

CS 412/512 – Term Project

Machine Learning Pipeline for Credit Risk Analysis with the

German Credit Data

Group Members:

Ömer Said Öztürk – 26471

Ahmet Mihça Aydın – 26753

Göktuğ Korkulu –27026

Sadi Gülbey – 27047

Musa Sadık Ünal – 28060

Date:

06.06.2023

Semester: 2023 Spring

Sabancı University



1. Abstract	2
2. Introduction	2
3. Dataset	2
4. Methodology	4
4.1. Logistic Regression (LR) Modal	5
4.2. Decision Trees (DT) Classifier Modal	5
4.3. Multilayer Perceptron (MLP) Classifier Modal	6
4.4. K-Nearest Neighbors (KNN) Classifier Modal	7
4.5. Support Vector Machines (SVM) Classifier Modal	7
4.6. Extreme Gradient Boosting (XGB) Classifier Modal	7
5. Experiments	8
5.1. Logistic Regression (LR) Modal Hyperparameter	8
5.2. Decision Trees (DT) Classifier Modal Hyperparameter	10
5.3. Multilayer Perceptron (MLP) Classifier Modal Learning Curve	10
5.4. K-Nearest Neighbors (KNN) Classifier Modal Hyperparameter	11
5.5. Extreme Gradient Boosting (XGB) Classifier Modal Hyperparameter	12
5.6. Summary Table	14
6. Discussion	15
7. Bonus	16
8. Conclusion	17

1. Abstract

To predict credit risk, we have tried Logistic Regression, Decision Trees, Multilayer Perceptron, K-Nearest Neighbors, Support Vector Machines (SVM), and Extreme Gradient Boosting (XGBoost) as primitive algorithms, which were easy to train and implement. We have observed their performances after hyperparameter tuning for each. After deciding our primitive models, we have created classifier combinations with Mixture of Experts and Voting algorithms. These algorithms were created to provide higher accuracy with trained primitive models. The best one among primitives was Extreme Gradient Boosting (XGB) with 0.74333 validation accuracy, resulting in 79% test accuracy. However, among complicated models, the best one was Mixture of Experts with 81% validation accuracy and 77% test accuracy.

2. Introduction

This CS412-Machine Learning Term Project aims to acquire practical knowledge and experience in the ML project pipeline. The project focuses on the application of various classification algorithms to solve the real-world problem of credit risk assessment, using the widely recognized German Credit (Statlog) Dataset as a benchmark.

The whole dataset consists of 1000 rows and 10 columns. The last column, “Risk” represents the class label, and it is labeled as either **Good** or **Bad**. The remaining 9 columns correspond to the 9 features of the dataset namely **age**, **sex**, **job**, **housing**, **saving accounts**, **checking accounts**, **credit amount**, **duration**, **purpose**

Based on the available features, the ultimate goal is to predict whether a given loan applicant will likely have good or bad credit. The importance of the problem comes from its goal. Estimating credit risk assessments would decrease the workforce in the finance industry, which would lead to more profit for these companies. Also, automatizing the process correctly will reduce wrong expenses, which again improves profits for such companies.

3. Dataset

The German Credit (Statlog) Dataset is a benchmark dataset for credit risk assessment. The whole dataset is provided in a single file format named *german_credit_data.csv*. It consists of 1000 rows/instances and 10 columns. The last column represents the class label, while the remaining 9 columns correspond to the 9 features of the dataset, such as age, sex, credit history, and job stability. The dataset includes 700 instances labeled as **good credit** and 300 instances labeled as **bad credit**. Figure 1 below exhibits an example of the first five instances of the dataset.

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	49	male	1	own	little	NaN	2096	12	education	good
3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	53	male	2	free	little	little	4870	24	car	bad

Figure 1: Example of the first five instances of german_credit_data.csv data

The dataset is separated into features and labels. Then, categorical features are converted into numerical values using both one hot encoding and ordinal encoding; and missing values are handled. Later, the numerical values of each feature are scaled by utilizing MinMaxScaler from the scikit library. After this preprocessing is applied, the shape of features data is (1000,17), and where label data is (1000,1). The first figure below represents the final features data, and the second shows each feature's before and after datatypes.

	Age	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Sex_female	Sex_male	Purpose_business	Purpose_car	Purpose_domestic appliances	Purpose_education	Purpose_furniture/equipment	Purpose_radio/TV	Purpose_repairs	Purpose_vacation/others
0	0.857143	0.666667	1.0	0.158303	0.000000	0.050567	0.029412	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	0.053571	0.666667	1.0	0.000000	0.500000	0.313690	0.647059	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.535714	0.333333	1.0	0.000000	0.325908	0.101574	0.117647	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.464286	0.666667	0.0	0.000000	0.000000	0.419941	0.558824	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.607143	0.666667	0.0	0.000000	0.000000	0.254209	0.294118	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 2: The first five instances of the features dataframe size of (1000, 17)

```
Age          int64
Sex          object
Job          int64
Housing      object
Saving accounts object
Checking account object
Credit amount int64
Duration     int64
Purpose      object
```

```
Age          float64
Job          float64
Housing      float64
Saving accounts float64
Checking account float64
Credit amount float64
Duration     float64
Sex_female   float64
Sex_male     float64
Purpose_business float64
Purpose_car   float64
Purpose_domestic appliances float64
Purpose_education float64
Purpose_furniture/equipment float64
Purpose_radio/TV float64
Purpose_repairs float64
Purpose_vacation/others float64
```

Figure 3: Before and after encoding the features datatypes

As the last step before conducting machine learning algorithms, we split the features and labels into train and test subsets with test size = 0.1. Therefore, shapes are:

```
X_Train = (900,17)
y_train = (900,1)
X_test = (100,17)
y_test = (100,1)
```

4. Methodology

To be able to use our algorithms and make their distributions the same we have used One Hot Encoding and Standard Scaling. In this way, we aimed to get more stable results than raw data since the data was noisy.

We have implemented **Logistic Regression**, **Decision Trees**, **Multilayer Perceptron**, **K-Nearest Neighbors**, **Support Vector Machines**, and **Extreme Gradient Boosting** models. We aimed to provide distinct models which work differently from each other since we were not able to provide a strong hypothesis about the best theory that fits the problem.

Logistic Regression is a simple and interpretable model that estimates class probabilities using a logistic function. It is efficient for large datasets but assumes a linear relationship and may struggle with non-linear problems.

Decision Trees, on the other hand, make decisions based on feature splits and handle both numerical and categorical features. They are easy to interpret but can overfit and create complex trees.

Multilayer Perceptron (Neural Networks) can learn complex patterns and handle non-linear relationships but require more data and computational resources.

K-Nearest Neighbors (KNN) is an instance-based model that assigns class labels based on the majority vote of its neighbors, and it handles non-linear relationships and supports various distance metrics.

Support Vector Machines (SVMs) aim to find a decision boundary with a maximum margin and are stable and robust, while

Extreme Gradient Boosting (XGBoost) combines weak learners into a strong model and excels in handling large datasets and complex relationships. Each model has its strengths and weaknesses, making it essential to choose the most appropriate one based on the problem at hand.

These are very distinct algorithms, and we knew that the problem is hard to set a hypothesis. That is why we wanted to try all of the aspects we could implement.

We have tested many hyperparameters for different algorithms. Below, we will explain why we prefer to tune them in our models.

4.1. Logistic Regression (LR) Modal

Best Hyperparameters: {'C': 0.1, 'max_iter': 100, 'penalty': 'l2',
'solver': 'liblinear'}

Best Validation Score: 0.7144444444444444

In the hyperparameter tuning process of the logistic regression model, we have tried C (regularization parameter), max iterations, penalty function, and solver method hyperparameters.

We first tried so many hyperparameters because of logistic regression's train time. It was lower than other algorithms' train time. Therefore trying more hyperparameters did not result in any loss of time.

The first hyperparameter, the regularization parameter, is decided since our data had high variance, which may lead to an overfitting regression model. That is why we wanted the optimal regularization parameter.

The second parameter max iterations are tried since we had more data than logistic regression's default max iterations, and our data was small for other problems. A high max iterations would lead to overfitting, as it did, and a low max iterations would result in weak performance.

The third parameter penalty was chosen since our data is complicated. The complication made us think about how we will penalize strong coefficients since one feature would dominate the others without a better accuracy.

Finally, the best solver function is needed in a complicated task. We have looked at the data, and it was not possible to estimate the best solver method.

4.2. Decision Trees (DT) Classifier Modal

Best parameters: {'criterion': 'entropy', 'max_depth': 5}

Best Validation Score: 0.7066666666666667

While we were implementing the Decision Tree model, we had two things in mind: firstly, what should be our criteria of punishment for wrong answers and how to avoid overfitting. The punishment method was important since most online algorithms resulted in the majority classifier's

accuracy results; therefore, our model should punish such biased algorithms. Also, in decision trees, it is important to provide good maximum depth to prevent overfitting in general. Moreover, we had a small dataset.

4.3. Multilayer Perceptron (MLP) Classifier Modal

Best parameters: {'activation': 'relu', 'alpha': 0.001,
 'hidden_layer_sizes': (100,), 'max_iter': 100,
 'solver': 'adam'}

Best Validation Score: 0.72

In the first part, we tried MLPs, and our results were unsuccessful with low accuracy. Therefore, we wanted to change all default parts of MLPs, to see if the problem is about MLPs hyperparameters.

The activation function was an important issue since we knew a linear activation function would not achieve what we wanted. Therefore, we tried two different activation functions: relu and logistic activation. Both of these may provide good estimations.

Our alpha regularization parameter was important to prevent the dominance of some neurons. That is why we tried different values.

We have tried different layer sizes since we did not know how complicated the model should be. We have tried many measures to serve this purpose.

Also, as in logistic regression, we thought that we should look for iteration count in a small dataset. The iteration count may lead to a gradient of accuracy scores, and we wanted to maximize accuracy.

We had two solver methods: **adam** and **sgd**. As far as we read, sgd may perform better in noisy and large datasets. However, we had a noisy dataset but not a large one. Therefore, we gave it a try.

An addition to these parameters, we tried a different number of epochs with another library, keras. Epoch size effectively changes MLP algorithms' performances, therefore, it is an important topic to look at.

4.4. K-Nearest Neighbors (KNN) Classifier Modal

Best Hyperparameters: {'n_neighbors': 5}

Best Validation Score: 0.6622222222

We knew that KNN would not solve our problem after we trained and validated our first KNN model. However, KNN-like models depend on their primary hyper parameter much. That is why we tried its primary hyper parameter: neighbors, which resulted in a failure.

4.5. Support Vector Machines (SVM) Classifier Modal

Best Hyperparameters: {'C': 1, 'gamma': 'scale', 'kernel': 'poly'}

Best Validation Score: 0.7133333333333333

We have already discussed the necessity of regularization in our models. However, the gamma hyperparameter is firstly tried here, which decides the shape and flexibility of the decision boundary. It was important to try since we knew that our data's shape is not something usual. The kernel parameter is how we map the input to features. We knew that linear mapping would not work for such a complex problem, that is why we tried others.

4.6. Extreme Gradient Boosting (XGB) Classifier Modal

Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 200}

Best Validation Score: 0.7411111111111112

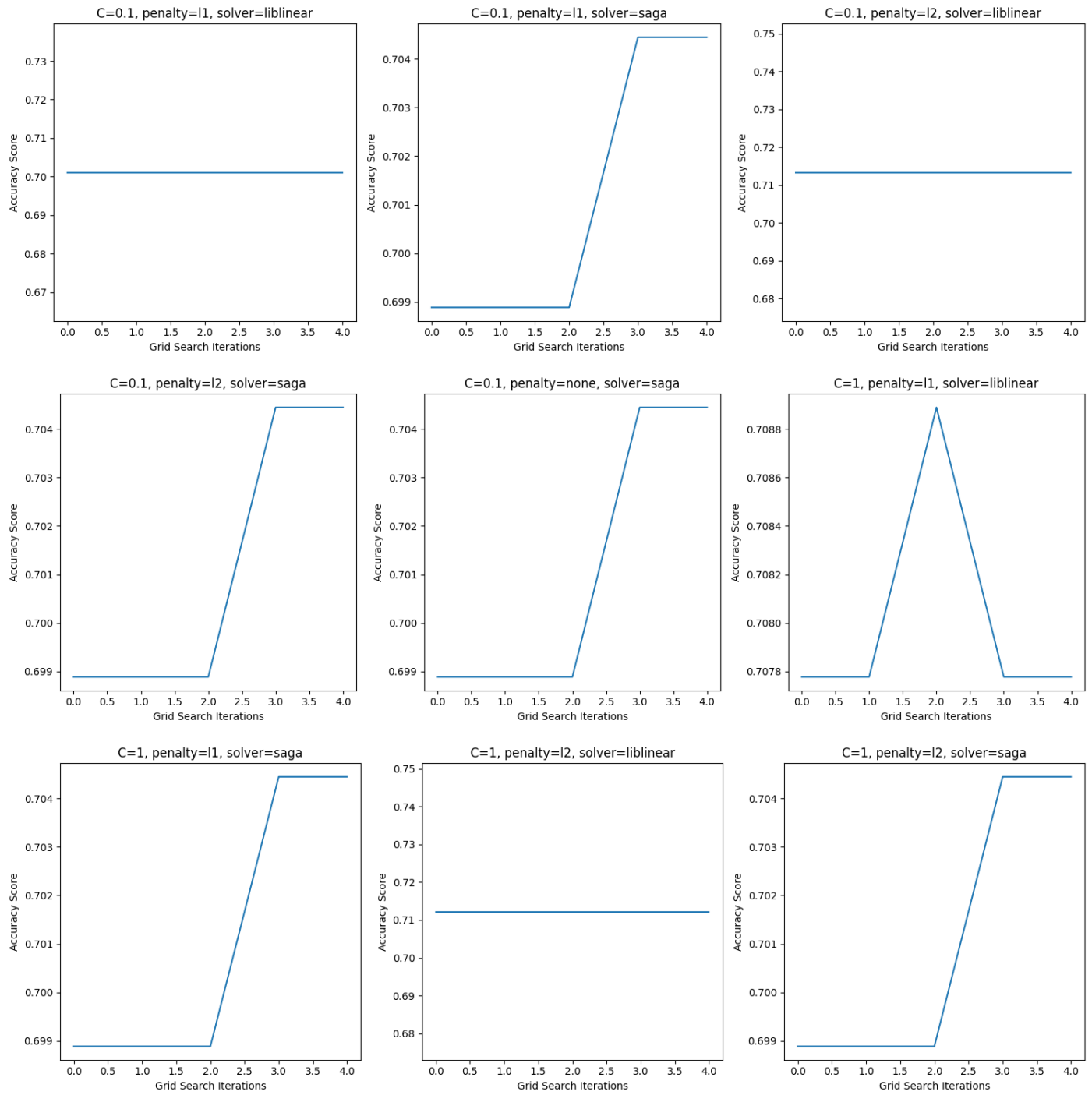
We decided that some features may be affecting too much during the training stage. That is why we wanted to control the learning rate in our XGB model.

As explained earlier, max depth is tuned since it would lead to overfitting or underfitting.

Also, the number of estimators was an important factor to idealize a complex model in a small dataset, that is why we tried a couple of them.

5. Experiments

5.1. Logistic Regression (LR) Modal Hyperparameter



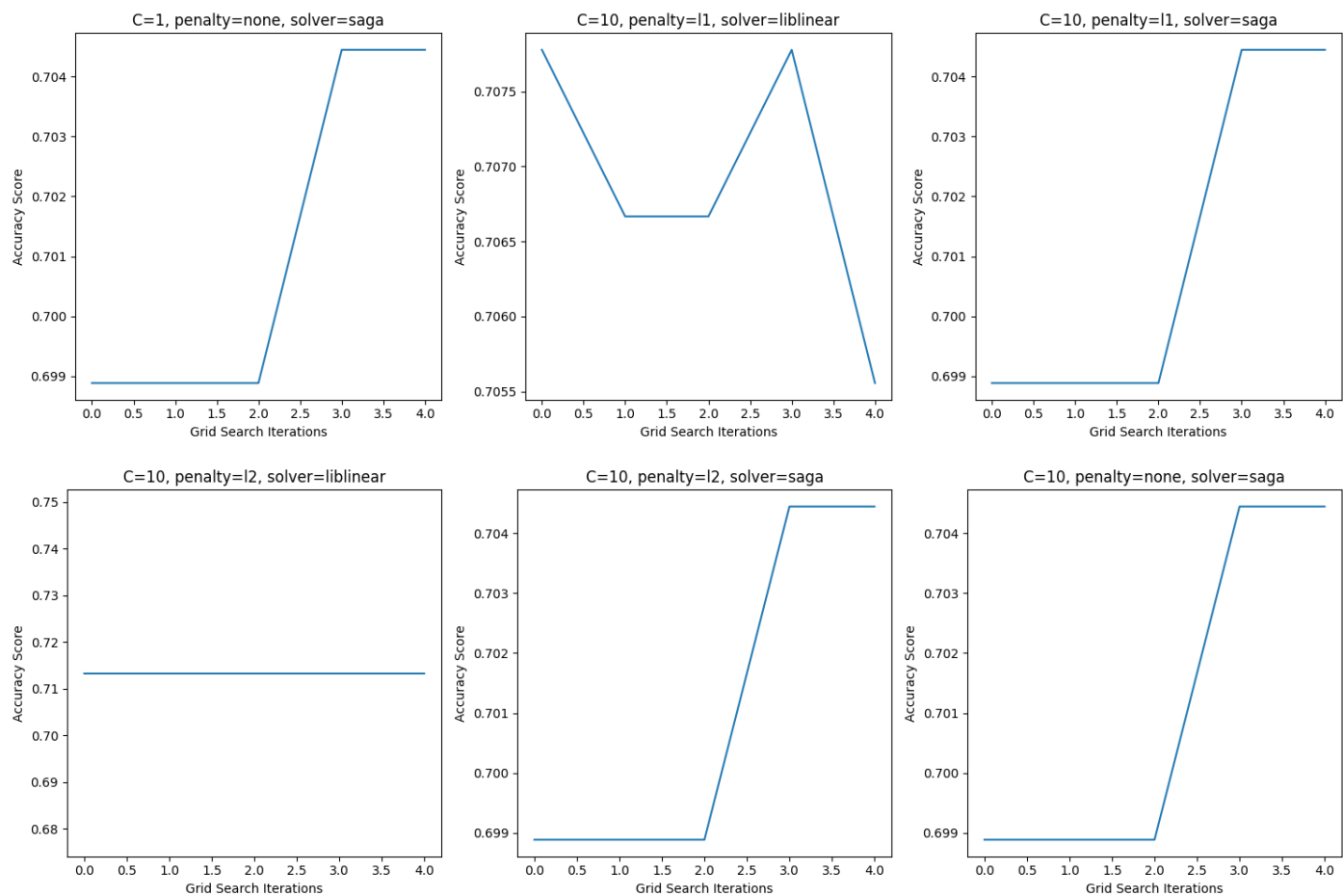


Figure 4: LR Figure

5.2. Decision Trees (DT) Classifier Modal Hyperparameter

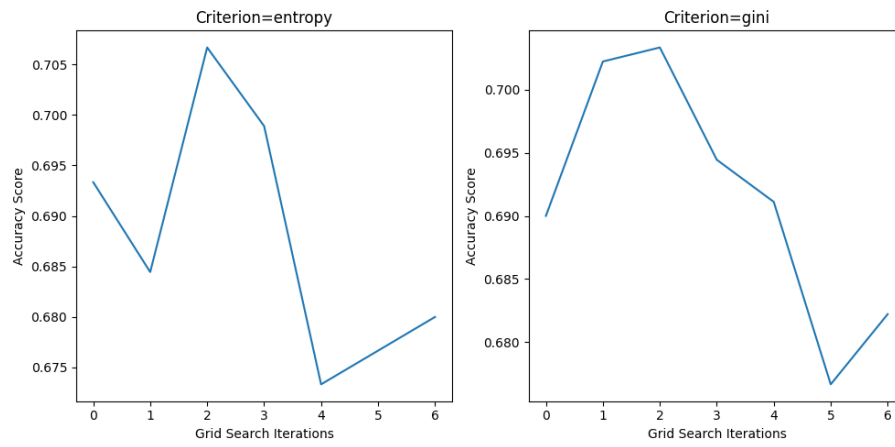


Figure 5: DT Figure

5.3. Multilayer Perceptron (MLP) Classifier Modal Learning Curve

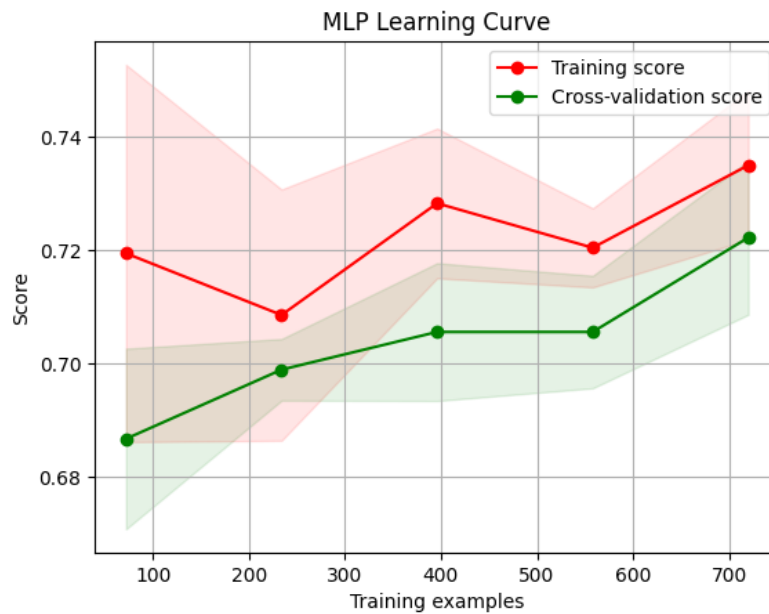


Figure 6: Learning curve for MLP algorithms

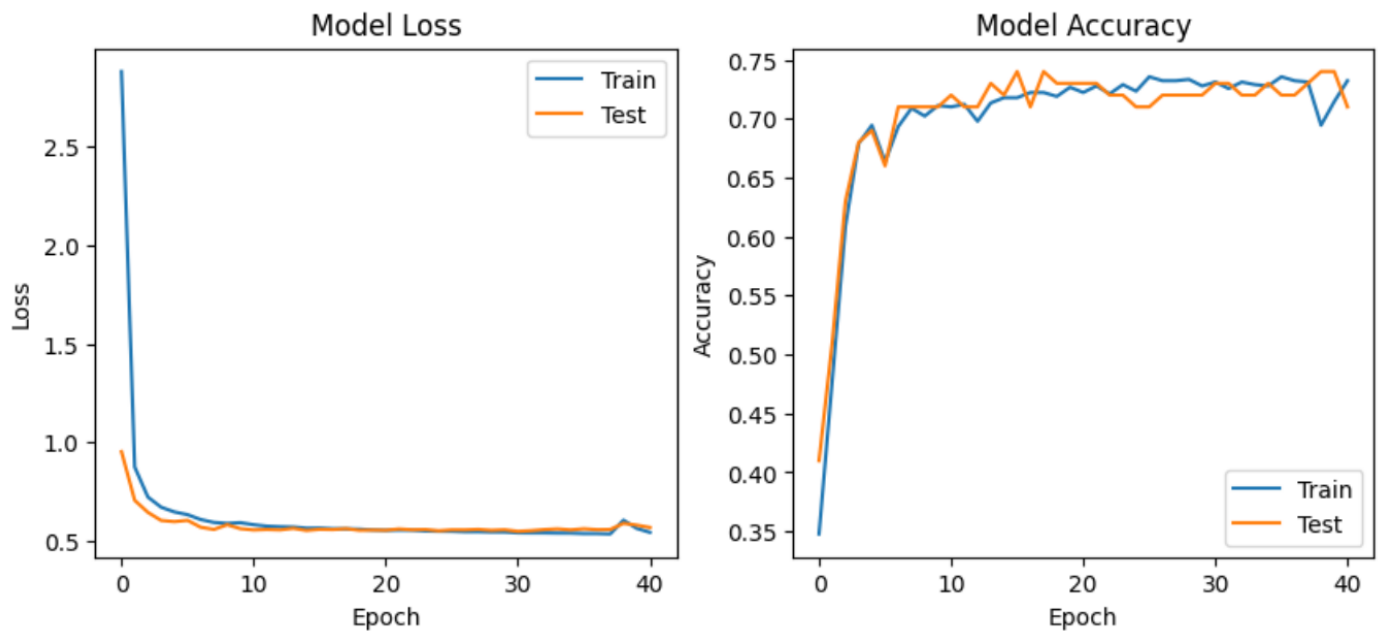


Figure 7: Accuracy and loss for different epoch numbers

5.4. K-Nearest Neighbors (KNN) Classifier Modal Hyperparameter

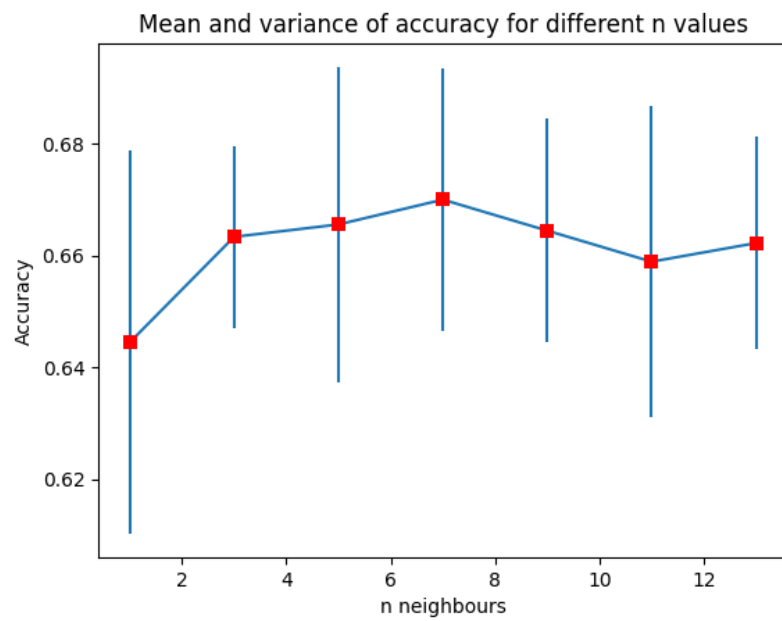
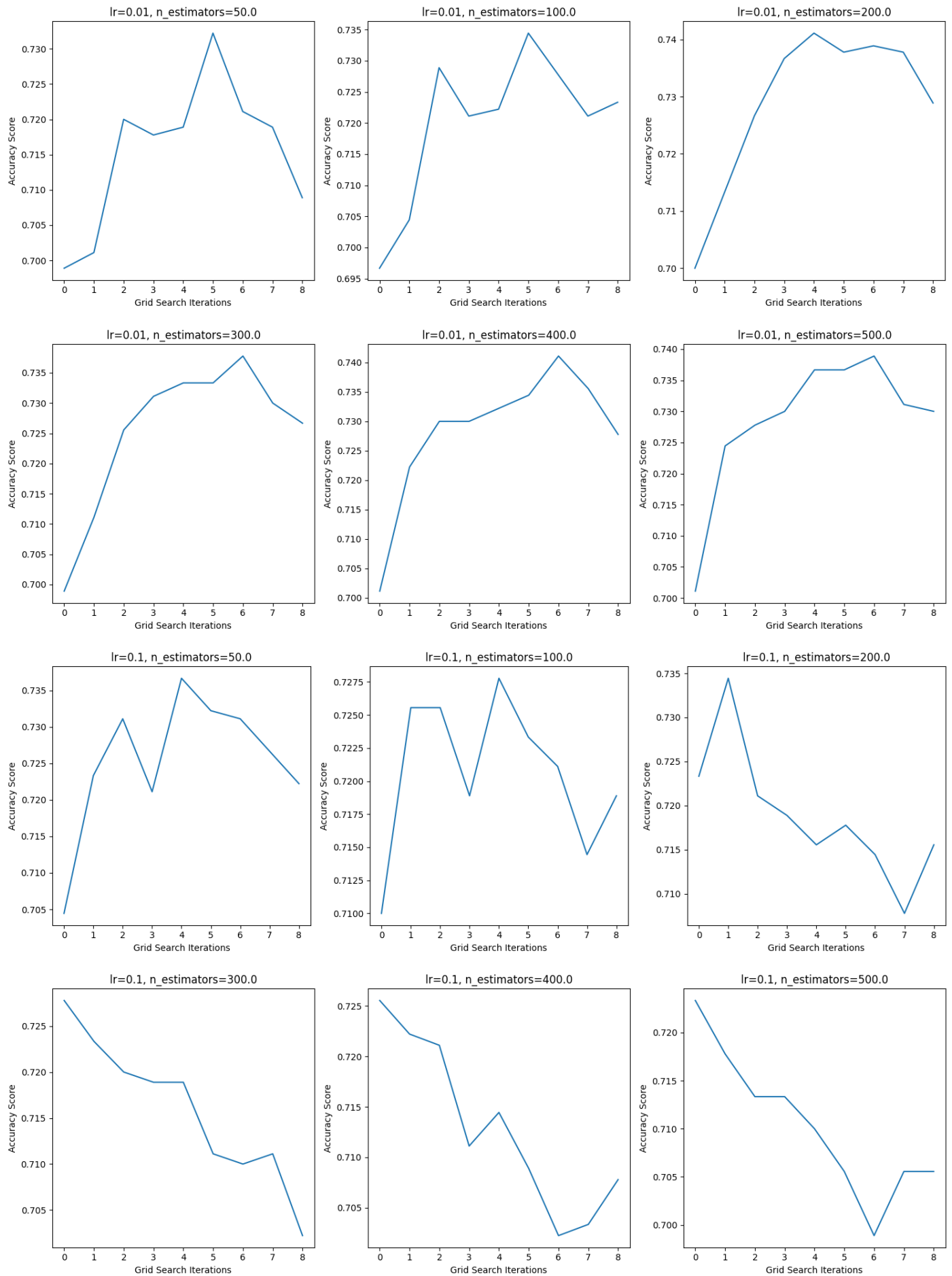


Figure 8: Mean and Variance of Accuracy For Different n Values

5.5. Extreme Gradient Boosting (XGB) Classifier Modal Hyperparameter



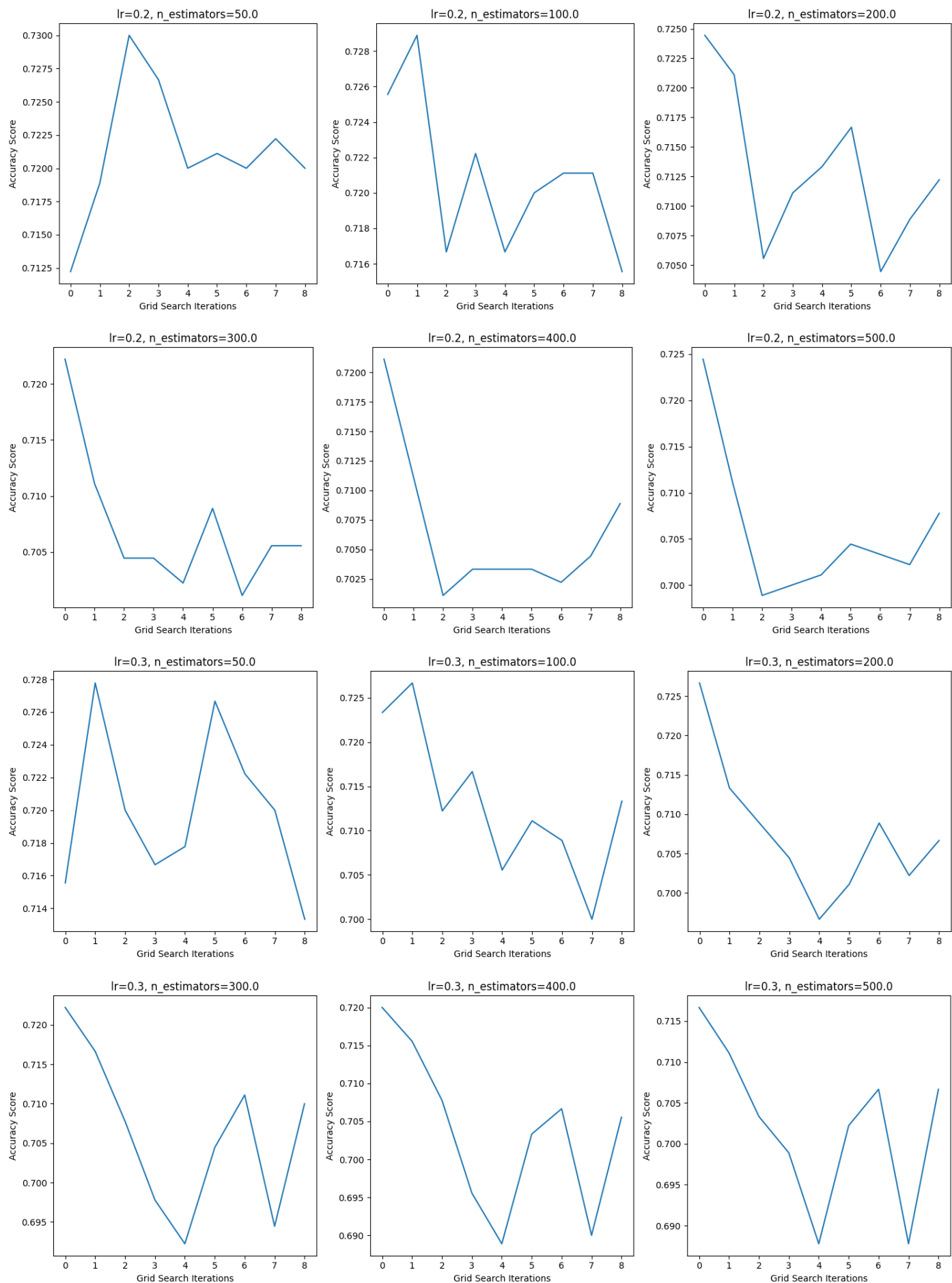


Figure 9: XGB Figure

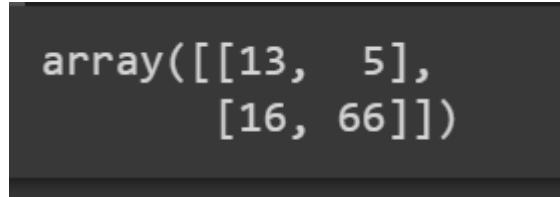
5.6. Summary Table

Table 1: Table of Summary

CLASSIFICATION ALGORITHMS	VALIDATION ACCURACY SCORES
Logistic Regression (LR) Modal	0.71333
Decision Trees (DT) Classifier Modal	0.70666
Multilayer Perceptron (MLP) Classifier Modal	0.72
Multilayer Perceptron Classifier with Keras	0.74
K-Nearest Neighbors (KNN) Classifier Modal	0.66222
Support Vector Machines (SVM) Classifier Modal	0.71333
Extreme Gradient Boosting (XGB) Classifier Modal	0.74333

6. Discussion

Our best algorithm is **XGB (Extreme Gradient Boosting model)**. The XGB Model's validation accuracy was **0.74111** with hyperparameter tuning. Suppose we use that algorithm to test data by training our model, including all samples and giving all necessary hyperparameters. In that case, accuracy becomes **79%**, which means our model estimates a sample with 79% correctness.



```
array([[13,  5],
       [16, 66]])
```

Figure 10: Arrays

This provided confusion matrix consists of TN, FP, FN, and TP with respect to rows and columns. Therefore, there are 13 True Negatives, 5 False Positives, 16 False Negatives, and 66 True Positives. When we inspect these ratios, we can see that our precision rate for the test is 0.92~. The XGB classifier mostly provides good intuition to find positive samples.

However, most of the negative samples are estimated wrongly. 0.55 of them were labeled as positive instead of negative.

These results show that our XGB classifier tends to estimate samples as positive rather than negative. This is caused by class imbalance. In other words, the sample's majority class dominates our best algorithm while estimating labels.

Therefore, our algorithm would have lower accuracy in an even dataset than in the testing. The algorithm is fuzzy while deciding on negative samples.

7. Bonus

For the bonus part, we have used 3 different models that use methods of classifier combinations. These 3 models were **Ensemble Averaging (hard voting)**, a **Mixture of Experts**, and a **Random Forest Classifier**. Ensemble Averaging and Random Forest Classifiers are static classifier combination algorithms, and a Mixture of Experts is a dynamic classifier combination algorithm.

Firstly, **Ensemble Averaging** gave us a result of **0.72** with the K-fold cross-validation method we use. While implementing the Ensemble Averaging method, we used the Logistic Regression, Decision Tree, XGB, and MLP models. The reason why KNN and SVM models are not used is that they can be considered stable algorithms. The results were interesting. Ensemble averaging gave a 0.74444 fold-validated accuracy score, which slightly improved XGB's validation performance.

Secondly, we used the **Mixture of Experts** models. This model is a dynamic model, so it is input based. We have trained our Mixture Expert Model with all algorithms we have ever trained. Results were surprisingly well: K-fold accuracy mean was **0.81**.

Thirdly, we have implemented a basic Random Forest model. The Random Forest model resulted in 0.73 accuracies after parameter tuning. Among all algorithms, Mixture of Experts was the most successful one, and we tested it. The testing result is a **0.77** accuracy score.

8. Conclusion

Our experiments resulted in what we estimated, mostly. We knew that Logistic Regression, KNN, and Decision Trees would not have high accuracies on the data since the data was so noisy. However, our trained MLP model and XGB model should have higher accuracies. MLP worked disappointingly and had a lower accuracy rate than some models discussed above.

However, one of the most complicated algorithms we used, the XGB model, worked well and saved the day with 0.74 accuracies in hyper parameter tuning. That is why XGB is chosen as the best classifier and tested. Our overall test results, except the bonus part, resulted in 0.79 accuracy.

However, with Classifier Combination models, we thought they would increase the accuracy. However, only the Mixture of Experts was able to be better than the XGB classifier, with 0.81 accuracy, which is a very good estimation rate for this problem since the data is noisy. On the other hand, the Mixture of Experts' test performance was surprisingly bad. It had a 0.77 accuracy rate, which is lower than the XGB. This may be caused by the amount of test data and the random state, which is why we still prefer using Mixture of Experts instead of XGB Classifier.