POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

POLITECNICO
MILANO 1863

TEXTUAL DATA QUALITY

Relatore
Prof. Pier Luca Lanzi

Correlatore
Dott. Andrea Lui

Candidato
Giuseppe Carone
Mat. 896669

Anno Accademico 2019-2020

Contents

List of Figures

List of Tables

Abstract

The term Data Quality nowadays brings a broad set of meanings. It is often referred as the fitness of data with regard to the intended uses [62] [81]. Specifically, Textual Data Quality refers to the analysis of natural language data. Being textual data one of the most abundant source of information available to companies, it is clear why more and more attention is posed on such problem.

The main goal of this work is hence to build a framework that allows to evaluate a set of target textual records given that it is not known a priori the classification task we are going to perform and hence without having prior knowledge of the label distribution among the records. We present an approach to effectively train a model in an unsupervised setting by leveraging advanced textual augmentation techniques and transfer learning. We will show how the use of a model pre-trained on similar data, with respect to target data, combined with different augmentation strategies can produce labels that can be effectively used to train a model on target data.

Eventually we tested the performance of the proposed approach on Imdb dataset (reference for sentiment analysis classification [42]) showing competitive results with others machine learning and deep learning approaches in a fully-supervised regime.

Chapter 1

Introduction

The massive amount of unstructured data that is available nowadays to big companies rises the issue of being able to leverage such data to extract relevant and useful information. More and more attention is posed on textual data as they represent by far the most abundant source of available data and yet understanding and producing valuable results from them represent a tough challenge.

1.1 What is Data Quality?

Information and communication technologies have extraordinarily increased the amount of information that is managed within organizational processes. It is estimated that almost 80% of world's data is unstructured [40]. In the last years has come to attention an high correlation between information and organizational processes that either consume or produce data. As a natural consequence of this, quality assumes a crucial role when it comes to data [63]. In fact, low quality of data on one hand can be an indicator of either low process quality or loose control of process performance, and on the other hand can have an impact on the organizations' ability to fulfill the needs of their customers and create value efficiently and effectively. Since data are almost ubiquitous in modern organizations, assessing and improving their quality can be complex. In the first place it should be considered, as [62] [81] points out when expressing the concept of quality as fitness for use, that data are always given with a specific context and hence its quality must always be evaluated according to expectations of users and consumers. In the second place organizational data are increasingly distributed in heterogeneous resources and represented with different formats, ranging from unstructured, to highly structured. As showed by [14] data sources are commonly classified depending on the level of structure that characterizes them. Data is distinguished between different sources in terms of:

1. Structured data, if their formal schema (i.e., formats, types, constraints, relationships) is defined and explicit.

2. Semi-structured data, if data are something in between raw data and strictly typed data, i.e., when they have some structure, but it is not as rigidly structured as in databases [1]. They are usually represented with XML markup language.

3. Unstructured data, if they are sequences of symbols, e.g., a string in natural language where no specific structure, type domains and formal constraints are defined.

Differences in the format of data are necessarily reflected in the methods and techniques that needed to assess and improve the quality of their data. The aim of this work is to focus on natural language data, by meaning with this all records containing human language data that can be analyzed and studied using all tools available in the NLP area from simple statistics to more complex language modeling.

1.2 Data Quality for Textual Data

Many data quality indicators for structured data exist (e.g., [70], [81], [4]). On the other hand, few works that present first conceptual ideas for data quality methods applied to text (e.g. [73]) can be found. The work proposed by Liu et al. [34] gives a former definition of data quality for textual data but is restricted to Judgment Documents area.

Data quality research on natural language data is still in its beginnings, but other related works can be found in the literature that may represent a good starting point for our research. In particular, works to automatically grade written student essays ([48], [52]) or Automatic Essay Scoring (AES) and posts in online discussions ([82]) have been shown as practical applications of NLP. In particular works on AES showed how an effective feature engineering and a rich text representation can lead to models trained to evaluate texts matching human hand-labeled samples with high correlation (0.76 Average Weighted Kohen Kappa Score in [52]).
Starting with these premises, it is clear that there is not a well-defined framework to work with, even thought the problem of unstructured data quality is known to the research community. Finding a way to evaluate in an effective way data that inherently have no structure and that is provided without its original context seems to pose a great challenge. The aim of this work is hence to propose a novel approach to address the 'lack of context' common in unstructured data quality settings by proposing a procedure that can be effectively applied to perform different types of classification tasks of textual data leveraging well known labeled datasets and the latest techniques in terms of language modelling ([79]), transfer learning ([17], [84]) and unsupervised data augmentation ([86]). The key concept is that, given that it is not known a priori the task that would be mostly suited to evaluate the quality of textual records, it would be appreciable if it was possible to design an approach that makes very easy to obtain high-confidence predictions on different classification task of such texts. In the following will be showed how to train a model to predict sentiment polarity on the Internet Movie Database (IMDb) [42] without using any of the label provided along with the data and with few epochs of training by leveraging transfer learning and textual data augmentation. Such approach could be extended to any classification task with the only requisite that it is available a labeled reference dataset for such task.

1.3 Contents of the Work

In the next chapters the work will be structured as follows:

- In Chapter 2 we will expose a brief overview of the main theoretical concepts employed in this research. Many of the works presented in this chapter are used or are directly related with the methodologies and the techniques used in the work. A brief overview of common approaches to data mining applied to textual data is presented followed by an in-depth illustration of most relevant concepts adopted in modern approaches. Eventually, we will present also a brief digression on knowledge distillation techniques as models obtained via this technique are employed to keep the number of parameters as small as possible.

- In Chapter 3 we will expose the methodology followed in this research and the architectural choices. We will show a general text classification pipelines and how our work relates to such framework, always focusing on the main differences and the reasons behind the implementation choices.

- In Chapter 4 we will expose the results of the experiments carried on following the proposed methodology. We will illustrate the model performances and metrics along all algorithm stages by highlighting the most relevant findings. Comparisons of the performances with different hyper-parameter settings will be showed, concluding with considerations on the obtained results and ideas for further developments.

Chapter 2

Data Mining On Textual Data

In this chapter will be introduced the main theoretical concepts taken into account while developing the proposed approach. In particular, we will start from the most relevant findings in the area of textual data quality assessment, proceeding with techniques for textual feature extraction and text classification (e.g. those proposed by Mikolov et al. [24]) ranging from simple approaches to more sophisticated techniques, including those that make use of transfer learning [84] applied in the context of semi-supervised learning [86] which will represent a starting point for the unsupervised approach proposed. Eventually, unsupervised data augmentation techniques for textual data will be showed and subsequently used in the following chapters along with other concepts exposed here.

2.1 Previous Works on Data Quality of Textual Data

Assessing the quality of text data have been a long-lasting challenge for researchers.

Readability Indexes

First attempts to represent quantitatively a quality for textual data have been done via readability measures. Readability represents the ease with which a reader can understand a written text. Such measure is somehow related to the content (the complexity of its vocabulary and syntax) and the presentation (such as typographic aspects like font size, line height, and line length) of the text itself. One of the firsts example of readability measure dates back to 1975 and is known as Flesh Readability index [29]. It tries to capture how easy and fast a human may read and understand a text. Flesch's formula is based on the number of words per sentence and the number of syllables per word. The original formulation proposed by Kincaid et al. is:

$$206.835 - 1.015\left(\frac{total\ words}{total\ sentences}\right) - 84.6\left(\frac{total\ syllables}{total\ words}\right)$$

The formula above returns a score that maps each text to an instruction grade level as in the table below. Many other indices of this kind have been

Score	Instruction Level	Notes
100.00-90.00	5th grade	Very easy to read.
90.0-80.0	6th grade	Easy to read.
80.0-70.0	7th grade	Fairly easy to read.
70.0-60.0	8th ot 9th grade	Plain English.
60.0-50.0	10th to 12th grade	Fairly difficult to read.
50.0-30.0	College	Difficult to read.
30.0-0.0	College graduate	Very difficult to read.

Table 2.1: Flesch reading-ease test results

presented during years (Dale-Chall formula [11] or Gunning fog index [21] to cite few). These readability measures by the way only capture a very limited set of text characteristics, namely the number of words, syllables and sentences.

Automatic Essay Scoring

Starting from these measures, a new research area spread in the field of NLP. The task proposed was that of assigning grades to essays written in an educational setting. Despite being applied to a very narrow and specialized area, these kind of approaches showed many different aspects that can be considered when assessing the quality of a text. Many different works have been proposed to address this task and an official competition (i.e. The Hewlett Foundation: Automated Essay Scoring [19]) hosted on Kaggle has been held. Most recent works on this field include publication from Liu et al. [35] who proposed a two-stage learning approach. The idea exposed in

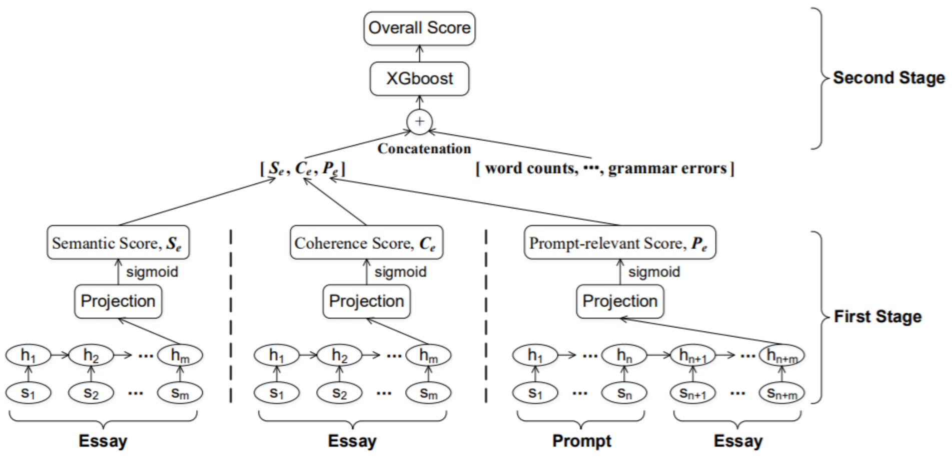


Figure 2.1: Learning stages proposed by Liu et al. [35]

this work is to leverage LSTM networks to model complex features e.g. text semantics and Coherence and then to use these features along with other (e.g. words statistics as POS-tags counts, number of grammar errors, words counts and so on), to train an XGboost model to output text scores. Specifically, in this work word embeddings are obtained via BERT models while the algorithms employed leverage hidden representations given by LSTMs networks to get scores about semantics, coherence and prompt-relevance. According to the authors of the publication, such approach revealed to be effective and outperformed all previous AES systems.
Another approach to this task was proposed by Taghipour et al. [74]. In the work published they proposed a convolutional recurrent neural network to efficiently extract representation of the text and map them to a score. Eventually, the great amount of research in this area and the increasing interest of big corporaitons and institutions (e.g. AES are effectively employed to score

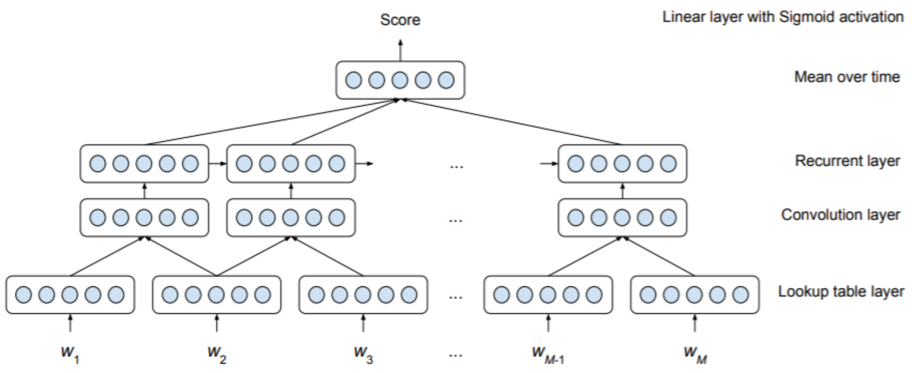
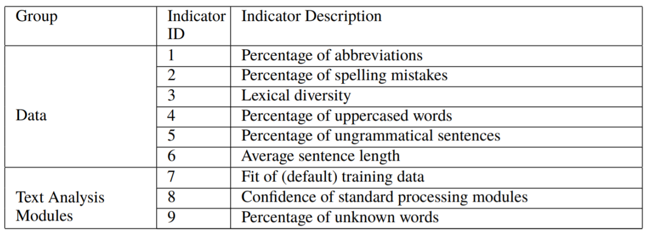


Figure 2.2: Convolutional recurrent neural network proposed by Taghipour et al. [74]

TOEFL/TOEIC written essays) lead to the development of new companies that specialized and focused on the task of correctly and reliably score texts (e.g. Educational Testing Service (ETS), founded in 1947, is the world's largest private nonprofit educational testing and assessment organization).

General Framework

In the following years many works appeared regarding Data Quality of structured data in many different domains but, as discussed before, applying such techniques to textual data showed to be a challenging task. The work presented by [27] proposes and illustrates a basic framework to analyse textual data. According to his definitions, a good quality measures considers two main components when evaluating such data: (1) the raw text itself (2) the text analysis modules. Moreover, he proposed a list of 9 data quality indexes to be evaluated. The evaluation procedures proposed by this paper, by the



Group	Indicator ID	Indicator Description
Data	1	Percentage of abbreviations
	2	Percentage of spelling mistakes
	3	Lexical diversity
	4	Percentage of uppercased words
	5	Percentage of ungrammatical sentences
	6	Average sentence length
Text Analysis Modules	7	Fit of (default) training data
	8	Confidence of standard processing modules
	9	Percentage of unknown words

Figure 2.3: Textual Data Quality metrics proposed by [27]

way, relies partially on gold labels annotated by human experts to assess the overall quality indexes and hence does not meet the requirements of our work. Other works in this area follows. Sonntag et al. [73] shows in his work an empirical approach for natural language text data, putting at the center the expectations of the consumer, human or machine. He proposed to split data quality measures into two different scores. If on one hand quality depends on the expectations of the user on the other hand it must be evaluated according to its suitability with respect to natural language processing tasks. Unfortunately, no practical implementation of such approach is given nor proposed. One of the main issue exposed by Sonnetag et al. is that:

> *"components lack methods to get at the real content of texts, especially the inferable knowledge contained in the text"*
>
> Assessing the Quality of Natural Language Text Data [73]

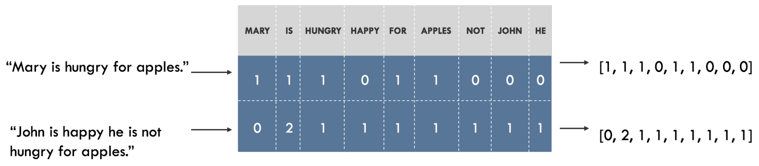
Interestingly, text quality for NLP traces back to questions of text representation.

2.2 Text Features Extraction

Leveraging textual data to extract information has always have been a challenging task for researchers as understanding natural language proved to be a tough task. The very first problem posed by such kind of data is how to represent them. On one hand all existing algorithms needs some numerical feature to work on, on the other hand textual data seemed not to be so prone to be numerically represented.

2.2.1 Bag Of Words

The bag-of-words (BOW) model is one of the earlier approaches to information retrieval in natural language processing.
In this model, a text (it can be a sentence or a document) is represented as the set of its words, disregarding grammar and even word order but keeping multiplicity. Each text is represented by a vector of the count of the



	MARY	IS	HUNGRY	HAPPY	FOR	APPLES	NOT	JOHN	HE	
"Mary is hungry for apples."	1	1	1	0	1	1	0	0	0	[1, 1, 1, 0, 1, 1, 0, 0, 0]
"John is happy he is not hungry for apples."	0	2	1	1	1	1	1	1	1	[0, 2, 1, 1, 1, 1, 1, 1, 1]

Figure 2.4: BOW representation of two sample sentences

words composing it as if the text were not much more than the set of its words. From this the name of the approach. Despite its success this model has many drawbacks as, for example, it builds a sparse representation of texts that is highly biased towards short words counts (i.e. stopwords) that frequently occur in texts but that carry very low semantic information. In addition to that, such representation completely ignores the word order and the context information carried. To remedy these shortcomings more sophisticated techniques increased the representative power of such count vectors by weighting counts.

Reducing Vocabulary Size

A first attempt to build more reliable representation of text document can be done by ignoring stopwords when building count vectors. BOW approach relies on the idea that most frequent words are also most relevant when extracting meaning from text. As pointed out in many reference manual (e.g. [33]) this approach proved to be not so effective as the most frequent

words will most surely be the common words such as "the" or "and" which help build ideas but do not carry any significance themselves. For this reason all such words (called stopwords) are removed from text before applying any algorithm for classification or feature extraction.
Another useful technique is to reduce inflected words to their base form also called stem. This procedure, also called Stemming, is known to the literature since many years (see works from Julie Lovins [38] and Martin Porter [58] who built de facto standard algorithm used for english stemming).

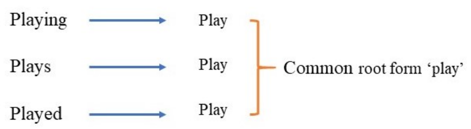


Figure 2.5: Stemming Example

Eventually, another technique commonly employed to reduce vocabulary size is lemmatisation. With this term it is usually indicated the process of determining the lemma of a word (i.e. its vocabulary form) based on its intended meaning. The main difference with stemming is that lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence and hence represents a complex task that is still an open research area (e.g. [5])

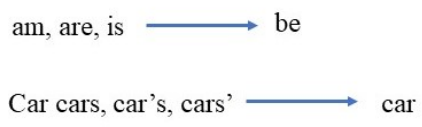


Figure 2.6: Lemmatisation Example

2.2.2 TF-IDF

When employing Term Frequency-Inverse Document Frequency approach (as [25] [30] each text is represented by the frequency of the terms appearing in input sequences, weighted by the inverse frequency of appearance of such words in all documents collection. In a more formal way we can define two terms:

- **Term Frequency**

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|}$$

where $tf_{i,j}$ stands for term frequency of term i in document j, $n_{i,j}$ for the number of times term i appears in document j and d_j for the number of terms in document j.

- **Inverse Document Frequency**

  $$idf_i = \log \frac{|D|}{\{d{:}i \in d\}}$$

  Where the numerator represents the total number of documents in the collection while the denominator represents the number of documents containing term i.

By combining these two terms we get:

$$(tf\text{-}idf)_{i,j} = tf_{i,j} * idf_i$$

This approach revealed to be very effective for a number of tasks but it also showed many drawbacks. Again, being this a "bag-of-words" representation of text, such technique does actually sees text as a set of words where the word order and co-occurrences as well as grammatical rules relating them are not relevant. To address some of these issues researchers moved towards dense representation of text called word embeddings. This term refer to a feature learning technique in which each word or phrase from the vocabulary is mapped to a N dimension vector of real numbers. Among these models Word2Vec, GloVe and fastText achieved outstanding results in the task of building models that are aware of words context when representing them.

2.2.3 Word2Vec

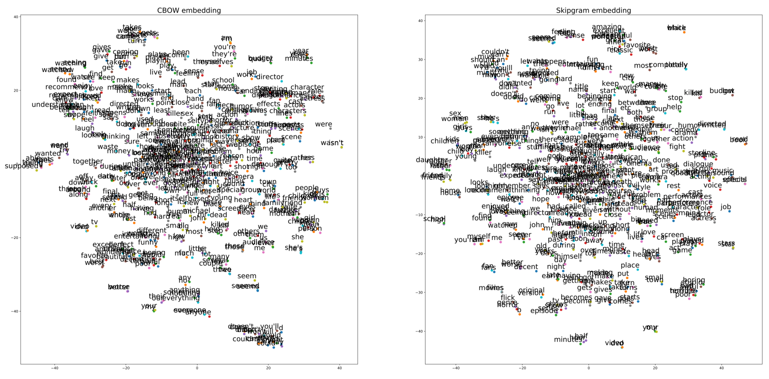


Figure 2.7: 2D-visualization for CBOW (left) and Skip-gram (right) embeddings trained on IMDB dataset.

Word2Vec [45] is a shallow three-layers neural network (it has two forms: CBOW model and Skip-gram model) with the objective of building context-aware representations of the words. More in detail a CBOW architecture

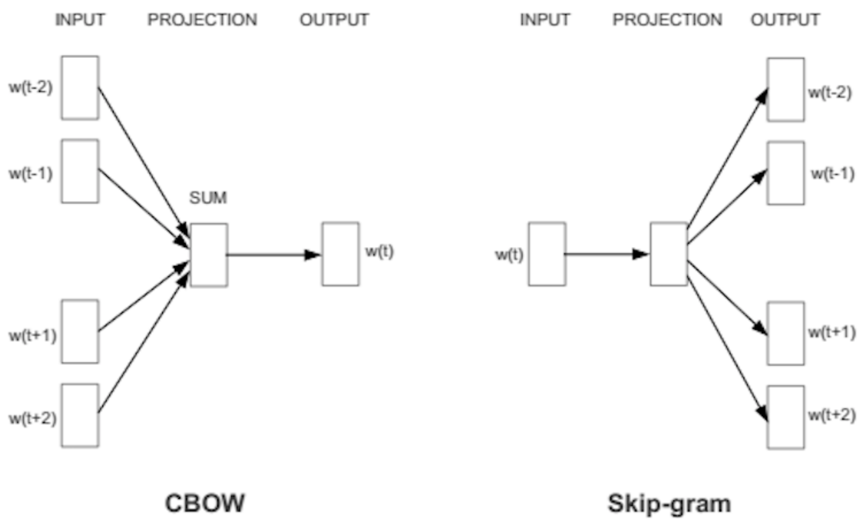


Figure 2.8: The CBOW model is trained to predict the current word given the context, while the Skip-gram predicts the context given an input word

is trained to predict a word given its context, while Skip-gram predicts the context given a word. If on one hand Skip-gram works well with a small amount of the training data (it represents well even rare words or phrases), on the other hand CBOW is several times faster to train and has slightly better accuracy for the frequent words. In both cases, Word2Vec takes as input a large corpus of text and produces a vector space with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space. Specifically, as showed by the work of Mikolov et al. [45] such representations allows for basic operations among word vectors such as computing the embedded representation $\boldsymbol{X}$ of the word "*smallest*" as the result of:

$$\boldsymbol{X} = \textit{vector}(\text{"biggest"}) \text{ - } \textit{vector}(\text{"big"}) + \textit{vector}(\text{"small"})$$

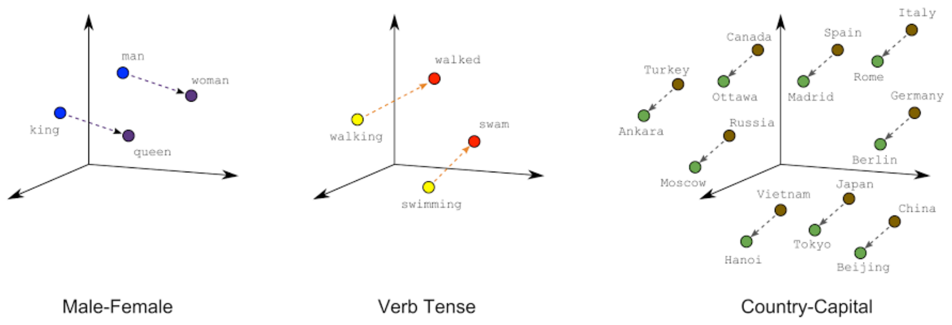


Figure 2.9: Some of the word level relationships is it possible to capture using word2vec approach

Skip-Gram

This model takes a word as input and outputs a probability distribution over the size of the vocabulary considered (V) The first issue we have to address when feeding a neural network with text is how to represent the input, given that no model can work with raw text. This step is performed via One-Hot Encoding, a procedure that maps each target word "W" to a vector of the same size V of the vocabulary and filled with zeroes except for the position corresponding to the target word "W". The dimensions of the input vector will hence be 1xV - where V is the number of words in the vocabulary. Following, the input is passed to a single dense hidden layer with dimension $V \times E$, where E is the size of the word embedding and is a hyper-parameter of the model. The output from the hidden layer would

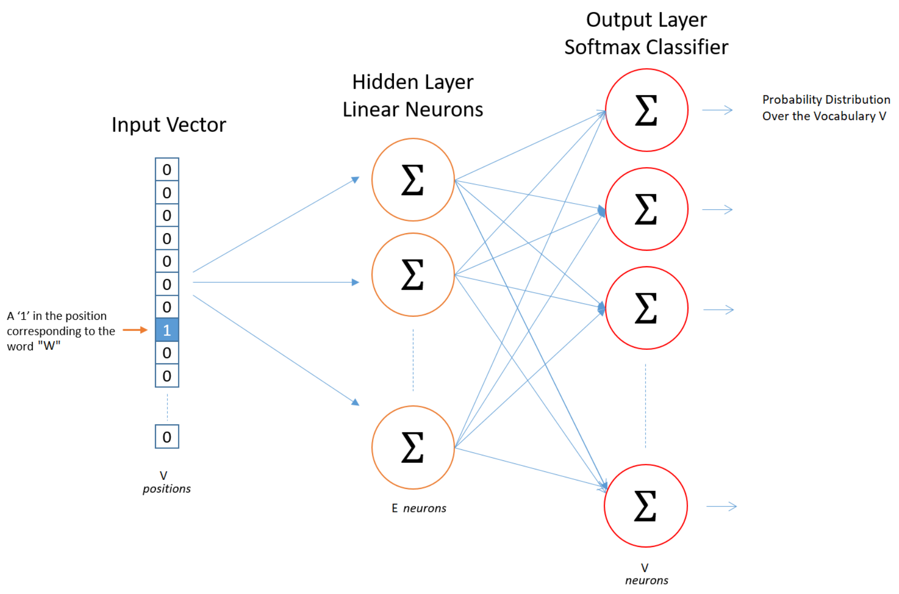


Figure 2.10: Skip-gram architecture

be of the dimension 1xE, which we will feed into an softmax layer. The dimensions of the output layer will be $1 \times V$, where each value in the vector will be the probability score of the target word at that position. The back propagation for training samples corresponding to a source word is done in one back pass. As showed in the work [46], given a sequence of words $\langle w_1, w_2, ..., w_T \rangle$ the model is trained to maximize the average log probability:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

where c is the size of the training context (is an hyper-parameter).

CBOW

CBOW architecture is somehow opposite to Skip-gram, as here we try to predict a single word given a context vector. The dimension of our hidden layer and output layer will remain the same as in Skip-gram so that we will have an hidden dimension N representing the size of the embedding of each word, and an output vector with shape $1 \times V$ representing the predicted word in the vocabulary. Only the dimension of our input layer and the calculations of hidden layer activation will change. Given C input words,

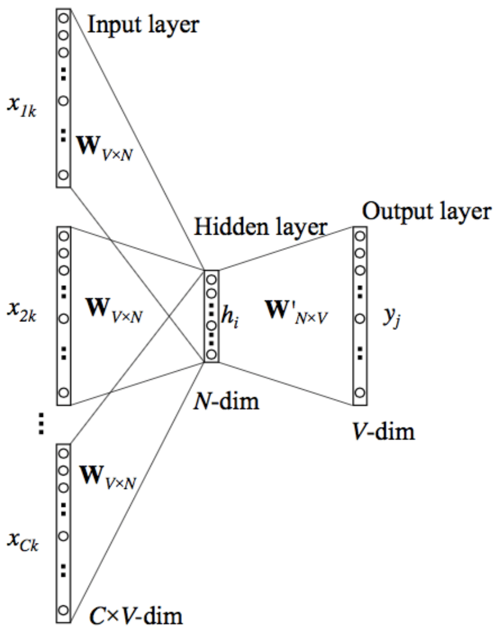


Figure 2.11: CBOW architecture

each mapped to a $1\times V$ vector, our input will be a $C\times V$ matrix. This matrix will be multiplied with $V\times N$ embedding matrix to obtain a $C\times N$ representation of the input. These vectors will then be averaged element-wise to obtain the final N-dimensional activation vector which then will be fed into the softmax layer to get our 1xV output.

2.2.4 FastText

One of the main issues of such models is related to the fact that they ignore the morphology of the words by assigning different vectors to different words. To handle this issue Facebook AI Research proposed a novel method (fastText [8]) in which sub-word level information is added to vector representations. In such context, words are considered as bags of character n-grams and then word-level embedding is obtained as the sum of sub-words representations. As an example, taking the word "where" and n=3 we would represent such word as the following char n-grams

$$\langle wh, whe, her, ere, re\rangle$$

and the special sequence

$$\langle where\rangle$$

This model allows sharing the representations across words, thus making possible to learn reliable representation for rare words. This lead to state-of-the-art accuracy in many NLP downstream tasks with very fast and reusable algorithms [24].

2.3 Transfer Learning

With the feature extraction techniques becoming more and more sophisticated, the idea was to build bigger and bigger models that were able to capture finest relationships among words by training them on enormous amount of data that internet offered with ease of access. By following this idea one would end up with a model of the language able to capture many aspect of the text that can be combined with a custom model head, e.g. for classification, and fine tuned for different downstream tasks (e.g. using discriminative learning rates in multiple stages).

2.3.1 CoVe

When modeling language, all the algorithms considered until now have the drawback of representing each word of a sentence with a vector that does not account for the context of the word itself. In such scenario the word "bank" in the sentences "bank institution" or "river bank" actually have the same representation. To address this issue, works from Peters et al. ([54]) tried to build word embeddings for adding pre-trained context embeddings from bidirectional language models to NLP systems. In a similar way, research from McCann et. al. ([43]) introduced CoVe (Context Vectors) a new type of contextualized word representation obtained extracting hidden states from a deep LSTM encoder trained in an attentional sequence-to-sequence model for machine translation (MT). Published results show that adding these context vectors (CoVe) improves performance over using only unsupervised word and character vectors on a wide variety of common NLP tasks. In both cases the key concept was to add context to words in order to better model the representation of each words based not only on the current word itself but also on the surrounding context.

2.3.2 ELMo

Soon contextualized representations of words become the main trend when it comes to word representation. A significant forward step is represented by ELMo embeddings [55] that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). In this setting word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. The main novelty of this approach was indeed in the way such representations are obtained. Peters et al. trained a bidirectional LSTM network on the next word prediction task. This is usually referred as Language Modelling and is an useful technique that can be used to train models in unsupervised settings. Being the model bidirectional, the model will perform two steps. Taken a sequence of **N** tokens $(t_1, t_2, ..., t_N)$, the model computes its inner states by taking a forward step and a backward step. In the forward step the task is to predict the probability of the next token t_k given the history

$$(t_1, t_2, ..., t_{k-1})$$

. In a formal way this is expressed as:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k|t_1, t_2, ..., t_{k-1})$$

In the same way we take the backward step. The only difference here is that it runs over the sequence in reverse and thus predicting the previous token given the future context. Again in a formal way we can define:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k|t_{k+1}, t_{k+2}, ..., t_N)$$

Once the forward and backward steps are taken the model is trained to jointly maximize the log likelihood of the forward and backward directions:

$$\sum_{k=1}^{N}(logp(t_k|t_1, t_2, ..., t_{k-1}) + logp(t_k|t_{k+1}, t_{k+2}, ..., t_N))$$

Once the model learned its parameters, the hidden states of the models itself are concatenated and then summed by linear combination with different and task-specific weights to obtain rich input representations for many different tasks.

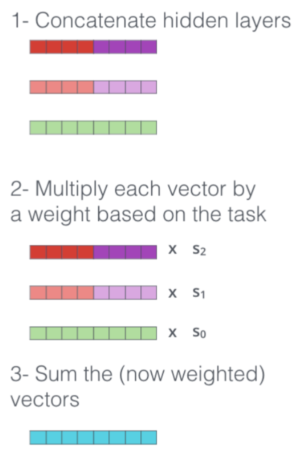

1- Concatenate hidden layers
2- Multiply each vector by a weight based on the task
x s2
x s1
x s0
3- Sum the (now weighted) vectors

Figure 2.12: Steps to go from biLSTM hidden states to ELMo embeddings

2.3.3 ULM-FiT

By following this path, text representations became richer and richer with the introduction of transfer learning techniques to train bigger models that can be effectively used on a number of downstream task. Universal Language Model Fine-tuning (ULM-FiT) [23] represents a milestone in this path because for the first time authors proposed novel techniques to avoid catastrophic forgetting [44] and to effectively

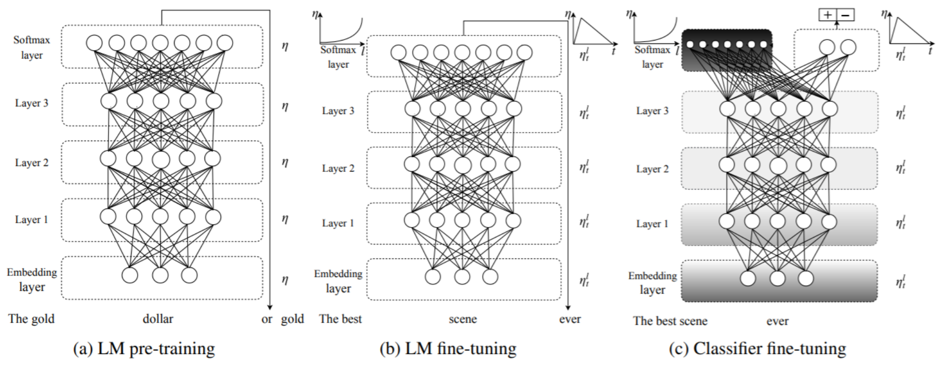


Figure 2.13: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('Discr') and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, 'Discr', and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

utilize a lot of what the model learned during pre-training. ULM-FiT introduced a language modeling approach and a process to effectively fine-tune that language model for various tasks. NLP had a way to do transfer learning as effectively as Computer Vision models could.

2.3.4 Transformer Models

In recent years the world of NLP has been taken by storm with the introduction of Transformer models and Attention [79]. Transformer architecture has

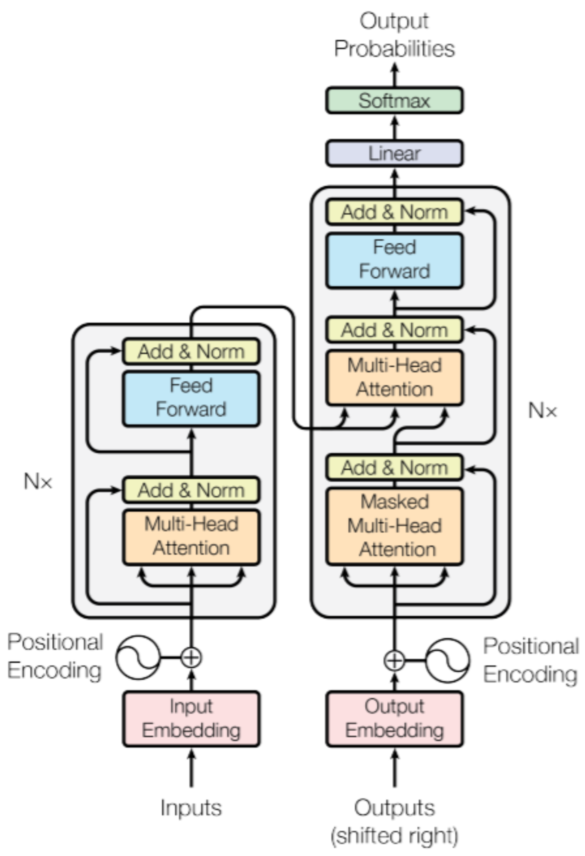


Figure 2.14: Transformer architecture from the original implementation

proven to be a disruptive approach to almost all NLP tasks. In few months, transformer models have taken the top of almost all the leaderboards in the NLP area [80].
The approach proposed by Vaswani et al. produced outstanding results as the architecture proposed was not only highly parallelizable (i.e. faster) with respect to all models with comparable performances, but also outperformed all convolutional or recurrent based model in seq2seq task like language translation. This was really because transformers deal with long-term dependencies better than LSTMs. What Vaswani et. al. proposed in his paper is an encoder-decoder model composed of stacked identical (yet they do not share weights) layers. By taking a closer look to encoder layers we can notice

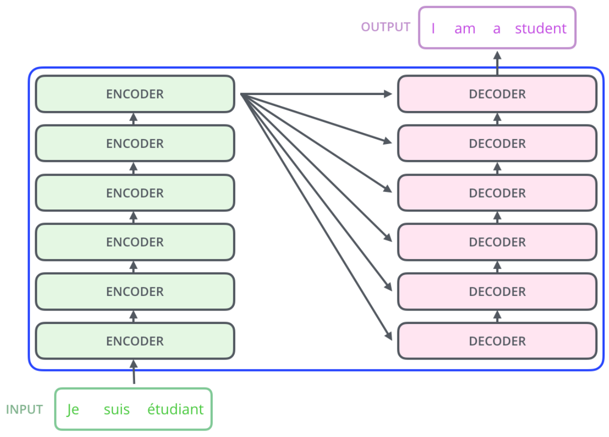


Figure 2.15: The stack of encoder and decoder layers proposed by Vaswani et al.

that each one is composed by two basic sub-blocks: a Self-Attention layer followed by a Feed Forward Layer. The output of each layer is added first to residual value for regularization, and then to the input tensor. Finally such value is normalized before being propagated to the next layer. The real

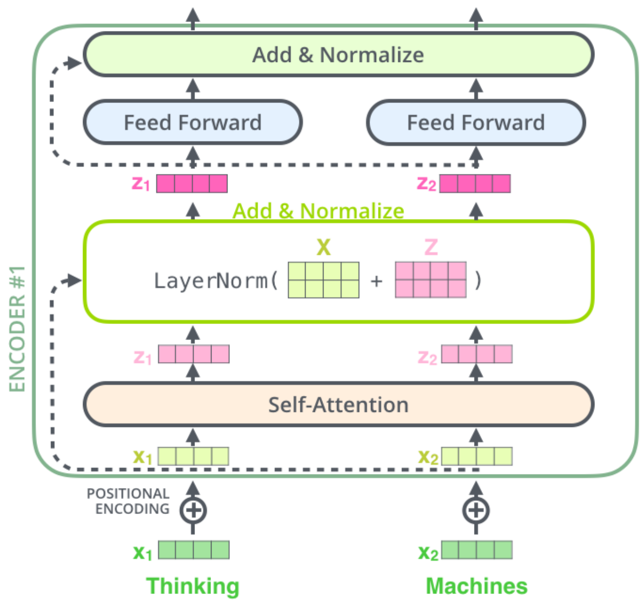


Figure 2.16: Encoder layer in detail

power of such layers is given by the representation of the input built at each step. When the attention layer is fed with input text, each word embedding is used to produce a triplet of vectors: Query, Key and Value by multiplying the embedding of each word with a matrix that is trained as a parameter of the model. These vectors represent an abstraction needed to evaluate

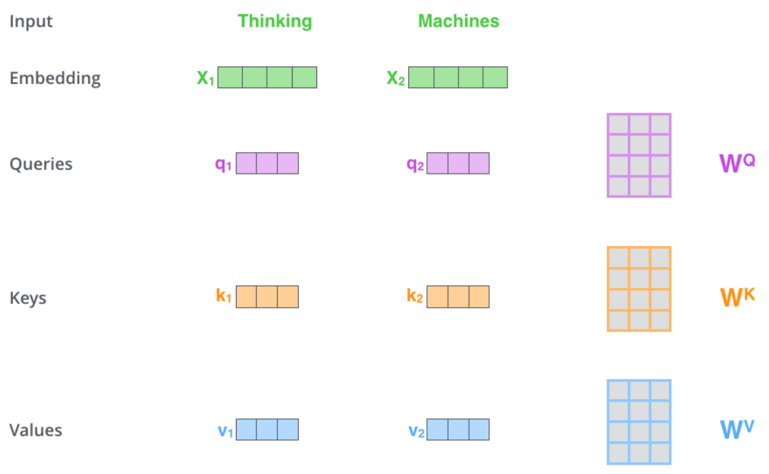


Figure 2.17: Words Embedding X_1 and X_2 are multiplied with matrices W^Q, W^K and W^V to obtain query, key and value vectors respectively

attention. Next the score of each word is evaluated by weighting its value vector with the softmax of the dot product between query and key vectors normalized for the square root of the dimension of the key vector (fixed to 64 in the paper). The intuition here is to keep intact the values of the words we want to focus on, and drown-out irrelevant words (by multiplying them by very small numbers). In a more formal way we can define:

$$Attention(Q, V, K) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where Q,V and K are the query, value and key matrices respectively and d_k is the dimension of the key vectors.
In the paper "Attention is All you Need" [79] this process is taken even further by employing "multi-headed" attention. The authors replicated the computations just shown many times giving the model multiple sets of Query/Key/Value weight matrices W^Q, W^K and W^V each of them trained as a new parameter of the model and initialized randomly. With this enhancement the model is supposed to have different "representation subspaces" of the input and to expand its ability to focus on different positions of the text. Eventually, it should be noted that all these calculations are performed in matrix form to evaluate multiple query,value and key vectors simultaneously.

One critical issue with this architecture is that the model actually sees the text as a bag of words, without accounting for the word order in the

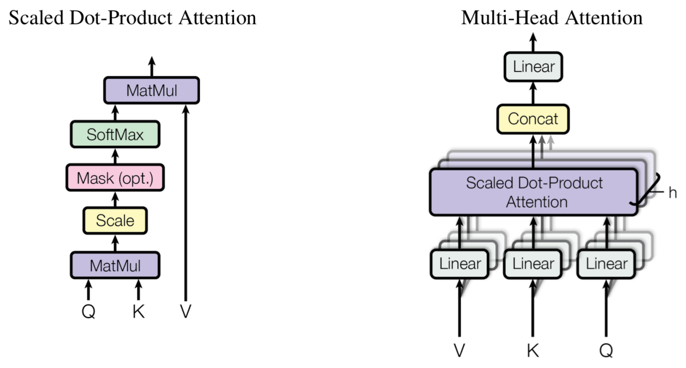
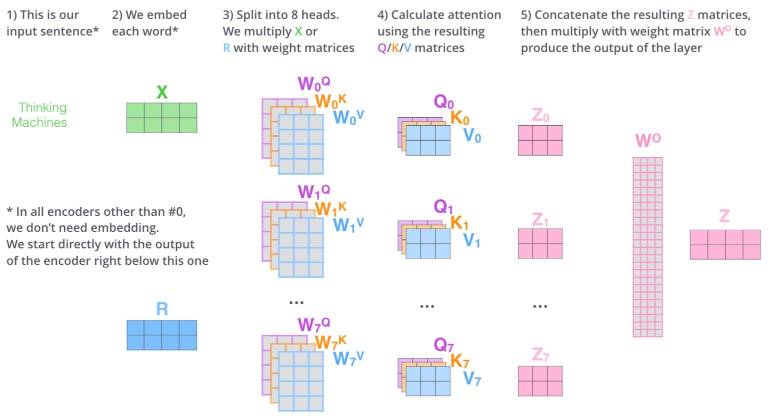


Figure 2.18: Visualization of the scaled dot-product attention (on the left) and multi-headed attention (on the right) calculations



1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
R
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z
...

Figure 2.19: Visualization of multi-headed attention calculations in matrix form

input text. To address this, a vector is added to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention. In the original implementation this is achieved via sine and cosine functions at different frequencies:

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}})$$

where pos is the position, i is the i-th dimension and d_{model} is the dimension of the model embeddings. In other words, to each dimension of the encoding corresponds a different sinusoid. The intuition was that this approach would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. If one wants to visualize the learned encoding it would be possible to color code such values. In the following the encoding of 20 words is represented. Each row represents the 512-valued positional encoding of the corresponding word. It can be noted that the encoding has a vertical split in the center.

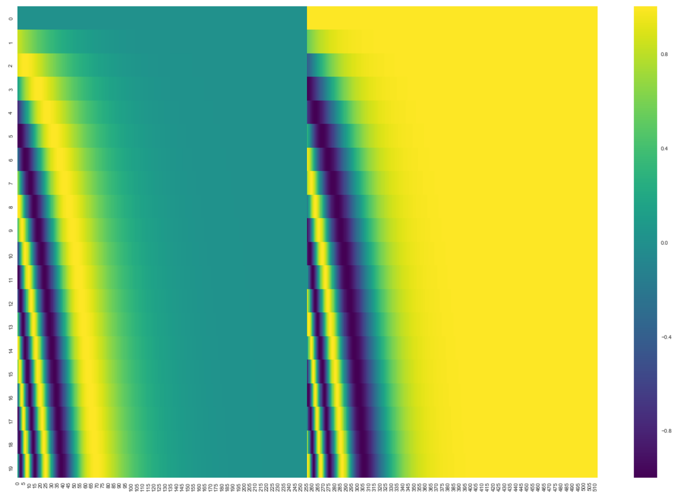


Figure 2.20: A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns)

This comes from the concatenation of two halves produced by different encodings, the leftmost is associated with a sine function and the rightmost is associated with a cosine function.
Once having all this set up, we can start the decoding step of the model. Each decoder layer receives the output matrices K and V of the last encoder in the encoder stack. These values are used in the "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence. At each time step t, the output of the decoder stack at each previous time step $t-i$ with $0 < i < t$, is used as input along with the K and V values of the encoder stack. Also here, the embedded input is enriched with positional encoding. The self attention layers in the decoder operate in a slightly different way than the one in the encoder: in this second stack, the self-attention layer is only allowed to attend to earlier positions of the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation. The "Encoder-Decoder Attention" layer works just like multi-headed self-attention, except

it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack. Eventually, the output

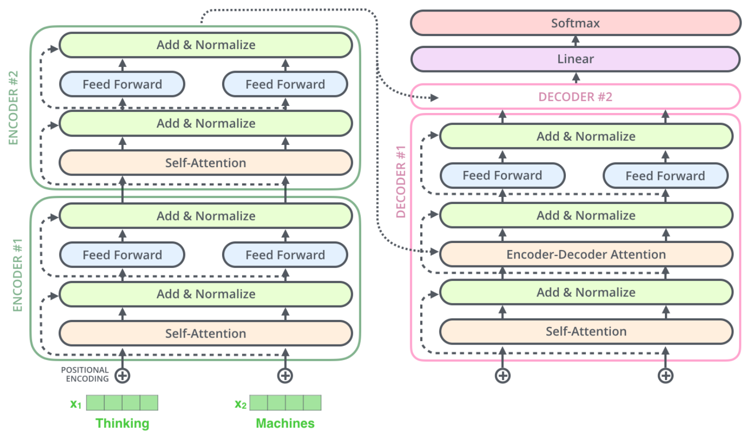


Figure 2.21: Encoder Decoder architecture in Transformers

of the decoder is fed to a feed forward layer that outputs a logits vector having the size of the vocabulary and then to a softmax layer to get token probabilities. This model was effectively employed for WMT 2014 English-to-German outperforming all previous models. The architecture presented was the first able to model complex language relations without using recursive or convolutional layers in its computations. Moreover it proved to be extremely effective on seq2seq task by achieving state-of-the-art results in machine translation task and many others.

2.4 Prediction Model

Once text samples are mapped into a feature vector we will need a model to learn patterns among such features in order to accurately classify our data. The models exposed below belongs to the family of supervised algorithms hence are are trained to minimize a loss function relating model's output to input labels. We will pay particular attention to the models that leverage transfer-learning as this plays a key aspect to exploit powerful representations built with techniques previously exposed. Moreover, using pre-trained models and then fine-tuning for downstream task allows to build accurate models with few training epochs (i.e. decreasing time-requirements) allowing analysis on large scale datasets for different sub-tasks in few hours. Most pre-trained models comes from Google Official repository, and training techniques are documented and exposed in the publications relative to each model as we will show shortly.

2.4.1 BERT

Among all transformers-based models, BERT [17] (Bidirectional Encoder Representation from Transformers) succeeded in the task of producing transformer's representation of words while taking into account right and left conditioning (i.e. by taking into account surrounding context). Since BERT's goal is to generate a language model, only the encoder mechanism from transformers is necessary. In the original implementation from [17] the model was trained to perform two unsupervised tasks. The first one is Masked Language Modeling (MLM). When feeding BERT's encoder stack, 15% of the words in the input sequence are selected for the prediction. For each token i among those selected, the replacement policy works as follows:

- The i-th item is replaced with the token [MASK] with a probability of 0.8
- The i-th item is replaced with a random word from the vocabulary with a probability of 0.1
- The i-th item is replaced with the original token with a probability of 0.1

. The purpose of this masking strategy is to reduce the mismatch between pre-training and subsequent fine-tuning, as the [MASK] token would never appear during the fine-tuning stage. The model then attempts to predict the original value of the masked words using standard cross-entropy and based on the context provided by the other, non-masked, words in the sequence. A drawback of this approach is that the model converges slower then classical language modeling, but this technique still leads to better performances in few epochs of training as showed by [17]. The second one is Next Sentence

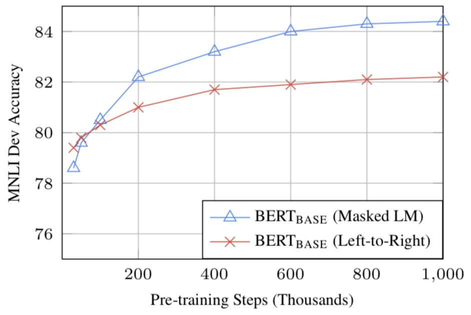
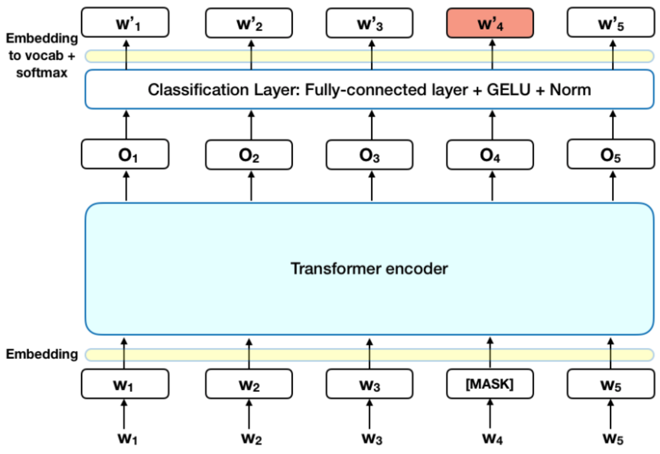


Figure 2.22: Test on Multi-Genre Natural Language Inference task [83]

Prediction. Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, the authors pre-trained the model for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext) [17]. In order to perform prediction for both of the task, on top of the encoder stack it is additionally required:

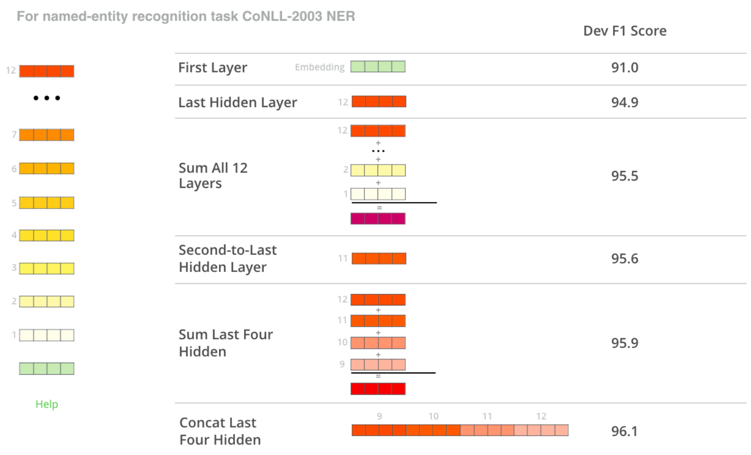


Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
Transformer encoder
Embedding
[MASK]

Figure 2.23: BERT architecture

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with softmax

Moreover, to make BERT handle a variety of down-stream tasks, input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., ⟨ Question, Answer ⟩) in one sequence. A ”sequence” refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together while a ”sentence” can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. In the original work, embeddings are produced via WordPiece embeddings [85] which has a vocabulary of 30,000 token. The first token of every sequence is always set to be a special classification token ([CLS]). This token aggregates the sequence representation for the classification task. Moreover, as sentence pairs are packed together into a single sequence, another special token ([SEP]) is be added to differentiate the two sentences. It is important to notice that this is the first attempt to build a truly bidirectional language model, as for the first time this model was able to take a whole sentence as input and compute its representations for each of such sequences at once, in contrast with previous approaches where the bidirectional conditioning was obtained combining a left-to-right and a right-to-left context (as in ELMo's bidirectional LSTMs). Moreover, it should be noticed, as the authors of the



	Dev F1 Score
First Layer	91.0
Last Hidden Layer	94.9
Sum All 12 Layers	95.5
Second-to-Last Hidden Layer	95.6
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1

Figure 2.24: Different approaches to extract embeddings from BERT's encoders stack

original work [17] point out, that once the model is trained, not only it can be used in a fine-tuning fashion, but also to extract input representations from the encoder stack. Specifically, combinations of the embeddings at different encoding layers (second-to-last hidden, last Hidden, weighted sum of the last four hidden, concatenation of the last four hidden, weighted sum of all 12 layers) showed different F1-scores for NER task on CoNLL-2003 dataset [77]. The hidden representations of the model are again used to build vectors to represent the input. The model can hence be used to extract contextualized representations of the input words. Such contextualization spans along all the layers of the encoder stack and each layer is able to capture a different aspect of the input text. As said, different layers, or combinations of them, can lead to better performance depending on the task. The main drawback of the proposed model lies in the hardware requirements as the implementation proposed is not intended for CPU-only systems but relies on parallelizable GPUs to keep execution time low. To partially address this issue, we decided to explore techniques to reduce model's number of parameters while preserving performances.

2.4.2 DistilBert

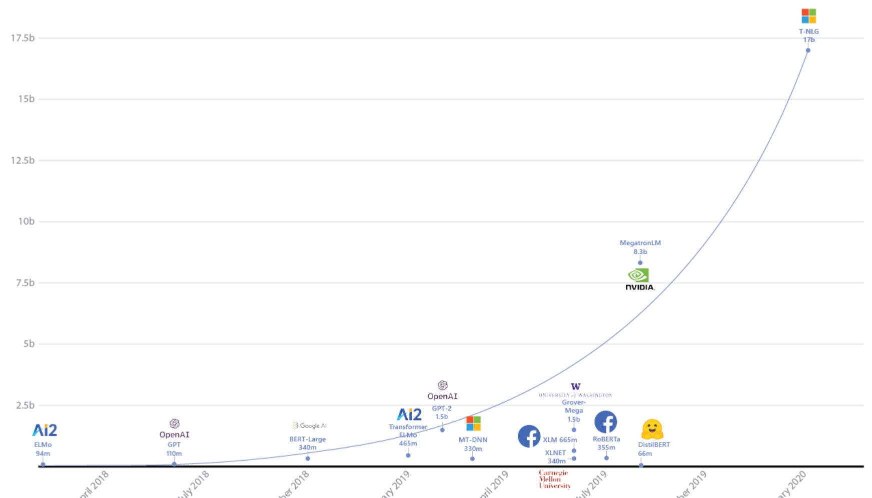


Figure 2.25: Comparison of number of parameters of different language models proposed in recent years

When it comes to language models, research trends in the last few years show clearly that there is a strong correlation between model size, available data and performance obtained. It actually seems that bigger and bigger models in terms of parameter (e.g. Turing Natural Language Generation or

T-NLG [65] which is a 17 billion parameter language model by Microsoft) trained on more and more data (e.g. RoBERTa model from Facebook AI [36] was trained on 160GB corpus) leads to better performances. In the opposite sense, research from Sanh et al. [68] aims to produce smaller transformer models with comparable performance with regard to their bigger versions. Knowledge distillation is not a model compression technique, but it has the same goals and effects. After training a big and slow model (the teacher), a smaller model (the student) is trained to mimic the teacher's behaviour - whether its outputs or its internal data representations. This leads to very straightforward improvements in both speed and size across different types of networks, from CNNs [64] to LSTMs [28]. Very interesting is the effect of teacher-student architectural differences: while [64] recommend students deeper and thinner than teachers, [39] and [49] present tricks for better knowledge transfer between very different architectures. In [75], the authors successfully learn a small biLSTM student from BERT. This process was

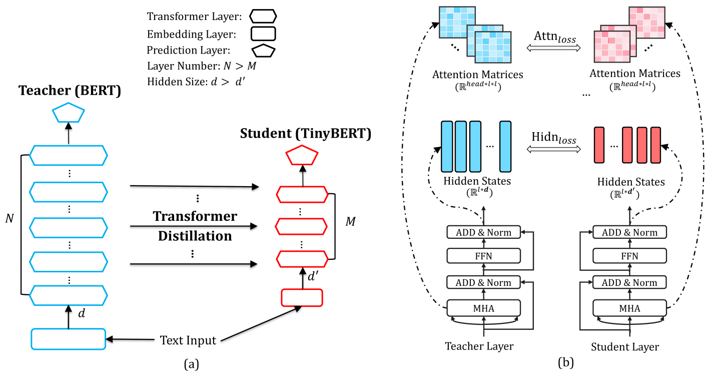
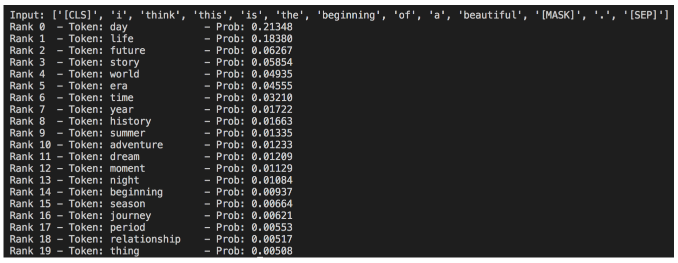


Figure 2.26: Distillation Process for BERT

introduced by Bucila et al. [10] and further analyzed by Hinton et al. [22] a few years later. In [68] authors followed the latter method. In supervised learning, a classification model is generally trained to predict a gold class by maximizing its probability (softmax of logits) using the log-likelihood signal. In many cases, a good performance model will predict an output distribution with the correct class having a high probability, leaving other classes with probabilities near zero. But, some of these "almost-zero" probabilities are larger than the others, and this reflects, in part, the generalization capabilities of the model. This can be easily visualized when evaluating token probabilities in language models. Here follows an example. We can clearly see how the model predicts two tokens with an high probability (i.e. 'day' and 'life') followed by a long tail of low-probability tokens. In a common



```
Input: ['[CLS]', 'i', 'think', 'this', 'is', 'the', 'beginning', 'of', 'a', 'beautiful', '[MASK]', '.', '[SEP]']
Rank 0  - Token: day           - Prob: 0.21348
Rank 1  - Token: life          - Prob: 0.18380
Rank 2  - Token: future        - Prob: 0.06267
Rank 3  - Token: story         - Prob: 0.05854
Rank 4  - Token: world         - Prob: 0.04935
Rank 5  - Token: era           - Prob: 0.04555
Rank 6  - Token: time          - Prob: 0.03210
Rank 7  - Token: year          - Prob: 0.01722
Rank 8  - Token: history       - Prob: 0.01663
Rank 9  - Token: summer        - Prob: 0.01335
Rank 10 - Token: adventure     - Prob: 0.01233
Rank 11 - Token: dream         - Prob: 0.01209
Rank 12 - Token: moment        - Prob: 0.01129
Rank 13 - Token: night         - Prob: 0.01084
Rank 14 - Token: beginning     - Prob: 0.00937
Rank 15 - Token: season        - Prob: 0.00664
Rank 16 - Token: journey       - Prob: 0.00621
Rank 17 - Token: period        - Prob: 0.00553
Rank 18 - Token: relationship  - Prob: 0.00517
Rank 19 - Token: thing         - Prob: 0.00508
```

Figure 2.27: Token predicted by BERT model to complete the famous quote from Casablanca film.

teacher-student training hence, we train a student network to mimic the full output distribution of the teacher network (its knowledge). The idea is training the student to generalize the same way as the teacher by matching the output distribution. To achieve this, rather than training with a cross-entropy over the true hard targets (one-hot encoding of the gold class), we transfer the knowledge from the teacher to the student with a cross-entropy over the soft targets (probability distribution over classes of the teacher). Our training loss function, according to [22], should hence have the following structure:

$$L = -\sum_i t_i \times \log(s_i)$$

where t_i are logits from the teacher and s_i those of the student In the same work Hinton et al. also introduced softmax temperature function:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

where z_i are model's output logits and T is the Temperature. When $T \to 0$, the distribution becomes a Kronecker (and is equivalent to the one-hot target vector), when $T \to +\infty$, it becomes a uniform distribution. The same temperature parameter is applied both to the student and the teacher at training time, further revealing more signals for each training example. At inference, T is set to 1 and recover the standard Softmax. In our setting, we will not consider softmax temperature function but we will employ, accordingly to [22], Kullback-Leibler function as loss over soft labels. Following

$$KL(p||q) = \mathbb{E}_p(log(\frac{p}{q})) = \sum_i p_i * log(p_i) - \sum_i p_i * log(q_i)$$

Hinton et al., the training loss is a linear combination of the distillation loss and the masked language modeling loss. The resulting student network is a small version of BERT in which the token-type embeddings and the pooler (used for the next sentence classification task) are removed, while preserving the rest of the architecture identical. This process reduces the number of layers by a factor of two. Overall, the distilled version of BERT model, DistilBERT, has about half the total number of parameters of BERT base but retains 95% of BERT's performances on the language understanding benchmark GLUE. Moreover, to boost performances, a few training tricks from the recent RoBERTa paper [36] (which showed that the way BERT is trained is crucial for its final performance) are performed. For example, following [36], DistilBERT was trained on very large batches leveraging gradient accumulation (up to 4000 examples per batch), with dynamic masking but without the next sentence prediction objective.

2.5 Unsupervised Data Augmentation for Consistency Training

Recent advances in semi-supervised learning allows to train model in very low-label regime. Work from Xie et al. [86] showed that BERT model can be trained with as much as 20 labels to achieve state-of-the-art results compared with models trained in a fully-supervised regime (25,000 labels). These achievements shows room for further investigations to evaluate how well this kind of approaches perform when the labeled data comes from a teacher model. Among all semi-supervised solutions, those employing consistency training framework ([3], [61], [76]) produced good results on many benchmarks. Specifically, the work from Xie et al. [86] shows how to effectively noise unlabeled examples, pointing out that the quality of data augmentation techniques plays a crucial role in the learning stage of the model. As discussed in [86], a focus on the methods employed to augment data can give many extra advantages with respect to simple approaches under many aspects:

- Advanced data augmentation methods that achieve great performance in supervised learning usually generate realistic augmented examples that share the same ground-truth labels with the original example. Thus, it is safe to encourage the consistency between predictions on the original unlabeled example and the augmented unlabeled examples. (valid noise)
- Advanced data augmentation can generate a diverse set of examples since it can make large modifications to the input example without changing its label, while simple Gaussian noise only make local changes. Encouraging consistency on a diverse set of augmented examples can significantly improve the sample efficiency. (diverse noise)
- Different tasks require different inductive biases. Data augmentation operations that work well in supervised training essentially provides the missing inductive biases (targeted inductive biases).

Results showed in this work are a promising starting point for the purpose of this research as combines many concepts involved in the methodology proposed showing that transfer learning of transformer's language models can be effectively employed in a low-label regime using consistency training. As discussed before, recent works in semi-supervised learning make use of unlabeled examples to enforce smoothness of the model. The general form of these approaches can be summarized in two main steps:

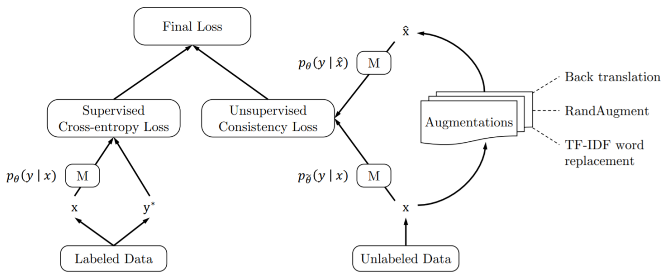


Figure 2.28: Overview of UDA [86] training objective

- Given an input x, compute the output distribution $p_\theta(y|x)$ given x and a noised version $p_\theta(y|x, \epsilon)$ by injecting a small noise ϵ. The noise can be applied to x or hidden states.
- Minimize a divergence metric between the two distributions $\mathcal{D}(p_\theta(y|x)||p_\theta(y|x, \epsilon))$.

The idea underlying consistency training is that model predictions can be regularized to be invariant to noise applied to the input ([50], [67], [15]) or to the hidden states ([3], [31]). This intuitively makes sense because a model should be invariant in its prediction to small perturbations in the input example or in its hidden states. This procedure enforces the model to be less sensible to noise ϵ and hence smoother with respect to changes in the input space. Said in other words, minimizing the consistency loss gradually helps propagating prediction confidence from labeled examples to unlabeled ones. What the authors of [86] noticed was that the "quality" of noising operations on the input can largely influence the performance of the consistency training framework. They showed that stronger data augmentations techniques, in contrast with simple Gaussian noise and trivial input augmentations, can also lead to better performance when used to noise unlabeled examples in the semi-supervised consistency training framework, since it has been shown that more advanced data augmentations that are more diverse and natural can lead to significant performance gain in the supervised setting. This opens to the use of large amounts of new samples built to let the model be more confident in its predictions. To account for such samples in the learning stage, the loss functions has to consider two components. The first one is supervised cross entropy, while the second one is KL divergence. The relative magnitude of the two is set using the hyper-parameter λ. Eventually, the training objective function was set to:

$$\min_\theta \mathcal{J}(\theta) = \mathbb{E}_{x,y^*\in L}[-\log(p_\theta(y^*|x)] + \lambda \mathbb{E}_{x\in U}\mathbb{E}_{\hat{x}\sim q(\hat{x}|x)}[\mathcal{D}_{KL}(p_{\hat{\theta}}(y|x)||p_\theta(y|\hat{x}))]$$

where $q(\hat{x}|x)$ is a data augmentation transformation, L and U represent Labeled and Unlabeled sets respectively, $\mathcal{D}_{KL}$ refers to Kullback-Leibler divergence and $\hat{\theta}$ is a fixed copy of the current parameters θ indicating that the gradient is not propagated through $\hat{\theta}$, as suggested by [50]. As already highlighted, another key aspect, other than the objective function, is the strategy used to augment data. The strategy proposed in [86] for textual data augmentation are:

- **Back-Translation** ([71],[18]) refers to the procedure of translating an

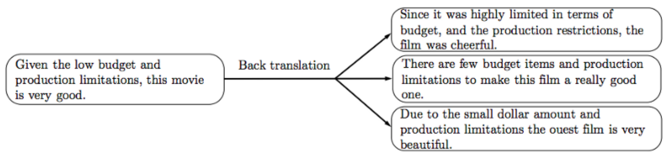

Given the low budget and production limitations, this movie is very good.
Back translation
Since it was highly limited in terms of budget, and the production restrictions, the film was cheerful.
There are few budget items and production limitations to make this film a really good one.
Due to the small dollar amount and production limitations the ouest film is very beautiful.

Figure 2.29: Back-Transaltion Example

  existing example x in language A into another language B and then translating it back into A to obtain an augmented example $\hat{x}$ (as observed by [87], back-translation can generate diverse paraphrases while preserving the semantics of the original sentences leading to significant performance improvements).

- **TF-IDF based word replacement** that allows to have more control on the retained words at the cost of loss in the overall semantic of the sentence. Such aspect is crucial when approaching tasks, e.g. topic classification, in which some keywords are more informative than other and hence retaining some specific words is relevant to preserve fundamental concepts of the sentence. In the proposed implementation words are replaced on the base of their tf-idf score, which is used as an "informativeness" metric of words inside the text.

Both these techniques show a great capability of building sentences that relates semantically to the original one by still differing in some structural or syntactical aspect. By taking these two techniques as a starting point we will show how the use of other approaches can be beneficial to the model learning stage. This is because different techniques produces different perturbations of the same input and help the model learn to discern among classes. Other examples of effective use of back-translation as augmentation technique can be found also in [16]. In this work several approaches to augment text are compared keeping the attention on their scalability. In the experiment proposed in such work, besides back-translation, other techniques to generate paraphrase or to noise input are showed. In all the experiments reported, models trained using an high number of noising methods appeared to perform better than their counterparts fed with a smaller number of augmentation strategies.

2.6 Summary

In this chapter we showed the main theoretical concepts that will be involved in our experiments. Starting from previous works on natural language quality assessment we over-viewed the main aspects that plays crucial role when measuring data quality for such data. We then moved to expose main concepts that comes useful when building rich text representations that allows models to learn robust function mapping samples to correct labels. Eventually we focused on Transformer Models, a real breakthrough in the field of NLP, both for their architecture lacking of any convolutional or recurrent layer and for the ease they can be used in many down-stream tasks.

Chapter 3

Proposed Methodology

According to Google developers best practices, the above steps should be

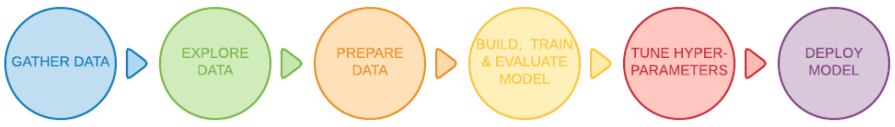
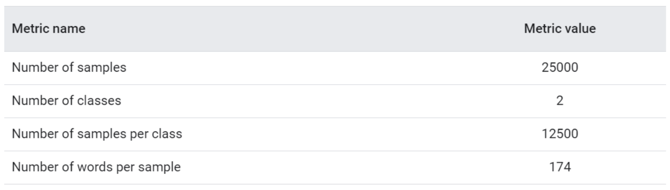


Figure 3.1: Text Classification Workflow proposed on Google Developers Website

followed when classifying text data. The first data gathering step,as will be better explained in the following pages, is crucial for the applicability of the proposed methodology. The data exploration step is then fundamental to assess the effectiveness of collected data. The first and easiest approach we can use is manual inspection. We can pick randomly some sentence from our set and verify that is consistent with our expectation. Once inspected the data, it turns useful to collect the following important metrics that can help characterize the current text classification problem:

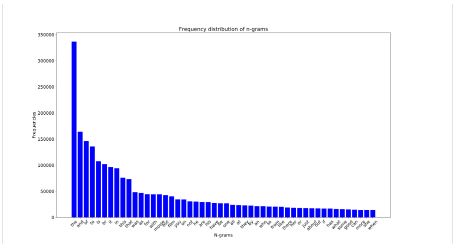
- Number of samples: Total number of examples you have in the data.
- Number of classes: Total number of topics or categories in the data.
- Number of samples per class: Number of samples per class (topic/category). In a balanced dataset, all classes will have a similar number of samples; in an imbalanced dataset, the number of samples in each class will vary widely.
- Number of words per sample: Median number of words in one sample.
- Frequency distribution of words: Distribution showing the frequency (number of occurrences) of each word in the dataset.

- Distribution of sample length: Distribution showing the number of words per sample in the dataset.

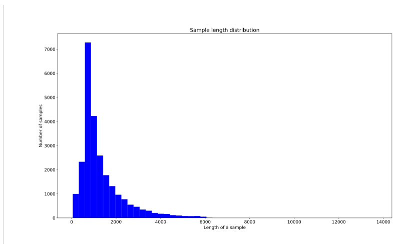


Metric name	Metric value
Number of samples	25000
Number of classes	2
Number of samples per class	12500
Number of words per sample	174

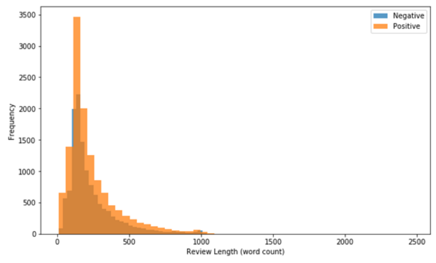
Figure 3.2: Metrics Measures From IMDb data



(a) Word Counts



(b) Sentence Length

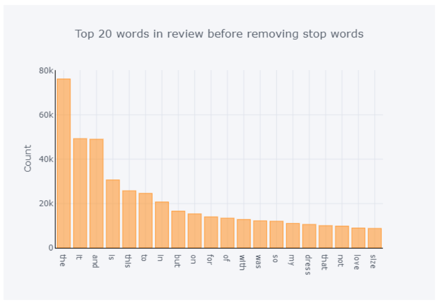


(c) Sentence Length Dived By Class Label

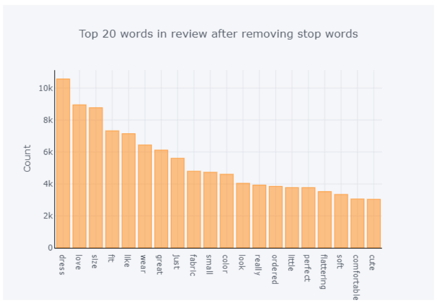
Figure 3.3: Simple Counts over IMDb Data

For further inspection we may evaluate textual heterogeneity. For this purpose it may be useful to evaluate n-grams counts and average sentence length as showed below.

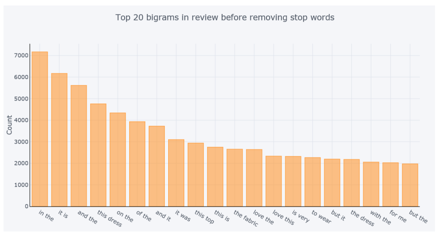
Once data we will be working on is fully explored and its main features and characteristics are outlined, next step is to select a model to correctly and effectively extract features from text starting from most relevant features highlighted from analysis. To this purpose the following flowchart comes in help when approaching a text classification task. This flowchart answers two key questions: (1) Which learning algorithm better fits our data (2) How should data be preprocessed to efficiently learn the relationship between samples and assigned label.



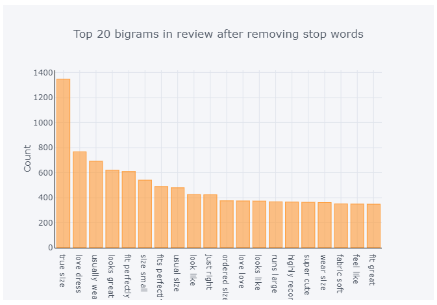
(a) Top 20 Unigrams Counts Before Removing Stopwords



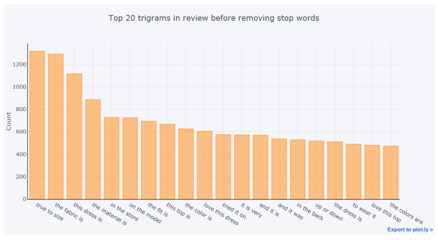
(b) Top 20 Unigrams Counts After Removing Stopwords



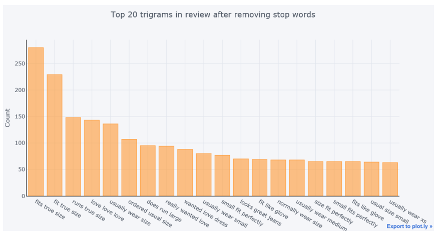
(c) Top 20 Bigrams Counts Before Removing Stopwords



(d) Top 20 Bigrams Counts After Removing Stopwords



(e) Top 20 Trigrams Counts Before Removing Stopwords



(f) Top 20 Trigrams Counts After Removing Stopwords

Figure 3.4: Stats on IMDb Records

These two questions are clearly correlated; the way we preprocess data to be fed into a model will depend on what model we choose. Models here are broadly divided into two categories: those that use word ordering information (sequence models), and ones that employ a bag of words approach (n-gram models). Types of sequence models include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Transformer Models. Types of n-gram models include logistic regression, simple multi- layer perceptrons (MLPs, or fully-connected neural networks), gradient boosted trees and support vector machines. As exposed in Machine

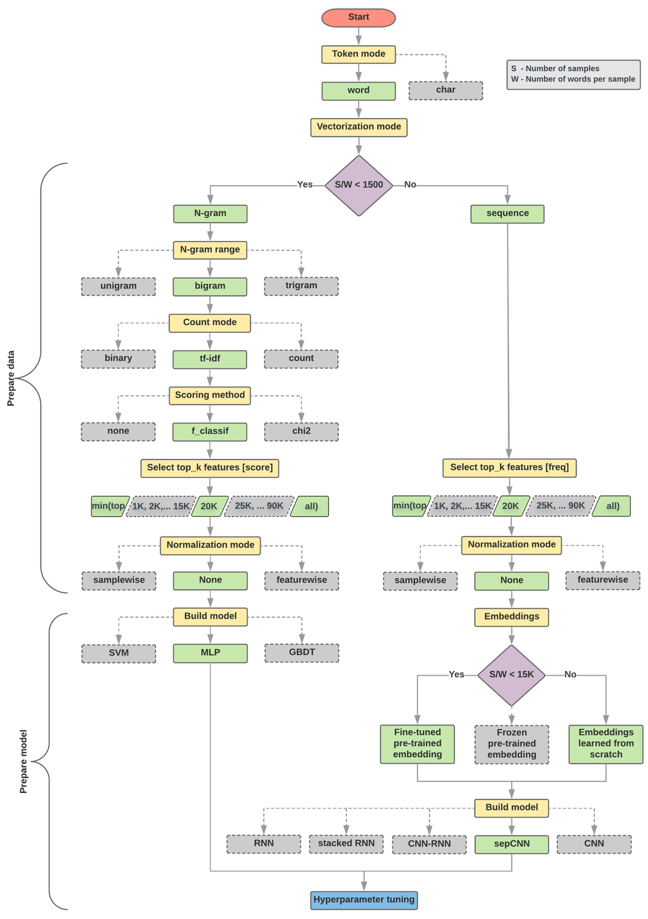


Figure 3.5: Text classification flowchart proposed by Google Developers blog

Learning Guides for Text Classification from Google, the ratio of "number of samples" (S) to "number of words per sample" (W) correlates with which model performs well. When the value for this ratio is small (< 1500), small multi-layer perceptrons that take n-grams as input perform better or at least as well as sequence models. MLPs are simple to define and understand, and they take much less compute time than sequence models. When the value for this ratio is large (≥ 1500), use a sequence model should be preferred. In the case of our IMDb review dataset, the samples/words-per-sample ratio is ~ 144 hence suggesting a simple MLP model. We will, by the way, recover to a Transformer Model as SOTA results obtained by employing this solution for many different tasks suggest that transformers are able to build a powerful representation of input data and to obtain high performance on heterogeneous natural language data sources.

3.1 Proposed Pipeline

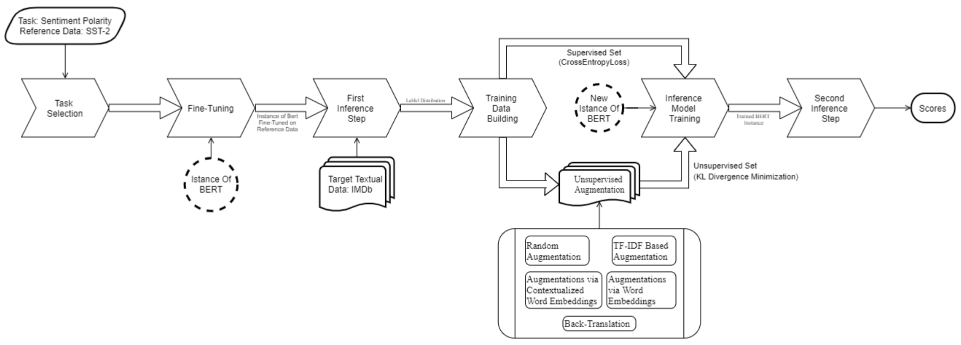

Task: Sentiment Polarity
Reference Data: SST-2
Task Selection
Fine-Tuning
Istance Of BERT
Instance of Bert Fine-Tuned on Reference Data
First Inference Step
Target Textual Data: IMDb
Lablel Distribution
Training Data Building
Supervised Set (CrossEntropyLoss)
New Istance Of BERT
Inference Model Training
Unsupervised Augmentation
Unsupervised Set (KL Divergence Minimization)
Random Augmentation
TF-IDF Based Augmentation
Augmentations via Contextualized Word Embeddings
Augmentations via Word Embeddings
Back-Translation
Trained BERT Instance
Second Inference Step
Scores

Figure 3.6: Overview Of Proposed Methodology

All the concepts introduced in chapter 2 converges in the proposed approach exposed in this section. As previously introduced the aim of the work is to develop a framework to score text with different possible task (e.g. sentiment polarity or linguistic acceptability, among many). Such approach rises from the necessities pointed out from previous works that repeatedly underline that quality in textual context is much a matter of what quality means to the user. Given that the end classification task is not known a priori, we should include the task itself as an input variable of the pipeline proposed. The framework hence should be capable of producing different labels for the same target data, depending on the chosen task, and accordingly train a given model to evaluate such data with the produced labels. This is achieved by leveraging knowledge extracted from other labeled dataset. Taken sentiment polarity as task, a standard reference dataset can be Stanford Sentiment Treebank [72]. Models like BERT can be used jointly with transfer learning to obtain instances pre-trained on huge corpora to unsupervisedly capture words relationship and sentence ordering (see chapter 2 for further details). This gives the model generalizing capabilities for many downstream tasks. In the following we will always make use of pre-trained instances unless differently specified. More in detail we will employ two instances of distilBert (66M parameters) both pre-trained on lower cased english text. The first one will be fine tuned on reference data (i.e. SST-2) to learn its parameters for the target task. The second one will be trained both supervisedly on the most-confident labels produced by the first one and unsupervisedly on the augmented remaining data to enforce consistency in its predictions.
As showed above, the algorithm spans for 6 different stages:

1. **Task Selection** Select the classification task to perform on target data and a reference dataset. As said, in our experiments "Sentiment Polarity" will be used as classification task and the Stanford Sentiment Treebank as reference data.

2. **Fine-Tuning** Fine-tune on reference data an instance of distiBert model.

3. **First Inference Step** Use the obtained model to perform inference on target data. In our scenario target data will be IMDb records.

4. **Training Data Building** Split target data in:

    - Labeled Set: Samples with a predicted probability higher than a given threshold k (in our experiments k is set to the 85^{th} percentile of the prediction probabilities on target data)
    - Unlabeled Set: All the other samples from target data, augmented n times (in our experiments n is set to 5) using different unsupervised augmentation strategies sampled randomly from a set of several techniques (random word swapping, random word deletion, TF-IDF word insertion, TF-IDF word substitution, word substitution via GloVe contextualized embeddings, word insertion via GloVe contextualized embeddings, word substitution via XLNet contextualized embeddings, word insertion via XLNet contextualized embeddings, back-translation)

5. **Inference Model Training** Train a new instance of DistilBertForSequenceClassification model on the newly built dataset using:

    - Standard CrossEntropy on Labeled Set
    - Minimization of the KL divergence between Softmax(O_1) and Softmax(O_2) where O_1 represents the logits of the fine-tuned model on the Unlabeled Set and O_2 those of the model we are training

6. **Second Inference Step** Once the model is trained use the confidence on predictions as a score for the given task.

We will now see details of each step to understand the relevance of each proposed step in the given pipeline.

3.2 Task Selection

The first step is to select the task we want to perform on target data. As already discussed before this step is crucial as it also defines the way the quality of the textual records will be evaluated. In our experiments the selected task is binary sentiment classification. This is a standard classification task included in many different benchmarks and evaluation tools. Once the task is selected a reference labeled dataset must be found. For our purposes, we selected the Stanford Sentiment Treebank (SST-2) as this dataset is often referenced in the literature as a standard for binary sentiment analysis and is also included in the GLUE benchmark [80]. In the context of our work we will show only models handling one sentence at time, by the way, all used model can be extended to the case of pair of sentences (e.g. question answering).

3.3 Fine-Tuning

The model used for fine-tuning is, as said, an instance of distilBert. Although a larger version of the model may lead to an increase of accuracy, the choice was driven by performance in terms of total execution time and hardware requirements. This step can be performed in two ways to optimize execution time. (1) By leveraging the increasing amount of fine-tuned checkpoints that are open-sourced it can be easy to find the needed instance ready to import. This is often the case of reference task e.g. the one considered in out our. A version of the model pretrained on SST-2 is available in the reference repository used for the project (i.e. HuggingFace [84]) and hence can be used as checkpoint. (2) If a specific fine-tuned model is not available, a fine-tuning step should be performed following the approach showed in the original implementation. This involves preprocessing data and encoding it, hence an instance of the desired model's parameters can be learned by following the approach exposed along with the original implementation as shown above. For fine-tuning, most model hyper-parameters are the same as in pre-training. Particularly, bigger batch size and smaller learning rate are used. However, authors from [17] points that the model seem to be invariant to hyper-parameter choice when applied to large dataset (e.g. 100k samples) which will be often the application case of our pipeline.

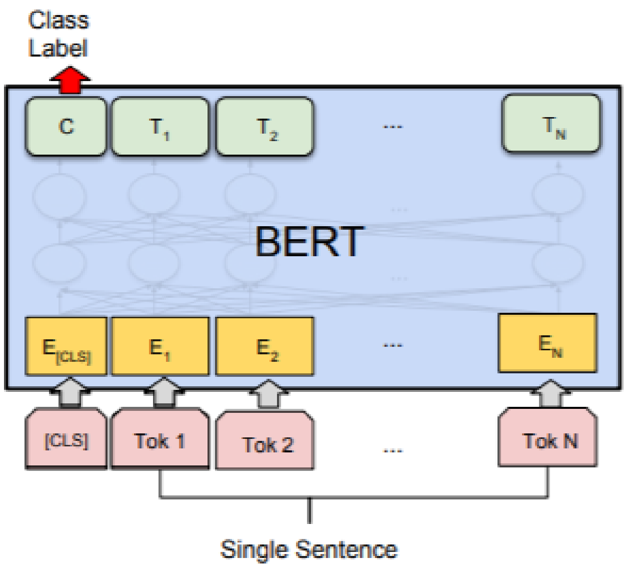
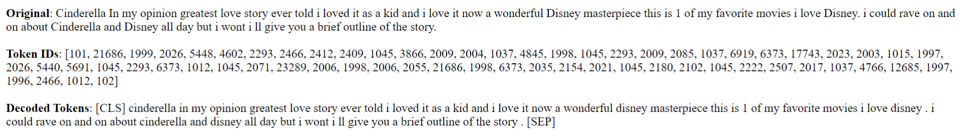


Figure 3.7: Bert fine-tuning illustration from [17]

3.4 First Inference Step

Once our fine-tuned model is available we can perform inference on target data. To perform this step, textual records need to be preprocessed first, as we've done in the previous step. This time we will make use of the target dataset. Once our model is fine-tuned to discern task-specific patterns among natural language data we can employ it to infer a probability distribution for out target dataset (i.e. IMDb). According to the training methodologies proposed along with the model implementation, input sentences must be tokenized and then words mapped to vocabulary indexes. The encoding employed is based on wordpiece. Once the sequence of to-


Original: Cinderella In my opinion greatest love story ever told i loved it as a kid and i love it now a wonderful Disney masterpiece this is 1 of my favorite movies i love Disney. i could rave on and on about Cinderella and Disney all day but i wont i ll give you a brief outline of the story.

Token IDs: [101, 21686, 1999, 2026, 5448, 4602, 2293, 2466, 2412, 2409, 1045, 3866, 2009, 2004, 1037, 4845, 1998, 1045, 2293, 2009, 2085, 1037, 6919, 6373, 17743, 2023, 2003, 1015, 1997, 2026, 5440, 5691, 1045, 2293, 6373, 1012, 1045, 2071, 23289, 2006, 1998, 2006, 2055, 21686, 1998, 6373, 2035, 2154, 2021, 1045, 2180, 2102, 1045, 2222, 2507, 2017, 1037, 4766, 12685, 1997, 1996, 2466, 1012, 102]

Decoded Tokens: [CLS] cinderella in my opinion greatest love story ever told i loved it as a kid and i love it now a wonderful disney masterpiece this is 1 of my favorite movies i love disney . i could rave on and on about cinderella and disney all day but i wont i ll give you a brief outline of the story . [SEP]

Figure 3.8: Sentence encoding example

ken IDs is obtained we must pad these sequences to a fixed length. This happens because input sentences are of variable length but our model feeds on fixed length inputs. For performance reasons we fixed such value to 64 tokens. Subsequently an attention mask for each sample is evaluated. Such masks are vectors of the same length of model's input, with a 1 in each

Model Name	Available Fine-Tuned Models
BERT	bert-base-uncased bert-large-uncased bert-base-cased bert-large-cased bert-base-multilingual-uncased bert-base-multilingual-cased bert-base-chinese bert-base-german-cased bert-large-uncased-whole-word-masking bert-large-cased-whole-word-masking bert-large-uncased-whole-word-masking-finetuned-squad bert-large-cased-whole-word-masking-finetuned-squad bert-base-cased-finetuned-mrpc bert-base-german-dbmdz-cased bert-base-german-dbmdz-uncased bert-base-japanese bert-base-japanese-whole-word-masking bert-base-japanese-char bert-base-japanese-char-whole-word-masking bert-base-finnish-cased-v1 bert-base-finnish-uncased-v1 bert-base-dutch-cased
DistilBERT	distilbert-base-uncased distilbert-base-uncased-distilled-squad distilbert-base-cased distilbert-base-cased-distilled-squad distilbert-base-german-cased distilbert-base-multilingual-cased distilbert-base-uncased-finetuned-sst-2-english

Table 3.1: Small sample of BERT and distilBert pre-trained models available on [84]

position corresponding to a word in the padded input vector and a 0 for all the padded positions. As explained before, such vector helps the model to learn the pattern of words' relative positions into the sentence. Token IDs along with attention masks are then fed with a random sampling strategy to the model. Output logits are hence calculated for each sample. A softmax head is added to return the probability distribution of the input over the output classes.

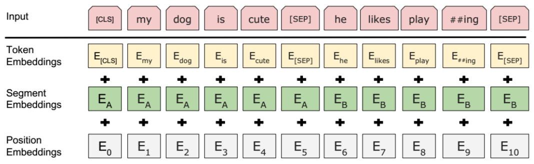


Figure 3.9: Input representation for BERT. Input embeddings is the sum of token embeddings, the segmentation embeddings and the position embeddings.

3.5 Training Data Building

At the end of the first inference step each sample of the target data is mapped to a probability distribution over the output classes. As the model that provided such distribution was trained on a different dataset, we can not expect reliable prediction over the whole set but still many samples will be predicted with high confidence. As showed in the work [86] as much as 20 labeled examples are necessary to achieve state of the art examples on sentiment classification task. The 85th and the 15th percentile of the probability distribution of class 1 is then calculated.

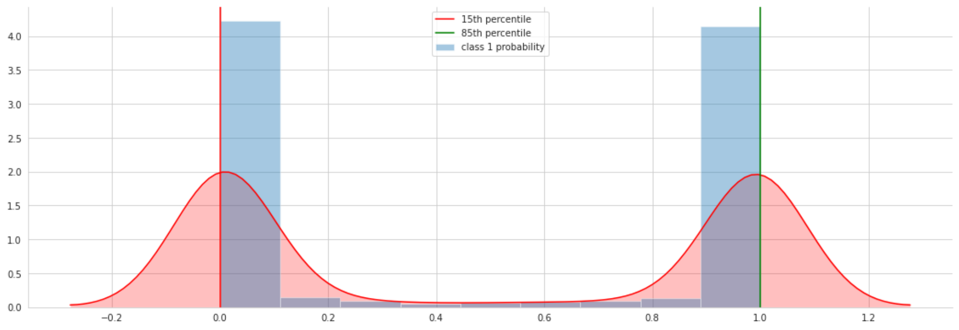


Figure 3.10: Class 1 probabilities from experiments on IMDb data

We split the dataset as follows:

- All the samples with a predicted probability below the 15th percentile (i.e. red line in the plot above) are assigned to the Labeled Set with label 0
- All the samples with a predicted probability below the 85th percentile (i.e. red line in the plot above) are assigned to the Labeled Set with label 1
- All other samples are assigned to the Unlabeled Set

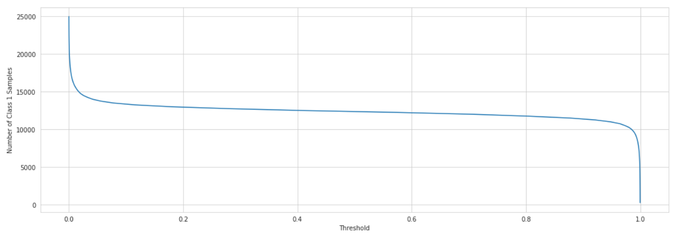


Figure 3.11: Number of samples assigned to class 1 with different probability thresholds

Such strategy was preferred against fixed threshold to ensure the population of the Labeled Set with variable ranges of confidence by, at the same time, keeping their count small. This was necessary as the model is prone to also shown in [86, p. 5] and addressed unfreezing gradually the number of labeled samples that the model is fed with. All samples belonging to Unla-

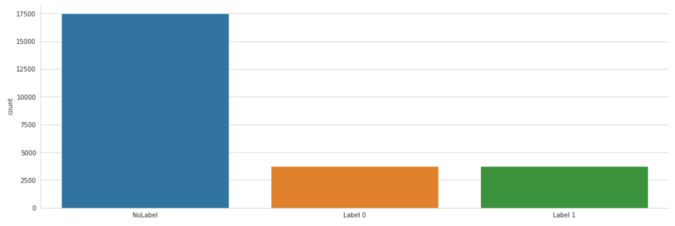


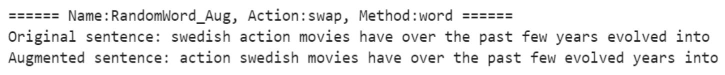
Figure 3.12: Counts of Samples divided by label, before augmentation

beled Set undergoes further preprocessing steps. First records are tokenized and cleaned from numbers and stopwords. This step is necessary to build models based on words statistics. As exposed in chapter 2 models based on words counts are higly biased towards stopwords hence we need to remove them from our data first. Now, text is ready to be used to train a TF-IDF model. At this point, each sample is augmented n times, with n being an hyper-parameter of the model. This step represents a bottleneck in terms of speed for the whole algorithm. In this stage several models are involved and computational time requirements increase accordingly. In our experiments n is empirically set to 2. Few experiments are performed on this parameter, as we will show in next pages, as computational time scales very fast for higher values. Each text produce hence 2 different augmented samples, each of them obtained with a different augmentation strategy sampled randomly from a dictionary. Many augmentation techniques are based on works from [16] and [86]. The implementations are based on [41] work. Techniques considered are:

1. **Random Augmentation** These augmentation are based on random

guess. An action is perfored by sampling randomly a token in the text.

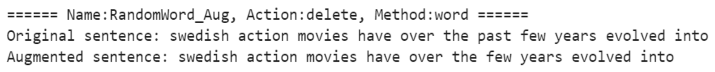
- Random Word Swapping: 10% of the words in the sentence (excluding stopwords) are swapped randomly. This approach is similar to next sequence prediction unsupervised technique used in [17] and is employed to enforce consistency in predictions on samples with small perturbations in word ordering.



```
====== Name:RandomWord_Aug, Action:swap, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: action swedish movies have over the past few evolved years into
```

Figure 3.13: Random swap example

- Random Word Deletion: 20% of the words in the sentence (excluding stopwords) are deleted randomly. This augmentation technique takes from the unsupervised pre-training objective function proposed with the original implementation of BERT [17] in which the model is trained to predict randomly masked tokens given the sentence (see section 2.4.1 for further details). This augmentation helps the model to be robust with respect to random word missing among similar sentences.

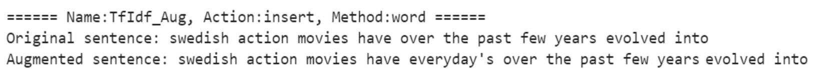


```
====== Name:RandomWord_Aug, Action:delete, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: swedish action movies have over the few years evolved into
```

Figure 3.14: Random deletion example

2. **Augmentations based on Word Statistics** The augmentations are based on word counts. Particularly, a TF-IDF approach showed to be effective, as also exposed in [86].

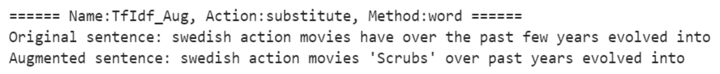
   - TF-IDF Random Word Insertion: A number equal to the 20% of the words in the sentence (excluding stopwords) is inserted in the sentence itself. A new word is injected to random position according to TF-IDF calculation. Top 15 score token are used for augmentation. As proposed in [86] we set a high probability for replacing words with low TF-IDF scores and a low probability for replacing words with high TF-IDF scores.



```
====== Name:TfIdf_Aug, Action:insert, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: swedish action movies have everyday's over the past few years evolved into
```

Figure 3.15: TF-IDF insertion example

- TF-IDF Random Word Substitution: A number equal to the 20% of the words in the sentence (excluding stopwords) is replaced according to TF-IDF calculations as before. Top 15 score token are used for augmentation. Probabilities are calculated as before.



```
====== Name:TfIdf_Aug, Action:substitute, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: swedish action movies 'Scrubs' over past years evolved into
```

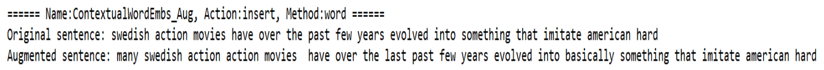
Figure 3.16: TF-IDF substitution example

3. **Augmentations based on Word Embeddings** Calculations made for these approaches are similar to previous ones, except that representations for words are taken from GloVe.

   - Word Insertion via GloVe embeddings: A number equal to the 20% of the words in the sentence (excluding stopwords) is inserted according to GloVe embeddings. Such vectors are used to find top 10 similar word for augmentation.
   - Word Substitution via GloVe embeddings: A number equal to the 20% of the words in the sentence (excluding stopwords) is replaced according to GloVe embeddings. Such vectors are used to find top 10 similar word for augmentation.

4. **Augmentations based on Contextualized Word Embeddings**

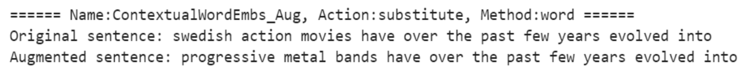
   - Word Insertion via distilBert contextualized embeddings: A number equal to the 50% of the words in the sentence (excluding stopwords) is injected according to distilBert predictions.



```
====== Name:ContextualWordEmbs_Aug, Action:insert, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into something that imitate american hard
Augmented sentence: many swedish action action movies  have over the last past few years evolved into basically something that imitate american hard
```

Figure 3.17: distilBert insertion example

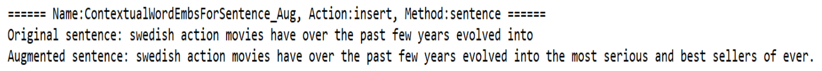
   - Word Substitution via distilBert contextualized embeddings: A number equal to the 50% of the words in the sentence (excluding stopwords) is substituted according to distilBert predictions.



```
====== Name:ContextualWordEmbs_Aug, Action:substitute, Method:word ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: progressive metal bands have over the past few years evolved into
```

Figure 3.18: distilBert substitution example

- Sentence Insertion via GPT2 contextualized embeddings: A number equal to the 50% of the words in the sentence (excluding stop-words) is injected according to GPT2 [60] predictions.



```
====== Name:ContextualWordEmbsForSentence_Aug, Action:insert, Method:sentence ======
Original sentence: swedish action movies have over the past few years evolved into
Augmented sentence: swedish action movies have over the past few years evolved into the most serious and best sellers of ever.
```

Figure 3.19: GPT2 sentence-level augmentation

5. **Back-Translation**

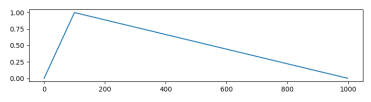
    - According to works like [18], [16] and [86] this is one of the best performing technique to build paraphrases of the input as it allows to produce different reliable semantic-equivalent sentences from a single sample. This is achieved using two models from Facebook AI Research Sequence-to-Sequence [53] published for WMT'19 [51] trained respectively on English to Russian and Russian to English tasks.

This step relates to [86]. As exposed in this work, augmentation techniques accurately designed to enforce consistency allows to gradually propagate confidence on prediction from labeled samples to unlabeled ones. With respect to mentioned approach we increased the pool of augmentation techniques. Some are inspired to objective task used to unsupervisedly pre-train BERT, others are variations of approaches presented in chapter 2. While random strategies are used to increase model capabilities to generalize over common mistakes in natural language data (e.g. missing word, uncommon word ordering), embeddings techniques focus on producing similar versions of the input that share the semantics of the original sentence and hence represent a valid sample for the model. A special mention goes to sentence augmentation approach that employs GPT2 [60] to build sentences completely new given an input sample.[1] Once the augmentation step is taken, sentences are encoded and masks are calculated. Resulting data is then fed to the model using a random sampling strategy.

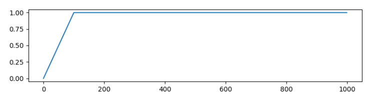
[1]This can be interactively used in the tool provided at: https://talktotransformer.com/

3.6 Inference Model Training

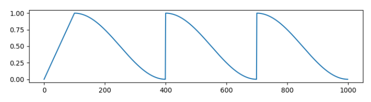
Following the approach proposed by [86], we change the original loss function implemented by [84] in distilBert model. In our implementation the model is trained with two different losses, depending on the current input being labeled or unlabeled. Once logits are calculated from model, if a label is available CrossEntropyLoss is evaluated, otherwise the model is forced to minimize consistency loss, that is the Kullback-Leibler divergence (as proposed by [50]), between the probability distribution predicted on the original data in the first inference step and the current predicted probability distribution. By employing this strategy, one enforces the model to be insensitive to the noise ϵ introduced by augmentations and hence smoother with respect to changes in the input (or hidden) space. In other words, minimizing the consistency loss gradually propagates label information from labeled examples to unlabeled ones. Results exposed in the paper [86] shows that following this training strategy on binary sentiment analysis tasks, with only 20 supervised examples, BERT outperforms the previous SOTA trained with full supervised data on IMDb. Most of the hyper-parameters are leaft as in the original implmentaion. Specifically, batch size was incresed to 32 and the optimizer was set to AdamW which is Adam with weight decay [37]. Eventually a linear schedule with warmup is set.



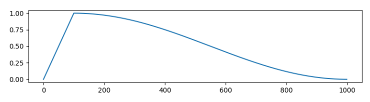
(a) Linear Schedule With Warmup



(b) Constant Schedule With Warmup



(c) Linear Schedule With Warmup and Hard Restarts



(d) Cosine Schedule With Warmup

Figure 3.20: Different Learning Rate Scheduling Strategies

3.7 Second Inference Step

In this last step we will employ the model just built to get metrics from target dataset. The obtained model is eventually used to predict labels for the target dataset. In this work, we will use a test set of 25.000 samples as provided in [42] to compare our performance evenly with other models.

Each sample is hence mapped to a probability distribution over the classes. The probability assigned to each class, in our framework, will be used as a confidence metric of the model with respect to its predictions and hence used as a quality index of the prediction itself.

3.8 Summary

In this Chapter we have seen how to apply many of the concepts exposed in chapter 2 to train a model on unlabeled data and to evaluate a confidence score for its predictions. We showed the various steps that makes up the evaluation pipeline. From task-selection and subsequent model fine-tuning to label generation on unseen data. We focused on how to perform thresholding for Labeled Set population and how to augment Unlabeled Set to enforce consistency in model's predictions with several different approaches as they represents the key aspects for the pipeline.

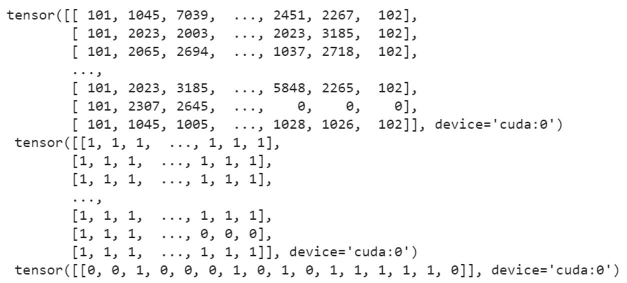
Chapter 4

Experimental Results

In this chapter we will expose the evaluation approach used to measure the results of our experiments together with comparisons of performance in different settings. We will dive deeper in the training loop analyzing how our evaluation metrics and loss function varies, at different granularities (both at epoch-level and step-level). A brief overview of some metrics used to evaluate performances is presented along with reasons that lead to such choices.

4.1 Evaluation Methodology

In all the process exposed in chapter 3 we do not make use of any true label to train the model on target data. By the way, to evaluate the performances of a classification model a referenced labeled dataset is needed. The choice of IMDb is indeed useful because its samples are labeled and, even if true classes are ignored during the whole algorithm, its original labels can be retrieved for samples belonging to the test set. We then split 25.000 samples from the original dataset [42] along with their labels, and use this set for evaluation purposes.



```
tensor([[ 101, 1045, 7039,  ..., 2451, 2267,  102],
        [ 101, 2023, 2003,  ..., 2023, 3185,  102],
        [ 101, 2065, 2694,  ..., 1037, 2718,  102],
        ...,
        [ 101, 2023, 3185,  ..., 5848, 2265,  102],
        [ 101, 2307, 2645,  ...,    0,    0,    0],
        [ 101, 1045, 1005,  ..., 1028, 1026,  102]], device='cuda:0')
 tensor([[1, 1, 1,  ..., 1, 1, 1],
        [1, 1, 1,  ..., 1, 1, 1],
        [1, 1, 1,  ..., 1, 1, 1],
        ...,
        [1, 1, 1,  ..., 1, 1, 1],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 1, 1, 1]], device='cuda:0')
 tensor([[0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0]], device='cuda:0')
```

Figure 4.1: Sample from test batch. The first tensor contains encoded sentences, the second tensor hold the vector mask for each sample and the last tensor stores the label of each sample in current batch.

Test set will eventually be composed of 12.500 class 1 and 12.500 class 0 labeled examples. This again helps the model to avoid any bias towards more numerous classes. Preprocessing steps are taken again for these samples hence sentences are encoded, padded and masks are computed. Logits are then calculated with the model and probability distribution over classes is assigned to each sample by applying a softmax head over the logits predicted. Once our predicted labels are available a set of performance metric

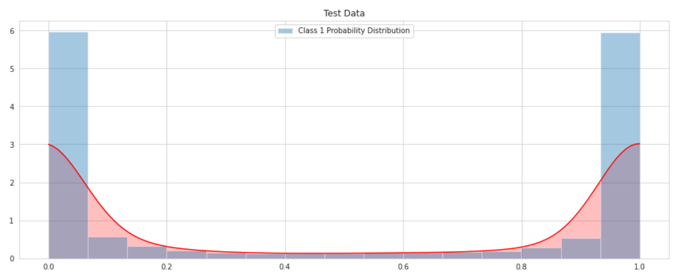


Figure 4.2: Distribution of class 1 probability over test samples

is evaluated against true label distribution.

4.2 Performance Metrics

To test results of our experiments, many different metrics are considered. Accuracy metric is useful to have a measure of how many labels were assigned correctly to proper class. In binary classification this equals to the Jaccard distance between outputs and true labels, i.e. the ratio between the intersection size and the union size of the two vectors. On the other hand, Matthew Correlation Coefficient resumes into a score the relatedness between inferred label distribution and the original one. Eventually, methods to visualize performance like confusion matrix and receiver operating characteristic will be presented and used to get different insights of model's prediction capabilities.

4.2.1 Accuracy, Precision and Recall

In classification tasks and information retrieval, precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of the total amount of relevant instances that were actually retrieved. Both precision and recall are a measure of relevance. In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total

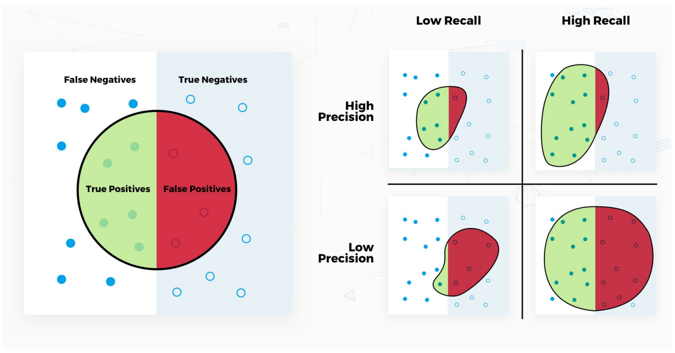


Figure 4.3: Precision and recall overview

number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). Recall in this context is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been). By keeping this naming we can evaluate precision and recall as:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

Accuracy is often referred as the closeness of a measured value to a known value. This measure is commonly used to evaluate statistical models, as it measures its statistical bias, the difference between the expect value of the results and the true underlying quantitative parameter being estimated. Within our framing we can define accuracy as the closeness of predicted label distribution to target one. It is evaluated as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Lastly we will also use F1-score, a measure that combines precision and recall by using the harmonic mean. It is evaluated as:

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

This measure may show some drawbacks since it gives equal importance to precision and recall. In general, different types of mis-classifications incur different costs. In other words, the relative importance of precision and recall is an aspect of the problem. By the way, as we have seen, our target test data consist of 12.500 labeled samples for each class hence we do not needed to address this problem directly. Nevertheless, to be in line with benchmarks and other works in related area, we moved towards more informative metrics so we can have a clearer image of algorithm performance.

4.2.2 Confusion Matrix

A common performance measurement for classification problems is confusion matrix. In a binary classification setting it is a specific table layout with 4 different combinations of predicted and actual values that allows visualization of the performance of an algorithm. It is useful for measuring Recall, Precision, and Accuracy.

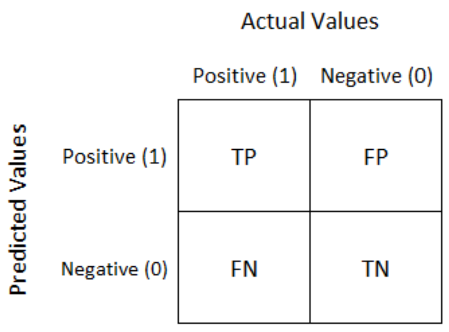


Figure 4.4: Confusion Matrix

- True Positive: Model predicted positive and it's true.
- True Negative: Model predicted negative and it's true.
- False Positive: Model predicted positive and it's false.
- False Negative: Model predicted negative and it's false.

Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is often mislabeling samples of a class as another.

4.2.3 ROC curve and AUC

The idea behind this metric is that a classification model is a mapping of instances between certain classes/groups. Because the classifier result can be an arbitrary real value (continuous probability values over [0,1]), the classifier boundary between classes must be determined by a threshold value.

A receiver operating characteristic curve (i.e. ROC curve) is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. To correctly understand this metric we should define further concepts. Particularly, keeping the naming defined just above, we can define true positive rate as

$$TPR = \frac{TP}{TP+FN}$$

this is the recall defined before; and false positive rate as:

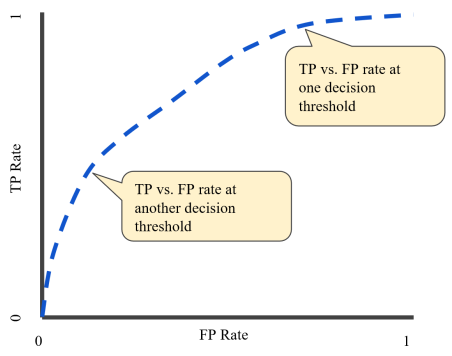


Figure 4.5: ROC curve

$$FPR = \frac{FP}{FP+TN}$$

this is a way to measure how often our model in our particular classification task is falsely rejecting the null hypothesis for the particular test. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (regardless of classification). The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. ROC analysis provides tools to select possibly optimal models and to discard sub-optimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis. ROC space is defined by FPR and TPR as x and y axes, respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Each prediction result or instance of a confusion matrix represents one point in the ROC space. The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is called a perfect classification. A random guess would give a point along a diagonal line (no-discrimination is performed) from the left bottom to the top right corners. The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random); points below the line represent bad results (worse than random). Note that the output of a consistently bad predictor could simply be inverted to obtain a good predictor. In the plot showed above 4 different points are highlighted each corresponding to a different classifier.

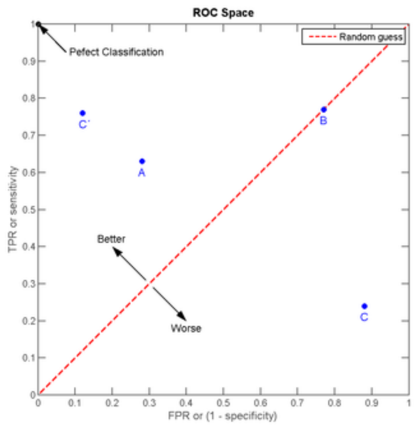


Figure 4.6: ROC space example

The result of method A clearly shows the best predictive power among A, B, and C. The result of B lies on the random guess line (the diagonal line), and hence we can expect an accuracy of 50% for such model. Among all, C is the worst performing method. However, when C is mirrored across the center point $(0.5, 0.5)$, one could obtain a method C' that is even better than the best A. Mirroring a model in the ROC space simply means to reverse the predictions of whatever algorithm that produced C confusion matrix. When the C method predicts positive (p) class or negative (n), the C' method would predict n or p, respectively. In this manner, the C' test would perform the best. The closer a point (representing a model) is to the upper left corner, the better it predicts.

The area under the curve (AUC) instead provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0. AUC has two desirable properties: (1) it is scale-invariant. It measures how well predictions are ranked, rather than their absolute values. (2) AUC is invariant with respect to classification threshold. It measures the quality of the model's predictions irrespective of what classification threshold is chosen. However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases: (1) Scale invariance is not always desirable. For example, sometimes we actually need differently weighted probability outputs, and AUC won't tell us about that. This however is not our test case as the in the dataset employed samples are well balanced among classes. (2) Invariance to classification threshold is not always desirable. In cases where the relative importance of false negatives and false positives is crucial, it is often relevant to only minimize one type of classification error. AUC isn't a useful metric for this type of optimization. Again, this is not our test case as for our task false negatives and false positives are indifferently weighted.

4.2.4 Matthew Correlation Coefficient

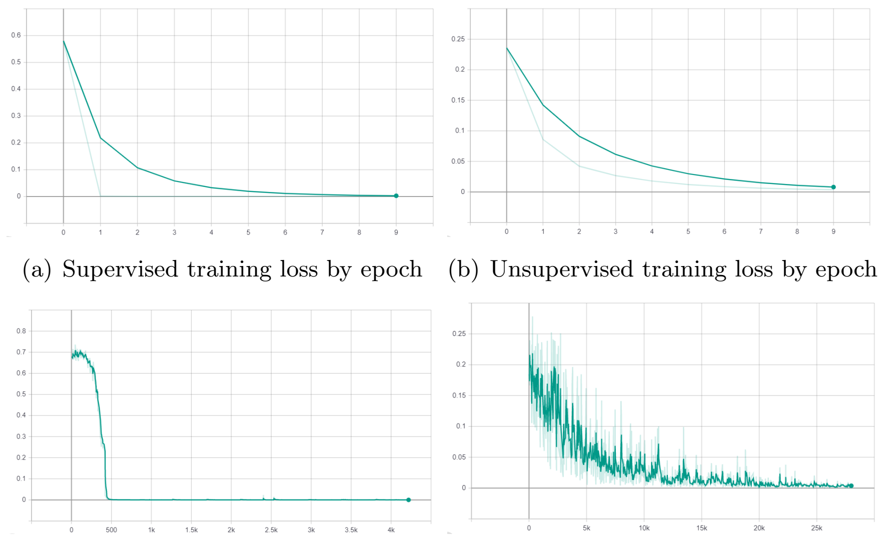
Matthew Correlation Coefficient takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. By employing the same naming introduced before, we can evaluate Matthew Correlation Coefficient as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

As explained in works like [13] by Jurman or [12] by Chicco, the Matthews correlation coefficient is more informative than F1 score and accuracy in evaluating binary classification problems, because it takes into account the balance ratios of the four confusion matrix categories (true positives, true negatives, false positives, false negatives). Matthews correlation coefficient is generally regarded as being one of the best way of describing the confusion matrix of true and false positives and negatives by a single number [59].

4.3 Results

Here we present results of the approach explained. As all experiments are performed on Nvidia Tesla K80 GPU offered on Google Colab cloud platform. As we can see below, loss function decrease fastly in few epochs, being both below 0.1 after 2 epochs of training. It should be noted that each training loop was implemented to perform a supervised learning step and an unspervised learning step in sequence. In the picture below we can see the model trained on 10 "*global*" epochs meaning that each iteration takes both a supervised and an unsupervised step (for a total of 20 epochs in the strict sense). These plots show the model quickly learning the param-



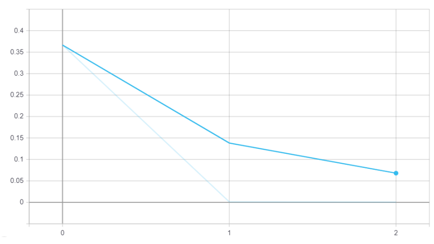
(a) Supervised training loss by epoch (b) Unsupervised training loss by epoch

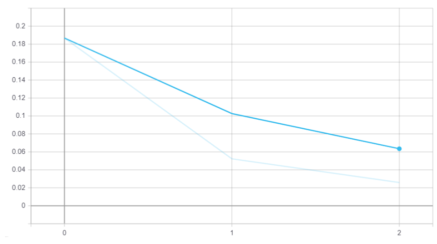
(c) Supervised training loss by step (d) Unsupervised training loss by step

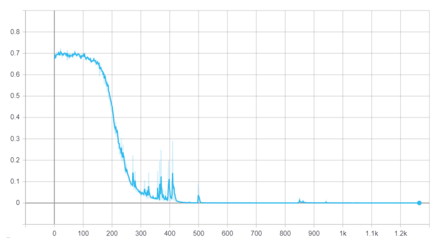
Figure 4.7: Loss functions at different granularity during 10 training epochs

eters for classification task, as we can see that both curves quickly flattens when the number of epochs increase. Moreover, letting the algorithm learn its parameters for an increasing number of steps, leads eventually to a decrease of performance as the model tends to overfit on supervised samples (as also showed in the work [86]). We hence decided to train our model for a total of 3 epochs as time requirements scales rapidly for higher numbers. Such number was chosen as an equilibrium between performance and evaluation time. We can see here a detail on 3 "*global*" epochs of training of loss functions both measured at each step and as an average at each epoch.

On the left we show performances of our model when the number of augmentations n is set to 2, while on the left we can see our model slightly


(a) Supervised training loss by epoch


(b) Unsupervised training loss by epoch


(c) Supervised training loss by step

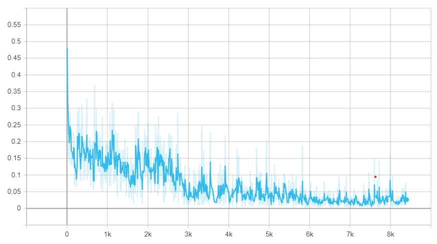
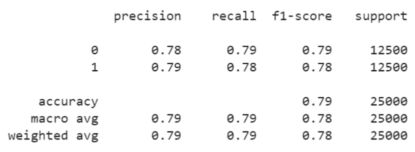
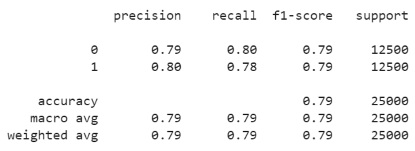

(d) Unsupervised training Loss by step

Figure 4.8: Loss functions at different granularity during training, detail at the first 3 epochs



```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79     12500
           1       0.79      0.78      0.78     12500

    accuracy                           0.79     25000
   macro avg       0.79      0.79      0.78     25000
weighted avg       0.79      0.79      0.78     25000
```

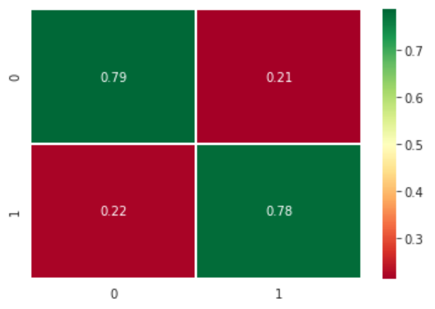
(a) Classification report from test data for a model trained for 3 epochs and with $n = 2$



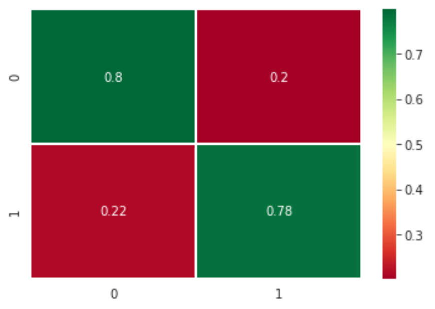
```
              precision    recall  f1-score   support

           0       0.79      0.80      0.79     12500
           1       0.80      0.78      0.79     12500

    accuracy                           0.79     25000
   macro avg       0.79      0.79      0.79     25000
weighted avg       0.79      0.79      0.79     25000
```

(b) Classification report from test data for a model trained for 3 epochs and with $n = 5$

benefits from further increasing n. As we can see, performance are well balanced among both classes and obtained F1 scores are comparable with other CNN or RNN approaches trained in a fully supervised regime. Support refers to the number of samples involved in the scoring calculations. Both class have 12.500 instances, this avoids the model learning any bias related to unbalanced counts. As said before, these metrics are summaries of techniques used to represent visually classification power of our model. By further inspecting predicted label distribution we can analyze how prediction accuracy varies among classes. By evaluating the number of True Positive, True Negative, False Positive and False Negative one can, as said, represent such values in a tabular form. Value shown below are normalized over rows (true values). The matrix shows that prediction error is balanced among classes. Specifically, we can calucalate:



(c) Confusion matrix on test set (normalized by rows) with $n = 2$



(d) Confusion matrix on test set (normalized by rows) and with $n = 5$

1. TP = 9875
2. TN = 9750
3. FP = 2625
4. FN = 2750
5. TPR = 0.78
6. FPR = 0.20
7. MCC = 0.58
8. AUC = 0.87

Another useful representation technique to understand output label probability distribution is a cluster map. As we have two classes it is easy to visualize with such tool the distribution of probabilities assigned over the whole dataset. Probability assigned to each sample for a given class is color coded following the scale reported in the picture. As we can see, the ma-

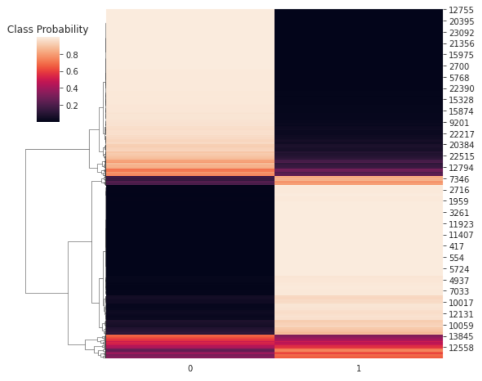


Figure 4.9: Clustermap of the probability distribution of class 1 over test samples.

jority of samples are markedly divided in two classes, while there is also a small fraction of the whole data where the model is less confident. These data may account for the ~ 0.21 loss in accuracy and may probably be better classified by enforcing consistency in predictions for a longer number of epochs. Our classificator performs well overall, by showing slightly better performance on negative samples when increasing the hyper-parameter n. Additionally, we present the ROC curve of the model. As explained before, our model appears to be near to upper left corner of the grid below, showing good classification capabilities. We evaluated the area under such curve and reported in the figure itself. Moreover, a dashed dotted line is plotted from point (0,0) t0 (1,1) to compare our model to a random classifier. Finally we

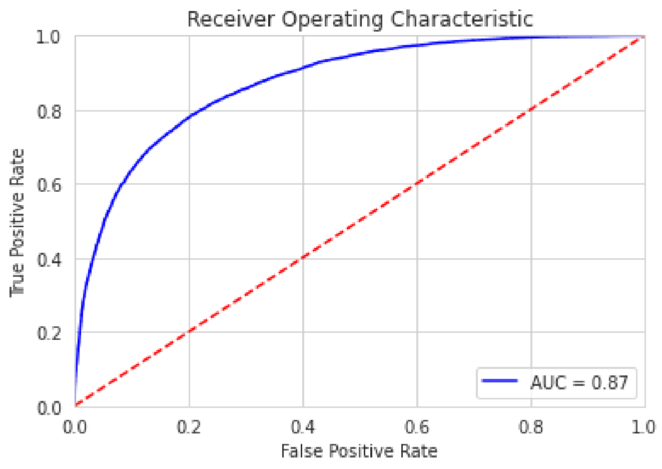
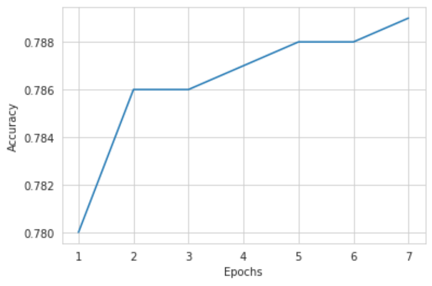


Figure 4.10: AUC and ROC curve for the model trained for 5 epochs with an augmentation factor of 5

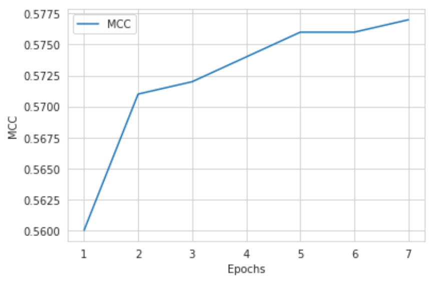
conclude with a table showing the results of a comparison of distilBert model trained with our approach on IMDb data. The results are showed in terms of Matthew correlation coefficient (MCC) and accuracy, as the number of epochs and the hyper-parameter n, representing the number of augmentations per sample, changes. As we can see, using our pipeline our model learns correctly its parameters from both labeled and unlabeled samples. If it would be possible to train such model with the given approach for a large number of epochs, consistency would be further enforced and this would be very beneficial to the model as directly results in an increased number of correctly labeled samples at inference time.

# Epochs	Accuracy	MCC	n
1	0.78	0.56	2
2	0.786	0.571	2
3	0.786	0.572	2
4	0.787	0.574	2
5	0.788	0.576	2
6	0.788	0.576	2
7	0.789	0.577	2
5	0.789	0.579	5
10	0.79	0.581	5

Table 4.1: Performance of the pipeline are measured on IMBb test set with a support of 12.500 samples.



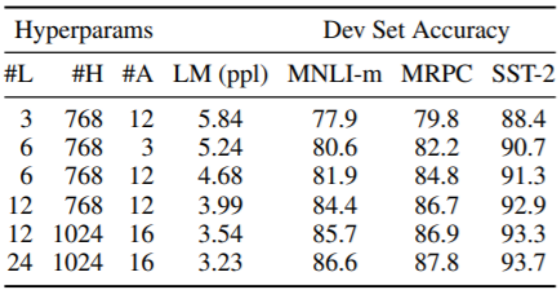
(a) Accuracy score for different number of epochs



(b) Matthew correlation coefficient score for different number of epochs

4.4 Further Works And Developments

The pipeline of operations proposed largely relies on works like [86], [79], [17] or [84] and plays with many concepts of recent NLP findings. Nevertheless it stands as a former step to build a framework for textual data quality assessment without the need of labels on target data. As the number of algorithms and concepts involved in the process is considerable, the work proposed seem to show room for many further developments. By having higher hardware capabilities and less requirements on computational time, many experiments with bigger model and higher number of augmentation could be performed. Even the quality of augmentations itself could be dramatically improved. With proper hardware settings, better adversarial samples could be built using not only augmentation techniques exposed in previous chapters but also language models like GPT2 [60] accurately fine-tuned to reproduce style from target corpus. Moreover, as pointed out by several works on this area, bigger models in term of size often are directly related to an increase in terms of performances, as shown in the picture below. This can also be



Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Figure 4.11: Ablation over BERT model size from original paper [17]. $\#L$ = the number of layers; $\#H$ = hidden size; $\#A$ = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

easily noted also giving a look to models that are populating top position in all NLP benchmarks, as in recent months their size kept increasing. More experiments should be performed also with different model architecture (e.g. Albert model [32]) to assess its impact in the proposed pipeline. As the use of BERT-like models is widespread, many open-sourced architectures are available, each built to better extract specific features from text. This may turn in a performance boost if combining a specific architecture to our work. Another critical issue that should be addressed in a more specific way is the approach used to select labeled samples after the first inference step. Even if thresholding using percentiles solves some of the issues related to such step, it is possible to take as labeled some samples assigned to the wrong class. This would introduce unwanted noise into the supervised step of our loss function. Overall, more ablation studies on the performance of each involved step should be performed as each step may be further optimized to reduce computational requirements. Moreover, further efforts should be made to identify specific metrics to evaluate textual quality, for example evaluating correlation between prediction confidence on different tasks for each record. As already introduced at the beginning of this work, data quality in the setting of natural language data is a broad open field which is now setting its frameworks for further developments.

Bibliography

[1] Serge Abiteboul. Querying semi-structured data. *Proc. ICDT 97*, 02 1970.

[2] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In *Lrec*, volume 10, pages 2200–2204, 2010.

[3] Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles, 2014.

[4] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):1–52, 2009.

[5] Toms Bergmanis and Sharon Goldwater. Context sensitive neural lemmatization with Lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1391–1400, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[6] Adam Bermingham and Alan F Smeaton. A study of inter-annotator agreement for opinion retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 784–785, 2009.

[7] Victoria Bobicev and Marina Sokolova. Inter-annotator agreement in sentiment analysis: Machine learning perspective. In *RANLP*, pages 97–102, 2017.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[9] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *CoRR*, abs/1010.3003, 2010.

[10] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[11] Jeanne Sternlicht Chall and Edgar Dale. *Readability revisited: The new Dale-Chall readability formula*. Brookline Books, 1995.

[12] Davide Chicco. Ten quick tips for machine learning in computational biology. *BioData mining*, 10(1):35, 2017.

[13] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020.

[14] C. Cichy and S. Rass. An overview of data quality frameworks. *IEEE Access*, 7:24634–24648, 2019.

[15] Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Semi-supervised sequence modeling with cross-view training, 2018.

[16] Claude Coulombe. Text data augmentation made simple by leveraging NLP cloud apis. *CoRR*, abs/1812.04718, 2018.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[18] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale, 2018.

[19] Hewlett Foundation. The hewlett foundation: Automated essay scoring, 12.

[20] Gaël Guibon, Magalie Ochs, and Patrice Bellot. From emojis to sentiment analysis, 2016.

[21] Robert Gunning et al. *Technique of clear writing*. McGraw-Hill, 1952.

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[23] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification, 2016.

[25] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*, 2008.

[26] Kian Kenyon-Dean, Eisha Ahmed, Scott Fujimoto, Jeremy Georges-Filteau, Christopher Glasz, Barleen Kaur, Auguste Lalande, Shruti Bhanderi, Robert Belfer, Nirmal Kanagasabai, et al. Sentiment analysis: It's complicated! In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1886–1895, 2018.

[27] Cornelia Kiefer. Quality indicators for text data. *BTW 2019*, 2019.

[28] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation, 2016.

[29] J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. *Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel.* Institute for Simulation and Training, University of Central Florida, 1975.

[30] Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, and Brown. Text classification algorithms: A survey. *Information*, 10(4):150, Apr 2019.

[31] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning, 2016.

[32] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[33] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, USA, 2nd edition, 2014.

[34] J. Liu, Z. Wang, G. Yan, and H. Lian. Quality measurement of judgment documents. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 296–299, July 2019.

[35] Jiawei Liu, Yang Xu, and Yaguang Zhu. Automated essay scoring based on two-stage learning, 2019.

[36] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[37] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[38] Julie Beth Lovins. Development of a stemming algorithm. *Mech. Translat. & Comp. Linguistics*, 11(1-2):22–31, 1968.

[39] Ping Luo, Zhenyao Zhu, Ziwei Liu, Xiaogang Wang, and Xiaoou Tang. Face model compression by distilling knowledge from neurons. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[40] Edward Ma. The biggest data challenges that you might not even know you have, 2016.

[41] Edward Ma. Nlp augmentation, 2019.

[42] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[43] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors, 2017.

[44] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109 – 165, 1989.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[46] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[47] George A Miller. *WordNet: An electronic lexical database.* MIT press, 1998.

[48] Eleni Miltsakaki and Karen Kukich. Automated evaluation of coherence in student essays. In *Proceedings of LREC 2000*, pages 1–8, 2000.

[49] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant, 2019.

[50] T. Miyato, S. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993, Aug 2019.

[51] Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. Facebook fair's wmt19 news translation task submission. In *Proc. of WMT*, 2019.

[52] Huyen Nguyen and Lucio Dery. Neural networks for automated essay grading, 2018.

[53] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[54] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models, 2017.

[55] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[56] R Plutchik. The emotions: Facts, theories and a new model. new york, ny, us, 1962.

[57] Robert Plutchik. A general psychoevolutionary theory of emotion. In *Theories of emotion*, pages 3–33. Elsevier, 1980.

[58] Martin F Porter et al. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[59] David Powers and Ailab. Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation. *J. Mach. Learn. Technol*, 2:2229–3981, 01 2011.

[60] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[61] Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. Semi-supervised learning with ladder networks, 2015.

[62] T.C. Redman. *Data Driven: Profiting from Your Most Important Business Asset.* General management. Harvard Business Press, 2008.

[63] Thomas C Redman and A Blanton. *Data quality for the information age.* Artech House, Inc., 1997.

[64] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2014.

[65] Corby Rosset. Turing-nlg: A 17-billion-parameter language model by microsoft. *Microsoft Research Blog*, 2020.

[66] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[67] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning, 2016.

[68] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[69] Klaus R Scherer et al. Psychological models of emotion. *The neuropsychology of emotion*, 137(3):137–162, 2000.

[70] Laura Sebastian-Coleman. *Measuring data quality for ongoing improvement: a data quality assessment framework.* Newnes, 2012.

[71] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data, 2015.

[72] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[73] Daniel Sonntag. Assessing the quality of natural language text data. *Informatik 2004–Informatik verbindet–Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik eV (GI)*, 2004.

[74] Kaveh Taghipour and Hwee Tou Ng. A neural approach to automated essay scoring. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1882–1891, Austin, Texas, November 2016. Association for Computational Linguistics.

[75] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks, 2019.

[76] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results, 2017.

[77] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, USA, 2003. Association for Computational Linguistics.

[78] Silvan S Tomkins and Imagery Affect. Consciousness, vol. 1, the positive affects, 1962.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[80] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[81] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.

[82] Markus Weimer, Iryna Gurevych, and Max Mühlhäuser. Automatically assessing the post quality in online discussions on software. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 125–128, 2007.

[83] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[84] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[85] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[86] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training, 2019.

[87] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

Appendix A

On Sentiment Analysis

Sentiment analysis is the interpretation and classification of emotions (positive, negative and neutral) within text data using text analysis techniques. Sentiment analysis allows businesses to identify customer sentiment toward products, brands or services in online conversations and feedback. In this Appendix, details about what sentiment analysis is, how it works and how it can be used to detect emotions within text. Sentiment analysis models detect polarity within a text (e.g. a positive or negative opinion). Categorizing texts by mining emotions hidden in them is essential for businesses. By automatically analyzing customer feedback, from survey responses to social media conversations, brands are able to listen atten-

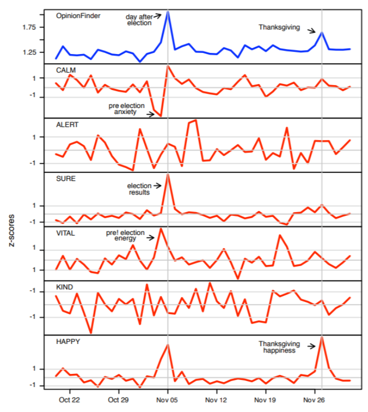


Figure A.1: Tracking public mood responses to presidential election and thanksgiving from tweets posted between October 2008 to December 2008 shows. From paper [9]

tively to their customers, and tailor products and services to meet their needs. It may go even further than customer segmentation and evaluation of products reviews and comments. Work from [9] shows how it is possible to accurately predict stock market trends by analyzing relevant Twitter feeds. A relevant effort has been made also in psychology research by works like [69] who defines each class of affective states by factors like its cognitive realization and time course.

Emotion	*Relatively brief episode of response to the evaluation of an external or internal event as being of major significance.* *(angry, sad, joyful, fearful, ashamed, proud, elated, desperate)*
Mood	*Diffuse affect state, most pronounced as change in subjective feeling, of low intensity but relatively long duration, often without apparent cause.* *(cheerful, gloomy, irritable, listless, depressed, buoyant)*
Interpersonal Stance	*Affective stance taken toward another person in a specific interaction, coloring the interpersonal exchange in that situation.* *(distant, cold, warm, supportive, contemptuous, friendly)*
Sentiment	*Relatively enduring, affectively colored beliefs, preferences, and predispositions towards objects or persons.* *(liking, loving, hating, valuing, desiring)*
Personality Traits	*Emotionally laden, stable personality dispositions and behavior tendencies, typical for a person.* *(nervous, anxious, reckless, morose, hostile, jealous)*

Table A.1: Scherer typology of affective states from [25]

A.1 Types of Sentiment Analysis

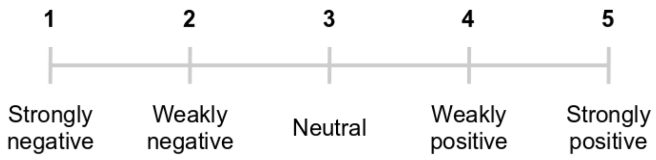
There exists various blends of sentiment analysis, from models that focus on polarity (binary: positive or negative) to those that detect feelings and emotions (angry, happy, sad, etc), or even models that identify intentions (e.g. interested v. not interested).

Some of the most popular types of sentiment analysis are encompassed as follows:

Binary Sentiment Analysis

Is the basic sentiment classification task. The task is to classify texts in two different categories (i.e. positive or negative) disregarding all possible sub-categorizations that may be evaluated in such texts. In other words, once each text is mapped to a value between 0 and 1, where a positive sentiment is a 1 and a negative is a 0, we consider a positive polarity for each sample assigned with a value higher than 0.5 and and negative for all the others. An example dataset is Stanford Sentiment Treebank (SST-2).

Fine-grained Sentiment Analysis



If understanding sentiment polarity is not enough, it is possible expand sentiment categories to include different shades:

- Strongly positive
- Weakly Positive
- Neutral
- Weakly Negative
- Strongly negative

This is usually referred to as fine-grained sentiment analysis. One can recover fine-grained sentiment labels from two-label setting by splitting the label range [0,1] into:

- Strongly Positive: Texts with a score from 0.8 to 1
- Weakly Positive: Texts with a score from 0.6 to 0.8
- Neutral: Texts with a score from 0.4 to 0.6
- Weakly Negative: Texts with a score from 0.2 to 0.4
- Strongly negative: Texts with a score from 0 to 0.2

An example dataset is Stanford Sentiment Treebank (SST-5).

Emotion detection

One of the most important affective classes is emotion, which [69] defines as a "relatively brief episode of response to the evaluation of an external or internal event as being of major significance". Detecting emotion has the potential to improve a number of language processing tasks. Automatically detecting emotions in reviews or customer responses (anger, dissatisfaction, trust) could help businesses recognize specific problem areas or ones that are going well for example. According to [25], there are two widely-held families of theories of emotion. In one family, emotions are viewed as fixed atomic units, limited in number, and from which others basic emotions are generated ([78], [56]). Perhaps most well-known of this family of theories is the wheel of emotion proposed in [57], consisting of 8 basic emotions in four opposing pairs: joy-sadness, anger-fear, trust-disgust, and anticipation-surprise, together with the emotions derived from them, as shown in figure A.2. The second class of emotion theories views emotion as a space in 2 or 3 dimensions ([66]). Most models include the two dimensions valence and arousal, and many add a third, dominance. These can be defined as:

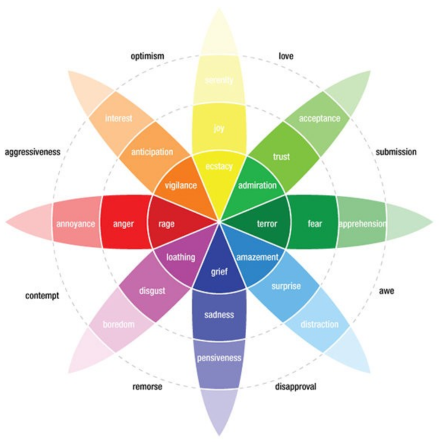


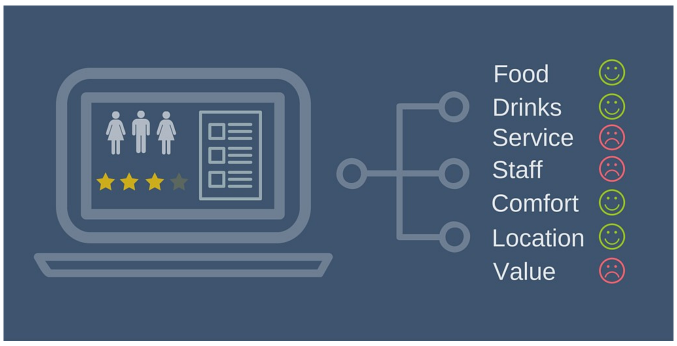
Figure A.2: Wheel of Emotions proposed by Plutchik

- valence: the pleasantness of the stimulus
- arousal: the intensity of emotion provoked by the stimulus
- dominance: the degree of control exerted by the stimulus

In particular, the valence dimension, measuring how pleasant or unpleasant a word is, is often used directly as a measure of sentiment ([25]).
Emotion detection aims at detecting emotions, like happiness, frustration, anger, sadness, and so on. Many emotion detection systems use lexicons (i.e. lists of words and the emotions they convey) or complex machine learning algorithms. One of the downsides of using lexicons is that people express emotions in different ways. Some words that typically express anger might

also express happiness. This is commonly known as polysemy and it is a long-known issue when analyzing text semantics.

Aspect-based Sentiment Analysis



A useful in-depth for companies when analyzing sentiments of texts (e.g. product reviews) may be to know which particular aspects or features are mentioned in a positive, neutral, or negative way. The big difference between sentiment analysis and aspect-based sentiment analysis is that the former only detects the sentiment of an overall text, while the latter analyzes each text to identify various aspects and determine the corresponding sentiment for each one. In other words, instead of classifying the overall sentiment of a text into positive or negative, this is another way of sub-categorizing sentiments, by associating specific sentiments with different aspects of a product or service.

Multilingual sentiment analysis

Multilingual sentiment analysis differentiates from the previous blends of sentiment analysis for the fact it is performed on texts written in different languages. Multilingual sentiment analysis can be difficult. It involves a lot of preprocessing and resources to train effective language models in various language and to set up a framework to combine them in order to obtain reliable scores.

A.2 Why Sentiment Analysis

It's estimated that 80% of the world's data is unstructured [40]. Huge amounts of text data (emails, support tickets, chats, social media conversations, surveys, articles, documents, etc), is created every day but not only it's hard but also time-consuming and expensive to analyze and understand such data. Sentiment analysis, however, helps businesses make sense of all this unstructured text by automatically tagging it. Benefits of sentiment analysis include:

- **Processing Data at Scale** Sentiment analysis helps businesses process huge amounts of data in an efficient and cost-effective way. Customer support conversations or customer reviews can be inputted to a pipeline and undergo several steps to be correctly categorized and subsequently sorted or further processed using other algorithms.

- **Real-Time Analysis Sentiment Analysis** Real-Time Analysis Sentiment analysis can identify critical issues in real-time (e.g. is an angry customer about to churn?) Sentiment analysis models can help you immediately identify these kinds of situations, so you can take action right away. Work proposed by [9] shows how a real time analysis on Twitter feeds can help perform accurate predictions on stock market.

- **Consistent criteria** Tagging text by sentiment is highly subjective, influenced by personal experiences, thoughts, and beliefs. By using a centralized sentiment analysis system, companies can apply the same criteria to all of their data, helping them improve accuracy and gain better insights.

A.3 Common Workflow

Sentiment analysis uses various Natural Language Processing (NLP) methods and algorithms, some of them exposed in chapter 2, that we will now see applied in the context of sentiment analysis. The main types of algorithms used are divided in:

- Rule-based systems that perform sentiment analysis based on a set of manually crafted rules. This is an approach that leverages smart features engineering to get the most from textual representations.

- Automatic systems that rely on machine learning techniques to learn from data. These approaches are often based on the techniques exposed in chapter 2

- Hybrid systems that combine both rule-based and automatic approaches. An example of an hybrid system can be seen in one of the works cited in chapter 2 for AES: [35] proposed an XGBoost model trained on both engineered features and scores produced by other algorithm's pipelines (see fig. 2.1 for further details).

Rule-based Approaches

Usually, a rule-based system uses a set of human-crafted rules to help identify subjectivity, polarity or the subject of an opinion. These rules may include various techniques developed in computational linguistics, such as: stemming and/or lemmatization (see subsection 2.2.1 for further details), tokenization, part-of-speech tagging and parsing, or word grouping based on semantics (WordNet [47]) or emotions (SentiWordNet [2]). This latter work focuses on Opinion Mining and shows a solution to addresses several subtasks of this particular NLP field, all of them having to do with tagging a given text according to expressed opinion. More in detail, each textual sample is assigned a ternary score: Obj(s), Pos(s) and Neg(s), respectively describing how Objective, Positive, and Negative the terms are. This was

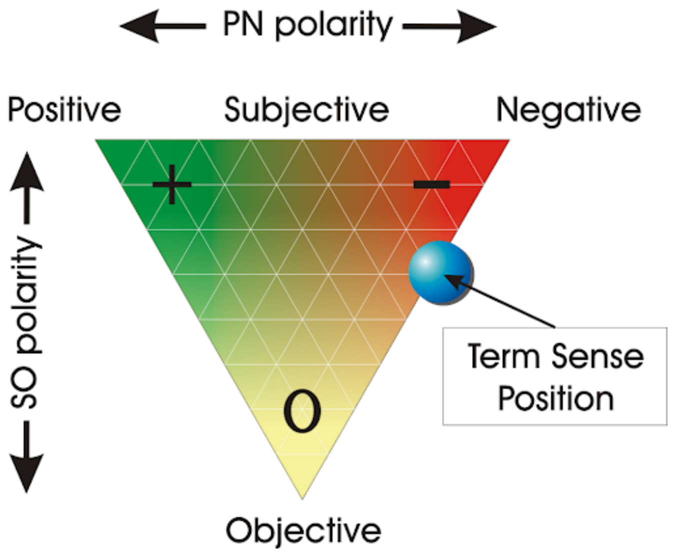


Figure A.3: SentiWordNet Visualization. Each word s is associated with three scores: Obj(s), Pos(s) and Neg(s)

obtained performing three related tasks: (1) determining text SO-polarity, or, in other words, deciding whether a given text has a factual nature (i.e. is an objective description of an event or a situation) or expresses an opinion on the exposed matter. This is performed via binary text categorization under categories Subjective and Objective. (2) determining text PN-polarity, or, in other words, deciding if a given text expresses a Positive or a Negative opinion on its subject matter (taking as known that the text express a fact in a subjective way). (3) determining the strength of text PN-polarity, or, in other words, deciding the opinion expressed in a text has a strong or a mild polarity.

By employing Sentiwordnet one could easily build a basic rule-based system by defining two lists of polarized words (e.g. negative words and and positive words) then count the number of positive and negative words that appear in a given text. If the number of positive word appearances is greater than the number of negative word appearances, the system returns a positive sentiment, and vice versa. If the numbers are even, the system will return a neutral sentiment. Rule-based systems are very naive since they don't take into account how words are combined in a sequence. Of course, more advanced processing techniques can be used, and new rules added to support new expressions and vocabulary. However, adding new rules may affect previous results, and the whole system can get very complex. Since rule-based systems often require fine-tuning and maintenance, they'll also need regular investments.

Automatic Approaches

Automatic methods rely on machine learning techniques. A sentiment analysis task is usually modeled as a classification problem, where a classifier is fed with a text and returns a category (e.g. positive, negative, or neutral). Usually, a machine learning classifier is designed to perform two steps:

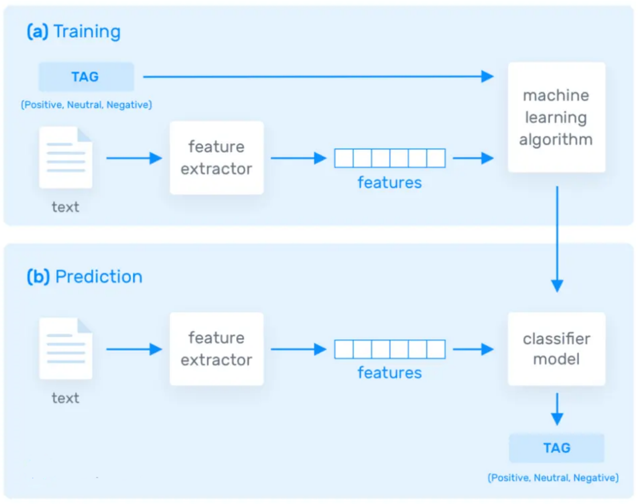


Figure A.4: General Workflow

The training process (a), in which our model learns to associate a particular input (i.e. a text) to the corresponding output (tag) while minimizing a problem-specific loss function. The feature extractor maps textual input into a feature vector (see subsection 2.2 for more details on such techniques). Pairs of feature vectors and tags (e.g. positive, negative, or neutral) are fed into the machine learning algorithm to train the model. The prediction process (b), in which the feature extractor is used to transform unseen text inputs into feature vectors. These feature vectors are then fed into the model, which generates predictions for such samples (i.e. tags).

Hybrid Approaches

Hybrid systems combine the desirable elements of rule-based and automatic techniques into one system. One huge benefit of these systems is that results are often more accurate.

A.4 Sentiment Analysis Challenges

When analyzing words semantics many different challenges may arise from the fact that often the meaning of a sentence is sometimes more than the simple sum of its words' meanings. This is the case, for example, of metaphors, anaphors and in general of rhetorical structures in text.

Subjectivity and Tone

The detection of subjective and objective texts is just as important as analyzing their tone. In fact, so called objective texts do not contain explicit sentiments. It often happens indeed, that sentences with similar structure in terms of POS tags and word ordering have different subjectivity. This is the case of simple sentences like 'the book is nice' and 'the book is red'. Both have the same parse tree but only one of them is subjective. Predicates (adjectives, verbs, and some nouns) should not be treated the same with respect to how they create sentiment.

Context and Polarity

Communication via natural language, often leverage context to express complex ideas. It is hence common to consider its context when trying to understand a sentence. In the same way, analyzing sentiment without context gets pretty difficult. However, machines cannot learn about contexts if they are not mentioned explicitly and this leads to model always increasing in complexity. Many steps have been taken in these years to address this issue and modern transformer models have shown the ability to consistently address contextualization when building words representations.

Irony, Sarcasm and other Rhetorical Structures

When it comes to irony and sarcasm, people express their negative sentiments using positive words, which can be difficult for machines to detect without having a thorough understanding of the context of the situation in which a feeling was expressed. The biggest problem is that there is no textual cue that can help the algorithm to learn, or in any other way discern that sentiment from the meaning conveyed by its words. This same issue extends to many other rhetorical structures (e.g. metaphors), and indeed represent one of the main problem when trying to build tools capable of correctly represent semantics of text.

Emojis

There are two types of emojis according to [20]: Western emojis (e.g. *:D*) are encoded in only one or two characters, whereas Eastern emojis (e.g. ¯_(ツ)_/¯) are a longer combination of characters of a vertical nature. Emojis play an important role in the sentiment of texts, particularly in tweets analysis as they represent a clear and intentional reference to sentiment regarding the fact exposed. To account for these additional features in texts, in addition to word-level analysis, character-level analysis should be performed too. Specific preprocessing might also be needed as some emoji may not apply to the encoding used for other text. For example, you might want to preprocess social media content and transform both Western and Eastern emojis into tokens and whitelist them (i.e. turn them in a subset of features) in order to help improve sentiment analysis performance.

Defining Neutral

Defining what we mean by "neutral" is another challenge to tackle in order to perform accurate sentiment analysis. As in all classification problems, defining categories -and, in this case, the neutral tag- is one of the most important parts of the problem. What you mean by neutral, positive, or negative does matter when you train sentiment analysis models. Since classifying data requires tagging criteria to be consistent, a good definition of the problem is a must.

A.5 How Accurate Is Sentiment Analysis?

Sentiment analysis is a complex task even for human beings. According to works like [6], [7] or [26] disagreement is not always negligible even between human annotators. With these premises it is easy to imagine that classifiers trained for such task might not be as precise as other types of classifiers. This is because machines learn from the data they are fed with (as said above) and if labeling is not consistent among data, models may be not able to correctly learn its parameters. On the other hand, by the way, it should be kept in mind that these noisy data represents often a small fraction of the huge amount of the data that modern models use to be trained.