
Beating Kaggle the easy way

Studienarbeit

Ying Dong

Wirtschaftsinformatik



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Ying Dong
Studiengang: Master Wirtschaftsinformatik

Studienarbeit
Thema: " Beating Kaggle the easy way"

Eingereicht: 22. Juli 2015

Betreuer: Dr. Frederik Janssen

Prof. Dr. Johannes Fürnkranz

Fachgebiet Knowledge Engineering Group
Fachbereich Informatik
Technische Universität Darmstadt
Hochschulstraße 10
64289 Darmstadt

Contents

1	Introduction	1
1.1	Kaggle	1
1.2	Outline	2
2	Theory and method	3
2.1	Machine learning and data mining	3
2.2	Learning Algorithms	5
3	A pipeline for Kaggle competitions	8
3.1	Data exploration	9
3.2	Data preprocessing	10
3.3	Feature construction	12
3.4	Feature selection	13
3.5	Model validation and selection	14
3.6	Hyperparameter tuning	15
4	Kaggle competition 1: Forest Cover Type Prediction	17
4.1	Problem Description	17
4.2	Dataset	17
4.3	Evaluation metric	18
4.4	Approach and progress	19
4.5	Conclusion	27
4.6	Lessons learned	27
5	Kaggle competition 2: Otto Group Product Classification	29
5.1	Problem Description	29
5.2	Dataset	29
5.3	Evaluation metric	29
5.4	Approach and progress	30
5.5	Conclusion	35
5.6	Lessons learned	36
6	Kaggle competition 3: Bike Sharing Demand	37
6.1	Problem Description	37
6.2	Dataset	37
6.3	Evaluation metric	38
6.4	Approach and process	38
6.5	Conclusion	43
6.6	Lessons learned	44
7	Conclusion	45
8	References	46

List of Figures

Figure 1: Overview of steps of KDD[2].....	5
Figure 2: Decision Tree	6
Figure 3: Overview of the pipeline	8
Figure 4: Outlier.....	9
Figure 5: Hillshade_Noon vs. Hillshade_3pm.....	19
Figure 6: Hillshade_9am vs. Hill_shade_3pm.....	20
Figure 7: Histogram of Elevation: different cover types are marked with different colors.....	20
Figure 8: Feature importance computed by random forest model.....	21
Figure 9: Confusion matrix of random forest	22
Figure 10: Confusion matrix of extra trees	25
Figure 11: Leaderboard Scores and Standings for competition forest cover type classification	27
Figure 12: Distribution of Product Categories	30
Figure 13: Correlation Matrix of 15 most important features.....	31
Figure 14: Probability distribution after calibration	33
Figure 15: Score vs. Standing for competition Otto group products.....	36
Figure 16: Correlation between features	39
Figure 17: Mean value of count/registered/casual per day of week.....	39
Figure 18: Mean value of count/registered/casual per hour	40
Figure 19: Score vs. Standing for competition Bike Sharing Demand Prediction	44

List of Tables

Table 1: Competition categories	2
Table 2: Weather Data as a Dataset Example	3
Table 3: Learning Algorithms	5
Table 4: Discretization example	11
Table 5: Split color feature into binary features	13
Table 6: Description of the features for competition Forest cover type prediction [35]	18
Table 7: New constructed features	23
Table 8: Validation scores and leaderboard scores	24
Table 9: Ordered One-vs. -All.....	25
Table 10: Public leaderboard overview of first competition	26
Table 11: Description of features for competition Otto group products classification [37]	29
Table 12: Private leaderboard overview of second competition	35
Table 13: Features for competition bike sharing demand [42].....	37
Table 14: Validation and leaderboard scores.....	41
Table 15: Validation and leaderboard log-loss errors of different combination of models.....	42
Table 16: Public leaderboard overview of third competition.....	43

1 Introduction

In recent years big data has become a hot topic. The amount of data in the world has been exploding in all science, engineering and business domains because of the fast development of computer technologies and data storage capacity [1]. The need for automated data analysis tools to accurately detect the knowledge contained within the huge volumes of data is urgent. Data mining was introduced as a corresponding solution [2].

Data mining has attracted lots of people. To improve their research methods, data scientists need lots of real data. On the other hand, companies need more accurate models to make predictions. Data prediction competitions are usually the right choice for both researchers and companies. Kaggle is one of the most popular platforms for predictive modeling and analytics competitions.

The main idea of the thesis is to achieve the goal of getting result as good as possible with as little effort as possible in the ranking of a variety of different Kaggle competitions by designing a pipeline based on the first competition and re-use of the pipeline on other tasks. Preferably, the 3 selected Kaggle tasks are different. The pipeline is applied on all tasks and it is evaluated based on where one places on the leaderboard.

1.1 Kaggle

Kaggle is a platform for predictive modeling and analytics competitions. Companies provide datasets and descriptions of the problems on Kaggle. Participants can then download the data and build models to make predictions and then submit their prediction results to Kaggle.

For each competition, Kaggle usually provides a training dataset and a test dataset. All submissions will be evaluated by certain evaluation metrics based on the test dataset. The evaluation results will be shown in a leaderboard to let the participants know their relative progress. The scores shown on the leaderboard during the competition are known as “public score”, which is calculated based on a fraction of the test data set, which is uniquely specified for each competition. The final score, which is called “private score”, will be given based on the complete test dataset after the competition is closed. The final ranking of the competition is calculated according to the private score.

There are many different categories of competition with different incentives and goals. Some of them [3] are listed in Table 1.

Table 1: Competition categories

Category	Description
Featured	<ul style="list-style-type: none">• public competitions• with significant prize money• goal is to solve commercial problems.
Masters	<ul style="list-style-type: none">• invitation-only competitions• with significant commercial value or sensitive data.
Research	<ul style="list-style-type: none">• public competitions• goals are research/ scientific in nature or serve a public good• with cash prizes or invitations to conferences or publication in peer-reviewed journals.
Playground	<ul style="list-style-type: none">• public competitions• set up to be fun, quirky and idea-driven• without any prize.
Getting Started	<ul style="list-style-type: none">• public competitions• without cash prizes• for machine learning beginner• always available and have no deadline.

1.2 Outline

Chapter 2 provides a general overview of data mining and several machine learning algorithms used throughout this thesis. In Chapter 3 a pipeline to approach data prediction competitions is presented. In Chapter 4, Chapter 5 and Chapter 6, results and the approach of three competitions on the Kaggle platform are described. Chapter 7 is the conclusion with a summary of the results.

2 Theory and method

In this chapter, I will briefly introduce the terms “machine learning” and “data mining”, then some machine learning algorithms will be introduced, which are used throughout this thesis.

2.1 Machine learning and data mining

Machine learning is a science of devising algorithms to get computers to learn from data automated without being explicitly programmed [4]. Such algorithms (learning algorithms) build a model after learning from the data and the model can be used to make predictions or decisions. The learning phase is also known as the training phase.

Table 2: Weather Data as a Dataset Example

Instances	Features/Attributes				Target
	Outlook	Temperature	Humidity	Windy	Play Time
1	Sunny	24	80	No	70
2	Sunny	23	85	Yes	12
3	Overcast	18	90	No	40
4	Rainy	15	95	Yes	0
5	Rainy	18	90	No	20
6	Overcast	20	87	Yes	50
7	Sunny	21	88	No	45

The data to be learned from is called a “dataset”. Table 2 containing weather data shows an example dataset. A dataset is composed of a set of instances, each instance is an individual and independent example [5]. In Table 2 the rows of the table are the instances. Each instance is characterized by a series of attributes or features. In Table 2 the features are the columns of the table. A special attribute is the target value, which is the desired output to be predicted or classified. The two most common types of features are nominal and numeric. Nominal attributes take on values in a finite set of possibilities and denote no discernible order. For example, the attribute *Outlook* in the weather data is nominal and has values Sunny, Overcast and Rainy. Numeric attributes are valued by either real or integer numbers, which are comparable. An example of numeric attributes is the attribute *Temperature* in the weather data example.

When the desired outputs for the example inputs are given, the machine learning task is known as a supervised learning problem. On the contrary if the corresponding target values of the example inputs are not given, then the task is an unsupervised learning problem. Only supervised learning problems are concerned in this thesis.

In the case of the desired output being numeric, the task is called “regression”. The weather data presented in Table 2 is an example of a regression problem. If the aim of a task is to assign each input into a finite number of categories or classes, the task is called “classification”. The function, which is implemented by a classification algorithm and used to assign input data to a class, is called a classifier. For example, changing the aim of the weather example to be prediction of “go out or not”, would cause the desired output to be either “yes” or “no”, given the weather data, this task is a binary classification problem. If the number of classes is more than 2, the task is a multi-class classification problem.

Machine learning provides the technical basis of data mining [5]. Data mining uses many machine learning methods. Data mining is an interdisciplinary subfield of computer science, aiming at extracting useful information from large datasets [6]. In [2] data mining is described as a particular step in Knowledge Discovery in Databases(KDD). “KDD” refers to the overall process of extraction of useful information from raw data, while “data mining” refers to the process of using specific algorithms to extract patterns from data.

Figure 1 shows an overview of the process of KDD. The KDD process is interactive and iterative, in [2] 9 basic steps are broadly outlined:

1. The first step of KDD is to identify the goal of the KDD process by developing an understanding of the relevant domain and knowledge.
2. The second step is to gather an appropriate dataset from a database or data warehouse, which is to be used to discover knowledge.
3. Then the selected dataset will be cleaned and preprocessed, including defining and handling missing data, removing noisy data, etc.
4. The preprocessed data will be then transformed to keep only useful features representing the data based on the goal of the task. In this step, procedures like dimensionality reduction will be applied.
5. Then an appropriate data mining method with regard to the goals of the KDD process should be determined: summarization, classification, regression, association or clustering and so on.
6. In this step the appropriate algorithm(s) and method(s) are chosen, which will be used to search for patterns in the data.
7. The dataset, algorithms and methods, which are created or selected from preceding steps, can be then used to search the data patterns.
8. The mined patterns will be interpreted and evaluated to specify which patterns can be treated as new knowledge.
9. Lastly, the knowledge can be put to further use.

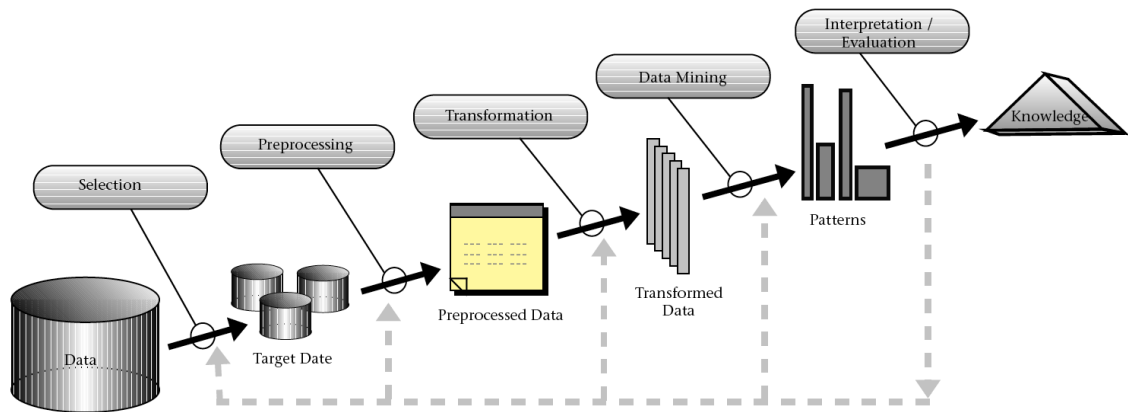


Figure 1: Overview of steps of KDD[2]

2.2 Learning Algorithms

There are many machine learning algorithms, Table 3 shows some of them as examples. This section will only focus on some brief overviews of the methods actually employed in the Kaggle challenges in Chapter 4, Chapter 5 and Chapter 6, which are Random Forest [7], Extremely Randomized Trees [8], Gradient Boosting Trees [9], and Extreme Gradient Boosting [10], to give a better understanding of how they work. A more detailed description of the machine learning algorithms can be found for example in [11].

Table 3: Learning Algorithms

Regression	Classification
Linear regression	Logistic regression
K nearest neighbors	Naive Bayes
Neural Networks	Neural Networks
Support Vector Machines	Decision Trees
Random Forest	Random Forest
Gradient Boosting	Gradient Boosting

2.2.1 Random Forest

The Random Forest algorithm is built based on decision tree models. The decision tree models are obtained by recursively partitioning the data set in various ways into branch-like segments and fitting a simple prediction model within each segment. The partitioning can be expressed

graphically as a decision tree [12]. A decision tree contains one root node, a set of internal (or split) nodes and leaf nodes [13]. Each internal node is labeled with a feature. The outgoing edges of that internal node represent possible values satisfying some constraints of that feature. Each leaf node is labeled with a target class. For example, in Figure 2 is a decision tree for a classification problem with 3 features: a, b, c, and 3 classes.

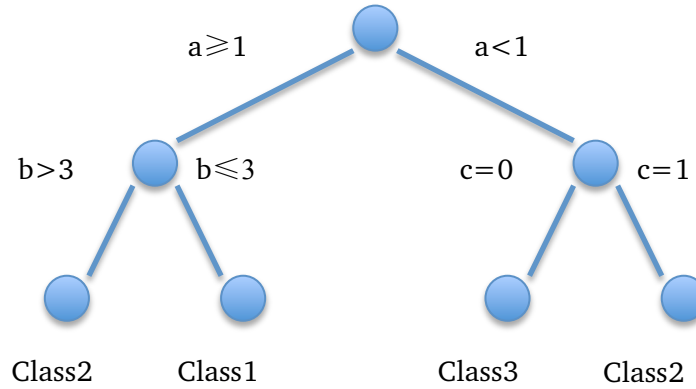


Figure 2: Decision Tree

Random forest uses an ensemble method by combining a multitude of decision trees. The main idea behind ensemble methods is to construct a single model by combining a set of base models [14]. It has been proven that using ensemble methods can give better results than using a single model when measured on predictive accuracy. Random forest uses a bagging method, which averages the predictions of multiple models trained on different samples to reduce the variance and achieve higher accuracy.

For a classification task, an instance from a dataset is classified according to each single tree, the trees will vote for their classification and the class having the most votes will be chosen as the final result. For a regression task, the result is calculated by averaging the predictions.

Each tree in this algorithm is grown in this way[15]:

1. Setting n as the number of instances in the training set, a bootstrap sample of size n is sampled at random with replacement from the original data. This subset containing n instances will be the training set for growing a tree. The rest of the data that is not included in the tree construction is utilized as a test set for making an error-estimate, which is called the “out of bag” error.
2. For some number m smaller than the total number of attributes M , m attributes are selected randomly out of M attributes at each node. According to some objective function, the optimal split on these selected m attributes is used to do a binary split on that node.
3. There is no pruning when each tree is growing and the trees are grown to the largest extent possible.

The accuracy of a Random Forest model depends on the strength of each individual tree and the correlations between any two trees [7]. A strong tree classifier has a lower error rate. In-

creasing the strength of each tree increases the forest accuracy, while increasing the correlation decreases the forest accuracy. Using random features selection (feature bagging) when finding the optimal tree split is a strategy to reduce correlation. The reason for feature bagging is that if some features have very strong power to predict the target attribute, they will be used in many trees, which causes the trees to become correlated [16]. Reducing the number of random selected features m can reduce the correlation but also reduce the strength of the trees. In between there is a balance for m . This number m and the number of trees are two adjustable parameters for a Random Forest model.

2.2.2 Extremely Randomized Trees

Extremely Randomized Trees (Extra Trees) is similar to the Random Forests algorithm in the sense that it is based on selecting at each node a random subset of m attributes to decide on the split [8]. But two differences compared to the Random Forest model are that each tree in the Extra Trees model is grown from the whole dataset (no bootstrap sample) and a discretization threshold (cut-point) is randomly selected for each of the features to define a split, instead of choosing the best cut-point based on some objective function as in the Random Forests method. For the Extremely Randomized Trees model the number of randomly selected attributes and the number of trees are also the most important two adjustable parameters.

2.2.3 Gradient Boosting

Boosting is another method of creating an ensemble of many weak predictors, which are then weighted in some way that is usually related to their accuracy. The weighted weak predictors are added to a final strong one [17].

Gradient Boosting Trees (GBRT) uses boosting techniques to integrate the strength of many different decision trees to build a stronger one. The model is built in an additive way such that the parameters of each new added base predictor is optimized by moving in the opposite direction of the gradient to minimize the loss function [9]. The loss function can be arbitrary, for example square loss, absolute loss, or Huber loss [18].

Extreme Gradient Boosting (Xgboost) [10] is a very fast and effective machine learning model, which implements algorithms under a gradient-boosting framework, including a generalized linear model and gradient-boosted regression tree. Xgboost is first introduced by [19], it is widely used in Kaggle competitions and is utilized in many winning solutions.

3 A pipeline for Kaggle competitions

In this chapter a pipeline for approaching Kaggle competitions is to be presented, the design of which is on the first competition in Chapter 4 and then reused in the remaining two competitions in Chapter 5 and Chapter 6.

The competitions hosted by Kaggle are different from data mining problems in the real world. Hamner, Kaggle Co-founder & CTO [20] states that there are three main things that Kaggle has done for competition participants that professional data scientists need to do:

- Identify the problem and how to address it using machine learning
- Collect the appropriate data
- Clean and transform the data, and split the data into training and test datasets

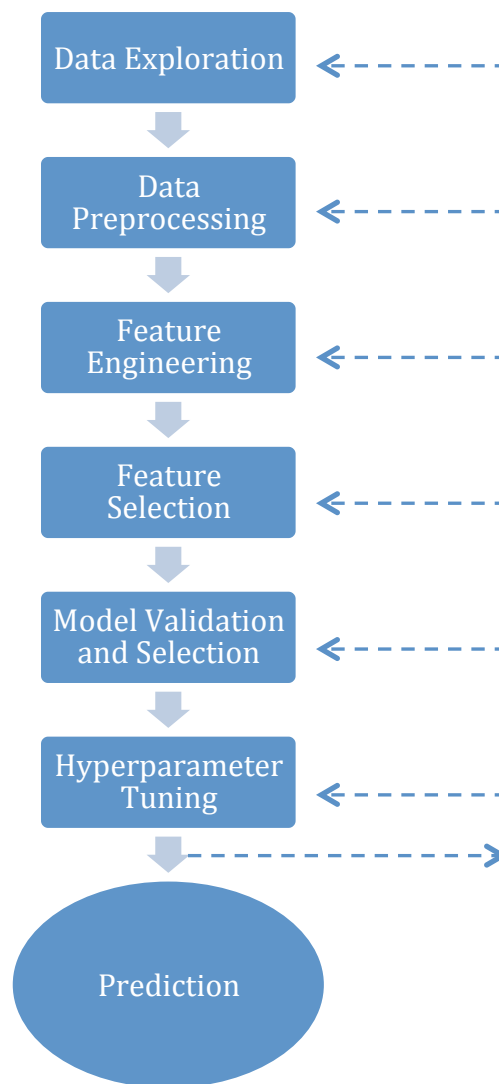


Figure 3: Overview of the pipeline

As described in Section 2.1, a real-world data mining or KDD problem begins with identifying the problem and the goal of the KDD process, the data used for solving the problem should be gathered from a database or data warehouse. Competitions in Kaggle have already well defined the final goals of the problems with no need for participants to dive deeply into the relevant domain to understand how their work and output will hit the goal. Kaggle also provides relevant, cleaned, well-split training and test datasets for competition participants. Different from the KDD process in Section 2.1, the pipeline introduced in this Chapter begins directly with data exploration. Figure 3 shows an overview of the pipeline. There are 6 steps before the final prediction task, like KDD process, the process of this pipeline is also interactive and iterative.

3.1 Data exploration

With the exception of reading the description of the competition, the first step in starting the real work is to do exploratory data analysis to summarize the data's main characteristics. There is no standard or perfect way to explore the data, it is time-consuming and always depends on the experience of the participants. Still, there are some methods, which are often a fine choice to consider when exploring data.

3.1.1 Outlier detection

A definition of an “outlier” given by [21] is an instance which is significantly different from the other instances and which therefore arouses suspicions that the instance was created using a different mechanism. An example of an “outlier” is shown in Figure 4. Despite the meaning of the axes, the encircled point in the figure does not fit the general trend of the other data points, so it can be seen as an outlier.

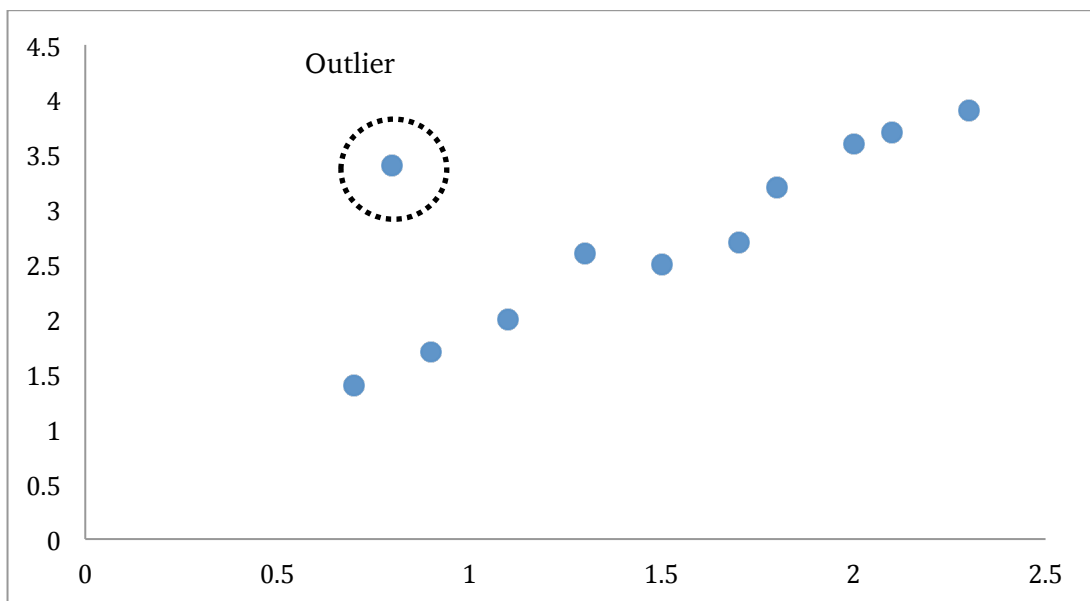


Figure 4: Outlier

A common way to find outliers is to make plots of the concerned feature, as in Figure 4. Another way is to review the extreme values (minimum and maximum) and the percentiles of the data numerically.

3.1.2 Data visualization

One of the easiest ways to understand the data is using different plotting methods to visualize the data. Manually checking thousands or even millions of instances is not only time-consuming, but also cannot give a direct insight into the data. There are various ways to do data visualization. Some very common and simple ones are for example box plot [22], scatter plot (Figure 5) and histogram (Figure 7).

3.1.3 Summary statistics

Summary statistics provide information that gives a quick and simple description of the large amount of data. Common summary measures include mean, median, minimum value, maximum value, range, standard deviation, and correlation between attributes if more than one attribute is measured.

3.2 Data preprocessing

Data preprocessing is a very critical step in any data mining process, it is more difficult to discover data patterns during the training step if there is significant noise and unreliability within the data [23].

The specific data preprocessing may depend on the data that is available and the learning algorithm that will be used to solve the problem. Some major tasks in data preprocessing are described in this section.

3.2.1 Data cleaning

Data cleaning primarily deals with filling in missing values and handling outliers to remove the noise of data.

A time-consuming way [6] to fill in missing values is to do it manually, but this is perhaps not feasible for a large dataset which has many missing values. Other common ways [6] are to use the attribute mean or most probable value to fill in missing values, or to use a learning algorithm to predict the missing values, an example of which is used in Section 4.4.3.

A simple way to handle outliers is to sort the attribute values and partition them into bins and then smooth the noisy data by bin mean, bin median, or bin boundaries. It is also possible to directly group values in clusters and then detect and remove outliers [24].

3.2.2 Data Transformation

Data transformation converts the data into appropriate form for data mining. Common data transformation methods can be:

- **Normalization**

For some attributes it varies much between minimum and maximum values, e.g. 0.1 and 1000. Normalization scales attribute values to fall within a specified range [23], like [0,1]. This is particularly useful for some algorithms such as neural networks and k-Nearest Neighborhood. The two most common normalization methods are:

Let v be the old value for feature A, v' is the normalized value for feature A, the values will be transformed to fall in $[new_min_A, new_max_A]$.

a) Z-score normalization:

$$v' = \frac{(v - mean_A)}{standard_deviation_A}$$

b) Min-max normalization:

$$v' = \frac{v - min_A}{max_A - min_A} (new_max_A - new_min_A) + new_min_A$$

- **Discretization**

Discretization transforms a numeric attribute into a discrete one by creating a set of contiguous intervals (or equivalently a set of split points) that spans the range of the attribute's values [25]. The number of value possibilities of the numeric attribute should be significantly reduced by applying discretization, since large amounts of possible attribute values arouse slowness or ineffectiveness in model building [23, 26].

As a very simple example in Table 4, the numeric attribute *Age* is discretized using conceptual labels (youth, adult, senior) to replace the original values of *Age*.

Table 4: Discretization example

Name	Age		Name	Age
Tom	12	Discretization →	Tom	youth
Amy	25		Amy	adult
Mike	60		Mike	senior
John	17		John	youth
Sophia	33		Sophia	adult
Lily	55		Lily	senior

According to the consideration of class attributes, discretization algorithms can be divided into unsupervised methods and supervised algorithms. Unsupervised discretization algorithms that do not take class information into account are very simple. Two common unsupervised discretization algorithms are equal-width discretization and equal-frequency discretization [23]. Equal-width discretization methods compute the maximum and minimum for the attribute that is to be discretized and divide the range into k equal-width intervals. Equal-frequency discretization method partitions the value range into k intervals so that each interval contains the same number of instances. Supervised discretization algorithms that take class information into account are more complicated. There are many manners in which to do supervised discretization. Chi merge [27] is a simple supervised method that uses the chi-square to do discretization. It sorts the values of the given attribute in ascending order and initially constructs one interval for each value so that they are separate. It then calculates the chi-square for any two adjacent intervals and merges those pairs with lowest chi-square values. It stops when all pairs of adjacent intervals are more than a specified threshold value. More detailed information about discretization can be found in [26].

3.3 Feature construction

In [28] the features used for prediction are described as “easily the most important factor” in determining the success of a machine learning project. In many cases the given features are not sufficient to give high prediction accuracy, so it is necessary to construct new features based on the old ones. This step is always done manually, and based on the understanding of the data with intuition and creativity.


A very common approach to creating new features is to combine numeric features with various operators. Given a set of features $x_1, x_2 \dots$ common new features to generate are like:

- $x_i + x_j$
- $x_i x_j$
- x_i^2
- $\log(x_i)$

A very simple example is, given the *width* and *length* of houses to predict their prices, a possible new feature can be *area*, by multiplying the features *width* and *length*.

Another way to do feature construction is to decompose or split features to create new features. One example is in Section 6.4.3, the *datetime* feature is split into year, month, day of week, hour. Another example is shown in Table 5, a nominal feature *color* with values [red, green, blue] is split into 3 binary features of *red*, *green* and *blue*, 1 means presence of the corresponding color and 0 means the opposite.

Table 5: Split color feature into binary features

color		red	green	blue
red		1	0	0
blue		0	0	1
red		1	0	0
green		0	1	0

3.4 Feature selection

According to [29] the role of feature selection in machine learning is 1) to reduce the dimensionality of data to speed up a learning algorithm, 2) to improve the predictive accuracy of an algorithm and, 3) to improve the comprehensibility of the results.

A simple way to undertake feature selection is to manually remove irrelevant features based on expert knowledge such as intuition, but every hypothesis must be tested whether an undesirable feature is removed, and feature selection based on intuition always depends on the number of features, it is hard to proceed if there is a large number of features. Another way is first to compute a ranking of all features and then choose features relating to it. Three common methods are described as follows:

- **Univariate feature selection**

Univariate feature selection methods consider each feature individually. A statistical test is applied to assign a score to each feature. All features are ranked according to the scores, which indicate the strength of the relationship of the features with the target attribute. Some common methods include the Chi squared test, correlation coefficient and information gain [30].

This method is very easy to employ but is sometimes inaccurate because of consideration of only univariate dependencies.

- **Recursive feature elimination**

The idea behind the recursive feature elimination method is to repeatedly build a model and eliminate the weakest features with every step. Support Vector Machine (SVM) algorithms are commonly used in this method. SVM can assign weights to features, which are used as a measure to eliminate features. At first, a SVM model is trained with all features and weights assigned to each feature. Then the features with the smallest absolute weights are eliminated from the present set of features [31]. The process continues until all features are exhausted. A ranking of features is created according to the sequence in which they were eliminated.

Recursive feature elimination method makes up the shortage of univariate method by considering the dependencies between features. But it is costly and is always dependant on the used model, i.e., the feature selection might be incorrect when using a different algorithm for feature selection, compared to the final algorithm for prediction.

- **Tree-based feature selection**

Some tree-based machine learning algorithms like Random Forest or Extremely randomized Trees can evaluate the relative importance of features. Referencing these feature importance values features are ranked.

Tree-based feature selection is very easy to use but it is unclear where to cut off to obtain the best combination of features, which is also a problem for the other methods, but it is quite unclear with a tree-based method.

Based on these rankings a feature selection approach can be applied. One method is to choose a learning algorithm, for example Random Forest in Section 4.4.6, and use it to repeatedly build models with recursively eliminated features based on the ranking. Each model is evaluated and the best subset of features is the one with the highest accuracy.

3.5 Model validation and selection

In this step, one or more effective models will be selected from a set of candidate models by validation using the given data. One prediction task can fall into two categories: regression or classification. As stated in Section 2.3, for each type of prediction task there are various candidate algorithms to choose from. To find out the most appropriate algorithms we have to do much validation to evaluate all candidates. According to the validation results it is able to choose the best-performing ones

3.5.1 Validation

As stated in Section 1.1 two datasets are given for a Kaggle competition, one is the training set and the other is the test set, which does not contain the target feature. To evaluate the performance of a model there are two possibilities.

- **One way is to use the training set to do local validation.**

A simple method for performing local validation is to split the training dataset into two parts: a training subset and a validation subset. This method is called the “holdout method”. The training part is used for training each algorithm and the validation part is used for estimating the accuracy of the algorithm. This method is simple to apply and takes a relatively short time. However, this method may give rise to a problem in that the subset used for training or validating may not be representative of the dataset as a whole [5], especially for a small amount of data, which can, in turn, give rise to inaccurate evaluations of the model.

An alternative is k -fold cross validation (CV) that can improve the holdout method. In k -fold cross validation the original training dataset is randomly split into k approximately equal-sized subsets. Then the holdout method will be repeated k times, and in each turn one of k subsets is used for validation, and the remaining $k-1$ subsets are used for training. The final validation result is the average of the k results. The commonly used value for k is 10.

-
- The other is the public leaderboard hosted by Kaggle.

The public leaderboard provides a score for each submission, which is calculated using an evaluation metric given by Kaggle. But the score is sometimes calculated only based on a fraction of the test dataset, the fraction varies from competitions. So it is unreliable to totally trust the public leaderboard, it is better to take both local validation and the public leaderboard into account when choosing a model.

3.6 Hyperparameter tuning

A traditional method of performing hyperparameter tuning is grid search, which is simply an exhaustive searching over manually specified parameter values for a model. Grid search will evaluate a model for each combination of parameter values specified in a grid. Sometimes a small grid is ran initially, in which the minimum and maximum and a set of values that fall between the parameters are specified. The goal of running this small grid is to find the ranges that the optimum values of the parameters lie in. Then a more accurate grid is expanded based on the ranges.

For example, to tune the hyperparameters of a Random Forest model, two parameters need to be tuned: number of trees and number of randomly selected features used to look for the best split. The two parameters are denoted as *n_estimators* and *max_features*. Initially, a rough grid can be designed as follows:

n_estimators: {50, 200, 400, 600, 800, 1000}

max_features: {0.1, 0.3, 0.5, 0.7, 0.9, 1.0}

where the float number specified to *max_features* means a percentage of features considered to look for split.

Suppose the optimum values of the two parameters are {600,0.5}, and then a more accurate grid can be expanded as follows:

n_estimators: {500, 550, 600, 650, 700}

max_features: {0.4, 0.5, 0.6 }

Grid search is very simple to set up but since grid search is exhaustive, it is therefore potentially expensive for a high-dimensional space. A random search that simply samples parameter settings from a random distribution for a fixed number of iterations can be an alternative [32]. A model will be evaluated for each combination of parameters chosen. For example, for hyperparameter tuning for a Random Forest model using a random search, a search space can be set as follows:

n_estimators: [50, 1000]

max_features: [0.1, 1.0]

Pairs of values will be chosen randomly from this space for 60 iterations.

Random search is significantly cheaper than grid search but it is also possible that random search cannot find the optimum found by grid search.

Both methods must be guided by some performance metric, typically measured by CV on the given training set.

4 Kaggle competition 1: Forest Cover Type Prediction

This has been chosen as the opening competition because it has a relatively low difficulty (Kaggle category: Playground) and suitable data amount given to approach the competition.

In this chapter I will at first describe the main goal of the challenge and the dataset given by the challenge host. Then I will describe the steps I took to solve this problem in chronological order. Finally, a conclusion will be drawn and lessons I learned from this competition will be given.

4.1 Problem Description

The Forest Cover Type Prediction [33] competition is a typical supervised multi-class classification problem. In this competition the participants are asked to predict the forest cover type using cartographic features. Each instance is sampled on 30 x 30 m squares of the Roosevelt National Forest in northern Colorado. The forests in these areas have seen few human-caused disturbances, therefore existing forest cover types are not a result of forest management practices but ecological processes.

4.2 Dataset

The dataset is a public dataset and is provided by Jock A. Blackard and Colorado State University and hosted by UCI machine learning repository [34]. Two datasets are provided to do the prediction task: a training set and a test set. The training set consists of 15,120 instances, containing both cartographic features and the forest cover type. The test set consists of 565,892 instances, containing only cartographic features. There are 54 cartographic features and one target feature that is to be predicted, all of them are listed in Table 6.

Table 6: Description of the features for competition Forest cover type prediction [35]

Feature	Type	Description
Elevation	numerical	Elevation in meters
Aspect	numerical	Azimuth from true north
Slope	numerical	Slope in degrees
Horizontal_Distance_To_Hydrology	numerical	Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology	numerical	Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways	numerical	Horz Dist to nearest roadway
Horizontal_Distance_To_Fire_Points	numerical	Horz Dist to nearest wildfire ignition points
Hillshade_9am	numerical	Hillshade index at 9am, summer solstice 0 to 255 index
Hillshade_Noon	numerical	Hillshade index at noon, summer solstice 0 to 255 index
Hillshade_3pm	numerical	Hillshade index at 3pm, summer solstice 0 to 255 index
Wilderness_Area	nominal	Wilderness area designation 4 binary columns, 0 = absence or 1 = presence
Soil_Type	nominal	Soil Type designation 40 binary columns, 0 = absence or 1 = presence
Cover_Type	nominal	Forest Cover Type designation 7 types, integers 1 to 7

4.3 Evaluation metric

Submissions are evaluated on a simple multi-class classification accuracy, which is defined as the number of correctly classified instances divided by the total number of instances:

$$\frac{\text{Number of correct predictions}}{\text{Number of instances}}$$

Every participant has 5 chances to submit an entry per day. The leaderboard is calculated based on all of the test data.

4.4 Approach and progress

The content in this subsection is written chronologically to give a better understanding of the decisions I received while trying to solve this problem. This method of describing is also used in the other competitions.

4.4.1 Explore the data

The data contains no dates and no strings and no null values. For the soil type and wilderness type it is necessary to inspect if some instances are allocated into more than one class, or not allocated at all. I summarized the binary columns respectively and all values equal 1, this states that there are no errors. For the numeric attributes we have to check if the zeros contain any missing values.

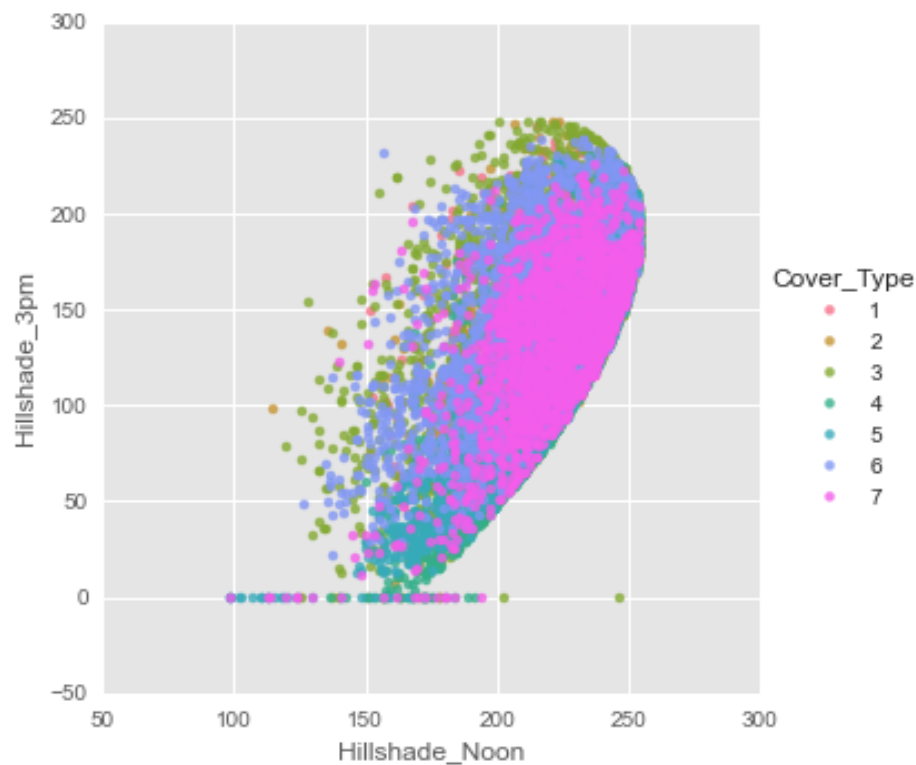


Figure 5: Hillshade_Noon vs. Hillshade_3pm

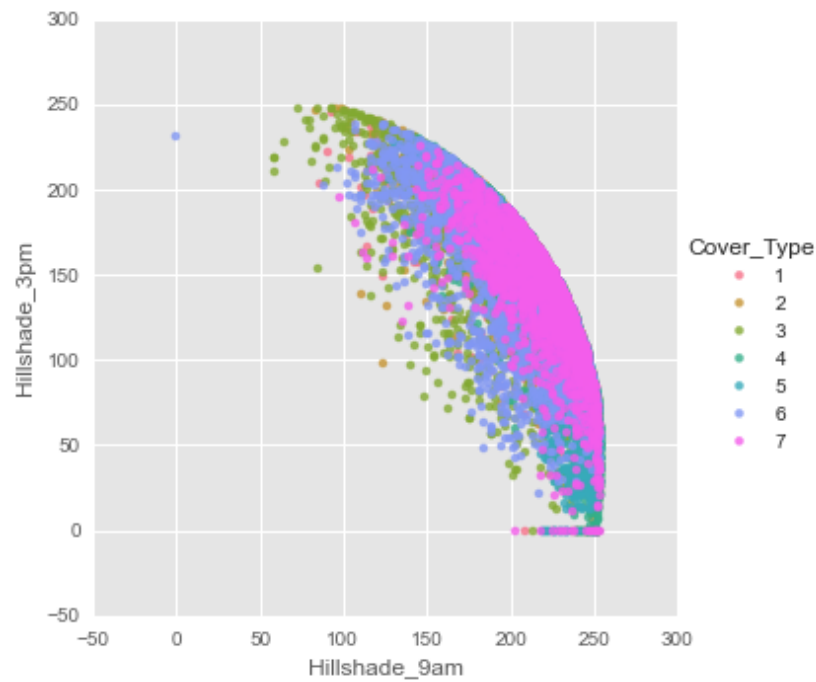


Figure 6: Hillshade_9am vs. Hill_shade_3pm

Feature *Hillshade_3pm* contains many zeros. If there was shade at 9 AM and noon, it is logical to expect little shade at 3 PM, but 0 does not sit right. Figure 5 and Figure 6 show that many points of *Hillshade_3pm* are zeros while in the same instances the shade is not zero at noon and 9 AM. It is possible to speculate the zeros of hill shade at 3 PM are very possibly missing values.

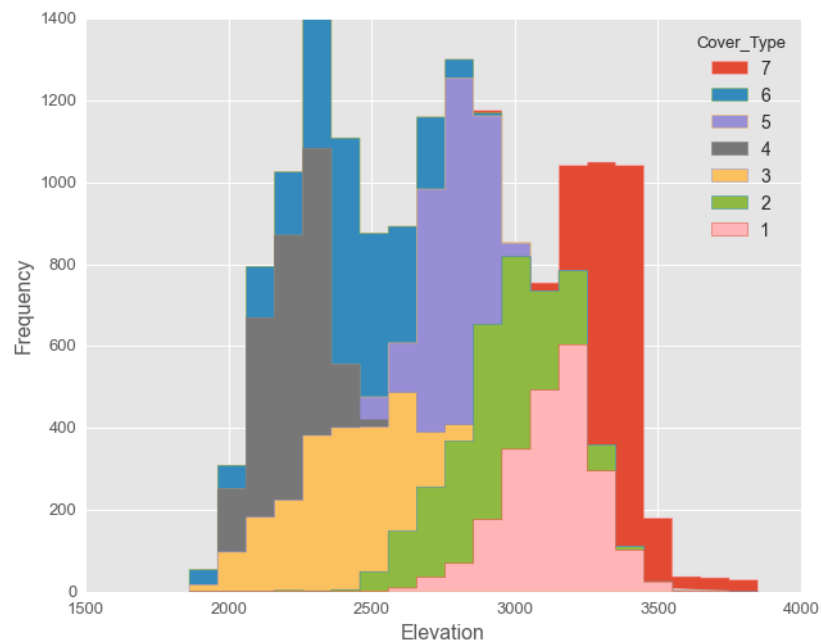


Figure 7: Histogram of Elevation: different cover types are marked with different colors

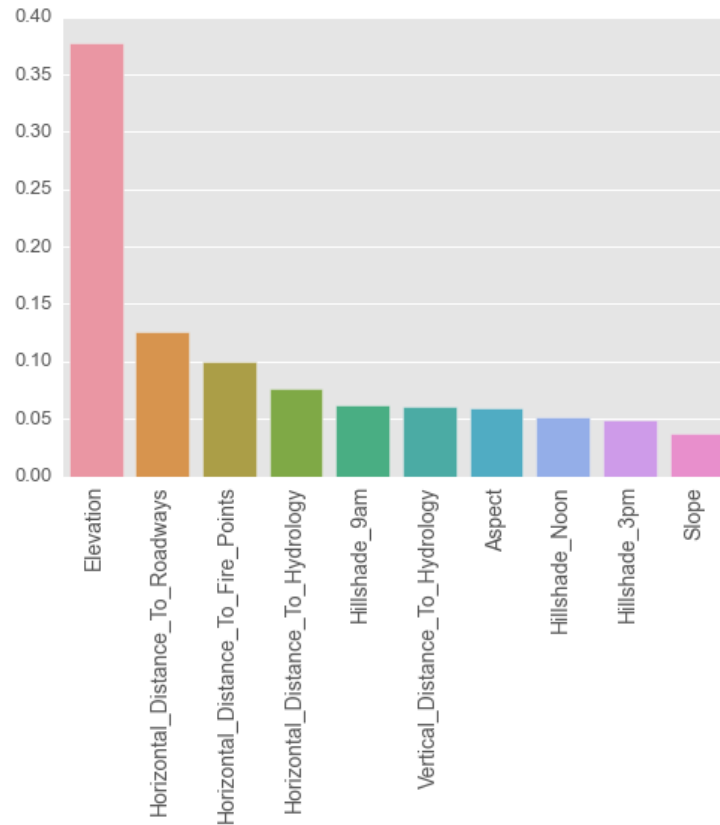


Figure 8: Feature importance computed by random forest model

To have a better understanding of the features' predictive power, I computed the feature importance of the attributes without *Willderness_area* and *Soil_Type* showed in Figure 8, which indicates *Elevation* has the strongest predictive power. Figure 7 shows that *Elevation* alone can effectively separate cover types 4, 5 and 7.

4.4.2 Get started

To quickly get started on the competition, first I have run a Random Forest model with 100 trees on the original data and submitted it to Kaggle. This model got a score of 0.75099 on the leaderboard with a ranking of 813, which is in the approximate middle of the leaderboard.

Using this Random Forest model with 100 trees and 30% of the train data as test data and the rest as training data I got a confusion metrics and it is visualized in Figure 9. The confusion matrix shows that class 1 and class 2 are the most frequently misclassified and this is also the key problem of this competition.

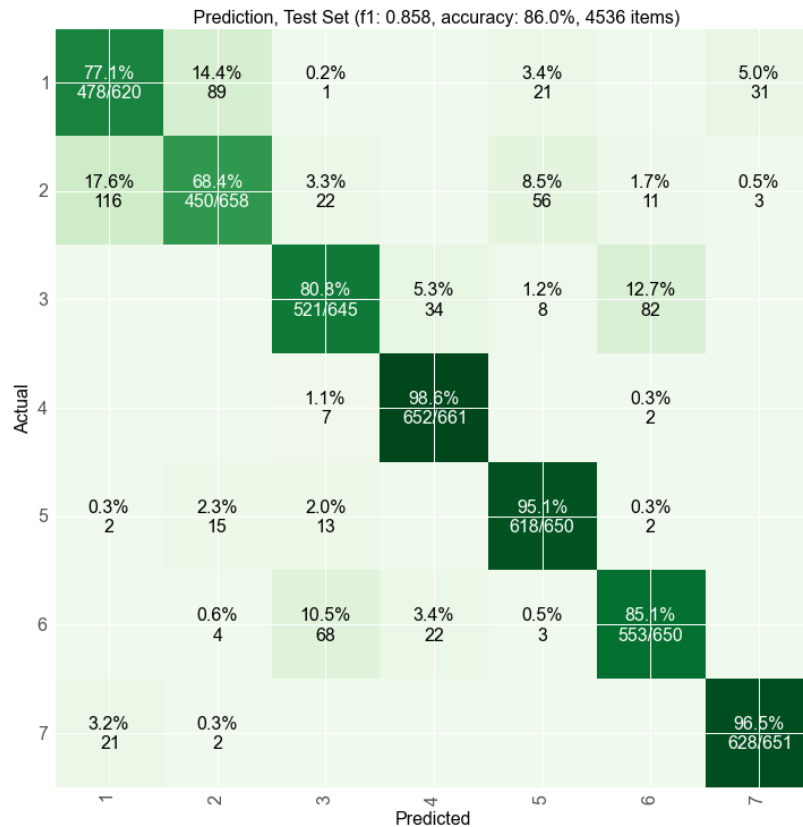


Figure 9: Confusion matrix of random forest

4.4.3 Filling in missing values

Since the forest cover type is not needed when filling in missing values of *Hillshade_3pm* we can utilize both the train and test data to train a model. With a 10-fold cross validation score of 0.999 as a convincing argument, I applied a Random Forest regressor to fill in the missing values for feature *Hillshade_3pm*.

After handling the missing values I used the new dataset to get a prediction with the Random Forest classifier. This submission got a score of 0.75301 on the leaderboard. This strategy gives a little improvement of about 0.003 on the last submission.

4.4.4 Combination of binary columns

The 4 binary columns of *Wilderness_Area* and 40 binary columns of *Soil_Type* can be respectively combined by summarizing as 2 nominal attributes.

After applying this procedure to the train and test dataset I tried again to submit a new entry, this scored 0.75084 on the leaderboard, which is not an improvement, the opposite, even. I used the original binary columns instead of the combined features in the following approaches.

4.4.5 Construction of new features

The data has 5 features in meters, I have done some calculation and combination based on these attributes. Distance features can be divided into two types: vertical and horizontal. For all vertical distance features I combined them pair-wise with plus and minus operators, and the same for all horizontal distance features. Using horizontal distance and vertical distance to hydrology I calculated the Euclidean distance to hydrology. All new features are listed in Table 7.

Adding the constructed new features to the dataset I got a score of 0.78359. This score is about 0.03 higher than the highest score mentioned above, which is a big improvement and allowed me to jump to 316th place on the leaderboard.

Table 7: New constructed features

New Features	Formulation
Ele_minus_VDtHyd	Elevation -Vertical_Distance_To_Hydrology
Ele_plus_VDtHyd	Elevation +Vertical_Distance_To_Hydrology
Distance_to_Hydrology	$\text{Horizontal_Distance_To_Hydrology}^{**2} + \text{Vertical_Distance_To_Hydrology}^{**2}^{**0.5}$
Hydro_plus_Fire	Horizontal_Distance_To_Hydrology+ Horizontal_Distance_To_Fire_Points
Hydro_minus_Fire	Horizontal_Distance_To_Hydrology - Horizontal_Distance_To_Fire_Points
Hydro_plus_Road	Horizontal_Distance_To_Hydrology+ Horizontal_Distance_To_Roadways
Hydro_minus_Road	Horizontal_Distance_To_Hydrology - Horizontal_Distance_To_Roadways
Fire_plus_Road	Horizontal_Distance_To_Fire_Points+ Horizontal_Distance_To_Roadways
Fire_minus_Road	Horizontal_Distance_To_Fire_Points - Horizontal_Distance_To_Roadways

4.4.6 Feature selection

I used a tree-based feature selection method with a Random Forest model. 63 Random Forest models were built with recursively-removed features beginning from the most unimportant, based on the feature importance ranking calculated by a Random forest model. The trend of scores becomes relatively flat from about 26 features onwards. I then tried to predict respectively with 26, 43 and 60 features, the other two had the highest validation scores. Table 8 shows the 10-fold cross-validation scores based on the training data and scores on the lead-

erboard of Kaggle. Feature selection provided very little improvement by eliminating the 3 least important features. But I still excluded these three features in the following steps.

Table 8: Validation scores and leaderboard scores

Number of features	Cross validation	Leaderboard
26	0.79702	0.76790
43	0.80529	0.78113
60	0.80708	0.78370
Total (63)	0.80714	0.78359

4.4.7 Model selection

Random Forest performed very well, but it is still worthwhile to try other algorithms. By validating other models with the default parameters, I selected the two best-performing models: Random Forest with 10-fold CV score 0.77612 and Extra Trees with 10-fold CV score 0.79120. To have a better understanding of the performance of Extra Tress I tried this model with 100 trees like Random Forest, this gave a leaderboard score of 0.80767, which took my ranking up to 133.

4.4.8 Further feature selection

As another feature selection approach I tried to exclude one of all features except soil type and wilderness type with each step of the evaluation. Setting the score of total features as a benchmark, the scores higher than this will degrade prediction performance. Since the cross-validation in every turn was random, the ranking was always different. I tried 4 loops and decided to reduce two features—*slope* and *Vertical_Distance_to_Hydrology*, which appeared below the benchmark all four times. Using an Extra Trees model and reducing these two features resulted in a score of 0.80986 on the leaderboard.

4.4.9 Parameter tuning

For a refinement of the performance of the models, Random Forest and Extra Trees, which are selected in the model selection phase, I used grid search to seek the best parameters. For these two algorithms two parameters are tuned, which are the number of trees and the number of randomly selected features to seek split.

The submissions by applying the tuned Random Forest got a score of 0.78871 on the leader board. The Extra Trees model with optimum parameters scored 0.81256 on the leaderboard.

4.4.10 Ordered One-vs.-All

As stated in Section 4.4.2, the key problem of this competition is to reduce the high rate of misclassification of cover types 1 and 2. Figure 10 shows a confusion matrix that is produced by an Extra Trees model. It shows that type 1 and type 2 are still hard to separate and some type 1s are also misclassified as type 5 and 7; some type 2s are misclassified as type 4, 5 or 6. Because of the high prediction accuracy of types 7, 6, 5, 4, I used an ordered One-vs.-All approach for classes 7, 6, 5, 4 to prevent inaccurate prediction for types 1 and 2. The classification process is demonstrated in Table 9.

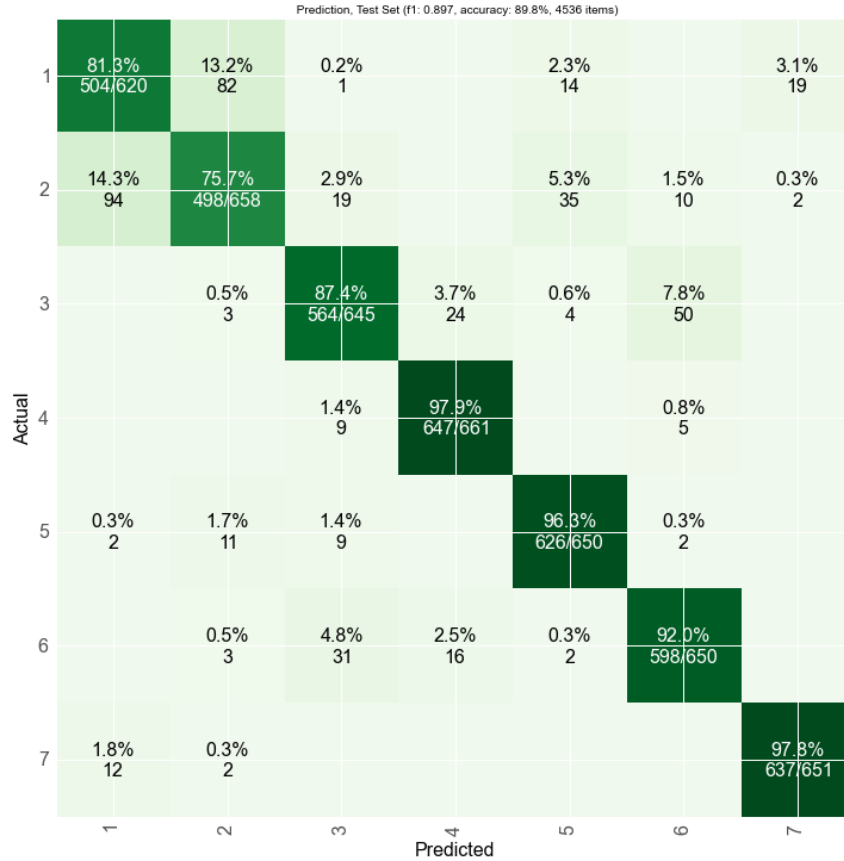


Figure 10: Confusion matrix of extra trees

Table 9: Ordered One-vs. -All

Cover type	Training set	Test set	Classified instances
7	x_7 vs. $(x_1+x_2+x_3+x_4+x_5+x_6)$	y	y_7
6	x_6 vs. $(x_5+x_4+x_3+x_2+x_1)$	$y - y_7$	y_6
5	x_5 vs. $(x_4+x_3+x_2+x_1)$	$y - y_7 - y_6$	y_5
4	x_4 vs. $(x_3+x_2+x_1)$	$y - y_7 - y_6 - y_5$	y_4
3, 2, 1	$x_3+x_2+x_1$	$y - y_7 - y_6 - y_5 - y_4$	y_3, y_2, y_1

Let x_i be the subset of the training data, which is labeled with cover type i , and y_j be the subset of test data y , which is predicted to be cover type j .

For example, to predict type 7, complete training dataset is used in the training phase, but the instances of cover type 7 are labeled as positives and all other instances are labeled as negatives to prevent other types (especially type 1) from being misclassified as type 7. After prediction, the test instances classified as cover type 7 are labeled “7” and are removed from the test set. The classification process for cover type 6 is the same as that of 7, but the training instances of cover type 7 are removed from the training set. Repeat the same approaches for cover types 5 and 4 and we get the classified instances y_7, y_6, y_5 and y_4 . Types 1, 2, 3 are classified using the general multi-class classification method.

The Extra Trees algorithm, with parameters selected from the previous step, was used in each step to get a submission, and it scored 0.82269 on the leaderboard.

4.4.11 Final result

Up to now the competition is closed, my final submission scored 0.82269 and ranked 29th out of 1694 teams, which is quite a satisfactory result. An overview of the leaderboard is shown in Table 10. The row in bold is my score.

Table 10: Public leaderboard overview of first competition

Ranking	Team Name	Score
1	antgleb	1.00000
2	Ashish Singh	0.99999
3	ucbw207_2_forest	0.99751
...
27	Michiel	0.82408
28	DATS36G	0.82281
29	Ying Dong	0.82269
30	hderksen	0.81963
31	Jack Dempsey	0.81950
...
1692	soumyajyoti	0.00003
1693	Ethan Rosenthal	0.00000
1694	Hiokei Chan	0.00000

4.5 Conclusion

The goal of this challenge is to predict forest cover type based on cartographical data. Different data processing strategies and classification models are used to make predictions. The new feature construction approach gave the most significant improvement and the Extra Trees model was chosen as the final model. The ordered One-vs.-All strategy with Extra Trees model obtained the best result. To have a clear insight of the total process of this competition, I plotted 9 representative scores and corresponding standings in Figure 11.

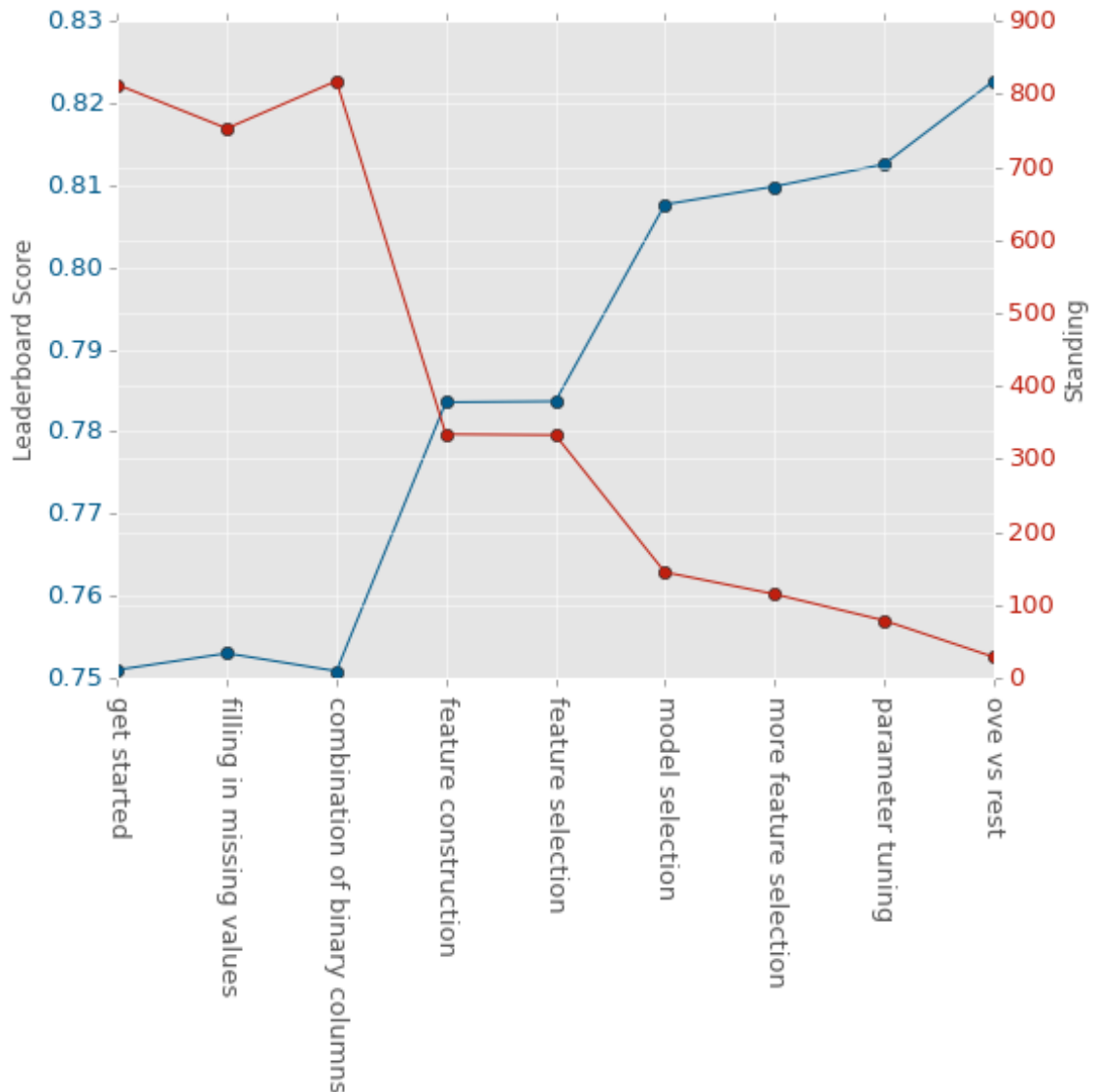


Figure 11: Leaderboard Scores and Standings for competition forest cover type classification

4.6 Lessons learned

1. It is very important to get started with a model as soon as possible.

-
2. Feature engineering can greatly improve prediction accuracy.
 3. Another lesson I can take away from this competition is that cross validation helps significantly in the model selection and parameter tuning phase.
 4. This is my first Kaggle competition and deciding how to approach it I felt overwhelmed. After an experimental trial, a pipeline was designed from this competition. This pipeline serves as a good guideline and should be used for upcoming competitions.

5 Kaggle competition 2:Otto Group Product Classification

This competition [36] has a higher-level difficulty (Kaggle category: Featured) than the first one. I wanted to check if the pipeline stemming from the first one is also applicative for harder competitions, so this competition was chosen. Also, it has a proper amount of data that a personal computer can manage.

At first I will describe the main goal of this competition, the dataset given by the competition host and evaluation metric. Then I will describe in chronological order the various approaches I have taken to solve this problem. Lastly, a conclusion and lessons learned from this competition will be given.

5.1 Problem Description

The Otto Group is a large e-commerce company. They posed this competition to ask researchers to classify products into one from nine categories, aiming at achieving better product analysis by improving the ability to accurately classify similar products.

5.2 Dataset

This competition provides a training dataset including 61,878 instances used for training models, and a test set containing 144,368 instances to be classified. Each instance represents a single product. 93 obfuscated features are provided for both datasets, which represent counts of different events.

The randomly selected products are to be classified into nine categories. Each target category corresponds to one of the most important product categories (like fashion, electronics, etc.).

All features are listed in Table 11.

Table 11: Description of features for competition Otto group products classification [37]

Feature	Type	description
id	numeric	anonymous id unique to a product
feat_1, feat_2, ..., feat_93	numeric	various features of a product
target	nominal	the class of a product

5.3 Evaluation metric

The evaluation metric uses the multi-class logarithmic loss (logloss). This evaluation metric is being used to penalize heavily in case a wrong prediction comes up. Each product has been defined with one true category. For each product, we are asked to submit a set of predicted probabilities (one for every category). The formula of logloss is as follows:

$$\text{logloss} = -1 * \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(p_{ij})$$

Where N represents the total number of products in the test set, M is the number of categories, \log is the natural logarithm, y_{ij} equals 1 if product i is in category j and 0 otherwise, and p_{ij} is the predicted probability that product i belongs to category j . p_{ij} lies in $[0,1]$, to prevent infinity ($\log(0)$) the submitted predicted probabilities are substituted with:

$$\max(\min(p, 1 - 10^{-15}), 10^{-15})$$

The logloss is always greater than 0. The lower value a logloss has, the better the prediction is.

For this competition the leaderboard is evaluated on approximately 70% of the test data and the final results will be based on the other 30%, so the final standings may be different from the initial leaderboard ranking.

5.4 Approach and progress

5.4.1 Data exploration

The features are completely obfuscated; the meaning behind the 93 features and what the 9 categories are remains unknown. We only know the features are integer counts and contain many zeros. Figure 12 shows how the 9 product categories are distributed.

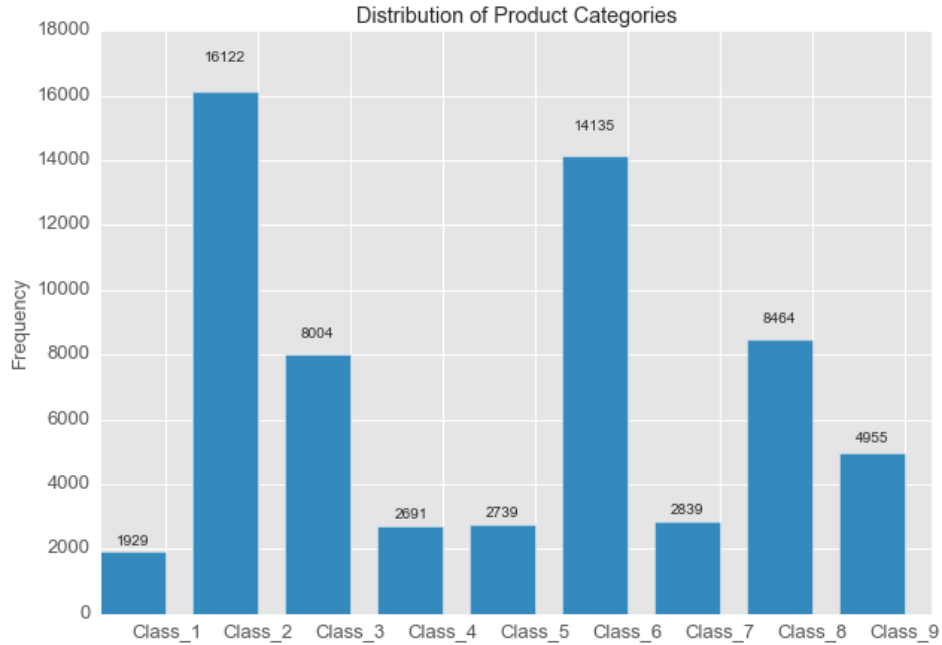


Figure 12: Distribution of Product Categories

There are many features, in Figure 13 I plotted the correlation matrix only for the 15 most important features selected by Random Forest to see if they are closely correlated. We can see most of them do not have high correlation, only *feat_14* and *feat_25*'s correlation is higher than 0.5.

The data seems tidy and clean and there is no necessity to preprocess the data.

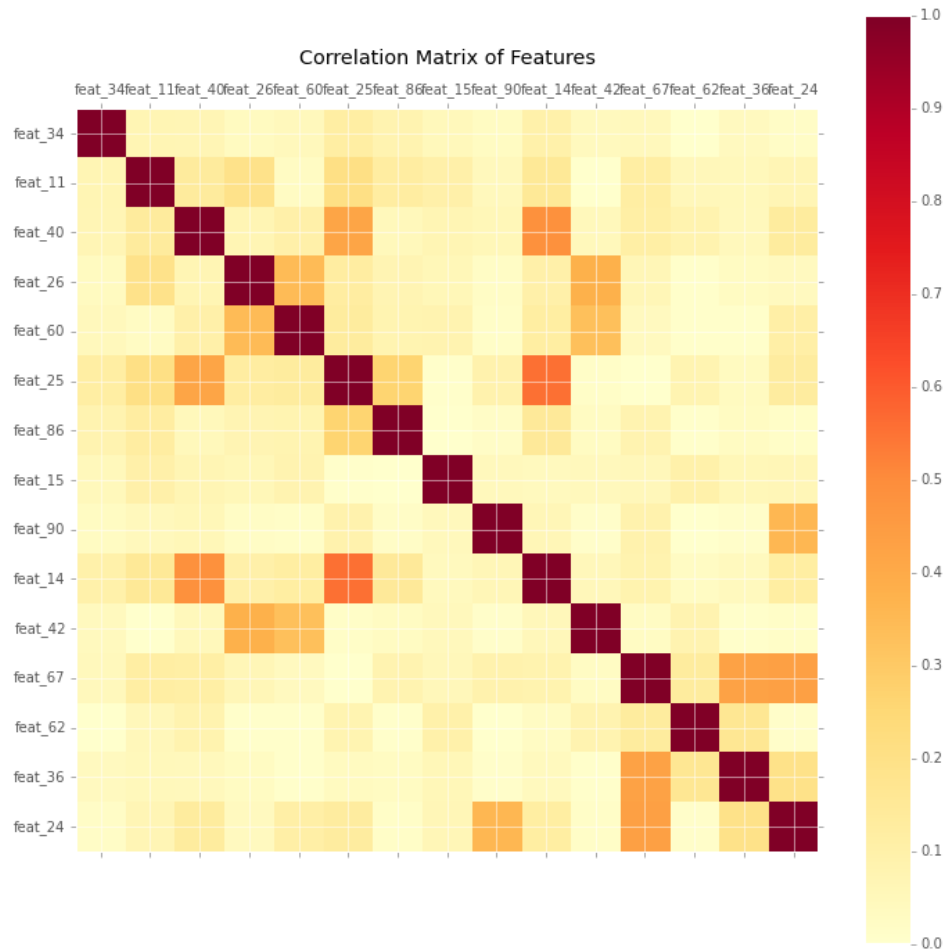


Figure 13: Correlation Matrix of 15 most important features

5.4.2 Beat the benchmark

The competition administrators have provided a benchmark using a Random Forest model with 10 trees, which resulted in a score of 1.50241 on the leaderboard. To get started on the competition and to have more insights into the subsequent procedure, I decided to run and submit the benchmark code, but setting the Random Forest model with 100 trees. This submission got a score of 0.58653, which has beat 40% of all competitors.

5.4.3 Feature construction

Since all features have been obfuscated, it is impossible to construct new features by learning from their meanings. I added only three new features: mean value, standard deviation, and count of non-zero values of every instance with the exception of id and target attributes. The standard deviation is calculated as:

$$standard\ deviation = \sqrt{\frac{\sum_{i=1}^N (x_i - mean_x)^2}{N}}$$

But the new features proved ineffective. With the new features I got a score of 0.59432 on the leaderboard.

5.4.4 Feature selection

At first I tried a recursive feature elimination method with a linear Support Vector Classification [38] model to do feature selection, but there was no significant evidence to discard a given feature.

Then as in the forest cover type competition I tried to run local validation recursively with top-n (n from 20 to 93) features based on the feature importance computed by Random Forest, the final result did not show an obvious improvement compared to the result of applying all features. Even so, I submitted one prediction with 90 features, which saw a slight advantage, and this submission achieved a score of 0.58840. Even after feature selection the score was lower than the prediction directly on raw data, so I used the raw dataset in the following steps.

5.4.5 Model selection

Two models are selected in this step: Random Forest and GBRT. Random forest got a score of 0.58653 on the leaderboard and GBRT scored 0.59601.

5.4.6 Parameter tuning for random forest

I ran a grid search to find the optimum parameters for Random Forest and GBRT. The parameters tuned are same as in Section 4.4.9. The tuned Random Forest achieved a score of 0.53472. The hyperparameters tuned for GBRT were number of trees and the learning rate. The tuned GBRT model scored 0.50902.

5.4.7 Probability Calibration for random forest

Random Forest averages a number of decision tree classifiers on various sub-samples of the dataset, according to [39] the high variance in the underlying based trees will bias probability estimation away from the values which should be near 0 or 1, model like Random Forest can have difficulty making predictions close to 0 and 1. That results in a greater logloss error because of increased $-1 \cdot \log(p_{ij})$ in the logloss formula which is introduced in Section 5.3.

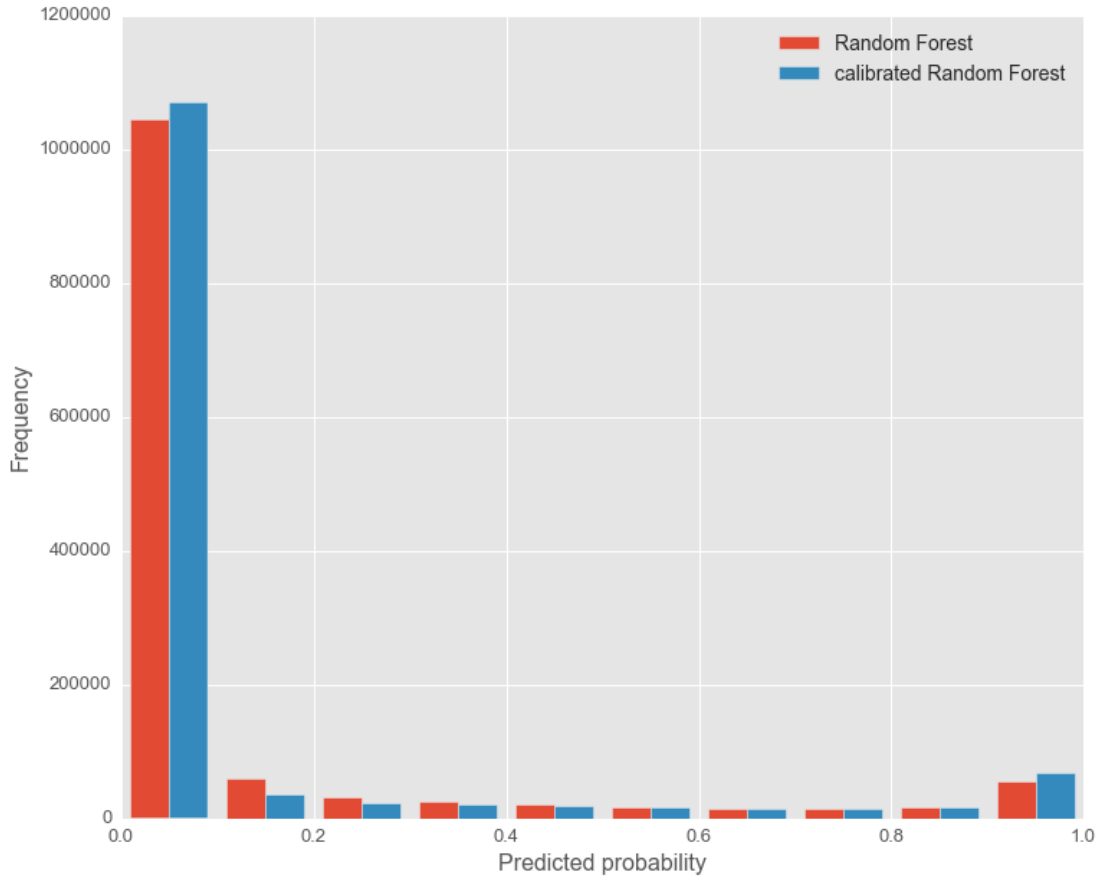


Figure 14: Probability distribution after calibration

Getting inspiration from the Kaggle competition forums, I used probability calibration methods to improve Random Forest. Probability calibration is a post-processing technique that can improve the probability estimation or the error distribution of an existing model. It performs a cross validation on the training instances and test instances. An average of predicted probabilities for each fold is then calculated for the final result to reduce variance. It has been tested by [39] that the calibration technique works for the Random Forest model once the calibration sets are large enough. The calibrated Random Forest model scored 0.48998 on the leaderboard. Figure 14 shows the change in distribution of the probability predictions after calibration. The percentage of predicted probabilities near 0 or 1 increased, which tends to decrease logloss error.

5.4.8 Using Xgboost

Referencing some advice in the Kaggle forum I tried Xgboost—Extreme Gradient Boosting. It is a very efficient Gradient Boosting package, which is widely used in Kaggle competitions. The default Xgboost scored 0.47409 on the leaderboard.

5.4.9 Tuning Xgboost

Xgboost has more parameters to be tuned, and the exhaustive grid searching required is expensive and very time-consuming. Detailed information regarding the parameters can be found in [40]. I ran a randomized search that sampled parameter settings 70 times to yield a superior model. This approach resulted in a score of 0.43237.

5.4.10 Mean stacking

From the 70 random searches of good parameters I encountered 7 outputs that hit a validation score between 0.43 and 0.44. I used a very simple ensemble method to get the average output produced by 7 Xgboost models with different parameter settings. The idea behind it is that the errors are also “averaged” when getting the mean. This little trick scored 0.42932 on the leaderboard.

5.4.11 Final result

The public leaderboard was calculated on approximately 70% of the submissions before the end of the competition and the final results would be based on all test datasets. So the final standings may be different after validation.

According to the rules of the competition, every participant was permitted 2 submissions and the best one would be my final leaderboard score. Since the public leaderboard appeared to be based on a large enough set (70% of test set), I selected the 2 submissions with the best public leaderboard scores. When the competition was concluded, my final private score was 0.43002, this is the score of the mean stacking of 7 Xgboost models described in section 5.4.9. Compared to the public score 0.42932, although the private score went down, the final ranking saw slight improvement, from 354th to 347th, out of 3,514 teams. A brief overview of the leaderboard is shown in Table 12.

Table 12: Private leaderboard overview of second competition

Ranking	Team Name	Private Score
1	Gilberto Titericz & Stanislav Semenov	0.38243
2	¬_(ツ)_/¬	0.38656
3	i dont know	0.38667
...
345	SagaU	0.42978
346	y	0.42997
347	Ying Dong	0.43002
348	TeamNeuland	0.43007
349	lvchen1989	0.43007
...
3512	jcis	33.49091
3513	stan	33.82664
3514	Pierre-Loic Doulcet	34.53878

5.5 Conclusion

In this competition I started with a Random Forest model and got a promising result. Feature engineering and selection did not contribute to the prediction, so I just used raw data in the later processes. The Random Forest and GBRT model saw a large improvement after tuning the parameters. The Random Forest model surpassed 0.5 on the public leaderboard, which was very encouraging. The Xgboost attempt greatly helped. The default Xgboost had already achieved a better score than the calibrated Random Forest. After using a random search of the parameters for Xgboost, and averaging the 7 best outputs, I reached the top 10% on the leaderboard. Figure 15 shows how the different approaches performed throughout the entire process.

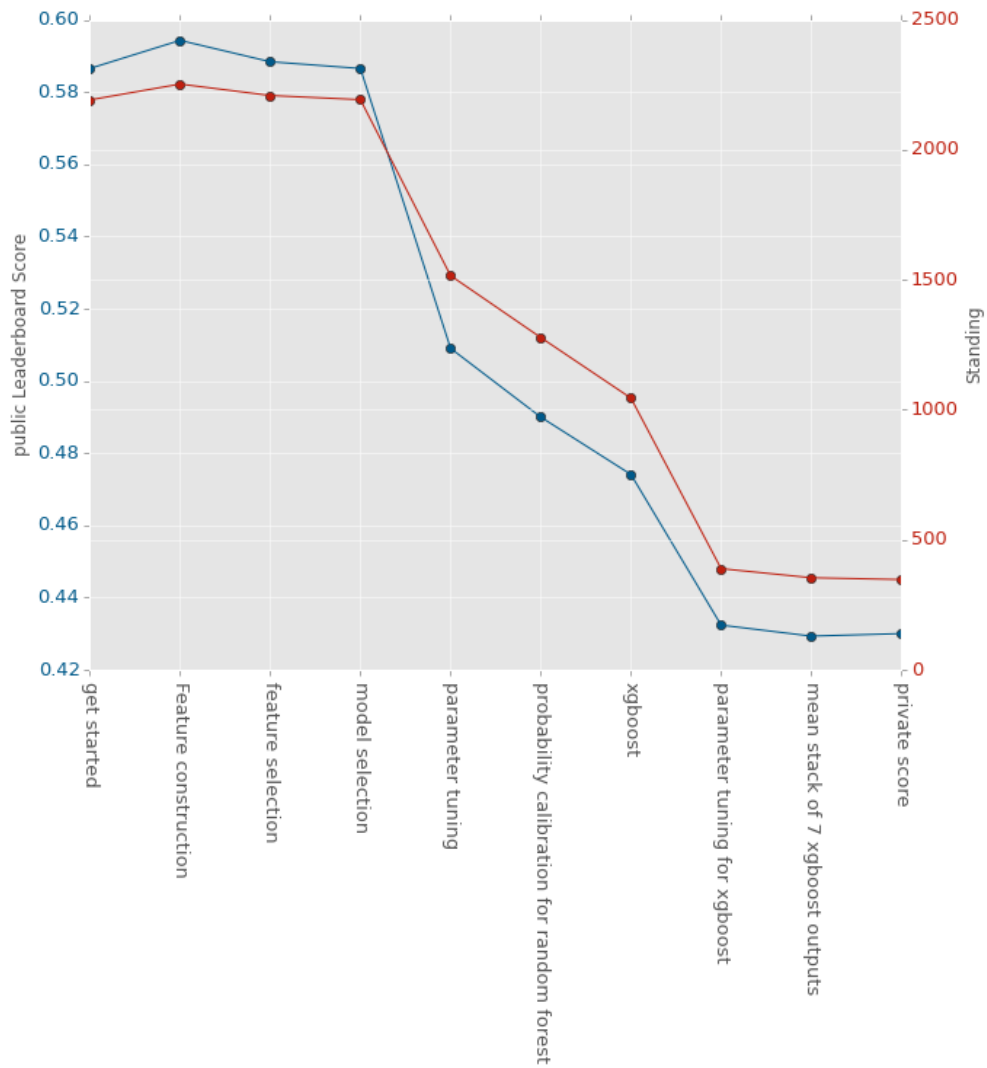


Figure 15: Score vs. Standing for competition Otto group products

5.6 Lessons learned

1. As more and more new and efficient machine learning models like Xgboost appear, we should always stay hungry for knowledge.
2. Some steps of the pipeline are not effective for the competition, but we cannot know whether they are useful without trying them.

6 Kaggle competition 3: Bike Sharing Demand

This competition [41] was chosen because it is a regression problem, which makes it different from the other two. The goal is to check if the pipeline is applicable to the Kaggle competition, which solves a regression problem.

In this chapter the main goal of the competition and the dataset given by the challenge host are firstly described. Then I will describe chronologically what I have done to solve this problem. Lastly, a conclusion and lessons learned from this competition will be given.

6.1 Problem Description

Bike sharing systems are a mean of renting bicycles that let people rent a bike from one location and return it to a different place, automated via a network of kiosk locations throughout a city. In this competition, participants are asked to use data available prior to the rental period to forecast bike rental demand in the Capital Bike share program in Washington, D.C.. Different from the other two competitions in Chapter 4 and Chapter 6 this is a typical regression problem.

6.2 Dataset

Table 13: Features for competition bike sharing demand [42]

Feature	Type	Description
datetime	datetime	hourly date + timestamp
season	nominal	1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday	nominal	whether the day is considered a holiday
workingday	nominal	whether the day is neither a weekend nor holiday
weather	nominal	1: Clear, Few clouds, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	numeric	temperature in Celsius
atemp	numeric	"feels like" temperature in Celsius
humidity	numeric	relative humidity
windspeed	numeric	wind speed
casual	numeric	number of non registered user rentals initiated
registered	numeric	number of registered user rentals initiated
count	numeric	number of total rentals

The data provided spans the two years from January 1, 2011 to December 31, 2012. The training set contains the first 19 days of each month and consists of 10,866 instances, while the test set data contains the remaining days in each month and consists of 6,493 instances. The final goal is to predict the total count of bike rentals during each hour covered by the test set. The numbers of casual and registered users are only provided in the training set. The features and description are listed in Table 13.

6.3 Evaluation metric

The submissions are evaluated by the Root Mean Squared Logarithmic Error (RMSLE). The formula of RMSLE is as follow:

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

Where n is the number of instances in the test set, p_i is the predicted number of total rentals, a_i is the actual number of total rentals, $\log(x)$ is the natural logarithm of x .

The RMSLE is higher when the discrepancies between predicted and actual values are larger. Compared to Root Mean Squared Error (RMSE), RMSLE does not heavily penalize huge discrepancies between the predicted and actual values when both values are huge. In this cases only the percentage differences matter since $\log(p_i+1) - \log(a_i+1)$ can be rewritten to be $\log((p_i+1)/(a_i+1))$.

6.4 Approach and process

6.4.1 Data exploration

The dataset is mixed with categorical and numerical features. First I wanted to check the correlations between features. Figure 16 shows some highly correlated attributes: temp & attempt, registered & count.

The feature *Datetime* in the dataset is playing the role of index, but contains significant information like year, month, day of the week, and hour. I anticipate these will serve as important features for characterizing the bike demand at any given moment.

Figure 17 shows the mean values of total count of rentals, number of registered users and number of casual users grouped by day of the week. As visualized in Figure 17, for registered users, the demand for bikes is higher on weekdays compared to the weekend. On the other hand, for the casual users, Figure 17 shows there is more demand on the weekend compared to weekdays.

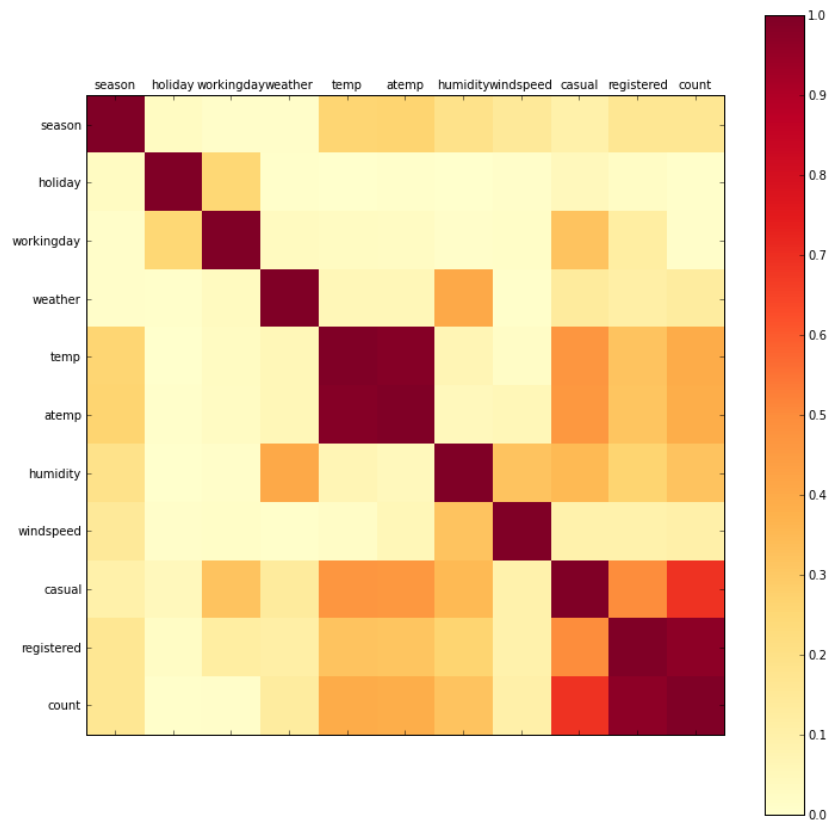


Figure 16: Correlation between features

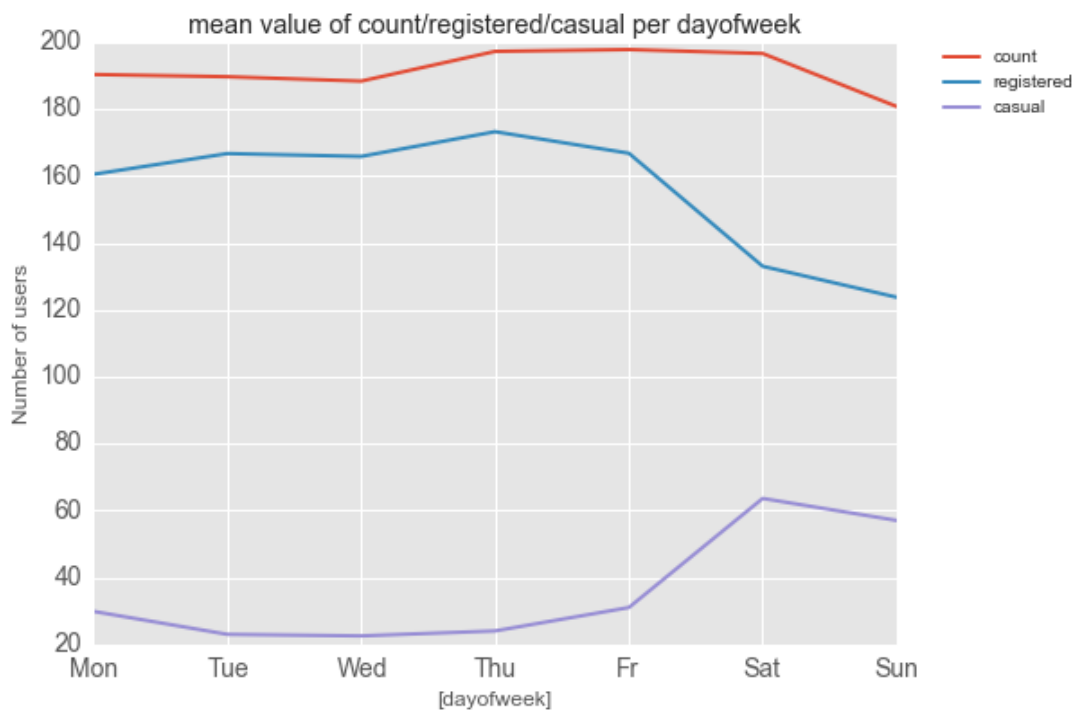


Figure 17: Mean value of count/registered/casual per day of week

I plot the mean values of the total rental count, number of registered users, and number of casual users per hour in Figure 18.

Demand for bikes by registered users tends to peak between 7am and 9am and between 5pm and 7pm, whereas casual users tend to rent bikes more often in the afternoon. As mentioned above, the total count of rentals is highly correlated with the number of registered users.

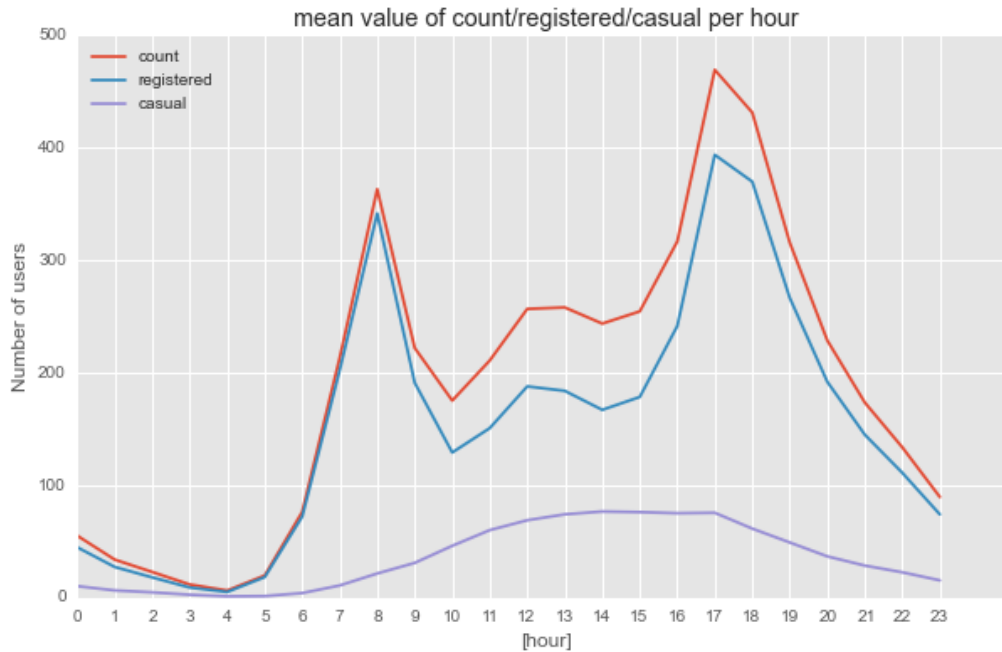


Figure 18: Mean value of count/registered/casual per hour

6.4.2 Get started

As I have proceeded in the other two competitions, I first tried a Random Forest regression model with 100 trees to get started on the competition. Directly predicting the count gave a RMSLE of 1.36508 on the leaderboard. This submission landed around 2,800th place. This beats about 25% of all participants.

6.4.3 Feature extraction from *datetime*

As mentioned in Section 6.4.1, we can extract several useful attributes from *datetime*. The new features are: year, month, day of week, hour. These new features have strong predictive power and improved the leaderboard score to 0.47144, the standing was also raised about 1,500 places.

6.4.4 Feature construction

Because of a lack of domain knowledge I only created two new features, which are the absolute value of difference between temperature and “feels like” temperature, and which temperature is higher. With these two new features and Random Forest model, I achieved a score of 0.47709 on the leaderboard, but the local 10-fold cross-validation error was only 0.32878.

Since the train and test datasets are separated according to date in the month, to decrease the validation deviation and referencing advice on the competition forum I tried to split the training set by dates and use the two consecutive date as test set in every turn of validation, like (10,11), (11,12), ... (18, 19), and the rest as training data, which is closer to the real station. This new validation approach only slightly decreased the deviation of the local validation score, from 0.32878 to 0.33629 for the last approach.

6.4.5 Predict logarithm of count

As written in section 6.3, the evaluation metric is RMSLE, our actual aim should be the natural logarithm of bike demand at different times, demand count plus one, in order to avoid infinities where demand is null. Predicting the $\log(\text{count}+1)$ gave a score of 0.43880 on the leaderboard, which was a big improvement. This entry reached about 680th place.

6.4.6 Feature selection

Referencing the feature importance rankings calculated by Random Forest, I ran many times of 10-fold cross-validations with from the most important one feature to all features. I dropped a new created feature, which indicates which temperature is higher. I also dropped *month* due to its high correlation with season. After eliminating these two features a new submission scored 0.39776.

6.4.7 Model selection and parameter tuning

I chose 3 optimum models from several popular models by doing local validations based on their performance at default settings. Using grid search to iteratively tune the parameters, I got big improvements on the leaderboard. The validation RMSLE errors and leaderboard RMSLE errors of every individual model are shown in Table 14.

Table 14: Validation and leaderboard scores

Model	Validation	Leaderboard
GradientBoostingRegressor	0.29426	0.37466
RandomForestRegressor	0.31537	0.39193
ExtraTreesRegressor	0.31441	0.38601

6.4.8 Ensemble

Based on the three outputs from the previous step I have produced an ensemble by weighted averaging. Here I used a weighted sum of all individual outputs. If each individual output is y_i and we have n outputs, then the overall result \tilde{y} can be defined as:

$$\tilde{y}(x; \beta) = \sum_{i=1}^n \beta_i y_i(x)$$

where β is a set of weights.

To find the best ensemble weights I used 10% of the training set as test set and used the “minimize” method provided by SciPy, which seeks minimization of scalar function of one or more attributes, more information can be found in [43]. The weighted ensemble scored 0.37175 on the leaderboard.

6.4.9 Separate models for registered and casual users

Table 15: Validation and leaderboard log-loss errors of different combination of models

Model ([registered]&[count])	Validation	Leaderboard
GBRT & GBRT	0.30230	0.36899
Extra trees & Extra trees	0.31715	0.38196
Random forest & Random forest	0.32037	0.38195
GBRT & extra trees	0.30239	0.36959
GBRT & Random forest	0.30256	0.36832
Extra trees & GBRT	0.31316	0.37745
Extra trees & Random forest	0.31670	0.38005
Random forest & GBRT	0.32001	0.38272
Random forest & Extra trees	0.32101	0.38352
Average of top 4		0.36674
Average of 9		0.37097

Reviewing the data, the two types of riders are not treated separately: casual and registered riders. The data exploration already shows that each group’s behavior differs, and we might be able to improve our performance by modeling each separately.

I repeated the hyperparameter tuning steps and resubmitted the results. The standing increased in the competition by a significant percentage. By submitting an average of the top 4 best predictions I got a score of 0.36674. All validation and leaderboard scores of model combinations are listed in Table 15.

6.4.10 Final result

As of the time of writing, the competition is closed but remains in the validation phase. With a leaderboard score of 0.36674, my submission ranked 36th out of 3,252 teams, while the first place submission scored 0.21545. An overview of the leaderboard is shown in Table 16.

Table 16: Public leaderboard overview of third competition

Ranking	Team Name	Score
1	Team Oliver	0.21545
2	Alliance	0.24976
3	A_Power	0.28820
...
34	mo	0.36607
35	Flyfish	0.36610
36	Ying Dong	0.36674
37	BoilerUp	0.36683
38	Endian Ogino	0.36683
...
3250	acai	4.76189
3251	JeffHansley	4.76189
3252	Muhammad Tahir	4.76189

6.5 Conclusion

In this competition I started with a bad score but saw a large improvement after extracting information from *datetime* feature. Predicting the logarithm of count obtained a much better score. Elimination of the some features helped greatly. After model selection and parameter tuning I reached the top 100 on the leaderboard. Separately building models for registered and casual users was a good choice. Choosing the best parameters for different model combinations required many cross validations and was very time consuming. The final and best submission was the output average of 4 best-combined models. Figure 19 shows how the approaches performed throughout the entire process.

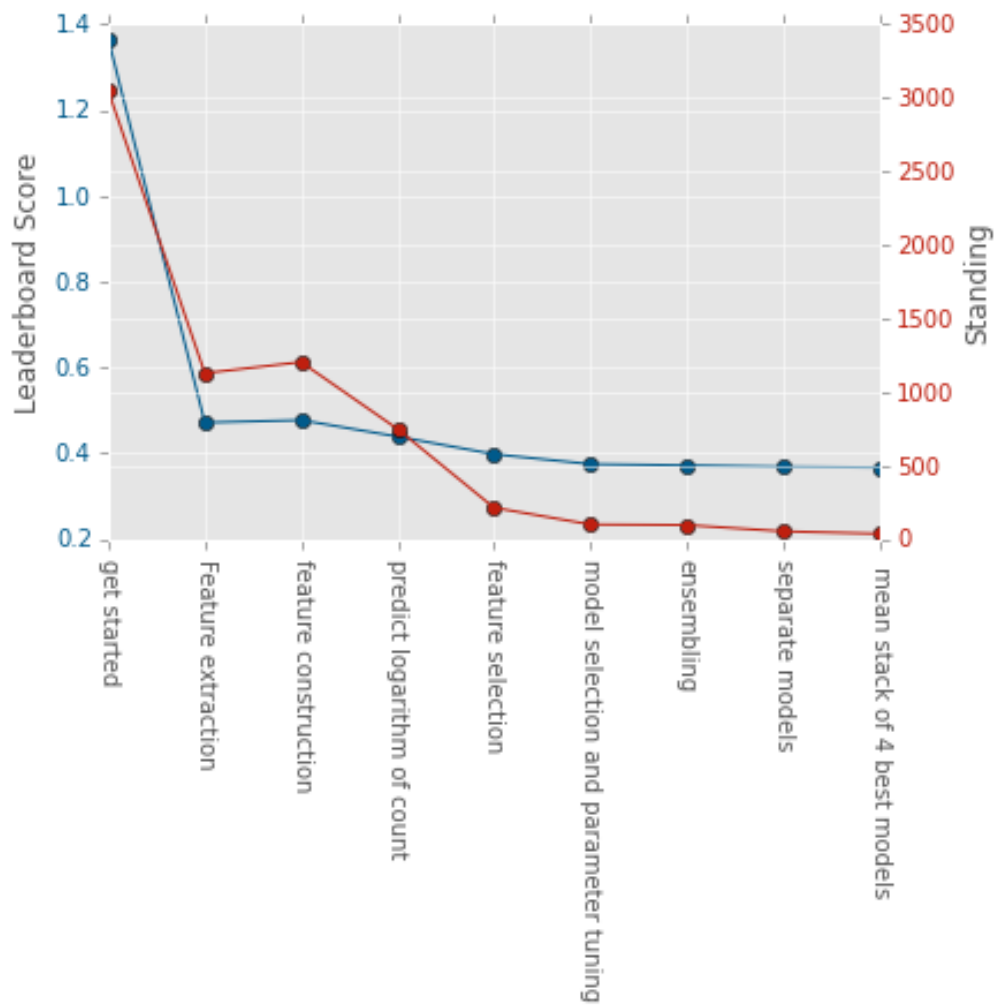


Figure 19: Score vs. Standing for competition Bike Sharing Demand Prediction

6.6 Lessons learned

1. It is important to correctly identify what needs to be predicted.
2. Do not choose only one model based on its performance at the default settings, it may give much better results after parameter tuning.

7 Conclusion

This thesis first presented a pipeline used for approaching a data prediction competition in Chapter 3. This pipeline is composed of data exploration, data pre-processing, feature engineering and selection, model validation and selection, and parameter tuning. Some common and simple methods are outlined for every step in this pipeline.

Chapter 4, Chapter 5 and Chapter 6 describe three competitions on Kaggle. The first one is the Forest Cover Type classification competition. In this competition participants are asked to classify forests into 7 classes using cartographic features. In this competition my final submission ranked 29th out of 1,694 teams. The second competition hosted by Otto Group, asked us to classify their products into nine categories using 93 obfuscated features. In this competition I ranked 347th out of 3,514 teams. The last competition described in Chapter 6 is the Bike Sharing Demand competition. In this competition we were asked to predict the count of bike rentals, given date, time, and weather data. My solution ranked 36th out of 3,252 teams on the leaderboard.

The results of the three competitions are all encouraging. Using the pipeline given in Chapter 3 can provide promising results in a predictive competition, but it requires much more effort and a more complicated solution to win a competition.

8 References

- [1] X. D. Wu, X. Q. Zhu, G. Q. Wu, and W. Ding, "Data Mining with Big Data," *Ieee Transactions on Knowledge and Data Engineering*, vol. 26, pp. 97-107, Jan 2014.
- [2] U. a. P.-S. Fayyad, Gregory and Smyth, Padhraic, "From Data Mining to Knowledge Discovery in Databases," *AI magazine*, vol. 17, pp. 37-54, 1996.
- [3] . *Kaggle Member FAQ*. Available: <http://www.kaggle.com/wiki/KaggleMemberFAQ>
- [4] P. Simon, *Too Big to Ignore: The Business Case for Big Data*: John Wiley & Sons, 2013.
- [5] I. H. a. F. Witten, *Eibe Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann 2005.
- [6] J. K. Han, Micheline, *Data mining: concepts and techniques*: Morgan Kaufmann, 2001.
- [7] L. Breiman, "Random Forest," *Machine learning*, vol. 45, pp. 5-32, 2001.
- [8] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3-42, 2006.
- [9] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics* pp. 1189–1232, 2012.
- [10] C. a. T. Tianqi, He. *XGBoost: eXtreme Gradient Boosting*. Available: github.com/dmlc/xgboost
- [11] C. M. Bishop, *Pattern recognition and machine learning*: Springer, 2006.
- [12] W. Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, pp. 14-23, 2011.
- [13] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81-106, 1986.
- [14] J. a. H. Friedman, Trevor and Tibshirani, Robert, *The elements of statistical learning* vol. 1: Springer series in statistics Springer, Berlin, 2001.
- [15] L. a. C. Breiman, Adele *Random Forests*. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- [16] J. Strickland, *Predictive Analytics using R*: Lulu.com, 2015.
- [17] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*: CRC Press, 2012.
- [18] A. a. K. Natekin, Alois, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, 2013.
- [19] C. Tianqi, "Introduction to Boosted Trees," 2014.
- [20] B. Hamner. *How different is data in Kaggle competitions from real data?* Available: <http://www.quora.com/How-different-is-data-in-Kaggle-competitions-from-real-data/answer/Ben-Hamner>
- [21] D. M. Hawkins, *Identification of outliers*: Springer, 1980.
- [22] K. Potter, Hagen, H., Kerren, A., & Dannenmann, P. , "Methods for Presenting Statistical Information: The Box Plot," *Visualization of Large and Unstructured Data Sets*, vol. 4, pp. 97-106, 2006.
- [23] S. a. K. Kotsiantis, D and Pintelas, PE, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, vol. 1, pp. 111-117, 2006.
- [24] Z. Markov. (2015). *Data preprocessing*. Available: http://www.cs.ccsu.edu/~markov/ccsu_courses/DataMining-3.html
- [25] J. L. a. G. Lustgarten, Vanathi and Grover, Himanshu and Visweswaran, Shyam, "Improving classification performance with discretization on biomedical datasets," *AMIA Annual Symposium Proceedings*, vol. 2008, pp. 445--449, 2008.
- [26] W. P. Krzysztof J. Cios, Roman W. Swiniarski, Lukasz Andrzej Kurgan, *Data Mining: A Knowledge Discovery Approach* Springer, 2007.
- [27] R. Kerber, "Chimerge: Discretization of numeric attributes " presented at the Proceedings of the tenth national conference on Artificial intelligence, 1992.

-
- [28] P. Domingos, "A Few Useful Things to Know about Machine Learning," *Communications of the ACM*, vol. 55, pp. 78-87, 2012.
- [29] H. a. M. Liu, Hiroshi, *Feature selection for knowledge discovery and data mining* Springer Science & Business Media, 1998.
- [30] J. Brownlee. (2014). *An Introduction to Feature Selection*. Available: machinelearningmastery.com/an-introduction-to-feature-selection/
- [31] . *Feature ranking with recursive feature elimination*. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html - [sklearn.feature_selection.RFE](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html)
- [32] J. a. B. Bergstra, Yoshua, "Random Search for Hyper-Parameter Optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281-305, 2012.
- [33] . *Forest Cover Type Prediction*. Available: <http://www.kaggle.com/c/forest-cover-type-prediction>
- [34] K. L. Bache, M. . UCI Machine Learning Repository [Online].
- [35] . *Data Fields for Forest Cover Type Prediction*. Available: <https://http://www.kaggle.com/c/forest-cover-type-prediction/data>
- [36] . *Otto Group Product Classification Challenge*. Available: <https://http://www.kaggle.com/c/otto-group-product-classification-challenge>
- [37] . *Data fields for Otto Group Product Classification Challenge*. Available: <https://http://www.kaggle.com/c/otto-group-product-classification-challenge/data>
- [38] C. a. V. Cortes, Vladimir, "Support-vector networks " *Machine learning*, vol. 20, pp. 273-297, 1995.
- [39] A. a. C. Niculescu-Mizil, Rich, "Predicting good probabilities with supervised learning," presented at the Proceedings of the 22nd international conference on Machine learning, 2005.
- [40] . *XGBoost Parameters*. Available: <http://github.com/dmlc/xgboost/blob/master/doc/parameter.md>
- [41] . *Bike Sharing Demand*. Available: <https://http://www.kaggle.com/c/bike-sharing-demand>
- [42] . *Data Fields for Bike Sharing Demand*. Available: <https://http://www.kaggle.com/c/bike-sharing-demand/data>
- [43] . *scipy.optimize.minimize*. Available: <http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.optimize.minimize.html>