Bilkent University

Department of Computer Engineering

CS 464 - Introduction to Machine Learning

Final Report

MÜŞTEBA ANIL ÖNDER

21604090

Introduction

In today's world, loans have a wide impact on the profits of banks. In order to maximize their profit, banks need to give loans to the right candidates. In order to decide whether a loan should be given to a new candidate, we used a dataset about historic customer behavior based on what banks have observed. For the machine learning models, we used XGBoost and Deep Feedforward Networks models to predict who is riskier on a sample dataset. After trying different methods to increase the performance of the models, we measured the accuracy and F1 score metrics and analyzed which model is better. Lastly, we answered whether this model can be used in real life, namely is its performance enough to be used in the sector.

Problem Description

Predicting risky people about the loan is important for banks. The reason is that banks lose money if customers defaulted on a loan. This is not the only disadvantage of giving a loan to a risky candidate. Giving a loan to a customer means eliminating another candidate in the loan process since resources are scarce. In order to prevent this, most of the banks approve a loan after additional validation processes. However, it is still not effective enough.

So, in order to improve the lending system of banks, we want to process the previous data of the bank and predict the new candidate's possible behavior on a loan, like whether he/she will default on the loan or not. With the help of the machine learning approaches, we will use, we hope that banks will minimize their losses due to the defaults on loans

Methods

Dataset

Dataset can be found here:
https://www.kaggle.com/subhamjain/loan-prediction-based-on-customer-behavior

There are 12 features which define a customer: income, age, experience, profession, married status, house ownership, car ownership, current job years, current house years, city, state along with the risk flag indicating whether he/she defaulted on a loan or not.

Data Preparation

Before training the models, data should be prepared for selected models. In our dataset, there are six features that needed to be converted to numerical values

because in both DFF and XGBoost, categorical labels are not accepted in the process of training. As a first step; “Married/Single”, “House_Ownership” and “Car_Ownership” feature labels converted to binary labels. Then we removed the unused features from the dataset such as “ID”. The remaining categorical features, “Profession”, “CITY” and “STATE”, have too many categories.

Encoding

To use them in our models, we needed to encode them using label encoding or one-hot encoding. Our categorical features were nominal which means they do not have an order, therefore, using one-hot encoding is more advantageous because it treats each category as a new feature.

One-Hot Encoding

We first used only one-hot encoding however it came with some disadvantages. One-hot encoding creates dummy features for each unique category, then each sample is marked as one for that dummy feature if the sample is in that category and the rest of the dummy columns marked as zero. This means we obtained far too many features with sparse labels on our dataset. Excessive number of features also decreased the performance of our algorithms and models. We also needed to use feature selection algorithms to decrease the number of features we obtained, however, that means we will not use some categories in the prediction which may create bias(overfit) in the system because the methods will give importance to the features according to the training data but not the test data. Consequently, we also used label encoding to compare two encoding techniques.

Label Encoding

In label encoding, categories in the categorical features are labeled starting from 0 to the number of unique categories - 1. In this way, the number of features remains the same however methods may relate some categorical features as if they are ordinal. For example, assuming New York is labeled as 1 and Texas is labeled as 2, the algorithm may think that if the label is less than 3, the prediction is 0, although New York and Texas are not related to each other. This problem may decrease the metrics of the methods.

Feature Selection

After encoding the features we have obtained too many features in the one-hot encoded dataset but not in label encoded dataset. To obtain fair comparison results we formed a feature selection system using mutual information.

Mutual Information

After we calculated mutual information for each feature we sum them up and obtained total mutual information that our features have according to the training

labels and try to create datasets that have the best features containing %70, %80 and %90 of the total mutual information.

Feature Importance

This feature selection method is only used in the XGBoost algorithm because it is provided by it. XGBoost algorithm provides feature importance for each feature according to two metrics; gain and weight. Gain based feature importance is calculated by the sum of the gain values when that feature is used in a tree. Weight based feature importance is calculated by the sum of the usages of a feature in a tree as a decision node. We used weight based feature importance in the rest of the experiments because it performed better in our models. To compare feature selection techniques, we also used the same feature selection system on feature importance. We created datasets that have the best features containing %70, %80 and %90 of the total feature importance.

Normalization

We used normalization only for the Neural Networks method because decision tree based algorithms do not perform any better under normalization. We performed two normalization techniques for comparison; min-max normalization, tanh-estimator normalization.

Tanh-estimator Normalization

Tanh-estimator is an efficient normalization technique used in the Neural Networks machine learning models proposed in 1986 [3]. We used a little modified version of the original technique which was proposed in 2011 [4]. According to the papers that propose this technique, tanh-estimator is efficient, robust, not sensitive to outliers and outperforms other normalization techniques in Neural Networks. Only problem with this technique is when it is used on the binary sparse data such as one-hot encoded dataset, the physical size of the dataset increases dramatically because it extends binary values to the float values between -1 and 1. Thus, we only used tanh-estimator on the label encoded datasets in the neural networks.

Min-Max Normalization

Min-max normalization is used on one-hot encoded datasets for neural networks. Although max-min normalization is not the best normalization technique for neural networks, it performs better than non-normalized data. One advantage over tanh is, it does not increase the size of the data because it does not change binary values.

XGBoost

After preparing the datasets in the first part, for each of the pre-processed datasets, we trained an XGBoost model and tried to tune some of the hyperparameters of it. Design decisions we made can be seen below.

- We used **logistic regression for binary classification** for the learning objective of XGBoost since our label is binary.
- For the **scale_pos_weight** parameter, we used the ratio between negative and positive samples in the dataset, which is nearly 7. The reason is that there is an imbalance between negative and positive samples and we had to train the model in a way that it would try to predict more positives.
- In the beginning of the project, we used Grid Search for the parameter tuning technique. However, after doing some research and seeing the long training times, we thought that Randomized Search would be a better technique since it tries random combinations of the parameters and the number of tried combinations can be set to any value. In this way, we decreased the training time a lot because Grid Search was trying every combination of the parameters. On the other hand, we thought that Randomized Search would help avoid overfitting because it does not pick the **best** possible combination, it just picks the best combinations over some random combinations.
- We used below values for the initial values for the parameters. We picked these values after reading documentation of the XGBoost and searching different resources [1].
    - max_depth: [3, 5, 7, 9]
    - min_child_weight: [0, 2, 4, 6]
    - gamma: [0, 0.5, 1, 1.5]
    - reg_lambda: [0, 3, 6, 9]
- For the evaluation metric, we used **AUCPR (Area Under Curve Precision-Recall)**. The reason is that our dataset is unbalanced with respect to negative and positive sample ratios (Negative samples are 7 times more than positive samples). We decided not to use **roc_auc** since we care about the positive predictions more [2].
- For the iteration count of Randomized Search, we set it to 50. The reason is that there are around 250 combinations of the parameter values therefore it would probably pick a good combination in 50 random combinations.
- We set the **cross-validation** count to 3 so that Randomized Search split the data into 3 chunks and fit the model 3 times for every parameter combination.
- Since there are not so many values (just 4) for the parameters, we tried an additional approach on top of the Randomized Search. At the end of Randomized Search, if one of the optimum parameters is the biggest one among the possible values, we set the possible values for that parameter to bigger values and rerun the Randomized Search again. For example, if gamma was 1.5 in the first iteration, we changed the values to [2, 2.5, 3, 3.5] and rerun the algorithm. However, we didn't apply this approach to

max_depth because at first, we didn't limit it and it increased up to 47, which is a very bad value since it causes a lot of overfitting. So, instead, we limited it to a maximum of 9.

- After getting the optimum parameters from the tuning process, we trained a new XGBoost classifier with these optimum parameters. In addition to that, we increased the **n_estimators** and decreased **learning_rate (eta).** The reason is that in this setup, the performance of the model would not increase dramatically in every weak learner tree and because the number of weak learner trees (n_estimators) is high, model would converge to a better result.
- In the end, we get results for validation metrics. Moreover, we used the test dataset in order to analyze how the trained model would perform in unseen data.

DFF

Tensorflow and Keras are used for training DFF. It uses dropout layers with 20% probability between hidden layers, also instead of adam optimizer, special optimizer with learning rate of 0.1 and momentum of 0.9 is used as recommended. Epoch is kept at 50 for each training dataset and batch size is fixed at 15. In hidden layers only relu activation is used and at the output layer sigmoid function is used, which gives a number between 0 and 1 and during testing we set a threshold value, such as 0.5 and if the prediction is bigger than 0.5, we classify as 1 and 0 otherwise. The reason we used dropout is because we observed that no improvement takes place after some epoch value and loss and accuracy stays the same. To introduce some randomness to the system would make it better, we thought. Also, in order to avoid vanishing gradient problems, we used kernel constraints upon layers, also it helps with overfitting.

We used the same datasets with the XGBoost part.

Results

XGBoost

From now on, we will use these abbreviations:

- **OHE:** One-Hot Encoding
- **LE:** Label Encoding
- **FI:** Feature Importance
- **MI:** Mutual Information

In the experiments we have done for the XGBoost model, we tried to answer these questions:

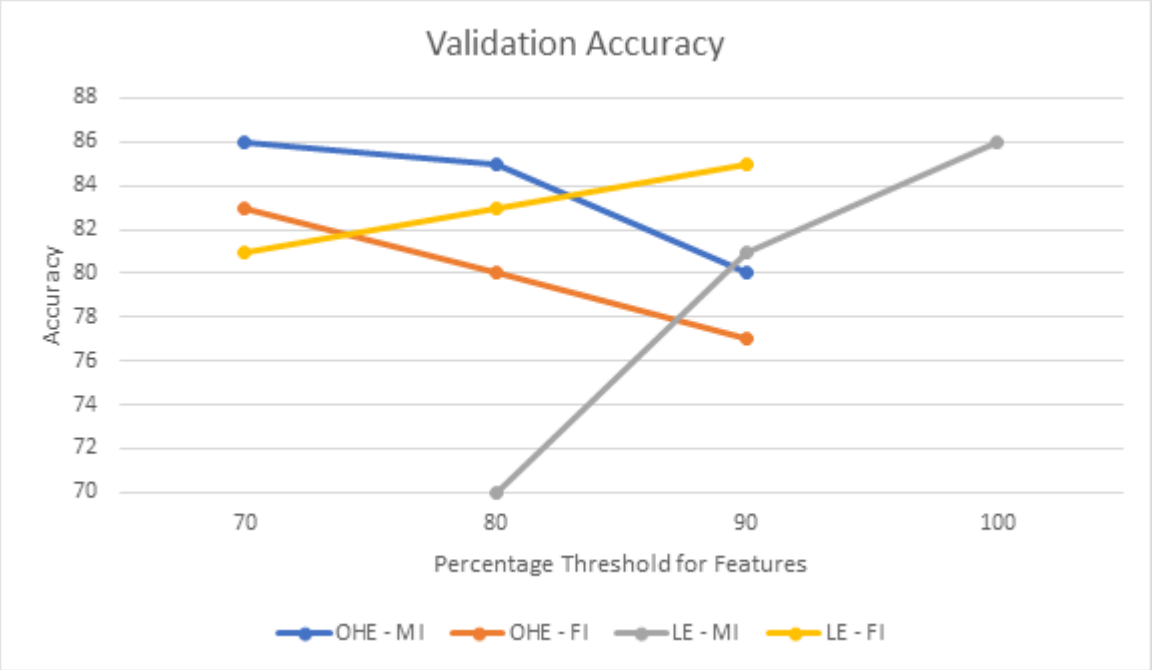
1. For processing categorical features, which encoding type gives better results: One-hot encoding or label encoding.
2. For selecting a subset of features, which criteria give better results: Mutual Information or Feature Importance (by weight)
3. For the number of features in the dataset, how many feature numbers give better results. In order to analyze this, we selected 70%, 80%, 90% of the features according to the corresponding feature selection criteria.

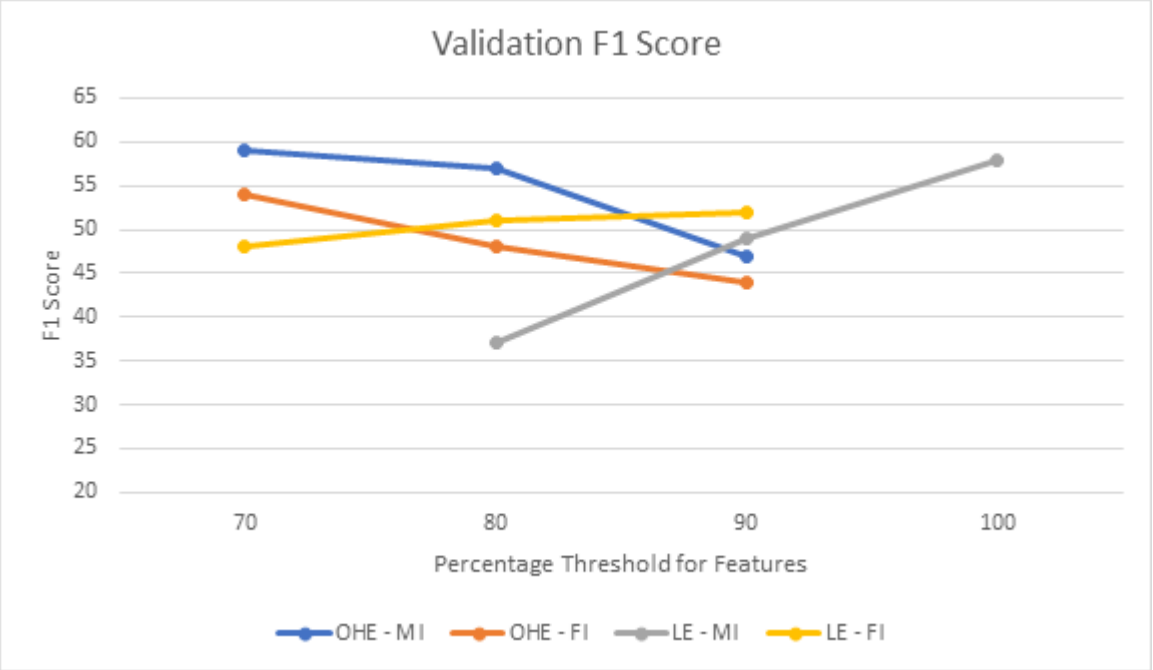
Results for the validation set

Encoding	Feature Selection Criteria	Feature Percentage	Accuracy	F1 Score
One-Hot	Mutual Information	90%	80	47
One-Hot	Mutual Information	80%	85	57
One-Hot	**Mutual Information**	**70%**	**86**	**59**
One-Hot	Feature Importance	90%	77	44
One-Hot	Feature Importance	80%	80	48
One-Hot	Feature Importance	70%	83	54
Label	Mutual Information	100%	86	58
Label	Mutual Information	90%	81	49
Label	Mutual Information	80%	70	37
Label	Feature Importance	90%	85	52
Label	Feature Importance	80%	83	51
Label	Feature Importance	70%	81	48

From the above table, we can see that the best combination of the answers to the questions described in the first part is **the One-Hot Encoded dataset where 70% of the features are used according to Mutual Information.**

The visualized version of the above table can be seen below. For better analysis, we split the accuracy and F1 Score metrics to separate figures.





As you can see from the above figures, for the One-Hot Encoded datasets, metrics are decreasing when the percentage threshold increases. It is the opposite for Label Encoded datasets: metrics increase as the percentage threshold increases.

Results for the test set

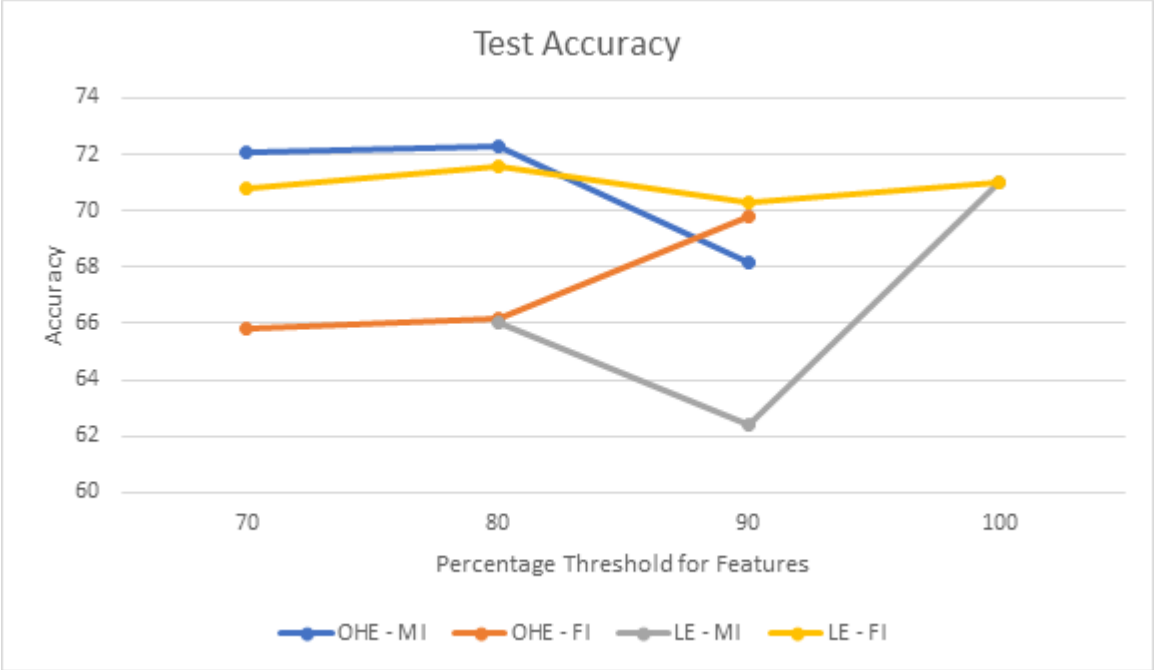
Encoding	Feature Selection Criteria	Feature Percentage	Accuracy	F1 Score
One-Hot	Mutual Information	90%	68.2	16.1
One-Hot	Mutual Information	80%	**72.3**	15.4

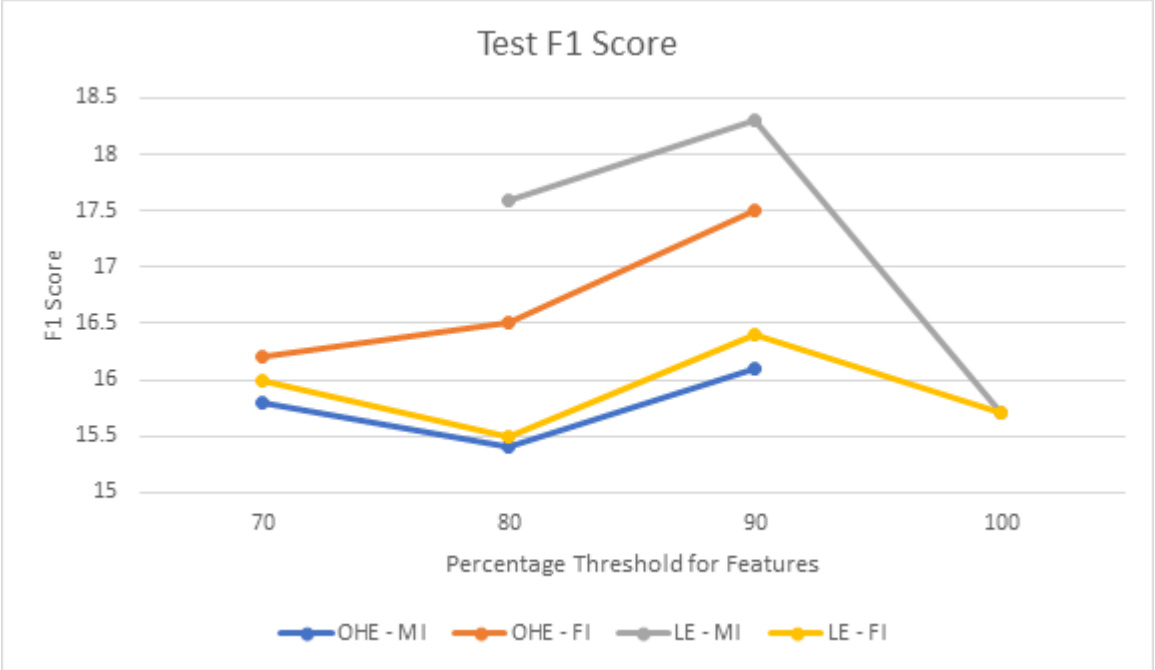
One-Hot	Mutual Information	70%	72.1	15.8
One-Hot	Feature Importance	90%	69.8	17.5
One-Hot	Feature Importance	80%	66.2	16.5
One-Hot	Feature Importance	70%	65.8	16.2
Label	Mutual Information	100%	71	15.7
Label	Mutual Information	90%	62.4	**18.3**
Label	Mutual Information	80%	66	17.6
Label	Feature Importance	90%	70.3	16.4
Label	Feature Importance	80%	71.6	15.5
Label	Feature Importance	70%	70.8	16

For accuracy metric, the best combination is **One-Hot/Mutual Information/80%** by 72% accuracy**.** However, we can see that the accuracies of these different combinations are less than the ones for the validation datasets. This is probably because of the overfitting caused by the XGBoost hyperparameters. On the other hand, selecting features according to the training dataset may lead to overfitting the training set.

For the F1 Score, the best combination is **Label/Mutual Information/90%** by 18%. This is very low compared to the validation metrics, again probably because of the overfitting.

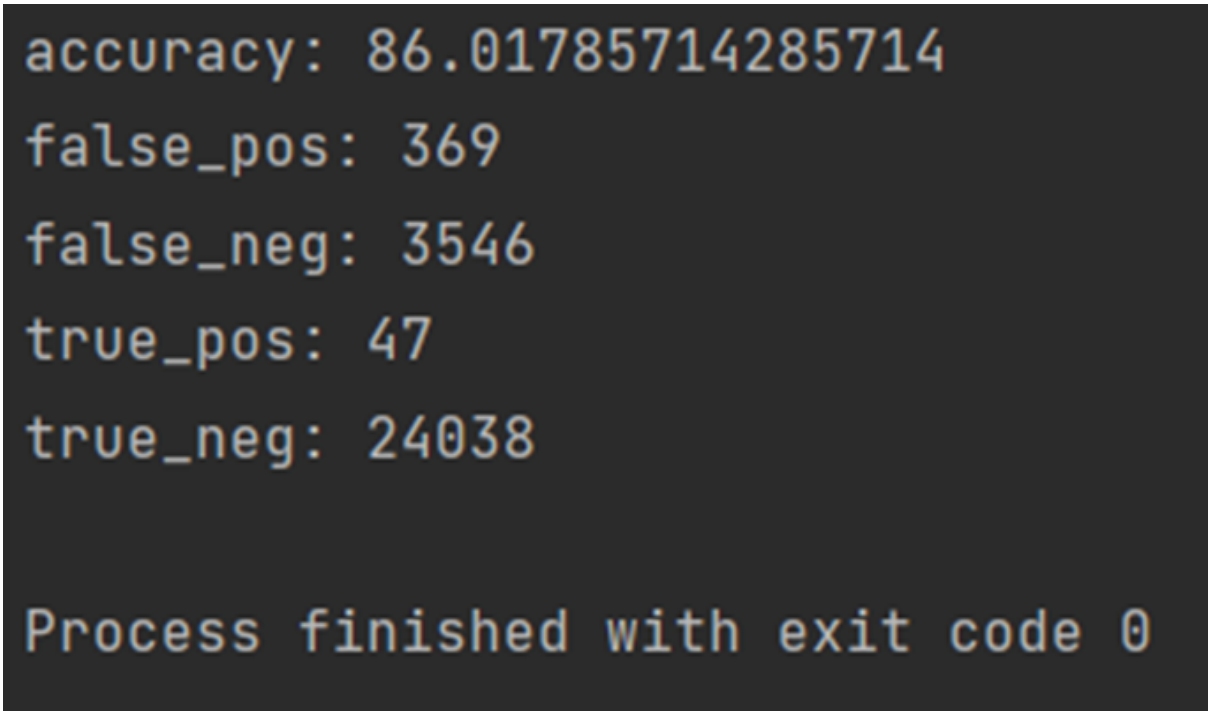
The visualized version of the above table can be seen below. For better analysis, we split the accuracy and F1 Score metrics to separate figures.





DFF

When 90% mutual information one hot encoded dataset is used for training, the following result is achieved from testing. Accuracy is 86% and this is the best we can get. Other datasets give about 85% accuracy.



Discussion

XGBoost

According to the results we have achieved, label encoding and one-hot encoding behaves differently.

Label Encoding and Feature Percentage

In validation tables:

- Label encoding accuracy and F1 score increases as the number of features increase because the number of total features in the label encoding is very few. Less numbers of features does not perform any better in the label encoding.
- One-hot encoding accuracy and F1 score decreases as the number of features increase because the number of total features in the one-hot encoding is excessively many and if we use all of the features, we introduce very high noise to the system. Decreasing the number of features up to some threshold makes XGBoost perform better.

In test tables:

- Both label encoding and one-hot encoding behave opposite to the validation results.
- Label encoding accuracy and F1 score decreases on the test data as the number of features goes to %100 because if all features are used in label

encoding XGBoost learns too much from the training data and predicts test data accordingly. This shows using %100 of the features creates overfit on the model.

- One-hot encoding accuracy and F1 score increases on the test data as the number of features goes to %90. This is probably because the best features that we selected are according to the training data and if a category is not intensively present in the training data but it exists in test data, this category is ignored while prediction.

Feature Selection

Feature selection methods can be compared according to obtained results on the test sets. Mutual information selection methods had an advantage on both metrics. According to accuracy, mutual information performed better, achieving %72.3 accuracy. According to the F1 score, again mutual information performed better, achieving %18.3 score.

After observing low performance on the test sets, we tried to avoid overfitting by manually changing the optimum hyperparameters of the XGBoost model. These are the parameters we tried to optimize:

1. **colsample_bytree:** This parameter determines the ratio of used features among the all features. For example, if it is 0.5, every weak learner tree uses a different set of features whose number is one half of the total feature count. Therefore, we thought that decreasing this parameter will help avoiding overfitting since the model will have less bias on a specific feature since it will not be used on every weak learner.
2. **subsample:** This parameter determines how many samples will be used on every weak learner tree. If it is 1, all samples will be used. So, we thought that if we decrease this parameter, the model will not overfit the training dataset since a specific sample will not be used on every tree.
3. **min_child_weight:** This parameter determines the minimum threshold for the sum of instance weights on a leaf node of a tree. If this parameter is high, it means that there will be no leaves with a few samples. This helps avoid overfitting because the model will not try to separate each sample into different leaves.

However, after trying different values for these parameters, we couldn't increase the performance of the model on test data. It was still overfitting the training dataset.

DFF

Conclusions

XGBoost

We have experimented on different data types, encoding types, feature selection methods, hyperparameters to obtain the results.

- For different encoding types, behavior of the model is different and changes when the number of features change.
- For feature selection algorithms, we have seen that mutual information performs better than feature importance although feature importance is provided by the XGBoost itself.
- For tuning hyperparameters, we have seen that it is easy to overfit to the training data by tuning hyperparameters. Although we even manually tried to change the parameters to decrease overfitting, we were not able to do so.

On the other hand, XGBoost is an efficient and high performance machine learning algorithm which may perform better in different classification tasks.

DFF

We found that one hot encoded dataset performs better than label encoded ones. However, the problem is that the dimension of the input layer increases. Solution could be frequency encoding instead of label encoding. We learned that increasing the number of hidden layers does not increase accuracy and precision. Using a dropout layer did not improve performance at all.

References

[1] “XGBOOST parameters: XGBoost parameter tuning,” *Analytics Vidhya*, 23-Nov-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/. [Accessed: 26-Dec-2021].
[2] Jakub Czakon Mostly an ML person. Building MLOps tools, J. Czakon, Mostly an ML person. Building MLOps tools, and F. me on, “F1 score vs ROC AUC vs Accuracy Vs PR AUC: Which evaluation metric should you choose?,” *neptune.ai*, 01-Dec-2021. [Online]. Available: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc. [Accessed: 26-Dec-2021].
[3] J. Law, “Royal Statistical Society Publications,” *Royal Statistical Society*, 05-Dec-2018. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/10.2307/2987975. [Accessed: 26-Dec-2021].
[4] L. Latha, “Efficient approach to normalization of multimodal biometric scores,” *IJCA*. [Online]. Available: https://www.ijcaonline.org/archives/volume32/number10/3952-5530.