Bot or Human: Identifying Bot-Generated Clicks Using Machine Learning

FILIP BORG, AXEL BROBECK, SAGA KORTESAARI
NERMIN SKENDEROVIC, ARVID SUNDBOM, CHARLES SUNDSTRÖM

Supervisor: Firooz Shahriari

Bachelor's thesis in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Abstract

"I'm not a robot" is a common CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) often shown today upon entering websites. The purpose behind the challenge is to distinguish humans from robots, or *bots*. However, the user experience becomes somewhat intrusive and is not always viable for many websites. This project explores, in collaboration with Prisjakt, how to retroactively identify clicks generated by bots, using historical data and various machine learning models. The models are trained and evaluated on the historical data in an effort to be able to classify future clicks automatically. The result of the project is an implementation of two models, a neural network and a gradient boosting model, as well as an application programming interface (API) to use the models with. The models show very promising results and suggest that an automated system, which identifies clicks generated by bots, is possible.

Sammandrag

“Jag är inte en robot” är en vanlig CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) som ofta visas idag när man besöker hemsidor. Syftet med utmaningen är att skilja människor från robotar eller *bottar*. Användarupplevelsen blir dock något påträngande och är inte alltid genomförbar för många webbplatser. Detta projekt utforskar, i samarbete med Prisjakt, hur man retroaktivt identifierar klick genererade av bottar med hjälp av historisk data och olika maskininlärningsmodeller. Modellerna tränas och utvärderas utifrån den historiska datan i ett försök att automatiskt kunna klassificera framtida klick. Resultatet av projektet är en implementering av två modeller, ett neuralt nätverk och en “gradient boosting”-modell, samt ett applikationsprogrammeringsgränssnitt (API) att använda modellerna med. Modellerna visar mycket lovande resultat och föreslår att ett automatiserat system, som identifierar klick som genereras av bottar, är möjligt.

Contents

1 Introduction **1**
- 1.1 Purpose and goal 1
- 1.2 Prisjakt and Problem definition 2
- 1.3 Limitations 2
  - 1.3.1 Non real-time 2
  - 1.3.2 Limited functionality for IPv6 addresses 3
  - 1.3.3 No human spam detection 3

2 Data Processing **4**
- 2.1 The Raw Data 4
- 2.2 Preprocessing 5
  - 2.2.1 Cleaning 5
  - 2.2.2 Data encoding 6
  - 2.2.3 Normalization 7
  - 2.2.4 Class weights 8
  - 2.2.5 Oversampling the minority class 9
  - 2.2.6 Dimensionality reduction 9
- 2.3 Feature engineering 10

3 Machine Learning Fundamentals **11**
- 3.1 Supervised learning 11
- 3.2 Performance Metrics 12
  - 3.2.1 Accuracy, precision and recall 13
  - 3.2.2 ROC curve and AUC score 14
  - 3.2.3 F1 score 15
- 3.3 Models 16
  - 3.3.1 Logistic Regression 16
  - 3.3.2 Decision Trees 16
  - 3.3.3 Bagging Classifier 17

3.3.4 Gradient Boosting 18
3.3.5 Neural Networks 18

4 Method 20
4.1 Frameworks, Libraries, and Languages 20
4.2 IP Lookups 21
4.3 Analysing Data and Feature Engineering 22
4.4 Evaluating Models 23
4.5 Adapting Models to Change 24

5 Results 25
5.1 Click statistics 25
5.1.1 Time of day 25
5.1.2 Countries 26
5.1.3 Spam distribution 27
5.2 Extracted Features 28
5.2.1 Night Clicks 28
5.2.2 Country of origin 28
5.2.3 Origin-Target Based Features 29
5.3 Manual data cleaning 29
5.4 Model results 30
5.4.1 Logistic Regression 30
5.4.2 Bagging Classifier 31
5.4.3 XGBoost 33
5.4.4 Neural Network 35
5.5 Application Programming Interface 38

6 Discussion 39
6.1 Model Results 39
6.1.1 XGBoost vs. Neural Network 40
6.1.2 Memory Limitations 40
6.2 Validity of the Labels 41
6.3 Features not used 41
6.4 Circumventing the Model 42
6.5 Application Programming Interface 42

7 Conclusion 44
7.1 Applicability of the system 44
7.2 Future work 45
7.3 Social and Ethical Aspects 46

A Feature Importance **54**

Glossary

Accuracy: The fraction of predictions which a classification model performed correctly.

AI: *Artificial Intelligence.* "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages" [1].

API: *Application Programming Interface*, used as an interface from one software to another.

Batch: A set of samples used to perform one iteration when training the model.

Batch size: The number of samples in a batch.

Bias (prediction bias): An error that occurs due to erroneous assumptions in the learning algorithm.

Binary classification: Task that involves classifying elements into two groups.

Boosting: A technique in machine learning which iteratively combines multiple weak classifiers into a single strong classifier by applying weights to misclassified samples.

Bot: "An Internet bot, web robot, robot or simply bot, is a software application that runs automated tasks (scripts) over the Internet. Typically, bots perform tasks that are simple and repetitive, much faster than a person could. The most extensive use of bots is for web crawling, in which an automated script fetches, analyzes, and files information from web servers. More than half of all web traffic is generated by bots." [2]

Classification: A model used for supervised learning where the output will select from a discrete number of classes.

Clustering: An algorithm used for unsupervised learning, the program will find clusters of similarities in the data and then assign the new data to the clusters.

Confusion Matrix: A quadratic table that provides a summary of how well a classification model performed.

Deep learning: A subset of ML which uses multiple layers of learning to extract a hierarchy of information.

Feature: The fields in the data that will be used as input into a model. A model can have many features.

Hyperparameter: A parameter used to control the learning process.

IPv4 and IPv6: Version 4 and 6 of the Internet Protocol. IPv4 uses a 32-bit address space and IPv6 uses a 128-bit address space.

Label: The correct answer or result for a given data point.

ML: *Machine Learning*, a subset of AI which learns from the data it is given in order to become better and better.

Model: The representation of what a machine learning system has learned from the training data [3].

Neural Network: Artificial neural networks are inspired by their biological counterparts. A network consists of several layers of *nodes*, or *neurons*. The value of each node is determined as a *weighted sum* of the nodes on the previous layer, as well as a *bias*. The input to the network is given through the first layer, and the output is then given by the values of the final layer. It is through adjusting the *weights* and *biases* you can get the network to classify the data.

Overfitting: When a model is created that matches the training data too closely, resulting in a model that fails to make correct predictions on new data.

Pandas: An open source data analysis and manipulation tool, commonly used by several machine learning frameworks.

Regression model: A model which output is within a continuous range.

Strong classifier: A high-quality classifier with good performance.

Supervised learning: A type of training method where the model is trained on data with predetermined labels.

TensorFlow: An end-to-end open-source platform for machine learning, developed by Google.

Unsupervised learning: A type of training method where the model is trained on unlabeled data, and the system tries to learn on its own.

Weak classifier: A simple, low-quality classifier that generally has quite low accuracy.

As a complement to this glossary, please see the Google Developers Machine Learning Glossary [3].

1 Introduction

The problem of unwanted traffic in the form of bots and spam can be found in most web applications serving content over the internet, as internet traffic generated by bots account for two-fifths of internet traffic [4]. Bots are used for several purposes, and many bots are created with malicious intentions. Bots can take the form of spam-bots on social media, posting large amounts of dangerous links and messages [5], or even automatically visit web-pages in order to commit click fraud [6]. Regardless of the specific type of bot, the problem of identifying bots in order to filter out and remove the traffic they generate is of interest to many web-service providers.

One of many companies that have to deal with such spam-bots is Prisjakt, (see section 1.2) who collaborated with us on this project by supplying data, providing feedback, and lifting interesting ideas.

1.1 Purpose and goal

The purpose of the project is to develop a system which, by the help of machine learning, is capable of automatically identifying clicks that have been generated with the purpose of spamming or crawling Prisjakt's website.

The project's goal is to create an AI which can classify which clicks in a set are spam. This classification task should be carried out by a machine learning model after it has been trained on data supplied by Prisjakt. The resulting model should be able to seamlessly replace the current filtering of spam clicks at Prisjakt without drastically changing the statistics of how many clicks are currently being flagged as spam. In doing so, one major goal is to make the model perform well according to a set of metrics that are deemed relevant for this classification task. This will be achieved by evaluating and comparing different machine learning models and approaches, according to the selected metrics. The final product should be a machine learning model together with a corresponding API which can be integrated into the existing systems used at Prisjakt.

1.2 Prisjakt and Problem definition

Prisjakt is an online price-comparison service that allows users to browse and compare the different prices offered by several vendors for a given product [7]. For each store, the price along with a link to the store is presented. With this price-comparison functionality, Prisjakt increases the visibility of the connected stores. As compensation for the potential customers, the stores pay Prisjakt a fee for each user that visits the store through the links on Prisjakt's website.

In this way, if a bot or spammer generates an unnatural amount of traffic, the stores' invoice can increase without any actual increase in sales or customer exposure. This is problematic, as paying a bot to visit one's e-commerce site without making any purchases is a rather poor business model. Currently, Prisjakt is addressing this issue by manually removing spam clicks before sending each month's invoices, a task which takes a large amount of time and labour, usually about one workday a month for two employees according to estimations from Prisjakt employees. If this spam-removal process was automated, the employees could instead focus on more productive tasks.

1.3 Limitations

This section describes the limitations made to the temporal scope of the project as well as the limitations due to the available data.

1.3.1 Non real-time

The classification of clicks will not be carried out in real-time. Instead, all click data will be recorded and the data set will be periodically fed to the machine learning model. This is due to the fact that it is possible to extract useful features, such as the number of clicks originating from a specific IP address, from compound data through Feature Engineering [8]. Most such compound data can only be generated by viewing clicks that have occurred over some time span, e.g. the clicks originating from some IP address during a certain time span. Utilizing such compound data should result in a more accurate model than one which tries to classify clicks in real-time.

1.3.2 Limited functionality for IPv6 addresses

In the logging process at Prisjakt IPv6 addresses are mapped to a reserved range of IPv4 addresses. This limits the ability of the models to make informed decisions for these addresses since interesting information is lost in the mapping. Of all clicks in the data set, 7.5% originate from addresses from the IPv6 mapped range. If Prisjakt were to change their logging so that the IPv6 addresses were kept, then the project could be extended to work with these just as well, with only minor adjustments.

1.3.3 No human spam detection

Spam clicks originating from human beings will not be considered as a possible class in the classification task. The reasoning behind this is simple: it is difficult to distinguish a spam-click made by an actual human user from a "normal" click, since the characteristics of the clicks, such as click-frequency, are similar. However, these kinds of spam clicks are common, for example, stores inspect competitor prices on a frequent basis (behaving like a bot) in order to stay competitive. Therefore, a possible extension to this work is the classification of clicks generated by spam-bots, human spammers, and regular users.

2

Data Processing

There is an old saying in computer science that goes “garbage in, garbage out”. In the context of AI, this means that a system’s capability of learning is dependant on the quality of the data the system is fed. Feed the model bad data - get bad results, improve the data - improve the results [9]. As such, improving the quality and usefulness of the data is a key aspect of producing a model which performs well. Therefore, the majority of the time and effort put into building a machine learning model goes into data exploration and data wrangling [8]. This is done in order to better understand the data the model will be based on, as well as for extracting useful features which can improve the performance of the machine learning model.

Beyond the extraction of additional features, the data usually undergoes a more general tidying process before being fed to the model. This process usually includes several stages, such as cleaning, encoding, normalization and dimensionality reduction, and is generally referred to as preprocessing [10].

2.1 The Raw Data

The data used throughout the project contains information logged by Prisjakt when a user clicks on a product. Clicks originate from both human beings and bots. Each time a click is made, data about the click is gathered. The click-data consists of several fields, shown in Table 2.1.

As seen in the table, the clicks in the data are labeled as spam or not spam in the field ‘spam_removed_id’. This comes from Prisjakt employees that have manually labeled the clicks, determining which clicks have been generated with the purpose of being spam. Since the data is labeled, supervised models are well-suited for the task of classifying clicks as spam or not spam, see section 3.1.

Table 2.1: Description of data fields for a given click

Field	**Description**
click_id	Click ID
tidpunkt	Timestamp
from_ip_int	Integer representing of IPv4 address
store_name	Store name
ftgid	Store ID
product_name	Product name
product_id	Product ID
pris_id	Price ID
category_name	Category name
category_id	Category ID
top_category_name	Parent category name
top_category_id	Parent category ID
spam_removed_id	0 if not spam, a number > 0 otherwise

2.2 Preprocessing

Data preprocessing is the act of transforming the data so that it better suits the particular needs of the model to be used. There are several well-known data preprocessing techniques, this section will highlight the most common.

2.2.1 Cleaning

Real-world data is often *dirty* [11]. One common representation of such dirt is referred to as *errors*, which represent information within the data which has been lost during data gathering [8]. An example of this is missing values in the columns of data entries, such as missing timestamps for when some data points were generated. It is usually not possible to correct such errors without gathering new data. However, it is usually possible to handle missing values gracefully without needing to gather new data.

One approach is simply to remove the data points with missing values and to use the rest of the data as planned [10]. This is a simple, straightforward approach which does not require a high amount of effort. If the number of missing values is quite low compared to the total number of data points, the removal of a few entries in the data should not have a noticeable effect on the model. However, if the data points with missing values represent a large portion of the available data, this might not be a viable approach.

2.2.2 Data encoding

Machine learning models are mathematical models. As such, they can only work with numerical data as input [8]. However, it is possible for data sets to include features of type *string*, *character* or *boolean* value. Therefore, some way of converting such values into numerical values is needed. One technique for doing so is *Label encoding*, also sometimes called integer encoding. Label encoding is the act of encoding non-numerical data with numerical representations [12]. Even though it is called Label encoding, this technique can be used to encode the features of each sample, not just the target labels. This is done by identifying the distinct values that appear in features which are to be encoded, and then assigning a numerical value for each distinct non-numerical value found. Additionally, depending on the implementation, label encoding can also be applied to already existing numeric data, in order to simplify and shrink it.

As an addition to encoding features, the labels representing the classes also need to be encoded. This is due to the fact that machine learning models, being mathematical models, can only produce numerical values as output [8]. The label encoding can be carried out in the same way as when encoding features, with each unique value being assigned a distinct numerical value. As such, the numerical value output by the model can be correlated to a specific class. For instance, in the case of a binary classification task, the majority class is usually assigned the value 0, and the minority class assigned the value 1. As such, the output of the model can be interpreted as a classification in one of two distinct classes.

One way to encode data other than using label encoding is to use what is called one-hot encoding [10]. Instead of assigning values of categorical features with numbers, one-hot encoding utilizes one-dimensional arrays of binary values to encode categorical data. The size of these arrays is equal to the number of existing categories in the data which is to be encoded. Which category each sample belongs to is represented by the position of the single positive binary value in this array, typically represented by a 1, with all other values being negative and set to 0.

Which encoding method should be used is not always obvious. The main difference between label encoding and one-hot encoding is the ordinal relationship between the encoded values. If label encoding is used, samples encoded with values 2 and 3 can be seen as more similar than values encoded with values 2 and 10 [13]. If this is a significant relationship between samples that should be preserved, label encoding is generally the preferred approach. However, if no such relationship exists in the data, one-hot encoding should be used. If label encoding is used and an ordinal relationship with it, the model could be misled when trained on the data.

2.2.3 Normalization

If the input data contains features with values varying in magnitude, the machine learning model working with this data might become biased towards certain values. In general, features with large ranges of values will affect the model a lot more than the features with small value ranges. Exactly how much different features affect the model depends on the type of model used. Models which internally use distance measurements are affected more than models which use other measurement types [14]. When normalizing data, each feature is scaled individually, without consideration of the values in other features.

There are different types of normalization techniques, one of them is *min-max scaling*. Min-max scaling scales the data-points to be within the range [0, 1] [9]. The technique is implemented using the following formula:

$$x := \frac{x - \min(x)}{\max(x) - \min(x)}. \tag{2.1}$$

x represents a value in a feature column, whereas $\max(x)$ *and* $\min(x)$ *represents the maximum and minimum value in that column.*

Another common type of normalization is known as *standardization*. Standardization follows the following formula:

$$x := \frac{x - \bar{x}}{\sigma}. \tag{2.2}$$

$\bar{x}$ represents the mean of all values in the given feature. σ represents the standard deviation of all values in the feature.

As shown by the formula, the mean value of the feature in question is subtracted from each value, and the difference is then divided by the standard deviation of the feature [15]. Standardizing a feature results in the values of the feature having a zero-mean and a unit-variance. As such, the values of the feature will follow a normal distribution quite closely.

Min-max scaling and standardization are two of the most common normalization techniques. However, which normalization technique should be used is not always clear, and can require a bit of experimentation [16]. The characteristics of the data and which machine learning model is used affects which normalization approach produces the best results. However, normalizing the data does not decrease the performance of the model and is therefore generally an advisable step in the data processing.

2.2.4 Class weights

One common challenge when developing machine learning models for classification tasks is an imbalance between the labels in the training data [17]. One way to manage the performance penalty caused by imbalanced data sets is to use class weights. As hinted by the name, when using class weights, each known class in the data set is assigned a weight. These weights affect the penalization of the model when a data point is misclassified. The higher the weight, the greater the penalty. In this way, the model is more sensitive to change that leads to greater success in classifying entries that belong to classes that have been assigned high weights [18].

The use of class weights can be illustrated with a simple example: If the data set consists of 100 samples belonging to class 1 and 1000 samples belonging to class 2, the data set is rather imbalanced. In order to make up for this imbalance, the classes can be assigned weights as follows:

$$\text{class\_weights} = \{\text{class\_1} : 10,\ \text{class\_2} : 1\}.$$

As a result of this, during training, the model would be penalized 10 times more when misclassifying samples belonging to class 1, compared to misclassifying samples belonging to class 2.

But which weight should be assigned to each class? As a rule of thumb, as the number of entries belonging to a given class decreases, the weight associated with that class increases, and vice versa [19]. One common approach is to assign weights to classes corresponding to the inverse of how common samples belonging to each class are. The above example showed a 1:10 ratio between class 1 and class 2 and thus class 1 was assigned the weight 10 and class 2 the weight 1. This seems to work especially well for binary data sets, where only two classes exist. When dealing with more complex data, where a higher number of classes are apparent, further experimentation might be required.

However, this is outside the scope of the project, and won't be addressed here.

2.2.5 Oversampling the minority class

Another common technique for dealing with imbalanced data sets is to *oversample* the minority class. Oversampling consists of replicating existing samples of the minority class in order to balance out the class distribution in the data set [20]. The most basic approach to oversampling is to simply copy existing entries of the minority class into the data so that several rows have exactly the same values in each column. This can then be repeated until the classes have a roughly equal number of samples. However, since the data points added in this way are exactly the same as entries which already existed in the data, the model is not exposed to any truly new samples. As such, this approach might lead to the model *overfitting* to these few entries in the minority class. This can result in the model improving its predictive performance for the specific samples it is trained on, but not its ability to generalize.

2.2.6 Dimensionality reduction

Dimensionality reduction is the process of transforming high-dimensional data into a lower dimensional representation, while retaining the most meaningful properties of the original data [21]. In the context of machine learning, this is usually done by either removing features entirely, or by modifying the feature columns to simpler, reduced versions. There are multiple reasons for why dimensionality reduction can be advantageous. Firstly, removing surplus features reduces the amount of data the model has to process, which in turn reduces the amount of time it takes to train the model. If the removed features are not meaningful, the predictive performance of the model will be mostly unaffected.

Secondly, reducing the number of features that are exposed to the model can reduce the risk of overfitting. This is especially true if the data contains a large number of features, which can cause it to suffer from "The Curse of Dimensionality" [22]. This "curse" occurs because the density of the samples in the data decreases at an exponential rate as the number of dimensions increase. As a result of this, the model can easily find what is seen as a perfect solution for the exact data it is shown. However, such perfect solutions usually perform poorly in the general case. With this in mind, reducing the number of dimensions in the data makes it more difficult for the model to adjust to such a perfect solution, improving the model's performance for data points not shown during training.

Finally, reducing the number of features makes it easier for the model to find a good solution, since unnecessary features without evident correlation to the problem are removed [10]. In rather crude terms, this allows the model to "focus" on the most important aspects of the data, discarding the parts that are irrelevant.

There exists a range of different methods and techniques which can be used in order to reduce the dimensionality of the data. These techniques range from sophisticated data transformation techniques, to just removing any columns deemed as unnecessary. It is not always easy to determine which features are important. However, it is possible to train the model on all features and then query the model for the importance of each feature after training, see appendix A.

2.3 Feature engineering

Feature engineering is the process of constructing new ways to represent data from available data in order to increase the effectiveness and performance of the model [23]. In the context of machine learning, these new representations of data take the form of additional features.

These features have to be created with quite a bit of thoughtfulness if they are to be useful. As mentioned in section 2.2.6, utilizing too many features can worsen the performance of the model. As such, it is desirable to extract only features which are meaningful in solving the problem at hand. For example, in classification tasks, such features help the model determine which class each sample belongs to. If a given feature is meaningful or not, and therefore worth extracting, is not always obvious. This problem coincides with the same issues as during dimensionality reduction, see section 2.2.6. In short, one could say that feature engineering is working towards getting the most out of the available data by exploring new aspects of it.

3

Machine Learning Fundamentals

This section covers some basic theory, terminology, and different types of machine learning models and some of the ways they differ from each other. Furthermore, some metrics which can be used to evaluate the performance of models are introduced.

3.1 Supervised learning

Supervised learning is a type of learning where the model is fed labeled data during training [10], [24]. Since the data is labeled, what class each sample in the training data belongs to is known. In supervised learning, these labels are used in order to validate the predictions produced by the model during training. After each training-prediction, the model receives feedback about whether the prediction it made was correct or not, allowing the model to adjust itself accordingly. In this way, the model is able to approximate a function to the training data which maps the given input data (the features) to the desired output data (the label). Once trained, the model will apply this function to unlabeled input data in order to produce labels [25].

From this description it should be quite clear why models trained using supervised learning are well suited for classification problems. In the case of binary classification, the prediction can represent the likelihood that a given sample belongs to the minority class. In such a case, the numeric value produced as output will lie in the range (0, 1) [13]. Each such value is then mathematically rounded to produce output values that are either 0 or 1, clearly indicating which class the sample belongs to according to the model.

3.2 Performance Metrics

In order to compare models, different metrics describing important aspects of model performance are used. In the case of a binary classification task, there are four possible outcomes that can occur. These outcomes are used as the basis for the metrics used to measure the performance of binary classification models. The four outcomes are:

- *True Positive* (TP)
  The sample belongs to the positive class and the classifier correctly labelled it as such.

- *True Negative* (TN)
  The sample belongs to the negative class and the classifier correctly labelled it as such.

- *False Positive* (FP)
  The sample belongs to the negative class, but the classifier incorrectly labelled it as positive.

- *False Negative* (FN)
  The sample belongs to the positive class, but the classifier incorrectly labelled it as negative.

Usually, the minority class is seen as positive, and the majority class as negative [10]. When a classifier is tested on a labelled test set, each of these occurrences will be counted and commonly displayed in a *confusion matrix* see Fig. 3.1.

		Predicted class	
		Negative	Positive
Actual class	Negative	TN	FP
	Positive	FN	TP

Figure 3.1: Confusion matrix.

In the case of spam filtering, these classes are usually referred to as *spam* or *ham* (not spam). With the occurrences of the outcomes counted, other more specialized metrics, which describe the performance of the model in a more meaningful way, can be derived. Three of the most common metrics used for this purpose are accuracy, precision, and recall.

3.2.1 Accuracy, precision and recall

Accuracy is the proportion of correctly labeled predictions made by the model [10]. Accuracy is calculated using the following formula:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}.$$

This metric can be misleading for skewed data sets. If the predictor for such a data set always predicts the majority class, the accuracy will seemingly be quite high. However, this does not mean that the model is performing well, as all of the samples belonging to the minority class will have been mislabeled by the model [8]. For this reason, accuracy alone generally does not suffice as a performance metric.

In order to discover deficiencies such as in the case above, the metrics *precision* and *recall* can be used together [10] [8]. Precision describes the accuracy of only the positive predictions and is calculated as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Precision can be described as a measurement of how exact the positive predictions are, where the precision value is reduced for each sample incorrectly labeled as belonging to the positive class. Recall is calculated as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

and describes the proportion of correctly labeled positive instances [10]. Less formally, recall describes what percentage of samples belonging to the positive class were marked as such by the model.

Precision and recall are related such that when adjusting the model in order to increase its precision, the recall will usually decrease, and vice versa. This relationship sets a bound for how good a combination of precision and recall can be and forces compromise in the model design. To help aid the selection of good precision and recall pairs, one can plot these against each other in a precision-recall curve. This gives a visualization of the trade-off between the two. Beyond this, there is another type of graph called an *ROC* curve as well as two additional metrics, called *F1-score* and *AUC*, which can further aid the trade-off between precision and recall [10], [26]. These will be explained in the following sections.

3.2.2 ROC curve and AUC score

The *receiver operating characteristic curve*, commonly referred to as the *ROC* curve, serves a purpose similar to the precision/recall curve. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) [26], [10]. The true positive rate is synonymous with recall and therefore defined using the same formula.

The FPR is defined by the formula

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

The FPR is a measurement of how many samples belonging to the negative class that are incorrectly labeled as positive by the model.

It is usually not as easy to interpret the ROC curve as it is to interpret the precision-recall curve. The ROC curve for a truly random classifier is a perfectly straight line between the lower left corner and upper right corner of the graph (see Fig. 3.2) [10]. As the performance of the model increases, the ROC curve will become skewed towards the upper left corner of the graph. Theoretically, the best possible ROC curve would be composed of straight lines, one from the lower left corner to the upper left, and another from the upper left to the upper right. A graphical explanation of how ROC curves change with model performance is given in Fig. 3.2.

Although visual representations of model performance can provide an overview of a model's performance, numeric values are often more useful when it comes to comparing the performance of several models. This is especially true for comparing models whose performance differs only slightly. One such numerical measurement which is related to the ROC curve is the Area Under the ROC Curve, usually referred to as *AUC* [26], [10]. The AUC is a measure of the total area below the ROC curve, with values in the interval [0, 1] where 1 represents the best performance and 0 the worst. Any model with an AUC value below 0.5 performs worse than just randomly guessing. This is in line with how the ROC curve changes along with the model performance, as curves that are higher in the graph will have larger areas beneath them.

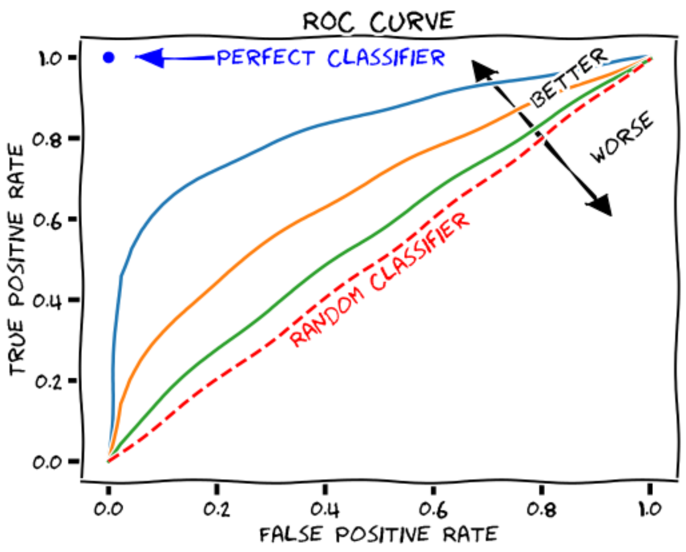

Figure 3.2: Explanation of an ROC curve [27].

The AUC metric can be described as the likelihood of a model assigning a higher probability of belonging to the positive class to a sample that belongs to the positive class, compared to one which belongs to the negative class [26]. This makes intuitive sense, as the model should be more confident in labeling a sample as positive if it actually is positive, rather than negative. Using AUC as a performance metric is usually a good idea if the ROC curve is to be used, as the AUC works as a numerical complement to the graphical representation of the ROC curve.

3.2.3 F1 score

Another numerical metric related to the precision and recall of a model is the so-called *F1 score* or *F-Score* [28], [13], [10]. The F1 score is defined as the *harmonic mean* of the precision and recall of a model [29]. The formula for calculating the F1 score given the model's recall and precision is:

$$2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

The F1 score for a model will contain values in the interval [0, 1], where the worst value is 0 and the best is 1 [28]. One way to interpret the F1 score is as the weighted mean of a model's precision and recall, where low values are given a higher weight [10]. Because of this, models that have similar values of precision and recall will usually have rather high F1 scores. This is due to the fact that such models generally avoid extremely low values for both precision and recall, which can occur when the value of one metric is much greater than the other [8]. As a result of this, in order for a model to achieve a high F1 score, both its recall and precision must have high values. In general, if recall and precision are useful metrics for the task at hand, the F1 score will also be useful.

3.3 Models

This section will provide a brief background for the most notable models used during the project. Short descriptions of each model are given while keeping detailed mathematical descriptions and notation to a minimum.

3.3.1 Logistic Regression

In order to produce classifications, logistic regression-models compute a weighted sum of all features fed to the model, and then outputs the value of the logistic function of this sum [10], [8]. The logistic function can be written in the form:

$$\sigma(t) = \frac{1}{1+e^{-t}}.$$

Logistic regression is well-suited for binary classification because the logistic function is a sigmoid function that outputs values in the range (0, 1). For each sample it is fed, the model produces a value in this range. The value is then rounded, and the sample fed to the model in order to produce the value is classified as positive if the rounding results in the value 1. Otherwise, the sample is classified as belonging to the negative class.

Sigmoid functions are quite common in machine learning due to the fact that they map values from a large range into a smaller range [30]. In a less formal formulation, sigmoid functions “squish” values into a narrower range of numbers [31].

3.3.2 Decision Trees

Decision trees are machine learning models that are capable of solving both classification and regression tasks [10], and are therefore sometimes called Classification and Regression Trees (CART) [8]. A decision tree is made up of a number of nodes. At each node, the tree branches into two or more new nodes, eventually reaching a leaf where the sample fed to the tree is

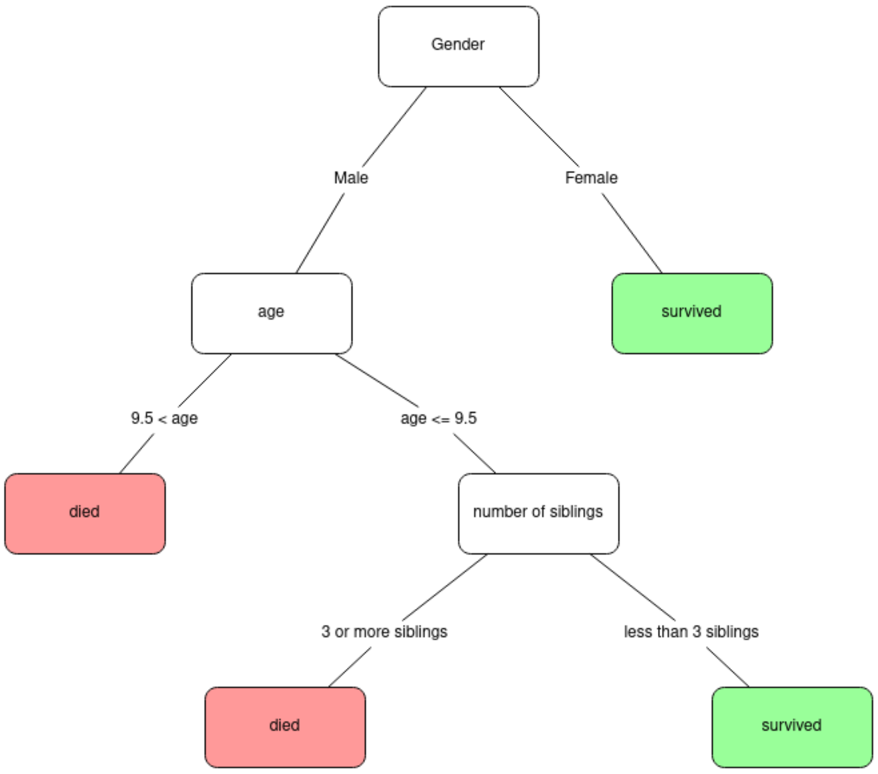

Figure 3.3: Decision tree for the fate of passengers of the Titanic [32].

labeled. These labels may be numerical values in regression tasks, or classes in classification tasks.

As a sample is fed to the model, some feature of the sample is examined and compared to a value in the tree and which branch in the tree to continue along is decided by the outcome of this comparison [8]. As different branches are chosen for the samples, different aspects of the data may be explored as the outcomes of these comparisons differ. A graphical representation of a decision tree is shown in Fig. 3.3.

3.3.3 Bagging Classifier

A bagging classifier is an instance of what is called *ensemble meta-estimators* [33]. These types of models utilize groups of other, so-called *base classifiers*, in order to classify samples. In most cases, the type of base classifier used is some form of decision tree [34].

Bagging classifiers are trained on data by training each base classifier on random subsets of the entire data set [33], [10]. This data is selected *with* replacement. This fact is represented in the name of the model, as bagging is an abbreviation of *Bootstrap Aggregating*. The predictions made by the base classifiers are aggregated by either averaging their values or by using voting. This is the case both during and after training. In general, bagging classifiers usually produce better results compared to individual base classifiers [10]. Therefore, if a decision tree is a suitable model for the task at hand, bagging classifiers may produce even better results.

3.3.4 Gradient Boosting

Similar to bagging classifiers, gradient boosting models are also ensemble classifiers [35]. Furthermore, gradient boosting models also usually rely on decision trees as base classifiers. However, gradient boosting models introduce the additional mechanic of *boosting* [8], which is an iterative technique combining multiple weak classifiers into a single strong classifier by applying weights to misclassified samples.

The motivation behind boosting is that the performance of the model can be improved by assigning different weights to the different samples fed to the model [8]. These weights are based on how successful the model has been in classifying samples belonging to each class. The more difficult it is to correctly classify samples belonging to a class, the higher the weight associated with samples of that class.

Specific to *gradient* boosting is the specific way that new base predictors are added. Each additional base predictor added to the model is fit specifically to the *residual errors* of the previous predictor [10]. In this way, each added predictor attempts to address the largest flaws in the model, effectively minimizing them.

3.3.5 Neural Networks

Artificial neural networks (ANNs), or just *neural networks* (NNs), are based on computational units called nodes or *neurons* [8]. These neurons are organized in layers, of which there are three kinds: input, output, and hidden [36]. The input layer is made up of n neurons, where n is the number of features in the input data. The shape of the output layer can vary depending on the task. For example, in a multi-class classification problem, the output layer could contain one neuron for each existing class, where the value of each neuron represents how likely a sample is to belong to that class. However, in a binary classification problem, the output layer usually consists of a single neuron, the value of which represents the probability of a sample belonging to the positive class.

NNs can have one or more hidden layers. Networks which consist of more than one hidden layer are called *deep* neural networks [37].

In a neural network, each neuron receives a numeric input which it applies a simple function to [13]. Therefore, one abstract way to look at a NN is as a series of data transformations that eventually produce results in the format specified by the output layer. For neurons that do not belong to the output layer, a weight is applied to the output of this function, and the resulting values are passed on as input to the nodes in the next layer of the network. As mentioned above, the output layer produces numerical values whose interpretation depends on the specific application of the neural net.

4 Method

This section outlines the methods used during the course of the project, as well as the underlying choices behind them. The tools used to develop the model and the system as a whole are presented. The main parts of a machine learning project are also presented and discussed in the context of this specific project.

4.1 Frameworks, Libraries, and Languages

Python [38] has been the main programming language used to carry out the project, and has been used both when developing the machine learning models and the API for accessing them. This choice is mainly motivated by the fact that Python is one of the most popular languages within the field of machine learning and therefore has a large selection of well-tested frameworks and libraries to support developers in this field [39].

One of the most important parts of developing a machine learning model is which data is selected as well as how that data is used [40]. There are a number of tools that have been developed with the purpose of making the task of managing large amounts of data easier. In this project, Pandas [41] has been used for loading, managing and pre-processing the data used to train and evaluate the models. The main rationale behind using Pandas was that it is quite user-friendly for users with previous knowledge of Python or programming in general. This is mainly because it is simply an additional library imported into the program and because it has a well-documented API.

Beyond this, other Python frameworks such as Matplotlib [42] and Numpy [43] have been used in order to support the development of a useful machine learning model. Matplotlib offers functionality for plotting data in numerous useful ways. This can be useful in a machine learning project, as visual plots can greatly improve one's understanding of the data at hand. The library Matplotlib in particular was chosen simply because it is the most popular library which offers this functionality. Numpy enables users to transform data stored in Python-objects into a more optimized, C-like data representation. Generally, this leads to increased computational performance and reduced memory usage, which in turn leads to being able to train the models faster and on a larger quantity of samples. Similar to Matplotlib, Numpy was chosen simply because it is the most popular library offering this kind of functionality.

The Pickle [44] library was used in order to serialize and de-serialize models in-between training sessions. This was done in order to circumvent the memory limitations of the available systems and train models on larger amounts of data.

The framework Flask [45] has been used to develop the API for interacting with the models. Flask was selected because of its small size, which does not force developers to include a lot of boilerplate code, and because it allows easy integration with the machine learning side of things.

For performing efficient IP lookups, the extension modules pyasn [46] and py-radix [47] were used. Pyasn implements fast historical lookups from an archive. The archives used were downloaded from routeviews.org [48]. For those IP-addresses not found in the archive, a TCP-based protocol called Whois [49], [50] was used. Whois is a protocol used for querying databases about information related to some Internet resource, such as a domain name or IP address. Py-radix implements radix trees, which are efficient for storing the new whois-lookups. After the Autonomous System Number (ASN) had been found, tables from ipinfo.io were used to find the hostname and origin country [51].

4.2 IP Lookups

Sometimes a bot uses multiple IP addresses and are therefore harder to detect. Usually though, the IP addresses are linked through one of the following columns:

- The Autonomous System Number (ASN).

- The host of the address, also known as the Internet Service Provider (ISP) for IP addresses.
- The range of the IP subnet the address belongs to.
- The country the IP address originates from.

This information was extracted from each IP address in the data set with the tools described in section 4.1. The four data categories mentioned above were added as new columns. If, for example, it is common for spam clicks to originate from a certain host, this might not be represented if only the IP addresses are fed to the model. However, by including these four features in the data, broader relationships between IP addresses and spam-tendencies can be represented.

4.3 Analysing Data and Feature Engineering

In the early stages of the project, it became apparent that using only the raw data from Prisjakt would not work well for training a model. Therefore a significant part of the project was designing new features that captured the difference in behavior between spam-bots and humans. This process is also known as *feature engineering*, see section 2.3. The first step was to find something measurable in the input data that could be relevant for distinguishing bots from regular users. For example, a bot would perhaps click more, and target a specific store, see section 5.2.3. Another approach was considering what an employee at Prisjakt would look for when manually clearing spam, perhaps which stores had an unusual increase in traffic. Lastly, an attempt to find statistical differences between spam and regular clicks was made. This was done by calculating and graphing statistical values for both sets. If any significant difference was found between values of samples marked as spam and regular clicks, then that was made into a new feature, see section 5.1.

After finding an interesting feature, it had to be represented as a number that could be fed to a model. A majority of features were already represented by numerical values, but if necessary, encoding was carried out using the methods discussed in section 2.2.2. Some features, which represented statistics of the data over large periods of time, required a large amount of computational effort to produce. These computations had to be carried out efficiently in order to be completed in a reasonable amount of time. In order to speed up this process, the Python extension Cython [52] was used. This allowed easy integration of compiled code with significantly faster run-time performance than regular Python code, thereby reducing the required time for calculating some features. The features that ended up being used are described in more detail in section 5.2. After a new feature had been designed it was then tested with a model to evaluate if adding it resulted in any improvements. The features that were found to be the most helpful with the least amount of overfitting were then put in a pipeline to be extracted from the raw data in one go.

4.4 Evaluating Models

Testing and evaluating the different machine learning models has been crucial. The primary way of evaluating the models has been comparing how well they perform when making predictions on previously unseen data, using the performance metrics discussed in section 3.2. First data from the same time as the clicks that the model was trained on, and if that showed promising results, clicks from later times were tested on as well. If the only thing compared is the model's performance during training, models which suffer from overfitting would seem like they perform best, even though they might perform worse when predictions are made on new data. Because of this, the data will be divided into two parts - training and test data.

Comparing models in this way allows the comparison of different types of models, as well as how the performance of a model changes as the values of its hyperparameters, the parameters used to control the learning process, are adjusted. Comparing models is generally quite straightforward, and it is generally possible to predict roughly how well suited a model will be for a certain task if the internal functionality of the model is understood. As a result, some models can be more or less ruled out depending on the nature of the task at hand.

Comparing how changes to a model's hyperparameters affect the performance can be more intricate. The number of available hyperparameters can vary greatly between models, and the extent to which the hyperparameters affect the model can also differ substantially. This means that finding the optimal values for the parameters usually requires quite a bit of time and effort, but the resulting increase in performance may well be worth it. There exist some tools which can aid in this. For example, the GridSearchCV module [53] supplied by scikit-learn, performs an exhaustive search over specified alternatives of hyperparameter values for a given model. Which values should be included in the search is specified by the user. By using GridSearchCV, a large part of hyperparameter optimization can be automated. Although some manual exploration of values may still be necessary to identify reasonable values to include in the search.

4.5 Adapting Models to Change

The predictions made by a model are dependent on that the model is "familiar" with the characteristics of the data provided. For example, such characteristics could be the range or magnitude of values, which values are common in the data, and if any values are exceedingly rare. If such characteristics were to significantly change, the model's predictive performance might worsen. For example, if a model for classifying spam is trained on a large amount of data representing new, current clicks, that model should perform reasonably well when making predictions on current data. However, if the behavior of the spam-generating bots changes profoundly, this could result in such a change in the characteristics of the data that the model is no longer able to successfully classify which clicks are spam. Since it is not possible to prevent the bots from changing their behavior, the models have to be able to adapt in order to recognize the new characteristics of spam clicks. The primary way to achieve this is to re-train the model using new data which includes these characteristics.

However, such re-training might not be a trivial task. If the behavior of the spam-generating bots changes profoundly, the model might fail in identifying the generated clicks as spam. Because of this, the labels needed to train the model would be incorrectly represented in the data, i.e. clicks that really were spam would not be marked as such. Therefore, if the model were to be trained on the data which had been generated by itself, wrong labels would be used as feedback to the model during training, resulting in a downward spiral and continuously worsening performance. To cope with such a situation, manual labeling could be performed until enough data has been gathered to re-train the model. If the behavior of spam-bots were to change often, the machine learning model could be rendered useless. However, this is probably not very likely as spam-generating bots are not infinitely complex and therefore have a limited range of behavior, meaning the amount of manual labeling required should also be limited.

5

Results

This section highlights the outcomes and findings of the project. Insightful statistics into the click-data are presented, in addition to the features used and the feature engineering behind them. An in-depth analysis of the model results and performance is presented as well.

5.1 Click statistics

In order to successfully use the data and generate new, effective features, the data had to be explored a bit. This mainly meant to investigate whether or not there were any interesting statistics that could be utilized when extracting new features from the data. Beyond this, these statistics could also provide insights into the data that made solving the task easier, even if they necessarily couldn't be fed to the model.

5.1.1 Time of day

Figure 5.1 shows plots of the distribution of clicks over the hours in a day. The left plot represents regular, non-spam activity, and the right plot represents clicks marked as spam. The timezone used was Central European Time (CET) and Central European Summer Time (CEST) for clicks that occurred during the summer.

There is a clear difference between the amount of non-spam and spam clicks generated during the night, most prominently between around 00:00 and 06:00. During this time span, the amount of non-spam activity severely decreases, but the amount of spam being generated stays consistent. This might be because bots don't need to sleep as humans do, or because spam may be generated in different time zones than CET/CEST. Whatever the

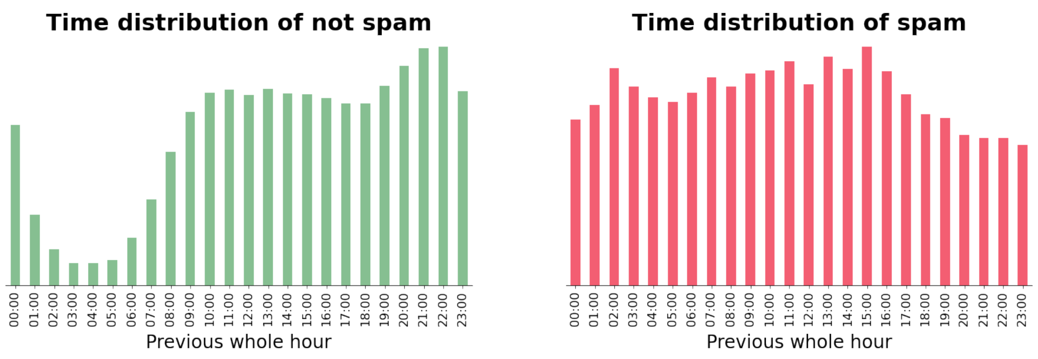

Figure 5.1: Time distribution comparison for spam and non-spam.

reason may be, the difference between the click distributions was significant enough for the time of day to be used as a feature to be fed to the model. The way this was represented in the data is discussed further in section 5.2.1.

5.1.2 Countries

As the data supplied by Prisjakt was generated by clicks on their Swedish website, a majority of clicks originated from Swedish IP addresses. However, this was not the case for clicks marked as spam. Fig. 5.2 shows two pie charts. The left one represents the clicks marked as non-spam and the right one represents clicks marked as spam. Each chart is divided into three sections, one for Swedish clicks, one for clicks that originate from Scandinavian countries other than Sweden, and one for clicks that originate from non-Scandinavian countries.

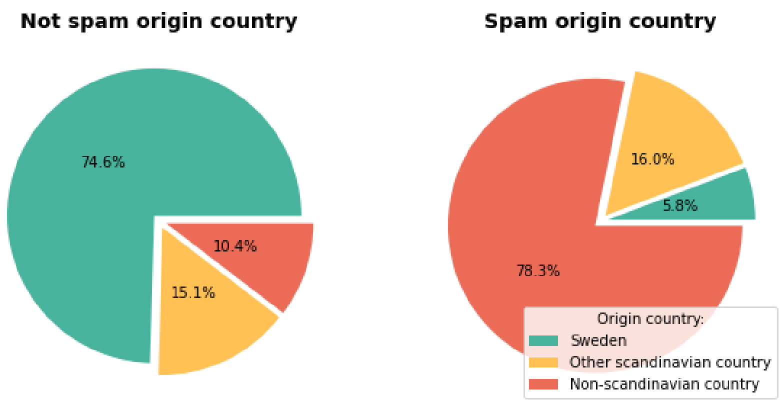

Figure 5.2: Origin country comparison for spam and non-spam.

As can be seen from the figure, there is a large discrepancy in the distribution of non-spam clicks and spam clicks between countries. While only about one-tenth of all non-spam clicks originate from a non-Scandinavian country, over three-quarters of all clicks marked as spam originate from countries outside of Scandinavia. This extreme difference in the amount of non-Scandinavian clicks justifies the use of country of origin as a feature. For more details, see section 5.2.

5.1.3 Spam distribution

One potential issue with carrying out a classification task on the data containing the click information is the large imbalance between the number of clicks marked as spam and the number of clicks marked as not spam. Of the approximately 38 million clicks of data supplied by Prisjakt, stretching over a time period of 6 months, only about 1.5% of clicks were marked as spam. This disparity between spam and non-spam results in the model being exposed almost exclusively to non-spam clicks during training. This can have a large impact on how well the model performs and has to be taken into account. As discussed in section 2.2.4, one of the most effective ways to tackle this problem is by assigning weights to each class as follows

$$\begin{aligned} \text{class\_weight\_for\_0} &= \frac{\text{total}}{2 \times \text{neg}}, \\ \text{class\_weight\_for\_1} &= \frac{\text{total}}{2 \times \text{pos}}, \end{aligned} \tag{5.1}$$

where *neg* and *pos* are the number of clicks marked as non-spam and spam respectively, and the *total* parameter is the total number of samples. The formula is based on the inverse proportionality of samples belonging to their respective weight, as discussed in 2.2.4. This meant that if non-spam clicks were 100 times more common than spam clicks, the spam clicks would be assigned a weight 100 times larger than the weight of non-spam clicks. However, the assigned weights varied slightly between training instances. This is because the subset of data that was used to train the model varied randomly between training instances, resulting in similar but slightly different distributions of spam and non-spam. The random variation was used in order to make sure that the model could perform well on different subsets of data, not just a certain sample of clicks. The size of the data set used to train the model was usually in the range of 1-10 million clicks and was mostly limited by memory constraints. For these data sets, the number of samples marked as spam was usually close to one percent, with class weights assigned accordingly.

5.2 Extracted Features

In order to increase the performance of models, having numerous features in the data set is usually advantageous [10]. In order to achieve this, a large number of features, beyond those in the data initially supplied by Prisjakt, were extracted. After all features had been extracted, the data set contained over 300 features. A large number of these were similar to each other, and described the same characteristic of the data, but over different time spans. Below is a concentrated presentation of the features used in order to train and evaluate the models.

5.2.1 Night Clicks

As shown in section 5.1.1, the activity of non-spam and spam clicks greatly differed during the night. This makes for a good indicator and was therefore added as a feature. In the data, a click that occurred between the hours of 00:00 and 06:00 was marked as being a *night_click*, represented by the numeric value 1. Clicks that occurred outside of this time span were marked as being not night clicks, represented by the value 0. With this representation, the feature could be fed directly to the model.

5.2.2 Country of origin

As discussed in section 5.1.2, there is a large difference between where non-spam and spam originate from. In order to incorporate this into the model's decision-making, the country of origin needed to be represented in the form of a feature. As shown in Fig. 5.2 of section 5.1.2, the possible countries of origin were split into three categories. As a result, two new features were developed, *swedish_click* and *scandinavian_click*. These features represent what category each click belongs to. The values associated with these features are the numerical values 0 and 1, where 0 represents false and 1 represents true.

As an example, a click originating from a Swedish IP address would have the value 1 assigned to both *swedish_click* and *scandinavian_click*, but a click originating from a Norwegian IP would have the value 0 assigned to *swedish_click* and the value 1 assigned to *scandinavian_click*. As might be obvious by now, a click originating from Germany would have both values set to zero. By limiting the representation of the country of origin of each click as belonging to one of three categories, the maximum memory consumption and complexity of the features are greatly reduced.

5.2.3 Origin-Target Based Features

Let the IP, ASN, host, and range columns be called origins and the store, product, category, and top-category columns targets. The following three features were then calculated for all origin-target pairs:

- The number of clicks the origin made on the target.
- The proportion all clicks the origin has made that is on the same target.
- The number different targets the origin has clicked.

Another feature counting the clicks the origin made independent of target was also calculated. These features were calculated over six time-spans centered around the time of the click, ranging from 10 seconds up to 80 hours. This means that both historical and future clicks are considered. In the end, this resulted in a total of 312 unique features.

The reasoning underlying these features is rather simple. As can be seen in Fig. 5.1, the activity of spam clicks is quite even for all hours of the day and night, meaning the sources generating spam generally don't take any breaks. Therefore, if we assume that the majority of the spam is generated by bots that can be left to run for long periods of time, the click counters should signal spam-like behaviour over longer time spans by being significantly larger than the normal non-spam user.

The centering of the time span was done in order to increase the chances of detecting both the first and last spam clicks in a series of spam-like clicks. For example, imagine the last click in a series of frequent clicks originating from a spammer. If the features were based on statistics from future clicks only, the counters for this click would not show any unusual activity. The same is true for the first clicks when only considering statistics from clicks backwards in time. By viewing each click in regards to both future and historical clicks the best of both worlds are combined.

5.3 Manual data cleaning

Some of the columns in the original data set supplied by Prisjakt, as shown in Table 2.1, had both numeric IDs and text descriptions. For example each store has a unique ID in the *ftgid* column as well as the name of the store in the *store_name* column. As discussed in section 2.2.2, the data fed to the model must be represented with numerical values, so when given to a model the text columns could be discarded. On the other hand, when analysing the data it was convenient to keep the text columns, for human-readable results.

One problematic feature which may not be entirely obvious is the IP address associated with each click. Due to the fact that the top 0.7% of spammers contribute to 50% of all spam clicks in the data, the IP addresses had to be removed before training the model to avoid the risk of the model over fitting the data. This was done in order to prevent a situation where the model would overfit by “remembering” the specific addresses that generate spam. A major underlying reason for why this is problematic is the fact that IP addresses can change somewhat frequently. Therefore, if the model’s classifications were based, to a sufficiently large extent, on the IP addresses the clicks originated from, the model would most likely fail to generalize well in the future.

However, as mentioned in section 5.2.3, some features related to the IP address were extracted and kept in the data. These features are more general than specific addresses, as many clicks can can have the same values in all or some of these features while originating from different IP addresses. Therefore, the risk of the model overfitting to specific values of these features should be kept reasonably small.

5.4 Model results

This section presents the results of the models most used throughout the development of the project. A large number of different models were trained and evaluated, and some of these models were discarded quite quickly. The reasoning behind this was simply that the discarded models were quite simple, and were outperformed by more advanced models. As the number of candidate models decreased, the thoroughness of the model evaluation increased. However, as the scope of the test cases widened, it became apparent that two models generally produced the best results. These two models were *XGBoost*, an implementation of parallel tree boosting, and a shallow neural network, with only one hidden layer.

5.4.1 Logistic Regression

One of the first models tested during the project was a simple logistic regression model. Using logistic regression as an initial model for classification tasks is quite common, usually with the purpose of inspecting the quality of the data before moving on to more advanced models [8]. This was also the case for this project. It is worth mentioning that the logistic regression model was more or less discarded even before most of the final features had been extracted and added to the data. Therefore, the results presented below might not be representative of the best possible performance achievable by using Logistic Regression. The results presented below were produced when training the model on about 1.5 million clicks, using twenty percent of the data as a test set.

		Predicted class	
		Ham	Spam
Actual class	Ham	303997	127
	Spam	1565	289

Accuracy	Precision	Recall	F1 Score
0.9945	0.6947	0.1559	0.2546

Figure 5.3: Confusion matrix and evaluation metrics for the linear regression model.

Figure 5.3 shows the results produced by the logistic regression model. Since the model was developed in the very early stages of the project, the amount of features used differs largely compared to the other final models'. As can be seen in the figures, the results produced by the model were unfortunately quite far from optimal. Instead of providing a useful solution to the problem, the results of the linear regression model acted as a baseline for future models. By using the results in this manner, models which performed worse than the logistic regression model could quickly be discarded.

5.4.2 Bagging Classifier

The first model to show substantial improvements in performance compared to the logistic regression model was a bagging classifier. As discussed in section 3.3.3, bagging classifier models are more complex than simple logistic regressors, and should therefore generally perform better in comparison.

Figure 5.4 shows the results produced by the bagging classifier in the form

		Predicted class	
		Ham	Spam
Actual class	Ham	592278	50
	Spam	161	7511

Accuracy	AUC	Precision	Recall	F1 Score
0.9996	0.9895	0.9934	0.9790	0.9861

Figure 5.4: Confusion matrix and evaluation metrics for the bagging classifier.

of a confusion matrix and the values of the performance metrics used to evaluate the model. These results were achieved after training the classifier using three million clicks, with 20% of the clicks used as a test set. The only hyperparameters which had been adjusted was the number of decision trees used in the model as well as the number of features used to train each tree. A total of 120 decision trees were used, as this was found to be a rough optimal point between the increase in performance and the increase in time required to train each tree. After some experimentation, the number of features used to train each tree was set to 20, partly inspired by the list of feature importance presented in appendix A. This was found to be a good balancing point, which led to a model that performed well while simultaneously avoiding overfitting.

As can be seen in Fig. 5.4, the results are far better than the ones produced by the logistic regressor, and are quite close to the maximum value of 1.0 [8]. However, it should be stressed that these results were not the result of only a change of model. Extensive feature engineering, which led to several additional features not used with the logistic regressor, helped improve the performance of the bagging classifier.

Even though the performance of the model might seem satisfactory at first glance, the bagging classifier has one flaw which is not immediately obvious. By skipping 10 million rows in the data set in order to create a gap between the data used to train the model and the samples used to evaluate it, the bagging classifier was found to be quite vulnerable to slight changes in the data. This was a more subtle form of overfitting, as the model performed well on the test set taken from the same contiguous block of data the model had been trained on. Still, the model performed significantly worse when evaluated on data further ahead in time. This poses quite a considerable problem, as it ideally should not be necessary to re-train the model all too often, but will be if the model's performance starts to deteriorate rather soon after it has been trained.

5.4.3 XGBoost

The XGBoost library was chosen primarily because it had been used to produce top-performing models in Kaggle competitions [54] [55] [56]. As the inner workings of XGBoost is pretty similar to that of a bagging classifier, it's reasonable to expect roughly similar performance as well. As can be seen in Fig. 5.5 the XGBoost model lived up to these expectations.

		Predicted class	
		Ham	Spam
Actual class	Ham	296972	22
	Spam	52	2954

Accuracy	AUC	Precision	Recall	F1 Score
0.9998	0.9913	0.9926	0.9827	0.9876

Figure 5.5: Confusion matrix and evaluation metrics for the XGBoost model.

In general, the XGBoost model achieved slightly better results compared to the bagging classifier, but the differences are quite minuscule, especially when talking about millions of samples. It should be pointed out that the XGBoost model was initially trained on only 1.5 million samples, less than half of what was used to train the bagging classifier. This is due to the fact that further feature engineering took place between the transition from bagging classifier to XGBoost. This resulted in a total of over 300 features, a much larger amount than what was used to train the previous models. Unfortunately, this also resulted in a very high memory consumption, limiting the model to be trained on 1.5 million samples on a computer with 32 gigabytes of RAM. Furthermore, the XGBoost model used 140 estimators, 20 more than the 120 used in the bagging classifier. This also contributed to the increased memory consumption, but was found to be crucial for the long-term performance of the model.

However, as can be seen in Fig. 5.6, the learning curve of the model did not really improve beyond the first 20 or so iterations. As one additional estimator is added per iteration, the graph could be interpreted as an indicator that 140 estimators were quite a bit more than required. However, these estimators were also found to improve the long-term performance of the model. Because of this, it could perhaps be interesting to investigate whether tuning other hyperparameters than the number of estimators could improve the long-term performance as well. If this would be the case, then the number of estimators used could perhaps be significantly reduced.

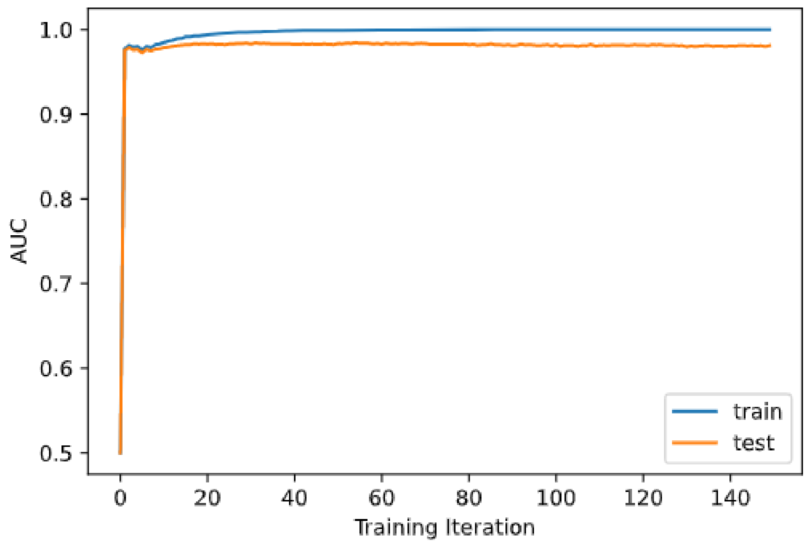

Figure 5.6: AUC-Learning Curve for the XGBoost model.

Beyond the slight improvements in performance metrics, the XGBoost model provided one significant advantage over the bagging classifier; its performance did not decline as much when an offset between the training data and evaluation data was introduced, as can be seen in Fig. 5.7. In the figure, the model is trained and evaluated on the first 5 million clicks, then further evaluated on consecutive chunks of 5 million clicks. The dip around the 25-30 million click chunk is most likely due to the Black Friday week, which is notorious for its unusual user behaviour. With the exception of the Black Friday week, the model seems to stay relatively performant. Because of this, the XGBoost model should not have to be re-trained as often as the bagging classifier would have.

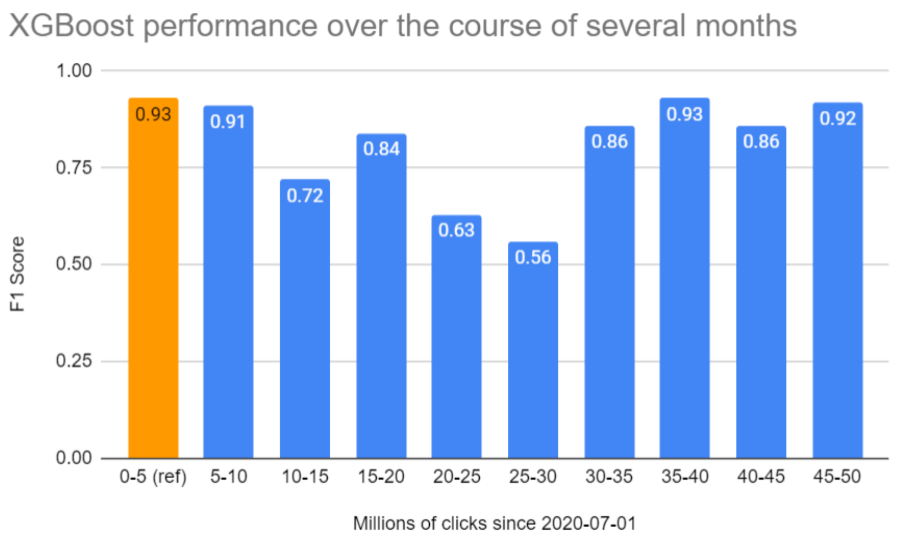

Figure 5.7: F1 Score when predicting clicks of consecutive chunks of 5 million. Each bar corresponds approximately to one month.

5.4.4 Neural Network

Except for quick prototyping in the early stages of the project, neural networks were saved for last. The reason being is that they are incredibly complex, non-intuitive, and require a lot of research and dedicated time. However, of all the classifiers they *might* also be the ones with the highest performance potential [57].

One of the hardest things with training a neural network is deciding where to start. There are many hyperparameters to adjust, and there is no clear, single "best" value for any parameter, since they ultimately depend on the specific data set. However, there are some conventions and commonly agreed "good", reasonable values and rules of thumb that tend to lead to a decent initial model. For example, a batch size of 32 is generally considered a good starting point [58], but a size of 64, 128 or 256 trains faster and could potentially give better results, and should therefore be tested. In fact, it turned out that a batch size of **512** is most likely optimal for this network, but sizes of 1024 and 2048 were 2x and 4x faster respectively, with marginally worse results. Therefore these were most often used during intensive training.

Another hard choice, if not the hardest, was determining the shape of the neural network, i.e. how many layers and nodes to use for the model. More

		Predicted class	
		Ham	Spam
Actual class	Ham	985741	852
	Spam	100	13307

Accuracy	AUC	Precision	Recall	F1 Score
0.9990	0.9958	0.9398	0.9925	0.9654

Figure 5.8: Confusion matrix and evaluation metrics for the neural network.

specifically, how many *hidden* layers and nodes. The general consensus is that one hidden layer, that uses a number of nodes somewhere in between the size of the input and output layer, is sufficient for the large majority of problems [59]. In addition, there's also a performance incentive in keeping the complexity low, since the less complex a model is, the faster it is to train. In fact, after extensive testing with many different combinations of layers and nodes used, it seemed like a network consisting of an input, output and one hidden layer with 150 nodes consistently gave the best results.

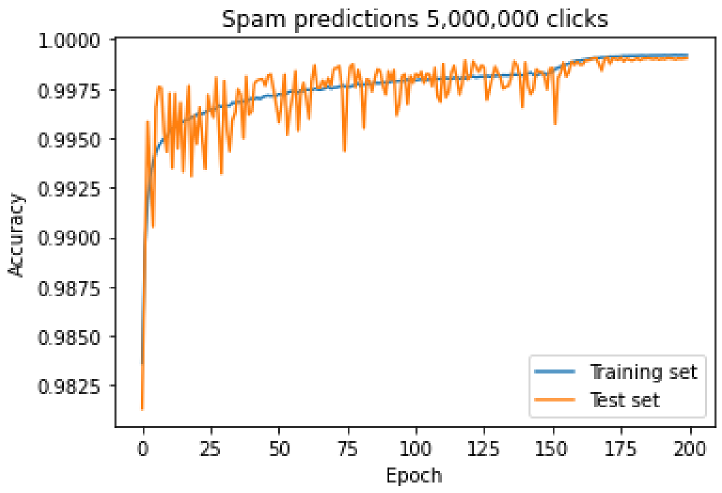

Figure 5.9: Learning curve for the neural network.

This network topology was ultimately used to test and obtain the best values for the rest of the hyperparameters. Training was set for 200 epochs since it seemed like the model converged well by then. Binary cross entropy was used as loss function since it's a natural choice for binary classification [60]. Lastly, the Adam (Adaptive Moment Estimation) was initially only used as a first, zero-config, easy-to-use optimizer which promised good results and efficient training [61]. But after comparing many other optimizers, especially SGD (Stochastic Gradient Descent) [62], it turned out that Adam with default parameters actually was the best optimizer for this data set and network.

Ultimately, a fine-tuned neural network model, with over 300 features, was trained on 5 million clicks. The results can be assessed in Fig. 5.8 in the form of a confusion matrix and the evaluation metrics. Additionally, the learning curve of the neural network is included in Fig. 5.9. The learning curve compares training vs. test results and shows how the model first classified everything as non-spam and then started learning from its mistakes, increasing its accuracy. As the accuracy started to converge, learning rate decay was applied from epoch 150, further squeezing out the last bit of performance.

The end result is a model that shows very good results, however the XGBoost seems still slightly better. Although the neural network requires less memory, allowing it to train on 5 million instead of 1.5 million clicks at a time, with the same amount of features, on 32 gigabytes of RAM.

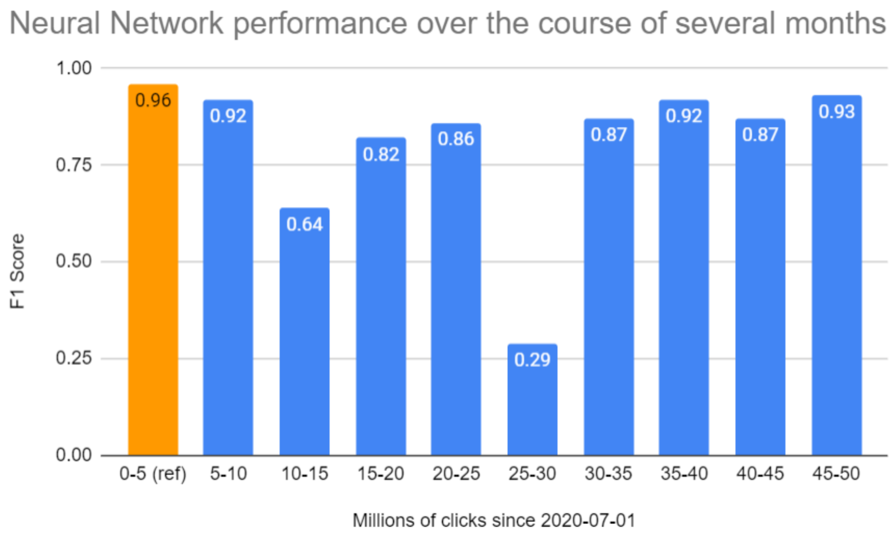

Figure 5.10: F1 Score when predicting clicks of consecutive chunks of 5 million. Each bar corresponds approximately to one month.

The model performs, in general, well even for newer data than it was trained on. As can be seen in Fig. 5.10, the model performs well on several consecutive chunks of 5 million clicks. The model was evaluated on the first 0-5 million clicks, which is the same 5 million it was trained on, and acted as a reference point. The evaluation continued with chunks of 5 million clicks since that is roughly equal to one month's worth of clicks on Prisjakt's Swedish market. The biggest outlier is the 25-30 million clicks chunk. This is expected since that chunk coincides with the Black Friday week, during which Prisjakt receives a greatly increased amount of legitimate clicks. Because of the seemingly “spammy-nature” of these clicks, the model falsely classifies a lot of legitimate clicks as spam. However, the total amount of classified spam is still around 1.5% and therefore the stores' monthly bills should not deviate far from what they should be.

5.5 Application Programming Interface

In addition to the machine learning models, an API was also developed in order to enable other systems to make use of the models. The purpose of this interface is to streamline the steps necessary for using the models, thereby making them easier to use. A part of this process was to collect all the feature-generating code into one cohesive logical module. This module is then used within the API with the purpose of generating features for the incoming data. In this way, the spam detection model and all feature extraction code can be run without having to manually oversee each part of the process.

In its current state, the API allows users to select a file containing click data, from which the features in section 5.2 are extracted. After all features have been created, a saved and pre-trained model is loaded and used to make predictions on the data. For each click in the data, the predicted classification is returned along with the id of the click in the original data. In this way, it is possible to generate data for which clicks are marked as spam, without having to manually execute each of the scripts that constitute the entire process.

6
Discussion

This section contains a discussion of the results presented in the previous chapter. The main topic of discussion is the practical usefulness of the models in industry applications. Discussions about possible future work and ethical aspects of the project are discussed in chapter 7.

6.1 Model Results

As shown in section 5.4, the final results of the XGBoost model and the neural network were very close to the ideal values for the selected performance metrics. As such, these models seem to perform well enough for an automated spam-identification system based on machine learning to be possible. However, the models have not been tested in an actual production environment, which could introduce new challenges. It is likely that most of these challenges should not directly affect the predictive performance of the models, only their ease-of-use. Therefore, it is probable but not guaranteed that the performance of the models presented in this paper should reasonably reflect their performance in a production environment.

Testing the performance of the models in a context more similar to the production environment at Prisjakt would allow further development of models and the project as a whole. By utilizing the models through the API, both the capabilities of the interface and the performance of models would be tested. If sufficient amounts of data were gathered, the long-term performance of using and re-training models through the API could also be explored further.

6.1.1 XGBoost vs. Neural Network

Which model is the best? It's hard to say since both models perform so evenly well and there is no clear winner in the side-to-side comparison shown in Fig. 6.1. However, the XGBoost model has a slightly higher average F1 score (0.816 vs. 0.808) in this comparison, as well as reaching the highest peak F1 score in previous tests. The fact that it doesn't dip as much during the Black Friday week, compared to the neural network, is also of high importance. In the end, both models perform really well but it seems XGBoost is slightly advantageous.

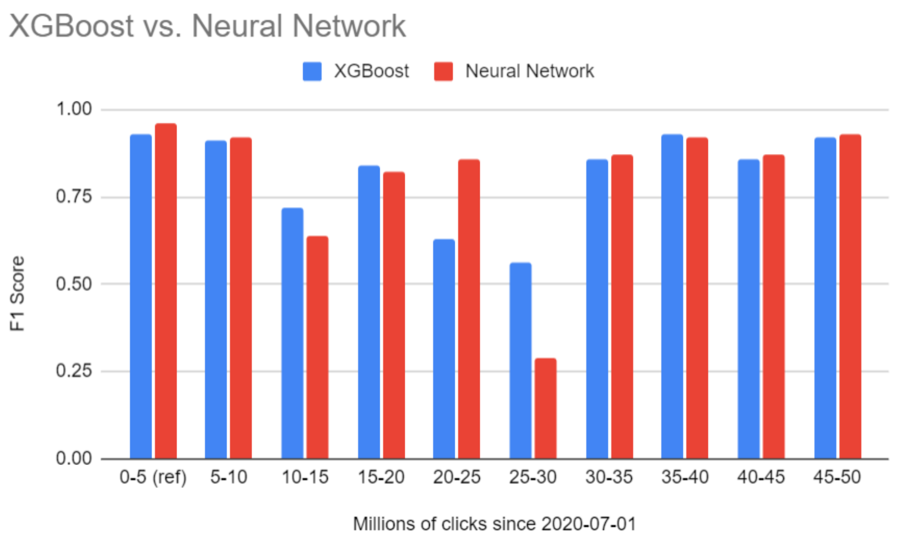

Figure 6.1: F1 Score comparison of XGBoost vs. Neural Network

6.1.2 Memory Limitations

As mentioned in section 5.4.3, the memory consumption of the models proved to be a significant limitation at first, especially in terms of their long-term performance. These memory limitations could be overcome by saving and loading models in-between training sessions. For example, as the memory consumption of the XGBoost model was quite high due to the large number of estimators used, many iterations had to be run in order to train the model on large amounts of data. For each iteration, the training set consisted of one million samples. With a simple script which incremented a counter controlling which data was read, the model could be trained on as much data as the time and amount of available data would allow. A similar construction could be used in order to benchmark the models performance on amounts of data larger than would simultaneously fit into memory. This approach is not specific to a certain type of model, and could be universally applied, effectively removing the memory limitations. As a result of using both the scripts for iterated training and for iterated predictions, the performance of models could be improved and performance flaws could be more easily spotted.

6.2 Validity of the Labels

An important factor to highlight is whether or not the spam-labels in the raw data provided by Prisjakt, see section 2.1, are accurate. The labels are created by Prisjakt employees that manually flag and remove clicks they believe originated from bots. As such, the accuracy of the labels will only be as good as the employees ability to correctly identify spam. Since the employees at Prisjakt are human and the bots generating spam can be quite sophisticated, the labeling of the data will probably not be perfect. As these labels are used to train the model, this will naturally affect its ability to detect actual spam.

This raises the question of whether or not the labels should be trusted. The answer to that question is quite simple; there is practically no other choice. The only other option is to use unsupervised learning with clustering, but in order to measure the clustering performance the labels are still needed, since they are the closest to the truth that is available.

6.3 Features not used

During the project's development there were many ideas for features that never were carried out. For example one idea was to look at if a certain IP-address had been marked as spam recently. Another idea was to first find which stores had unusually much traffic, and then find the IP-addresses mostly responsible.

A reasonable question to ask is why the project was proclaimed as finished even though further features could be added. The reasoning behind this can be summarised in two points. First, the results that the later models had were so satisfactory that there was not much meaning in spending time developing more features. The other reason is that adding more features would not always give better results. For example, if the model got to look at historical data of spam, it might get biased and only follow the old predictions, but never adapt to any changes.

6.4 Circumventing the Model

One point of interest is to consider what a bot could do in order to avoid detection from the model. The relative feature importance can be found as discussed in appendix A. The most important features are the features from section 5.2.3 that describe whether the IP-, range-, ASN-, and host-columns, from section 4.2, tend to target a specific category, or store. Additionally, features that describe the absolute number of clicks from the same IP, range, ASN, and host are also important.

To counter detection, someone making a bot might try to spread out its clicks over multiple stores. However, they would then either have less impact towards their targeted store or have to increase the number of clicks. In the latter case they would get an even higher chance of being detected, as the click-data would stand out even more. Another strategy might be to decrease the number of clicks to avoid detection, but then again, the bot would have to decrease its impact until it could no longer be considered spam. It is also noteworthy that Prisjakt's website functions identically whether or not the user gets marked as spam. Therefore a bot would have to be built without any direct feedback about if it is detected, which makes it much more difficult to evade being marked as spam.

One approach which seems both feasible and rather easy to implement is to adjust the activity of the spam-bots according to the time of day. By doing so, the bots could become less active during the night in order to mimic the overall activity on the site and resemble ”normal” activity. However, this would only have a major effect on a single feature in the data and would therefore not substantially affect the decision made by the model.

6.5 Application Programming Interface

As mentioned in section 5.5, an API was developed in order to make the models more usable. In its current state, it enables a streamlined process for marking individual clicks as spam or non-spam. However, there is still potential for further development, which will be presented more in detail below.

One area where the API could be improved is in its flexibility. Currently, what model is used to generate predictions is not up to the user of the API. Instead, what model to use is hard-coded in the interface. An easy solution to this would be creating different endpoints for different models. In combination with this, the features to be extracted and fed to the model are also pre-determined and not chosen by the user. However, the selection of features might not be a significant issue. As long as the representation of click data used at Prisjakt does not change, the feature extraction part of the API will work. Additionally, as seen in section 5.4, these features seem to produce good results. Furthermore, allowing the user to select a certain model for making predictions is likely not a problem which requires significant development effort. Possible solutions could include offering several pre-trained models for the user to choose between, or allowing users to supply their own pre-trained model along with the data to be used for feature extraction.

Along with future improvements to the API, the development process used up to its current state should also be discussed. One interesting aspect to note was that Jupyter Notebooks were the main development environment used for the machine learning aspects of the project. Because of the interactive nature of these notebooks, the code was not initially organized as ”regular”, more cohesive software systems usually are. This proved to make the development of the API more difficult, as the code had not been written entirely with the API in mind. As a result of this, the development of the API required additional time and effort in the later stages of the project. If the API had been more central in the earlier development stages, this could perhaps have been avoided and more effort been put into other, more productive work.

Since no specification for the API or details of the underlying system were supplied by Prisjakt, the API-development followed no strict guidelines and therefore resulted in a very general interface. The API will most likely be used locally within Prisjakt, and thus no API security was implemented. However, if the API will be exposed to the public, a more secure API needs to be developed in order to prevent data breaches.

7 Conclusion

This section presents some key points and conclusions which can be drawn from the results and the discussion surrounding them. Additionally, possible future work related to the project is discussed, along with the social and ethical aspects of the project and its results.

7.1 Applicability of the system

As mentioned in section 5.4, the performance achieved by the models were quite close to the ideal values for the chosen metrics. These results indicate that a machine-learning based system for detecting spam is plausible. When in use, the input data to a model will be clicks made at a later date than those the model were trained with. As mentioned in 4.5, some models might not work with the new data if the pattern the model has found is different for these new clicks. But, even when classifying a few months newer clicks, the final model would still perform well. Therefore, it is likely that the model will work when put into use by Prisjakt.

Further testing the models, using the API and data from various points in time for training and making predictions, could provide more insight into the performance of the models. However, the most effective way of discovering how well these models perform in the actual production environment at Prisjakt is probably to deploy them in that environment and measure how well they perform. Since this could be done alongside the current manual spam-filtering, the models do not necessarily need to affect the spam statistics. Furthermore, the hyperparameters of the models are quite easily altered, thereby making adapting the models to the production environment rather straightforward.

The question of how often the model needs to be re-trained still remains. As the results of the neural network indicates in section 5.4.4, the performance of the models seems to stay relatively the same with time. Therefore it should be possible to use a trained model for between six months to a year, before re-training it with new manual predictions. However, special occurrences such as holidays and Black Friday, which significantly increases the amount of traffic on the site, could be problematic. As this increase in traffic only constitutes a small part of the overall data, it could likely affect the performance of the model, most likely leading to a higher amount of clicks being falsely marked as spam. Although, it might be possible to mitigate this effect by training the model on sufficient amounts of data, which contain several occurrences of sales-increasing occasions. Nonetheless, it is highly recommended to use the model together with some common sense to assess if the model's predictions are reasonable.

7.2 Future work

One possibility for improving the project is to develop a system which automatically uses the classifications made by the ML model in order to actively remove spam from the data. In its current form, the model correctly identifies clicks which would have been marked as spam by employees at Prisjakt to a high degree. However, the removal of data marked as spam is left to be manually carried out by employees at the company, reducing the degree of automation in spam-removal. Developing such a system should most likely be quite easy, although some care needs to be taken in order to prevent the accidental removal of click-data from the bills each month.

Some further work on the machine learning model is also possible. As mentioned as a limitation in section 1.3, the model does not distinguish between spam generated by humans and spam generated by bots. In certain contexts, this distinction could be useful. For example, both interactions with and reactions to content on social media is often generated by humans that repeatedly click on content using several different devices [63], [64]. In order for this to be possible, additional data with human-spam labels is required. Producing such data might not be trivial, but should be possible to generate by manual labeling as long as it is possible to distinguish between human-generated spam and bot-generated spam by inspecting the data.

7.3 Social and Ethical Aspects

One of the main ethical dilemmas of the project is the collection and management of user data. As the data contains information such as the IP address of the user and what products the user has clicked on, it could be possible to roughly trace what products someone has clicked on. As with any personal information, it could be regarded as sensitive and should therefore be handled with care. Furthermore, some users might object to any personal data being collected when using online services. The cookie-selector on the Prisjakt website allows users to opt out of most data-collection, with the exception of information necessary for maintaining the current web-session such as usernames and passwords [65].

According to the General Data Protection Regulation, individuals have the right to ask organizations to delete any personal data related to the individual [66], [67]. However, the use of machine learning models might complicate the implications of this right. If a machine learning model is trained on personal data, does deleting the personal data also entail deleting, or at least re-training, the model? In some cases, it might be possible to draw some conclusions about the nature of the underlying data used to train the model, simply by using and querying the model. Therefore, it could be argued that some aspects of user data are retained in the behaviour of the machine learning model.

As a result of this, it could also be argued that in order to completely remove the personal data associated with an individual, the model has to be re-trained without the use of that individual's personal data. On the other hand, it can be argued that since the model itself is not an actual database containing the personal data, the likelihood of someone being able to reconstruct actual data by using the model is highly unlikely. For large-scale such as ours probably the case since each individuals personal data is a minuscule fraction of the entire data set used for training.

As previously mentioned, the data collected from users' clicks could be considered sensitive, and should therefore be treated as such. Because the machine learning system will be integrated with the existing system at Prisjakt, there is no need to transport this data outside the local network at Prisjakt. However, malicious users exist and data leaks happen. Therefore, protective precautions should be taken, perhaps storing the data using an approach similar to the hash-and-salt approach used for storing passwords [68].

Which clicks are marked as spam only affects the bills sent to stores at the end of the month. Because of this, users are not at all directly affected by the decisions taken by the model. Any eventual bias in the model towards some group of individuals is therefore very unlikely to affect any users.

Prisjakt and the stores that use Prisjakt's services could be monetarily affected by the machine learning system. As Prisjakt bills stores depending on the amount of clicks each store receives, it is in Prisjakt's interest to maximize these click amounts. One way to do this would be to minimize the amount of clicks marked as spam, thereby maximizing revenue. This approach would not be very ethical. However, it is also in Prisjakt's interest to increase the number of stores that use their services. In order to do so, their service has to be fair and correctly implemented, something which most likely prevents the implementation of a skewed spam-detection system. During the project, the goal has been to ensure that the classification of spam is as accurate as possible. This has been done by maximizing both of the metrics precision and recall, aiming at marking all spam and only spam as such. However, mistakes can happen, and thus Prisjakt will be recommended to evaluate the model regularly, in order to detect any flaws or biases which may exist.

Bibliography

[1] Oxford Dictionary. 2021. URL: https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095426960.

[2] Wikipedia. 2021. URL: https://en.wikipedia.org/wiki/Internet_bot.

[3] Google. 2021. URL: https://developers.google.com/machine-learning/glossary.

[4] Imperva. 2021. URL: https://www.imperva.com/blog/bad-bot-report-2021-the-pandemic-of-the-internet/. Accessed on: 19/04/2021.

[5] Cloudflare. *What is a Spam Bot?* 2021. URL: https://www.cloudflare.com/learning/bots/what-is-a-spambot/.

[6] Cloudflare. *What is Click Fraud.* 2021. URL: https://www.cloudflare.com/learning/bots/what-is-click-fraud/.

[7] Prisjakt. 2021. URL: https://www.prisjakt.nu/.

[8] Steven S. Skiena. *The Data Science Design Manual.* Vol. 1. Springer, 2017.

[9] *Feature Engineering for Machine Learning.* O'Reilly media, 2018. URL: https://www.repath.in/gallery/feature_engineering_for_machine_learning.pdf.

[10] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow.* Vol. 2. O'Reilly, 2019.

[11] Joel Grus. *Data Science from Scratch - First Principles with Python.* O'Reilly Media, 2015.

[12] Aakarsha Chugh. 2021. URL: https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/. Accessed on: 04/05/2021.

[13] *Deep Learning with PyTorch.* Manning, 2020.

[14] SKLearn. URL: https://scikit-learn.org/stable/modules/preprocessing.html.

[15] scikit learn. *Preprocessing data*. 2021. URL: https://scikit-learn.org/stable/modules/preprocessing.html.

[16] Jason Brownlee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/. Accessed on: 14/04/2021.

[17] Jason Brownlee. *A Gentle Introduction to Imbalanced Classification*. 2019. URL: https://machinelearningmastery.com/what-is-imbalanced-classification/.

[18] *On the Class Imbalance Problem*. 2021. URL: https://www.researchgate.net/profile/Gongping-Yang/publication/228612392_On_the_Class_Imbalance_Problem/links/5808252308aefaf02a2c6734/On-the-Class-Imbalance-Problem.pdf.

[19] TensorFlow. *Classification on imbalanced data*. 2021. URL: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.

[20] *Learning From Imbalanced Data*. Springer, 2018.

[21] *Dimensionality Reduction: A Comparative Review*. 2009. URL: https://members.loria.fr/moberger/Enseignement/AVR/Exposes/TR_Dimensiereductie.pdf.

[22] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[23] *Feature Engineering And Selection: A Practical Approach for Predictive Models*. 2019. URL: www.feat.engineering.

[24] Peter Harrington. *Machine Learning in Action*. Manning, 2012.

[25] *Artifical Intelligence - A Modern Approach*. Prentice Hall, 2009.

[26] Google. *Classification: ROC Curve and AUC*. 2021. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[27] Martin Thoma. *Receiver Operating Characteristic (ROC) curve with False Positive Rate and True Positive Rate*. 2018. URL: https://commons.wikimedia.org/wiki/File:Roc-draft-xkcd-style.svg.

[28] scikit-learn. *sklearn.metrics.f1_score*. 2021. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[29] Thomas Wood. *F-Score*. 2021. URL: https://deepai.org/machine-learning-glossary-and-terms/f-score.

[30] Thomas Wood. *What is the Sigmoid Function?* URL: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function. Accessed on: 09/04/2021.

[31] Grant Sanderson 3Blue1Brown. *But what is a Neural Network?* URL: https://www.youtube.com/watch?v=aircAruvnKk. Accessed on: 09/04/2021.

[32] Gilgoldm. *Survival of passengers of the Titanic (modified)*. 2020. URL: https://commons.wikimedia.org/wiki/File:Decision_Tree.jpg.

[33] scikit-learn. *Bagging Classifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html. Accessed on: 09/04/2021.

[34] Jason Brownlee. *How to Develop a Bagging Ensemble with Python*. URL: https://machinelearningmastery.com/bagging-ensemble-with-python/. Accessed on: 09/04/2021.

[35] Jason Brownlee. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. URL: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/. Accessed on: 09/04/2021.

[36] Michael Nielsen. *Neural networks and deep learning*. URL: http://neuralnetworksanddeeplearning.com. Accessed on: 09/04/2021.

[37] Eda Kavlakoglu. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?* URL: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. Accessed on: 09/04/2021.

[38] Guido van Rossum. 2021. URL: https://www.python.org/.

[39] Santi Seguí Laura Igual. *Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications*. Vol. 1. Springer, 2017.

[40] Peter Norvig Alon Halevy and Google Fernando Pereira. *The Unreasonable Effectiveness of Data*. 2009. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35179.pdf.

[41] Wes McKinney. 2008. URL: https://pandas.pydata.org/.

[42] John D. Hunter. 2003. URL: https://matplotlib.org/.

[43] Travis Oliphant. 2006. URL: https://numpy.org/.

[44] *Pickle*. URL: https://docs.python.org/3/library/pickle.html. Accessed on: 16/04/2021.

[45] Armin Ronacher. 2010. URL: https://flask.palletsprojects.com/en/1.1.x/.

[46] Hadi Asghari. *pyasn*. URL: https://github.com/hadiasghari/pyasn. Accessed on: 16/04/2021.

[47] *py-radix*. URL: https://github.com/mjschultz/py-radix. Accessed on: 16/04/2021.

[48] *Route Views Archive Project Page*. URL: http://routeviews.org/. Accessed on: 16/04/2021.

[49] *Whois.net*. URL: http://whois.net/. Accessed on: 08/04/2021.

[50] Marco d'Itri. *whois*. URL: https://manpages.debian.org/stretch/whois/whois.1.en.html. Accessed on: 08/04/2021.

[51] *ANSs by countries*. URL: https://ipinfo.io/countries. Accessed on: 16/04/2021.

[52] *Cython*. URL: https://cython.org/. Accessed on: 03/05/2021.

[53] scikit-learn. *GridSearchCV*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed on: 14/04/2021.

[54] Kaggle. URL: https://www.kaggle.com/. Accessed on: 15/04/2021.

[55] Rachael Tatman. *Machine Learning with XGBoost (in R)*. URL: https://www.kaggle.com/rtatman/machine-learning-with-xgboost-in-r. Accessed on: 15/04/2021.

[56] Dan Becker. *XGBoost*. URL: https://www.kaggle.com/dansbecker/xgboost. Accessed on: 15/04/2021.

[57] Fast Forward Labs. *Accuracy and Interpretability*. 2021. URL: https://ff06-2020.fastforwardlabs.com/#accuracy-and-interpretability.

[58] *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. URL: https://arxiv.org/pdf/1609.04836.pdf.

[59] Jeff Heaton. *Introduction to Neural Networks with Java, 2nd Edition*. Heaton Research, 2008.

[60] Jason Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks*. 2019. URL: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

[61] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.* 2021. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[62] Shaoanlu. *SGD > Adam?? Which One Is The Best Optimizer: Dogs-VS-Cats Toy Experiment.* 2017. URL: https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/.

[63] Charles Arthur. *How low-paid workers at 'click farms' create appearance of online popularity.* URL: https://www.theguardian.com/technology/2013/aug/02/click-farms-appearance-online-popularity. Accessed on: 15/04/2021.

[64] *Microsoft Sues Over Online Advertising 'Click Fraud'.* URL: https://web.archive.org/web/20150525160200/http://www.bloomberg.com/news/articles/2010-05-19/microsoft-sues-web-site-over-new-form-of-online-advertising-click-fraud-. Accessed on: 15/04/2021.

[65] Prisjakt. *Personaliserade annonser.* URL: https://www.prisjakt.nu/privacy-settings. Accessed on: 16/04/2021.

[66] Ben Wolford. *What is GDPR, the EU's new data protection law?* URL: https://gdpr.eu/what-is-gdpr/. Accessed on: 16/04/2021.

[67] Ben Wolford. *Everything you need to know about the "Right to be forgotten".* URL: https://gdpr.eu/right-to-be-forgotten/. Accessed on: 16/04/2021.

[68] William Stallings. *Cryptography and Network security.* 7th ed. Pearson, 2017.

[69] scikit-learn. *Permutation feature importance.* URL: https://scikit-learn.org/stable/modules/permutation_importance.html. Accessed on: 14/04/2021.

[70] scikit-learn. *Feature importances with a forest of trees.* URL: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. Accessed on: 28/04/2021.

[71] Jason Brownlee. *Feature Importance and Feature Selection With XGBoost in Python.* URL: https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/. Accessed on: 28/04/2021.

[72] Jason Brownlee. *How to Calculate Feature Importance With Python.* URL: `https://machinelearningmastery.com/calculate-feature-importance-with-python/`. Accessed on: 14/04/2021.

A

Feature Importance

If the data used for training models contains a large number of features, the amount of time and memory required for training can be quite large. One way to deal with this issue is to simply use fewer features when feeding data to the model. However, using fewer features can also result in the model performing worse. In order to determine which features should be removed and which to keep, some measure of feature importance needs to be used. In fact, this problem of feature selection is so common that the API's of most models contain a function for retrieving the relative importance of features after a model has been trained [69], [70], [71].

In order to effectively use such functions, the model in question first has to be trained using data which contains all candidate features. After training, the model can be queried for the importance of the features in the data used to train it. The higher the value assigned to a feature is, the more important it is for the performance of the model. By removing the least important features, the memory and time used for training can be reduced while simultaneously retaining as much of the model's performance as possible.

In the case of this particular project, the largest data set generated after extensive feature engineering contained more than 300 features. As the models grew in complexity, this large amount of data became difficult to use, as the amount of available memory was limited. Therefore, an effort was made to reduce the amount of features used when training models, in order to speed up the process and to enable training on larger amounts of data. This was done by querying models trained on the entire data set for the importance of each feature.

As can be seen in figure A.1, three of the more than 300 features drastically stand out in terms of their importance. However, training models using only these three features resulted in models with very poor performance. Therefore, a larger set of features needed to be used. Figure A.1 also shows that there is a quite large variety in feature importance beyond the three most important features, meaning which features are most effective is not immediately apparent.

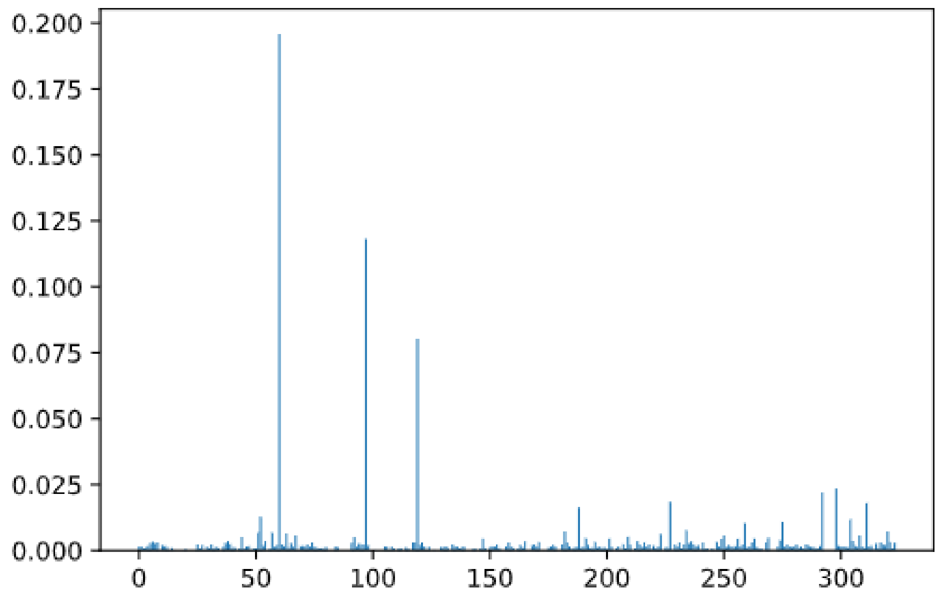

Figure A.1: The relative importance of each of more than 300 features.

The approach used for selecting which features to keep in the data was slightly experimental in nature, at least in deciding how many features to keep. Different feature-subsets were used in order to determine which features should be used, all of them prioritizing features in accordance to their importance as assigned by the model. For example, one subset consisted of the fifty most important features, one of the ten most important features, etc. After a set of features had been selected, they were used in order to train a model. The resulting performance was then compared to that of models trained using all available features. In this way, trade-offs between reductions in performance and memory-consumption could be established for each feature set. The goal of this was to find a set of features which was quite a bit smaller than the full set, while producing a well-performing model.

However, it should be noted that the exact importance assigned to each feature by a given model varied depending on the exact data used to train it [72]. This complicated matters slightly, as smaller feature-sets containing only the most important features were quite likely to vary depending on the data the model was trained on. This problem could largely be avoided by selecting a sufficiently large set of features. A rough estimate for avoiding this issue was about 50 features. This estimate was produced by repeatedly generating the 50 features assigned the highest importance and comparing the generated sets. In doing so, it was revealed that the generated sets were mostly different permutations of the same features, i.e. the same features with a different importance assigned to them. An example of such a set, extracted from an XGBoost model, can be seen in table A.1 below.

Furthermore, by using the 50 most important features, significantly less memory was required when training the models, while only very slightly reducing their performance. As other sets of features were explored, it became apparent that the *exact* number of features in the set did not matter to a significant degree. Therefore, models trained on the 60 most important features did not generally achieve much better performance than models trained using 50 features. In the end, a set of the 65 most important features were used for training models and using trained models in order to classify samples.

Table A.1: Relative importances of the most important features extracted from an XGBoost model

Feature Name	Relative Feature Importance
ip_proportion_same_category_40_h	0.114757575
ip_proportion_same_category_10_h	0.11266905
ip_proportion_same_product_80_h	0.0879142
ip_proportion_same_category_80_h	0.04502224
host_proportion_same_category_10_h	0.03500797
asn_different_stores_10_h	0.02503607
ip_proportion_same_product_40_h	0.023604788
host	0.019678049
ip_proportion_same_store_80_h	0.017856166
host_proportion_same_store_10_h	0.016970403
asn_different_stores_40_h	0.011041589
host_different_top_categories_40_h	0.010540295
host_proportion_same_category_40_h	0.010260639
host_different_top_categories_10_h	0.009066003
asn_different_categories_80_h	0.007799757
range_proportion_same_category_80_h	0.0071118427
asn_proportion_same_store_10_m	0.006121386
asn_proportion_same_store_10_h	0.0058054435
range_different_top_categories_10_h	0.00554897
host_proportion_same_category_80_h	0.0055161584
ip_different_top_categories_30_m	0.0052157133
range_proportion_same_store_80_h	0.0049197227
host_proportion_same_store_40_h	0.0048059835
ip_proportion_same_product_10_h	0.004753329
host_proportion_same_top_category_40_h	0.00460298
asn_proportion_same_category_40_h	0.004262292
ip_different_categories_30_m	0.004187511
range_different_categories_30_m	0.0041209455
ip_different_categories_80_h	0.0039748317
range_different_categories_80_h	0.0036985266
range_proportion_same_top_category_80_h	0.003641434
asn_proportion_same_product_10_h	0.003599033
range_proportion_same_product_10_m	0.0034849304
host_different_top_categories_10_s	0.0032750817