CMPS261 Final Proeject

Bashar Karaja, Ebaa Ibrahim, Hasan Hammoud

April 18, 2023

1 Abstract

The objective of this project was to build a machine learning model for classification task.Three different algorithms were used: XGBoost, Logistic Regressor, and MLPClassifier. The models were trained on the HIGGS data set given and evaluated based on their accuracy metrics.

The results showed that XGBoost performed decently on the given data with a training accuracy of 79 percent and testing accuracy of 74 percent while Logistic Regressor had the lowest training accuracy of 64 percent and testing accuracy of 65 percent. Our best model was MLPClassifier , which had an accuracy of 76.74 percent on the training data and 75.52 percent on the testing data, making it the best performing model overall. These findings suggest that the XGBoost algorithm performs well in training but may suffer from over fitting on testing data. On the other hand, the MLPClassifier showed good performance on both training and testing data, indicating its ability to generalize well.

2 Introduction

By examining the structure of matter and the laws regulating its interactions, the field of high-energy physics seeks to understand the basic principles of the cosmos. The main equipment used by experimental high-energy physicists to smash protons and/or antiprotons and produce exotic particles that only exist at extremely high energy density is modern accelerators. Solving challenging signal-versus-background classification issues, for which machine-learning methods are frequently applied, is necessary in order to find these uncommon particles. This research deals with a classification challenge that separates a signal process that generates hypothetical new Higgs bosons from a background process that has similar decay products but different kinematic characteristics. As part of our project we have used multiple types of model to solve this classification problem such as tree ensemble using XGBOOST, Neural Networks using MLPClassifiers as well as Logistic Regression

3 Manipulation of data

3.1 Data Characteristics

After Loading the data set, we can see that the data consists of 600000 data point where each data point have 27 features of which 21 of them are raw measurements and the rest are human made features using feature engineering

3.2 Missing data and cleaning

First we need to identify any missing data in the data set so that we can either remove the point or encoded accordingly, Fig.1 summarizes the above:

	features	number of missing data	percebtage of missing data
0	class label	0	0.000000
1	lepton pT	0	0.000000
2	lepton eta	0	0.000000
3	lepton phi	0	0.000000
4	missing energy magnitude	0	0.000000
5	missing energy phi	0	0.000000
6	jet 1 pt	0	0.000000
7	jet 1 eta	0	0.000000
8	jet 1 phi	0	0.000000
9	jet 1 b-tag	0	0.000000
10	jet 2 pt	0	0.000000
11	jet 2 eta	1	0.003645
12	jet 2 phi	1	0.003645
13	jet 2 b-tag	1	0.003645
14	jet 3 pt	1	0.003645
15	jet 3 eta	1	0.003645
16	jet 3 phi	1	0.003645
17	jet 3 b-tag	1	0.003645
18	jet 4 pt	1	0.003645
19	jet 4 eta	1	0.003645
20	jet 4 phi	1	0.003645
21	jet 4 b-tag	1	0.003645
22	m jj	1	0.003645
23	m jjj	1	0.003645
24	m lv	1	0.003645
25	m jlv	1	0.003645
26	m bb	1	0.003645
27	m wbb	1	0.003645
28	m wwbb	1	0.003645

Figure 1: shows the summary of missing data.

After cleaning and removing any strings or NAs, the data summary becomes as in fig.2:

	features	number of missing data	percebtage of missing data
0	class label	0	0.0
1	lepton pT	0	0.0
2	lepton eta	0	0.0
3	lepton phi	0	0.0
4	missing energy magnitude	0	0.0
5	missing energy phi	0	0.0
6	jet 1 pt	0	0.0
7	jet 1 eta	0	0.0
8	jet 1 phi	0	0.0
9	jet 1 b-tag	0	0.0
10	jet 2 pt	0	0.0
11	jet 2 eta	0	0.0
12	jet 2 phi	0	0.0
13	jet 2 b-tag	0	0.0
14	jet 3 pt	0	0.0
15	jet 3 eta	0	0.0
16	jet 3 phi	0	0.0
17	jet 3 b-tag	0	0.0
18	jet 4 pt	0	0.0
19	jet 4 eta	0	0.0
20	jet 4 phi	0	0.0
21	jet 4 b-tag	0	0.0
22	m jj	0	0.0
23	m jjj	0	0.0
24	m lv	0	0.0
25	m jlv	0	0.0
26	m bb	0	0.0
27	m wbb	0	0.0
28	m wwbb	0	0.0

Figure 2: shows the summary of missing data after cleaning

4 Models

4.1 XGBOOST

Now that our data is ready we will start implementing our models. The first model we implemented was tree ensemble using XGBOOST.

4.1.1 XGBOOST and its parameters

XGBoost is an acronym for "Extreme Gradient Boosting," a well-known and strong machine learning technique for regression, classification, and ranking applications. It is an ensemble learning approach that integrates numerous weak predictive models into a single, powerful model.

XGBoost is notably effective in the setting of large-scale, complicated data sets with numerous features, as it can handle missing values, regularization, and feature selection automatically. It leverages a gradient boosting strategy to repeatedly train decision trees using the residual errors of prior trees, while also including regularization to prevent over fitting.

it consists of the following parameters:

`n_estimators: controls number of decision trees used`

`learning_rate: controls the step size at each iteration of the gradient boosting process.`

`max_depth: controls the maximum depth of each decision tree in the ensemble`

`subsample: This parameter controls the fraction of training examples that are used to train each tree. Setting this parameter to a value less than 1 can help to reduce overfitting.`

`colsample_bytree: This parameter controls the fraction of features that are used to train each tree. Setting this parameter to a value less than 1 can help to reduce overfitting.`

`gamma: This parameter controls the minimum reduction in the loss required to split a node during the tree building process. Increasing this parameter can help to reduce overfitting.`

`min_child_weight: This parameter controls the minimum sum of instance weight (hessian) needed in a child. Increasing this parameter can help to reduce overfitting.`

`reg_alpha: This parameter controls L1 regularization on the weights of the decision trees. Increasing this parameter can help to reduce overfitting.`

`reg_lambda: This parameter controls L2 regularization on the weights of the decision trees. Increasing this parameter can help to reduce overfitting.`

`objective: This parameter defines the loss function to be optimized during training. XGBoost supports a wide range of loss functions for different types of problems, such as binary classification, multiclass classification, and regression.`

While training our XGBOOST Model, we have been tuning the different hyper parameters , the table below summarizes the most significant results:

learning rate	depth	min_ch	gamma	alpha	reg_lambda	subsample	colsample	object	n_estimators	train acc	test acc]
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	10000	0.914965958	0.739139493
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	2000	0.795607	0.741747848
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	1500	0.783402	0.741397845
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	1000	0.738585	0.729322744
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	200	0.782131	0.7227946
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	150	0.768002	0.719171231
0.1	5	3	0.8	0.1	3	0.8	0.8	binary	120	0.759123	0.722182443
0.1	6	1	0.5	0.1	3	0.8	0.8	binary	100	0.754534	0.71223
0.1	5	1	0.5	0.1	3	0.8	0.8	binary	100	0.781248	0.712434234
0.1	5	1	0.5	0.1	3	0.8	0.8	binary	120	0.759129	0.723458543
0.1	8	1	1	0.1	3	0.8	0.8	binary	150	0.759123	0.718123123
0.1	5	1	0.8	0.1	3	0.8	0.8	binary	150	0.751293	0.722304124
0.1	5	1	0.8	0.1	3	0.8	0.8	binary	200	0.760124	0.71912312

Figure 3: shows the variation of training and testing accuracies as a fcuntion of XGBOOST Hyper parameters

4.1.2 testing effect of hyper parameters

we will now test the effect of some parameters on the accuracy of our model. Note that the initial values for all the hyper parameters are:

```
learning_rate=0.1,
max_depth=5,
min child weight=3,
gamma=.3,
alpha=0.1,
reg_lambda=3.0,
subs_ample=0.8,
colsample_bytree=0.8,
objective='binary:logistic',
n_estimators=100,
```

These parameters have continuously been tuned as we were try different combinations to see the best model possible. the below graphs show sample of the tuning process

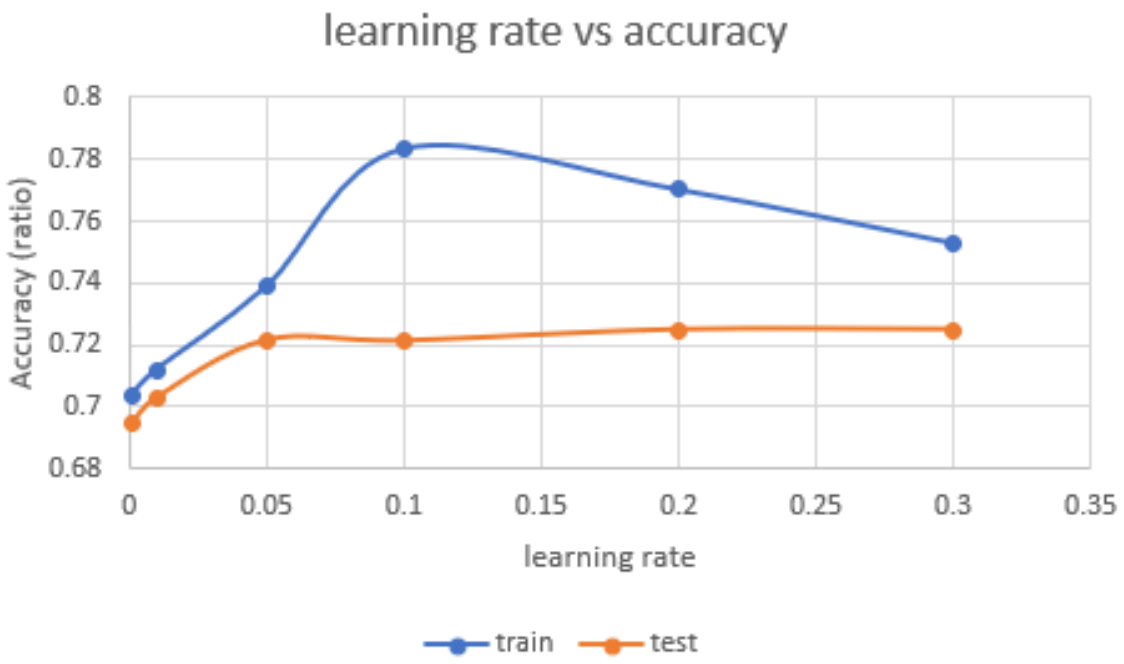

Figure 4: shows the variation of training and testing accuries as a fcuntion the learning rate

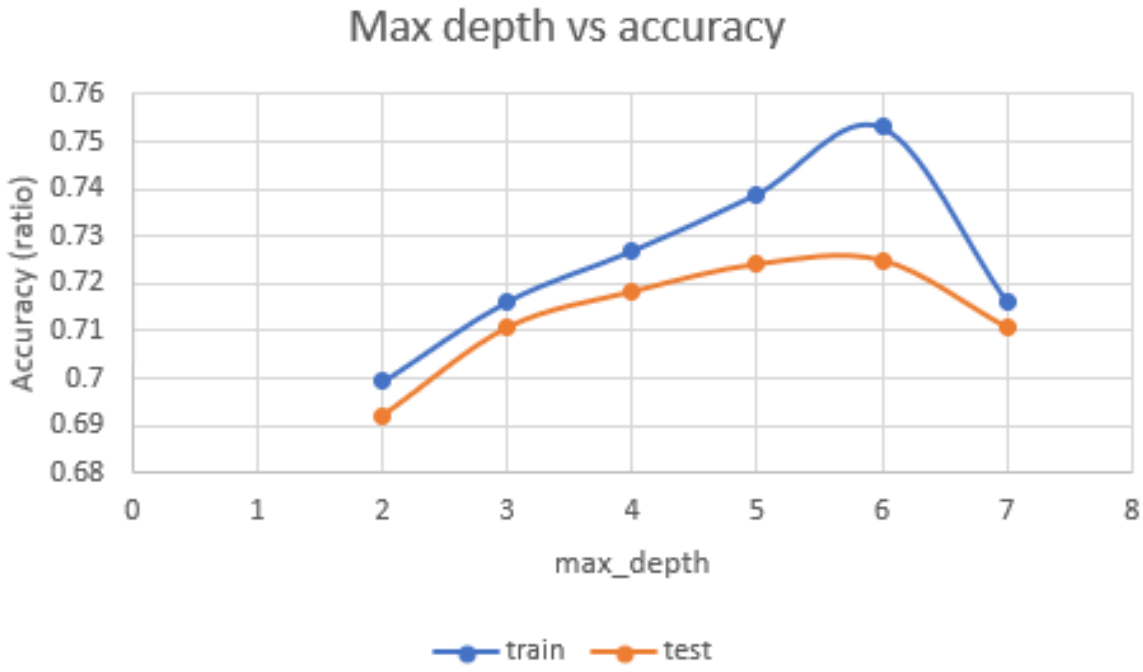

Figure 5: shows the variation of training and testing accuries as a fcuntion of the max depth

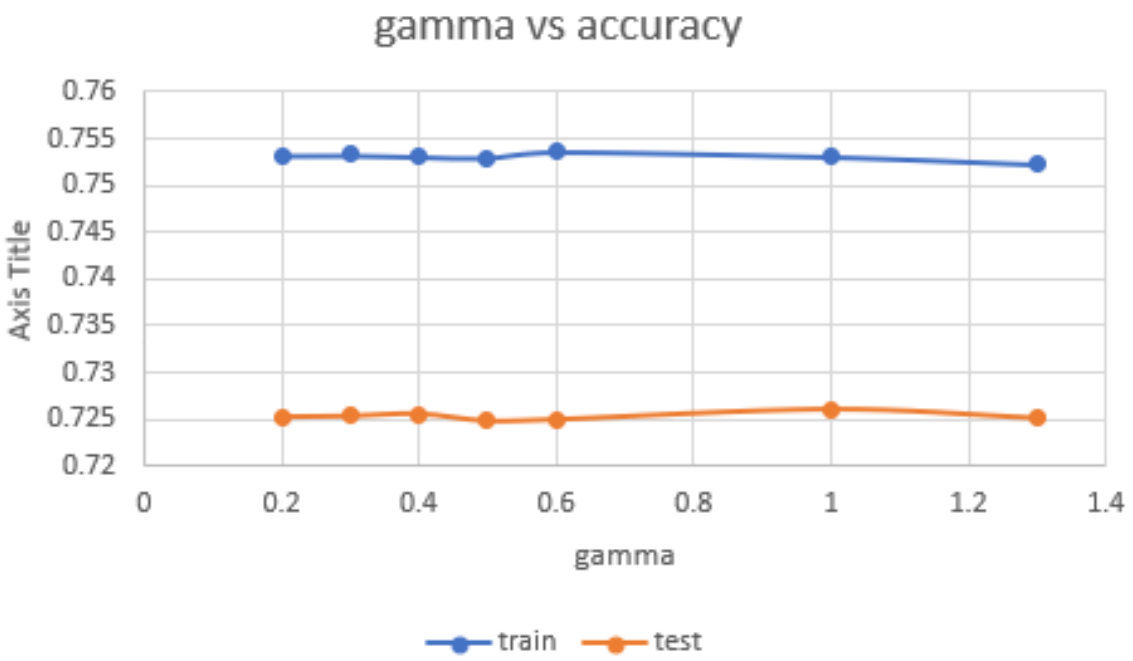

Figure 6: shows the variation of training and testing accuries as a fcuntion ofgamma

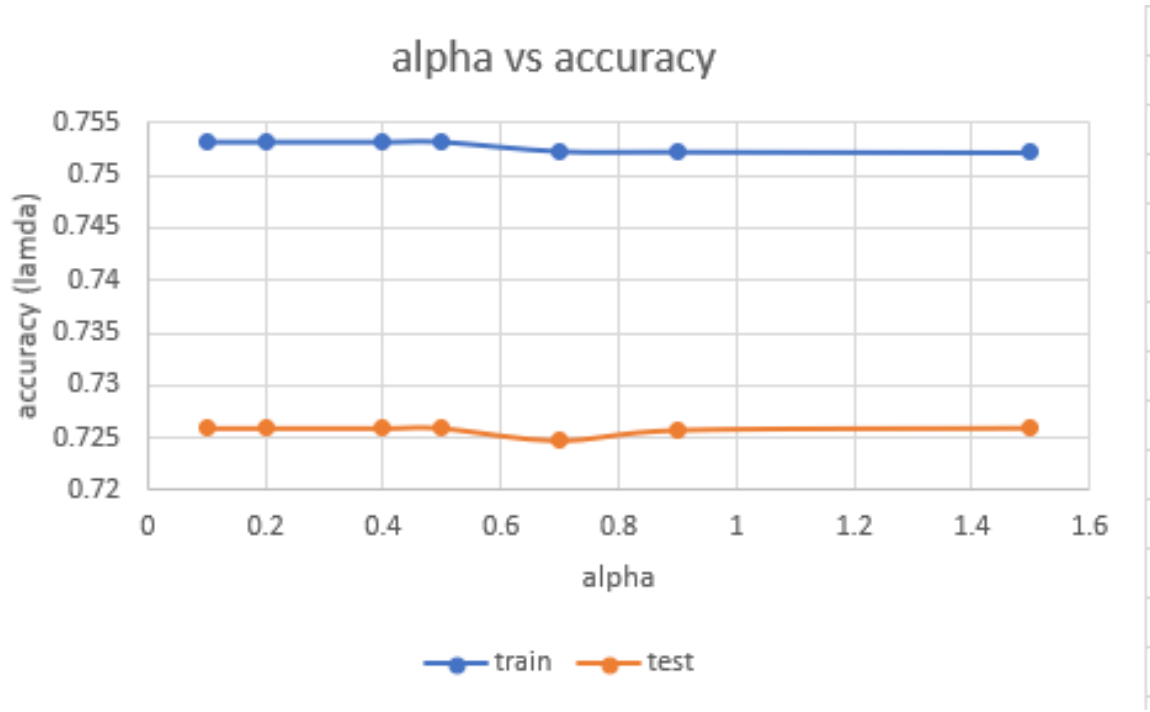

Figure 7: shows the variation of training and testing accuries as a fcuntion of alpha

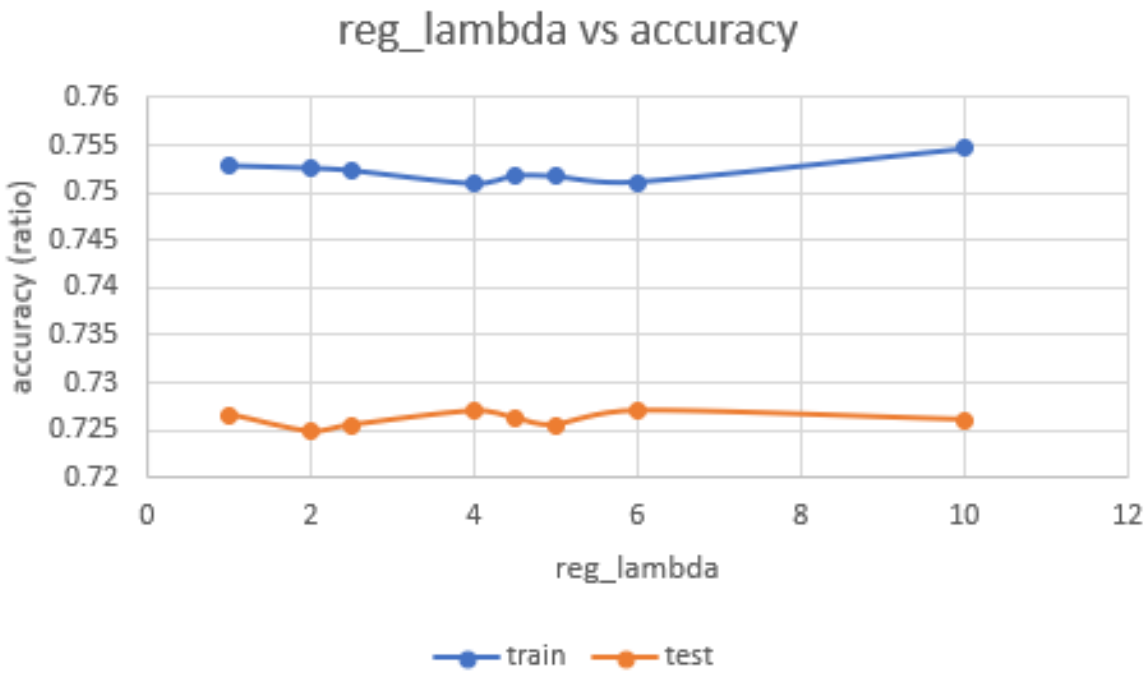

Figure 8: shows the variation of training and testing accuries as a fcuntion of reg lambda

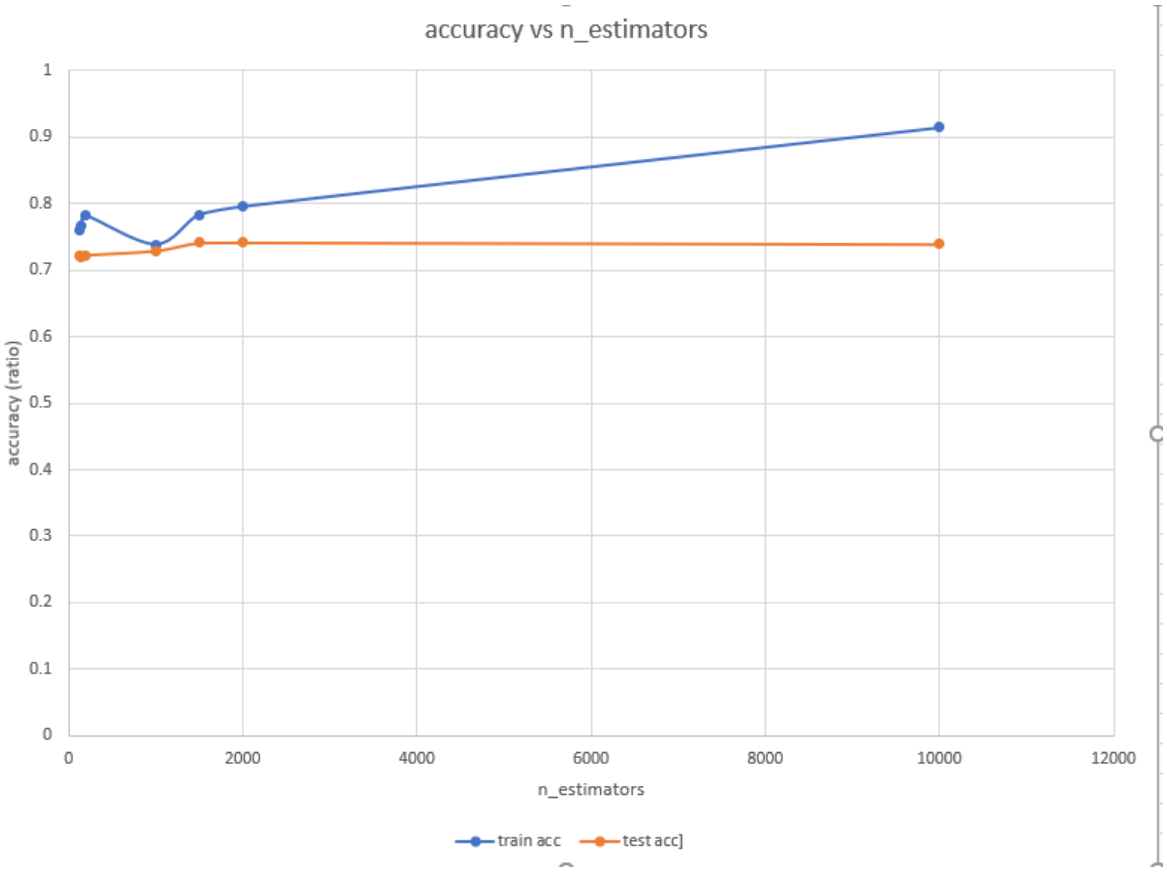

Figure 9: shows the variation of training and testing accuries as a fcuntion number of decision trees in XGBOOST

4.1.3 Best XGBOOST Model

After several combinations of the different hyper parameters values we found that:
the best XGBOOST Model performence model gave us an accuracy ratio of 0.7417 for the testing and 0.795 for the training,
where the best choice of parameters is concluded to be:

learning_rate=0.1,
max_depth=5,
min child weight=3,
gamma=.8,
alpha=0.1,
reg_lambda=3.0,
subs_ample=0.8,
colsample_bytree=0.8,
objective='binary:logistic',
n_estimators=2000,
However this is not the our best performence model of all and better accuracy
have been achieved using MLPClassifier

4.2 MLPClassifier

4.2.1 MLPClassifier and its parameters

MLPClassifier is a class in the scikit-learn Python library that implements a multi-layer perceptron (MLP) algorithm for classification tasks. An MLP is a type of neural network that consists of multiple layers of interconnected nodes, each of which applies a non-linear activation function to the weighted sum of its inputs. The MLP algorithm is trained using backpropagation, which adjusts the weights in each layer to minimize the error between the predicted and actual outputs. The MLPClassifier in sci kit-learn allows you to create an MLP model with customizable parameters, including the number of hidden layers, the number of nodes in each layer, and the activation function used in each layer. it consists of the following parameters:

hidden_layer_sizes: A tuple that specifies the number of nodes in each hidden layer of the neural network. For example, if you set hidden_layer_sizes=(50, 20), the neural network will have two hidden layers, the first with 50 nodes and the second with 20 nodes.

activation: The activation function used for the hidden layers. Popular choices include 'relu', 'tanh', and 'logistic'.

solver: The optimization algorithm used to update the weights of the neural network. Popular choices include 'sgd' (stochastic gradient descent), 'adam' (a variant of stochastic gradient descent),and 'lbfgs' (a quasi-Newton method).

alpha: The regularization parameter that controls the amount of L2 regularization applied to the weights. Larger values of alpha result in more regularization.

batch_size: The size of the minibatch used for stochastic gradient descent. Larger minibatches typically result in faster convergence, but smaller minibatches may lead to better generalization.

learning_rate: The learning rate used for stochastic gradient descent. This controls the step size used for updating the weights.

random_state: a random initial conditions of parameters to start with

the table below summarizes the change of accuracy as a function of different parameters:

Labels	hidden layers sizes	activation	solver	alpha	random_state	train accuracy	test accuracy
1	300,150,165,135,90	relu	adam	0.1	3	0.755889632	0.749464579
2	100,50,55,45,30	relu	adam	0.03	7651	0.767454312	0.755281294
3	100,50,55,45,50	relu	adam	0.03	7651	0.76604805	0.753906283
4	100,50,55,45,30	relu	adam	0.01	7651	0.773079359	0.753881282
5	100,50,55,45,30	relu	adam	0.03	7651	0.773079359	0.753881282
6	100,50,5,1	relu	adam	0.01	7651	0.733730179	0.731629233
7	100,50,50,10,1	relu	adam	0.1	7651	0.76169342	0.693239796
8	10000,5000,500,1000	relu	adam	0.1	7651	0.768710573	0.751847932

Figure 10: shows the variation of training and testing accuracies as a function of mlpclassifier parameters

NOTE: in our coming histogram in fig.11, we will be showing the effect of different hidden layers sizes on the accuracies, however we have labeled the different choices as in the table in fig.9 so that it would be easier to visualize. e.g: 1 would correspond to the first raw in the table of fig.10. All the different models have a size of 5 hidden layers but of different number of neurons

4.2.2 Graphs

This section shows the different relations and effects of the different hyper parameters individualy on th testing and training accuracies.The graphs below summarizes the results. we have set initially the hyper parameters for all the models to be the same except for the one we are varying to study its effect. The hyper parameters are initially set to:

```
hidden_layer_sizes: 5 hidden layers with sizes: 100,50,55,45,30

activation: relu.

solver: adam

alpha: 0.03

batch_size: default value set by the model

learning_rate: default value(0.001)

random_state: 7651
```

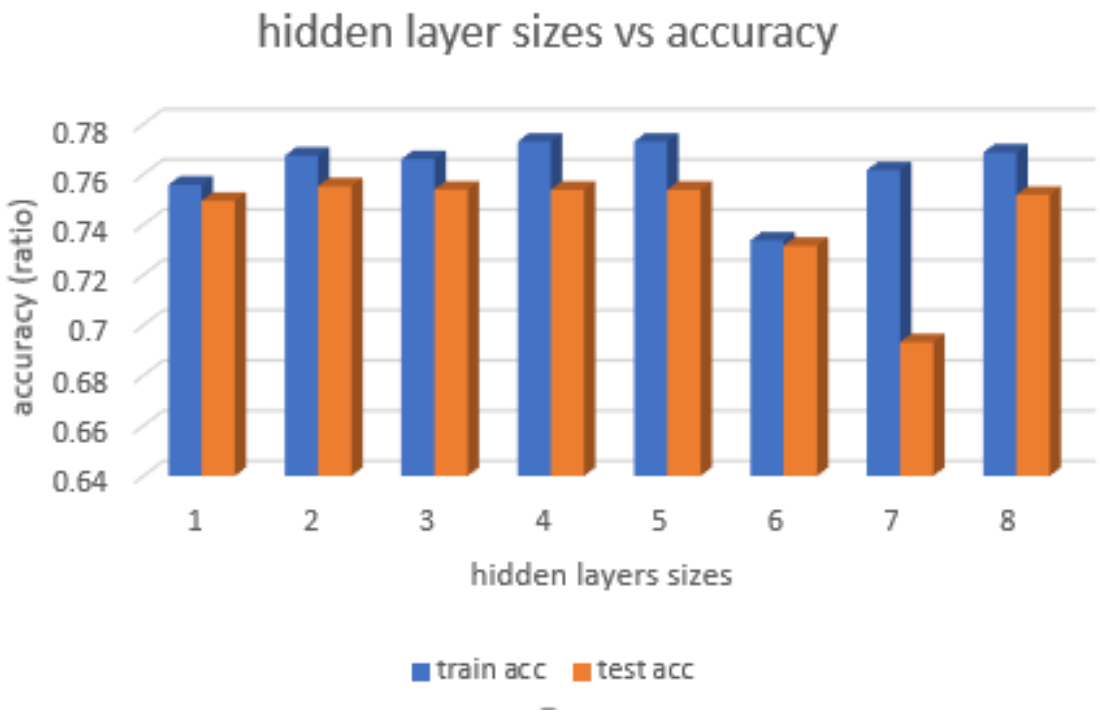

Figure 11: shows the variation of training and testing accuracies as a function of hidden layers sizes

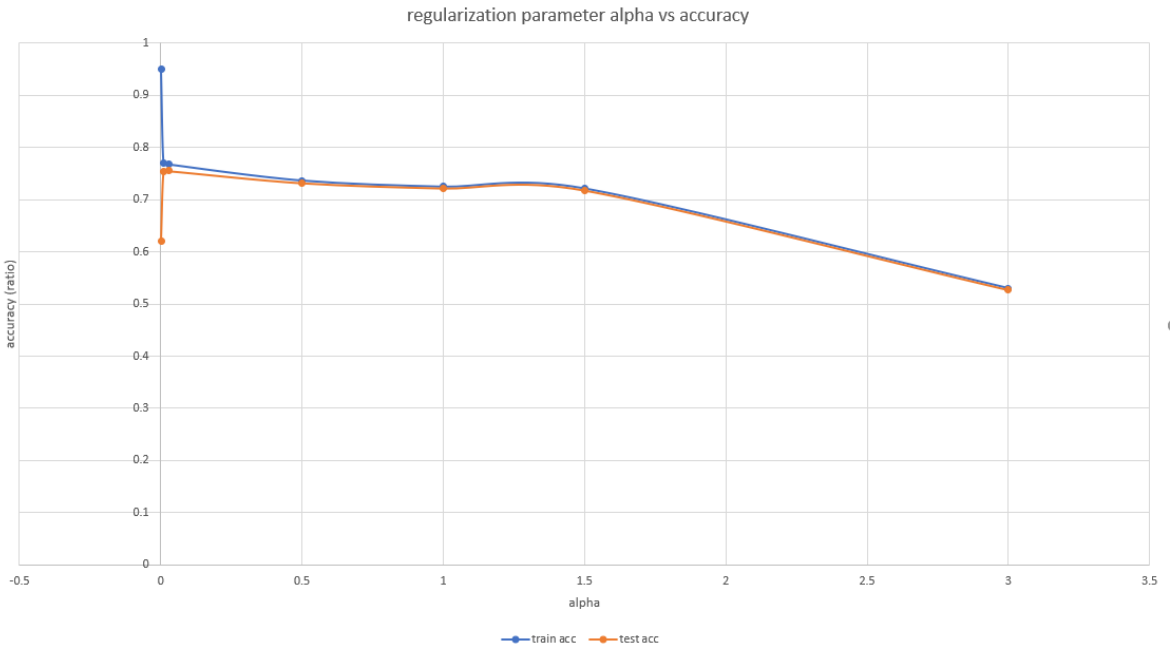

Figure 12: shows the variation of training and testing accuracies as a function of the regularization parameter alpha

when the regularization parameter is too small we can see that the model significantly over fitted with a training accuracy ratio of more than 0.9 and a testing accuracy of almost 0.62, while we can see that when it is too large, the model becomes very simple taht it starts to under fit with both testing and training accuracy ratios to be almost 0.52, which here indicate that the model is not learning and the output it is giving is just the mean value of our label column in the data

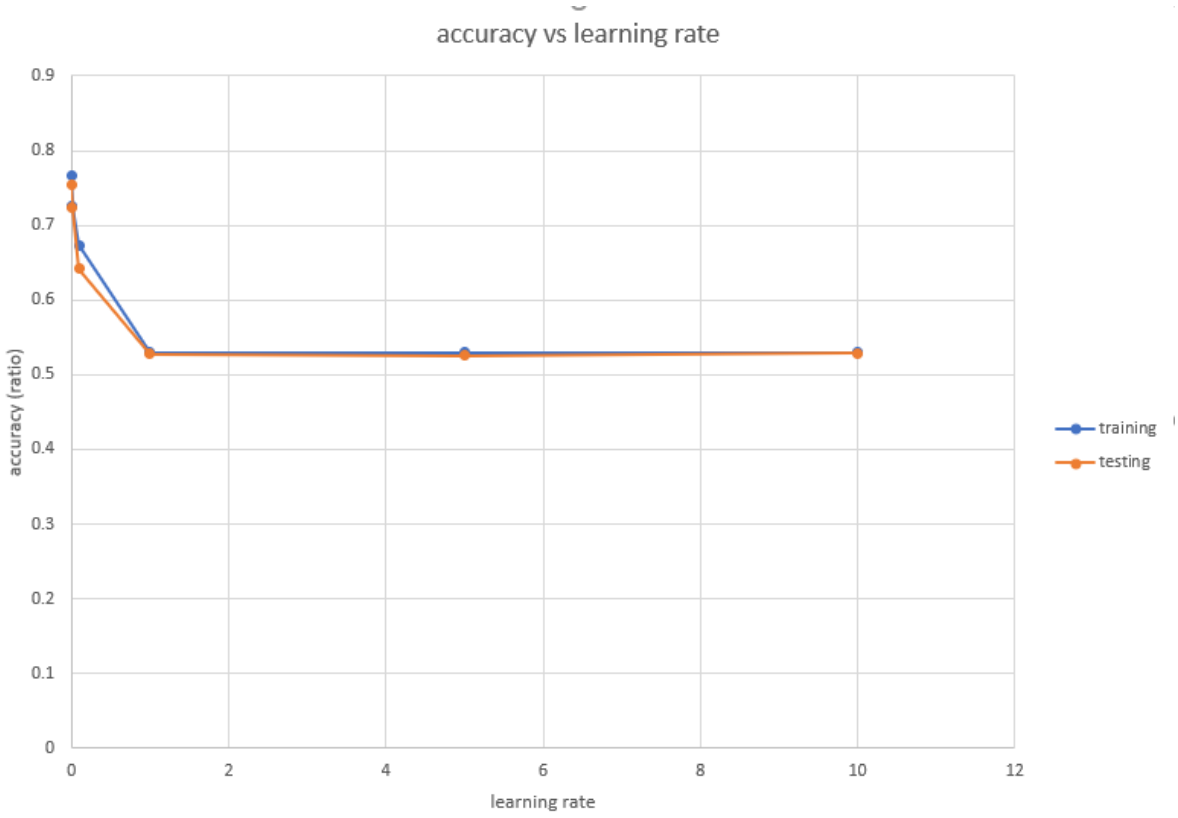

Figure 13: shows the variation of training and testing accuracies as a function of the learning rate

Note that as the learning rate increases starting 1, the model is no more learning, and the value it is giving is the mean value of our label data column, while the best accuracy is achieved when the learning rate is 0.001 which is the default value of the model with testing accuracy ratio of 0.7552

4.2.3 Best MLPClassifer Model

During the training process, we have tried over a 100 MLPClassifer models with different parameters, and we have noticed that the best number of layers to go with is 5 hidden layers and a regularization parameter of 0.03. The number of Neurons of each of the 5 hidden layers for our model of the best performance is: 100, 50, 55, 45 ,30, where the training accuracies was 0.76645 ratio while the testing accuracy was 0.75528 by ratio. Over all the whole parameters' values of the best MLP models are:

```
hidden_layer_sizes: 5 hidden layers with sizes: 100,50,55,45,30

activation: relu.

solver: adam

alpha: 0.03

batch_size: default value set by the model

learning_rate: default value(0.001)

random_state: 7651
```

4.3 Logistic Regression

4.3.1 Logistic Regression and its parameters

Logistic regression is a statistical method for predicting whether an observation belongs to one of two categories. It is a type of regression analysis in which the likelihood of a binary response variable (either 0 or 1) is modeled using one or more predictor variables.

In logistic regression, the logistic function (also known as the sigmoid function) is used to depict the relationship between the predictor variables and the response variable. Every real-valued input is converted to a value between 0 and 1. The logistic function is used to determine the likelihood of an observation belonging to the positive class given the predictor factors.

it consists of the following parameters:

```
penalty: This parameter determines the type of regularization to be applied to the model
. The default is 'l2', which corresponds to Ridge regularization.

C: This parameter controls the strength of the regularization. A smaller value of C leads
to stronger regularization, and a larger value leads to weaker regularization.
The default value is 1.0.

solver: This parameter determines the algorithm to be used for optimization. The default
is 'lbfgs', which is a quasi-Newton method.
```

the table below summarizes the change of accuracy as a function of the different Logistic Regression models:

Models	Solver	Penalty	C	Training Accuracy	Testing Accuracy
Model 1	Newton-Cg	L2	0.1	63.547	64.821
Model 2	Newton-Cg	L2	1	63.987	65.000
Model 3	Newton-Cg	L2	3	64.026	65.178
Model 4	Newton-Cg	None	0.1	64.045	65.076
Model 5	Newton-Cg	None	1	64.045	65.076
Model 6	Newton-Cg	None	3	64.045	65.076
Model 7	Lbfgs	None	0.1	64.045	65.127
Model 8	Lbfgs	None	1	64.045	65.127
Model 9	Lbfgs	None	3	64.045	65.127
Model 10	Liblinear	L1	0.1	63.841	64.898
Model 11	Liblinear	L1	1	64.038	65.127
Model 12	Liblinear	L1	3	64.051	65.051
Model 13	Liblinear	L2	0.1	63.560	64.821
Model 14	Liblinear	L2	1	63.981	65.025
Model 15	Liblinear	L2	3	64.019	65.178

Figure 14: shows the variation of training and testing accuracies as a function of Logistic Regression Model's parameters

4.3.2 Graphs

This section shows the different relations and effects of the different hyper parameters individualy on th testing and training accuracies.

The graphs below summarizes the results

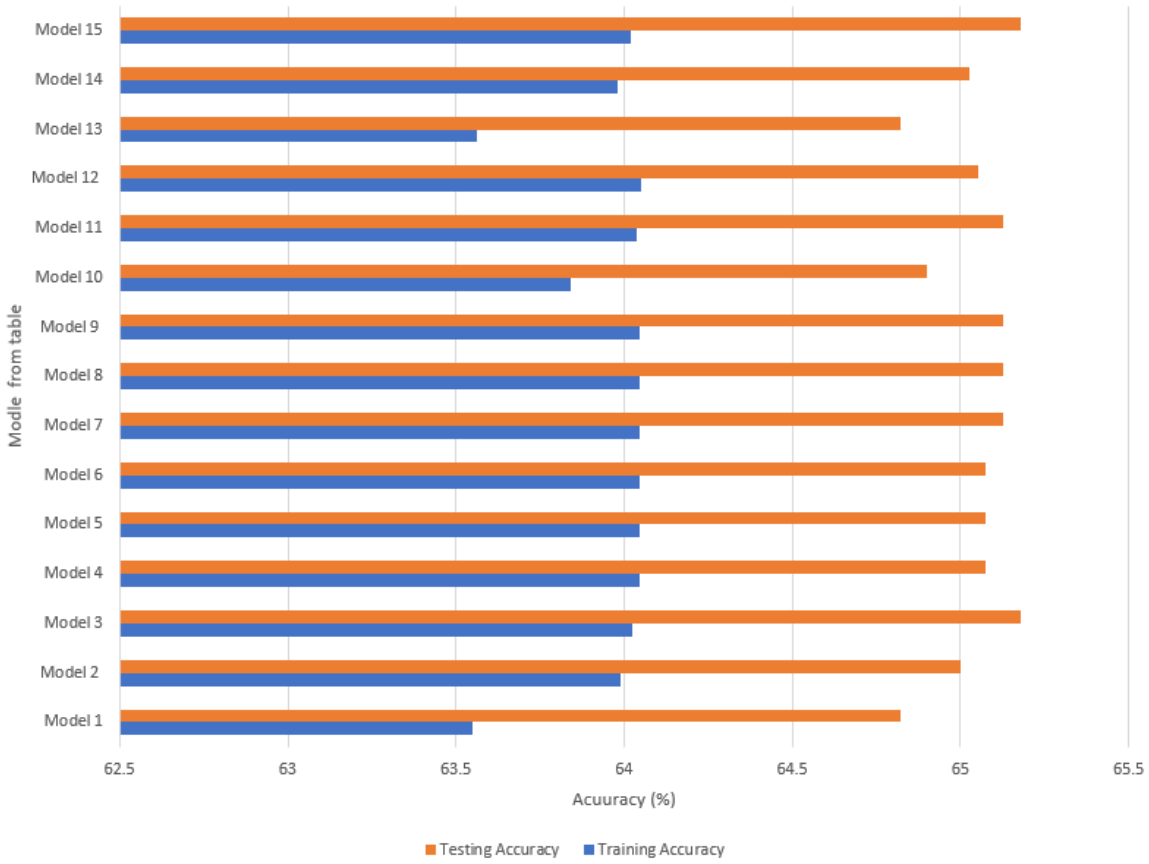

Figure 15: shows the variation of training and testing accuracies as a function of the models in the table of fig.11

4.3.3 Analysis

Upon implementing the model, we got a training accuracy of 0.65 using a ratio of 0.8 of the training examples that we were provided. And by testing the model on the other 0.2 by ratio of the data we got a testing accuracy of 0.64127 by ratio. Those numbers were close so there were no indications of over fitting. However, a ratio of 0.64127 accuracy is relatively low considering that in the research paper provided they were able to reach 0.78.

5 Summary of Results of the Different Models Best Performance

In this section we will be showing the a table of the best accuracy achieved by each model. and then we will uinclude each models parameter for the best performance

Models	Best Training Accuracy (Train)	Best Training Accuracy (Test)
MLPClassifier	0.7674	0.7552
Logistic Regression	0.65	0.64
xGBOOST	0.79	0.741

Table 1: Table caption goes here.

`MLPClassifier best performence parameters:`

number of hidden layers: 5
number of neurons in each: 100,50,55,45,30 respectively.
activation='relu',
solver='adam',
alpha=.03,
random_state = 7651

XGBOOST best performence parameters:

learning_rate=0.1,
max_depth=5,
min_child_weight=3,
gamma=.8,
alpha=0.1,
reg_lambda=3.0,
subsample=0.8,
colsample_bytree=0.8,
objective='binary:logistic',
n_estimators=1500,
scale_pos_weight=1

Logistic Regression:

solver:Lbfgs
penalty:None
C:1

Therefore, our best model implementation is MLPClassifier which give us the best accuracy of 75.52%