**Applied Machine Learning**

**Group Practical Project**

**IoT Intrusion Detection Competition using Machine Learning**

**Alberto Matuozzo, David Kirby, Adam Elshimi, Ali Sahin and Andrew Robinson (13122132)**

**Student Numbers**

**Introduction**

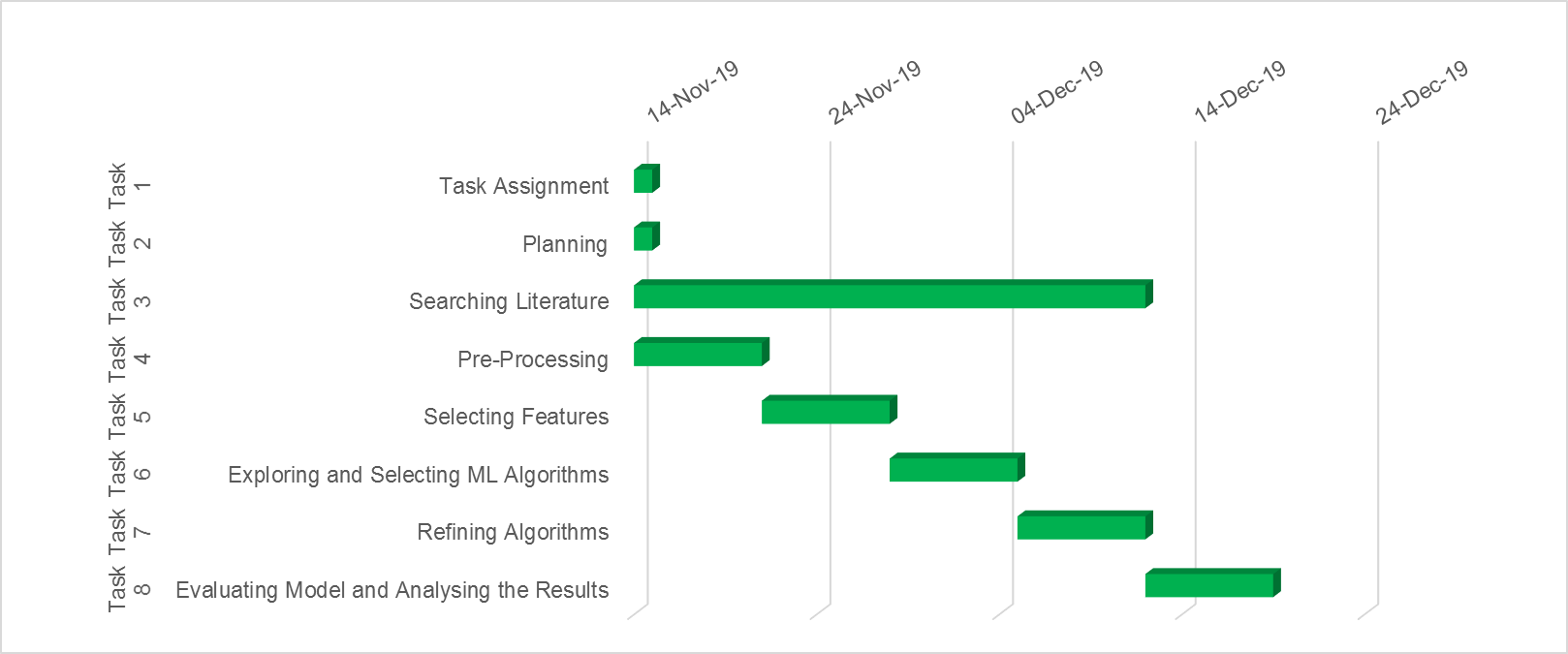
The Internet of Things (IoT) is a paradigm that involves networked physical objects with embedded technologies to collect, communicate, sense and interact with the external environment through wireless or wired connections. With rapid advancements in IoT technology, the number of IoT devices is expected to surpass 50 billion by 2020, which has also drawn the attention of attackers who seek to exploit the merits of this new technology for their own benefits [1]. The aim of this project is to address the issue of security threats in IoT devices by building a predictive machine learning model to distinguish between “intrusive” traffic, called intrusions or attacks, and “good” normal traffic. The predictive machine learning model will focus on detecting impersonation attacks in a reduced CLS portion of the Aegean Wi-Fi Intrusion Dataset (AWID) which was originally prepared and managed by George Mason University and the University of the Aegean.

**Planning**

Given the tight timeline of the project, the team agreed it was imperative to outline the project plan as quickly as possible. As such, a total of 8 tasks were defined and assigned to the members of the team. A deadline of 12th December was set for all literature research and practical programming to be completed, the idea being to put some time aside for fine-tuning the machine learning models and writing up each task before the project deadline.

The 8 tasks were assigned as follows:

* Task Assignment – Group Discussion
* Planning – Group Discussion
* Searching Literature – Each group member to read around their assigned task and record references.
* Pre-Processing – Alberto Matuozzo
* Selecting Features – David Kirby
* Exploring and Selecting Machine Learning (ML) Algorithms – Adam Elshimi
* Refining Algorithms – Ali Sahin
* Evaluating and Analysisng the Results - Andrew Robinson

In addition to the above it was agreed that GitHub would be used to develop and store our project pipeline and other documents and that all code would be written using Juypter notebooks. Furthermore, the below Gantt Chart was put togther to provide an overview of the project timeline:

**Pre-Processing - Alberto Matuozzo**

The aim of the pre processing stage is to analyse the raw dataset and make the necessary modifications to put it in the most effective conditions to perform the actual machine learning task.

Pre processing steps can be grouped into data cleaning, data reduction and data transformation.

Data Cleaning

As shown by Little and Rubin[2], It’s important to identify the presence of missing data and implement an appropriate policy to solve the issue. In this case the training set is complete.

The presence of duplicate cases can create a bias in the dataset causing the algorithm to learn improperly the persistence of a pattern. McKinney [3] highlights two options to deal with duplicates: keep them or discard them assuming they are redundant. The training set has 547 duplicate observations. Our assumption is that intrusion attacks are independent therefore duplication in this case is considered as persistence of certain patterns. Therefore we decided to keep the duplicates.

**figure 1**. Training set. Features Matrix descriptive statistics.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Min** | **Max** | **Std\_Dev** | **Skew** |
| Min | 0.000000 | 0.000000 | 0.000000 | -67.957971 |
| Mean | 0.016447 | 0.438865 | 0.055441 | 13.283325 |
| Max | 1.000000 | 1.000000 | 0.473930 | 311.518859 |

**figure 2**. Training set. Features Standard deviation: top 5 value counts.

|  |  |
| --- | --- |
| **Std\_Dev** | **Nr Features** |
| 0.000000 | 74 |
| 0.014709 | 11 |
| 0.354444 | 2 |
| 0.003210 | 2 |
| 0.062765 | 2 |

Data Reduction

This step aims to simplify the dataset eliminating features that are irrelevant at best and constitute noise at worst. Effective data reduction results in greater learning speed and potentially better accuracy.

Inspecting the standard deviation of the features matrix highlighted a minimum of zero (figure 1). Looking at the value counts for each level of standard deviation (figure 2) revealed that 74 predictors assume the exact identical value across every training instance. They are redundant, therefore have been eliminated using the *Scikit Learn VarianceThreshold* selector. The resulting dataset has been named *Xselected.*

Data Transformation

Different algorithms make an assumption about the probability distribution of data and or work best when it comes to optimizing parameters if input features are within a certain range.

Given that there isn’t an algorithm that performs better than others a priori on a given task, the output of the data transformation phase consists of several datasets to experiment on.

The predictors are heterogeneous: some variables appear discrete, others binary, others continuous. As suggested in Kolias *et al.* [4] and in Aminanto *et al*. [5] a normalization step has been performed so that every predictor’s value range is between zero and one. *Scikit Learn* *MinMaxScaler* has been applied to the *Xselected* dataset.

Using skew as a measure of symmetry around the mean of each feature, highlights that the respective frequency distributions deviate substantially from the Gaussian (figure1). Kolias et al. [4] showed that the Naive Bayes Algorithm was the most accurate (albeit not to a satisfactory level) in detecting impersonation attack. This algorithm, when dealing with continuous features, assumes a Gaussian distribution of the predictor given a class.

Yeo and Johnson [6] proposed a power transformation to reduce skew that does not require specific underlying assumptions on the variable to be transformed. This power transformation has been implemented on the *Xselected* dataset using *Scikit Learn PowerTransformer.* As a result the parameters of each feature distribution are closer to a Gaussian.

A final and alternative transformation has been performed on the *Xselected* dataset. Given the heterogeneity of features, it’s useful to scale each instance to have a norm of one in case at a later stage of the project an algorithm like Neural Network that weights input values, or that uses kernels (for example an SVM) to quantify the similarity of different examples will be deployed. This transformation has been implemented using *Scikit Learn Normalizer*.

At the end of the pre processing phase, four datasets are available to experiment with different feature and model selections:

1. X.selected: features with variance greater than zero
2. NX3df: Xselected with features value in range [0,1]
3. PX2df: Xselected with a power Gaussian tranformation
4. NX2df: Xselected with each training instance to have unit norm.

The best results have been achieved just removing the features with zero variance.

**Selecting Features - David Kirby**

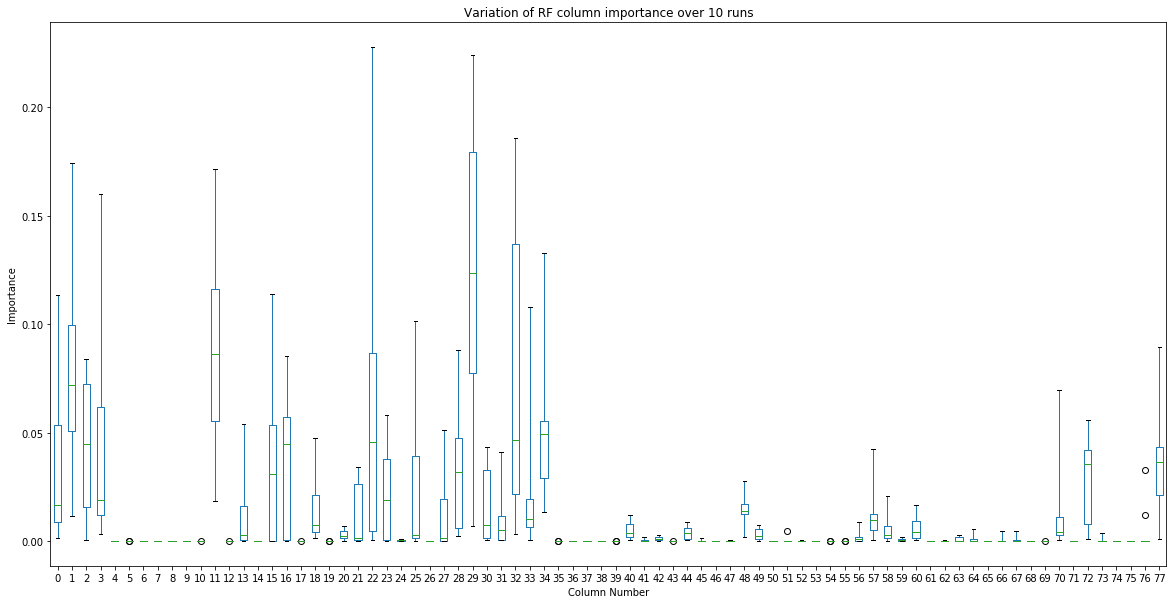
Feature Selection

The purpose of feature selection is to reduce the dimensionality of the problem space by eliminating features from the model that are redundant or unnecessary for prediction. This can improve accuracy if the eliminated features are irrelevant to the model or are strongly correlated to other features. Feature selection can also speed up training and prediction times and make the model more interpretable.

Embedded Feature Selection Via Tree Based Algorithms

The initial method chosen for feature selection was to use tree based supervised learning algorithms to identify the most relevant features. Three algorithms were selected for comparison: Random Forest [7], XGBoost [8] and CatBoost [9]. As explained in the previous section columns with zero variance had been removed leaving 78 features as a starting point for further elimination.

Each model was fit to the training data and their feature importance ratings compared to see which columns they had in common. However it was observed that the ratings given by Random Forest were not stable - repeatedly fitting to the same data set gave substantially different results each time. Figure 3 shows a boxplot for each feature with the variation in feature importance assigned by Random Forest over 10 runs. Both XGBoost and CatBoost were consistent in their importance measures. Due to this variability Random Forest was dropped as a means of selecting features.

  
**figure 3**

The top twenty features selected by the remaining algorithms were (going from lowest importance to highest):

xgboost: [31, 25, 28, 52, 32, 49, 13, 44, 20, 56, 18, 0, 72, 2, 48, 11, 34, 57, 29, 22]  
catboost: [77, 49, 31, 20, 70, 30, 1, 0, 33, 32, 72, 18, 48, 2, 3, 34, 57, 29, 22, 11]

N.B. These are the column indexes after the zero-variance columns have been removed. These both accounted for over 99% of the importance values assigned by the respective algorithms.

The intersection of these gave a set of 14 features that both XGBoost and CatBoost considered important:

XGBoost and CatBoost Intersection: {0, 32, 2, 34, 72, 11, 48, 49, 18, 20, 22, 57, 29, 31}

To validate the usefulness of these features 5-fold cross validation on the training set was performed with both XGBoost and CatBoost using the set of 14 features. The results were promising, with accuracies upwards of 99.96% and for some folds 100%. The number of features was further reduced to six using the RFECV (Recursive Feature Elimination with Cross Validation [10]) and the CatBoost algorithm. However although this still gave a cross-validation accuracy in excess of 99.98%, when the model was used to predict the test data set the accuracy dropped to 53-54%, barely better than chance.

Investigation into the cause of the poor accuracy against the test data set found that some features in the test data were radically different from the same features in the training data. The data had been pre-processed by a third party before we received it to scale all the features into the range 0 to 1. It appears that for some columns this had been done incorrectly so the distributions in the test data set did not match the distributions in the training data set. Any models that depended on these erroneous features would behave poorly against the test data. See appendix I for further information.

Simple feature selection using SelectKBest, Chi-squared and ANOVA metrics

It was decided that selectively filtering out the mismatched columns between the training and test sets would be liable to overfitting the model to the test data, so an alternative approach was taken of using a simpler method for feature selection and using a wider range of features. It was found that selecting 20-40 features by doing a Chi-squared or ANOVA comparison with the target data and selecting the K best gave good results for some algorithms. Chi-squared was tried initially but does not work with the negative values created by the power transformation, so ANOVA was added as an alternative.

The exact number of features to select and which metric to use was left as part of the model tuning stage of the process.

**Exploring and Selecting Machine Learning Algorithms - Adam Elshimi**

**Introduction**

After having preprocessed the dataset, used feature selection to reduce our dataset, categorised the problem and understood the data, we identified the algorithms that are applicable and practical to implement. Furthermore, before having set out to select the correct algorithm we chose the following 3 measurements to be the bases of our evaluation and point of comparison of other algorithms:

1. The accuracy of the model.
2. Time taken to make predictions using the model
3. Time taken to build, train, and test the model

**ML Algorithms**

Having set the groundwork with the phase1 and 2 the algorithm we decided to use as a baseline for our machine learning algorithm phase, was Xgboost, which in the paper [11] outline the benefits over many other machine learning models.

Xgboost is a greedy gradient boosting algorithm, which is a decision-tree ensemble method that performs well on structured data. A common issue that arises with tree learning is finding the best split that leads to optimal results indicated by equation in [12]. Xgboost tackle this problem by using a depth-first approach, which is a tree pruning method that has shown to significantly increase performance whilst reducing the time complexity of the algorithm [12].

The following algorithms Catboost, logistic regression, naïve Bayes, Random Forest and decision tree classifier (Adaboost) are the five extra algorithms we decided test.

Catboost is a unbiased gradient boosting algorithm having shown to outperform many of the latest and best algorithms [13], this was a step up from Xgboost, however, it took a significant amount longer for making predictions. Logistic regression, Random Forest, and naïve Bayes have previously been proven to work well with AWID dataset, as shown in the paper by Dr Paul and his counterparts [14] where tree learning and simple algorithms like logistic regression proved to be both interpretable and predict with high accuracy

**Classifier**

Boosting is generally a technique that derives from Probably Approximately Correct (PAC) models – a theoretical learning framework with the aim to show that an algorithm achieves low empirical risk (ER) error with high probability. However, for this model we are adopting a more general learning framework where we use Adaboost – an ensemble boosting algorithm created by Yoav Freund and Robert Schapire in 1996 [15] to tackle the problem of overfitting, increase accuracy through combination of weak learners and an algorithm which is simple to implement.

The formula for classifying with Adaboost is the following:

A picture containing object

Description automatically generated

Figure 4

Referencing figure4, f(x) is the function representing the mth classifier and theta is tha mth corresponding weight. Thus, this formula is the combination of all the weights of all mth classifiers (ensemble method).

After thorough investigation the Adaboost classifier was chosen as a the classifier for our model, which is evaluated in the evaluation section.

**Refining Algorithms - Ali Sahin**

Model hyperparameters are parameters that cannot be derived from the data and therefore must be specified before utilising the model. Default parameters are initially used at the beginning of the machine learning process and are adjusted after model selection. The aim of optimisation is to find a set of parameters which minimise a predefined loss function; exponential loss in adaboost and log loss in logistic regression [16].

Figure 5: Formularisation of hyperparameter search. Where represents the hyperparameters, X (te) represents the test data, represents the training data [1].

Figure 5 shows the mathematical representation of hyperparameter search. There are many algorithms which automate this process. The most commonly used are grid search and Bayesian search. These algorithms are preferential to the alternative of manually adjusting the parameters, as research shows this results in models with greater predictive performance [17]. Bayesian search was not used due to the time constraints. The sklearn library at the time of conducting the project, did not have a dedicated class to utilise the algorithm and therefore would have required significant time to code. Hyperparameters were evaluated using a tenfold cross validation (cv). Initially kbest was included in the grid search but then removed. Including kbest resulted in poor predictive performance on the test data for the adaboost models, accuracy decreased from 99.07% to 50%. This was the result of the adaboost models overfitting the training data.

For the final three pipelines, the parameters which were tuned are as follows: number of estimators, learning rate, ‘C’ and solver. The number of estimators (N) and learning rate (L) are a parameter of adaboost. There is a trade-off between N and L, the optimal number of estimators required increases as L decreases [18]. The number of estimators is the number of models that are iteratively trained: the values searched were 10,50,100,150. The learning rate is the contribution each model has on the weights of the algorithm. The values 0.001,0.01,0.1 were searched, as past studies suggest values below 0.1 are optimal [18]. The lowest L values should have the greatest cross validation score as a result of reduced overfitting however, the opposite was true. The highest L achieved the best cv score. The parameters Solver and ‘C’ are used to optimise logistic regression. The inverse of regularisation strength is referred to as ‘C’. The lower the ‘C’ the greater the regularisation strength. This reduces the variance of the model and therefore should reduce overfitting however, the highest C values resulted in the greatest cv score [19].

**Evaluating Model and Analysing the Results – Andrew Robinson**

After selecting and refining numerous algorithms three final models were selected as the focus of the evaluation. Pipelines were set up for each model before final refinements were made. Each model was trained using a version of the, previously mentioned, AWID dataset which was balanced to show equal numbers of attack and non-attack instances. The models were then evaluated using a testing dataset which was subjected to the same modifications and was made up of 40,158 observations (20,079 normal traffic (non-attacks) and 20,079 impersonation attacks).

In order to give a full overview of the performance of each model a number of performance measures have been calculated. The majority of these measures involve calculating the: True Positive (TP) values which show intrusions correctly identified as an attacks; True Negative (TN) values which show normal instances correctly identified as non-attack; False Positive (FP) values or normal instances incorrectly identified as an attack; and False Negative (FN) values showing Intrusions incorrectly identified as non-attack. These values were obtained from the classification report run on each pipeline.

A brief summary of each performance measure is outlined below:

In addition to the above measures Time to Build (TTB) in seconds will also be calculated for each model. In order to maintain a level playing field, all models will be run in the same environment using a mac OS with 1.4 GHz Intel i5 CPU and 8 Gb of RAM. Furthermore additional information on each performance measure above can be found in the DEMISe Techniques for IoT Intrusion Detection paper [14] should it be needed.

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pipeline 1 | Pipeline 2 | Pipeline 3 |
| Acc (%) | 99.08 | 99.08 | 98.63 |
| DR (Recall) (%) | 98.15 | 98.15 | 97.96 |
| Precision (%) | 100 | 100 | 99.29 |
| FAR (%) | 0 | 0 | 0.01 |
| FNR (%) | 1.85 | 1.85 | 2.04 |
| F1 (%) | 99.07 | 99.07 | 98.62 |
| MCC (%) | 98.17 | 98.17 | 97.26 |
| TTB (seconds) | 7.24 | 7.4 | 5.57 |

**Evaluation – Pipeline 1**

Pipeline 1 is made up of the following:

* Variance Threshold is used to remove variables with zero variance.
* Fclassif for feature selection.
* Adaboost as the algorithm.

The parameters of each element of the model were tuned using grid search.

As can be seen from the above results table Adaboost performed much better than logistic regression on all measures except time to build.

**Evaluation – Pipeline 2**

Pipeline 2 is made up of the same elements as Pipeline 1 with one addition, a Min/Max scaler is added to scale every variable between 0 and 1.

There is no impact of the scaler on performance other than to increase the time to build the model.

**Evaluation – Pipeline 3**

Pipeline 3 is made up of:

* Variance Threshold is used to remove variables with zero variance.
* Min/Max Scaler to scale variables between 0 and 1.
* Chi Square for feature selection.
* Logistic regression for the algorithm.

Whilst not performing at the same level as the Adaboost model. Logistic Regression still performed well on our test data. In addition, the time to build for a Logistic Regression was over 1.8 seconds faster the Adaboost model.

**Conclusion**

Following experimenting with multiple different algorithms it was clear that the Adaboost model performed significantly better than the other models that were trained on the AWID dataset. This was backed up when the final three models were used on the test data. Therefore, the pipeline 1 model is chosen as our best performing model for this project.

**Future Work**

Further work could be done to test the change in accuracy of our model combining existing selected features with predictors extracted using autoenconders. Our analysis shows that there’s a strong tendency of complex models to exhibit high variance and overfit. Ensembles based learning techniques have been tried to mitigate this issue with Adaboost showing the best results. More research could be done exploring state of the art complex classification algorithms in the area of Deep Learning in conjunction with dropout or regularization techniques to improve generalization.

This research focused on impersonation attacks. It would be interesting to extend our work to different kinds of intrusion techniques analyzing to what extent the features and the algorithms selected for impersonation attacks remain effective. The analysis of the test set revealed substantial differences with the training set, perhaps reflecting ever changing intrusion techniques. Further research could be done to look into the robustness of our methodology in the face of different datasets.

**References**

[1] X. Du, H. Chen, L. Zhu, J. Li and Z. Chang, "Security and Privacy in Wireless IoT," in IEEE Wireless Communications, vol. 25, no. 6, pp. 10-11, December 2018.

[2] R.J. Little, D.B. Rubin. Statistical Analysis with Missing Data. John Wiley & Sons Inc., 2019.

[3] W. McKinney. Python for Data Analysis. O’Reilly Media Inc., 2017.

[4] C. Kolias, G. Kambrourakis, A. Stavrou, S. Gritzalis. Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset. IEEE Communication Surveys and Tutorials, vol.18, no. 1, pp 184 -186, 1st Quarter, 2016.

[5] M.E. Aminanto, R. Choi, T.C. Tanuwidjaja, P.D. Yoo, K. Kim. Deep Abstraction and Weighted Feature Selection for Wi-Fi- Impersonation Detection. IEEE Transactions on Information Forensics and Security, vol. 13, no. 3, pp 621 – 636, March 2018.

[6] IK. Yeo, R. Johnson. A new family of power transformations to improve normality or symmetry. Biometrika,  no 87, pp 954 – 959. 2000.

[7] L. Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001.  
<https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf>

[8] T. Chen &C. Guestrin "XGBoost: A Scalable Tree Boosting System" KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining August 2016 Pages 785–794 <https://doi.org/10.1145/2939672.2939785>.

[9] A.V. Dorogush, V. Ershov & A. Gulin. "CatBoost: gradient boosting with categorical features support" Workshop on ML Systems at NIPS 2017.  
<http://learningsys.org/nips17/assets/papers/paper_11.pdf>

[10] I. Guyon, J. Weston, S. Barnhill, & V. Vapnik (2002). "Gene selection for cancer classification using Support Vector Machines". Machine Learning, 46(1-3), 389–422. <https://doi.org/10.1023/A:1012487302797>

[11]Liu L, [Yu Y](https://www.ncbi.nlm.nih.gov/pubmed/?term=Yu%20Y%5BAuthor%5D&cauthor=true&cauthor_uid=30463545), [Fei Z](https://www.ncbi.nlm.nih.gov/pubmed/?term=Fei%20Z%5BAuthor%5D&cauthor=true&cauthor_uid=30463545), [Li M](https://www.ncbi.nlm.nih.gov/pubmed/?term=Li%20M%5BAuthor%5D&cauthor=true&cauthor_uid=30463545), [Wu FX](https://www.ncbi.nlm.nih.gov/pubmed/?term=Wu%20FX%5BAuthor%5D&cauthor=true&cauthor_uid=30463545), [Li HD](https://www.ncbi.nlm.nih.gov/pubmed/?term=Li%20HD%5BAuthor%5D&cauthor=true&cauthor_uid=30463545), Pan Y, [Wang J](https://www.ncbi.nlm.nih.gov/pubmed/?term=Wang%20J%5BAuthor%5D&cauthor=true&cauthor_uid=30463545). An interpretable boosting model to predict side effects of analgesics for osteoarthritis. IEEE International Conference on Bioinformatics and Biomedicine 2017, pp 5-6.

[12]Tianqi Chen, Carlos Guestr XGBoost: A Scalable Tree Boosting System. University of Washington, pp 3

[13] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, Andrey Gulin Yandex, CatBoost: unbiased boosting with categorical features. Moscow Institute of Physics and Technology, Dolgoprudny, Russia, p

[14]L.R.Parker, Paul D Yoo, Taufiq A Asyhari, Lounis Chermak, Yoonchan Jhi, Kamal Taha. (2019) DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection. University of Birkbeck, pp 8

[15]Yoav Freund and Robert E. Schapire. (1996) A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. AT6T Labs, 180 Park Avenue, Florham Park, New Jersey 07932

[16] M. Claesen and B. De Moor, “Hyperparameter Search in Machine Learning,” in *MIC 2015: The XI Metaheuristics International Conference in Agadir, Morocco*, 2015.

[17] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization Yoshua Bengio,” *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012.

‌

[18] S. Kim, M. Son, K. Kwon, and D. Lee, “Electricity Demand Forecasting Algorithm Development Based on the Gradient Boosting Machine by Selecting the Optimal Combination of Weather Data and Stations,” *The transactions of The Korean Institute of Electrical Engineers*, vol. 68, no. 7, pp. 834–843, Jul. 2019.

‌[19]G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning : with applications in R*. Springer, 2013.

Appendix a: Investigation of Training / Test Data Sets Mismatch

Introduction

It was suspicious that models with a high accuracy (over 98%) when run with cross validation on the training set should score barely better than chance (52-54%) on the test set. Prediction is only possible if both the training set and test set are drawn from the same underlying population and have features that have been identically preprocessed so it was hypothesised that there were some difference between the training data and test data that was causing this.

A comparison of the provided training set and test set was done and it appears that for some features the values in the tests set are considerably different from the values in the training set. The data sets provided had already been scaled so that all values lay in the interval 0 to 1 so a possible cause of the discrepancies was from errors introduced by this pre-processing.

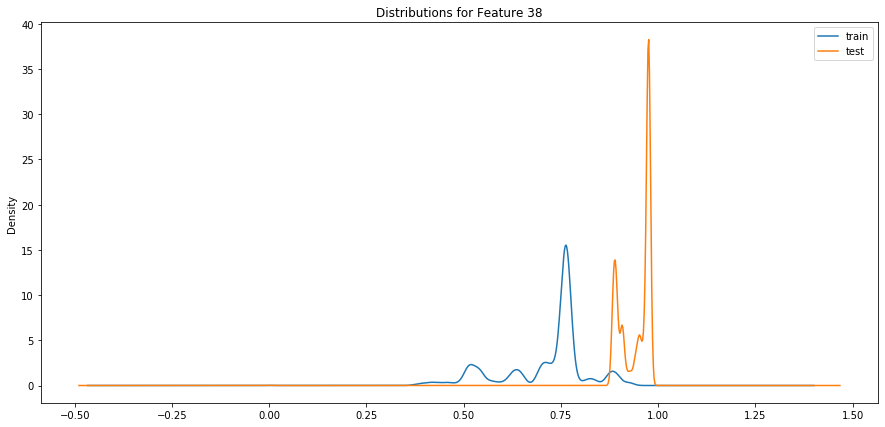
Initial investigation - Kolmogorov-Smirnov test

To get an overview of the problem the Kolmogorv-Smirnov test[1] was used to compare each column in the training data with the same column in the test data. The k-s statistic tests the null hypothesis that two sample sets come from the same underlying distribution. First all columns with zero variance in either the test set or training set were removed, leaving 69 columns. The ks\_2samp function from the scipy package[2] was used to compare the training data and test data for each of the remaining features. Of the 69 features 40 had a p-value at or close to zero (rejecting the null hypothesis) and 18 had a p-value at or close to 1. The remaining 11 features were somewhere in between. This gives an indication that there may be a problem but is not conclusive, particularly since the k-s test assumes that the underlying distribution is continuous.

Direct Comparison of Single Features

To investigate further individual features were selected so that they could be compared directly.

Feature 38

First to be compared was feature “38” (where “38” is the column header in the original CSV file). The distributions were compared by a kde plot for both the training and test data for the feature.  **Figure I.1**

From this plot it appears that the overall shapes are similar but shifted up on the test set. One possible reason for this could be that the data sets were wrongly scaled. You could get this effect if it the training set and test set had been scaled independently instead of the scaler being fit to the training data and applied to the test data - an outlier in the training set could push all the other values down after scaling. Unfortunately we do not have access to the original data to test this hypothesis.

Categorical Features

Several features were found that appeared to be categorical (they only took 2 or 3 values in both data sets) but the values for the categories were completely different. For example feature “129” has this distribution in the test set:

|  |  |
| --- | --- |
| value | count |
| 0.0 | 38897 |
| 1.0 | 1261 |

Which seems reasonable for a 2-value categorical feature. However the training data has

|  |  |
| --- | --- |
| value | Count |
| 0.0 | 94954 |
| 0.0000158 | 2090 |

The count ratios are similar, but the values are different by four orders magnitude.

Similar results have been found for other categorical features. For example feature “128” appears to have three categories and the training set is similarly mis-scaled:

|  |  |  |  |
| --- | --- | --- | --- |
| test set values | test set counts | training set values | training set counts |
| 0.0 | 38897 | 0.0 | 94954 |
| 0.5 | 141 | 0.0000155 | 147 |
| 1.0 | 1120 | 0.0000309 | 1943 |

Several other features were found to have similar problems, mostly in the range “108”-“129”.

Conclusion

It is seems from the above that some of the features have been corrupted by the scaling process.  
For this reason the data sets should be considered unfit for purpose and regenerated from the original raw data if available.

**Appendix b** code:

Pre-processing code

*# Initial modules*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **sklearn**

*#load data*

mdata= pd.read\_csv('train\_imperson\_without4n7\_balanced\_data.csv')

mdata.head()

### **Investigating the presence of missing values**

mdata.info()

*#data is complete*

m1= mdata.dropna()

m1.info()

### **Investigating the presence of duplicate cases**

*#We should assume data are iid, hence keep duplicates*

m2 = mdata.drop\_duplicates()

m2.info()

*#duplicate rows for reference*

mdata[mdata.duplicated() == **True**].head()

### **Descriptive statistics on the training set**

mdata.describe().T.describe()

*# Separate X and Y*

X, Y = mdata.loc[:, mdata.columns != '155'], mdata['155']

X.head()

### **Statistical analysis of the features matrix**

X.describe().T.describe()

X.std().value\_counts()

### **Dataset 1: features with variance greater than zero**

*#Xselected with 0 var taken out*

**from** **sklearn.feature\_selection** **import** VarianceThreshold

selector = VarianceThreshold()

selector.fit(X)

col = X.columns[selector.get\_support()]

Xselected= X.loc[:, col ]

Xselected.head()

Xselected.shape

Xselected.skew().median()

### **Dataset 2: variance greater than zero and scaled to a range [0, 1]**

*#normalised range 0 1*

**from** **sklearn.preprocessing** **import** MinMaxScaler

scaler1 = MinMaxScaler().fit(Xselected)

scaled0\_1=scaler1.transform(Xselected)

*# dataframe normalised range 0\_1*

NX3df = pd.DataFrame(scaled0\_1, columns= Xselected.columns)

NX3df.head()

### **Dataset 3: variance greater than zero and power transform**

*#Power transform*

**from** **sklearn.preprocessing** **import** PowerTransformer

pt = PowerTransformer(method = 'yeo-johnson').fit(Xselected)

poweredX2 = pt.transform(Xselected)

*#dataframe power gaussian*

PX2df = pd.DataFrame(poweredX2, columns= Xselected.columns)

PX2df.head()

PX2df.skew().median()

### **Dataset 4: variance greater than zero and norm one**

*#Normalizer transforms to norm 1*

**from** **sklearn.preprocessing** **import** Normalizer

scaler = Normalizer().fit(Xselected)

normalizedX2 = scaler.transform(Xselected)

*#dataframe with norm 1*

NX2df = pd.DataFrame(normalizedX2, columns = Xselected.columns)

NX2df.head()

**Datasets**

Reduced dataset (down to 78 features) by removing zero variance features - dataset 1

* Xselected

Dataset 1 + normalised the dataset by rescaling each row to add up to length one - dataset 2

* NX2df

Dataset 1 + rescaling each feature between 0 to 1 - dataset 3

* NX3df

Dataset 1 + transform the datset to be gaussian like (mean=0 abd s.d=1) - dataset 4

**import** **numpy** **as** **np**

**import** **scipy**

**import** **pandas** **as** **pd**

**import** **sklearn**

**from** **sklearn.feature\_selection** **import** SelectKBest

**from** **sklearn.feature\_selection** **import** chi2

**from** **sklearn.model\_selection** **import** cross\_validate

**from** **numpy** **import** random, arange

**from** **xgboost** **import** XGBClassifier

Xtrain, Ytrain = Xselected, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))selected\_features = set((25,16,22,13,77,15,2,3,23,34))

**import** **numpy**

X\_top\_features = Xselected.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

indexes = arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

cb = CatBoostClassifier(verbose=**False**, early\_stopping\_rounds=50)

cb\_cv = cross\_validate(cb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

cb\_cv

{'fit\_time': array([16.88924599, 18.57903004, 19.61032724, 16.98861003, 17.83964992]),

'score\_time': array([0.01466632, 0.0195148 , 0.01424694, 0.01427627, 0.0139029 ]),

'test\_score': array([0.99953632, 0.9994848 , 0.99963932, 0.99948475, 0.99969085])}

gb = XGBClassifier(verbosity=0)

gb\_cv = cross\_validate(gb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

gb\_cv

{'fit\_time': array([2.38882828, 2.80635309, 2.65520096, 2.61705399, 2.46459889]),

'score\_time': array([0.0297358 , 0.03513479, 0.03065014, 0.03140807, 0.03049994]),

'test\_score': array([0.99783617, 0.99835137, 0.99773289, 0.99762984, 0.99824815])}

Xtrain, Ytrain = NX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,27))

X\_top\_features = NX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

cb = CatBoostClassifier(verbose=**False**, early\_stopping\_rounds=50)

cb\_cv = cross\_validate(cb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

cb\_cv

{'fit\_time': array([19.48945689, 20.40593696, 18.60622501, 19.67188191, 21.3007338 ]),

'score\_time': array([0.01459622, 0.01379013, 0.01371908, 0.01435018, 0.01378512]),

'test\_score': array([0.99747553, 0.99726945, 0.99701154, 0.99690849, 0.99696002])}

gb = XGBClassifier(verbosity=0)

gb\_cv = cross\_validate(gb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

gb\_cv

{'fit\_time': array([4.14661694, 4.41106415, 4.12002206, 4.19900227, 4.16188002]),

'score\_time': array([0.03509212, 0.03049994, 0.03502893, 0.03120995, 0.03449702]),

'test\_score': array([0.99690881, 0.99696033, 0.99634171, 0.99654782, 0.99690849])}

Xtrain, Ytrain = NX3df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,34))

X\_top\_features = NX3df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

cb = CatBoostClassifier(verbose=**False**, early\_stopping\_rounds=50)

cb\_cv = cross\_validate(cb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

cb\_cv

{'fit\_time': array([18.59923005, 16.90425181, 16.79899788, 17.6864748 , 18.00517201]),

'score\_time': array([0.01949787, 0.0240171 , 0.01366305, 0.01427627, 0.01497293]),

'test\_score': array([0.9997424 , 0.9997424 , 0.99953627, 0.99969085, 0.9991756 ])}

gb = XGBClassifier(verbosity=0)

gb\_cv = cross\_validate(gb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

gb\_cv

{'fit\_time': array([2.54442596, 2.60541296, 2.54903793, 2.38389492, 2.48685789]),

'score\_time': array([0.03097701, 0.03572512, 0.02722311, 0.03145814, 0.026968 ]),

'test\_score': array([0.99804225, 0.99783617, 0.99778442, 0.99809357, 0.99747527])}

**from** **sklearn** **import** feature\_selection

Xtrain, Ytrain = PX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((22,32,25,13,16,15,29,28,23,0))

X\_top\_features = PX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

cb = CatBoostClassifier(verbose=**False**, early\_stopping\_rounds=50)

cb\_cv = cross\_validate(cb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

cb\_cv

gb = XGBClassifier(verbosity=0)

gb\_cv = cross\_validate(gb, X\_cv, Y\_cv, cv=5, scoring="accuracy")

gb\_cv

**from** **sklearn.ensemble** **import** RandomForestClassifier

**from** **sklearn** **import** feature\_selection

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(k=10)

fit = bestfeatures.fit(Xtest,Ytest)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtest.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

f = msb / msw

selected\_features = set((66,72,64,76,35,47,48,44,78,5))

X\_top\_features = Xtest.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytest.shape[0])

numpy.random.shuffle(indexes)

Xtest\_cv = X\_top\_features.iloc[indexes,:]

Ytest\_cv = Ytest.iloc[indexes]

Xtrain, Ytrain = Xselected, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,34))

**import** **numpy**

X\_top\_features = Xselected.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

**from** **sklearn.ensemble** **import** RandomForestClassifier

*#Create a Gaussian Classifier*

clf=RandomForestClassifier(n\_estimators=100)

*#Train the model using the training sets y\_pred=clf.predict(X\_test)*

clf.fit(X\_cv,Y\_cv)

*# prediction on test set*

y\_pred=clf.predict(Xtest\_cv)

*#Import scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

*# Model Accuracy, how often is the classifier correct?*

print("Accuracy:",metrics.accuracy\_score(Ytest\_cv, y\_pred))

Xtrain, Ytrain = NX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,27))

X\_top\_features = NX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

*#Create a Gaussian Classifier*

clf=RandomForestClassifier(n\_estimators=100)

*#Train the model using the training sets y\_pred=clf.predict(X\_test)*

clf.fit(X\_cv,Y\_cv)

*# prediction on test set*

y\_pred=clf.predict(Xtest\_cv)

*#Import scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

*# Model Accuracy, how often is the classifier correct?*

print("Accuracy:",metrics.accuracy\_score(Ytest\_cv, y\_pred))

Xtrain, Ytrain = NX3df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,34))

X\_top\_features = NX3df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

*#Create a Gaussian Classifier*

clf=RandomForestClassifier(n\_estimators=100)

*#Train the model using the training sets y\_pred=clf.predict(X\_test)*

clf.fit(X\_cv,Y\_cv)

*# prediction on test set*

y\_pred=clf.predict(Xtest\_cv)

*#Import scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

*# Model Accuracy, how often is the classifier correct?*

print("Accuracy:",metrics.accuracy\_score(Ytest\_cv, y\_pred))

Xtrain, Ytrain = PX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

**import** **numpy**

selected\_features = set((22,32,25,13,16,15,29,28,23,0))

X\_top\_features = PX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

*#Create a Gaussian Classifier*

clf=RandomForestClassifier(n\_estimators=100)

*#Train the model using the training sets y\_pred=clf.predict(X\_test)*

clf.fit(X\_cv,Y\_cv)

*# prediction on test set*

y\_pred=clf.predict(Xtest\_cv)

*#Import scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

*# Model Accuracy, how often is the classifier correct?*

print("Accuracy:",metrics.accuracy\_score(Ytest\_cv, y\_pred))

**from** **sklearn** **import** feature\_selection

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(k=10)

fit = bestfeatures.fit(Xtest,Ytest)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtest.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

f = msb / msw

selected\_features = set((66,72,64,76,35,47,48,44,78,5))

X\_top\_features = Xtest.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytest.shape[0])

numpy.random.shuffle(indexes)

Xtest\_cv = X\_top\_features.iloc[indexes,:]

Ytest\_cv = Ytest.iloc[indexes]

**from** **sklearn.naive\_bayes** **import** GaussianNB

Xtrain, Ytrain = Xselected, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,34))

**import** **numpy**

X\_top\_features = Xselected.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

%time

model = GaussianNB()

model.fit(X\_cv, Y\_cv)

GaussianNB(priors=None, var\_smoothing=1e-09)

model.score(X\_cv,Y\_cv)

0.9606673261613289

model.score(Xtest\_cv,Ytest\_cv)

%time

Xtrain, Ytrain = NX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,27))

X\_top\_features = NX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

%time

model = GaussianNB()

model.fit(X\_cv, Y\_cv)

model.predict(X\_cv)

model.score(X\_cv,Y\_cv)

model.score(Xtest\_cv,Ytest\_cv)

%time

Xtrain, Ytrain = NX3df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(score\_func=chi2, k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

selected\_features = set((25,16,22,13,77,15,2,3,23,34))

X\_top\_features = NX3df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes]

model = GaussianNB()

model.fit(X\_cv, Y\_cv)

model.predict(X\_cv)

model.score(X\_cv,Y\_cv)

model.score(Xtest\_cv,Ytest\_cv)

Xtrain, Ytrain = PX2df, Y

*#apply SelectKBest class to extract top 10 best features*

bestfeatures = SelectKBest(k=10)

fit = bestfeatures.fit(Xtrain,Ytrain)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns = pd.DataFrame(Xtrain.columns)

*#concat two dataframes for better visualization*

featureScores = pd.concat([dfcolumns,dfscores],axis=1)

*#naming the dataframe columns*

featureScores.columns = ['feature','Score']

*#print 10 best features*

print(featureScores.nlargest(10,'Score'))

**import** **numpy**

selected\_features = set((22,32,25,13,16,15,29,28,23,0))

X\_top\_features = PX2df.iloc[:, list(selected\_features)]

indexes = numpy.arange(Ytrain.shape[0])

numpy.random.shuffle(indexes)

X\_cv = X\_top\_features.iloc[indexes,:]

Y\_cv = Ytrain.iloc[indexes

model = GaussianNB()

model.fit(X\_cv, Y\_cv)

model.predict(X\_cv)

model.score(X\_cv,Y\_cv)

model.score(Xtest\_cv,Ytest\_cv)

refining

*#load data*

traindata= pd.read\_csv(r'C:\Users\Student\Git\AML\_project\datasets\train\_imperson\_without4n7\_balanced\_data.csv')

testdata= pd.read\_csv(r'C:\Users\Student\Git\AML\_project\datasets\test\_imperson\_without4n7\_balanced\_data.csv')

*# Separate X and Y*

X\_train, Y\_train = traindata.loc[:, traindata.columns != '155'], traindata['155']

X\_test, Y\_test = testdata.loc[:, testdata.columns != '155'], testdata['155']

**from** **sklearn.model\_selection** **import** GridSearchCV

pipeline1 = Pipeline([

('zero variance', VarianceThreshold()),

('top features', SelectKBest(f\_classif)),

('ada',AdaBoostClassifier())])

grid=GridSearchCV(cv=10,

estimator=pipeline1,

param\_grid={'ada\_\_n\_estimators': [10,50,100,150],

'ada\_\_learning\_rate':[0.001,0.1,0.01],

'top features\_\_k':[15,20,30,35]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

print(grid.best\_params\_)

print(grid.best\_score\_)

{'ada\_\_learning\_rate': 0.01, 'ada\_\_n\_estimators': 100, 'top features\_\_k': 35}

**from** **sklearn.model\_selection** **import** GridSearchCV

pipeline2= Pipeline([

('zero variance', VarianceThreshold()),

('norm 1', Normalizer()),

('top features', SelectKBest(chi2)),

('model',LogisticRegression())])

grid=GridSearchCV(cv=10,

estimator=pipeline2,

param\_grid={'top features\_\_k':[15,20,30,35],

'model\_\_C': [0.01, 0.1, 1, 10, 100, 1000]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

*# sorted(pipeline2.get\_params().keys()) #list of the parameters you can tune*

print(grid.best\_params\_)

print(grid.best\_score\_)

C:\Users\amatu\Anaconda3\lib\site-packages\sklearn\linear\_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.

FutureWarning)

{'model\_\_C': 1000, 'top features\_\_k': 20}

0.98313136309303

**from** **sklearn.model\_selection** **import** GridSearchCV

pipeline2a= Pipeline([

('zero variance', VarianceThreshold()),

('minmax', MinMaxScaler()),

('top features', SelectKBest(chi2)),

('model',LogisticRegression())])

grid=GridSearchCV(cv=10,

estimator=pipeline2a,

param\_grid={'top features\_\_k':[15,20,30,35],'model\_\_C': [0.01, 0.1, 1, 10, 100, 1000]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

*#sorted(pipeline1.get\_params().keys()) #list of the parameters you can tune*

print(grid.best\_params\_)

print(grid.best\_score\_)

C:\Users\amatu\Anaconda3\lib\site-packages\sklearn\linear\_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.

FutureWarning)

{'model\_\_C': 1000, 'top features\_\_k': 20}

0.9875520382506904

pipeline3 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top features', SelectKBest(f\_classif)),

('ada',AdaBoostClassifier())])

grid=GridSearchCV(cv=10,

estimator=pipeline3,

param\_grid={'ada\_\_n\_estimators': [10,50,100,150],

'ada\_\_learning\_rate':[0.001,0.1,0.01],

'top features\_\_k':[15,20,30,35]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

*#sorted(pipeline1.get\_params().keys()) #list of the parameters you can tune*

print(grid.best\_params\_)

print(grid.best\_score\_)

{'ada\_\_learning\_rate': 0.01, 'ada\_\_n\_estimators': 100, 'top features\_\_k': 35}

pipeline1 = Pipeline([

('zero variance', VarianceThreshold()),

('top20 features', SelectKBest(f\_classif, k=35)),

('ada',AdaBoostClassifier(n\_estimators=100,learning\_rate=0.01))])

pipeline1.fit(X\_train,Y\_train)

Y\_predict=pipeline1.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

pipeline2= Pipeline([

('zero variance', VarianceThreshold()),

('norm 1', Normalizer()),

('top20 features', SelectKBest(chi2, k=20)),

('model',LogisticRegression(C=1000,solver='newton-cg'))])

pipeline2.fit(X\_train,Y\_train)

Y\_predict=pipeline2.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

pipeline3 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(f\_classif, k=35)),

('ada',AdaBoostClassifier(n\_estimators=100,learning\_rate=0.01))])

pipeline3.fit(X\_train,Y\_train)

Y\_predict=pipeline3.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

pipeline5 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(chi2, k=20)),

('ada',AdaBoostClassifier())])

grid=GridSearchCV(cv=10,

estimator=pipeline5,

param\_grid={'ada\_\_n\_estimators': [10,50,100,150],

'ada\_\_learning\_rate':[0.001,0.1,0.01]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

print(grid.best\_params\_)

print(grid.best\_score\_)

pipeline5 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(chi2, k=20)),

('ada',AdaBoostClassifier())])

grid=GridSearchCV(cv=10,

estimator=pipeline5,

param\_grid={'ada\_\_n\_estimators': [10,50,100,150],

'ada\_\_learning\_rate':[0.001,0.1,0.01]},

scoring = 'accuracy',

n\_jobs=-1)

grid.fit(X\_train, Y\_train)

print(grid.best\_params\_)

print(grid.best\_score\_)

pipeline1 = Pipeline([

('zero variance', VarianceThreshold()),

('top20 features', SelectKBest(f\_classif, k=20)),

('ada',AdaBoostClassifier(n\_estimators=150,learning\_rate=0.1))])

pipeline1.fit(X\_train,Y\_train)

Y\_predict=pipeline1.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

pipeline2 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(f\_classif, k=20)),

('ada',AdaBoostClassifier(n\_estimators=150,learning\_rate=0.1))])

pipeline2.fit(X\_train,Y\_train)

Y\_predict=pipeline2.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

pipeline3= Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(chi2, k=20)),

('model',LogisticRegression(C=1000,solver='newton-cg'))])

pipeline3.fit(X\_train,Y\_train)

Y\_predict=pipeline3.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict)\*100

evaluation

*Pipeline 1 - Accuracy*

pipeline1 = Pipeline([

('zero variance', VarianceThreshold()),

('top20 features', SelectKBest(f\_classif, k=20)),

('ada',AdaBoostClassifier(n\_estimators=150,learning\_rate=0.1))])

pipeline1.fit(X\_train,Y\_train)

Y\_predict1=pipeline1.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict1)\*100

matrix = confusion\_matrix(Y\_test, Y\_predict1)

print(matrix)

p1\_TP = matrix[0][0]

p1\_TN = matrix[1][1]

p1\_FP = matrix[1][0]

p1\_FN = matrix[0][1]

print('')

print('True Positive = **%.0f**' % p1\_TP)

print('True Negative = **%.0f**' % p1\_TN)

print('')

print('False Positive = **%.0f**' % p1\_FP)

print('False Negative = **%.0f**' % p1\_FN)

[[19708 371]

[ 0 20079]]

True Positive = 19708

True Negative = 20079

False Positive = 0

False Negative = 371

*#Pipeline 1 - Evaluation Metrics*

p1\_Acc = (p1\_TP + p1\_TN) / (p1\_TP + p1\_TN + p1\_FP + p1\_FN)

print('Acc = **%.4f**' % p1\_Acc)

p1\_DR = p1\_TP / (p1\_TP + p1\_FN)

print('DR(Recall) = **%.4f**' % p1\_DR)

p1\_Prec = p1\_TP / (p1\_TP + p1\_FP)

print('Precision = **%.4f**' % p1\_Prec)

p1\_FAR = p1\_FP / (p1\_TN + p1\_FP)

print('FAR = **%.4f**' % p1\_FAR)

p1\_FNR = p1\_FN / (p1\_FN + p1\_TP)

print('FNR = **%.4f**' % p1\_FNR)

p1\_F1 = (2 \* p1\_TP) / (2\*p1\_TP + p1\_FP + p1\_FN)

print('F1 = **%.4f**' % p1\_F1)

p1\_MCC = ((p1\_TP \* p1\_TN) - (p1\_FP \* p1\_FN)) / math.sqrt((p1\_TP + p1\_FP)\*(p1\_TP + p1\_FN)\*(p1\_TN + p1\_FP)\*(p1\_TN + p1\_FN))

print('MCC = **%.4f**' % p1\_MCC)

Acc = 0.9908

DR(Recall) = 0.9815

Precision = 1.0000

FAR = 0.0000

FNR = 0.0185

F1 = 0.9907

MCC = 0.9817

*#Pipeline 1 - ROC Curve*

probs = pipeline1.predict\_proba(X\_test)

probs = probs[:, 1]

auc = roc\_auc\_score(Y\_test, probs)

print('AUC: **%.3f**' % auc)

fpr, tpr, thresholds = roc\_curve(Y\_test, probs)

pyplot.plot([0,1],[0,1], linestyle = '--')

pyplot.plot(fpr,tpr,marker='.')

pyplot.show()

*#Pipeline 2*

pipeline2 = Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(f\_classif, k=20)),

('ada',AdaBoostClassifier(n\_estimators=150,learning\_rate=0.1))])

pipeline2.fit(X\_train,Y\_train)

Y\_predict2=pipeline2.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict2)\*100

*#Pipeline 2 - Confusion Matrix*

matrix = confusion\_matrix(Y\_test, Y\_predict2)

print(matrix)

p2\_TP = matrix[0][0]

p2\_TN = matrix[1][1]

p2\_FP = matrix[1][0]

p2\_FN = matrix[0][1]

print('')

print('True Positive = **%.0f**' % p2\_TP)

print('True Negative = **%.0f**' % p2\_TN)

print('')

print('False Positive = **%.0f**' % p2\_FP)

print('False Negative = **%.0f**' % p2\_FN)

True Positive = 19708

True Negative = 20079

False Positive = 0

False Negative = 371

*#Pipeline 2 - Evaluation Metrics*

p2\_Acc = (p2\_TP + p2\_TN) / (p2\_TP + p2\_TN + p2\_FP + p2\_FN)

print('Acc = **%.4f**' % p2\_Acc)

p2\_DR = p2\_TP / (p2\_TP + p2\_FN)

print('DR(recall) = **%.4f**' % p2\_DR)

p2\_Prec = p2\_TP / (p2\_TP + p2\_FP)

print('Precision = **%.4f**' % p2\_Prec)

p2\_FAR = p2\_FP / (p2\_TN + p2\_FP)

print('FAR = **%.4f**' % p2\_FAR)

p2\_FNR = p2\_FN / (p2\_FN + p2\_TP)

print('FNR = **%.4f**' % p2\_FNR)

p2\_F1 = (2 \* p2\_TP) / (2\*p2\_TP + p2\_FP + p2\_FN)

print('F1 = **%.4f**' % p2\_F1)

p2\_MCC = ((p2\_TP \* p2\_TN) - (p2\_FP \* p2\_FN)) / math.sqrt((p2\_TP + p2\_FP)\*(p2\_TP + p2\_FN)\*(p2\_TN + p2\_FP)\*(p2\_TN + p2\_FN))

print('MCC = **%.4f**' % p2\_MCC)

Acc = 0.9908

DR(recall) = 0.9815

Precision = 1.0000

FAR = 0.0000

FNR = 0.0185

F1 = 0.9907

MCC = 0.9817

*#Pipeline 2 - ROC Curve*

probs = pipeline2.predict\_proba(X\_test)

probs = probs[:, 1]

auc = roc\_auc\_score(Y\_test, probs)

print('AUC: **%.3f**' % auc)

fpr, tpr, thresholds = roc\_curve(Y\_test, probs)

pyplot.plot([0,1],[0,1], linestyle = '--')

pyplot.plot(fpr,tpr,marker='.')

pyplot.show()

*#Pipeline 3*

pipeline3= Pipeline([

('zero variance', VarianceThreshold()),

('scale 0\_1', MinMaxScaler()),

('top20 features', SelectKBest(chi2, k=20)),

('model',LogisticRegression(C=1000,solver='newton-cg'))])

pipeline3.fit(X\_train,Y\_train)

Y\_predict3=pipeline3.predict(X\_test)

accuracy\_score(Y\_test, Y\_predict3)\*100

*#Pipleline 3 - Classification Report*

report = classification\_report(Y\_test, Y\_predict3)

print(report)

precision recall f1-score support

0 0.99 0.98 0.99 20079

1 0.98 0.99 0.99 20079

accuracy 0.99 40158

macro avg 0.99 0.99 0.99 40158

weighted avg 0.99 0.99 0.99 40158

matrix = confusion\_matrix(Y\_test, Y\_predict3)

print(matrix)

p3\_TP = matrix[0][0]

p3\_TN = matrix[1][1]

p3\_FP = matrix[1][0]

p3\_FN = matrix[0][1]

print('')

print('True Positive = **%.0f**' % p3\_TP)

print('True Negative = **%.0f**' % p3\_TN)

print('')

print('False Positive = **%.0f**' % p3\_FP)

print('False Negative = **%.0f**' % p3\_FN)

True Positive = 19669

True Negative = 19938

False Positive = 141

False Negative = 410

*#Pipeline 3 - Evaluation Metrics*

p3\_Acc = (p3\_TP + p3\_TN) / (p3\_TP + p3\_TN + p3\_FP + p3\_FN)

print('Acc = **%.4f**' % p3\_Acc)

p3\_DR = p3\_TP / (p3\_TP + p3\_FN)

print('DR(recall) = **%.4f**' % p3\_DR)

p3\_Prec = p3\_TP / (p3\_TP + p3\_FP)

print('Precision = **%.4f**' % p3\_Prec)

p3\_FAR = p3\_FP / (p3\_TN + p3\_FP)

print('FAR = **%.4f**' % p3\_FAR)

p3\_FNR = p3\_FN / (p3\_FN + p3\_TP)

print('FNR = **%.4f**' % p3\_FNR)

p3\_F1 = (2 \* p3\_TP) / (2\*p3\_TP + p3\_FP + p3\_FN)

print('F1 = **%.4f**' % p3\_F1)

p3\_MCC = ((p3\_TP \* p3\_TN) - (p3\_FP \* p3\_FN)) / math.sqrt((p3\_TP + p3\_FP)\*(p3\_TP + p3\_FN)\*(p3\_TN + p3\_FP)\*(p3\_TN + p3\_FN))

print('MCC = **%.4f**' % p3\_MCC)

Acc = 0.9863

DR(recall) = 0.9796

Precision = 0.9929

FAR = 0.0070

FNR = 0.0204

F1 = 0.9862

MCC = 0.9726

*#Pipeline 3 - ROC Curve*

probs = pipeline3.predict\_proba(X\_test)

probs = probs[:, 1]

auc = roc\_auc\_score(Y\_test, probs)

print('AUC: **%.3f**' % auc)

fpr, tpr, thresholds = roc\_curve(Y\_test, probs)

pyplot.plot([0,1],[0,1], linestyle = '--')

pyplot.plot(fpr,tpr,marker='.')

pyplot.show()