutti i metodi necessari per il funzionamento della struttura e per il suo addestramento. Il codice è stato realizzato nell'ottica di consentire una eventuale futura espansione della libreria per quanto riguarda ad esempio le tipologie di reti supportate e nel caso si desiderasse parallelizzare alcune operazioni. Vista la complessità della funzione che si vuole far apprendere alla rete ho deciso di orientare le funzionalità della libreria sulla creazione di reti multistrato acicliche ignorando i meno potenti percettroni, mediante i quali difficilmente sarei riuscito a realizzare previsioni efficaci.

Struttura della rete

Nel codice in B.22 è possibile vedere le scelte attuate durante l'implementazione della struttura a strati della rete. La classe *Layer* contiene una funzione matematica ed un insieme di nodi. Tale funzione, durante la propagazione degli *input* nella rete, viene applicata a tutti i valori entranti nei nodi appartenenti allo strato ed è selezionabile dall'utente tra una serie di funzioni implementate. Ogni nodo contiene al suo interno il riferimento ai pesi sugli archi uscenti da esso oltre al valore di *input* ed il suo *output*. Sono presenti anche ulteriori campi che vengono utilizzati durante l'applicazione della tecnica di b*ackpropagation* al fine di aggiornare correttamente i pesi.

Listing B.22: *Nodi e Layer*

```
1       public class Node
2       {
3         public  double[] OutputWeights;
4         public double[] WeightsVariation;
5         public double InputValue;
6         public double OutputValue;
7         public double Delta;
8       }
```

```
9
10        public class Layer
11        {
12          public IMathematicalFunction Function;
13          public Node[] Nodes;
14          public Layer(){}
15
16          public void LayerInitialize(IMathematicalFunction function,int numNodes)
17          {
18                  this.Function = function;
19                  Nodes = new Node[numNodes];
20                  for (int i = 0; i < Nodes.Length; i++)
21                  {
22                    Nodes[i] = new Node();
23                  }
24          }
25        }
```

Funzioni matematiche implementate

Le funzioni matematiche attualmente implementate, visibili in B.23 sono le seguenti:

- Lineare
- Tangente iperbolica
- Sigmoide

Listing: B.23: *Funzioni matematiche definite per la libreria*

```
1         public interface IMathematicalFunction
2         {
3           double Value(double x);
4           double Derivative(double x);
5         }
6
7         public class HyperbolicTangent : IMathematicalFunction{...}
8
9         public class LinearFunction : IMathematicalFunction{...}
10
11        public class SigmoidFunction : IMathematicalFunction{...}
```

Al fine di consentire una futura espansione delle funzioni utilizzabili nei nodi della rete tutte le funzioni matematiche sono state implementate mediante l'uso di una interfaccia *IMathematicalFunction*. Grazie a questo accorgimento risulta possibile aggiungere ulteriori valori alla lista mosrata in precedenza a patto che essi implementino i metodi definiti nell'interfaccia.

La classe Network

La classe *Network*, si occupa di processare i vari esempi ed apprendere da essi implementando tutti i metodi necessari all'esecuzione della rete.

Listing B.24: *Classe Network*

```
1       public class Network
2       {
3         public String Name { get; set; }
4         public Layer[] layers;
5         private Random random = new Random();
6
7         public Network() { }
8         public void NetworkInitialize(IMathematicalFunction[] layersFunction, int[]topology)
9         {...}
10        public void LoadNetwork(double[][][] weights, IMathematicalFunction[]
11        layersFunction){...}
12
13        public void UpdateWeights(Sample sample, double eta){...}
14        public void Flush() {... }
15        public void ForwardPropagation(double[] inputs) {...}
16        public double[] ComputeOutputs(double[] inputs){...}
17      }
```

Nel codice in B.24 è possibile vedere l'elenco dei metodi forniti dalla classe *Network* oltre ad i campi in essa contenuti. Il metodo *NetworkInitialize* consente di inizializzare una nuova rete fornendo in ingresso un *array* contenente la sua topologia, ossia il numero di nodi che devono andare a comporre ogni strato. Riceve inoltre un ulteriore *array,* utilizzato per associare ad ogni strato della rete una funzione matematica da utilizzare. Vi è da notare che, a meno di caricare configurazioni precedenti, durante l'inizializzazione della rete i pesi vengono avviati in modo casuale. Il metodo *LoadNetwork* consente di utilizzare una rete già addestrata in precedenza, oppure determinare una precisa configurazione dei pesi prima di iniziare l'addestramento. La parte di addestramento della rete è gestita dai metodi *UpdateWeights*, *ForwardPropagation* e *Flush* i quali implementano la tecnica di propagazione a ritroso dell'errore commesso ed aggiornamento iterativo dei pesi. La tecnica di *backpropagation,* descritta nella sezione 4.4.2, viene applicata per gruppi di esempi, quindi l'aggiornamento dei pesi non avviene istantaneamente dopo la lettura di un solo dato. I campi *weightsVariation* e delta, visibili nel codice in B.22 e contenuti nella classe *Node*, servono per tenere traccia dei cambiamenti proposti dall'algoritmo per i pesi dopo la lettura di ogni esempio. Al termine della scansione del gruppo di esempi la configurazione della rete viene variata dall'algoritmo utilizzato in base alla aggregazione dei cambiamenti proposti per ogni singolo campione. Tale sistema di addestramento risulta più stabile rispetto al cambiare continuamente i pesi della rete nel momento in cui viene commesso un errore su un singolo esempio.