Machine learning theory

Tran Thien Thi

June 2019

Contents

1 Introduction

In this document, you will learn what is Internet traffic classification and why it is needed. After that, you will learn basics of machine learning and its usual procedure. Finally, you will learn how the Internet traffic classification could be implemented using machine learning.

TODO: figures seems to be too far from their text so find a way to deal with Latex placements

2 Internet traffic classification

Internet traffic classification is about classifying unknown traffic into Internet applications or classes of Internet applications. For example, recognizing if the network traffic was Skype, Youtube, FTP, P2P, etc. [1]. Other traffic class categories are also possible, such as if traffic is normal or malicious [9]. The choice for deciding categories is huge and won't matter much when analyzing them.

There are many reasons that motivate classifying Internet traffic, such as network monitoring, network optimization, pricing decisions, intrusion detection, and energy saving. For example, for network operator it would be beneficial to know what kind of traffic flows through their network in order to support the network accordingly by prioritizing certain traffic before else.

There are several ways to distinguish applications from each other by analyzing the captured Internet traffic. We will now take a look at two traditional approaches and one modern state-of-art approach: port based approach, payload based approach, and statistical approach.

Traditional **port based approach** relies on the fact that certain applications use only certain ports registered by Internet Assigned Numbers Authority (IANA). For example, SSH using TCP port 23. Even though port based approach is simple and quick, this approach is nowadays unreliable because of the dynamic ports. Also, many P2P applications don't have standard port numbers. Moreover, currently many applications run on top of HTTP and HTTPS ports regardless its functionality. [1]

Another traditional approach is **payload based approach** which is also known as deep packet inspection (DPI). This approach checks the packet's payload content to see if it matches well known signatures, such as GET requests. This is very quite accurate method to determine Internet application. However, payload based approach is slow and expensive since they require maintaining huge database all the time. Moreover, they don't work at all in case of encrypted IP traffic (pretty much everything is nowadays encrypted) and also have privacy

issues. Also, many encrypted packets utilize packet obfuscation and randomized packet sizes in order to avoid any identification. [1]

Nowadays, flow-based **statistical approach** is used widely to classify Internet traffic. It relies on certain Internet traffic characteristics, such as flow length, packet size, and packet inter-arrival-time. The benefit of this approach is that these characteristics are independent from the payload information and can be unique for certain classes of applications [5]. Statistical approach uses **machine learning (ML)** as a tool to get the results via training and testing the available data.

3 Very basics of machine learning

Machine learning has become hot topic in most recent years. There are many reason for this, such as the availability of big data, the improvement of ML methods, and the improvement in technology. ML is great to deal with patterns, large datasets, and datasets with multi-dimensional features [5]. ML has wide range of applications in many fields. But what is ML exactly about?

Let's start from very simple. First, let's consider a traditional programming. It is about finding desired output Y when input X and function F is given. Unlike traditional programming, ML is about being able to find function F when input X and output Y are given. The said function F is found by training set of input/output pairs, i.e., mapping them together. Having suitable F is really useful to predict future outcomes. Now, using correct ML terms, the said function F is called as **model**, input X is called as **features**, and output Y is called as **target**. [1]

Feature (also known as attribute) can be any statistics (such as mean, median, standard deviation) which can be calculated directly based on the original dataset. Target (also known as label) is some desired thing that we want to examine. Model is then created by ML algorithm that maps features to their respective targets. Very important to note is that in order to get successful model, a lot of data will be needed. It is true that bad algorithm with large amount of data is able to produce better model than excellent algorithm with small amount of data. However, keep in mind that as dataset size increases, the model accuracy improves at the cost of more sampling time, storaging, and training time. [6]

So instead of programming the model ourselves, we let the ML process to create the model instead. This can be effective since ML process is able to model difficult tasks which would have been maybe even impossible to program ourselves. Now we are able to see the truth behind ML: it is actually pure statistical mathematics. In this course, we will not delve much into its mathematical background, but we will use ML as tool to analyze Internet traffic. Many programming languages have vast amount of tools to utilize machine learning. For example, **Scikit-learn** or **Tensorflow** (more deep learning focused library) for Python, and **Caret** for R language.

Generally, ML algorithm can be supervised or unsupervised. Let's talk about

them next.

3.1 Supervised learning

In **supervised learning**, the available data is labeled priori so the possible outcomes are known already [1]. For example, in case of Internet traffic, it could be labeled as application such as Youtube and Skype. **Classification** is about managing labeled data into their correct groups. Ideally, the training data contains traffic type that one wished to see later and the different traffic that might occur also in future [10]. It is important that the training set contains all possible instances so that no unknown instances will appear later during testing. Performance of supervised learning might be bad if the training data is skewed and not representative enough. One of the main challenges of supervised learning is that both training data and test data needs to be labeled before, which may require human expertise with the aid of DPI or pattern matching [1].

Supervised ML algorithms deals with different kinds of problems. In **regression problems**, we are usually dealing with continuous variables, i.e., finding most optimal numerical value as output. In **classification problems**, we deal with discrete set of variables, i.e., finding most optimal variable from the set. [1]

Most of the ML problems nowadays deal with supervised learning [7]. Regarding Internet traffic classification, we could utilize supervised ML to train a model that is able to map set of flows to their respective application. Each flow has same set of features (such as number of packets, packet inter-arrival-time). Usually the value of such features depends on the captured application but sometimes there can be two almost identical application, which is why the dataset should be large enough or number of features as high enough to counter this. Then we would be able to utilize the trained model to classify unknown network flows. [6]

3.2 Unsupervised learning

In **unsupervised learning**, the available data is unknown and therefore they are not labeled priori [1]. Unsupervised learning is about learning patterns in the dataset and **clustering** each different patterns their own classes. In other words, when speaking about unsupervised learning, we usually use clustering algorithms to deal with unlabeled dataset. During clustering, some instances may overlap to more than one category, in which case probability is used to determine their category. Labeling is required after this, which can be expensive and difficult to maintain and update. Another problem of unsupervised learning is that in practice, the amount of clusters tend to be more than the actual amount of classes so mapping a cluster to certain application can be then difficult [10]. Therefore, unsupervised learning is often more difficult to evaluate than supervised learning.

Usually many unsupervised algorithm require the number of clusters as hyperparameter. The closer the chosen hyperparameter is to the real number of

clusters, the better result will outcome. The number of clusters is usually chosen based on initial knowledge or assumption about the number of targets. Unsupervised learning algorithms tends to be more robust than supervised learning algorithms, i.e., the testing error is close to the training error [1].

4 Process of machine learning (Internet traffic classification)

On high level, the procedure of ML is same for all applications, regardless of the field of research or study. It usually contains some usual steps, such as getting dataset, pre-processing, training, testing, and evaluation.

Usually the most time-consuming phase is pre-processing which actually contains lots of things to do in order to make the available data as useful as possible. The actual training in practice require only couple lines of code with modern programming language, although the training time itself depends on the size of dataset.

4.1 Getting data

The data itself is the essential part of ML process. Without sufficient amount of data, it is not possible to create suitable models at all. Sometimes acquiring data is more easier said than done, and can come as costful. Some of the datasets are vital for company's product line which they will never share or sell for the public.

There are still many options for getting relevant data for ML purposes. Many organizations rely on their own datasets that they have collected over years, so being in the organization can give access to the dataset. Then there is also an option to purchase it from other organizations if they are willing to sell it. Collecting the data itself can often be costly for many reasons, such as respecting privacy concerns. There are also many datasets available publicly for free of charge in several sources. For example, Kaggle and UCI machine learning repository.

In case of Internet traffic analysis, there are some software for capturing packets to get data, such as **Wireshark** or **Tcpdump** (due to legalization of data collection, make sure you have permission for capturing!). However, the raw .pcap file is not quite suitable for ML model training since it alone doesn't contain many useful statistics for learning purposes. Therefore, usually the flow information is extracted from .pcap files by using other software such as **CoralReef** and **NetMate** to find out some useful features.

4.2 Pre-processing data

Most often pre-processing the data is the most time-consuming and complicated part of the ML process. It is very important to pre-process the data properly to save computation time later, and also to make the data suitable for further

analysis. There are many things to perform during pre-processing to make the raw dataset as 'clean' as possible, such as

- labeling the data
- sampling
- dealing with empty values (imputing)
- dealing with non-numerical values (categorization)
- standardization
- normalization
- feature selection
- feature extraction
- splitting dataset to training set and test set

4.2.1 Labeling the targets in dataset (supervised learning)

Supervised learning needs to have labeled dataset in order to create the model and evaluate its performance. This can be sometimes difficult to do perfectly since most of time human expertise is required, and the amount of dataset can be huge. Sometimes we have to do things in approximate ways in order to have proper label dataset to start with.

Regarding labeling in Internet traffic classification, usually we can't exactly label the captured applications perfectly from the .pcap files. One "heuristic" method to label particular Internet application is to capture it separately alone and later add an additional column that describes its label. However, this is still not perfect way to label things. For instance, in case of Youtube, in addition to the video contents, the website contains lots of other data due to the nature of the website. Another method to label the applications would be to use DPI as help. Most applications have specific signature on their packets but usually they are encrypted. Or easiest method would be to get some available pre-labeled Internet traffic dataset somewhere.

Remember that the Internet traffic can be labeled in many points of view, depending on what we are our goals. You could label the packets as application (such as Youtube, Netflix, Outlook) or you could label the packets as its application class (such as P2P, FTP, DNS). So, usually labeling it is kind of approximate in practice since its difficult to know priori information about all applications since often network operators don't know all applications running on the network [1].

4.2.2 Sampling

Usually in data science, the purpose of sampling is to pick up only few amount of data from vast amount of data in order to reduce computation process later. This also applies to ML process. In addition, sampling serves another purpose as well.

In order for the model learning properly from the training data, the training data should contain as diverse amount of targets as possible. Each targets should have equal amount of instances in order to avoid later bias in evaluation. In other words, the dataset should not be imbalanced for ML learning purposes [8]. Practically, this is unlikely to happen, so the dataset tend to be imbalanced naturally. For example, in case of Internet traffic, currently video traffic is most dominating Internet traffic. If training set would contain mostly video traffic, the model would heavily favor it when it is used to predict new unseen data. Therefore, sampling need to be done to keep the training set as balanced as possible. Let's next look different sampling methods to combat dataset's class imbalance: random sampling, undersampling, and oversampling. [1]

One simple way to sample the dataset is to choose same amount of instances from each target. Other option is to undersample the dataset, i.e., sampling the dataset so that each target have as same number of instances as the target that has fewest instances. Undersampling has a disadvantage that it actually reduces the amount of data further which can be problematic if the original dataset was already small. Another option is to oversample the dataset, i.e., sampling the dataset so that each target have the same number of instances as the target that has most instances. The disadvantage of oversampling is that some instances will be repeated in the dataset, which might lead to overfitting (more about overfitting later). Check out Figure 1 for easier understanding regarding undersampling and oversampling.

There also exist some feature selection algorithms (more about feature selection later) that are able to choose certain features in a way so that the imbalancedness of the dataset won't affect the result at all [11]. However, such algorithms are quite specific so it probably would be good idea to sample instead.

In Python and R, you can perform sampling with sample() function.

4.2.3 Dealing with empty values

Sometimes the available data contains some missing values. For example, if person's age is unknown, it is marked as NaN or as zero. Many machine learning algorithms are not able to deal with such empty and meaningless data properly. There are many options to deal with this. The simplest method would be to remove such columns and rows but then this will leave us reduced amount of data. Other option is to fill these empty values manually by our own, but then we risk for biased results (i.e., low accuracy results). Another option is to impute the missing values with a statistical value, such as average value of the particular column that the data belongs to. More advanced method would be

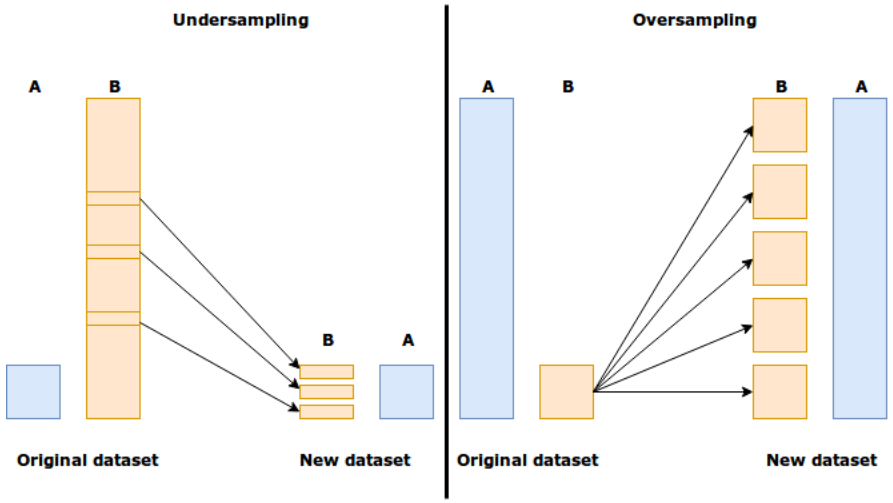

Figure 1: Illustration for undersampling and oversampling for the dataset with two different target values

to take into all features from the dataset and impute the missing values based on them.

In Python, you can impute the values with SimpleImputer() function or just drop such instances with dropna(). In R, you could use preProcess() function with imputing options, such as 'knnImpute'.

4.2.4 Dealing with non-numerical values

Most of ML algorithms are only able to deal with numerical values but usually the dataset contains some categorical values such as strings or dates. For example, scikit's ML library is not able to handle strings well at all.

To overcome this problem, they must be encoded somehow. One simple way would be to encode these values as number in increasing order. This simple labeling method is not suitable for every occasion though especially when dealing with categories that don't have "natural order". Let's imagine that we encode "red", "blue", and "yellow" as 0, 1, and 2, respectively. There might be few cases where some ML algorithm would calculate the average value of "red" and "yellow" as "blue", which can lead to wrong evaluations.

Other more advanced method would be to separate different non-numerical values into their own columns so that they will obtain binary values (see Figure 2). This is called as OneHotEncoding. However, this kind of method might increase the size of the dataset tremendeously, which might exceed the computation capability.

In Python, you can encode relevant categorical values with by manually or by using some pre-defined library functions, such as LabelEncoder and OneHotEncoder. In R, you could use dummy variables to perform OneHotEncoding

or use functions that can transform meaningful string into integers, such as ip_to_numeric().

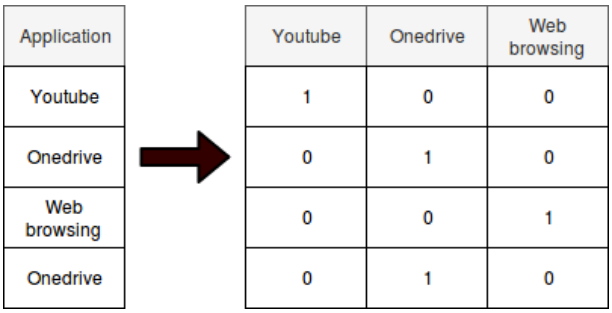

Figure 2: One hot encoding

4.2.5 Standardization

Standardization is about making every values in dataset having zero mean and an unit variance. This can be important because there might be a feature with very high variance or high value compared to other features. Therefore, it could dominate too much, making the learning impossible. In addition, some ML algorithms work more efficiently with standardized values. For example, if using linear support vector classifier (LinearSVC), the algorithm might not converge if the standardization was not done, leading to inaccurate results. In addition, KNN that is based on Euclidean distance, also would require standardization (although usually it is already internally implemented).

Furthermore, the standardization will improve the efficiency of gradient descent (GD), which many ML algorithms utilize. Standardization will affect the loss function of the dataset to be less skewed and more Gaussian-like instead of being skewed towards certain direction. This could be helpful for GD convergence. By having the loss function to be more Gaussian-like, each weight directions update in GD are more equal regarding the convergence steps (read more about loss function and GD in extra section if interested).

Typically in case of regression analysis, both features and targets need to be standardized before training. However, in case of classification problems, only features need to be standardized. This is good to keep in mind since some classification algorithms will not be able to deal with continuous target values. Usually, the scalers are created via training data only in order to keep it separate from the test set for more realistic evaluation in later phases.

Standardization could be done manually but in Python, StandarScaler() could be utilized for this as well. In R, preProcess() could be used with two options: "center" and "scale".

4.2.6 Normalization

Normalization is about scaling the values into column to specific range, such as range between of [0, 1]. The reason for normalization is to prevent the gradient descent algorithm changing too radically since the derivatives can be high in original data. In other words, normalization will ensure that the weight updates are not oscillating too much (see more about GD in extra section if interested). Normalization is especially required for deep learning models.

As in standardization, apply normalization to both features and targets in case of regression analysis. In case of classification, apply it to features only. Again, usually good idea to create scalers based on training data and then apply it to the rest of the data.

In Python, you could utilize MinMaxScaler() for scaling the values between specific range. In R, preProcess() could be used with "range" option.

4.2.7 Feature extraction & feature selection

In some sources, both **feature extraction** and **feature selection** are utilized as same meaning, but in some other sources they differ little bit.

Usually feature extraction is about deriving new features based on existing features, such as adding statistical values [1]. In case of Internet traffic features, suitable features could be protocol, mean packet length, packet inter-arrival time, etc. By using ML, it is possible to train the model to map features into different Internet applications. For example, average packet size in cloud storage and streaming categories are much larger than in messaging applications [3]. However, these statistical values are not clearly available in .pcap files. You need to write own scripts based on libpcap library which is available for many programming languages. You also have another options, for example, some other tools could be utilized to extract useful features, such as NetMate or CoralReef. Both of them extract flow information from the captured packet files. NetMate is able to extract 44 features whereas CoralReef is able to extract 11 features. You can also add even more features on top of them on your own when handling the dataset.

On the other hand, feature selection is about choosing only subset of the existing features [1]. The dataset itself might contain some very irrelevant features that do not help the ML learning process at all. Irrelevant features might even decrease the accuracy of the result, especially in the case of linear modeling. For example, some features are the ones that are completely useless (such as the name of person when classifying if person has a fever) or the ones that has constant value in all cases. Another example of useless features are the features that are strongly correlated with each other, i.e., correlation amongst features. This would not improve the model much and might actually worsen the model. We would want to remove either of them and keep more simple one (due to Occam's Razor) to reduce the feature space. Computation time will be decreased tremendeously as the number of features are reduced because the description of the instance will require less space to the matrix representation. Feature

selection might slightly decrease the accuracy of the created model, but usually the tradeoff between accuracy and computational complexity is excellent at the end of the day. Both accuracy and the training time are very relevant in case of real-time classification systems. [5]

Regarding feature selection in Internet traffic classification, some of the features can easily be reduced manually as humans since some features can be easily understood as totally irrelevant to the target value. However, sometimes there might exist some useful correlation between some features and targets which are not clearly visible. Statistical methods are used to make this effortless. In fact, some ML methods could be utilized here as well. There exist many different **feature selection algorithms** that could be utilized to automatize the feature selection process.

Feature selection algorithms could be divided into three categories: filter methods, wrapper methods, and embedded methods. Filter method (also known as information gain) selects the best subset of features based on their correlations without the need of ML. Wrapper method utilize ML to find out the best subsets via training and learning correlations. Embedded method is combination of both methods. Wrapper and embedded methods are computationally heavier so you should not consider them if you have about more than 20 different features. In addition, most recently, deep learning algorithms have been proven to successfully extract useful features without the need of humans. [1]

In Python, feature selection could be performed with SelecktKBest() function. In R, Recursive Feature Elimination (RFE) could be utilized by using rfe() function. You can also do feature selection by own custom function, for example, calculating the correlation of feature-target pairs and choose the most correlated ones.

4.2.8 Splitting the dataset to training set and test set

At some point, the dataset will be splitted into two groups: training set and test set. Training set is used for building the model by utilizing some ML algorithm. Test set is used later for evaluation purposes, it should be kind of absolutely invincible during training process. In addition, there can also exist validation set which purpose is to tune hyperparameter optimally (such as most suitable ML algorithm) but if the model architecture or hyperparameters are already decided or pre-determined, validation set is not necessary. [1]

In other words, the purpose of this train-test-splitting is to later quickly evaluate the performance of the model and also to avoid overfitting (more about this later). Having test set available is an inexpensive and efficient way to evaluate the model after the training, instead of immediately testing the model with new unseen dataset. For real final implementations after this evaluation and optimal parameter tuning, we would use whole dataset for re-training a new model. This will perform better with all available data than just subset of it.

The split ratio between training set and test set depends on the use case and amount of the data available. The goal is often to minimize the variance of the outcome but at the same time avoid overfitting.

During pre-processing, regarding any data transformations (such as standardization) which might get information from the test set (such as mean or variance): it is often good practice to only "build" this transformation based on training data only, and then apply the transformation to both sets. This way we prevent any "data leakage" for the training process, i.e., introducing any future information during training process. Test set should not be part of any model creation process, only training set is for that purpose.

Python and R also provide functions for splitting the instances to both sets if this is needed, such as train_test_split() and createDataPartition(), respectively.

4.3 Training the model

After the dataset has been pre-processed, the actual model creation will take place.

ML algorithm is applied to training set to fit a model to the dataset. There are many ML algorithms available, such as linear regression, naive Bayes, Support Vector Machine (SVM), decision trees, random forest, K-means clustering, K-nearest neighbor (KNN), and deep belief network. Every ML algorithms have different approach to sort and prioritize set of features [10]. Some ML algorithms are better suited than others, depending on the task we are aiming for. For example, some works poorly with small dataset but excellently with massive dataset. According to most recent studies, C4.5 decision tree has achieved best results regarding Internet traffic classification [13], [5], [11].

For understanding the ML model training process in low level, see the extra section at the end. In short, it is about finding optimal weight parameters that minimize loss function for the model with gradient descent algorithm.

In Python, model training is done with fit() function. In R, model training can be implemented with train() function. After the model have been created, it will be evaluated to see if it was suitable at all.

4.4 Evaluating the model

Now that the model exist, it is time to validate if it is actually suitable or not. This is usually done by utilizing the model on test set (which we can pretend to be 'new unseen dataset'), and then compare its predicted target results to real target values. It is very important that the model testing never happens on the same trained dataset because it would be meaningless result since we would be just using the model on dataset that it has already learned. The performance of the model usually depends on the size of the dataset, selection of features, and the used ML algorithm [6].

Widely used metrics for 'classification model' evaluation are accuracy, precision, recall, confusion matrix, and k-fold cross-validation. Let's take a look at them in next subsections.

		Predicted			
		Ant	**Bird**	**Cat**	**Dog**
Actual	**Ant**	4	0	1	0
	Bird	1	3	0	1
	Cat	0	1	3	1
	Dog	0	0	0	5

Table 1: Confusion matrix for classifying four types of animals.

4.4.1 Confusion matrix

Confusion matrix is one of the effective methods to see immediately quick facts regarding the performance of the model. It is important for traffic identification measurement, for example. Let's see a simple example about a confusion matrix at Table 1 and interpret it.

In this confusion matrix, the predicted values are displayed vertically, and the actual real values are displayed horizontally. This is how the scikit's confusion matrix method works by default (in R, this is vice-versa). We can interpret that in reality, there was in total of 5 ants, 5 birds, 5 cats, and 5 dogs in the dataset. However, the model predicted that there was 5 ants, 4 birds, 4 cats, and 7 dogs in the dataset. Looking in more detail, we can see that the model misclassified 1 bird as 1 ant, 1 cat as 1 bird, 1 ant as 1 cat, and both 1 bird and 1 cat as 2 dogs. In both Python and R, the confusion matrix can be achieved with confusion_matrix() and confusionMatrix() functions, respectively.

4.4.2 Accuracy, precision, and recall

In ML evaluation, there is concept of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TP means that the model labeled an actual positive instance as positive, TN means that the model labeled an actual negative instance as negative, FP means that the model labeled an actual negative instance as positive, and FN means that the model labeled an actual positive instance as negative (positive instance means that instance is X, and negative instance means that instance is not X) [1]. In other words, model is doing good if number of instances in TP or TN are high, and number of instances in FP or FN are low. Let's take a look at the confusion matrix below and interpret it.

Accuracy score is globally same for whole model, whereas the precision or recall depends on the target that are being examined. Accuracy of the model means the ratio between correctly labeled instances and all instances. However, be careful since accuracy alone might be misleading in case of, e.g., imbalanced dataset. The intuitive meanings of precision and recall (also known as sensitivity) can be vague to grasp since they are calculated for each targets separately. But all you need to know that the value of these three metrics mentioned range from 0-1. The bigger value, the better model. The following formulas show the

		Predicted			
		Ant	**Bird**	**Cat**	**Dog**
Actual	**Ant**	TN	TN	FP	TN
	Bird	TN	TN	FP	TN
	Cat	FN	FN	TP	FN
	Dog	TN	TN	FP	TN

Table 2: Similar confusion matrix as earlier, but now aim is to calculate metrics from cat's perspective.

mathematical expression for the three metrics.

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

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

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

Let's take an example to calculate some metrics for the earlier model that classified four types of animals. We are especially focusing on cat's perspective when considering the quantities for TP, TF, FP, and FN. Let's first consider how many TP instances there are. The model predicted 3 actual cats correctly so there are in total of 3 TP instances. The model misclassified 1 ant as 1 cat, and none of the birds nor dogs were mistook as cats, so there is in total of 1 FP instance. The model misclassified two actual cats as something else, so there is in total of 2 FN instances. The rest numbers are TN instances since the model correctly labeled them as 'non-cats', so in total there are 4+1+3+1+5=14 TN instances. From the Table 2 we can intuitively see which cells belong to which instance in case of cat's perspective.

Therefore, the accuracy of the model would be $\frac{TP+TN}{TP+TN+FP+FN} = \frac{3+14}{3+14+1+2} = 0.85$, precision of the cat would be $\frac{TP}{TP+FP} = \frac{3}{3+1} = 0.75$, and recall score of the cat would be $\frac{TP}{TP+FN} = \frac{3}{3+2} = 0.60$. Notice that only accuracy score alone can describe the model in general, the other metrics takes certain target value's perspective.

In scikit library, all of these scores can be found with classification_report() method. In R, the you can calculate each metrics with separate functions.

4.4.3 K-fold cross-validation

Getting single result from this one particular test is not absolutely reliable. What if the test set happened to be very easy to be predicted? Or what if test set was difficult instead? This is why K-fold cross-validation, which is one of the cross-validation methods, would be good option for evaluating the whole

modeling process. K-fold cross-validation is very cost-efficient way to evaluate the created ML model. This is also very good and valid way to compare different kind of possibilities together and find the best among them. For example, sometimes we have lots of different ML techniques to train the model such as KNN, random forest, and naive Bayes. Or sometimes we have certain model but unsure which hyperparameter would be the most optimal, such as number of neighbors for KNN algorithm. Utilizing k-fold cross validation, we can quickly find such hyperparameters. [4]

Let's consider a dataset which is randomly splitted to k number same sized sets. One of them will be test dataset and the rest will be the training dataset. The model is created via training set, and then utilizing the test set, the accuracy score is calculated. After each scoring, the training set and test set are shifted in round-robin fashion and then second scoring for the second takes place. This process is repeated until each test set have been tested (see Figure 3). Finally, an average score could be taken to evaluate the model. This score is kind of "honest" since it ensures that the evaluation did not depend on specific training set or test set. Usually a decent parameter for k-fold cross-validation would be k=10.

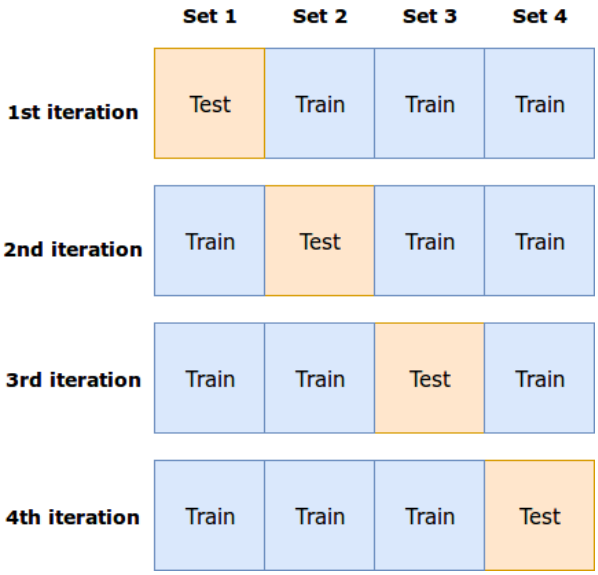

Figure 3: K-fold cross-validation, with parameter k=4

Remember, that K-fold cross-validation is only used for model evaluation purposes (i.e., finding suitable hyperparameters), it does not build any model.

4.4.4 Avoid underfitting and overfitting

Underfitting means that the model did not work well with training data, i.e., the resulted accuracy score was low [1]. Since training score was low, it means that the model is not suitable for estimating the results. In order to avoid underfitting, usually increasing dataset or using different ML algorithm would fix it.

Overfitting is one of the fundamental problems regarding ML (and also deep learning). It means that the created ML model fits too well to the given training dataset but might not actually fit well to new unseen data [2]. In other words, the model does not generalize the situation well because it 'memorizes' the training data too well. One method to detect overfitting is to compare the resulted training error and testing error. If testing error is significantly bigger than training error, overfitting happened most likely.

Overfitting only depends on the complexity of the model, if it is too complex then overfitting will happen. To see if overfitting exist, k-fold cross-validation could be utilized or checking model's training history if validation score becomes too different to training score. To combat against overfitting, the dataset could be collected at different times and measurement points, early stopping coulb be performed, or regularization could be performed which is modifying loss function to ignore certain high-ordered terms (more about loss function in Extra material section).

See the Figure 4 for their intuition meaning.

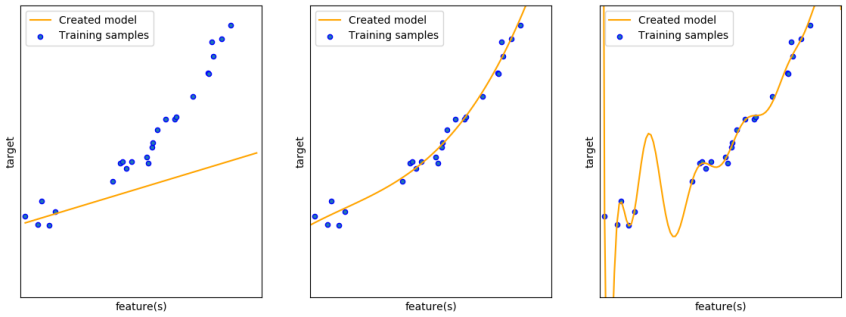

Figure 4: Left: Underfitted model. Middle: "Suitable" model. Right: Overfitted model.

5 Extra material

So far we have understood in high level what the ML is about: utilizing training set of feature/target pairs to create a model. But how the model creation actually work in low level details?

This extra section will introduce you to the concept of loss function to minimize the error of model performance, and gradient descent algorithm to find suitable parameters for minimizing loss function. Having basic understanding of calculus and linear algebra can be helpful for this section. This is all extra material but they can be useful to understand for later deep learning section.

We will now consider very simple ML problem where we create suitable linear model for the dataset with only one feature, based on a few training samples. See Fig 5. The created linear model would be expressed as $h(x) = w_0 + w_1 x$, where both w_0 and w_1 are weight parameters, x is the single feature of the dataset, and $h(x)$ is called as **hypothesis** which in ML field means a candidate model for mapping inputs to the outputs. In other words, we aim to approximate such candidate model as well as possible. We want to find out the best weight parameters for this model somehow so that it behaves according to the training samples and, therefore, is able to generalize the situation.

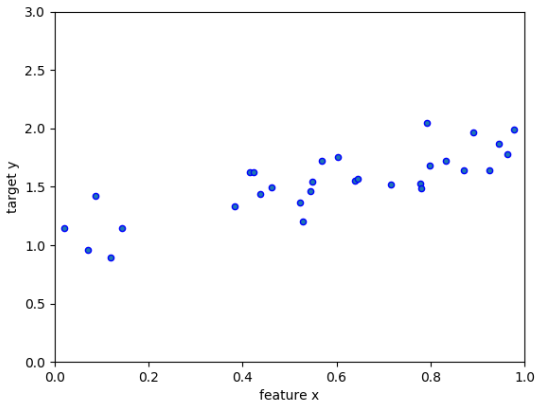

Figure 5: Training samples of single-featured dataset, and the aim is to model it linearly

5.1 Loss function

What kind of weight parameters should we choose so that model suits best for the previous example? Answer: we want to choose such parameters so that the error between the predicted result and real actual result is low as possible.

Loss function $J(w_0, w_1)$ (also known as cost function) express the amount of error between the predicted result and real actual result [12] . The exist many kind of loss functions, such as hinge loss, mean absolute error (MAE), mean squared error (MSE), and mean bias error (MBE). One of the most popular loss function for regression problems is MSE where the errors are squared and then normalized. For classification problems, cross entropy losses is often used.

Each of the loss function look different. The following Figure 6 illustrates loss function that would represent the previous linear regression example.

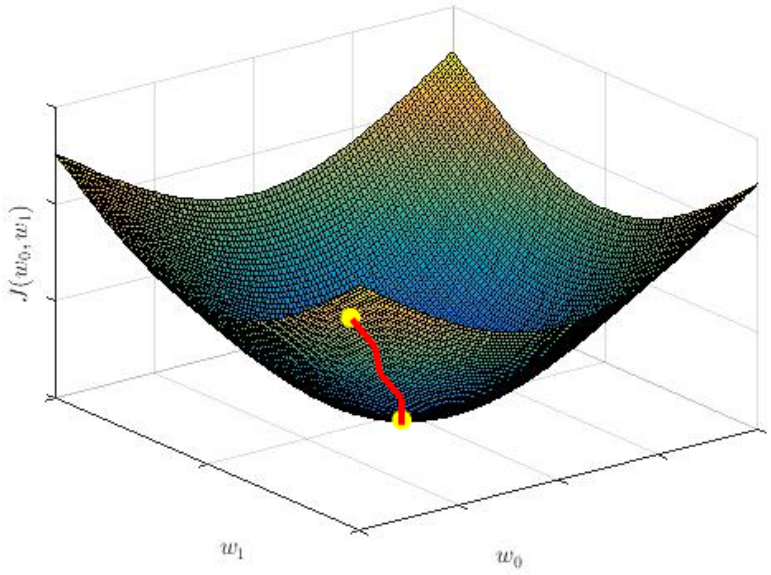

Figure 6: Loss function $J(w_0, w_1)$ minimization with gradient descent

The aim is to minimize the error, so we need to minimize the loss function by finding its best weight parameters. Minimizing loss function could be expressed as following (in case of MSE):

$$\min_{w_0,w_1} J(w_0, w_1) = \min_{w_0,w_1} \{\frac{1}{n} \sum^{n} (h(x) - y)^2\}$$

The intuitive meaning of the formula is to find weight parameters w_0 and w_1 so that the cost function $J(w_0, w_1)$ is minimized. But now another question arises: how to find such weight parameters? The most used algorithm is gradient descent (GD).

5.2 Gradient descent algorithm

Gradient descent algorithm is one of the widely used algorithms in many problems, including the ML problems. It is often used algorithm to find optimal weight parameters for the loss function in order to minimize it [12] . Intuitive way how it works: first it will pick a random starting point at the loss function. Then it will step towards the direction that has steepest curve to the minimum point of loss function. This iteration is repeated until the (local) minimum has

been reached. The formula of GD algorithm for the previous loss function can be expressed as:

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} J(w_0, w_1)$$

The parameter α in the formula is learning rate (i.e., "step size") and the $\frac{\partial}{\partial w_i} J(w_0, w_1)$ is partial derivative for certain weight to express the next steepest direction towards to (local) minimum. The learning rate α is adjustable. If the learning rate α is too small, the steps will be small and algorithm will converge slowly (but surely) to the optimal point. If the learning rate α is too big, the steps will be huge and algorithm might converge quickly or it might never converge at all since it can oscillate forever.

In case of loss function in Figure 6, the GD algorithm would converge easily since the loss function is convex, and starting from any point will always reach to same minimum destination. Therefore, GD can handle such single-featured linear regression problems easily.

However, generally in case of non-convex loss functions (something that resembles the one in Figure 7), they are harder to optimize. The GD algorithm might converge in local minimum instead of global minimum, or it might get stuck on 'saddle points', depending on its starting point. But in practice these problems do not matter since local minimum is often good enough [7]. There are also other algorithms than GD for finding best weight parameters, such as conjugate GD, BFGS, and L-BFGS. They are usually more efficient for certain kind of models but much more complicated to understand intuitively and to implement. Some of the algorithms are less greedy, i.e., they look more around the minimum they achieved to find better minimum, but the drawback is the increased computation time.

So in short, we will first have some sort of 'candidate model' which we will feed with data (input-output pairs). As the time goes on, the weights of the 'candidate model' will be adjusted until it optimally can model the training set with minimal errors. Next step would be the model evaluation for test data or real unseen data.

5.3 Multiple features, logistic regression

So far, we went through linear regression (with single feature) example in order to understand how the ML modeling works at low level: At first, some kind of model expression (i.e., function) is assumed. Next the optimal parameters for such model need to be found so that the resulted loss function is minimized. Such optimal parameters are found with GD algorithm, and then we will have the model fully created.

Modeling such dataset containing single feature were simple to illustrate via graphs. However, usually in ML we are dealing with dataset containing multiple different features. In case of linear regression for multiple features x, it could be expressed as following:

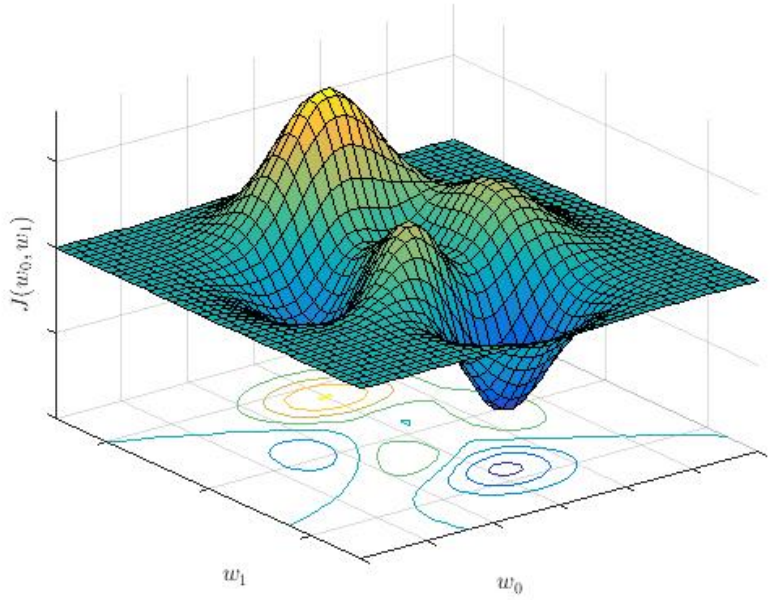

Figure 7: Loss function $J(w_0, w_1)$ (non-convex)

$$h(x) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

In this case, loss function are much harder illustrate via graphs because the overall error is calculated using many dimensions. However, GD algorithm could still be used to find the optimal weight values. Therefore, the model creation process is still same. The non-linear decision boundaries can be achieved with higher order hypothesis. For example, in the Figure 8 we have two features available, and the suitable model would be something like $h = f(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2)$.

5.4 Logistic regression and multiclass classification

Our example was about linear regression which outputs continuous real values, so it deals with regression problems. In case of classification problems, we could use **logistic regression** (despite its name, it actually deals with binary classification problems). Since we are dealing with classification instead of regression, a **decision boundary** will be implemented to classify each values to their own category. Logistic regression's hypothesis is almost similar to linear regression, but in addition, the final value will be scaled to specific range with **activation function** (such as sigmoid function $f(z) = \frac{1}{1+e^{-z}}$). So, the hypothesis of logis-

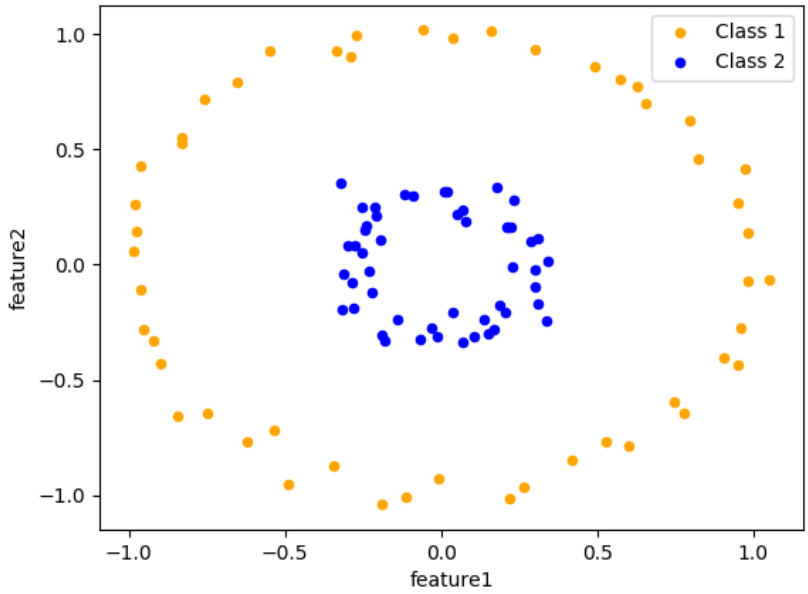

Figure 8: Two target classes distributed in such way that non-linear modeling is required

tic regression would be $h(x) = f(w_0 + w_1 x) = \frac{1}{1+e^{-(w_0+w_1 x)}}$. The output being in specific range and the decision boundary (created by the weight parameters in hypothesis) will then determine how the instance will be classified.

In case of multiclass classification (e.g., recognizing handwriting digits, recognizing type of Internet traffic), the problem is actually solved by breaking it into multiple binary classification problems. Therefore, we will get multiple hypothesis functions. We will see that one of the models seem to maximize its value, so we then classify such instance to the class which the model belongs to. In case of Internet traffic classification, it is good to note that as the number of applications grow, the classification accuracy will decrease since it is more likely that the new application features might be more similar to the existing ones [6]. For example, it is easier to distinguish Youtube and OneDrive traffic compared to distinguishing Youtube and Vimeo traffic.

5.5 Classification and regression metrics

To further clarify some metrics used in different situations, next we will list some common metrics used in both cases.

In case of classification (i.e., dealing with discrete target values), usual metrics are accuracy, recall, precision, confusion matrix, logistic regression, and

categorical cross-entropy. [1]

In case of regression (i.e., dealing with continuous target values), usual metrics are MAE, MSE, and RMSE. [1]

5.6 ML algorithms for model training

Each ML algorithms have its own mathematical method to train a model. They all have strengths and weaknesses, for example, some algorithms work better for bigger sized data and some algorithms can train model faster. There is no 'best' ML algorithm for every situation, which is the reason to try many different ones to find most suitable for particular case. In general regarding Internet traffic classification, it has been often concluded that C4.5 decision tree would be one of top choices.

Let's now take a look at few ML algorithms and how they work at high level: support vector machines (SVM) and k-nearest neighbor (KNN).

5.6.1 SVM

SVM models are type of models that try to estimate a hyperplane (which dimension is one less that whole dataset) that could separate different instances from each other as best as possible. Using different kinds of kernel tricks, the hyperplane can become non-linear. SVM algorithms can be used both for regression or classification tasks.

In case of linear SVM, it assumes that the created model's inputs and outputs have linear relationship. The created model could be easily used for classification problems. Let's have a look on simple example where we have a classification problem. We have to decide if certain point in the space should be classified as 'orange' or as 'blue' based on their two features (X-coordinate and Y-coordinate). We are given in total of 100 training samples and we are training the model with LinearSVC algorithm (see Figure 9). As a result, the model will be able to solve the problem in linearly as we can see from the picture. Values that are on the orange area (below the decision line) will be classified as 'orange' and values on the blue area (above decision boundary) will be classified as 'blue'. The linear regression algorithms are stable (i.e., not likely overfitted) but sometimes inaccurate.

5.6.2 KNN

KNN algorithm doesn't make any assumption regarding the relationship between model's inputs or outputs. All that KNN cares about is the Euklidean distance from training sample to other training sample. Depending on its hyperparameter's, KNN is able to classify if certain area belongs to certain target by taking into account the k nearest neighbors. In case of 2-class classifications, usually the hyperparameter is odd-valued in order to avoid tie situations, in which the coinflip will decide the result. Simple example at Figure 10 demonstrates that the unknown point belongs to 'blue' because most of its three nearest

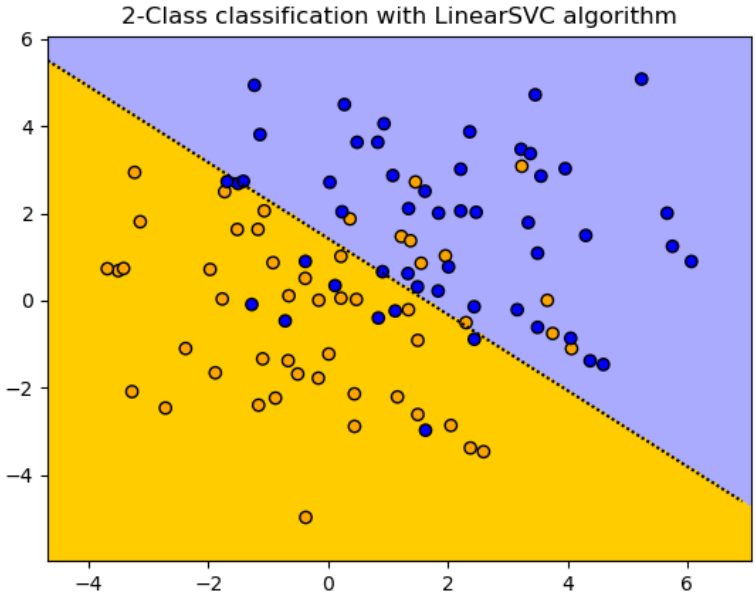

Figure 9: Dataspace with orange (gaussian, mean=0, var=2) and blue (gaussian, mean=2, var=2) training samples classified with LinearSVC algorithm

neighbors were 'blue' (2 were 'blue', 1 was 'orange').

KNN can be accurate but sometimes unstable (i.e., overfitting. So, avoid using k=1 at least). Nearest neighbor modeling might struggle if there are lot of features because the algorithms need to calculate each neighbors distance. Therefore, search trees could be used for finding neighbors efficiently. Figure 11 demonstrates how KNN have created the classification decision for all possible areas. KNN is very scalable and it works well with any sized dataset.

References

[1] Raouf Boutaba et al. "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities". In: *Journal of Internet Services and Applications* 9.1 (2018), p. 16.

[2] Davide Chicco. "Ten quick tips for machine learning in computational biology". In: *BioData mining* 10.1 (2017), p. 35.

[3] Sina Fathi-Kazerooni, Yagiz Kaymak, and Roberto Rojas-Cessa. "Tracking User Application Activity by using Machine Learning Techniques on Network Traffic". In: *2019 International Conference on Artificial Intelli-*

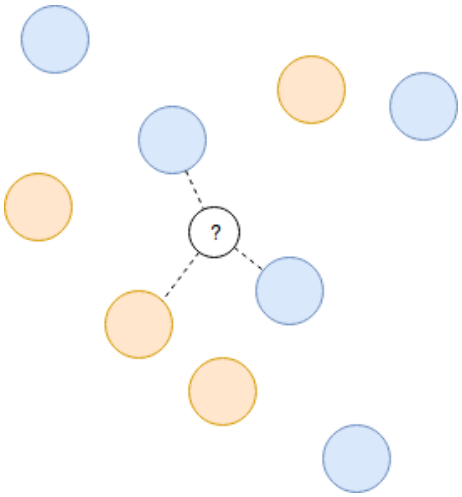

Figure 10: KNN classification with k=3 would result unknown point to be 'blue'. What would have the unknown point been classified, if k=5?

gence in Information and Communication (ICAIIC). IEEE. 2019, pp. 405–410.

[4] Tadayoshi Fushiki. "Estimation of prediction error by using K-fold cross-validation". In: *Statistics and Computing* 21.2 (2011), pp. 137–146.

[5] Rupesh Chandrakant Jaiswal and Shashikant D Lokhande. "Machine learning based internet traffic recognition with statistical approach". In: *2013 Annual IEEE India Conference (INDICON)*. IEEE. 2013, pp. 1–6.

[6] Li Jun et al. "Internet traffic classification using machine learning". In: *2007 Second International Conference on Communications and Networking in China*. IEEE. 2007, pp. 239–243.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[8] Naoto Mizumura et al. "Smartphone Application Usage Prediction Using Cellular Network Traffic". In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2018, pp. 753–758.

[9] Andrew W Moore and Denis Zuev. "Internet traffic classification using bayesian analysis techniques". In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 33. 1. ACM. 2005, pp. 50–60.

[10] Thuy TT Nguyen and Grenville J Armitage. "A survey of techniques for internet traffic classification using machine learning." In: *IEEE Communications Surveys and Tutorials* 10.1-4 (2008), pp. 56–76.

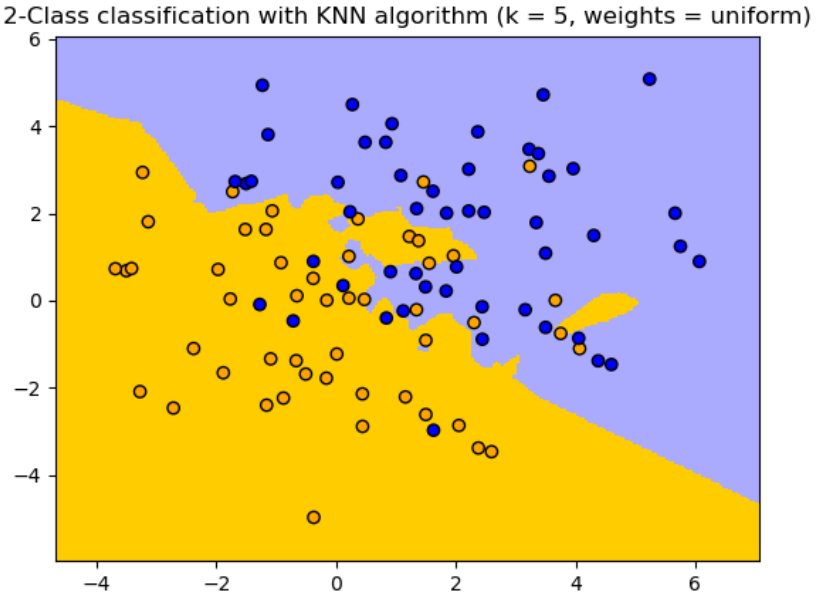

Figure 11: Dataspace with the similar training samples now classified with KNN algorithm

[11] Muhammad Shafiq et al. "A machine learning approach for feature selection traffic classification using security analysis". In: *The Journal of Supercomputing* 74.10 (2018), pp. 4867–4892.

[12] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. "Applying deep learning approaches for network traffic prediction". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2017, pp. 2353–2358.

[13] Nigel Williams, Sebastian Zander, and Grenville Armitage. "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification". In: *ACM SIGCOMM Computer Communication Review* 36.5 (2006), pp. 5–16.