Customer Churn Prediction

A study carried out in the context of the Advanced Machine Learning Principles and Concepts course

Vasileios Nikiforidis
MSc, School of Science and Technology
International Hellenic University
Thessaloniki, Greece
v_nikiforidis@outlook.com

***Abstract*—This paper presents an application of Machine Learning principles, with the goal of predicting customer churning. This act has relied on human-brain power for many years, but with the rise of Machine Learning in the recent years we can now replicate the procedure, most times with higher accuracy, using computers. The main advantage of using Machine Learning is that, once an algorithm is taught what to do with the data given to it, it can do its work automatically.**

Keywords—Machine Learning; churn; algorithm; prediction; automation

I. INTRODUCTION

Sometimes after viewing the data, we cannot interpret the pattern or extract information from the it. In such a case, we apply machine learning. With the abundance of datasets available, the demand for machine learning is in rise. Many industries from medicine to military apply machine learning to extract relevant information [1].

The purpose of machine learning is to learn from the data. Many studies have been done on how to make machines learn by themselves [2] [3]. In this paper we will demonstrate the application of a Machine Learning techniques to predict whether a customer will change telecommunications provider, which falls under the category of Classification problems. The dataset that was used had a total of 4250 entries and 19 features for each one, not including the "churn" column, which was the target variable.

II. CONCEPTS

In this section we provide a definition and briefly discuss the core concepts of this paper.

A. Machine Learning

Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead [4]. As a branch of artificial intelligence, Machine learning is the ability for computer programs to analyze big data, extract information automatically, and learn from it.

Data is being generated faster than at any other time in history. We have reached a point where data analysis cannot be done manually due to the sheer amount of daily produced data.

Many still think of Machine Learning as 'just another algorithm'. The key difference between a regular algorithm and a Machine Learning Model is that a human has not written any code for the model to be able to differentiate between, for example, a dog and a cat. Instead the model has been taught how to reliably discriminate between the two by training on a huge amount of labelled data, also called training set, containing features describing the two animals

B. Supervised Learning

Supervised learning algorithms construct a mathematical model of a set of data that includes both the inputs and the expected outputs [5]. This set is called training data and consists of a set of training examples. Each example has one or more inputs and the expected output, also known as a supervisory signal. In the mathematical formula, each example is portrayed by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through repetitive optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new unlabelled inputs.[4] An optimal function allows the algorithm to correctly determine the output for these unlabelled inputs that were not a part of the training data.

Supervised learning algorithms include classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, typically binary, and regression algorithms are used when the outputs may have any numerical value within a range. For the purposes of this study, we will use a classification algorithm, since we want to predict whether a customer will change telecommunications provider or not, and thus our problem falls under the Supervised Learning spectrum. There are also other approaches like, Unsupervised Learning, Transfer Learning, Self-Learning, Reinforcement Learning, Feature Learning and Semi-supervised Learning but since they are not inside the scope of our study, they will not be further explained here.

III. TECHNOLOGIES

In this section, we present the technologies used for this study along with a brief discussion on how they were chosen.

A. Programming Language

For the purposes of this study, **Python** was the language of choice. In most cases, when one wants to play around with machine learning models, Python is the go-to language. This is even more true when delving deeper, into the area of Deep Learning.

There are many reasons as to why Python is such a big deal in machine learning. To start with, it is one of the easiest to understand languages. In very few words, machine learning is recognizing patterns in data and making intelligent decisions based on those patters. Due to Pythons' readability and non-complexity, one can easily understand how the model fulfils its purpose and improve on it. Furthermore, Python has a huge number of supporting libraries, some of which are specialized in machine learning.

Finally, while with other programming languages one would need to familiarize themselves with the language before being able to use it for machine learning, this is not the case for Python. Even with basic knowledge of the language, one can directly use it for machine learning due to the huge number of libraries and community support available.

B. Framework

In layman terms, a framework provides a standard way to build and deploy an application and is a universal, reusable software environment that provides functionality as part of a larger software platform to facilitate development of software applications, products and solutions. Frameworks may include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or system.

Frameworks have key distinguishing features that separate them from normal libraries [6]:

- **inversion of control:** In a framework, unlike in libraries or in standard user applications, the overall program's flow of control is not dictated by the caller, but by the framework.
- **extensibility:** A user can extend the framework – usually by selective overriding – or programmers can add specialized user code to provide specific functionality.
- **non-modifiable framework code:** The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but cannot modify its code.

C. Platform/IDE

When coding in any language, most developers tend to use an IDE which best suits their needs. An IDE provides functionalities and easiness for some features of the language while also automating some processes. Thankfully, there are many easy to use and powerful IDEs out there for Python, but what is the core need for machine learning, is computational power, especially when dealing with Big Data.

For this reason, Google Colaboratory [7] (also known as Colab) was chosen to develop the Neural Network presented in this paper. It is a part of the Project Jupyter [8], thus it provides the user with a free Jupyter notebook environment that runs in the **cloud.** This means that all the computational power is provided by the cloud and even low-end computers can now run complex and taxing models with no problem, which makes it an ideal environment for any Machine Learning project.

IV. DATA & PROBLEM DESCRIPTION

The ability to predict that a customer is at a high risk of churning, while there is still time to do something about it, represents a huge additional potential revenue source for every online business. Besides the direct loss of revenue that results from a customer abandoning the business, the costs of initially acquiring that customer may not have already been covered by the customer's spending to date. (In other words, acquiring that customer may have been a losing investment.) Furthermore, it is always more difficult and expensive to acquire a new customer than it is to retain a current paying customer. [9]

The dataset provided was split in two separate csv files, one containing the training data and one containing the test data. The two files contained the same columns except for the "churn" column. Taking a quick look at the dataset, one can easily see that some preprocessing of the data must be done before moving on to creating a model. To make our lives a little bit easier, there were no missing values in the datasets, but unfortunately, there was a high imbalance on the target variable.

First, in the training set we switch the values of the "churn" (or target) column, from "yes" and "no" to 1 and 0. This was simple enough to do programmatically, but the Label Encoder could also be used, but more on that later. Next, we have the "area_code" column which has values like "area_code_123". A model will most likely understand the number 123 better than the string "area_code_123" (for this reason we also swapped the "churn" column), thus we keep only the number part of these values. Then, on the test set we drop the "id" column, since it offers no value on predictions and would simply be noise.

Moving on, we can see that there are more categorical features in the set. These are: "state", "international_plan" and "voice_mail_plan". The two most well-known approaches on handling categorical data, are using Label Encoder and One-hot Encoder. What the Label encoder does is, taking categorical text data and converting it to model-understandable numerical data. The problem with label encoding is that it can translate a

text field to a field with multiple numbers. Thus, when this is done in the whole dataset, we have different numbers in the same column with no relation to each other, which the model may misunderstand. So we opted to use One Hot Encoder, which takes a column which has been Label encoded (if not it does so itself in the latest versions) and splits it into multiple new columns, each containing either 1 or 0 depending on which column has what value. After testing with both encoders, the OneHotEncoder gave slightly better results overall.

Moving on, another problem to be tackled would be the high class imbalance of the "churn" target value.

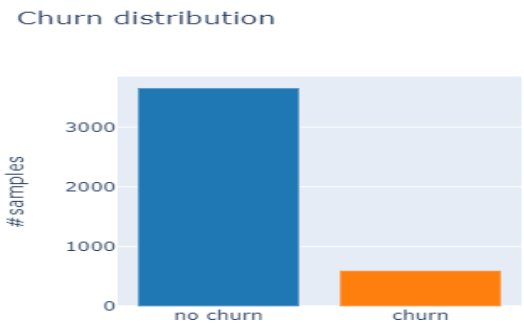

Figure 1. Target variable class imbalance

We can see that there are clearly more samples for customers churning than not. So, we have a class imbalance for the target variable which could lead to predictive models biased towards the majority. To get around this issue, a mode of cross-validation called StratifiedKFold was used.

Finally, due to the large variance in the numerical features of the dataset, some being in the hundreds (i.e. area_code) while others being below ten, a scaling was necessary in order to help the model better understand these features and not ignore the ones with small numbers. Hence, the scikit-learn StandardScaler was used.

V. Description of Models

The current section is devoted to introducing the reader into the methodology and the models used in this study.

The first attempt at this problem was done using a simple Multi-Layer Perceptron Neural Network. The network had two (2) hidden layers between its input and output layers. All layers except the output had the ReLu activation function, whereas the output layer, as with most classification problems, had the Sigmoid. The network was fitted over 200 epochs and produced extremely high accuracy on the training data, close to 100%, while it did quite worse on the test data, ~92%. This naturally raised suspicions that the network had overfitted, so Dropout Layers were added after each layer but the output to combat this issue. After refitting the network, the accuracy stabilized around 96% with slightly worse results on the test data, around 95%.

Another batch of submissions was based on XGBoost which seemed to have the highest accuracy of any model or ensemble technique used and produced the best submission accuracy (98.2%). To figure out the best hyperparameters for the XGBoost model [10], a grid search was done while lowering the models' learning rate so that we get a better understanding of the hyperparameters impact on training.
The best number of estimators for the model was found to either be 100 or 1000 with any other value providing worse accuracy. Next, the min_child_weight parameter was tunned, for which A smaller value is chosen because it is a highly imbalanced class problem and leaf nodes can have smaller size groups, along with the max_depth, both of which have high impact on the model performance. The rest of the hyperparameters had only a slight effect on the performance of the model but still were tuned, nonetheless. The final model after tuning was:

```
xg_boost = XGBClassifier(
    learning_rate = 0.3,
    n_estimators = 1000,
    max_depth = 8,
    min_child_weight = 0,
    gamma = 0.8,
    subsample = 0.9,
    colsample_bytree = 0.7,
    reg_alpha = 0.1,
    objective = 'binary:logistic',
    nthread = 4,
    scale_pos_weight = 1,
    seed = defaultSeed
)
```

Furthermore, it was possible to also get more insights on our data thanks to the built-in feature importance attribute of XGBoost. Below we can see the top 10 features. The features with numbers for names are produced by OneHotEncoder.

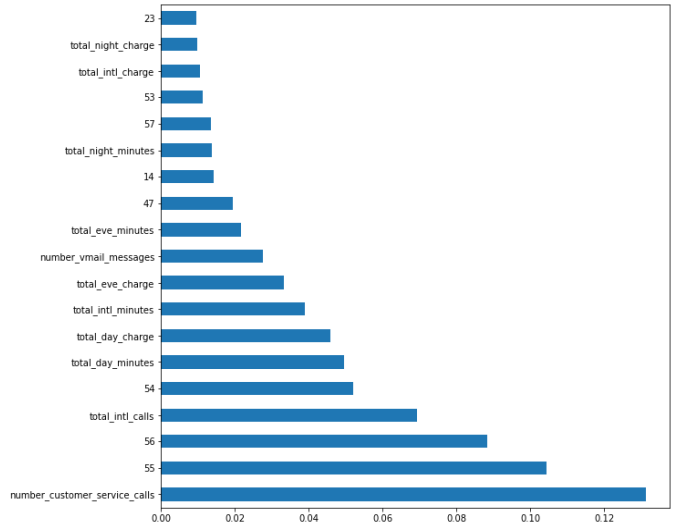

Figure 2. XGBoost Feature Importance

As expected, the more one needs to call customer service, the more likely they are to switch provider.

Finally, an interesting approach was stacking other models. This model approach is called super learner [11].
The super learner algorithm is an application of stacked generalization [12], called stacking, to k-fold cross-validation where all models use the same k-fold splits of the data and a meta-model is fir on the out-of-fold prediction from each

model. All in all, the super learner is an ensemble machine learning algorithm that combines all of the models that you might investigate for a predictive modeling problem and uses them to make a prediction as-good-as or better than any single model that you may have investigated [11]. To implement a super learner algorithm there were two options. The first one was to write the logic behind it, which is not very complicated and was actually done with the same results as the second option. The second option was to use the SuperLearner class from the ml-ensemble library. This option was chosen due to the under-the-hood optimization such a library would have. Three models were tested as meta-models for the super learner as described before, Logistic Regression, Random Forest and Gradient Boost. The models comprising the super learner were initially eleven (11): Logistic Regression, Decision Tree, SVC, Gaussian Naive Bayes, KNN, Ada Boost, Bagging, Random Forest, Extra Trees, Gradient Boost and XGBoost. After some test, four (5) of them were dropped due to having quire worse results in comparison to the other models in the super learner which resulted in worse learning. Those were Logistic Regression, Gaussian Naive Bayes, SVC, KNN and Ada Boost. While building the learner, we can add layers of models in its pipeline where each layer passes information to the next one. This process reminds one of a Neural Network but instead of nodes we have models. Some tests proved that the best option for this problem was to have all the models in one layer, which meant that the learner would have two (2) layers, one being the models and the other its own meta-model. Additionally, each meta-model added created one additional layer. The final learner looked like this:

```
----------Super Learner----------
Train (2975, 73) (2975,) Test (1275, 73) (1275,)

Fitting 4 layers
Processing layer-1             done | 00:00:26
Processing layer-2             done | 00:00:00
Processing layer-3             done | 00:00:00
Processing layer-4             done | 00:00:00
Fit complete                        | 00:00:26
                                score-m  score-s  ft-m  ft-s  pt-m  pt-s
layer-1  baggingclassifier         0.95     0.01  0.50  0.02  0.00  0.00
layer-1  decisiontreeclassifier    0.91     0.01  0.07  0.01  0.00  0.00
layer-1  extratreesclassifier      0.92     0.02  0.64  0.02  0.03  0.00
layer-1  gradientboostingclassifier 0.95    0.01  1.71  0.02  0.00  0.00
layer-1  randomforestclassifier    0.94     0.01  0.86  0.02  0.03  0.00
layer-1  xgbclassifier             0.95     0.01  0.81  0.04  0.00  0.00


Predicting 4 layers
Processing layer-1             done | 00:00:00
Processing layer-2             done | 00:00:00
Processing layer-3             done | 00:00:00
Processing layer-4             done | 00:00:00
Predict complete                    | 00:00:00
Super Learner: 95.843
```

Figure 3. Super Learner

Without any tuning, the super learner has the highest accuracy between any model tested. After tuning a bit some models the learner used, its accuracy increased to match that of the tuned XGBoost. The tuning was match faster because not many hyperparameters were tested for each algorithm, but the running time was quite slower compared to that of XGBoost, thus XGBoost was chosen for more test. Despite that, this approach has much more to be investigated and tested compared to XGBoost where a grid search was performed on every hyperparameter. Perhaps with more research in this approach we could achieve higher results.

VI. Comparative Experiments and Results

This section is devoted to displaying the results of this study.

Many out-of-the-box algorithms were tested from scikit-learn package. For each of those algorithms, a 10-fold cross validation was run in order to be proactive about overfitting. Initially over 10 algorithms were tested, but some were found to be quite inaccurate, so they were dropped from further testing. The final candidates were Decision Tree, Random Forest, SVC, Gradient Boost, XG Boost, and Extra Trees. Along with those, three additional ensemble techniques were tested, Super Learner, Bagging and Voting/Committee (not taking into account the boosting algorithms who also fall under the ensemble category). Except the super learner, the others were quite straightforward by using models directly out of scikit-learn. The only slight issue was that we had to create all possible combinations of "voters" for the Voting Classifier, which was done using the iteration Python package.

Classifier	Accuracy (training)	Accuracy (submission)
MLP Neural Net	97.7%	94%
Decision Tree	92.4%	-
SVC	88%	-
Bagging	94.9%	~93%
Random Forest	95.2%	94.6%
Extra Trees	92.4%	-
Gradient Boost	95.5%	96%
XG Boost	95.6%	96.6%
Super Learner	95.8%	94.6%
Voting	95.4%	95%

After getting the initial results, it was clear that Boosting was the way to go, since those models behaved as expected, without being victims of overfitting. To be clearer, it was expected than when a model would be trained on the whole dataset, instead of a part of it, the other part being kept for validation and testing, it would perform better on the submission. Close after were Random Forest and Super Learner, but after further testing with Random Forest, it failed to surpass XG Boost, although it came quite close. The Super Learner matched XGBoost on training after some tuning of its models but had quite a slower runtime thus it was not chosen for more tests. Voting also performed quite good both on the training and test set but its accuracy score proved to be bottlenecked.

Tuned XGBoost	98.212%	98.2%
Tuned Super Learner	97.2%	97.8%

Initially, PCA was tested on each approach, and improved the results for the lower scoring ones, whereas the top scoring ones had worse results both on training and on testing sets. Thus, PCA was later dropped, in favor of higher accuracy, despite the considerable increase in speed during computation, especially for the Voting and Super Learner classifiers.

Next, we will take a quick look at the confusion matrices of the top algorithms for comparison.

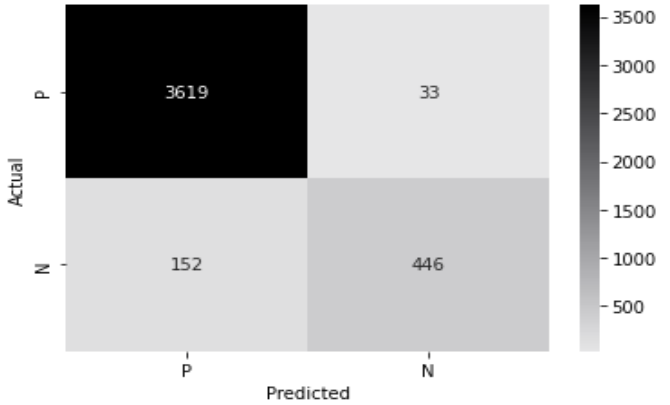

Figure 4. Gradient Boost Confusion Matrix

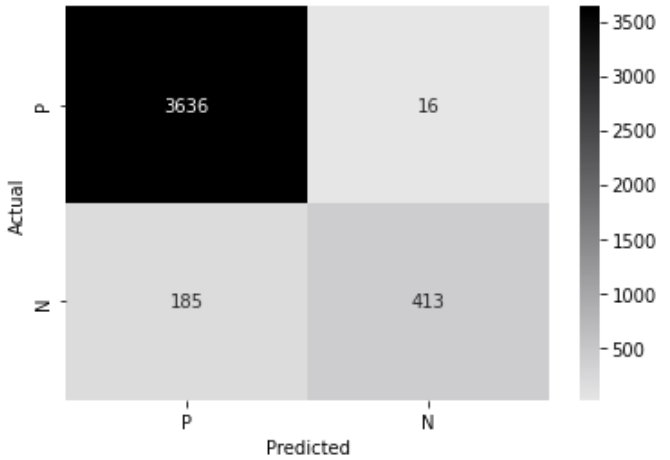

Figure 5. Random Forest Confusion Matrix

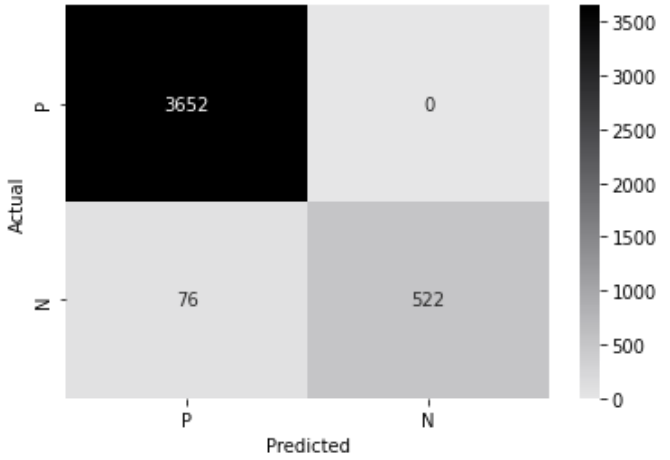

Figure 6. Tuned XGBoost Confusion Matrix

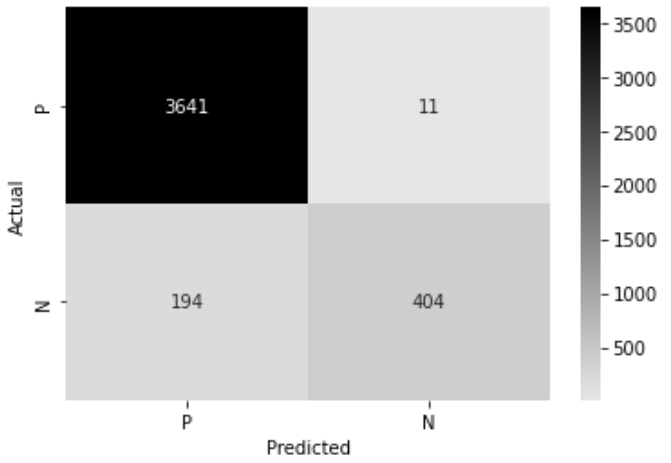

Figure 7. Voting Confusion Matrix

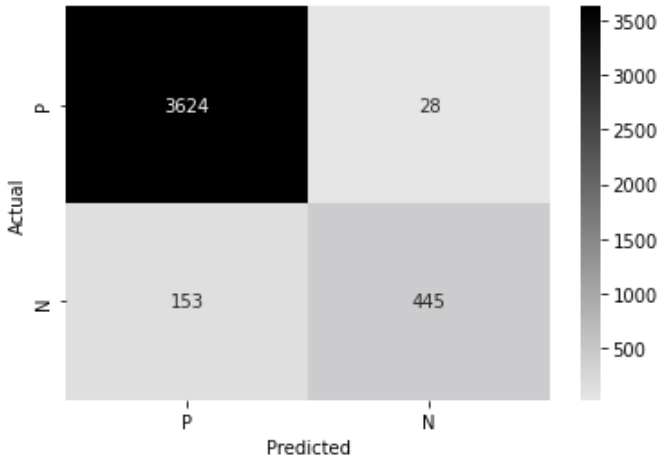

Figure 8. Super Learner Confusion Matrix

As we can see from the figures, the only approach with zero (0) false positives was the tuned XGBoost. This means that the algorithm would never mistake a loyal customer being a "churn" one, which could provide the telecommunications company with valuable information and confidence to move on to better loyalty reward programs. Unfortunately, it also failed to recognize 76 loyal customers and treated them as ones that would churn. Despite that, XGBoost was still the best in that area as well, compared to the other top performing algorithms.

Voting was the second best in False Positive rates, with only 11 misclassified in contrast to 33 for Gradient Boost, 28 for Super Learner and 185 for Random Forest.

Gradient Boost and Super Learner were very close in general, with almost identical False Negative rates

Comparing XGBoost with the other approaches, we can see that every approach did almost as good on identifying True Positives but XGBoost was quite ahead on identifying True Negatives as well, something all other approaches were weak at, especially Voting.

VII. Conclusions

Tackling overfitting proved to play a major role in this problem. Many algorithms tested had extremely high accuracy, but when their predictions were submitted, it was obvious that overfitting was a big problem. There was a variance on 4-8% accuracy which meant that measures needed to be taken. This was done by using Cross Validation and to be more precise, StratifiedKFold which helped with the imbalance of the target value ("churn"). In the case of Neural Networks, Dropout layers were added which crippled the model quite a bit but at least we had a clearer vision of its prediction capabilities and generalization.

The Voting Classifier seemed to be quite promising despite it not being in the top two performing approaches. There were still areas left to investigate in this approach and tuning to be made, although due to its nature, the tuning process would be a long and tedious one having to tune each model of the committee and to go even further test how a tuned model would perform with the other committee members. The potential is certainly there.

Implementing the Super Learner approach was something completely unknown before this challenge. It seemed to have the same potential and tuning capabilities as the Voting Classifier when used through ML-Ensemble with some added complexity due to the meta-model, which would offer additional optimization options. The library itself has a few hyperparameters that can be tuned on the super learner. Overfitting was also very easy to tackle using the built-in cross-validation of the SuperLearner class. What was surprising is the fact that, when the regular XGBoost model was swapped for the tuned one, the learner had worse results, which leads to further investigation on the correlation of the models comprising the learner.

Finally, Boosting in a very well-known technique for a reason. Both Gradient and XG Boost performed very good without any tuning. Ada Boost had a slightly lower accuracy score but was still better than most algorithms tested. Weights play a major role in feature selection and manipulation so it is quite logical that an approach which computes and readjusts the weights of features constantly by learning from previously misclassified samples, would be able to have higher accuracy. It all started when Kearns and Valiant asked the question: "Can a set of weak learners create a single strong learner?" [13][14]. For this challenge, the results speak for themselves.

References

[1] Ayon Dey / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 7 (3) , 2016, 1174-1179

[2] M. Welling, "A First Encounter with Machine Learning"

[3] M. Bowles, "Machine Learning in Python: Essential Techniques for Predictive Analytics", John Wiley & Sons Inc., ISBN: 978-1-118- 96174-2

[4] https://en.wikipedia.org/wiki/Machine_learning

[5] Russell, Stuart J.; Norvig, Peter (2010). Artificial Intelligence: A Modern Approach (Third ed.). Prentice Hall. ISBN 9780136042594.

[6] Riehle, Dirk (2000), Framework Design: A Role Modeling Approach (PDF), Swiss Federal Institute of Technology

[7] https://colab.research.google.com/

[8] https://en.wikipedia.org/wiki/Project_Jupyter#Colaboratory

[9] https://www.optimove.com/resources/learning-center/customer-churn-prediction-and-prevention

[10] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[11] https://machinelearningmastery.com/super-learner-ensemble-in-python/

[12] https://machinelearningmastery.com/implementing-stacking-scratch-python/

[13] Michael Kearns(1988); Thoughts on Hypothesis Boosting, Unpublished manuscript (Machine Learning class project, December 1988)

[14] Michael Kearns; Leslie Valiant (1989). Crytographic [sic] limitations on learning Boolean formulae and finite automata. Symposium on Theory of Computing. 21. ACM. pp. 433–444. doi:10.1145/73007.73049. ISBN 978-0897913072.