POLITECNICO DI MILANO
Master of Science Degree in Computer Science and Engineering
Department of Electronics, Information and Bioengineering

GBDT-Based Stacking Ensemble Experiments on an Imbalanced Challenge Dataset

Supervisor: Prof. Paolo Cremonesi
Co-Supervisor: Cesare Bernardis

Author:
Luca Bartoccioni
ID Number 918963

Academic Year 2019-2020

Abstract

Recommender Systems are algorithms which main purpose is to provide personalized recommendations to users about items to buy or content to interact with. In the last years, because of their effectiveness, they spread on a large scale, being widely applied on many platforms and services. According to the available information, many techniques may be employed in order to provide such suggestions, each of which exploits the data in different ways. It is possible to combine them using a set of techniques that falls under the name of Hybrid Recommender Systems, in this way it is possible, by exploiting different kinds of information, to have an increase in accuracy.
Among the most competitive and performing hybrids techniques there are the stacking ensembles. They consist in aggregating the forecasts made by different models in order to overcome their weaknesses and make more accurate predictions by using a *meta-model*. A *meta-model* is a machine learning algorithm trained with the sole purpose of aggregating the predictions made by other models. Also stacking ensembles are in general more reliable and robust to the noise in the data.
This thesis studies such techniques, in particular the ones performed using *gradient boosting decision trees* algorithms, with the purpose of exploring the ways ensembles can be implemented and trying to propose innovative alternatives. By doing so, two different categories of experiments were proposed, each of which exploited a particular aspect of the ensemble, with the goal of enhancing their performance. The performed division resulted in architecture and data manipulation experiments.
The dataset used to perform such work is the one provided in occasion of the *RecSys Challenge 2020*, where we, a group of MSc students form Politecnico di Milano, took part achieving the overall 4^{th} position and the 1^{st} among academics teams.

Sommario

I Recommender Systems sono algoritmi il cui scopo principale è quello di suggerire agli utenti, oggetti o contenuti con i quali interagire. Negli ultimi anni, per via della loro efficacia, si sono diffusi su larga scala venendo applicati in diverse piattaforme e servizi. A seconda dei dati che hanno a disposizione, questi impiegano strategie differenti per le raccomandazioni, ognuno dei quali li sfrutta in modo differente. Esiste la possibilità di combinare questi suggerimenti usando un insieme di tecniche che prende il nome di Hybrid Recommender Systems. Usandoli è possibile sfruttare differenti tipi di informazioni per poter aumentare l'accuratezza delle predizioni.
Tra gli approci ibridi più competitivi e performanti si trovano gli *stacking ensmeble*. Questi consistono nell'aggregare le predizioni fatte da modelli diversi, in modo tale da compensare le loro debolezze e fare suggerimenti con maggiore accuratezza per mezzo di un *meta-modello*. Un *meta-modello* è un algoritmo di machine learning utilizzato con il solo scopo di aggregare al meglio le predizioni fatte dagli altri algoritmi. Un'altra caratteristica che rende gli *stacking ensemble* interessanti è la loro robustezza rispetto al rumore nei dati.
In questa tesi vengono studiate queste tecniche, in particolare quelle che impiegano i *gradient boosting decision trees* nell'*ensemble*, esplorando il modo in cui questi modelli vengono combinati e proponendo soluzioni innovative. Nel fare ciò, due differenti categorie di esperimenti sono state proposte, ognuna delle quali ha sfruttato un aspetto particolare dell'*ensemble* con lo scopo di aumentarne le performance. Le soluzioni proposte sono quindi state divise in: manipolazione dei dati e esperimenti sull'architettura.
Il dataset su cui è stato fatto tale lavoro è quello fornito in occasione della *RecSys Challenge 2020*, alla quale il nostro team, composto da cinque studenti magistrali iscritti al Politecnico di Milano, ha preso parte ottenendo la 4^a posizione nella classifica complessiva e la 1^a tra i team accademici.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recommender Systems are a class of algorithms, which goal is to make personalized suggestions to users. In their most common form, given the history of interactions between a user and an item, they aim to predict their future connections. They make use of different approaches, as machine learning, data mining and information retrieval techniques, taking into account also social dynamics, psychological aspects and trends.
In the last twenty years, the use of e-commerce services and social networks grew exponentially and so did the use of Recommender Systems. They earned popularity, becoming a crucial resource in many successful applications.
Among the famous companies which heavily rely on Recommender Systems stand out examples like Amazon or Facebook. They use these engines to make their service more pleasant and enhance the user experience. The field of application of these kind of algorithms is very large, they are widely employed, as previously said, both in e-commerce and social networks, but also in services of music and film streaming as in job research platforms.
Due to the nature of these problems, several competitions are organized every year in order to experiment innovative solutions to ever new problems. One of the greatest competitions in the field was the Netflix Price, held in 2016 and hosted by Netflix. They offered a million dollar to whoever could raise the accuracy of their Recommender System of at least 10%. The competition lasted three years and was won by a joint-team composed by several researcher called *BellKor in BigChaos*.
This thesis work is evolved around one of these challenges: the *RecSys Challenge 2020*. This competition is held every year by a different company in occasion of the *ACM International Conference on Recommender Systems*. In 2020 the host was Twitter the company that owns the homonym social

network. We took part at this challenge as a team formed by five MSc students of Politecnico di Milano, under the name *BanaNeverAlone*. We achieved the 4^{th} place in the competition (first considering only academic teams). The paper we presented after our solution, *Multi-Objective Blended Ensemble For Highly Imbalanced Sequence Aware Tweet Engagement Prediction*, was accepted and published in the proceedings of the conference.
The task presented by Twitter was in the domain of social networks recommendations, but with several peculiarities. Instead of asking to recommend a list of contents to each user, it was asked to predict the possibility that a user would've interacted with a specific content. Making forecasts, as accurate as possible, about users' tastes, it is crucial for social networks. By doing this they increase the pleasantness of their service, in order to arouse interest in users and make them spend more time on it.
The works performed in this thesis is based on the dataset and the ensemble solution provided by our team, *BanaNeverAlone*, for the challenge. Ensemble approaches are commonly used in the fields of Machine Learning and Recommender Systems, to combine the solutions generated by different models. This approach is particularly good because it allows to overcome some weaknesses of non-combined algorithms and provide solution that, in terms of accuracy and reliability, overcomes the single models.
There are a lot of possibilities when it comes to implementing this technique. Each of them tries to exploit diverse aspects with the goal of increasing the performance. In the context of this thesis two main approaches are explored: the manipulation of the dataset and the use of different structures. For each of them, some experiments were designed over well known and brand new ideas. These solutions are then tested within an ad-hoc environment developed starting from the solution our team used for the *RecSys Challenge 2020*.

1.1 Thesis Structure

This thesis is structured as follows

- Chapter 2 provides a brief description about the *RecSys Challenge 2020* task and the solution our team submitted.
- Chapter 3 contains a review of the literature works that are used in this thesis and inspired some of the provided solutions.
- Chapter 4 analyzes the environment more in depth, illustrating the dataset provided by Twitter and some of the approaches our team

used.

- Chapter 5 describes the design of the proposed solutions and the setup of the environment on which to run them.
- Chapter 6 shows and compares the solutions to the experiments.
- In Chapter 7 reside the conclusions and considerations drawn from the results.

Chapter 2

Problem Description

This section briefly presents the problem in the context of the attended competition, hence an insight on the object of study of this thesis. In the first section the challenge as long as the problem are shown and briefly analyzed, while the second part shows the approach used to compete.

2.1 ACM RecSys Challenge 2020

Every year since 2010 in conjunction with the *ACM International Conference of Recommender Systems* is held the *RecSys Challenge*, sponsored by a company that has interest in the field.
In 2020, RecSys Challenge [2] was hosted by Twitter, the company that owns the famous social network. Unlikely many previous years' problems, the one presented in 2020 was quite different, in fact it was requested to classify some *user-tweet* interactions. Specifically it was asked to predict, in percentage, which action a user would have taken with respect to a tweet. There were four different kind of interactions, respectively *like*, *retweet*, *reply* and *retweet with comment*, each of which came with a different distribution of positive and negative samples. In the dataset there were a high number of negative interactions, in which either a user saw a tweet and ignored it or didn't see it at all. The imbalance among positive and negative samples ranged from an almost balanced problem for the *like* class to a heavily imbalanced one for the *retweet with comment* class.
The challenge was split in two different phases. In the first one, competitors were allowed to test their model on a *public test set* with tweets sampled a week after the sampling of the training set. While in the last phase, which lasted more or less a week, the *private test set* was released with tweets sampled two weeks after the training set. Unlikely the other challenges in

both phases the competitors had the chance to submit an unlimited amount of solutions, this was possible due to the nature of the dataset, containing roughly three-hundred millions of interactions *user-tweet*, so the submission limit was implicitly forced by the duration of the training phase of the models.
One additional peculiarity of this challenge was that two different metrics, *PR-AUC* and *RCE* have been employed in order to better evaluate the solutions. The final score on the leaderboard was calculated by summing the rank obtained by the two metrics, the lower was the total score the higher the solution was ranked.

2.2 Problem Solution

Our team, *BanaNeverAlone*, was composed by five members, working remotely due to *covid-19* restrictions.
The first part of the proposed solution consist an accurate engineering of the dataset. Initially we began to extract brand new features from it, regarding both the users and the tweets. This resulted in almost 300 new features of which only half were used. Among the ones about the tweets, there were: text, hashtags and other domain specific information. These were converted into tokens by using a multilingual pretrained version of *BERT* [10], a transformer model introduced by Google. Due to the complexity of textual data, we decided to process it separately from the other features. In order to do so, we fine-tuned a *DistilBERT* [30] model, that allowed us to transform the text tokens into embeddings: vectors of fixed length encapsulating the core textual data. Their excessive size didn't make them suitable to be used as features for *gradient boosting decision trees* algorithms, so we created an apposite model. It was a multi-classification *feed forward neural network* trained on both the embeddings and a set of other core features. Its purpose was to output a vector of predictions for each class. The preprocessing pipeline of the textual features is shown schematically in figure 2.1.

To process the non-textual features we followed a different approach. We chose to use two distinct implementations of the *gradient boosting decision trees* (GBDT) algorithm: *XGBoost* and *LightGBM*. Unlikely the neural network, these models didn't allow to make multi-class predictions, so we had to split the problem training both of them for each class. To optimize their hyperparameters we used an automated hybrid optimization technique, which merged both random and bayesian optimizations.
In order to combine the predictions we used a stacked ensemble approach using a *LightGBM* meta-model. Due to the impossibility to perform the

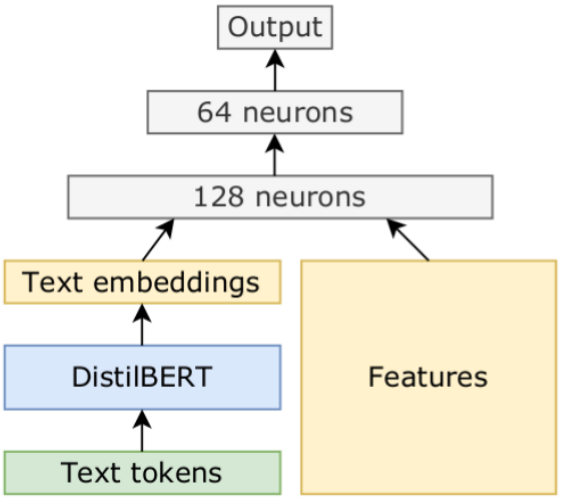


Figure 2.1: Schematics of the text processing pipeline.

multi-classification with these models, a meta-model had to be employed to combine the forecasts made on each label. Their training set consisted in a combination of a subset of core features and predictions made from the other models. Moreover two different training set choices were employed for meta-models, the first for the *like* class and the second for the remaining classes. The structure of the ensembles, along with their features is provided in figure 2.2. Our solution achieved the 4^{th} position in the challenge, an overview of

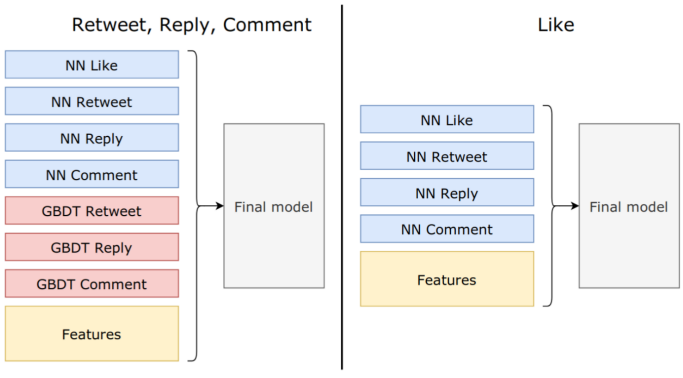


Figure 2.2: Schematics of BanaNeverAlone solution ensemble.

which is provided in the paper *Multi-Objective Blended Ensemble For Highly*

Imbalanced Sequence Aware Tweet Engagement Prediction [14].

Chapter 3

State of the Art

In this chapter we provide an analysis of the state of the art regarding Recommender Systems and related machine learning techniques.

3.1 Recommender Systems

Recommender systems are a subset of information filtering systems that try to predict the ratings or the preferences that a user would give to an item. They have become very popular in recent years and they are used in a lot of areas such as movies, musics, e-commerce, advertising and social network, in order to provide different and personalized results for each user [28].
The most common problem they're picked to solve is the recommendation of an item to a user, one of the machine learning techniques most used for recommendations are the *Gradient Boosting Machines* [15][16][24]. These algorithms, being powerful and robust, in a short time became a standard among the challenges and high-end applications.
Recommender systems can be roughly split in two macro categories, collaborative filtering, content-based.

3.2 Collaborative Recommender Systems

This technique is one of the most popular in the field of recommender systems due to its simplicity and accuracy. Collaborative filtering methods produce user specific recommendations based on their history patterns, without using exogenous information about either users or items. This kind of recommender systems relies on the feedbacks provided by users either implicit or explicit. Due to the lack of explicit feedbacks it is not rare to use the implicit ones, that are a pleasantness index extracted from the user behavior

with respect a specific item.
This data is stored in a matrix named *User Rating Matrix (URM)*, that on the rows has the users, on the columns the items, while on the cells the ratings. Collaborative filtering algorithms make use of similarity among the users' interactions contained in the URM in order to find the most likely item to recommend.

3.3 Content-based Recommender Systems

This approach is also very popular in the field of recommender systems. While the collaborative filtering methods try to exploit similarity among the users' interaction history pattern, this kind of recommender systems try to exploit the similarities among items' or users' informations, called features. In this case there are two kind of matrices that can be used in order to compute the similarities: *Item Content Matrix* (ICM) and *User Content Matrix* (UCM). The first one on the row, has the items, on the columns the items' features and expresses items' attributes. The second similarly have the users on the rows, the features on the columns and expresses the users' characteristics.

3.4 Hybrid Recommender Systems

These recommender systems are based on the combination of two or more simple models. The idea is that the prediction of a model can be used to reinforce the predictions of another one and cover its weaknesses [6]. In order to maximize the revenue provided by this approach it is necessary that the diversity among the predictions, hence among the models, is as great as possible. This status can be reached by using different models, also by training them on subset of features or samples, so each of them would capture only a unique part of the underlying dynamics of the data, rather than a single, but possibly noisier representation of the overall data distribution.

3.4.1 Averaging and Voting

The most common techniques used in hybrid recommender systems and more in general hybrid machine learning models is the averaging for regression problems and voting for classification problems. These techniques are computationally fast, in situations where there are a large number of models to hybridize these approaches are very competitive due to absence of training time and simplicity.

3.4.1.1 Averaging

The idea behind this algorithm is to train several models and take the suggestions from all of them [3]. Their contribute may be weighted and usually proportional to their individual accuracy, in order to compute the weighted average among them and taking that as prediction. The main contribution of this algorithm is to reduce the variance of the system, hence increasing its robustness and decreasing overfitting, because various guesses are taken into account.
Another approach used in averaging is to introduce a machine learning model to tune the weights of the linear combination [36] [19]. In this way the weights are no longer arbitrarily proportional to the model accuracy, but finely tuned by an apposite machine learning *meta-model*. The most common technique is a linear regression, but many other approaches were explored.

$$y = \frac{\sum_{m=1}^{M} \alpha_m p_m}{\sum_{m=1}^{M} \alpha_m} \tag{3.1}$$

Equation 3.1 represents the averaging formula, where M are the number of models to hybridize α_m is the weight of the m^{th} algorithm and p_m its predictions.

3.4.1.2 Voting

In case of classification, a democratic choice between classes is made by using voting [20], where, after training several models, each of them votes for the class it has predicted, even in this case it is possible to assign weights to the votes proportional to the model's accuracy.
Majority voting [23], one of the most common solutions for this approach, consists of selecting a set of models to ensemble, let every model vote for the class it has predicted and finally choose the class with the major number of votes for prediction. Voting:

$$y = \arg\max_{m=1}^{M}(p_m)$$

Where M are the number of models to hybridize and p_m the models' predictions.

3.4.2 Switching

A switching hybrid [17] uses some criteria or rules in order to choose the best available prediction among the ones provided by hybridized models. By choosing this approach the predictions are not directly combined as in

averaging or voting, which in absence of a strong rule to do so could be a winning approach.
This technique is often used to deal with the cold start problem and use *ad-hoc* predictions for recently added users or items.

3.4.3 Cascade

Cascade hybridization involves a staged process. The first stage produces a coarse ranking among recommendations while the second one refines this ranking in order to increase its accuracy.
Cascading allows the system to avoid employing the second, lower-priority, technique on items that are already well-differentiated by the first. In addition, the cascade is by its nature tolerant to the noise introduced by the low-priority technique, since recommendations provided by the high-priority model can only be refined, not overturned [5].

3.4.4 Stacked Generalization

Stacked Generalization [39] formalizes the idea of using a meta-model of machine learning to ensemble other models.
This approach is based on a multi level schema. There are two or more generalizers in each level apart from the last one, which has only one. A generalizer is a mapping from both a dataset $\{x_k \in \mathbb{R}^n, y_k \in \mathbb{R}\}_{k=1}^{M}$ of M samples and a question $\in \mathbb{R}^n$, into a guess $\in \mathbb{R}$, where $\mathbb{R}$ and $\mathbb{R}^n$ are coordinate spaces over the real numbers with dimension respectively of 1 and n. These models are trained on the predictions made by the previous level, with the only exception of the first one, trained on the original dataset. The *level-0* learning set is simply the original dataset $D_0 = \{(f_n, y_n), n = 1 \dots N\}$ where f_n are the features of the original dataset and y_n its label on the n^{th} sample. *Level-0* generalizers G_{k,D_0} are processes which generalize directly from it, their guesses on the training set are defined as y_{k,D_0}, whose are made by cross validating it, in order to make them more reliable.
Level-1 learning set, is built by aggregating the guesses of the K previous level's models, being $D_1 = \{(y_{0,n,D_0} \dots y_{K,n,D_0}), n = 1 \dots N\}$, while *level-1* generalizers G_{k,D_1} are processes which learns and subsequently guess on the D_1 dataset the same way *level-0* models did on D_0. Following this insight it is possible to build an undefined number of levels. It is important that the last level uses a single generalizer in order to ensemble all the previous level predictions into a single one, which may require to be transformed in a *level-0* prediction if its form has been altered through the various stages.
This approach performs generally better with respect to other hybridization

or ensemble methods, as it reduces the generalization error rate. The main drawback is that training and prediction times could become difficult to manage by adding layers.

3.5 Bagging

Bagging [4][13], also called bootstrap aggregating, is a machine learning ensemble meta-algorithm whose goal is to reduce variance of machine learning models, hence increasing stability and accuracy while reducing overfitting.
Before actually speaking of bagging a core concept needs to be introduced: bootstrapping, whose goal is, given a dataset D, to create pseudo-independent subsets D_m maintaining the data distribution. From the full starting dataset of fixed size N, this technique requires to sample with replacement, a fixed number of samples M times, one for each subset.
While theoretically simple, some requirements must be fulfilled in order to let this approach work in a real-world environment. The initial dataset of size N should be large enough in order to capture the underlying data dynamics even under an appropriate subsampling. Second, the number of samples of the original dataset N should be large enough with respect to the times the dataset gets sampled for each subset in order to keep a low correlation among the data. These two requirements falls under the names of representativity and independence.
Once obtained the D_m learning subsets, either with totally independent or pseudo-independent observations, the following step is to train M weak learners, one on each subset. The insight is to learn the complete dynamics of the underlying data by means of several weak learners, then combine their results in order to overcome their weakness and have the accuracy of a strong learner. The technique used to combine these models are averaging in case of regression problems and voting in case of classification problems. Figure 3.1 provides an overview of the bagging process.

3.6 Boosting

Boosting [31][32] is an ensemble meta-algorithm whose main goal is reducing bias. The idea behind this technique is that a set of weak learners can create a single strong learner.
The core concept of boosting is the re-weighting of the dataset. By doing so iteratively after the sequential training of several weak learners through various stages the future learners will apprehend easily the dynamics of the

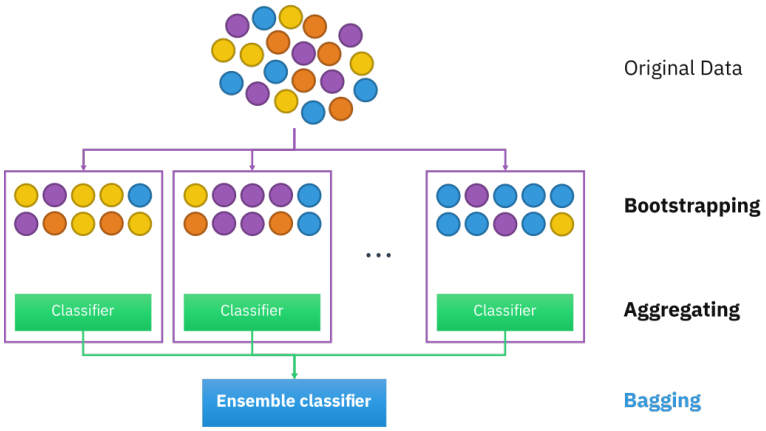


Figure 3.1: Schematic example of bagging.

data that its predecessors weren't able to learn, being more valuable in terms of weight.
The final predictions are obtained by computing the weighted average among all the predictions of the weak learners.
To have a more precise idea of how this works the most important steps are shown below.

1. Weight equally all the train samples in the dataset.

    1.1 Train the weak learner using the weighted samples.

    1.2 Compute the error of the weak learner on the training set.

    1.3 Increase the weight of the mispredicted samples.

    1.4 Repeat until satisfied.

2. Final prediction as the weighted average of predictions of each model.

Algorithm 1 shows the pseudocode of one the most popular boosting algorithms, nonetheless the first one to achieve a great success: *AdaBoost* in its simplest version. The pseudocode is really similar to the description of a general boosting algorithm, but it formalizes the problem giving a more practical insight.

3.7 Gradient Boosting Decision Trees

This algorithm rely on the support of several weak learners in order to increase its accuracy.
A weak learner, is a learning algorithm with error rate sightly less than $\frac{1}{2}$,

Algorithm 1 AdaBoost.M1.

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data x using weights w_i.
   (b) Compute the error on the training samples.
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   where x_i is the i^{th} training sample, and y_i is its true label.
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

so it is an algorithm which performs barely better than a random guessing. A strong learner, instead, is an algorithm that has high recognition accuracy and its execution time is polynomially upper bounded. Gradient boosting decision trees algorithm exploits both bagging and boosting techniques in order to enhance the accuracy by reducing bias and variance. Its aim is to combine many decision trees whose are weak learners in order to get a strong learner out of them.
The overall model $F(x)$ is defined as:

$$F(x) = \sum_{i=0}^{M} \gamma_i h_i(x) \tag{3.2}$$

where x is the input sample, h represents a decision tree, γ is the weight of such tree and M is the maximum number of regression trees.
Let $D = \{(x_i, y_i)\}_{i=1}^{N}$ be the collection of samples of size N which constitute the dataset, $h(x)$ be the weak learner, L the loss function and M the number of iterations, then the general algorithm of a gradient boosting decision trees can be synthesized by the following steps [37].

1. The initial constant value of model γ is given by:
$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma) \tag{3.3}$$

2. For $m = 1$ to M:

   1.1 Compute the so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))}, \text{for } i = 1, 2, .., n \quad (3.4)$$

   1.2 Fit a weak learner $h_m(x)$ to the *pseudo-residuals* using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.

   1.3 Compute multiplier γ_m by solving the following one dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)) \quad (3.5)$$

   1.4 Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \quad (3.6)$$

3. Output $F_M(x)$.

3.7.1 GBDT Implementations

There are several algorithms which implement GBDT algorithm, the ones treated in this thesis are *XGBoost* [8], *LightGBM* [22] and *Catboost* [11]. These implementations in the last years were widely used in challenges and high-end applications [33], [21], [12].
Every model shows some core similarities to the other, as they're based on the same algorithm, but there are important differences among them that justify their use in a hybrid context. Among these there surely are structural differences, as in hyperparameters, in treatment of categorical features and in computational engines.

3.7.1.1 XGBoost

Among the unique characteristics of these model we have the regularization, *XGBoost* in fact has the possibility of using both Ridge and Lasso regularizations independently in order to deal with complex models and reduce overfitting. Another notable characteristic is that it optimizes also the *out-of-core* computation, in order to process efficiently datasets that wouldn't fit in memory [8].

3.7.1.2 LightGBM

One of the greatest differences between *LightGBM* and the other models is that this one implements GOSS (Gradient-based One-Side Sampling) [22] a novel sampling method which down-samples the instances on basis of gradients. This technique retains instances with large gradients, which have the greater training error, while performing random sampling on instances with small gradients, that have a small training error.
LightGBM is lighter in terms of computation, because with its innovative sampling approach it increases the convergence speed of the algorithm.

3.7.1.3 CatBoost

One of the unique characteristics of *CatBoost* is the algorithm used to process the categorical features. The main advantages of this innovative approach are two. These features no longer need to be pre-processed as it is performed out of the box and the model's performance is competitive with respect to the other implementations when categorical features are involved.
Another important characteristic is *ordered boosting*. While other implementations tend to have some problems of overfitting dealing with small datasets, *CatBoost* implements a special modification for such cases. This increases *CatBoost*'s robustness and reliability when dealing with these problems [12].

3.8 Deep Forest

The Deep Forest [41] is a technique based on the stacked generalization insight which also takes inspiration from the *Deep Neural Networks*.
This model is developed on a multi level structure. Every level has a predefined number of generalizers, as described in section 3.4.4. The *level-0* learns directly from the original dataset, while following levels learn from a dataset build aggregating the predictions of the precedent level's models and some features drawn from the original dataset.
To formalize this, given θ_0 which is the original dataset, we have K different generalizers G_{k,θ_0} which learn from it. Defining $y_{k,0}$ the array of predictions made on the training set by one of the k^{th} models at *level-0* and f_{m,θ_0} one of the M features from the original dataset θ_0, then the dataset of *level-1*: θ_1, is constructed as follows $\theta_1 = [f_{1,\theta_0} \dots f_{M,\theta_0}, y_{1,0} \dots y_{K,0}]$. Then every *level-1* generalizer G_{k,θ_1} learns from θ_1. By following this insight it is possible to go on building as many level as required. Once reached the final layer, the last prediction is obtained through the averaging methodology.
This approach tends to build a structure similar to a *Fully Connected Feed*

Forward Neural Network, where neurons are strong ensemble algorithms. All the models on middle layers are trained with an aggregation of the predictions made by the previous level and a set of features extracted from the original dataset, hence the predictions are combined through averaging on the final layer. The structure of this algorithm is schematically represented in figure 3.2
The Deep Forest showed to be competitive in many tasks with respect to neural networks and other machine learning algorithms, however it is strongly computationally demanding, so the cases in which it could be used must be carefully analyzed.

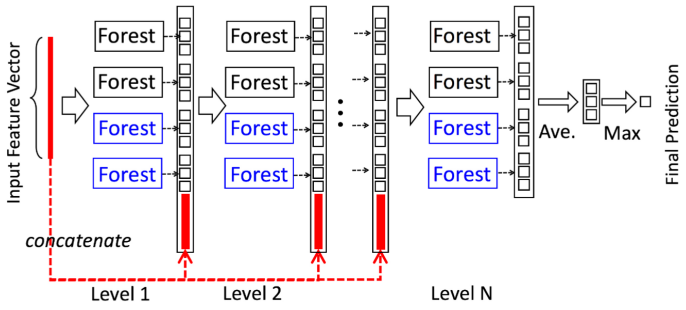


Figure 3.2: Structure of Deep Forest.

3.9 Evaluation

In the field of recommender systems, as in machine learning, choosing how to evaluate the accuracy of the recommendations is a difficult problem. An accuracy metric empirically measures how close a recommender system's prediction for a user is from the user's true preference [7]. Different evaluation metrics are used to optimize different aspects of the problem, so a good metric should be able to model accurately the objective of the problem. When a single metric doesn't model well enough the goal of the problem it is possible to combine two or more of them in order to achieve a better optimization.
In the last years many metrics have been proposed and used quantitatively to evaluate the performance of recommender systems and to this day, due to their nature, there isn't a standard in the field [18].

3.9.1 Precision-Recall Area Under Curve

This metric, also called (PR-AUC) is very useful when classes are highly imbalanced.
In order to compare different classification algorithms and to define metrics, four sets, usually contained in a confusion matrix, are employed. These sets compare the predicted values with respect to the *ground truth*: the target values.

- *True Positive - T_p:* Number of positive predictions that are positive in the *ground truth*.
- *True Negative - T_n:* Number of negative predictions that are negative in the *ground truth*.
- *False Positive - F_p:* Number of positive predictions that are negative in the *ground truth*.
- *False Negative - F_n:* Number of negative predictions that are positive in the *ground truth*.

Precision and Recall are two metrics defined using these sets as:

$$\text{Precision} = \frac{T_p}{T_p + F_p} \tag{3.7}$$

$$\text{Recall} = \frac{T_p}{T_p + F_n} \tag{3.8}$$

Precision, which can be seen as a measures the number of correct identifications over all the positive predictions. Recall instead measures the number of actual positives that was identified correctly.
The precision-recall curve shows the tradeoff between precision and recall for different thresholds. An elevated value of the area under curve, which indicates a good prediction, states that both precision and recall have a high value.
Once computed precision and recall it is possible to calculate the PR-AUC, by calculating the area under curve (AUC) of these two metrics using an approach called trapezoidal rule.

$$\text{PRAUC} = \text{AUC}(\text{Precision}, \text{Recall}) \tag{3.9}$$

3.9.2 Cross Entropy

Cross entropy [9] measures the performance of classification models with probabilistic output, giving a measure of how far is the predicted value from the true label.

$$\mathrm{CE} = -y \cdot \log(\hat{y}) \tag{3.10}$$

Where y is the prediction vector of the model and $\hat{y}$ is the ground truth: the vector containing the actual values to predict.

3.9.3 Relative Cross Entropy

This metric corresponds to the relative improvement of a prediction compared to a naive prediction baseline, measured in cross entropy [2]. Naive predictions can only assume the value p, which expresses the average probability of the positive class, given by:

$$p = \frac{\hat{y_p}}{\hat{y_p} + \hat{y_n}} \tag{3.11}$$

where $\hat{y_p}$ represents the number of *ground truth*'s positive labels while $\hat{y_n}$ the number of the negative ones.
At this point the naive cross entropy CE_{naive} can be defined as:

$$\mathrm{CE}_n aive = -p \cdot \log(\hat{y}) \tag{3.12}$$

While the relative cross entropy (RCE) is defined as follows:

$$\mathrm{RCE} = \frac{(\mathrm{CE}_{naive} - \mathrm{CE})}{\mathrm{CE}_{naive}} \cdot 100 \tag{3.13}$$

Where CE is the cross entropy described computed over the predictions as described in subsection 3.9.2.

3.9.4 Root Mean Squared Logarithmic Error

This metric, also called RMSLE describes a variation of the root-mean-square error, which aims to aggregate the magnitudes of the errors in predictions into a single measure of predictive power. These kind of metrics provide an estimation of the accuracy of the model, in order to have the possibility to compare predictions made by different models.

$$\mathrm{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2} \tag{3.14}$$

Where n is the total number of observations in the dataset, y_i is the prediction made by the model while $\hat{y}$ is the ground truth.

3.10 Optimization

In most of machine learning alorithms there are also external variables that need to be tuned in order to achieve a good performance. They are called hyperparameters and it is important to have a methodology to automatically optimize them.
Many optimization algorithms try exploit some rules in order to search the multidimensional space of the hyperparameters to find the best possible configuration.

3.10.1 Random Optimization

This family of techniques is one of the simplest, yet enough effective to take into account [25]. With this approach the space of the hyperparameters is explored randomly or pseudo-randomly, useful in exploring functions with either more local maxima or minima. Due to its random exploration approach, this technique is particularly useful to deal with non-continuous or non-differentiable functions' optimization.

3.10.2 Bayesian Optimization

This approach provides very good performance if compared with other state of the art techniques, because it is able to balance effectively exploration and exploitation. Bayesian Optimization [34] tries to find the minimum of a function $f(x)$ on some bounded set X. Building probabilistic model for $f(x)$, this optimization technique uses it to pick the points of X in which the function will be evaluated the next time.
Bayesian optimization approach has two main components, the first one is a prior over the function $f(x)$. The most used is the Gaussian process prior. The second one, instead, is the acquisition function, which choose the points to evaluate. The bayesian optimization, in fact, treats the function to optimize as a random function placing a prior over it, which captures the beliefs about that function.
A Gaussian process [26] is a distribution over functions, such that the results of the evaluations made on these functions at arbitrary points, jointly have Gaussian distribution.
Acquisition functions are approaches that guides the hyperparameter space exploration. They predict the mean and the variance of the Gaussian process and combine them according to some criteria that will direct the search towards the optimum.

Algorithm 2 Simple Bayesian Optimization algorithm.

1. **Input**: Data $D_1 = \{(x_i, y_i)\}_{i=1}^{N}$ containing prior knowledge about the function f.

   (a) Choose x_t by optimizing the acquisition function, a, over a Gaussian Process (GP) such that:

   $$x_t = \arg\max_x a(x|D)$$

   (b) Sample the objective function $y_t = f(x_t) + \epsilon_t$

   (c) Augment the data $D_t = \{D_{1:t-1} \cup (x_t, y_t)\}$

   (d) Repeat from point (a) until the maximum number of iteration is reached.

In order to speed up the convergence of this algorithm, it is possible to initialize the prior by using the knowledge acquired from random optimization run in advance. After evaluating several points in the function, the prior is updated according to these results that are nothing less than the posterior distribution over the objective function.

3.11 Troika Architecture

Troika is an approach to solve classification and multiclassification problems based on stacking ensemble [27].
The architecture of this model, shown in figure 3.3, is distributed among four different levels. In the *level-0* the problem's domain is split among several *base-classifiers*. Their predictions are fed in input to the *specialized-classifiers* of *level-1*, each of which focus on a specific task exclusively classifying only a single pair of classes. In this way, being c_i and c_j two distinct classes each *level-1* model predicts a unique vector of complementary probabilities $\{[P(c_i), P(c_j)] \text{ where } i \neq j\}$.
Being N the number of classes in the problem domain, then the exact number of *level-1* classifiers is computed as $\binom{N}{2}$.
At this point it is introduced a layer with the main purpose of learning the dynamics of the previous one. *Level-2* models are called *meta-classifiers*, each of them aims to predict the probability for a single class. A *meta-classifier* with target class c_k, receives inputs exclusively by *specialized-classifiers* whose

predictions are in the form $\{[P(c_k), P(c_j)]$ where $k \neq j$ and $c_j \in C\backslash\{c_k\}\}$. Where $C\backslash\{c_k\}$ is the set containing all the domain's classes but c_k. The *level-3* of the Troika architecture contains only a single classifier, namely *super-classifier*, that simply ensembles the predictions of the previous level producing N predictions vectors, one for each class.
While being slow to train, this structure have been proven to successfully increase accuracy with respect to traditional stacking and surpassed simpler classifiers.

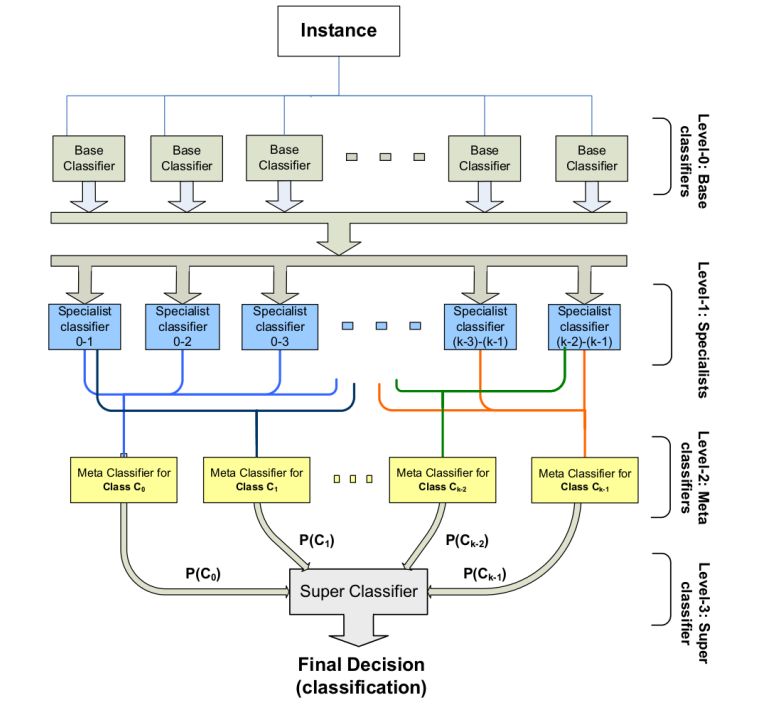


Figure 3.3: Troika ensemble architecture.

3.12 Robust Application of Stacked Generalization

The solution proposed in [35] is an application of stacked generalization on two levels, as shown in figure 3.4, using several algorithms based on decision trees.
In *level-0* four different algorithms make their prediction on the overall training data, using among the others *gradient boosting decision trees*. In *level-1* a the linear regression was used in order to ensemble the predictions of the previous level, generating the final forecast.
This approach predicted the demand on an *e-commerce* site being compet-

itive with respect to simple classifiers. In particular, by training it using only 20% of the original data, its predictions were at least good as the ones provided by simple models trained on the overall dataset. This result is interesting because stacked generalization is proved to be robust and reliable even with a significant reduction in the dataset.

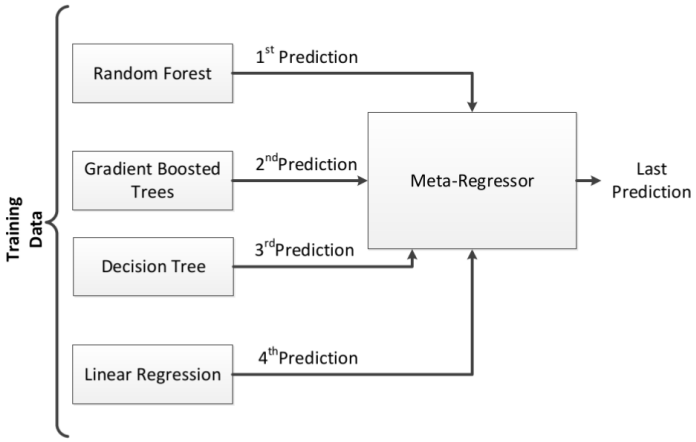


Figure 3.4: Structure used to perform robustly the prediction over the demand of goods in an e-commerce site.

3.13 RecSys Challenge 2020 Winning Solution

This work [33] refers to the winning solution of *ACM RecSys Challenge 2020*. The model employed in the context of this challenge is an implementation of the generalized stacking model on two levels. While the architecture is similar to the one described in section 3.12 there are some notable and interesting differences. All the algorithms used were XGBoost, that they found out to be the best in terms of performance. Moreover the three *level-0* models were trained on different subsets of features and different subsets of samples. While two of these models were validated using the last two days of sampling, the third was validated using two folds that didn't have tweets in common. The strongest point of this solution was to learn different dynamics in the data by manipulating the dataset and by using different strategies of validation in *level-0* models.

3.14 Differentiation Approaches for Ensembles

The review [29] talks about the importance of diversity in the ensemble methods. In fact the employ of diverse models lead to uncorrelated prediction errors, hence, by combining their forecasts, the overall accuracy is increased. There are several approaches that is possible to use in order to increase the diversification, the most used are:

- *Manipulation of the generalizer*:
    1. *Manipulation of the generalizer's hyperparameters*: By changing the hyperparameters of a model, it will learn different dynamics of the underlying data.
    2. *Differ starting point in hypothesis space*: Some inducers can gain diversity by starting the search in the Hypothesis Space from different points. For example the simplest way to manipulate the back-propagation inducer is to assign different initial weights to the network.
    3. *Hypothesis space traversal*: This strategy, similarly to the previous one, tries to provide diversification by acting on the hypothesis space. In particular it tends to differentiate the way an optimization process travels to the hypothesis space through two main strategies: random and collective performance.
        - *Random-based strategy*: This approach simply inject a certain level of randomness in the traveling policy, so two different runs would end up into two different positions.
        - *Collective Performance based strategy*: This strategy exploits the evaluation function, by adding a penalty term that encourages diversity.
- *Manipulating the training samples*: This method aims to train each generalizer on a different variation or subset of the original training set.
    1. *Resampling*: Resampling technique can be resumed in two main approaches, the ones used in bagging and boosting. The first approach is sampling with replacement, while the second is the weighting of the dataset, explained respectively in section 3.5 and section 3.6
    2. *Creation*: This approach relies on the iterative training of models. Each time a model is trained new artificial instances are added

to the original dataset. These samples are generated according to the original distribution, but their target value is chosen in order to differ maximally from their current ensemble prediction.

- *Manipulating the target attribute representation*: This approach aims to split the original problem into many sub-problems. In order to do so many different algorithms are employed. In this way, instead of having a single complicated problem, the various models can specialize on a simpler task. An example may be the classification on K different classes split on among $K - 1$ classifiers.

- *Partitioning*: This technique consists of generating cuts into the original dataset in order to have two or more sub-sets. Then feed the sub-sets to different models. With partitioning every generalizer learns only from part of the data, which is particularly helpful in dealing with complex problems.

    1. *Horizontal Partitioning*: The horizontal partitioning generates different subsets, which maintain the same number of features, but differs in sample. Many approaches can be employed in order to achieve such result, for example cutting horizontally the dataset or a sampling approach.
    2. *Vertical Partitioning*: In this case a vertical cut on the dataset is performed, in this case the number of sample remains the same among all the different sub-sets created, but the number of features in each sub-set is different.

- *Use different types of generalizer in ensembling frameworks*: This strategy, differently from the others suggests to use different algorithms within the ensemble to vary the predictions.

Chapter 4

Environment Analysis

This chapter contains the description of the environment around which the thesis project was developed. This work in fact evolved around the solution of our team, *BanaNeverAlone*, provided for the *ACM RecSys Challenge 2020*, sponsored by Twitter, the famous social network.

4.1 Domain and Problem Description

Twitter is an international social network which allows its users to share their thoughts and information attached with images and link through a channel known as "Tweet". With the large amount of data available nowadays it is very important for this provider to recommend the right content to each user, in order to maximize the interest with respect to the content itself and the number of engagement towards it, hence increasing the pleasantness of the social network usage. Twitter's home timeline, where a model predicts which kind of tweets to display and how to rank them according to the user tastes, is the most likely place where interactions among users and tweets happen. So it is crucial to have the most accurate predictions as possible, in order to enhance the user experience.
Unlike the most common recommendation tasks where it is asked to suggest an item to a user by ranking a list of possible candidates, the 2020's problem was a classification task. More specifically, *user-tweet* pairs were given, with four kind of possible labels *like*, *retweet*, *reply* and *retweet with comment*, and the request was to predict the probabilistic likelihood of each class' interactions.

4.2 Dataset Description

This section provides information and explainations about the dataset used during the challenge and, sampled, for the purposes of this thesis.

4.2.1 Sampling over Time

The dataset provided for this challenge contained roughly 160 millions of samples. The training set's interactions were sampled during an entire week, the test set during the following weeks, while the final test set, in the same way, two week after the training set. The goal of this sampling approach is to have models that learn the weekly invariant dynamics of the data.

4.2.2 Privacy Ensurance

In this competition Twitter decided to focus on the respect of the privacy of its sampled users. The most impactful decisions were to anonymize the data and forbid both the decryption of the encoded tweets and the information gathering about users from its APIs. This policy was reinforced by keeping the dataset continuously compliant with the *General Data Protection Regulation* (GPDR) policy, so if users would have decided to either remove tweets or set them private, or for example to delete their account, all the related data would also be removed from the challenge dataset. Due to this practice several versions of the dataset were released, each with a reduced number of tweets, reaching 121 millions of samples in the last release of the training set.

4.2.3 Imbalance

An important characteristic of this challenge's dataset was the imbalance. In fact not only the dataset was provided with positive and negative samples, but also with pseudo negative ones in order to ensure users' privacy, hence avoiding the possibility of creating a privacy leak. The positive ones were obtained by sampling the effective interactions of tweet account, while the negative could either be true negatives, when the users did see the content and chose to not interact with it, or pseudo-negatives which are tweets the user didn't see at all. The positive samples in the various classes are 43.4% for *like* which is almost a balanced problem, 10.9% for *retweet*, 2.5% for *reply* and 0.7% for *retweet with comment* being the most heavily unbalanced. Hence each class needed to be tackled with some differences.
Moreover among positive and negative samples there also was a high number

of cold users: around 24% of all the samples had a cold user engager (the one who interacted with the tweet). Cold users are customers that are new to the system, hence they contextually have no history and their past interactions cannot be used in order to make recommendations. In this specific task, the cold users are the ones that appear only once in the dataset by making a single interaction with a tweet.

4.2.4 Features Description

The features provided in order to describe the *user-tweet* interaction were split in four main categories each of which described a particular aspect.
The categories were respectively *Tweet features*, which encapsulated the core information about the tweet. *Engaged With User Features* which were the most relevant information about the tweets' creators account. *Engaging User Features*, having the same content as the previous category, were act to describe the accounts of the interacting users. The last category, *Engagement Features* described the interaction itself. Where a feature for each class had either a timestamp, in case of positive interaction or a null-value in case of a negative one. The timestamp indicated the moment when an interaction occurred.
Table 4.1 shows the names of these features and their related category.

4.3 Feature Engineered Dataset

Feature engineering was an important step in the development of the project. It permitted, in fact, to achieve competitive results by extracting and making explicit some hidden dynamics of the dataset. In this way we obtain around 150 useful features. They were generated following two distinct policies.
Timestamp-aware, which took into account only information prior to the interaction and *cumulative*, which, by also taking into account the "future" information, generated the time-independent features.
The obtained user-related features can be split in three main categories:

- *Number of Active and Passive Engagements.* This feature group takes into account: both the active and passive engagements for each class and also the positive interactions made and received on tweets. These features, by considering a class at a time, count the number of engagements, in each of the previous cases for all the users.
  The active users' features model how likely the users are to interact with tweets, on the other hand the passive users' features model how popular the users are.

Table 4.1: List of Twitter dataset's features.

Tweet Features	Text tokens Hashtags Tweet id Present media Present links Present domains Tweet type Language Timestamp
Engaged With User Features	User id Follower count Following count Is verified? Account creation time
Engaging User Features	User id Follower count Following count Is verified? Account creation time
Engagement Features	Engagee follows engager? Reply engagement timestamp Retweet engagement timestamp Retweet with comment engagement timestamp Like engagement timestamp

- *Number of Engagements with Language/Hashtag.* This features group counts the number of previous engagements a user had with respect to tweets of the same languages, or similarly to tweets containing a specific hashtag.
  While the features related to the languages model the likelihood of users to interact with certain languages, the ones related to the hashtag provide a semantic indicator of users' interests.

- *User Similarity.* By means of an undirected graph, where nodes represent users and edges engagements, this group of features expose the similarity among users. The same edge captures active engagements happened among two users it is connecting and its weight is equal the sum of the number of these engagements.

The similarity is expressed for directly connected users and users connected by two edges. In both cases it is represented by both a binary value that tells if the users are connected or not and the overall weight of the path that connects them.

Tweet features, instead, were processed in two different ways. In order to use them in a neural network we generated text embeddings using *DistilBERT*, as described in section 2.2. While to use them in *Gradient Boosting Decision Trees* we created specific features from the tokens. Their purpose was to represent some important text information usable by GBDT models. The most important are:

- *Unique Word Frequency.* This features consider how many times users wrote unique words: vocables used only once. It is computed by concatenating all text the user wrote in previous tweets, hence calculating the ratio between the number of unique tokens and the total number of tokens. It can be useful to identify bots and recurrent patterns.
- *Tweet Topic.* We identified some trend topics in the text and manually associated each of these to a list of most used words. Then, for each tweet, we counted the number of words of a certain topic it contains.

4.4 Optimization and Metrics

The hyperparameters tuning was performed with a combination of both Random and Bayesian Optimization, by using the scikit-learn library.
In order to perform the tuning, a custom metric to maximize was designed, as the one used to rank teams in the ladder couldn't be replicated locally.
This new metric is a nonlinear function dependent on the two score metrics, PRAUC and RCE combined as follows.

$$obj(\text{PRAUC}, \text{RCE}) = \begin{cases} \text{RCE} \cdot \text{PRAUC} & \text{if RCE} > 0 \\ \frac{\text{RCE}}{\text{PRAUC}} & \text{otherwise} \end{cases} \tag{4.1}$$

There were several problems in finding good combination rules. The first one was the diversity in the range of possible values of the two metrics. While PRAUC is between 0 and 1, RCE varies from $-\inf$ to 100. By having two such different ranges, it becomes crucial to scale the metrics in a correct way to give the right importance to each of them.
The second difficulty arose by noticing a flaw in PRAUC metric. By always predicting 0, in fact, it had a high score, hence when this prediction had too much weight in the combination all the predictions tended to shrink towards

zero. This, however, lowered significantly the score of the RCE, leading to an overall bad solution.
While a possible solution could be to evaluate only RCE, since being not flawed a raised in this metric lead to a consequent raise in PRAUC, the objective would be less representative of the problem. Our solution instead addressed the two difficulties differently. It was about giving less importance to PRAUC in the combination of the two metrics, in such a way that the RCE would drive the optimization. In order to do this it wasn't necessary to scale the metrics, because with their range values most of the times we had that $|\text{RCE}| \gg \text{PRAUC}$. Then we found the nonlinear function 4.1, where by respectively multiplying and dividing RCE by PRAUC the smallest metric acted as penalty, letting the bigger one leading the optimization.

Chapter 5

Experimental Evaluation

Nowadays most of the competitive models in recommender systems are built using hybrid approaches. A method of particular interest is the stacking ensemble, described in section 3.4.4. This technique, by combining several algorithms, is in general more reliable and accurate with respect to the single models. Since the usage of this architecture in combination with the *gradient boosting decision trees* (GBDT), in the last years became a common solution in challenges and high-end applications, arose the necessity to further increase their reliability and accuracy even with imbalanced datasets.
By exploiting the environment described in chapter 4, here are provided an analysis of such problem and a set of possible solutions.

5.1 Problem Analysis

To perform stacking ensemble with GBDT models, there are several design choices to take into account which lead to total different implementations. Many of the solutions in literature try to find a tradeoff between the accuracy and the complexity of the ensemble. The first choices that need to be made regards its architecture. The number of layers to employ and the number of models for each layer, in fact, vary a lot its complexity and consequently its training time. Another important choice regarding the architecture is how to connect two adjacent levels. In fact there are a lot of possibilities and variations to do so. For example it is possible to aggregate the predictions and feed them directly to the next level, to aggregate not only the predictions, but also the original features or employing a strategy where only a subset of predictions are fed to each model as in the solution presented in section 3.11.
Once designed the architecture another fundamental choice is, which imple-

mentations of GBDT to use. This choice needs to be careful as these models may learn different dynamics from the same dataset. The most popular implementations of GBDT, also used in this work, can be found in section 3.7. Another possibility to use in order to enhance the accuracy of the ensemble is the partitioning of the dataset, according to the techniques described in section 3.14. Then a consequent problem is which model train with which sub-set.
These choices need to be made in order to have a robust approach able to make accurate predictions in both balanced and imbalanced problems. In a way the task performed in this chapter can be seen as an optimization problem where the accuracy has to be optimized for three different problems in function of the datasets, the choice of implementations and the design parameters of the architecture. In order to look for good solutions it is important to balance exploration and exploitation.

5.2 Environment Setup

5.2.1 Sampling

Due to the huge dimension of the challenge's dataset, for the purposes of this thesis a reduction was performed. In order to maintain the distribution of both the samples and the labels, the dataset was sampled according to a uniform distribution. The sampling ratio was chosen in order to significantly reduce the dimension of the dataset, while keeping enough instances to perform valuable predictions. A fitting value to do so was 0.02, hence picking a sample every fifty. As may be seen from table 5.1, despite the strong sampling, the distribution of the samples among the classes it has been about maintained.
This procedure was replicated on each of the available set: the training set, the test set and the final test set.

Table 5.1: Percentage of positive interactions' samples in the original and sampled dataset.

Label	Original	Sampled
Like	43.4%	41.2%
Retweet	10.9%	9.5%
Reply	2.5%	2.4%

5.2.2 Models

The choice of the models to use fell on three different implementations of GBDT: XGBoost, LightGBM and CatBoost. Despite CatBoost wasn't used in the solution provided for the challenge, it is a popular and performing implementation which is worth using in the context of these experiments. The main motivations behind the use of this family of algorithms are their elevated accuracy, their robustness with respect to the noise in the data and their capability to work on non-normalized data.
The reason why three different implementation were chosen, instead, resides in some of the motivations explored in section 3.14. Despite all belonging to the family of algorithms, GBDT, they have different internal dynamic and different hyperparameters, which justifies the diversity among their predictions.

5.2.3 Optimization and Early Stopping

The optimization in this case was used in order to find a good hyperparameters configurations to enhance the performance of the models constituting the ensemble. It is performed, for each of them, using a combination of random and Bayesian optimizer, which are described in section 3.10. The number of points evaluated for each technique was respectively 15 for the random and 5 for the Bayesian, giving more importance to the exploration of new hyperparameters configuration.
Each model had a different set of hyperparameters to optimize, they were: 13 for XGBoost, 11 for LightGBM and 9 for CatBoost.
In order to deal with the overfitting problem, the models, during the optimization phase, were trained with the support of early stopping. This technique consists in training a model iteratively and test its performance on an apposite validation set. The number of iterations is then incremented until the error on the validation set don't decrease for an arbitrary number of rounds.
While the test set was used to perform optimizations, the final test set for evaluating the accuracy of the ensemble and the training set to make the models learn, it was necessary to create an apposite validation set for the purpose of early stopping. This new set was created by cutting the last 10% samples from the training set, which was about ten times bigger than the other sets, with the purpose of making it similar to them, in terms of dynamics and dimensions.

5.2.4 Cross Validation

In order to treat a model's predictions as features for the subsequent level models, the use of cross validation was crucial. In fact in order to do so, it is necessary to have predictions available for every set. It is not a big problem when dealing with validation and test sets, but predictions made on the training set are they are highly subject to overfitting. To solve this a *leave K out approach* was employed: the training set was divided in five chunks of almost equal dimension by generating cuts every 20% of samples. The model was then trained using only four of them of while making predictions on the fifth. By repeating this process on every chunk, reliable predictions for the test set are available.

5.2.5 Features and Ensemble Strategy

In order to perform these experiments, almost all the features from the challenge solution's dataset were employed, using 123 of them. Two feature division strategies are used in this work.

5.2.5.1 Category Split

The first approach aim to split them by category generating five completely different subsets:

- *Raw*: These features are the ones coming from the non processed dataset provided by Twitter describing both the creator of the tweet and its *engager*.
- *Tweet*: In this category there are the ones which provide information about the tweet, from its most relevant metadata to the topic detection.
- *Engagement*: These features, describe the interactions and the dynamics among the users.
- *Creator*: Creator features are the ones that describe the users who created the tweets, by means of an analysis on interactions given and received.
- *Engager*: This last group is like the previous, with the difference that the analysis is performed on the user that interacted with the tweet.

Each of these categories represent a different aspect of the problem, while maintaining enough information to be a representative subproblem.

5.2.5.2 Relevance Split

The second division, as long as the ensemble strategy is bound to a preliminary analysis performed on the sampled datasets. An XGBoost model was trained and a features analysis was performed. By doing this, we found that using only 60% of the features the accuracy of the models didn't almost suffered any reduction. In light of this, the idea of the second split configuration came out. It was about maintaining the 60% core features and split the remaining equally, in a random way, to the other subsets.
This idea is also used to build the datasets to feed to the meta-models of each ensemble. In fact, on the last layer, the predictions are aggregated to a dataset containing the 60% core features. This is motivated by a computational speedup and by the possibility for the model to focus more its learning on the second-last layer predictions. This dataset is then fed, in every ensemble, to the last layer, where an XGBoost model is trained on it and makes the final predictions. The choice of using XGBoost implementation was due to the high accuracy shown during the challenge [33] [14].

5.3 Experimental Design

In this section are shown the proposed solutions and the ideas behind them. The experiments are divided into two categories. The first one shows differences among different data partitioning approaches, while the second one shows different strategies employed in the architecture. These experiments are then replicated three times for the labels *like*, *retweet* and *reply*. Due to the different balance of the samples, predicting each of these three target classes is a different problem. The distribution of positive, negative and pseudo-negative samples on the reduced dataset, for each class, it is almost equal to the original, due to the uniform sampling approach.

5.3.1 Dataset Partitioning Experiments

Here is presented the first slot of experiments. The ensemble used for these ones is one of the most common in literature: the two level stacked generalization. Applications of this model can be seen both in section 3.12 and in section 3.13.
The idea is to propose different solutions based on dataset partitioning and different choices of models, based on this architecture.

5.3.1.1 Full Dataset with Different Models

This first experiment's structure is developed on two levels. In *level-0* there are three different models: XGBoost, LightGBM and CatBoost, while in *level-1*, there is an XGBoost model that performs the ensemble as shown in figure 5.1.
Being the full dataset $\mathcal{D} = \{f_i, y_i\}_{i=1}^{n}$ of n samples, where f_i is the set of the features of this dataset, y_i its target labels, the models on the first level are trained using $\mathcal{D}$. Given their predictions p_x, p_l and p_c, where the subscript it is the initial of the model that generated it. Being $\mathcal{D}_{60\%} = \{d_i, y_i\}_{i=1}^{n}$, where $d \subset f$, a subset of the original dataset containing the same amount of samples of $\mathcal{D}$, but 60% of the most important features, the predictions are aggregated with it. Then the dataset on which the *level-1* model is trained is $\mathcal{D}_{ens} = \{[d_i, p_{x,i}, p_{l,i}, p_{c,i}], y_i\}_{i=1}^{n}$.
The goal of this experiment is, given that GBM are strong learners, to test their performance in an ensemble while using different implementations of the algorithm trained on the same complete data.

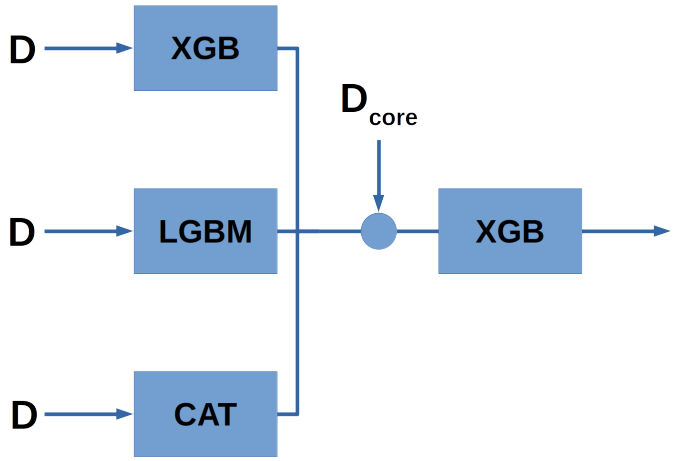


Figure 5.1: Full dataset with different models, solution layout. Where XGB represents the XGBoost implementation, LGBM represents LightGBM and CAT represents CatBoost.

5.3.1.2 Dataset Split with Same Models

The structure of this experiment is also developed on two levels. On *level-0* there are 5 models, all of them are the same implementation of the XGBoost algorithm. On *level-1* instead, there still is an XGBoost ensemble model.

This whole architecture is shown in figure 5.2.
The full dataset $\mathcal{D} = \{f_i, y_i\}_{i=1}^{n}$, this time is split into 5 subsets each containing different features, but the same amount of samples $\mathcal{D}_j = \{h_{i,j}, y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$. Being h_j a subset of features, notice that $h_j \subset f \ \forall j \in [1, \ldots, 5]$ and, having two different subsets, $h_j, h_k \subset f$ where $j \neq k$, then $h_j \cap h_f = \emptyset$. The subsets' features categories were: raw, tweet, engagements, creator and engager. Using them, five different XGBoost implementations were trained in *level-0*, with predictions: p_r, p_t, p_{et}, p_c, p_{er} (the subscripts *et* and *er* represents respectively forecasts made by models trained on engagement and engager datasets). These features were aggregated to $\mathcal{D}_{60\%}$ the same way of the previous experiment in order to generate $\mathcal{D}_{ens} = \{[d_i, p_r, p_t, p_{et}, p_c, p_{er}], y_i\}_{i=1}^{n}$ to train the *level-1* model. The goal of this experiment is to evaluate the goodness of this split approach, without having differentiation in the used models.

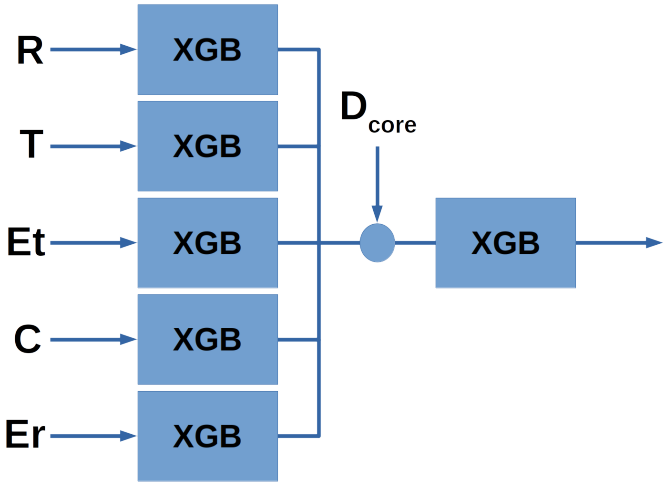


Figure 5.2: Dataset Split with same models, solution layout. The labels indicate the group of features used.

5.3.1.3 Dataset Split with Different Models

This third experiment is very similar to the one presented in section 5.3.1.2, maintaining the same splits of the dataset, and the same number of models, the only difference was the differentiation in the *level-0* algorithms as can be seen in figure 5.3.
The subsets attributes' categories were: raw, tweet, engagements, creator and engager. The distribution of the datasets among the three kind of GBDT

is the following: two XGBoost implementations are trained respectively using raw and creator subsets, LightGBM using tweet and engager ones, while CatBoost was trained using the engagement subset. The choice of which models to use twice is to accredit to their popularity and their proved efficiency. The aggregation process and the generation of $\mathcal{D}_{ens}$ is the same as the second experiment. The goals of this experiment are two, to analyze the relative difference in terms of metrics from the previous experiment and to see if more in general, this split approach combined with diversification in the models, performs well.

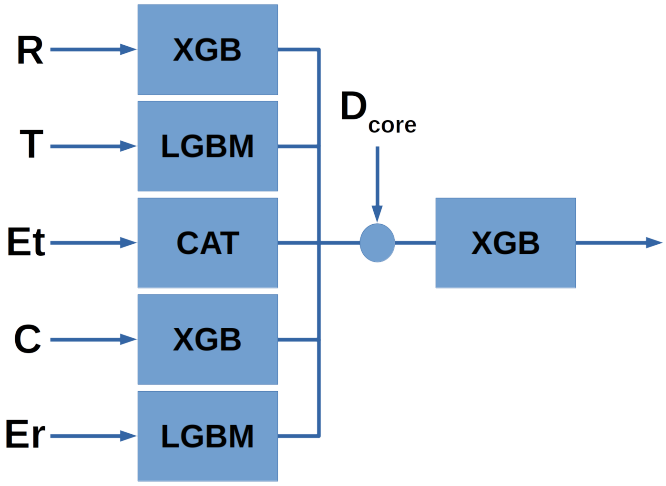


Figure 5.3: Dataset split with different models, solution layout. The labels indicate the group of features used.

5.3.1.4 Dataset Split with Common Features and Same Models

This fourth experiment is employs the same structure and the same models as the one in section 5.3.1.2, but it differs in the dataset splits.
The dataset is still divided in five different subsets maintaining the same number of samples, but this time not all features are different.
Given the $\mathcal{D}_{60\%} = \{d_i, y_i\}_{i=1}^{n}$, we have that d, the 60% most performing features, are present in all the subsets. The remaining $f \setminus d$, instead, are assigned, in a random way, equally to all the splits.
In this case the dataset split can be formalized as follows. The full dataset $\mathcal{D} = \{f_i, y_i\}_{i=1}^{n}$, is still split into 5 subsets $\mathcal{D}_j = \{h_{i,j}, y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$ with the same amount of samples. Differently, being h_j a subset of features, notice that $h_j \subset f \ \forall j \in [1, \ldots, 5]$ and, having two different subsets, $h_j, h_k \subset$

f where $j \neq k$, then $h_j \cap h_f = d$.
The aggregation phase, hence the *level-1* model training are equal to the second experiment. The complete layout of the experiment can be seen in figure 5.4. The goal of this experiment is to test another splitting approach, on a configuration that doesn't have model diversity. This time instead of dividing the problem in four uncorrelated sub problems, maintaining the core information on each split.

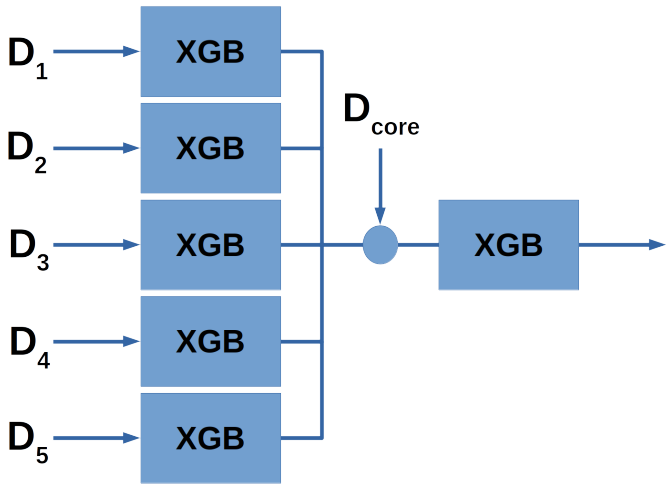


Figure 5.4: Dataset Split with Common Features and Same Models, solution layout. Due to the impossibility to distinguish the datasets by their content, they are numbered.

5.3.1.5 Dataset Split with Common Features and Different Models

This fifth experiment is mostly equal to the one in section 5.3.1.4.
The only difference resides in models that are employed in *level-0*, in fact, it makes use of all the three implementations, with two subsets assigned to XGBoost, two assigned to LightBGM and one to CatBoost.
The overall layout can be seen in figure 5.5. The goals of this experiment are to test the difference in accuracy with respect to the previous by adding the differentiation in the models, and in general if this strategy is a good one.

5.3.2 Architecture Experiments

This second category of experiments focus more on exploiting different kind of architectures.

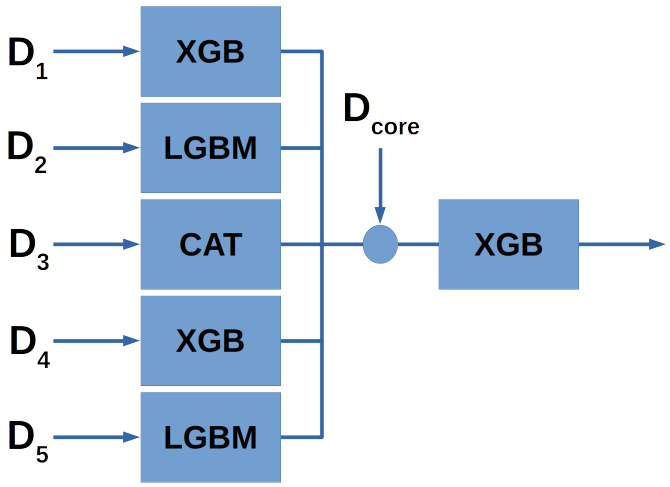


Figure 5.5: Split dataset with different models, solution layout.

The proposed architectures all evolve around a *base architecture*, the one proposed in section 5.3.1.3. They use the same dataset split, because of the insight of section 3.14: raising the ensemble accuracy by increasing the diversity in its models. Also following this idea, all the implementations of GBDT treated in this thesis are employed. The choice fell on this *base architecture* in order to test not only the effectiveness of the proposals, but also how the differentiation affects the ensembles.
The proposed architectures were mostly inspired by the works presented in section 3.8 and in section 3.11.

5.3.2.1 Three Layer Fully Connected

This solution expands the base structure by adding an intermediate layer having the same five models of the first one.
Level-0 is the same as the base structure, the first difference resides into its prediction aggregation. To feed them to the *level-1* layer, in fact, they are aggregated to each of the *level-0* subsets $\mathcal{D}_j = \{h_{i,j}, y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$.
The predictions made by the *level-0* models $p_{0,r}$, $p_{0,t}$, $p_{0,et}$, $p_{0,c}$, $p_{0,er}$ are aggregated to each $\mathcal{D}_j$ resulting in: $\mathcal{D}_{1,j} = \{[h_{i,j}, p_{0,r}, p_{0,t}, p_{0,et}, p_{0,c}, p_{0,er}], y_i\}_{i=1}^{n}$ $\forall j \in [1, \ldots, 5]$.
The strategy used to choose which *level-1* model employ for each $\mathcal{D}_{1,j}$ subset is to keep the correlation *group of features - implementation*. For example, in case $j = 4$, if $\mathcal{D}_4$ was used to train an XGBoost implementation then $\mathcal{D}_{1,4}$ will also be used to train XGBoost.

To clarify, the $\mathcal{D}_{1,j}$ subsets are used as follows. The ones containing the raw features and the creator features are used to train XGBoost models, the ones containing the tweet features and the engager features were used to train LightGBM models while the one containing the engagement features to train CatBoost. As in *level-0*.
The predictions made by the *level-1* models are then aggregated with the dataset containing the core features as in the base structure and fed to the XGBoost model at *level-2* in order to make the final forecasts.
The architecture presented may be seen in figure 5.6. The motivation behind this experiment is the will to test the performance of a multilayered structure with a strong differentiation in the datasets and a fully connected aggregation strategy of the predictions.

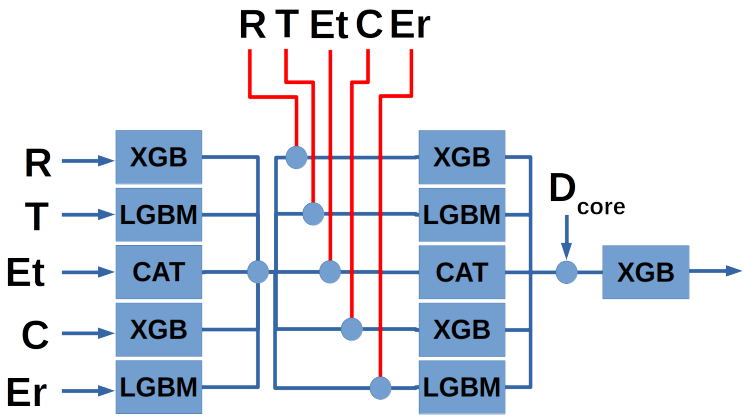


Figure 5.6: Three layer fully connected, solution layout.

5.3.2.2 Four Layer Fully Connected

This solution evolves the previous, adding two intermediate layers, instead of one, a schematic example of which is shown in figure 5.7.
The structure dynamics are the same, but it is important to specify how the *level-2* dataset are formed. In fact, the *level-1* predictions $p_{1,r}$, $p_{1,t}$, $p_{1,et}$, $p_{1,c}$, $p_{1,er}$ are aggregated to the $\mathcal{D}_{1,j}$ datasets, resulting in: $\mathcal{D}_{2,j} = \{[h_{i,j}, p_{0,r}, p_{0,t}, p_{0,et}, p_{0,c}, p_{0,er}, p_{1,r}, p_{1,t}, p_{1,et}, p_{1,c}, p_{1,er}], y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$.
The number and the kind of models for each level are maintained such as the datasets assignment strategy.
The goal of this experiment is to look for an increment in terms of accuracy

increasing the ensemble complexity by means of more layers.

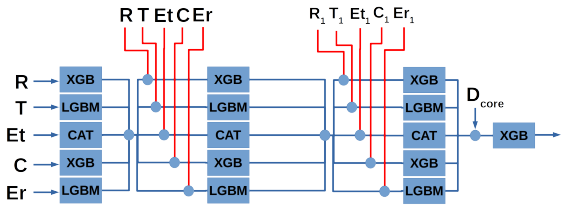


Figure 5.7: Four layer fully connected, solution layout. R_1, T_1, Et_1, C_1 and Er_1 are the original subsets with aggregated all the predictions made by level-0 models.

5.3.2.3 Three Layers Circularly Connected

This experiment uses the same structure of the one presented in section 5.3.2.1, but it varies the predictions are combined to form the *level-1*. Its architecture is the same shown in figure 5.6, where, in this case, the red node implements the circular aggregation strategy.
This solution, in fact, implements a circular distribution of the features. This strategy relies on the fact that, with this split of the dataset, the subsets generated to train the models of intermediate level maintain their characterizing features. When generating the training set for these levels, instead of aggregating a feature to every available set, only one is picked to be attached to every dataset using the circular approach. Consider the *level-0* models predictions: $p_{0,j}\ \forall j \in [1, \ldots, 5]$. Given the *level-0* training sets $\mathcal{D}_{0,j} = \{h_{i,j}, y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$ then the *level-1* datasets are built as $\mathcal{D}_{1,j} = \{[h_{i,j}, p_{0,(j+1) \mod 5}], y_i\}_{i=1}^{n} \forall j \in [1, \ldots, 5]$. In general, datasets to feed to *level-k*, with $k \geq 1$, are defined as $\mathcal{D}_{k,j} = \{[\mathcal{D}_{k-1,j}, p_{k-1,(j+k) \mod 5}]\} \forall j \in [1, \ldots, 5]$.
To better understand this technique let us consider the feature categories ordered in a circular list as shown in figure 5.8.

The order of the attachments is the one shown in the figure. The names in the linked list refer to the subsets holding that group of features. Then,

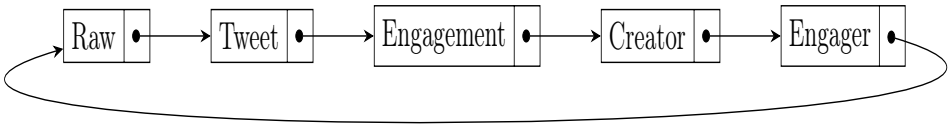


Figure 5.8: Subset aggregation order with circular strategy.

the predictions made on one of these subsets, are initially attached, in *level-0*, to the subsequent dataset in the list. Then, by adding layers, the order is followed.
To provide an example, tweet predictions are aggregated to the engagement subset in *level-0*, in *level-1* they are attached to the creator subset, ending up to be attached to the tweet subset in *level-4*.
It is important to notice that, as in experiments numbers 6 and 7, the predictions made by a certain level are aggregated with the datasets used to train such level. For example predictions made by *level-0* are aggregated with the dataset to train the models of that level, in order to provide a training set for *level-1*. This obviously results in an increase of features over layers.
The other details of this solution are the same of the sixth experiment. The goal of this solution is to test a different prediction aggregation technique on a multilayer structure and to compare the results of this approach to the ones obtained from fully connected strategy.

5.3.2.4 Four Layers Circularly Connected

This experiment combines the architecture of the one presented in section 5.3.2.2 with the aggregation strategy of section 5.3.2.3, resulting in a four layer architecture connected circularly. A panoramic of this architecture appears in figure 5.6, where, also in this case, the red node implements the circular strategy.
The goal of this solution is to check the relative difference from the eighth experiment in terms of accuracy, testing, in case of a weaker aggregation approach, the variation of accuracy by increasing the ensmeble complexity.

5.3.2.5 Persistent Backpropagation

The tenth solution uses the architecture of section 5.3.1.3, assumed to be the base structure, without adding any additional layer. The only difference is that a particular form of backpropagation is implemented.
The base structure produces its final solution, then, at this point, instead of taking them as ensemble solution they are aggregated with each of the initial

level-0 subsets, then the whole ensemble is re-trained and process repeated as many times as desired.
We called this technique *persistent backpropagation*, because commonly the feedback provided by backpropagation is used to update values, while in this case it is used as a new source of information which "persist" through the various iterations.
The structure layout can be seen in figure 5.9.
The goal of this approach is to test this new technique on different numbers of iterations, in order to see if it could lead to an increment in accuracy.

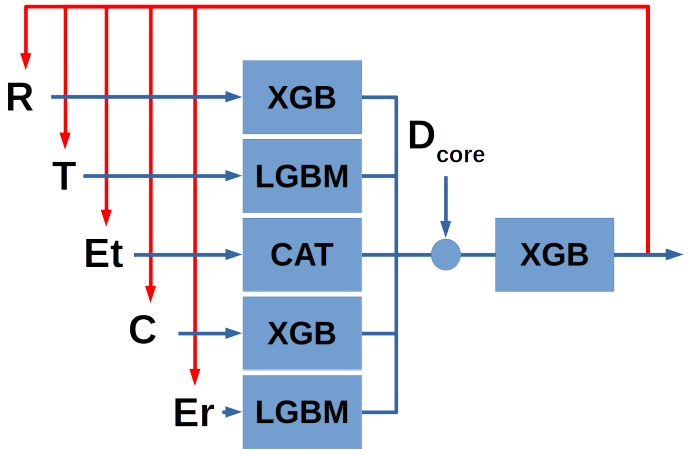


Figure 5.9: Backpropagation model, solution layout. The red line attaches the result produced by the level-1 model to the level-0 datasets.

5.3.2.6 Multiple Persistent Backpropagation

This last experiment uses the structure and the insight of the previous one, but instead of backpropagating only the final predictions, each prediction made by each level of the ensemble is attached to each *level-0* subset. Its layout can be seen in figure 5.10
This last structure's goal it is to see if, by backpropagating more information there is an increment or a decrement, in terms of performance, in the ensemble.

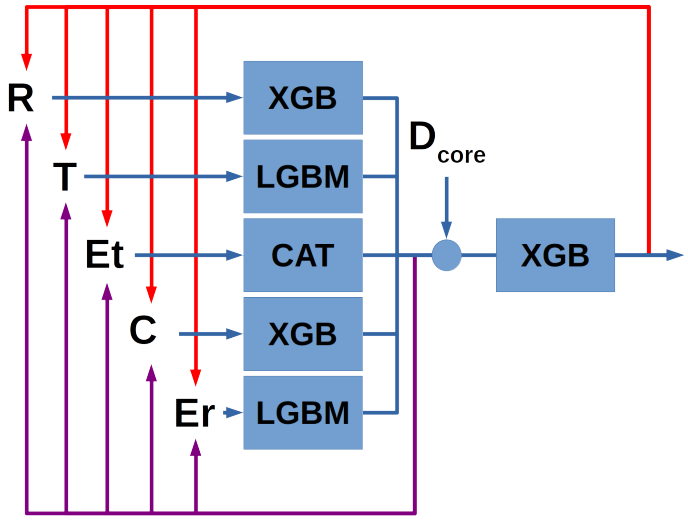


Figure 5.10: Multiple ackpropagation model, solution layout. The red line attaches the result produced by the level-1 model to the level-0 datasets. The violet one instead attaches to those datasets each prediction made by level-0 models. 5.9

Chapter 6

Results and Considerations

This chapter illustrates the results of the experiments which are commented in dedicated sections. These solutions were tested on three out of four different labels of the original problem: *like*, *retweet* and *reply*.

6.1 Data Partitioning Results

In this section we show the outcome of the data partitioning experiments. All of them exploit the same architecture in order to be more comparable: they use the stacked generalization approach, which is presented in section 3.4.4, implemented on two levels.
Table 6.1 shows the results of the data partitioning experiments for each label. In this table we show RCE and PRAUC values along with the value of the objective function in order to provide a more impactful comparison among the solutions' outcomes. For each label it is possible to see an emphasized row with the caption *Challenge Solution*. These rows provide the performance of the models used by our team in the challenge, which details are discussed in section 2.2. For the sake of comparability, such models are trained on the same information as the proposed experiments, hence without the data processed by the neural network. Without this information the models had weaker results, being surpassed, in terms of objective function, by every proposed solution.
It is possible to observe that for each label the scores aren't too far from each other. This is due to the robustness of both GBDT algorithms and stacked generalization technique, that trained on a dataset with a large amount of features generated in every case reliable predictions. The three classes, as described in section 4.2.3, held different levels of imbalance. In table 6.1 can

be seen that the more the classes are imbalanced the more the difference among the scores and the scores theirself are in general lower. This is because, reducing information about the positive interactions makes harder to learn their dynamics, reducing the learning ability of models.
Another interesting insight that is possible to infer from table 6.1 is that not always increasing the diversity in the ensemble by providing different dataset splits or using different models lead to an increase in performance. In fact the *Dataset Split with Different Models* solution, which can be considered the one with the most diverse ensemble, didn't have an outstanding outcome. On the other hand, an ensemble that trains its *level-0* models on all the information: *Full Dataset with Different Models*, provided the best predictions for both *like* and *retweet* classes. Another experiment that follows this dynamic *Dataset Split with Common Features and Same Models*, which, for every label, scored the second best objective function maintaining stability over the different imbalance of the classes. The motivations behind the success of this last experiment, presented in section 5.3.1.4, are to find mostly in the combination of the architecture with five *level-0* models, combined with a feature solution that is similar to the full dataset. The reasons for its high performance in the *like* and *retweet* classes, instead, are attributable to its training set's choice of features, regarding its high accuracy on the *reply* class however there is another reason. In fact the architecture used for this experiment works really well with such level of imbalance. This theory is reinforced by the scores of the *Dataset Split with Same Models* solution, which employs the same architecture, but different training sets. It is also able to provide the best predictions for the *reply* label.
We can state that for the more balanced classes: *like* and *retweet* the solutions that perform better are the ones with a larger amount of features in the training set, while for the most imbalanced increasing the diversification in the training data is a winning approach.
The *Dataset Split* technique used in both section 5.3.1.2 and 5.3.1.3 didn't have an outstanding performance. This was probably due to the incapability of the ensemble model to fully catch the information divided among the base models. Such approach had however good results in *reply* class. In fact with enough imbalance in the problem an incisive split strategy, which for balanced classes resulted in a loss of performance, was able to focus more on the hidden dynamics of the data, resulting in an increase in performance.

Table 6.1: Performance of the dataset partitioning experiments. The comparison shows the PRAUC and RCE metrics along with the objective function that combines them.

LIKE	PRAUC	RCE	**obj**
Challenge Solution	*0.6357*	*11.6899*	***7.4312***
Full Dataset with Different Models	0.6609	14.0137	**9,2616**
Dataset Split with Same Models	0.6447	12.2785	**7,9159**
Dataset Split with Different Models	0.6456	12.2709	**7,9220**
Dataset Split with Common Features and Same Models	0.6599	13.7938	**9,1025**
Dataset Split with Common Features and Different Models	0.6574	13.6369	**8,9280**
RETWEET			
Challenge Solution	*0.2786*	*10.7851*	***3.0047***
Full Dataset with Different Models	0.2910	11.7686	**3,4246**
Dataset Split with Same Models	0.2874	11.4228	**3,2829**
Dataset Split with Different Models	0.2873	11.3922	**3,2729**
Dataset Split with Common Features and Same Models	0.2911	11.7643	**3,4245**
Dataset Split with Common Features and Different Models	0.2910	11.7471	**3,4184**
REPLY			
Challenge Solution	*0.0740*	*8.3411*	***0.6172***
Full Dataset with Different Models	0.0782	7.9840	**0,6243**
Dataset Split with Same Models	0.0768	8.4168	**0,6464**
Dataset Split with Different Models	0.0763	8.1415	**0,6211**
Dataset Split with Common Features and Same Models	0.0778	8.2200	**0,6395**
Dataset Split with Common Features and Different Models	0.0780	7.9264	**0,6182**

6.2 Architecture Results

In this section are presented the results obtained from the experiments which exploited the architecture of the ensemble.
Table 6.2 shows the outcomes in terms of RCE, PRAUC and objective function of the various experiments. Also in this table there are emphasized rows which represent a baseline that models have to surpass to be actually considered good. The baseline is the outcome of the *Dataset Split with Different Models* shown in table 6.1, because all the experiments evolve around this ensemble which is presented in section 5.3.1.3. The reasons of this choice, were discussed in section 5.3.2.

As can be seen in table 6.2, even by varying the architecture, the general considerations on the results made in section 6.1 still hold. Among these experiments the one that is always able to surpass the baseline is the *Persistent Backpropagation* one. Its goodness it is hence independent from the imbalance of the class, having stability and a considerable increase in accuracy. Of particular interest are its scores for the *reply* class. The objective function in this case stands out as it is significantly higher with respect to the others. This solution, in fact, was able to replicate the strategy our model used in the context of the challenge: slightly lowering the RCE as it considerably

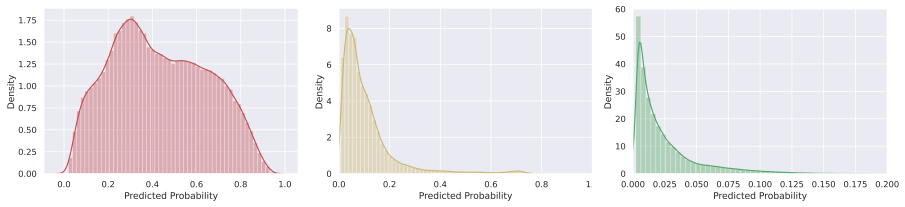
Table 6.2: Performance of the architecture experiments. The comparison shows the PRAUC and RCE metrics along with the objective function that combines them.

LIKE	PRAUC	RCE	**obj**
Challenge Solution	*0.6357*	*11.6899*	***7.4312***
Dataset Split with Different Models	*0.6456*	*12.2709*	***7,9220***
Three Layer Fully Connected	0.6448	12.1284	**7,8203**
Four Layer Fully Connected	0.6448	12.1813	**7,8545**
Three Layers Circularly Connected	0.6466	11.4338	**7,3930**
Four Layers Circularly Connected	0.6477	12.4754	**8,0803**
Persistent Backpropagation	0.6513	12.5149	**8.1509**
Multiple Persistent Backpropagation	0.6489	12.1593	**7,8901**
RETWEET			
Challenge Solution	*0.2786*	*10.7851*	***3.0047***
Dataset Split with Different Models	*0.2873*	*11.3922*	***3,2729***
Three Layer Fully Connected	0.2845	11.1975	**3,1856**
Four Layer Fully Connected	0.2847	11.1609	**3,1775**
Three Layers Circularly Connected	0.2848	11.0822	**3,1562**
Four Layers Circularly Connected	0.2864	11.1858	**3,2036**
Persistent Backpropagation	0.2893	11.5276	**3,2959**
Multiple Persistent Backpropagation	0.2885	11.4625	**3,3069**
REPLY			
Challenge Solution	*0.0740*	*8.3411*	***0.6172***
Dataset Split with Different Models	*0.0763*	*8.1415*	***0,6211***
Three Layer Fully Connected	0.0761	8.0120	**0,6097**
Four Layer Fully Connected	0.0741	7.8247	**0,5798**
Three Layers Circularly Connected	0.0762	7.9444	**0,6053**
Four Layers Circularly Connected	0.0767	8.1498	**0,6250**
Persistent Backpropagation	0.1000	8.0327	**0,8032**
Multiple Persistent Backpropagation	0.0770	7.8887	**0,6074**

enhances the value of PRAUC, resulting in a consistent increase in the objective function.
Another good experiment that almost always surpasses the baseline is the *Four Layers Circulary Connected*, exceeding it in both *like* and *reply* classes. Moreover its objective function is incremented with respect to its three layer version: *Three Layers Circularly Connected*, leading to the possibility of a further increase in accuracy by tuning the complexity of the ensemble.
The approaches that aggregated the greatest number of features: both the *Fully Connected* experiments and the *Multiple Persistent Backpropagation* experiment didn't perform very well. This suggests that even if the used GBDT implementation had an internal mechanism of features selection, in general attaching too many of them to the dataset is counterproductive and lowers the outcome.

6.3 Distribution Comparison

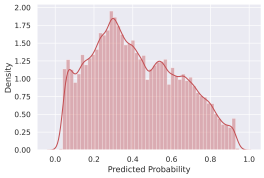
In this section we provide an analysis of the models and their prediction distributions among the labels. As it is possible to state by looking at figure 6.2 and figure 6.3 the different number positive interaction causes substantial differences in the forecast distributions. For the *like* label the predictions range through a wide range of the domain, covering it almost completely. The same cannot be said for *retweet* and *reply*. In fact, the predictions are condensed towards zero, increasing the amount of certain negative predictions. Regarding the data partitioning experiments it is interesting to show



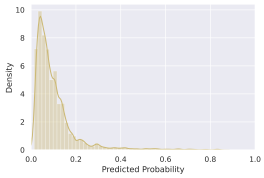
(a) Challenge Solution, like *(b) Challenge Solution, retweet* *(c) Challenge Solution, reply*

Figure 6.1: Distribution of predictions of the Challenge Solution for each label. On the x-axis there are the predicted probabilities, while on the y-axis there is their density.

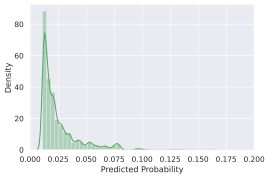
some differences among models that did perform well along with models that didn't. An example of that is given by figure 6.2a and figure 6.1a: respectively the most and the least performing models for the *like* label in the data



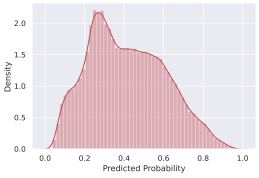
(a) Full Dataset with Different Models, like



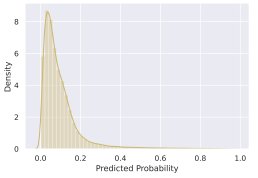
(b) Full Dataset with Different Models, retweet



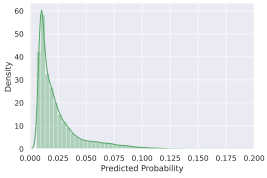
(c) Full Dataset with Different Models, reply



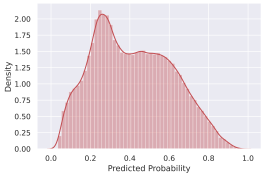
(d) Dataset Split with Same Models, like



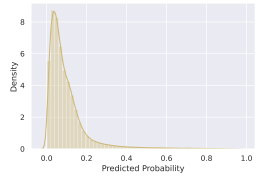
(e) Dataset Split with Same Models, retweet



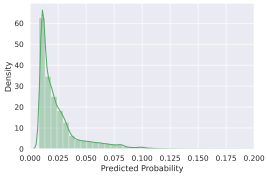
(f) Dataset Split with Same Models, reply



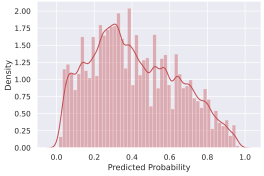
(g) Dataset Split with Different Models, like



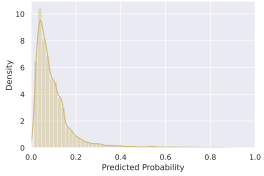
(h) Dataset Split with Different Models, retweet



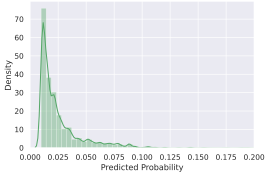
(i) Dataset Split with Different Models, reply



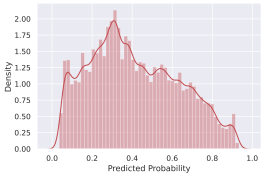
(j) Dataset Split with Common Features and Same Models, like



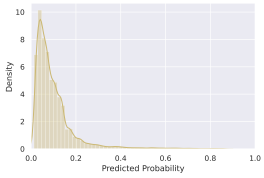
(k) Dataset Split with Common Features and Same Models, retweet



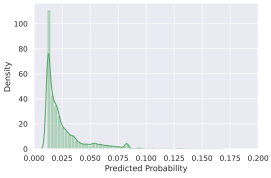
(l) Dataset Split with Common Features and Same Models, reply



(m) Dataset Split with Common Features and Different Models, like

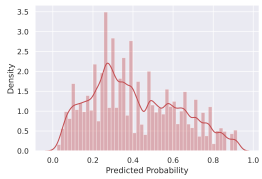


(n) Dataset Split with Common Features and Different Models, retweet

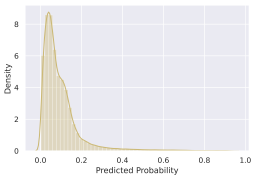


(o) Dataset Split with Common Features and Different Models, reply

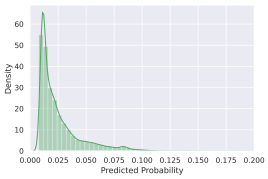
Figure 6.2: Distribution of predictions of the data partitioning experiments for each label. On the x-axis there are the predicted probabilities, while on the y-axis there is their density.



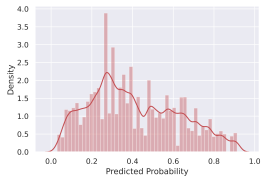
(a) Three Layers Fully Connected, like



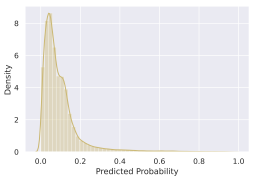
(b) Three Layers Fully Connected, retweet



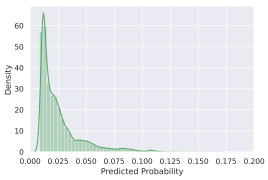
(c) Three Layers Fully Connected, reply



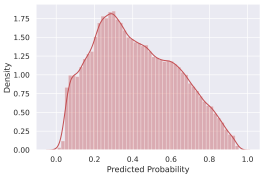
(d) Four Layers Fully Connected, like



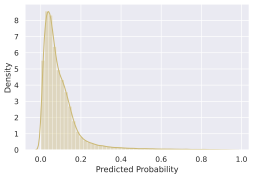
(e) Four Layers Fully Connected, retweet



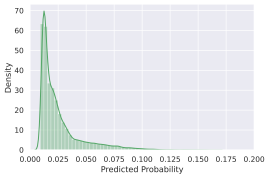
(f) Four Layers Fully Connected, reply



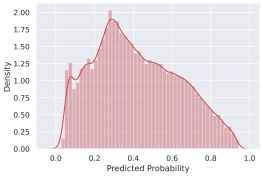
(g) Three Layers Cirularly Connected, like



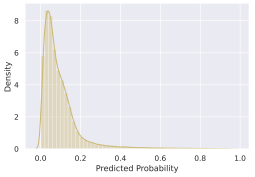
(h) Three Layers Cirularly Connected, retweet



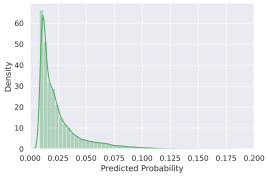
(i) Three Layers Cirularly Connected, reply



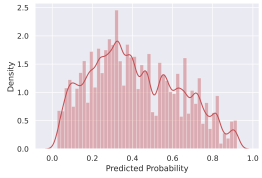
(j) Four Layers Cirularly Connected, like



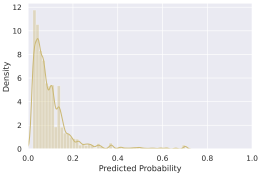
(k) Four Layers Cirularly Connected, retweet



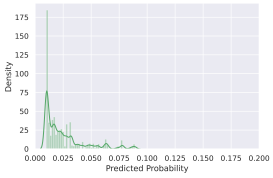
(l) Four Layers Cirularly Connected, reply



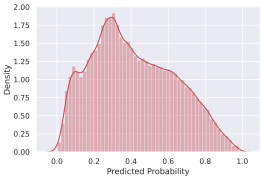
(m) Persistent Backpropagation, like



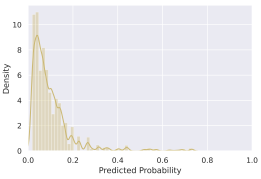
(n) Persistent Backpropagation, retweet



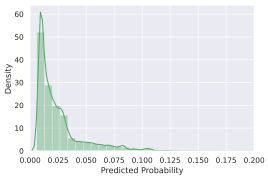
(o) Persistent Backpropagation, reply



(p) Multiple Persistent Backpropagation, like



(q) Multiple Persistent Backpropagation, retweet



(r) Multiple Persistent Backpropagation, reply

Figure 6.3: Distribution of predictions of the architecture experiments for each label. On the x-axis there are the predicted probabilities, while on the y-axis there is their density.

partitioning experiments. These two charts are quite similar, although they have some differences that justify the gap between their performance. While the curve of figure 6.1a, the *Challenge Solution*, seems smoother it has less certain prediction in the boundaries, respectively in the regions of probability $0 < p < 0.2$ and $0.8 > p > 1$. By pushing more predictions in such regions, the *Full Dataset with Different Models*, in figure 6.2a, is able to reach higher scores in the metrics. Another alike comparison, but for *retweet* label, can be obtained by comparing the charts in figure 6.2b and in figure 6.1b. They also show the most performing prediction in the data partitioning experiments is compared with the worst one, which were generated by the same models as before. In this case the distributions are quite similar, but there are some small differences that justifies their difference in performance. As can be seen the distribution shown in figure 6.2b has a higher pike in the proximity of 0, and less density of predictions near the probability 0.2 with respect of the histogram of figure 6.1b. Also this last chart shows that the model with worst performance also predicted more positive values, especially with probability $0.6 < p < 0.8$. So due to the imbalance it is a better strategy to predict negative interactions as these predictions are less likely to be wrong. These differences, albeit small, justified an increment of 12% in the objective function.

The most interesting analysis to perform belongs to the architecture experiments. In fact the PRAUC unexpectedly high in the *Persistent Backpropagation* experiment generated a way different distribution with respect to the other models. In this case it is interesting to compare it with the second best solution, the one provided by the *Four Layers Circularly Connected*, to show how the prediction of an already good solution change to obtain such increase.

As may be seen in figure 6.4 the two histograms are quite different. The figure 6.4a shows the chart of the *Persistent Backpropagation* in which the distribution of the prediction is way more discontinuous with respect to figure 6.4b, also the pike of the top chart's predictions near the probability of 0 have almost three times the density of the lower chart. Moreover, in figure 6.4 we can observe that the histogram's bars are thinner, implying that such model predicted a wider range of values, which suggest a higher complexity. This strategy slightly penalized the RCE score, certain incorrect predictions results in a penalty in terms of this metric, but raises significantly the PRAUC as most of certain predictions are correct. This is mainly possible because of the imbalance, the model that generated such forecasts in fact exploited the tradeoff between these two metrics. In order to do so the model condensed most of its predictions in a range of values trying to both mini-

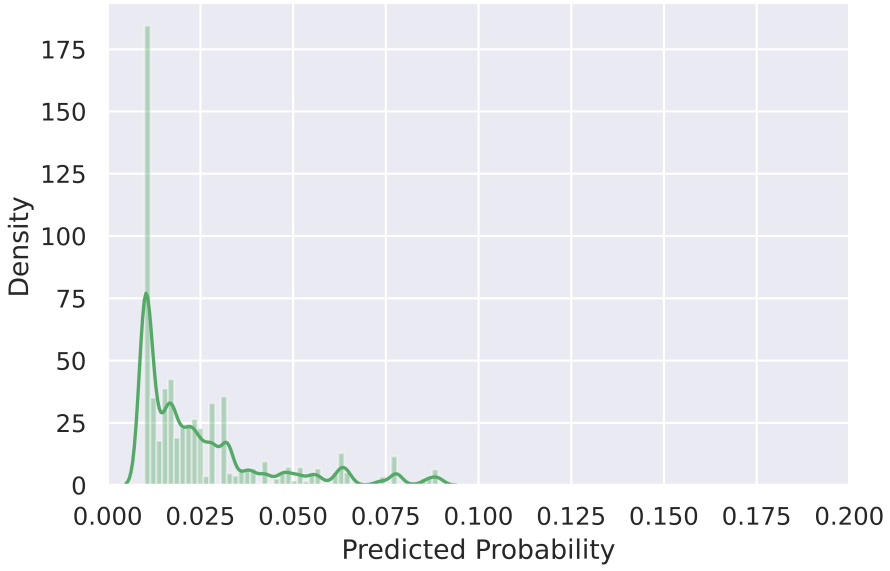
mized the misprediction penalty of RCE and maximized the PRAUC.
In highly imbalanced problems, in fact, there are little chances of mispredicting a certain value, in particular in a class where only about 2.4% of the overall interactions are positive. While this strategy cannot be applied to balanced or almost balanced classed, the trained models didn't succeed on exploiting it in the *retweet* class, which, still imbalanced, have around 10% of positive interactions, so it probably grow harder to use with respect to the increment of positive interactions in the dataset.

6.4 Complexity Inspection

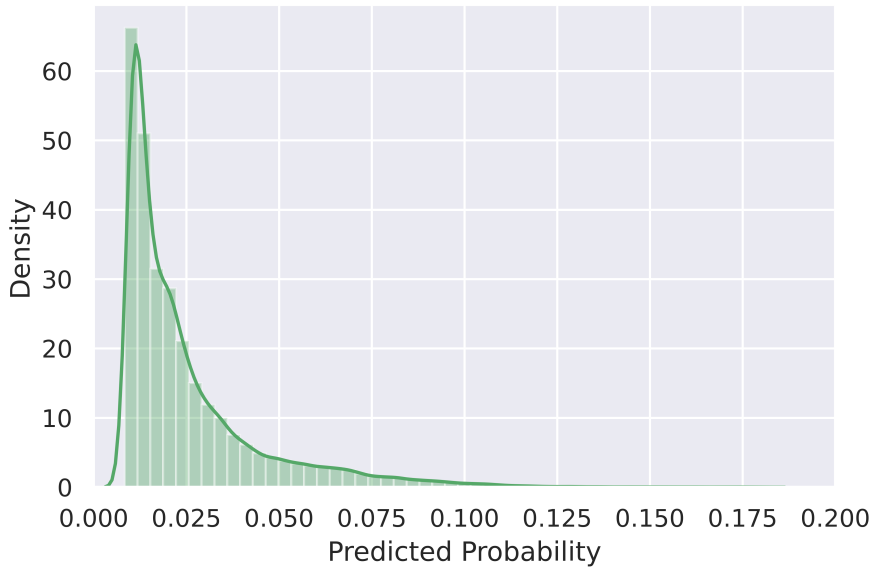
Among the proposed solutions, two of the experiments had an interesting behavior, the first is the *Circularly Connected* one, which surpassed the baseline in the *like* and the *reply* classes. Also its relative increment in the *Four Layers* model with respect to to the *Three Layers* opened up the possibility of a further increase by tuning the complexity of such technique.
The second is the *Persistent Backpropagation* model that did surpass in every case the baseline and was the only one able to reach such high results in the *reply* label. Even in this case the number of backpropagation cycles was chosen arbitrarily, hence not tuned. So in this section we present an inspection of these models' performance, in order to study what happens by increasing their complexity.

6.4.1 Circular Connection Inspection

In this section we study the behavior of the *Circularly Connected* model architecture increasing the number layers up to eight.
Figure 6.5 shows nine different line charts, describing the evolution of RCE, PRAUC and the objective function over the increasing complexity. They show this for all the three labels, which lead to similar conclusions. We can state that in general this kind of models obtain the highest value only using few layers. The only exception is the chart shown in figure 6.5i for the *reply* label which, increasing the complexity up to the eighth layer, becomes able to exploit the tradeoff between the RCE and the PRAUC explained in section 6.3, almost doubling the objective functions with respect to the other predictions. This behavior can be observed also in figure 6.5c and figure 6.5c, they show respectively a pike of PRAUC and a drop of RCE corresponding to the eight layers model, confirming the existence of an exploitable tradeoff between the two metrics.
For this approach, doesn't seem necessary to push the complexity too far, as

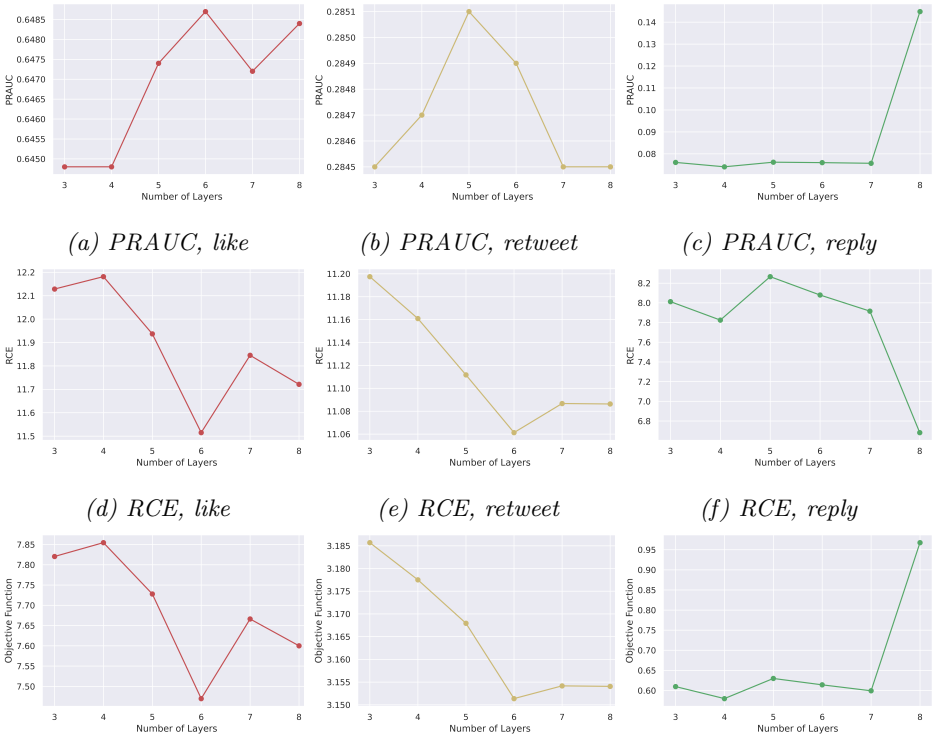


(a) Persistent Backpropagation



(b) Four Layers with Circularly Connected

Figure 6.4: Comparison among the distribution of the two most performing architecture experiments on reply label. On the x-axis there is the predicted probability of such models, while on the y-axis the density of the predictions.



(a) PRAUC, like *(b) PRAUC, retweet* *(c) PRAUC, reply*

(d) RCE, like *(e) RCE, retweet* *(f) RCE, reply*

(g) Objective Function, like *(h) Objective Function, retweet* *(i) Objective Function, reply*

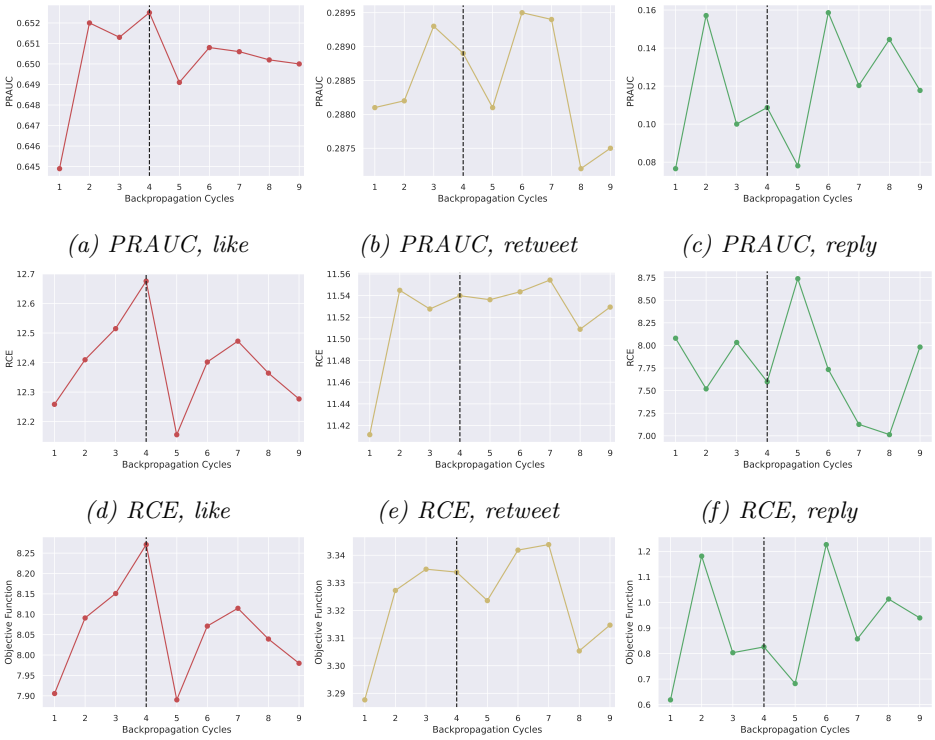
Figure 6.5: Circularly Connected models' performance with different number of layers evaluated on PRAUC, RCE and objective function, for each label. On the x-axis there is the number of layers, on the y-axis there are the metrics scores.

it always produced better predictions with less complex models, granting a shorter training time, but denying the hoped-for accuracy increase.

6.4.2 Persistent Backpropagation Inspection

In this subsection we study the behavior of the *Persistent Backpropagation* model, trying to figure out how its prediction accuracy would change by exploring the number of backpropagation cycles up to a maximum of 9.
In figure 6.6 there are nine different charts showing the evolution of PRAUC, RCE and the objective function with respect to the number of backpropagation cycles for each label. The number of these rounds was arbitrarily set to 3, however in each of these plots a higher value is reached beyond the third. So the best approach over the three label, as stated in section 6.2, it is to tune the number of cycles due to the possibility of a further accuracy raise. Another interesting aspect of this technique is in the *reply* label, in fact,

it carries several solution which exploit the PRAUC and RCE tradeoff. In particular, over nine solutions only two of them didn't use it ending up to be the less performing, respectively, the first and the fifth cycle solutions.
Another interesting aspect is the objective function charts' likelihood. Each of them in fact presents a minima on the fifth cycle and two pikes, one before and one after it. In order to enlighten this behavior there is a dashed black line on the charts. The empirical observations suggest that up to the fourth round, increasing the number of backpropagation cycles raises the accuracy of the models. From the fifth to the ninth one, instead, the models act unexpectedly, in fact almost all the charts in figure 6.6 present a sudden and impactful drop on the fifth cycle, then a growth on the sixth.
Even with different labels, hence diverse numbers of positive samples, these models, which random optimization ran on different seeds, have a similar learning dynamic with respect to the imbalance and the number of rounds.
Using this technique a fine extended tuning it is fundamental in order to maximize its accuracy, which is also one of this technique's drawbacks. In fact, for each cycle the entire structure need to be trained again on the new backpropagated dataset, resulting in an increase of training time.



(a) PRAUC, like *(b) PRAUC, retweet* *(c) PRAUC, reply*

(d) RCE, like *(e) RCE, retweet* *(f) RCE, reply*

(g) Objective Function, like *(h) Objective Function, retweet* *(i) Objective Function, reply*

Figure 6.6: Persistent Backpropagation model's performance with different number of backpropagation cycles evaluated on PRAUC, RCE and objective function, for each label. On the x-axis there is the number of cycles, while on the y-axis there are the metrics scores.

Chapter 7

Conclusions

The state of the art shows that nowadays most competitive and high-end solutions in recommender systems make use of the ensemble approaches. Among these a recurrent choice due to its accuracy and robustness is the stacking ensemble. Despite this is one of the most common techniques to perform the models ensemble, most of the literature work use very similar structures: a two level stacking generalization, lacking of exploration.
The goal of this work is to propose and explore new ideas to further increment such technique's performance and scalability over imbalance, relying on two main strategies: the dataset partitioning and the modification of the ensemble architectures. The results of these solutions, discussed in chapter 6, presented several unexpected implications.
From the data partitioning experiments' results, presented in section 6.1, it can be said that, in general, for balanced and for slightly unbalanced problem the best approach, among the tested ones, is to diversify the GBDT implementations within the ensembles, then train them with the complete dataset.
However there is another competitive solution, which strong points reside in robustness and scalability. In fact, by dividing the dataset in five different subsets, each containing the 60% most relevant features, with the strategy presented in section 5.2.5, it is possible to achieve a high accuracy on each one of the labels.
Considering that the two most performing results weren't the most diverse ensemble, we can state that the diversification level must be related to the complexity of the problem dealt with. Hence trying to maximize the diversification in the ensemble, as the results in table 6.1 show, isn't always a good strategy. The dataset diversification instead, should be carefully taken into account as a design hyperparameter.

The architecture experiments held the most unexpected results. The best model, according to its high accuracy in the tested environment, was the one implementing the *persistent backpropagation* approach, singularly. Its performance turned out to be excellent on each class, surpassing in each case the score of the baseline, revealing to be an accurate, robust and scalable approach. This technique, appositely created for the purpose of this thesis opens up new possibilities. As demonstrated in section 6.4.2, it has a wide margin of growth by tuning the number of backpropagation cycles. It could also be tested with most performing splits of the dataset or on multilayer structures in order to have a further enhancement of its qualities. Moreover it would be interesting to investigate its behavior after the fourth backpropagation cycle with further work.
Another approach belonging to the architecture experiments that gave interesting results is the circular aggregation technique. Despite it did not notably increased the accuracy over the complexity tuning, the proposed *Four Layers Circularly Connected* model obtained results higher than the baseline's ones for both the *like* and *reply* labels.
By taking up a consideration made in section 6.2, that is to aggregate too much predictions in an ensemble could lead to a performance drop, we can state that a maximum number of predictions to add on each dataset should be set as a design hyperparameter. Once reached that number, an elimination system could be taken into account. For example keeping the most recent predictions and removing the older ones could be a good strategy.
In conclusion the most promising results were obtained by exploiting the architecture of the ensembles, providing two new approaches to test in different contexts. While the data partitioning solutions showed that, for simple ensembles, approaches trained on a larger portion of features tend to perform better. Also, when employing a dataset division strategy, it is better to use a single, more performing GBDT algorithm, according to these results.

Bibliography

[1] M. Abdar, M. Zomorodi-Moghadam, X. Zhou, R. Gururajan, X. Tao, P. D. Barua, and R. Gururajan. A new nested ensemble technique for automated diagnosis of breast cancer. *Pattern Recognition Letters*, 132:123–131, 2020. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2018.11.004. URL `https://www.sciencedirect.com/science/article/pii/S0167865518308766`. Multiple-Task Learning for Big Data (MTL4BD).

[2] V. W. Anelli, A. Delić, G. Sottocornola, J. Smith, N. Andrade, L. Belli, M. Bronstein, A. Gupta, S. Ira Ktena, A. Lung-Yut-Fong, F. Portman, A. Tejani, Y. Xie, X. Zhu, and W. Shi. Recsys 2020 challenge workshop: Engagement prediction on twitter's home timeline. page 623–627, 2020. doi: 10.1145/3383313.3411532. URL `https://doi.org/10.1145/3383313.3411532`.

[3] G. Biau, L. Devroye, and G. Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9(9), 2008.

[4] L. Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, Aug. 1996. ISSN 0885-6125. doi: 10.1023/A:1018054314350. URL `https://doi.org/10.1023/A:1018054314350`.

[5] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12, 11 2002. doi: 10.1023/A:1021240730564.

[6] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[7] M. Chen and P. Liu. Performance evaluation of recommender systems. *International Journal of Performability Engineering*, 13(8), 2017.

[8] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. page 785–794, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

[9] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[11] A. V. Dorogush, A. Gulin, G. Gusev, N. Kazeev, L. O. Prokhorenkova, and A. Vorobev. Fighting biases with dynamic boosting. *CoRR*, abs/1706.09516, 2017. URL http://arxiv.org/abs/1706.09516.

[12] A. V. Dorogush, V. Ershov, and A. Gulin. Catboost: gradient boosting with categorical features support. 2018.

[13] S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 06 2003. ISSN 1367-4803. doi: 10.1093/bioinformatics/btg038. URL https://doi.org/10.1093/bioinformatics/btg038.

[14] N. Felicioni, A. Donati, L. Conterio, L. Bartoccioni, D. Y. X. Hu, C. Bernardis, and M. Ferrari Dacrema. Multi-objective blended ensemble for highly imbalanced sequence aware tweet engagement prediction. page 29–33, 2020. doi: 10.1145/3415959.3415998. URL https://doi.org/10.1145/3415959.3415998.

[15] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364. URL http://www.jstor.org/stable/2699986.

[16] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. ISSN 0167-9473. doi: https://doi.org/10.1016/S0167-9473(01)00065-2. URL https://www.sciencedirect.com/science/article/pii/S0167947301000652. Nonlinear Methods and Data Mining.

[17] M. Ghazanfar and A. Prugel-Bennett. An improved switching hybrid recommender system using naive bayes classifier and collaborative filtering. April 2010. URL https://eprints.soton.ac.uk/268483/. Event Dates: 17-19 March, 2010.

[18] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004. ISSN 1046-8188. doi: 10.1145/963770.963772. URL `https://doi.org/10.1145/963770.963772`.

[19] M. Jahrer, A. Töscher, and R. Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, page 693–702, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300551. doi: 10.1145/1835804.1835893. URL `https://doi.org/10.1145/1835804.1835893`.

[20] N. Jamali and C. Sammut. Majority voting: Material classification by tactile sensing using surface texture. *IEEE Transactions on Robotics*, 27(3):508–521, 2011. doi: 10.1109/TRO.2011.2127110.

[21] P. Jankiewicz, L. Kyrashchuk, P. Sienkowski, and M. Wójcik. Boosting algorithms for a session-based, context-aware recommender system in an online travel domain. 2019. doi: 10.1145/3359555.3359557. URL `https://doi.org/10.1145/3359555.3359557`.

[22] G. Ke, Q. Meng, T. Finely, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. December 2017. URL `https://www.microsoft.com/en-us/research/publication/lightgbm-a-highly-efficient-gradient-boosting-decision-tree/`.

[23] L. Kuncheva and J. Rodríguez. A weighted voting framework for classifiers ensembles. *Knowledge and Information Systems*, 38, 02 2014. doi: 10.1007/s10115-012-0586-6.

[24] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent, 1999.

[25] J. Matyas. Random optimization. *Automation and Remote control*, 26 (2):246–253, 1965.

[26] M. McIntire, D. Ratner, and S. Ermon. Sparse gaussian processes for bayesian optimization. 2016.

[27] E. Menahem, L. Rokach, and Y. Elovici. Troika - an improved stacking schema for classification tasks. *Inf. Sci.*, 179(24):4097–4122, Dec. 2009. ISSN 0020-0255. doi: 10.1016/j.ins.2009.08.025. URL `https://doi.org/10.1016/j.ins.2009.08.025`.

[28] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, Mar. 1997. ISSN 0001-0782. doi: 10.1145/245108.245121. URL https://doi.org/10.1145/245108.245121.

[29] L. Rokach. Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography. *Computational Statistics & Data Analysis*, 53(12):4046–4072, 2009. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2009.07.017. URL https://www.sciencedirect.com/science/article/pii/S0167947309002631.

[30] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. 2020.

[31] R. E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2): 197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL https://doi.org/10.1023/A:1022648800760.

[32] R. E. Schapire. A brief introduction to boosting. page 1401–1406, 1999.

[33] B. Schifferer, G. Titericz, C. Deotte, C. Henkel, K. Onodera, J. Liu, B. Tunguz, E. Oldridge, G. De Souza Pereira Moreira, and A. Erdem. Gpu accelerated feature engineering and training for recommender systems. page 16–23, 2020. doi: 10.1145/3415959.3415996. URL https://doi.org/10.1145/3415959.3415996.

[34] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. page 2951–2959, 2012.

[35] R. Tugay and c. Gündüz Öğüdücü. Demand prediction using machine learning methods and stacked generalization. In *Proceedings of the 6th International Conference on Data Science, Technology and Applications*, DATA 2017, page 216–222, Setubal, PRT, 2017. SCITEPRESS - Science and Technology Publications, Lda. ISBN 9789897582554. doi: 10.5220/0006431602160222. URL https://doi.org/10.5220/0006431602160222.

[36] D. A. Unger, H. van den Dool, E. O'Lenic, and D. Collins. Ensemble regression. *Monthly Weather Review*, 137(7):2365 – 2379, 01 Jul. 2009. doi: 10.1175/2008MWR2605.1. URL https://journals.ametsoc.org/view/journals/mwre/137/7/2008mwr2605.1.xml.

[37] L. Wang, D. Zhou, H. Zhang, W. Zhang, and J. Chen. Application of relative entropy and gradient boosting decision tree to fault prognosis in electronic circuits. *Symmetry*, 10(10), 2018. ISSN 2073-8994. doi:

10.3390/sym10100495. URL https://www.mdpi.com/2073-8994/10/10/495.

[38] Z. Wei, Y. Meng, W. Zhang, J. Peng, and L. Meng. Downscaling smap soil moisture estimation with gradient boosting decision tree regression over the tibetan plateau. *Remote Sensing of Environment*, 225:30–44, 2019. ISSN 0034-4257. doi: https://doi.org/10.1016/j.rse.2019.02.022. URL https://www.sciencedirect.com/science/article/pii/S003442571930077X.

[39] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[40] W. Xiao, X. Xu, K. Liang, J. Mao, and J. Wang. Job recommendation with hawkes process: An effective solution for recsys challenge 2016. 2016. doi: 10.1145/2987538.2987543. URL https://doi.org/10.1145/2987538.2987543.

[41] Z.-H. Zhou and J. Feng. Deep forest: Towards an alternative to deep neural networks. page 3553–3559, 2017.