Title:

Final Project of Applied Machine Learning

Department of Mathematics

Center of Data Science

Advisor:
Professor Hajiabolhassan

Team members:

Fatemeh Saberi Khomami
Kavan Alipanahi
Parham Mohammadpanahi
Arash Sadeghi Bablan

1. Introduction

In a world overwhelmed with data, interpreting and gaining insights from massive datasets is a valuable skill.
Machine Learning is the field focused on developing learning algorithms.
In this project, we are going to implement some of the most famous supervised ML algorithms to build predictive models out of MoleculeNet datasets.

1.1 Motivation and Purpose

Machine Learning plays a key role in Chemistry. From testing drugs' effectiveness to the toxicity of chemical compounds, all rely on predictive models. What happens in chemistry affects our lives' quality severely. Hence, we decided to dedicate this research to "*Quantum chemistry structures and properties of molecules*"*,* and "*Toxicity prediction of chemical compounds*".

1.2 Datasets

For the task of Toxicity prediction we have two data sets with properties shown in the below table.

Name	Number of Chemical Compounds(Observations)	Number of Tasks
Tox21	7831	12
ToxCast	8576	617

For predicting Quantum Mechanics' properties of molecules we used a dataset with properties shown in the below table.

Name	Number of Chemical Compounds(Observations)	Number of Quantum Mechanics' properties
Qm9	133885	12

1.3 Organization

In the following sections, we are going to analyze and investigate Tox21, ToxCast, and QM9 datasets respectively.

2. Tox21 Dataset

This dataset is gathered via non-animal methods and is focused on Toxicology in the 21st Century. The objective of this research is to efficiently test thousands of chemicals for potential health effects. Screening these compounds will facilitate the EPA organization in issuing regulations concerning environmental protection.

2.1 Preprocessing

We have a feature-set for Tox21 which is the transition of chemical structures(smiles) of compounds into 201 numeric features.
The label-set corresponding Tox21 consisted of 12 experiments on chemical compounds to test whether they are toxic in those settings.

Data Wrangling Stages:

- In the label set we dropped the smiles column
- In the future set we changed the column names into natural numbers
- In the feature set we dropped the columns which had the same value over all the records
- In the feature set we had 101 Na records in the following columns: {38,44,42,40}, because the rows corresponding to these Na values were identical for the mentioned columns, we dropped those observations. (we only lost 0.01 of data which is negligible)
- We omitted the same rows from the label sat as well
- From the below 12 tasks, we want to choose the most suitable response which has the lowest Na rate.

Column Name	Number of Na values
SR-MMP	1995
NR-Aromatase	1969
SR-ARE	1965
NR-ER	1602
NR-PPAR-gamma	1357

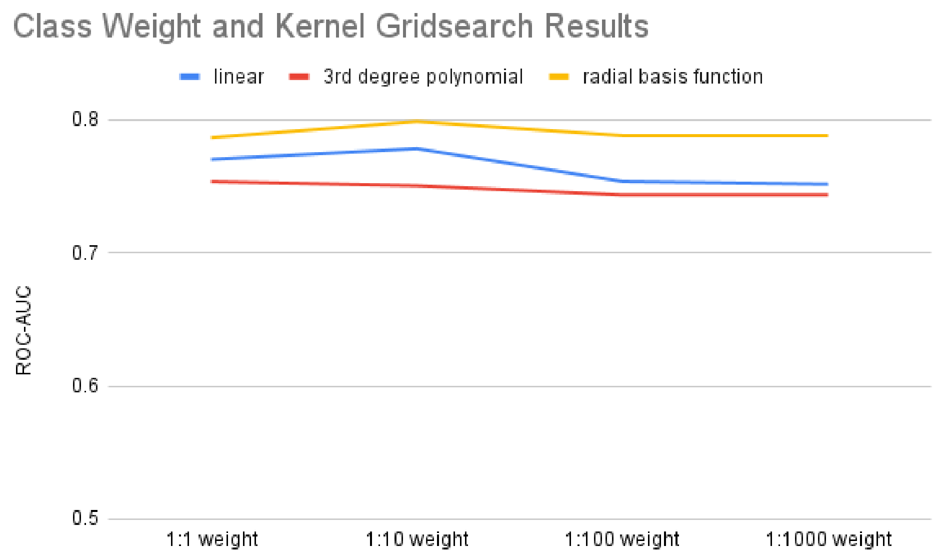
SR-HSE	1342
NR-AHR	1255
NR-AR-LBD	1049
SR-p53	1033
NR-ER-LBD	865
SR-ATAD5	739
NR-AR	564

- Regarding that NR-AR has the lowest amount of missing data, we decided to choose this task as our response variable.
- According to the correlation matrix, the task which has the highest correlation rate with NR-AR is NR-AR-LBD which is understandable, because both of them are predicting the antagonists of Aromatase enzyme which catalyzes the conversion of androgen to estrogen and thereby keeps the balance between these two hormones. (imbalance secretion of each of these hormones into the bloodstream can have the same effect as the toxins have on endocrines[1])
- We filled some of the Na records from the NR-NA task with the corresponding non-Na values of NR-NA-LBD
- In the end, 493 records of NR-NA were missing. Regarding that the portion of missing values to the entire dataset was negligible (0.06), we omitted the Na rows of the selected task.
- To match the labeled dataset with the feature set, we also dropped the same rows from the feature set.

2.2 Support Vector Classification

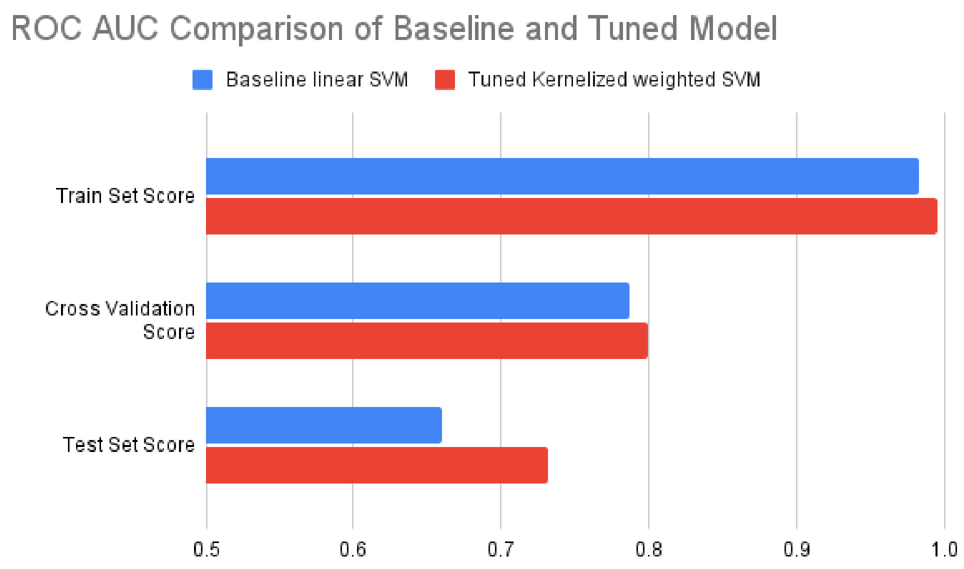
Support Vector Machines are flexible and powerful tools for classification tasks especially when few data points are available, as their sample complexity is independent of the dimensionality of the space. Furthermore, SVMs can easily learn non-linear decision boundaries through the use of kernel functions.

Employing SVMs on the NR-AR Task of the Tox21 dataset, we face an interesting challenge of data imbalance because most of the data points are negative. We will therefore use a variation of the SVM called weighted SVM that assigns more weight to points belonging to the less occurring class to mitigate bias toward the more numerous class.
We report ROC-AUC scores for a baseline linear SVM then perform a grid search on kernel and class weight to fine-tune hyper parameters, comparing models with a stratified 5-fold cross-validation (to mitigate effects of class imbalance).



The grid search shows a consistent lead for the radial basis kernel across different class weights achieving the best result in the 1 to 10 case for negative to positive weight.

A final model is trained with the optimized hyperparameters evaluated with 5-fold stratified cross-validation and a never before seen stratified test set.



The test set result shows significant improvement for the tuned model achieving almost 0.1 higher roc-auc score over the base model.

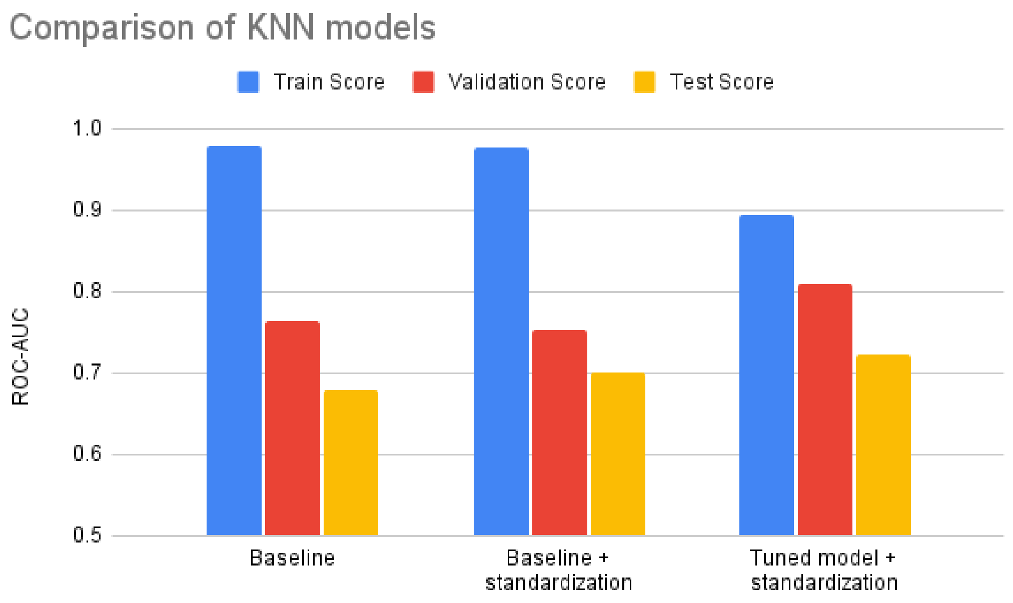
2.3 K-Nearest Neighbors

K Nearest Neighbors is one of the oldest classification methods, it operates using the distance between points and a new point is classified based on the value of its nearest neighbors.
Interestingly, data imbalance is not necessarily an issue in KNN models because the prediction is based on a local search, effectively ignoring macro-level details such as class imbalance.

To measure the distance between points we use the Lp distance and perform a grid search to choose the parameter p,

$$||x-y||_p = \left(\sum_{i=1}^{d} |x-y|^p\right)^{1/p}$$

Another parameter to be considered is K, the number of closest neighbors that decide on the class labels of a new point. We found, using trial and error, that K values less than 10 are not effective and lead to overfitting of the model. Therefore we put the bounds of our grid search at 10 and 50.



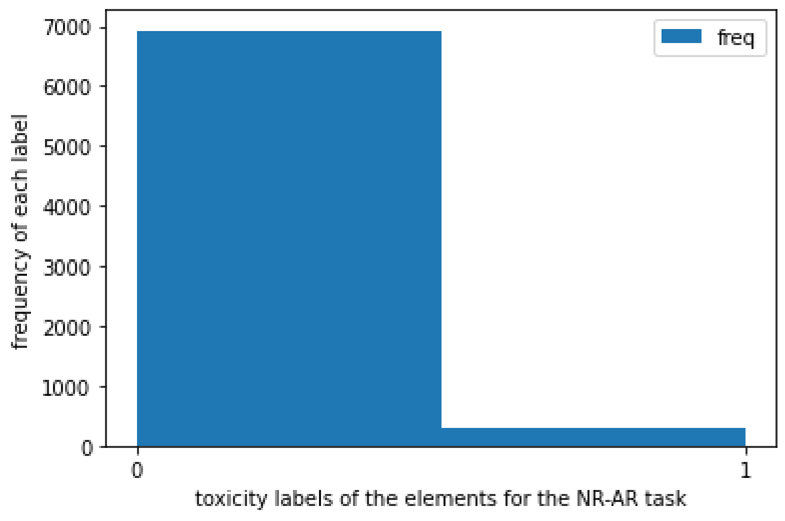
The best result was achieved by selecting K = 50 and using an L2 distance. Achieving over 0.7 ROC-AUC scores and standardization had only a tiny improvement as the features were already scaled to be between 0 and 1.

The final results are not very impressive, this is due to the curse of dimensionality that affects KNNs. In high dimensions, points similar to each other may have large distances from each other, in fact, all points will be far from each other, diminishing the predictive power of KNN.

Another telltale sign is that the grid search tends to pick larger K values as if the model does not get enough information from only a few neighbors.

2.4 Neural Network

In this section, we are going to implement a Multi-Layer Neural Networks algorithm to predict whether antagonists of Aromatase will succeed in making the corresponding compounds toxic or not.
First, we checked the distribution of labels.

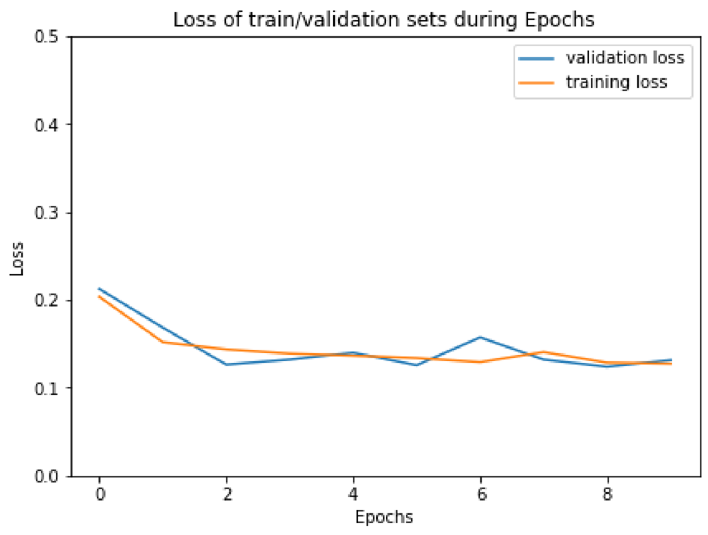


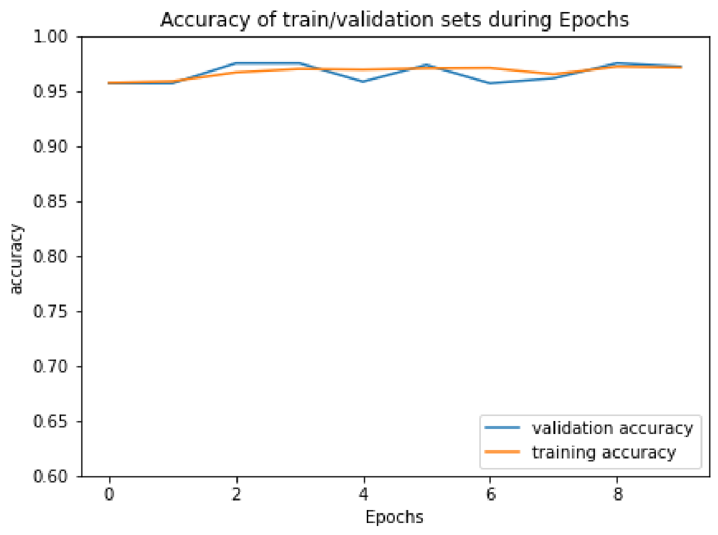
As you can see, labels are highly imbalanced. Regarding the distribution of our labels, we are prone to overfitting. So, to mitigate this issue, we will use K-fold cross-validation.
Neural Network architecture:

	Number of Neurons	Activation Function
Layer 1	**64**	**Relu**
Layer 2	**32**	**Relu**
Layer 3	**16**	**Relu**
Layer 4	**1**	**Sigmoid**

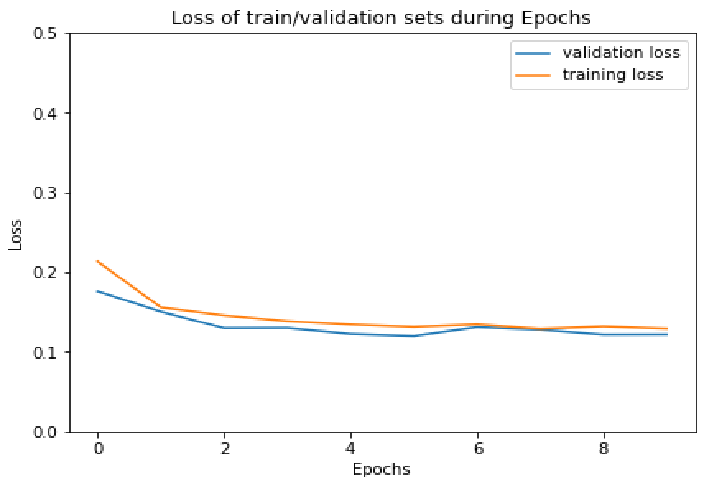
We also used layer weight regularizers for each layer with a penalty of 0.001.
We trained our model in a stratified 10-fold cross-validation manner. In building each fold, stratified sampling was used to keep the distribution of the original data.
You can see the performance of the model on each fold-over different epochs:

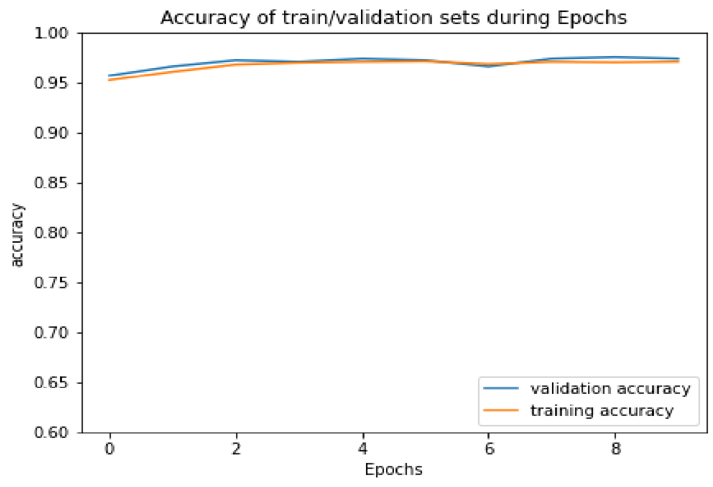
Fold 1



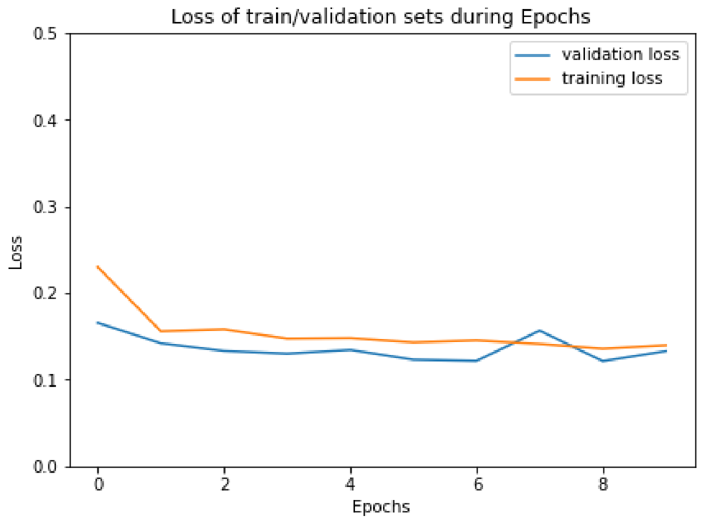


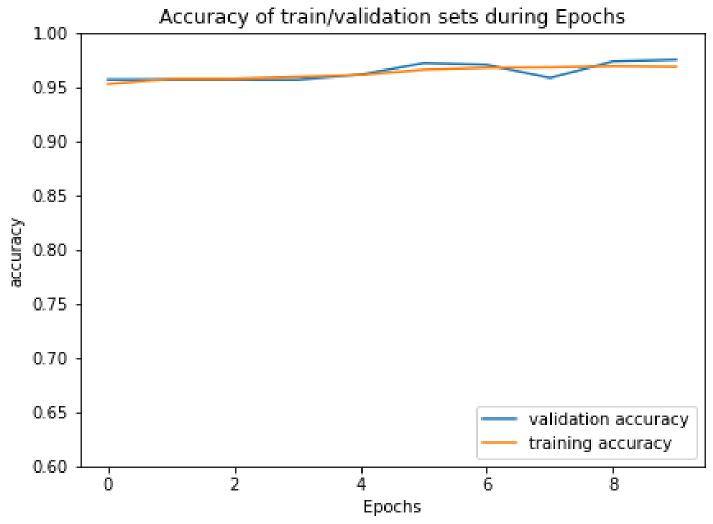
Fold 2


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8

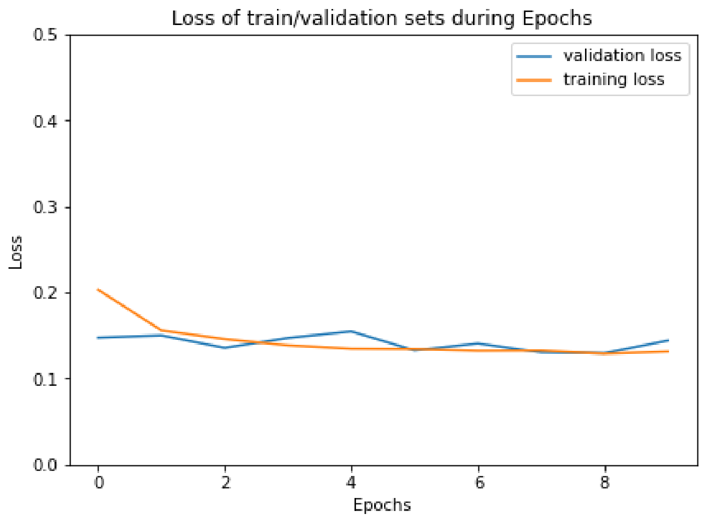

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8

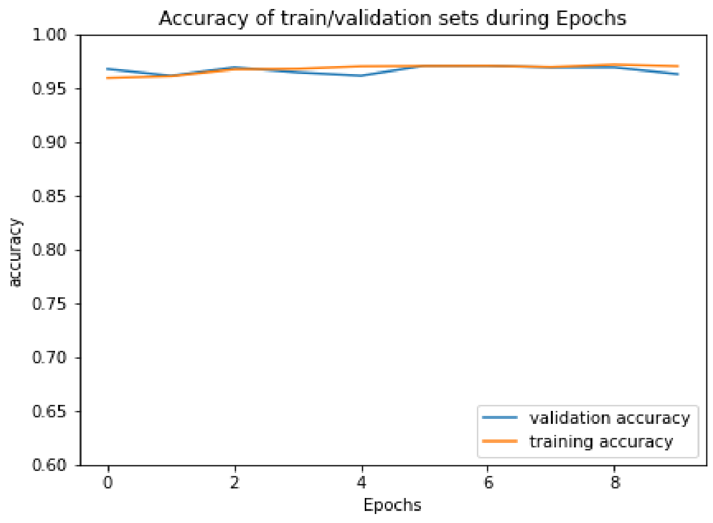
Fold 3


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8

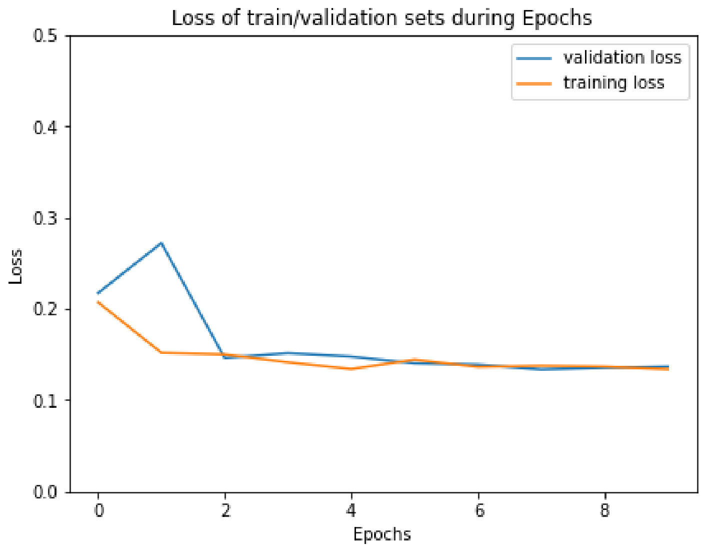

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8

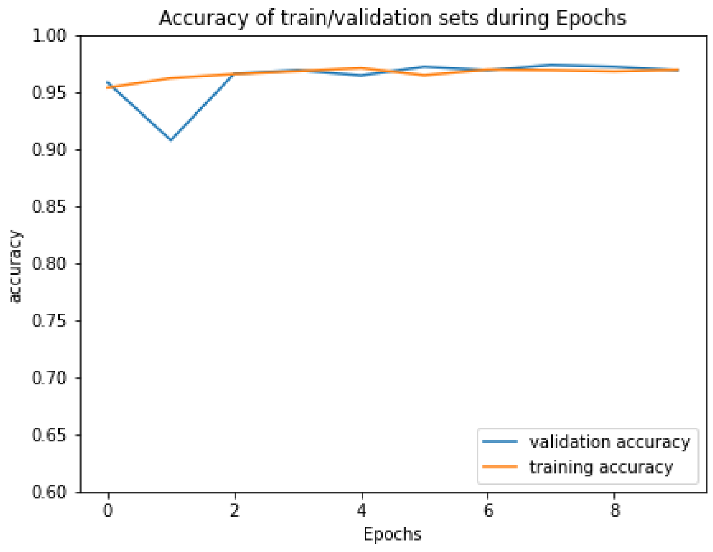
Fold 4


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs

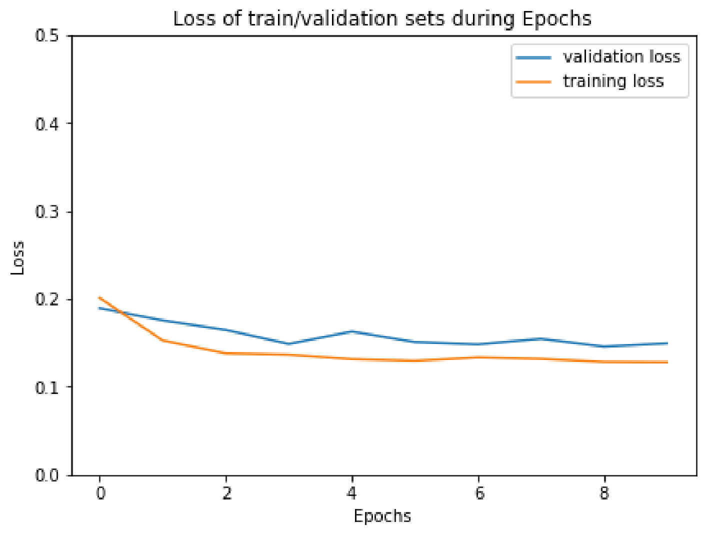

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs

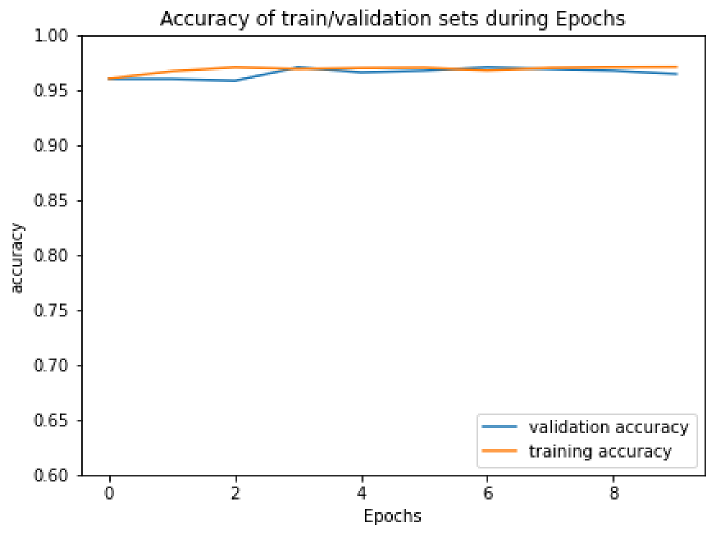
Fold 5


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs

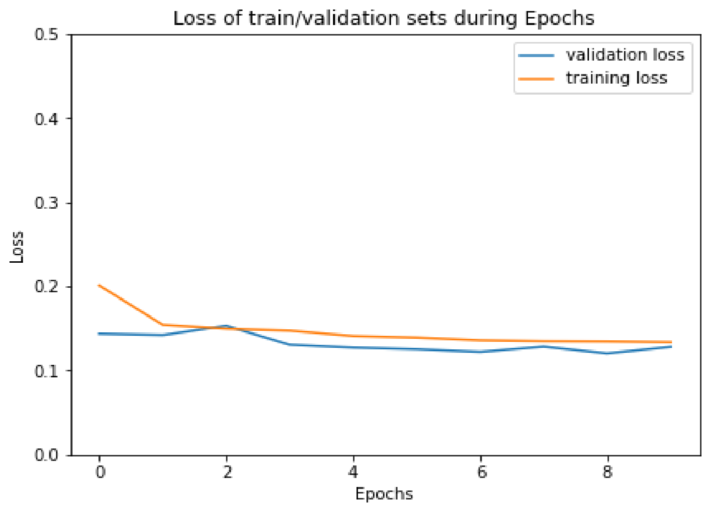

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs

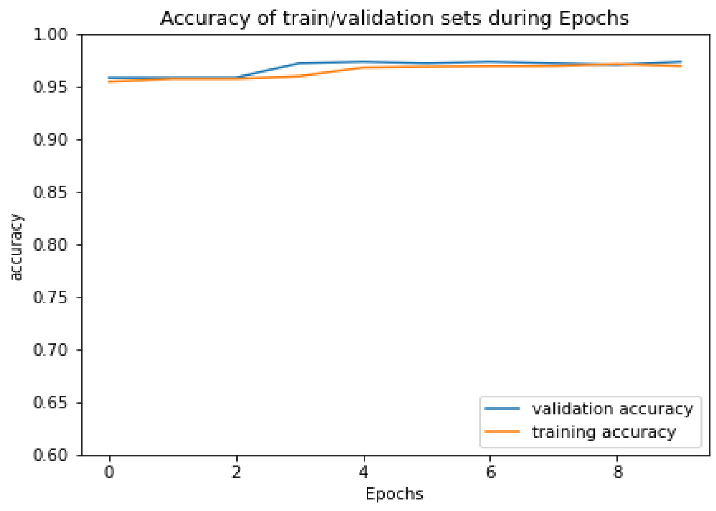
Fold 6


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs

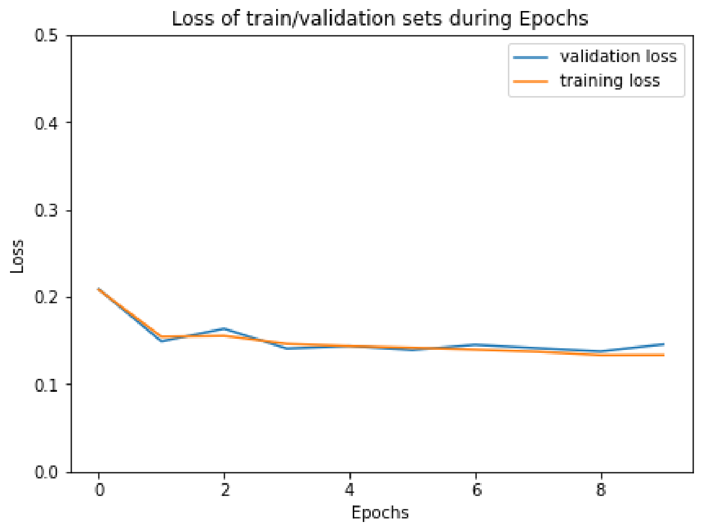

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs

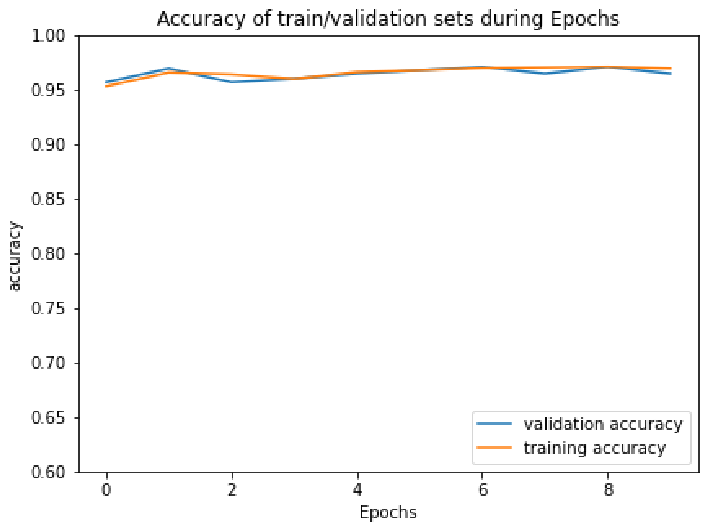
Fold 7


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs

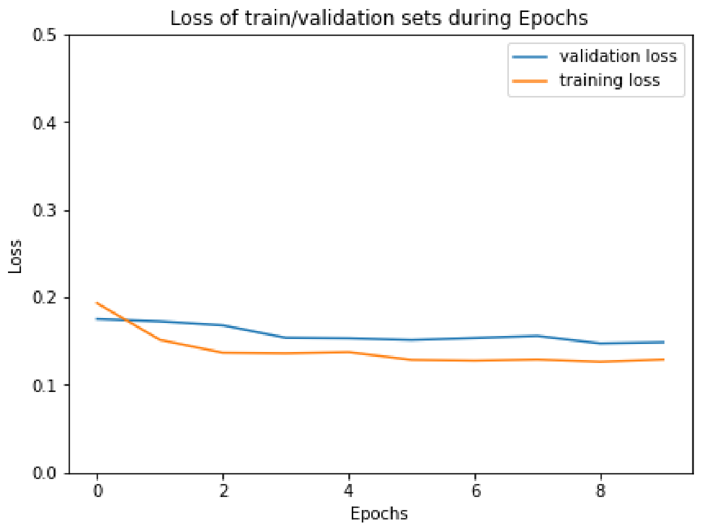

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs

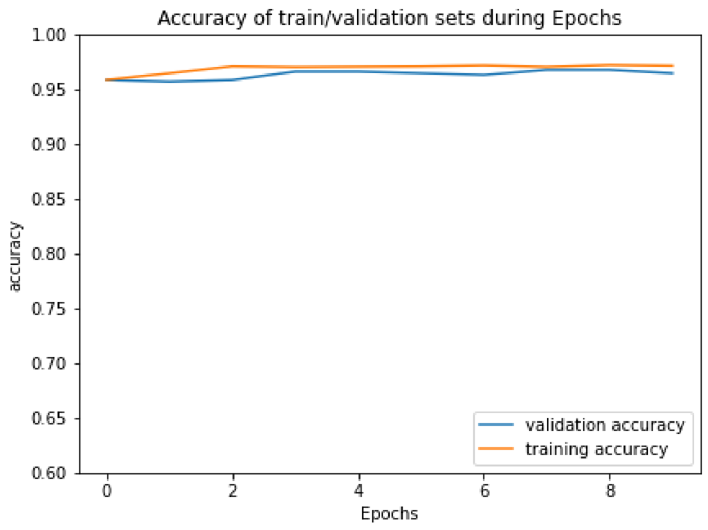
Fold 8


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8

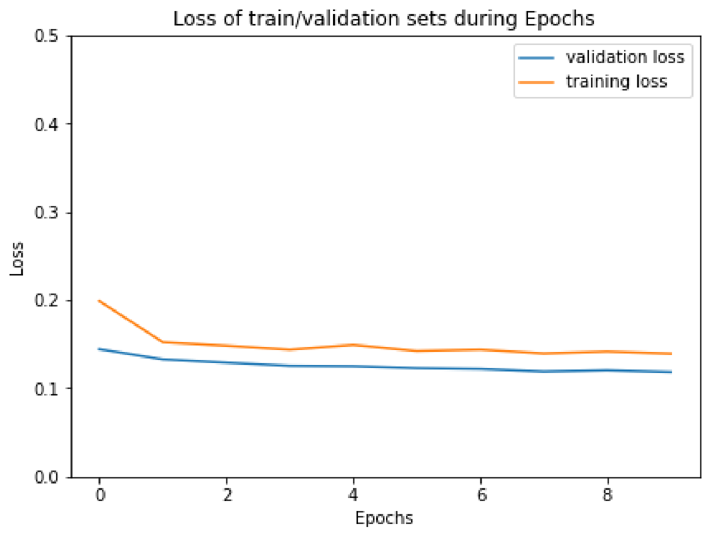

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8

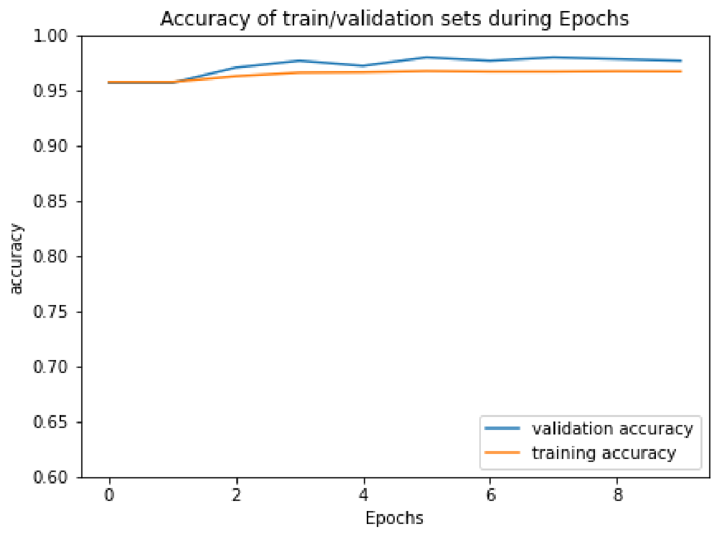
Fold 9


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8


Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
0
2
4
6
8

Fold 10





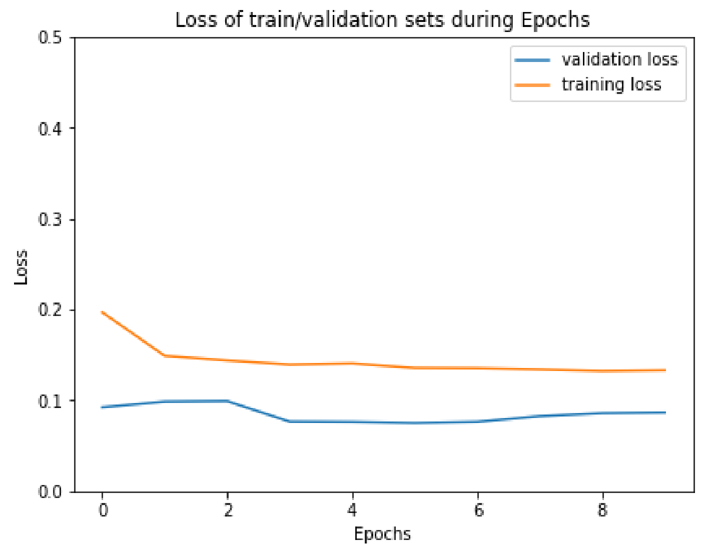
The average accuracy for all folds is:

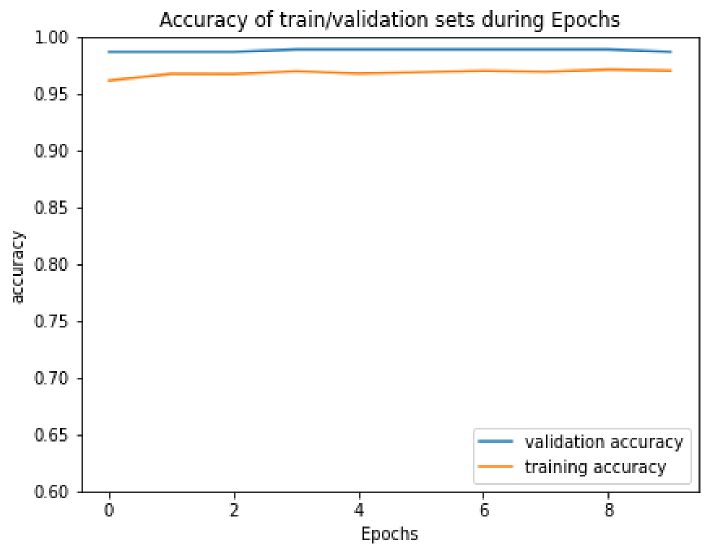
$96.99\% \pm 0.49$

The average Loss for all folds is:

0.135

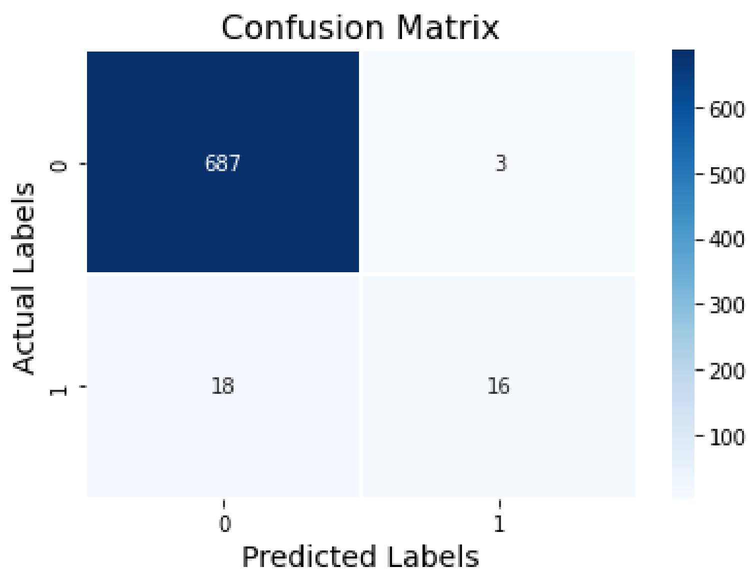
Now that we know our model is good, we will train the network on all the training data. The results are in the below figure.

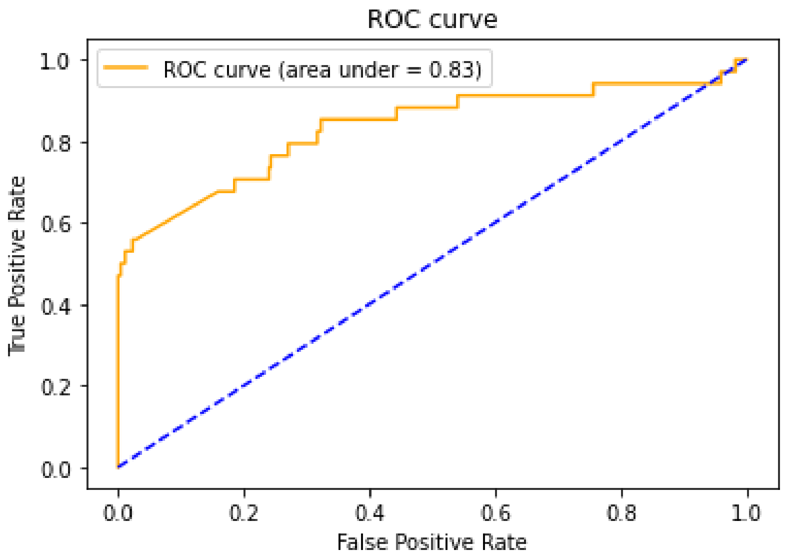




Loss and the accuracy of the above model on the test set are 0.129 and 97.09%, respectively. We also calculated the below metrics for the unseen data.

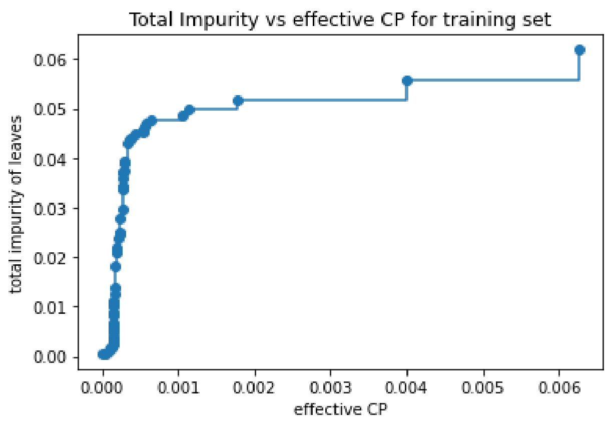
	Precision	Recall	F1-score	Support
On label 0	0.97	1.00	0.98	690
On label 1	0.84	0.47	0.60	34

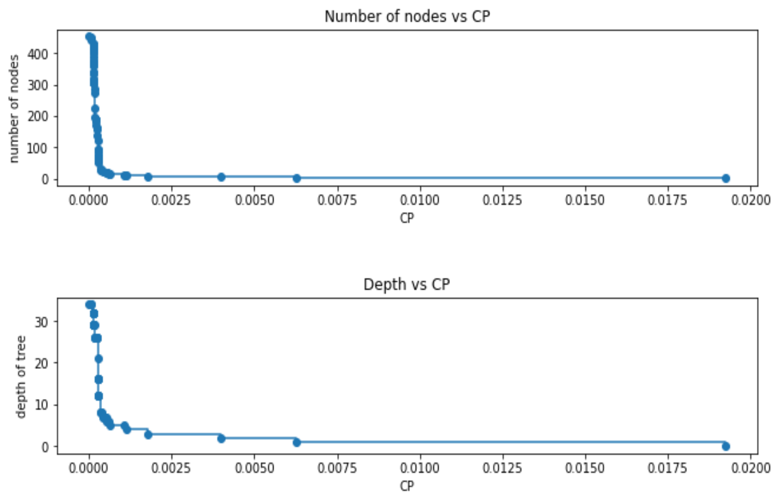




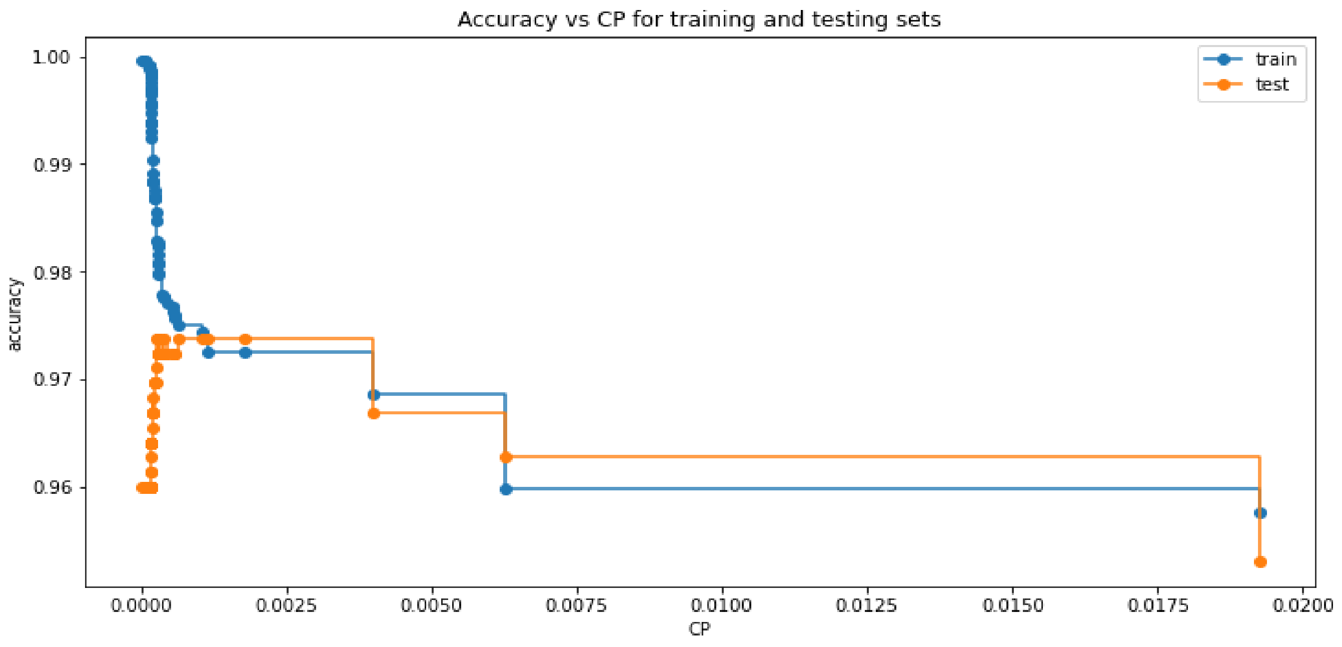
2.5 Decision Tree

Here, we are going to address the toxicity by fitting a decision tree model to our data.
We allocated 10% of the data to the test set, and with the rest, we are going to build a prediction model.
We chose the Gini impurity measure as The optimum split criteria.
One approach to prevent overfitting is using Complexity Parameter (CP) to prune the excessive leaves.
Below, we are going to show the relationship of a wide range of CP values with different properties of trees.



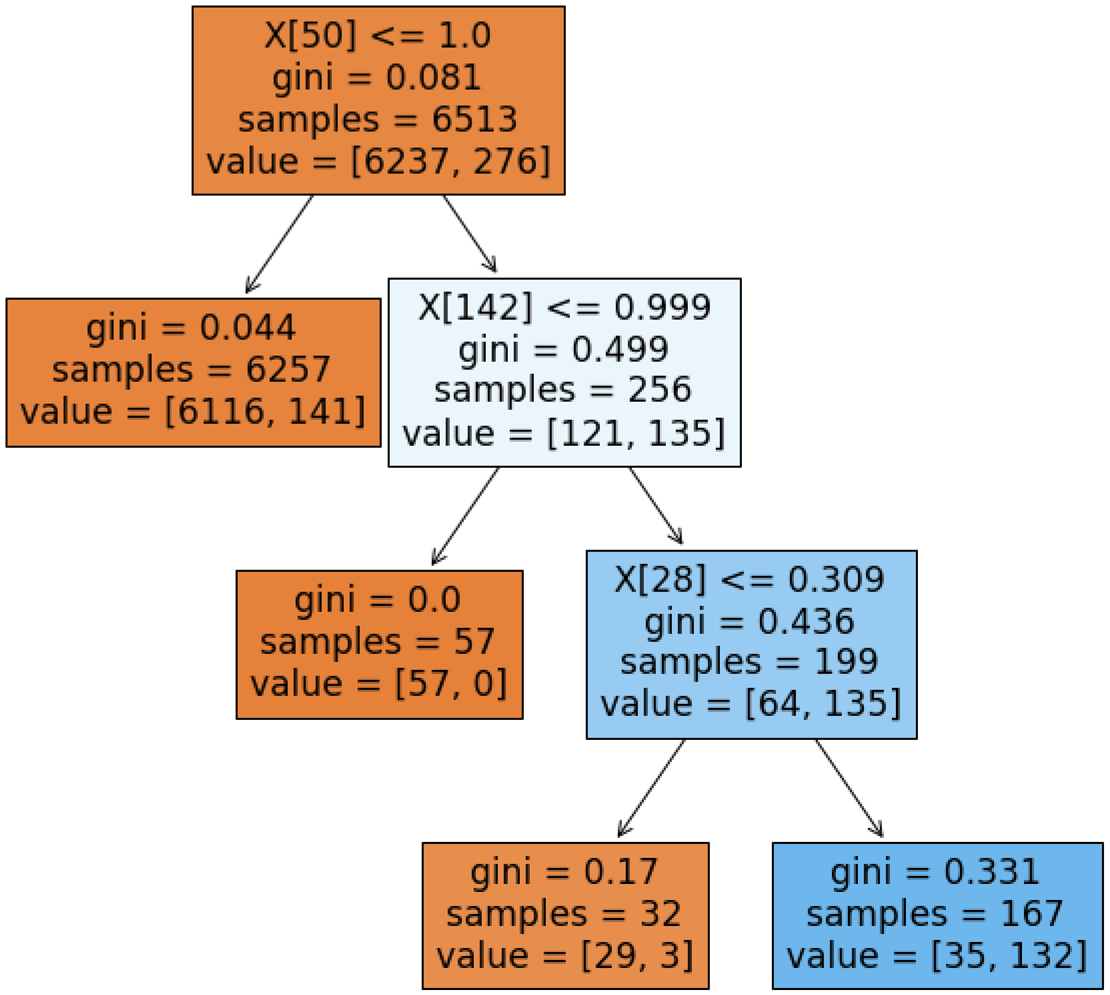


As you can see, greater CP values are corresponding to less complex trees with fewer nodes, shallower depth, and more impurities.
Now, it's time to choose the most suitable value for CP. We will pick the one that has the best accuracy on the test set.



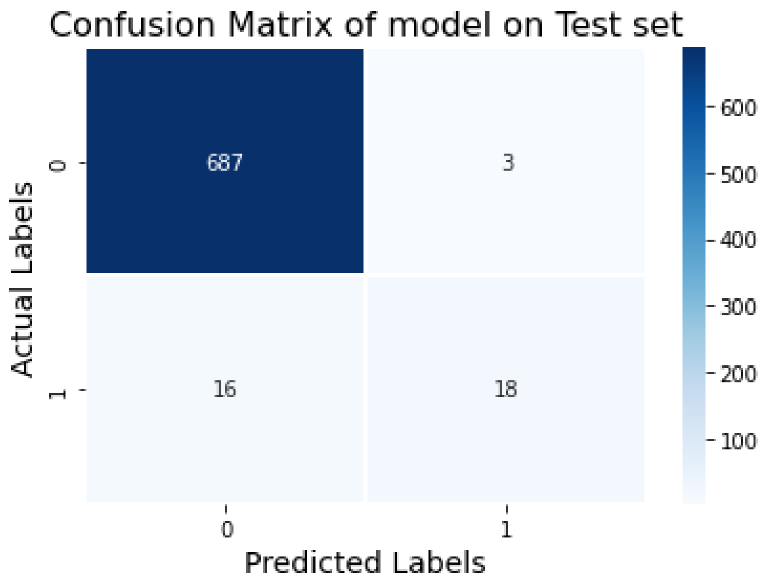
It appears that for α in range $0.0025 \pm \varepsilon$, test set has the best accuracy.

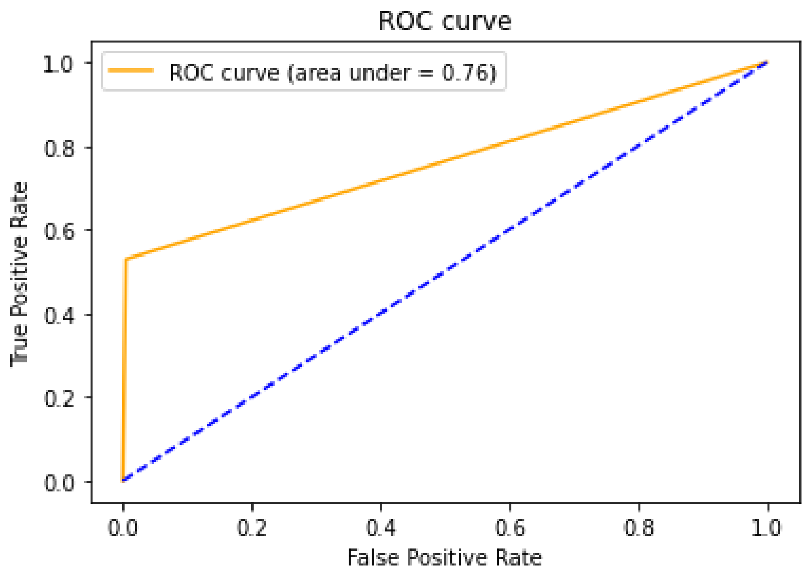
With Gini criteria and α_{CP}=0.0025, we have a tree as the following:



This tree has an accuracy of 97.3% on the test set. We also calculated the below metrics for the test data.

	Precision	Recall	F1-score	Support
On label 0	0.98	1.00	0.99	690
On label 1	0.86	0.53	0.65	34



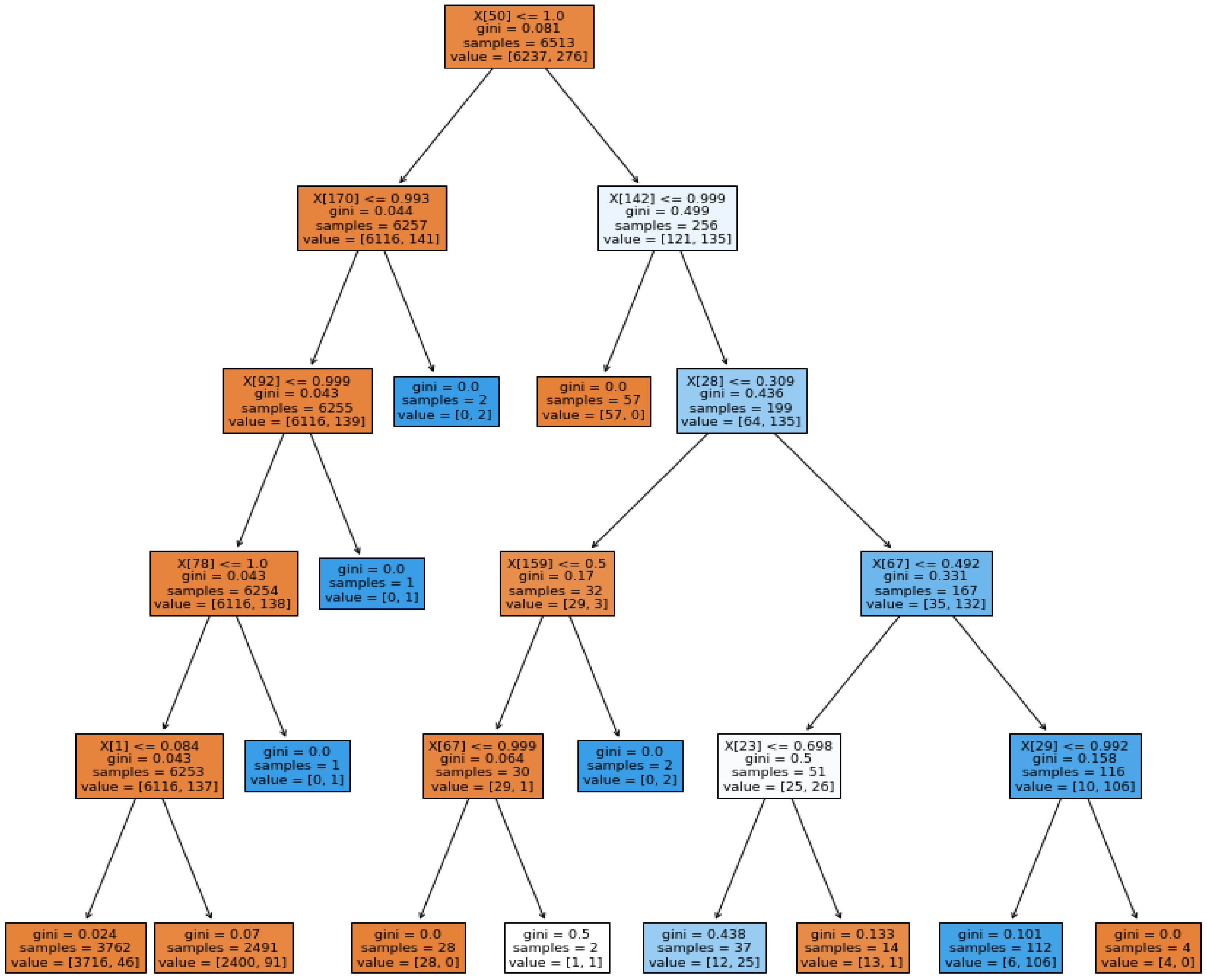


Another way to prevent overfitting is to test different tree configurations with a wide variety of depths and leaf nodes, and choose the setting with the most accurate result.
While we care about the most accurate results, we aim to choose the less complicated settings.

After testing trees with depth 3 to 19, and number of leaf nodes from 3 to 19, we got the below results.

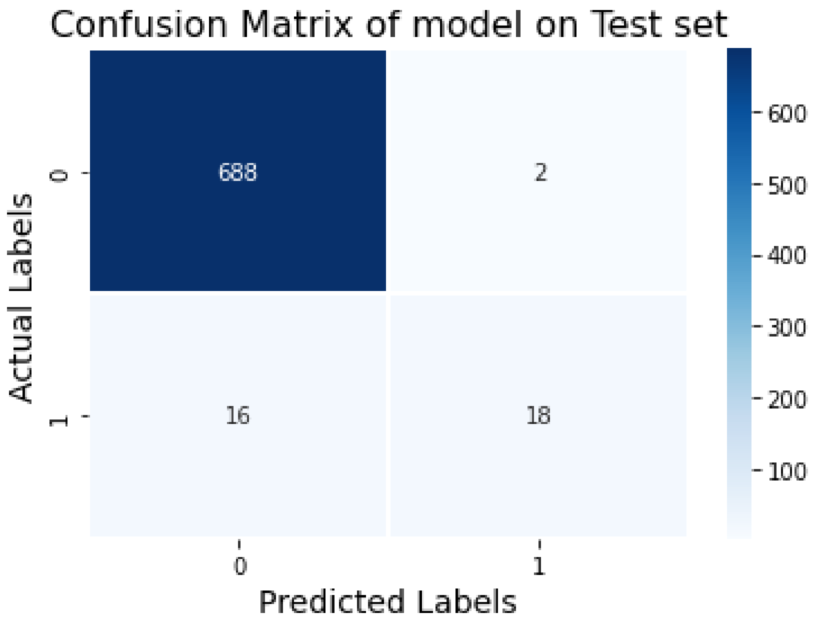
	Accuracy
Number of leaf nodes = 7	97.26%
Depth = 5	97.31%

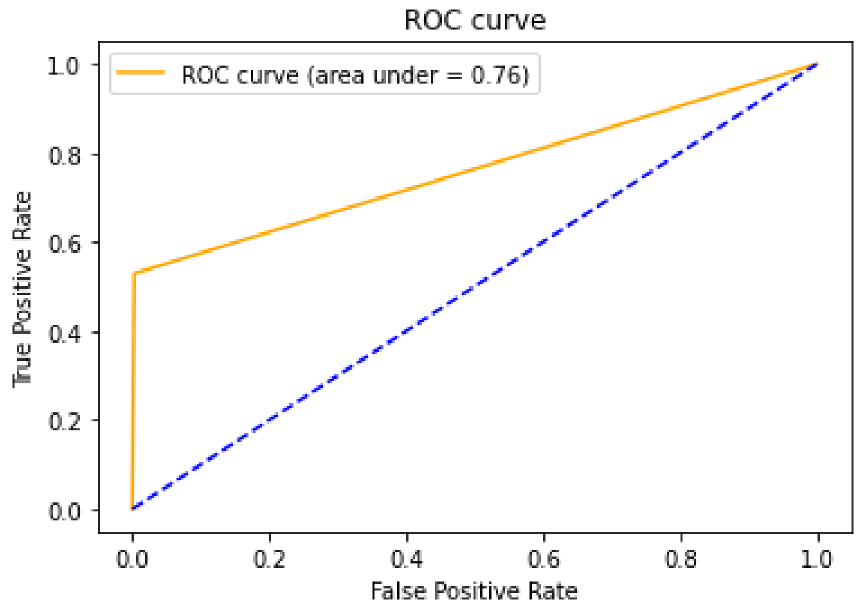
We decided to build a tree based on the depth parameter.


X[50] <= 1.0 gini = 0.081 samples = 6513 value = [6237, 276]
X[170] <= 0.993 gini = 0.044 samples = 6257 value = [6116, 141]
X[142] <= 0.999 gini = 0.499 samples = 256 value = [121, 135]
X[92] <= 0.999 gini = 0.043 samples = 6255 value = [6116, 139]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.0 samples = 57 value = [57, 0]
X[28] <= 0.309 gini = 0.436 samples = 199 value = [64, 135]
X[78] <= 1.0 gini = 0.043 samples = 6254 value = [6116, 138]
gini = 0.0 samples = 1 value = [0, 1]
X[159] <= 0.5 gini = 0.17 samples = 32 value = [29, 3]
X[67] <= 0.492 gini = 0.331 samples = 167 value = [35, 132]
X[1] <= 0.084 gini = 0.043 samples = 6253 value = [6116, 137]
gini = 0.0 samples = 1 value = [0, 1]
X[67] <= 0.999 gini = 0.064 samples = 30 value = [29, 1]
gini = 0.0 samples = 2 value = [0, 2]
X[23] <= 0.698 gini = 0.5 samples = 51 value = [25, 26]
X[29] <= 0.992 gini = 0.158 samples = 116 value = [10, 106]
gini = 0.024 samples = 3762 value = [3716, 46]
gini = 0.07 samples = 2491 value = [2400, 91]
gini = 0.0 samples = 28 value = [28, 0]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.438 samples = 37 value = [12, 25]
gini = 0.133 samples = 14 value = [13, 1]
gini = 0.101 samples = 112 value = [6, 106]
gini = 0.0 samples = 4 value = [4, 0]

This tree has an accuracy of 97.51% on the test set. We also calculated the below metrics for the test data.

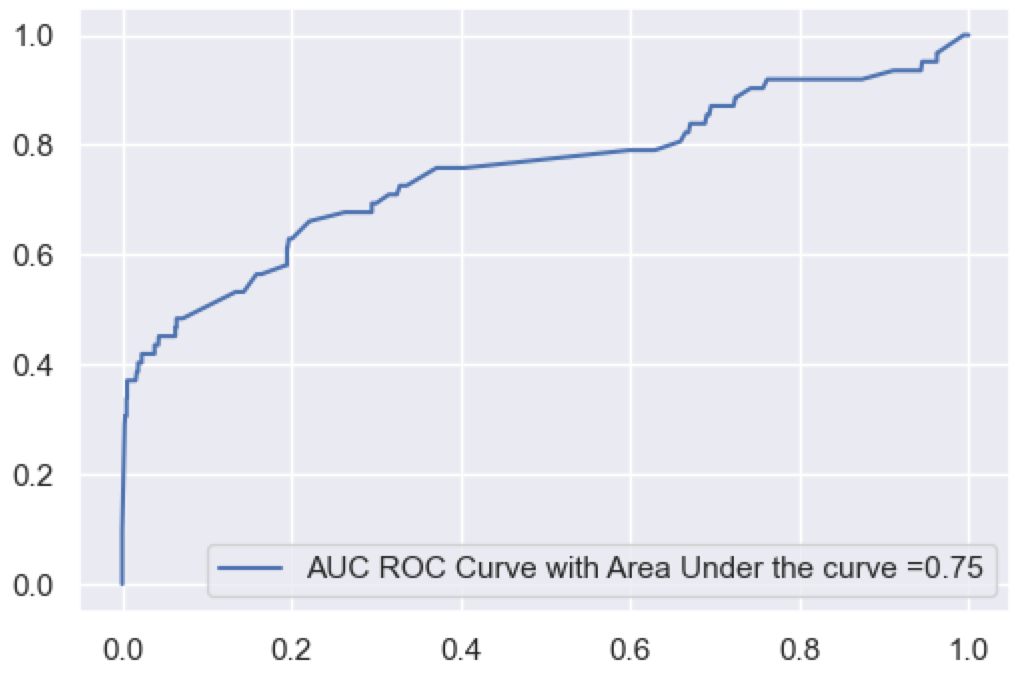
	Precision	Recall	F1-score	Support
On label 0	0.98	1.00	0.99	690
On label 1	0.90	0.53	0.67	34





2.6 Adaboost

As we may know, this algorithm will not give us enough information on how our model is degenerated and this leads us to just seek for better accuracy with tuning large of hyper parameter such as n_estimater which shows us the maximum number of estimators for the whole model for Eg: n decision trees with a max depth of 1. This is a base model of how the algorithm boosts the weights toward a stronger model.
After tuning the model with different hyper parameters(Base, n_estimate, learning_rate). The result of our model is the following plot

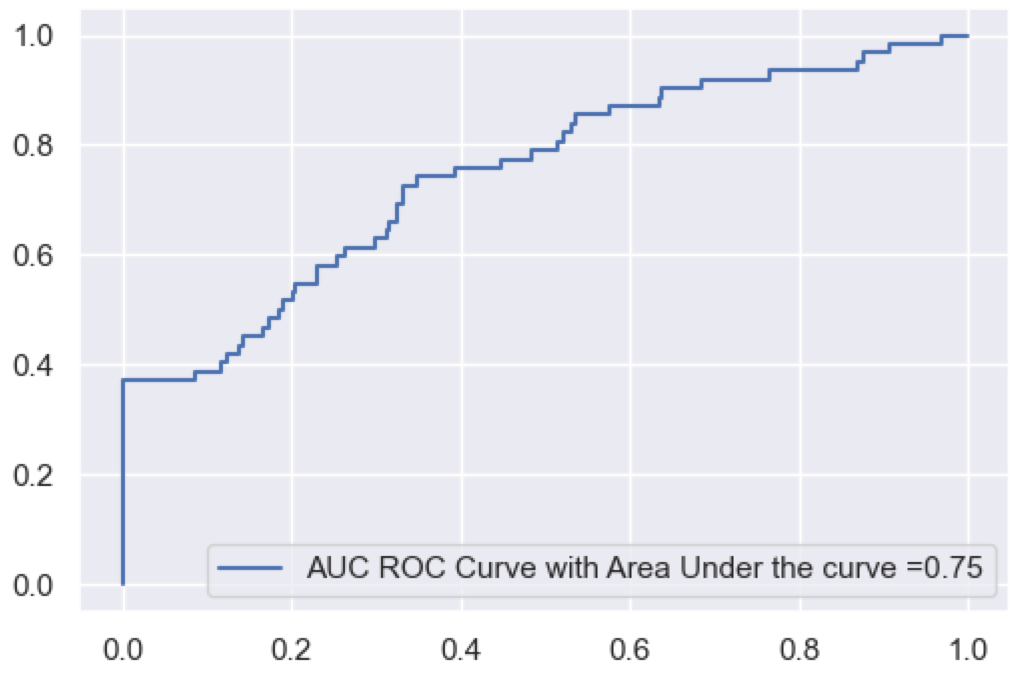


This plot shows that the model works totally fine, the training score gets to 88, and we can see the overfitting very well which was probable in ensemble models.

2.7 Random Forest

This a Bunch of trees are bagged and the final labels are voted by shallow trees this simple model is very strong model in machine learning and lead us to win competition as well.

In Tox21 we Tune random forest with different hyper parameters.
Like bootstrap, max depth, max features, min_samples_leaf, min_samples_split.
These hyper parameter is good for tuning our trees we use grid search with n_split = 5 and roc_auc area as our score.



With training score of 0.99 this score shows us intuition of overfitting in this model so we should change our trees
We chose our max_depth to be 1 and the results increases so good
We get 79 and 80 for our testing and training data

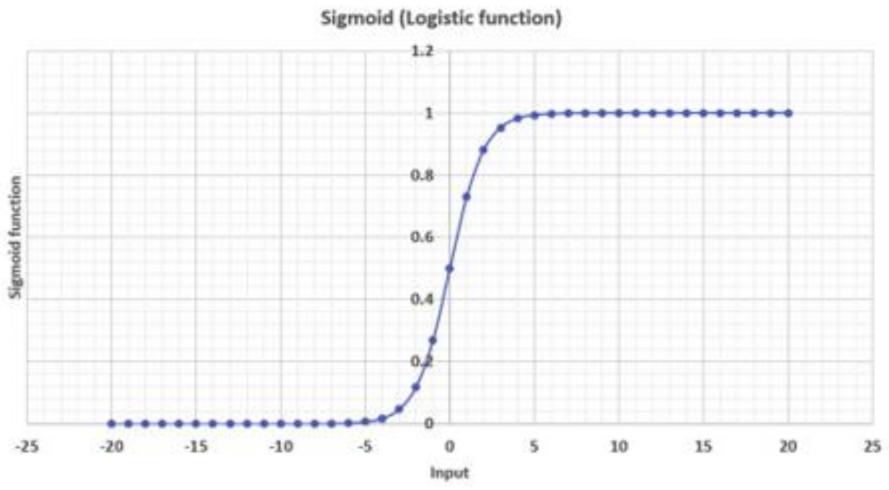
2.8 Logistic Regression and perceptron

Logistic Regression without Equalizing

Logistic regression is a powerful supervised ML algorithm which we use for predicting the binary labels of Tox21 dataset.

Logistic function:

$$Sigmoid\ Logistic\ function = \frac{1}{1+e^{-x}}$$



Logistic regression loss function cross entropy:

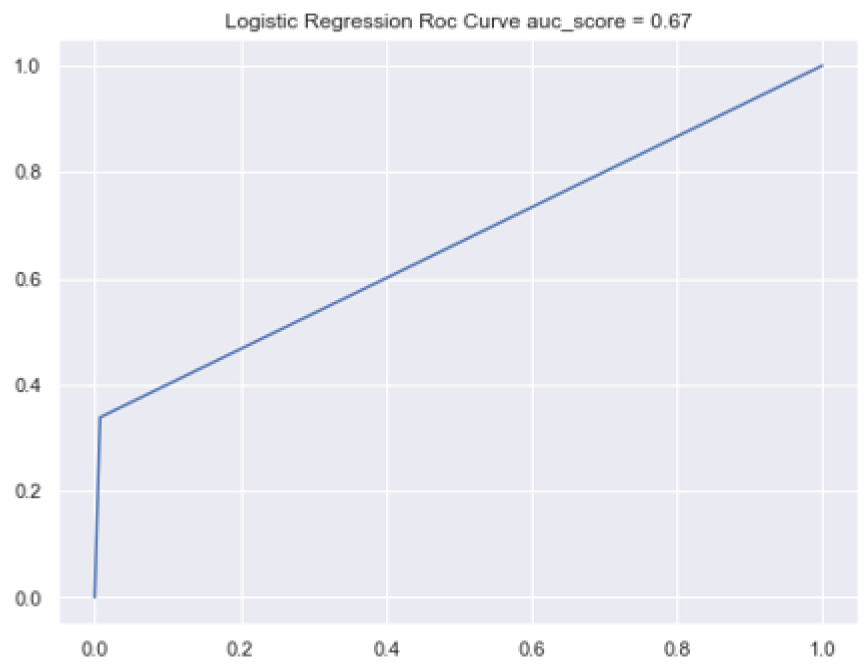
$$-\frac{1}{N}\sum_{i=1}^{N} y_i\, log\big(P(y_i)\big) + (1-y_i)\, log\big(1-P(y_i)\big)$$

There are 190 features in tox21 dataset, and we are going to work with a single label (pre-processed data).

We split the data with 80% as train dataset and 20% as test. We apply StandardScaler before feeding it to the models and use stratify to ensure that there is always a portion of positive in both train and test data (only a small proportion of our data set is labeled 1).

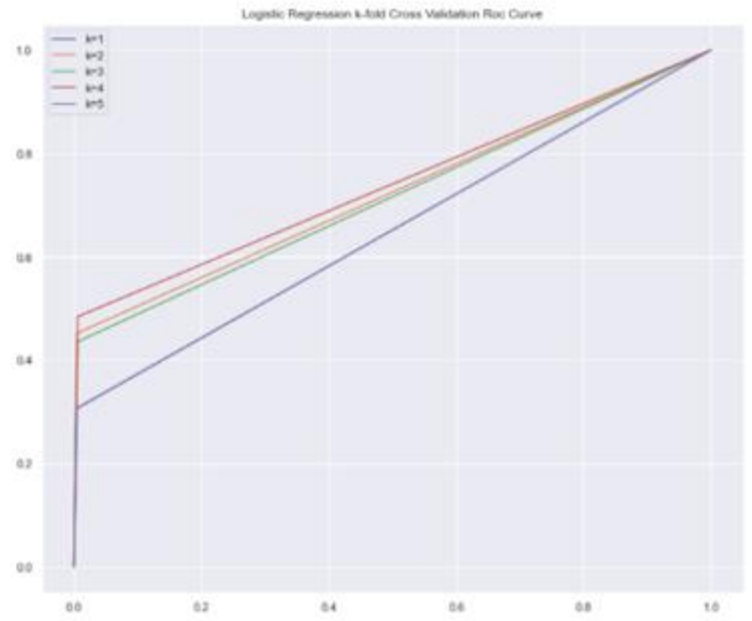
```
Logistic Regression recall score = 0.3387

Logistic Regression classification report:
              precision    recall  f1-score   support

        safe       0.97      0.99      0.98      1386
       toxic       0.68      0.34      0.45        62

    accuracy                           0.96      1448
   macro avg       0.82      0.67      0.72      1448
weighted avg       0.96      0.96      0.96      1448
```

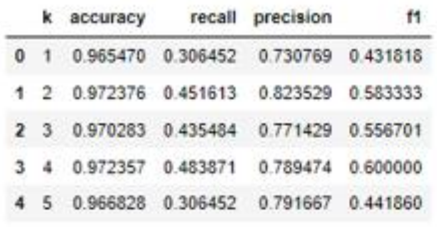
Please note recall score for toxic label (=1) is 0.34, although we've got a 0.96 accuracy but this is because our data is not balanced and we'll deal with this problem later.



Logistic Regression with cross-validation (n_splits = 5)

We will use StratifiedKfold to ensure that each fold of dataset has the same proportion of observations given a label





	k	accuracy	recall	precision	f1
0	1	0.965470	0.306452	0.730769	0.431818
1	2	0.972376	0.451613	0.823529	0.583333
2	3	0.970283	0.435484	0.771429	0.556701
3	4	0.972357	0.483871	0.789474	0.600000
4	5	0.966828	0.306452	0.791667	0.441860

Perceptron

The Perceptron algorithms is a two-class (binary) classification machine learning algorithms.

It is a very simple type of neural network model (single neuron that takes a row of data as input and predicts a class label.

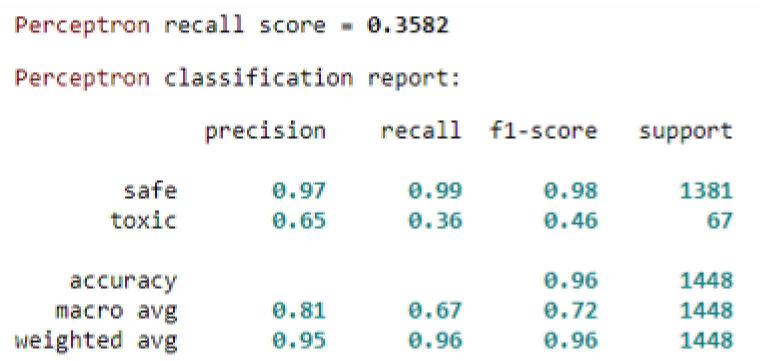
*Activation = (Weights*Inputs) + Bias*

if Activation > 0 : Predict 1

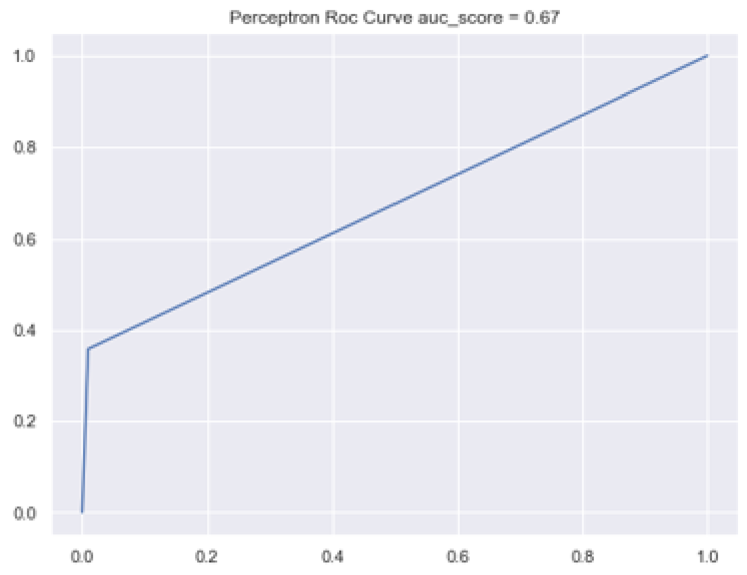
if Activation < 0 Predict 0

Note: given that the inputs are multiplied by model coefficients, like linear and logistic regression, it is a good idea to standardize data prior to using the model

Note 2: this part is done without balancing the data.



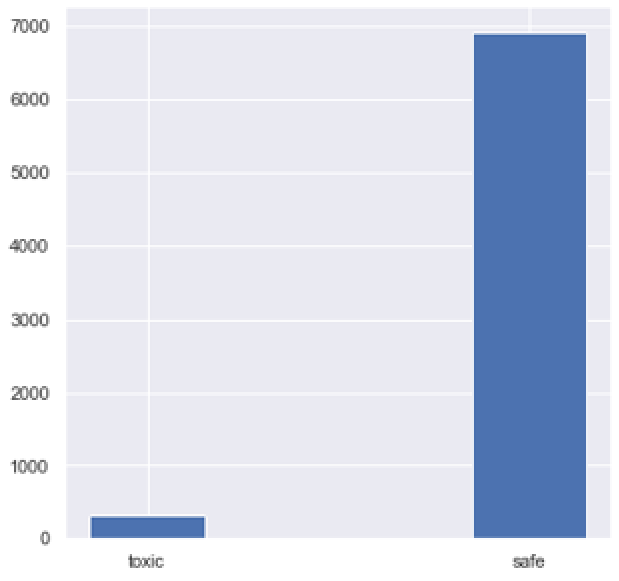
```
Perceptron recall score = 0.3582

Perceptron classification report:

              precision    recall  f1-score   support

        safe       0.97      0.99      0.98      1381
       toxic       0.65      0.36      0.46        67

    accuracy                           0.96      1448
   macro avg       0.81      0.67      0.72      1448
weighted avg       0.95      0.96      0.96      1448
```



Please note recall score for toxic labels is reported 0.36 (a bit lower than logistic regression).

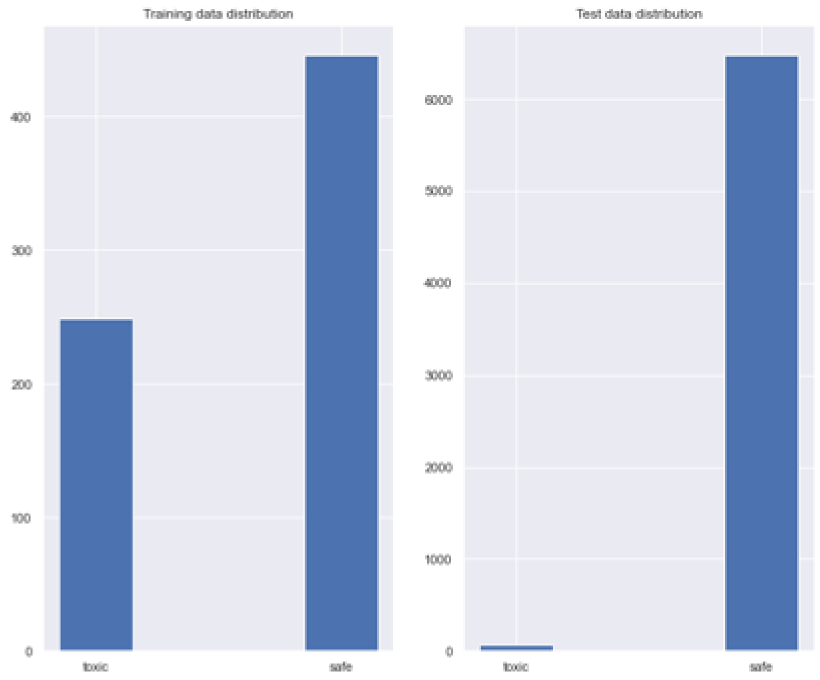
Equalizing the data distribution

At first glance, the models work 'ok' since the overall accuracy is 97% but are they really?! Our purpose here is to train a model to distinct toxic gasses from the safe ones so we need a model that precisely detects the toxic gasses and doesn't label a toxic gas as non-toxic hence we need to maximize 'True Positives' and minimize the 'False Negatives' in our model and since then we need a model that it's recall score is high but as we can see most of the time (based on the data distribution in train and test dataset) our recall score is even below 0.3, why is this happening?



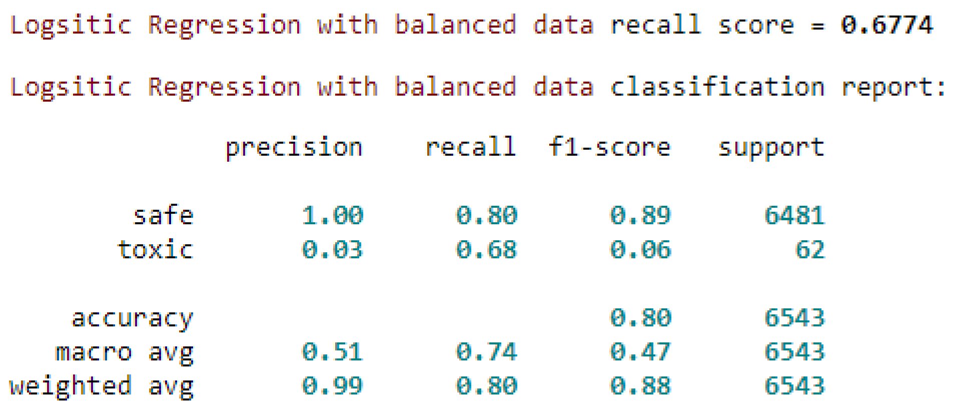
As we can see, our dataset distribution is quite uneven and that's why the recall score for models is so low. In this section we try to even out the distribution of the training dataset.

After balancing the train dataset:



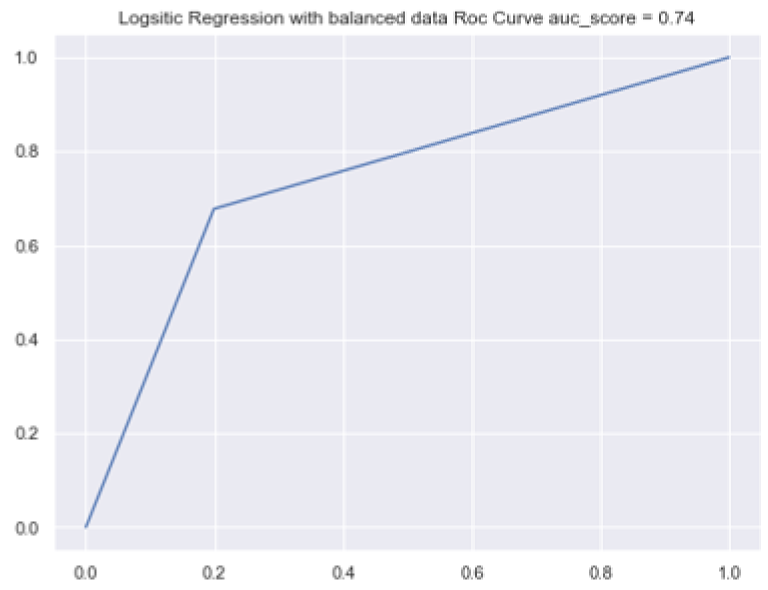
We have chosen 80 percent of our positive cases for our train data set and we have chosen a sample size of 180 percent of size of positive cases from negative cases; the rest is for testing the models.

Logistic Regression with balanced data

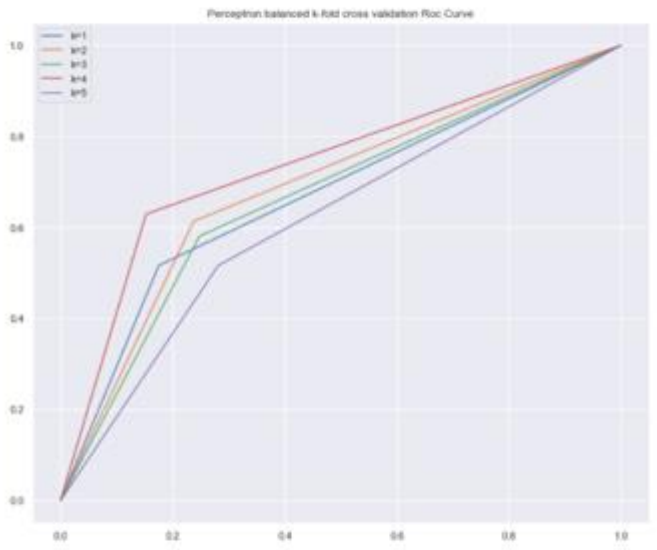


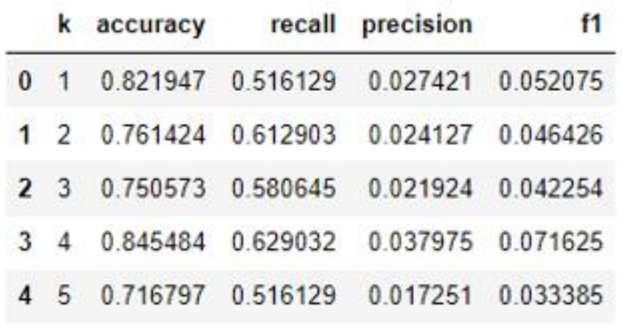
```
Logsitic Regression with balanced data recall score = 0.6774

Logsitic Regression with balanced data classification report:
              precision    recall  f1-score   support

        safe       1.00      0.80      0.89      6481
       toxic       0.03      0.68      0.06        62

    accuracy                           0.80      6543
   macro avg       0.51      0.74      0.47      6543
weighted avg       0.99      0.80      0.88      6543
```

Please note the significant increase in recall score 0.6774.



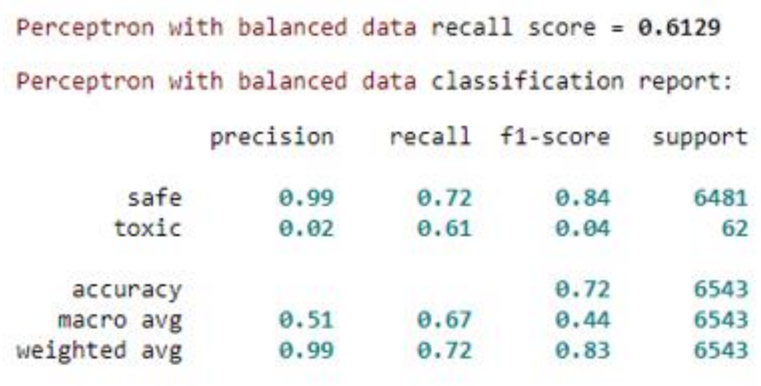
Logistic Regression with balanced data and cross validation





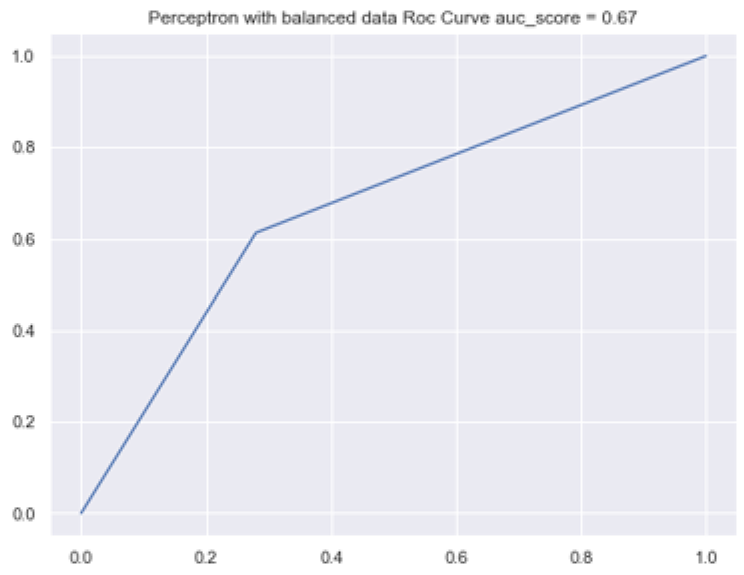
	k	accuracy	recall	precision	f1
0	1	0.821947	0.516129	0.027421	0.052075
1	2	0.761424	0.612903	0.024127	0.046426
2	3	0.750573	0.580645	0.021924	0.042254
3	4	0.845484	0.629032	0.037975	0.071625
4	5	0.716797	0.516129	0.017251	0.033385

Perceptron with balanced data

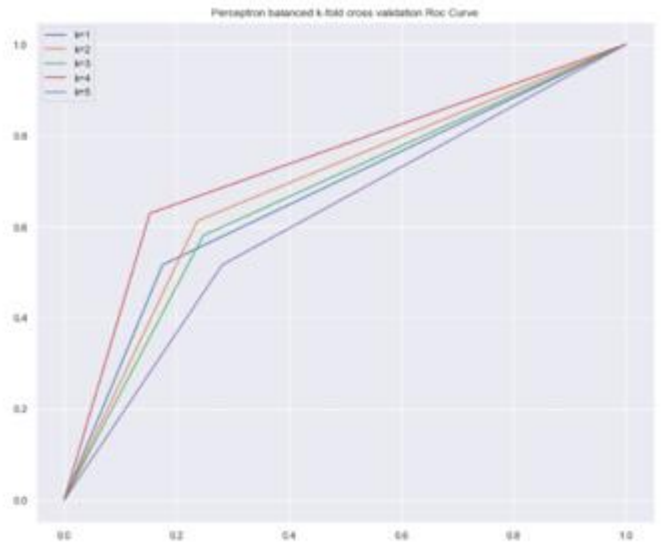


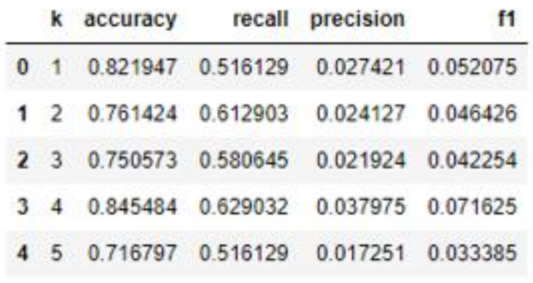
```
Perceptron with balanced data recall score = 0.6129

Perceptron with balanced data classification report:

              precision    recall  f1-score   support

        safe       0.99      0.72      0.84      6481
       toxic       0.02      0.61      0.04        62

    accuracy                           0.72      6543
   macro avg       0.51      0.67      0.44      6543
weighted avg       0.99      0.72      0.83      6543
```

As we expected, recall score is increased again and reported 0.6129.



Perceptron with balanced data and cross validation





	k	accuracy	recall	precision	f1
0	1	0.821947	0.516129	0.027421	0.052075
1	2	0.761424	0.612903	0.024127	0.046426
2	3	0.750573	0.580645	0.021924	0.042254
3	4	0.845484	0.629032	0.037975	0.071625
4	5	0.716797	0.516129	0.017251	0.033385

3. ToxCast Dataset

3.1 Preprocessing

3.1.1 Labels

We have 618 labels, and we want to choose one of them. First, we take a look at Na values and the minimum is 645 in 8576 rows. So, we chose among these, and most of our data are badly imbalanced; so, we choose the best one which is TOX21_TR_LUC_GH3_Antagonist.

We acknowledged that 28 percent of data has value 1.

3.1.2 Training

We start looking at our shape which is (8576,201).
So our data has good volume (not enough large to be considered as big data and not enough small). But very large columns.

3.1.2.1 Handling missing values

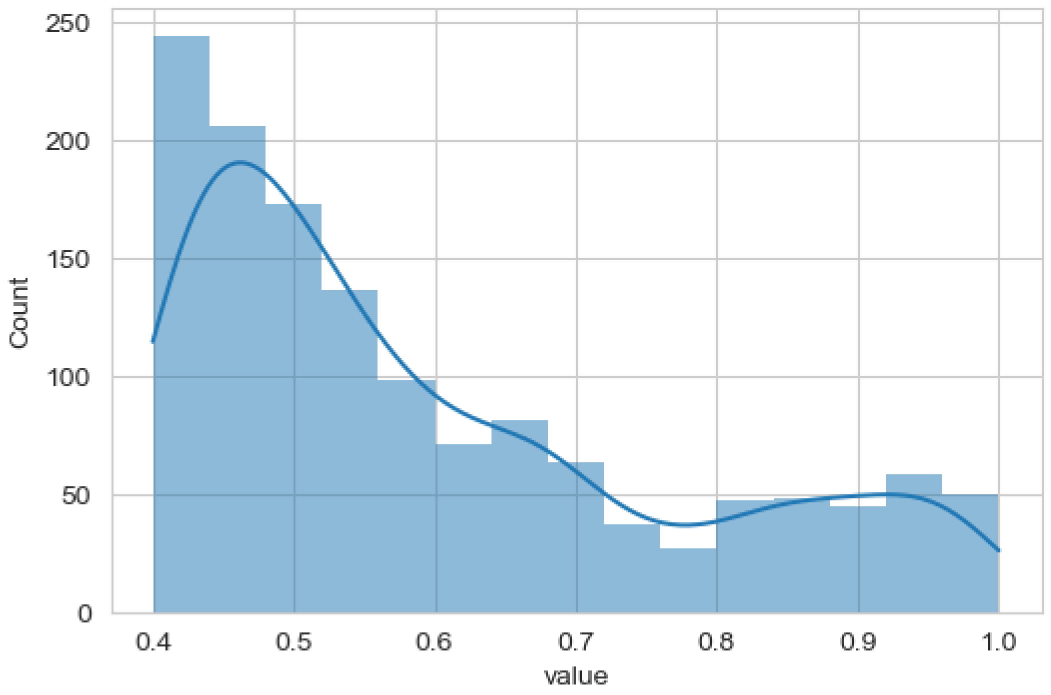
We have about 1081 null values in the whole dataframe (109, 109, 109, 109, 645 in columns)
We will remove them with no doubt just a small portion of the data has nun values so we will remove that points.

3.1.2.2 Seeking for features

There are several constant values that we remove.

3.1.2.2 Removing Correlate data

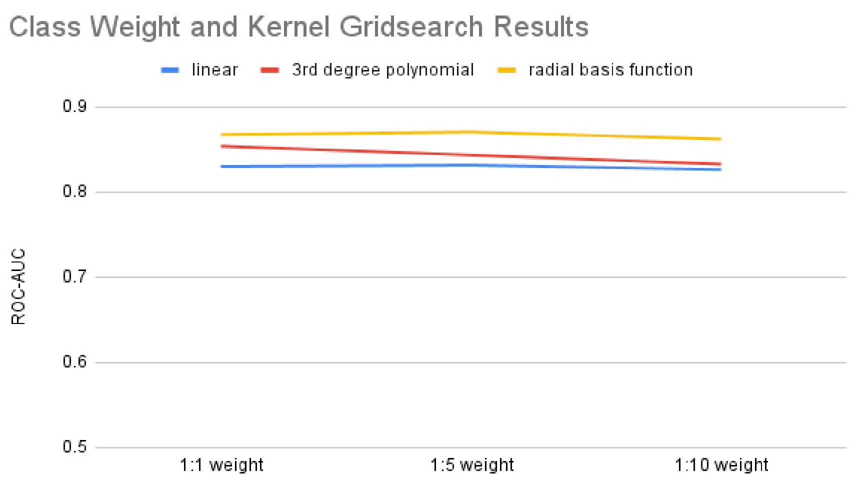
According to the below figure, we have many columns with over 0.8 correlation with removing these columns the information would not be changed But for achieving more accurate models we do not do this.

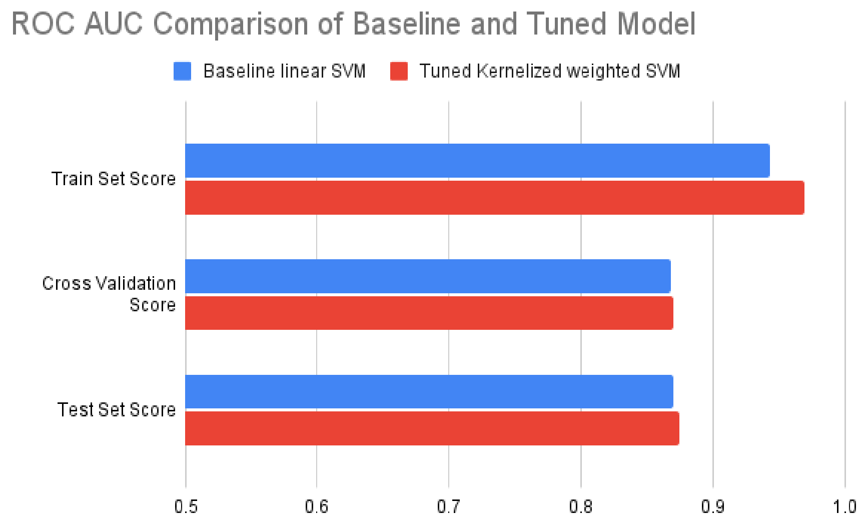


Finally, we will get to 7831 * 135 data in our Toxcast, and TOX21_TR_LUC_GH3_Antagonist as our labels.

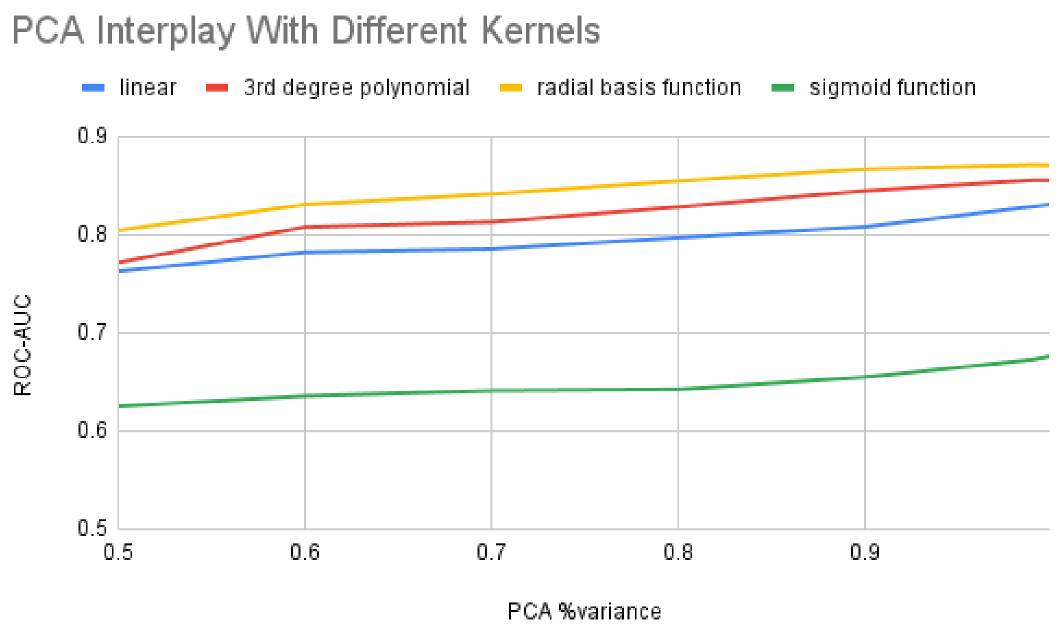
3.2 Support Vector Classification

The selected task from the ToxCast dataset is more balanced than the tox21 task, but there is still around a 1 to 5 class imbalance. We performed the identical procedure as the tox21 SVM model and got similar results.





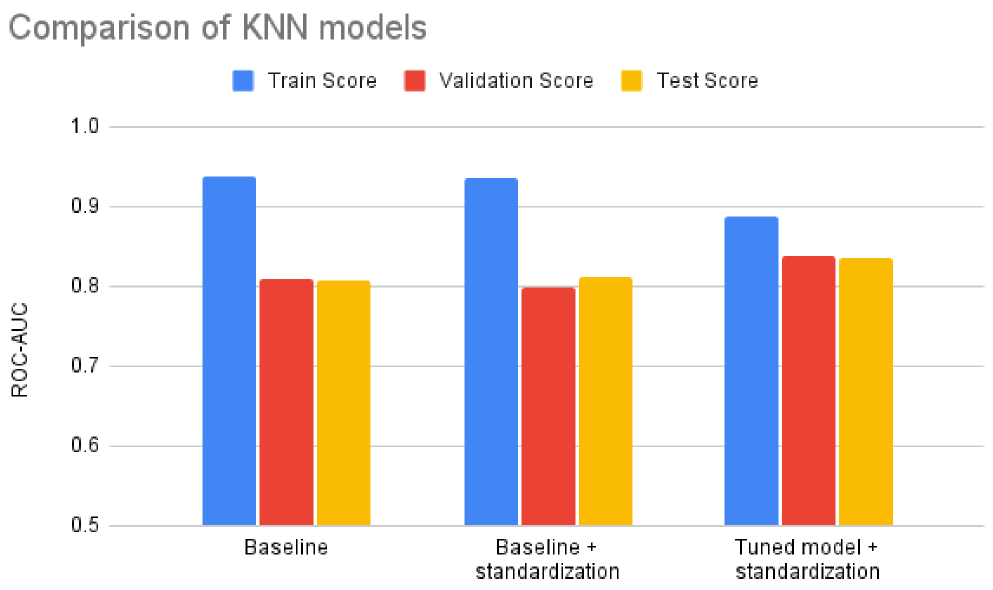
As a secondary objective, we investigate the choice of kernel and how it interplays with different levels of dimensionality reduction. For this reason, a stratified grid search is employed to look for the pattern in growth of roc-auc of the model with increasingly higher levels of variance present in the data.



It is interesting to observe that even though different kernels map the data to different feature spaces, there is strict monotonicity in the roc-auc score of the model with increasing the amount of information in the data. More interestingly, different kernels do not gain an advantage over each other in different levels of variance.

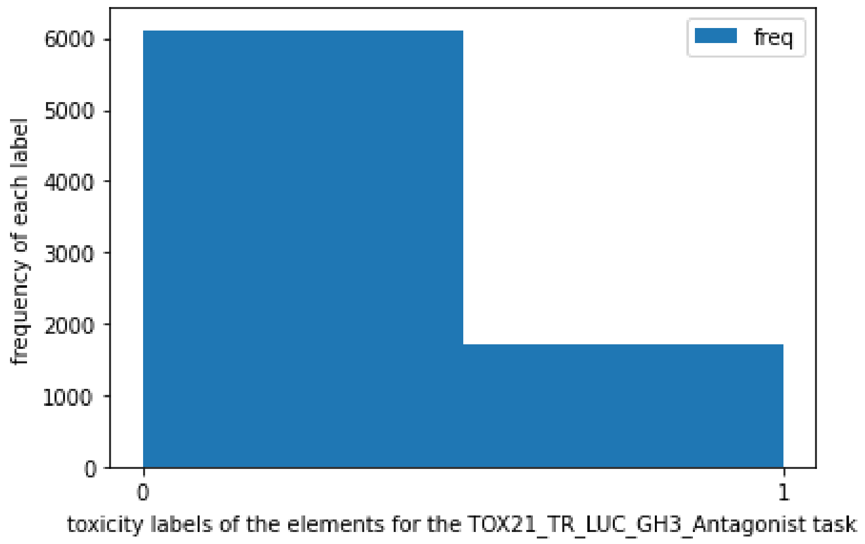
3.3 K-Nearest Neighbors

We employed a similar approach to the tox21 KNN model, with the same grid search methodology and got similar results. However, the scores in this task are higher, this can be due to more feature pruning in the preprocessing of ToxCast.



3.4 Neural Network

In this section, we are going to implement a Multi-Layer Neural Networks algorithm to predict whether the chemical compounds are toxic or not.
First, we checked the distribution of labels.

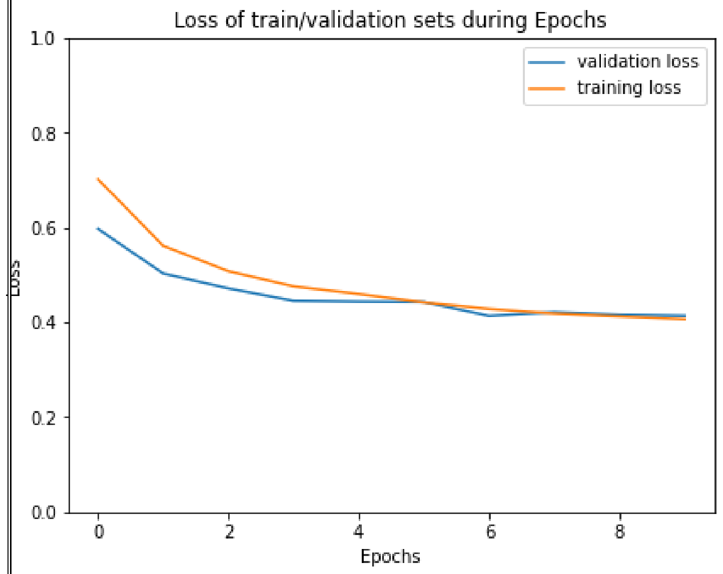


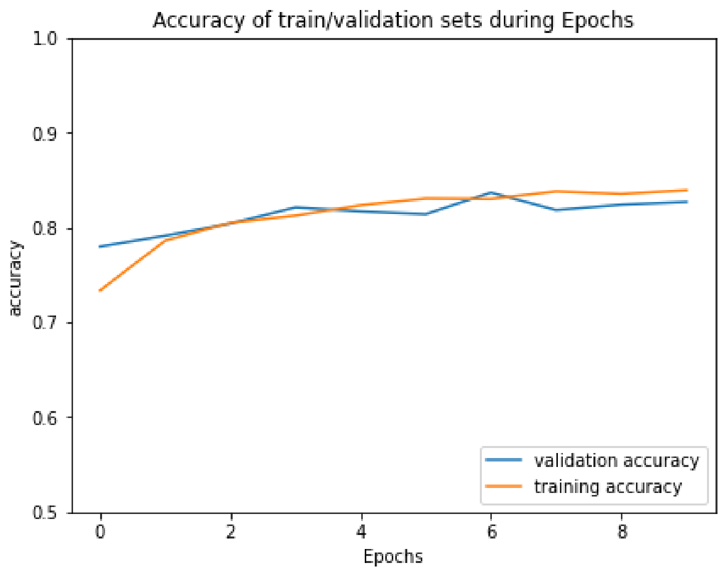
As you can see, labels are highly imbalanced. Regarding the distribution of our labels, we are prone to overfitting. So, to mitigate this issue, we will use K-fold cross-validation alongside dropout layers and weight regularizers for each layer with a penalty of 0.001.
Neural Network architecture:

	Number of Neurons	**Activation Function**
Layer 1	**64**	**Relu**
Drop out Probability 1	**0.3**	
Layer 2	**32**	**Relu**
Layer 3	**16**	**Relu**
Drop out Probability 2	**0.1**	
Layer 4	**1**	**Sigmoid**

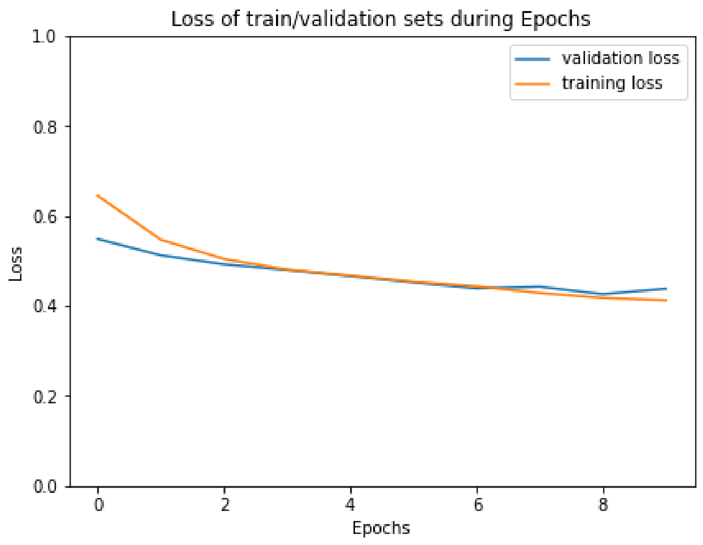
We trained our model in a stratified 10-fold cross-validation manner. In building each fold, stratified sampling was used to keep the distribution of the original data.
You can see the performance of the model on each fold-over different epochs:

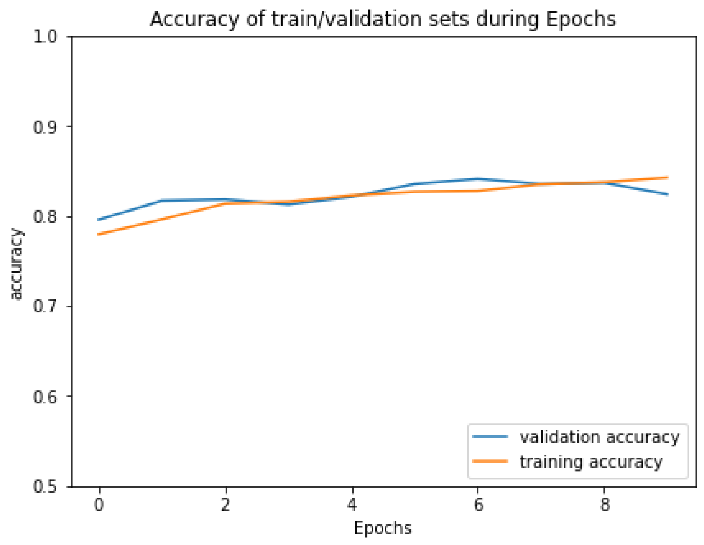
Fold 1


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

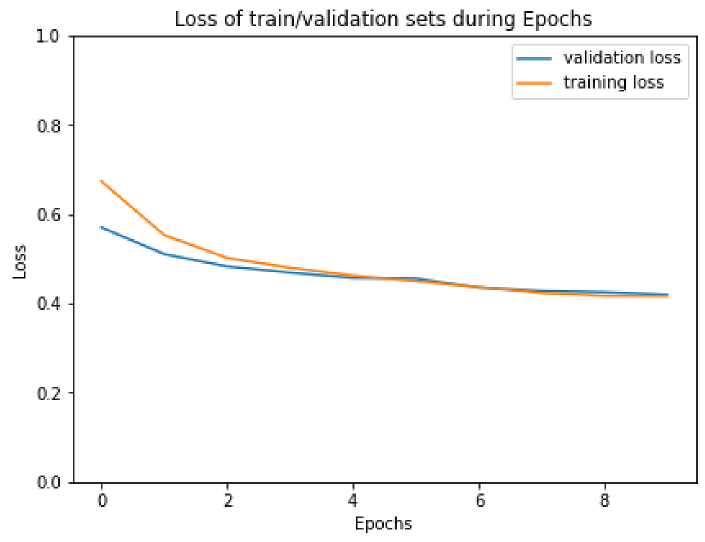

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

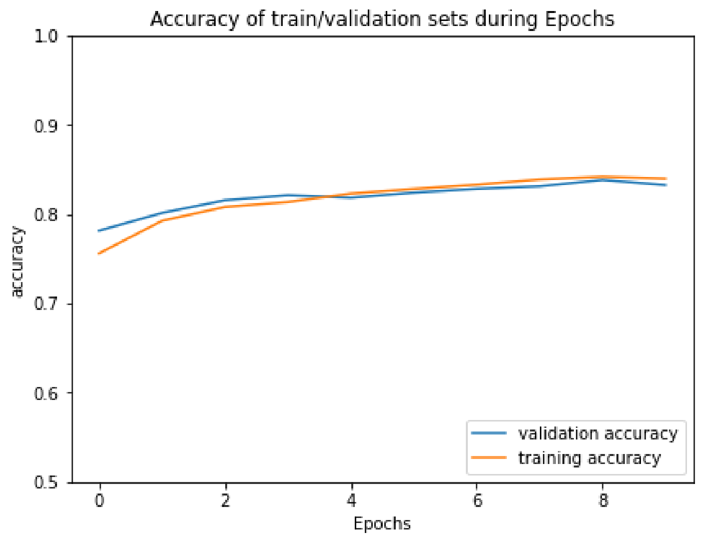
Fold 2


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

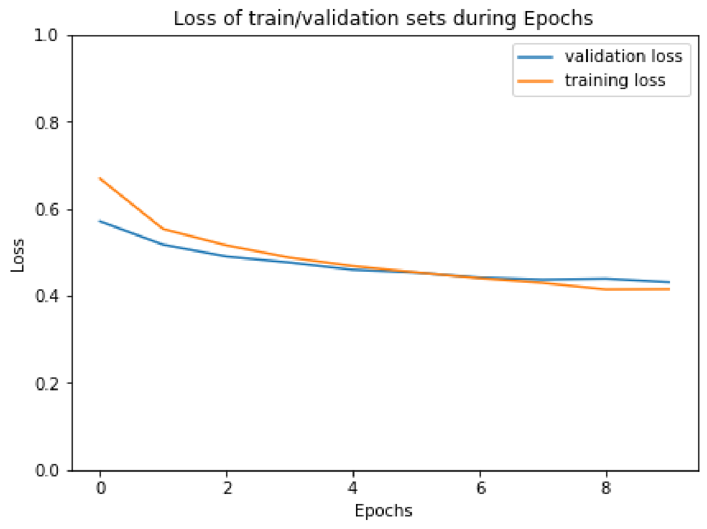

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

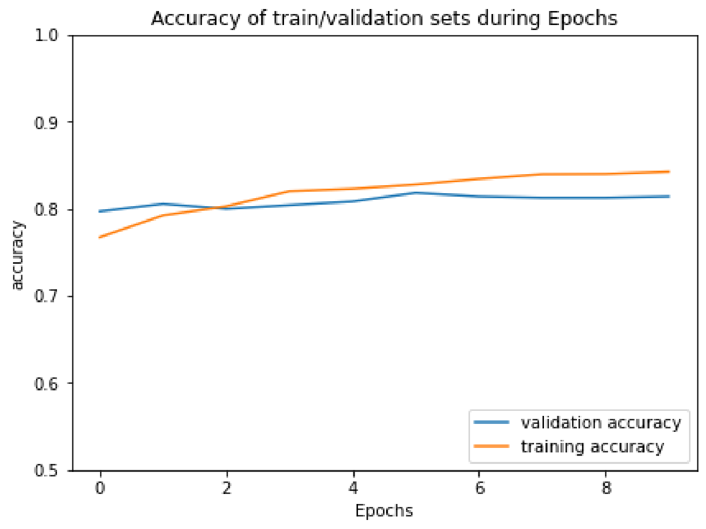
Fold 3


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

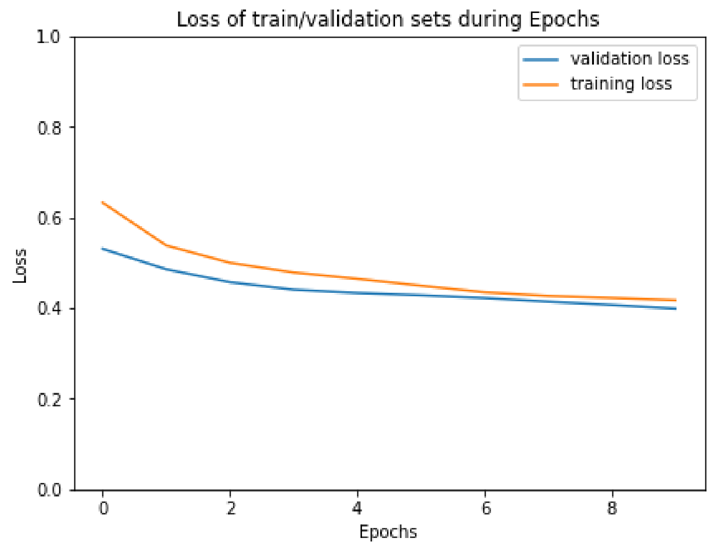

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

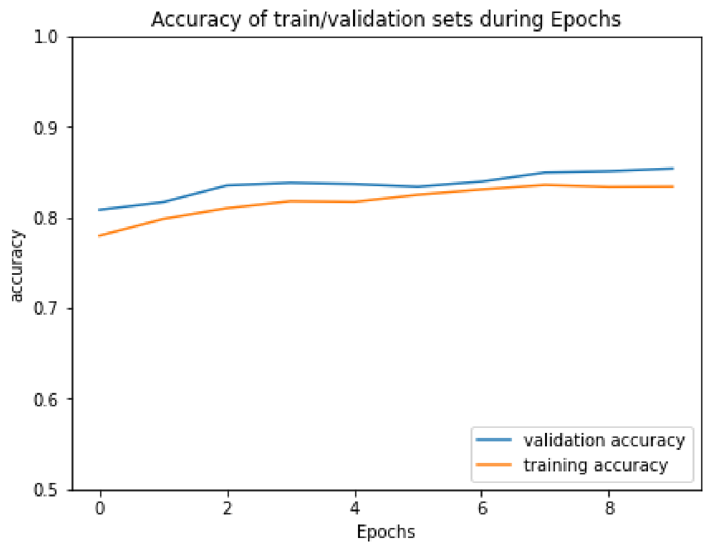
Fold 4


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

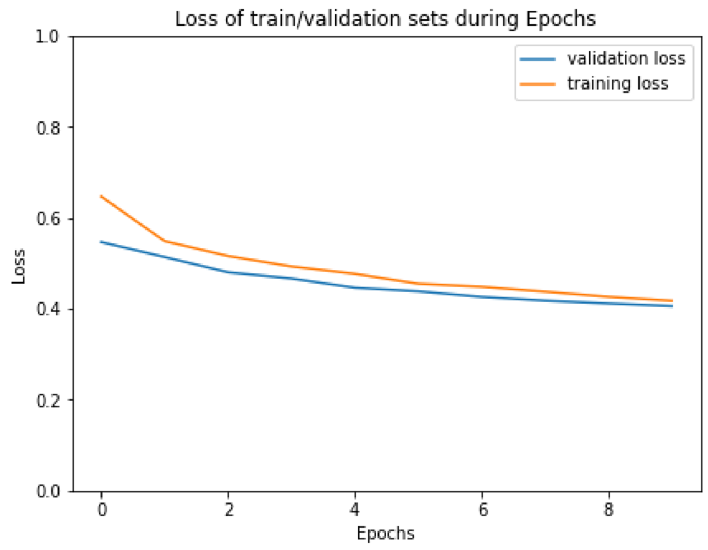

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

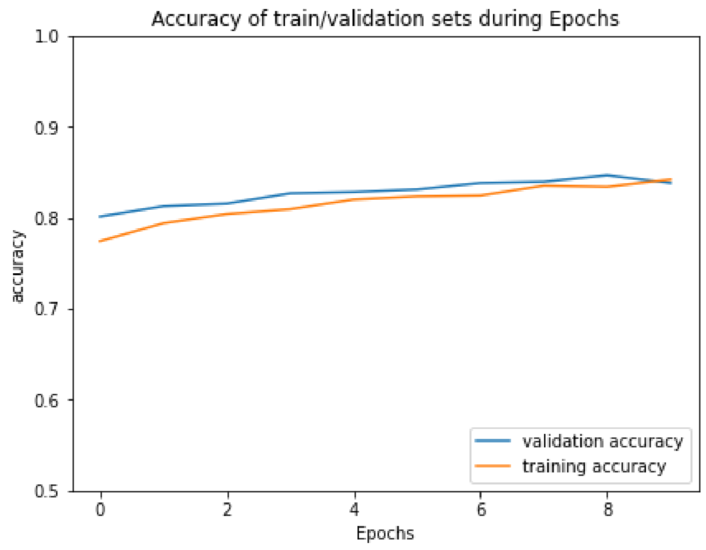
Fold 5


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0

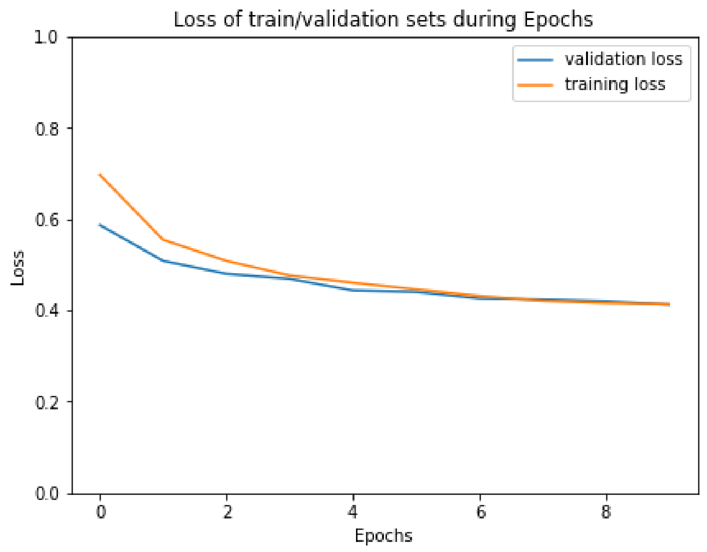

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0
2
4
6
8
0.5
0.6
0.7
0.8
0.9
1.0

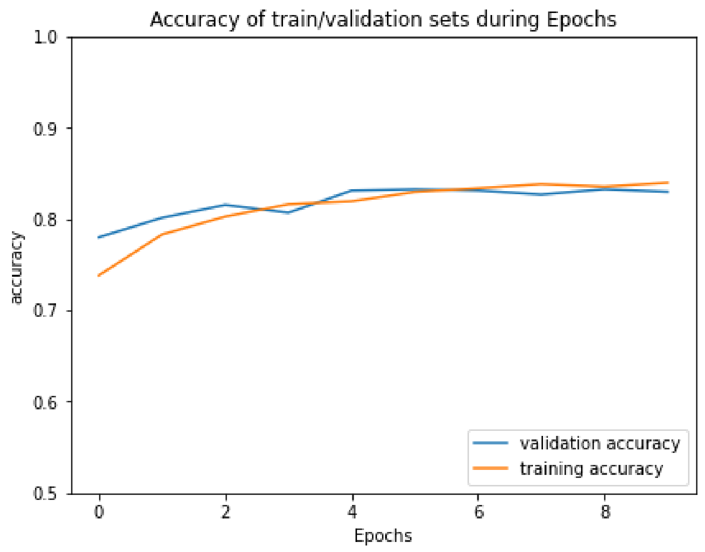
Fold 6


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0

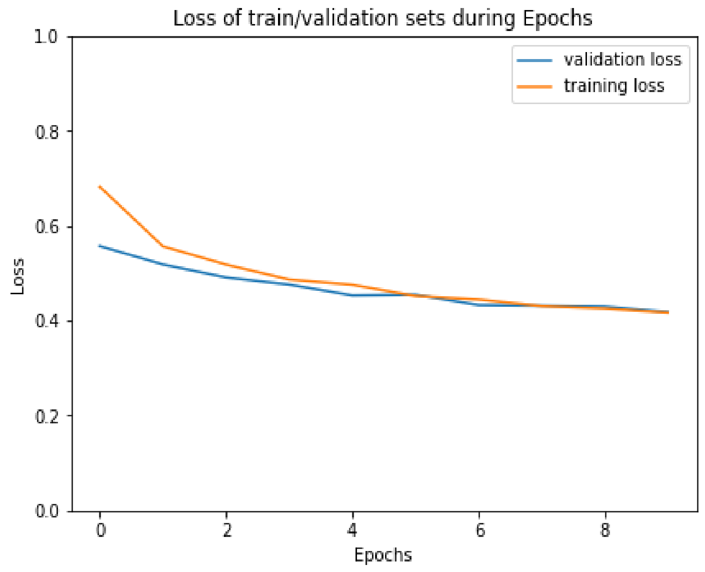

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0
2
4
6
8
0.5
0.6
0.7
0.8
0.9
1.0

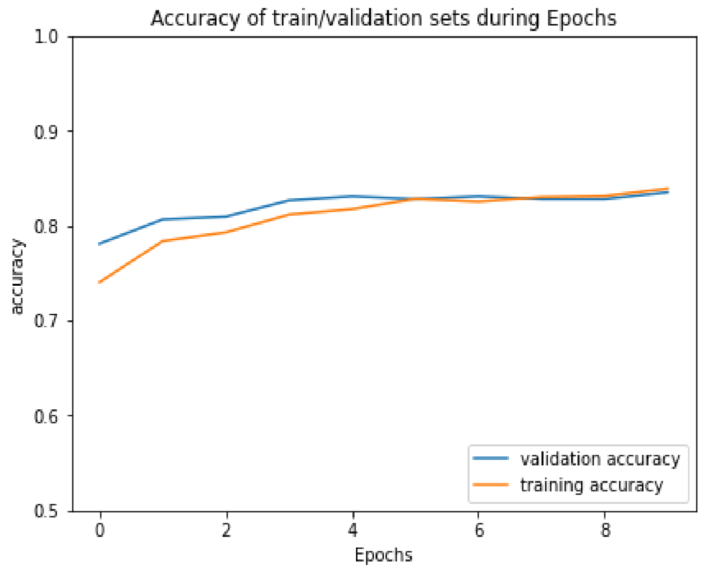
Fold 7


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

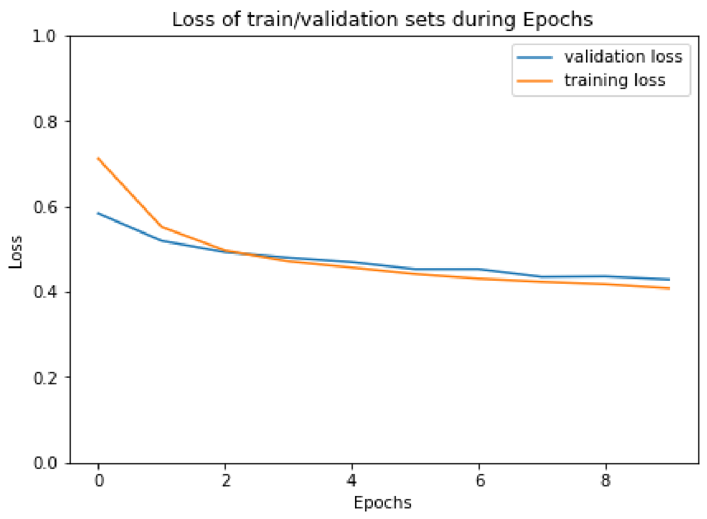

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

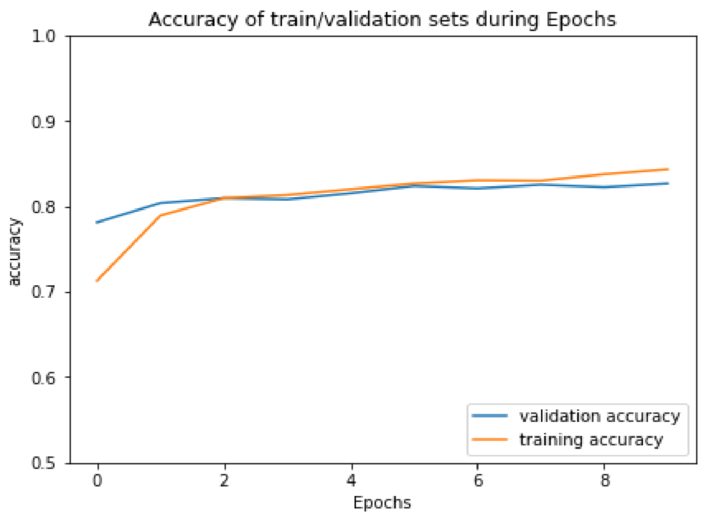
Fold 8


Loss of train/validation sets during Epochs
validation loss
training loss
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8

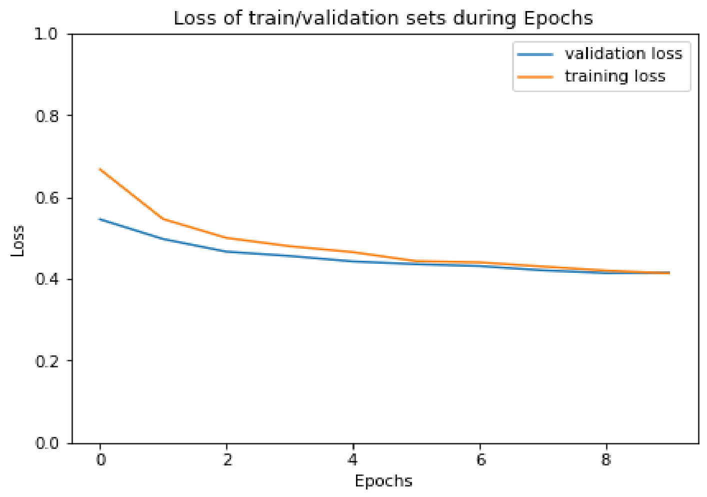

Accuracy of train/validation sets during Epochs
validation accuracy
training accuracy
accuracy
Epochs
0.5
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8

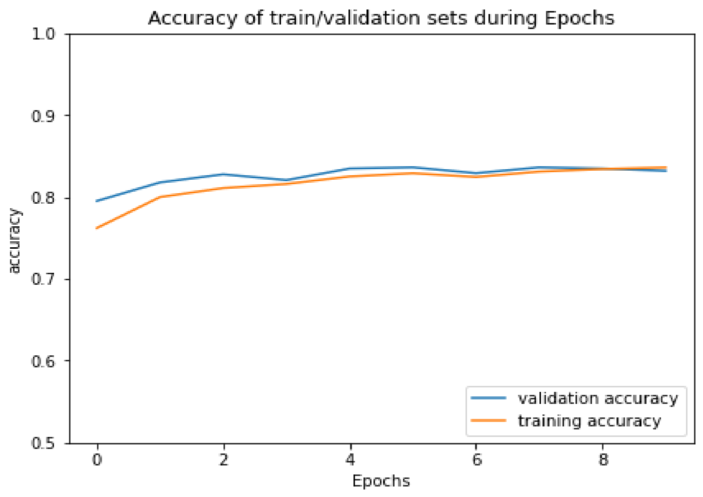
Fold 9





Fold 10





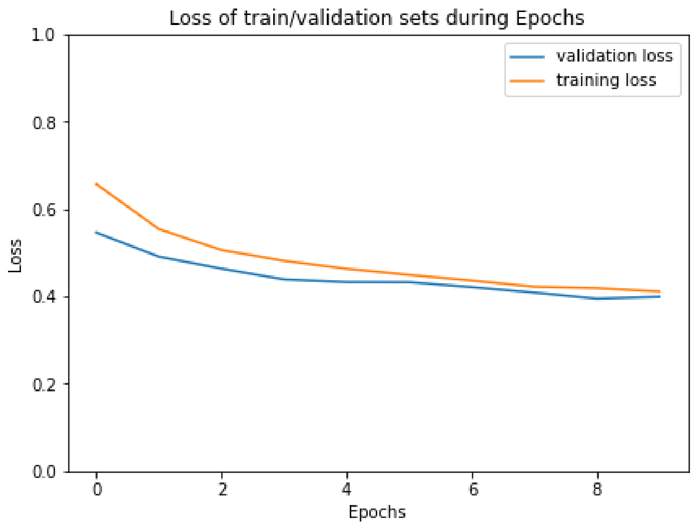
The average accuracy for all folds is:

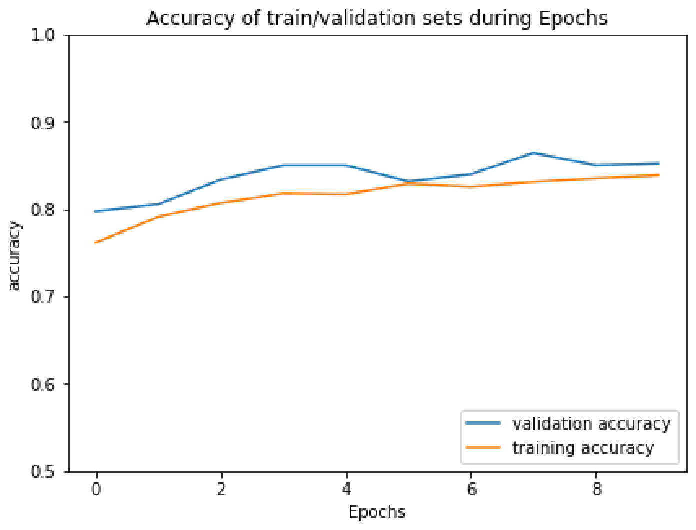
$83.14\% \pm 0.98$

The average Loss for all folds is:

0.418

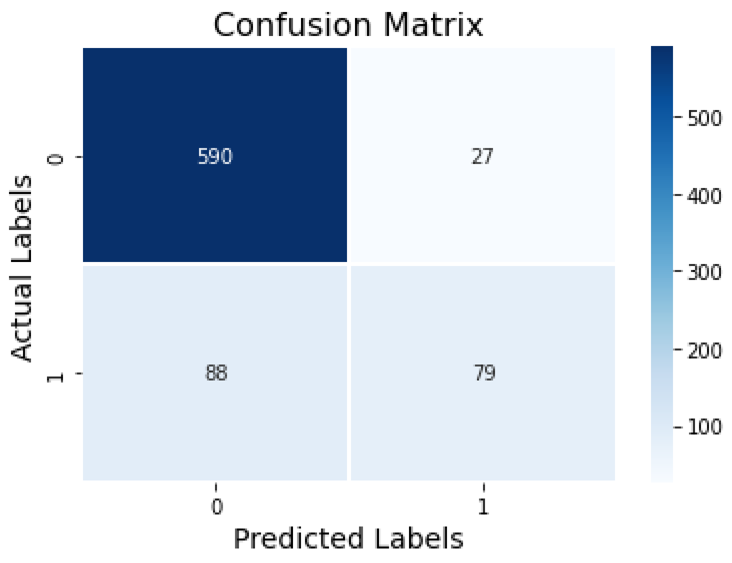
Now that we know our model is good, we will train the network on all the training data. The results are in the below figure.

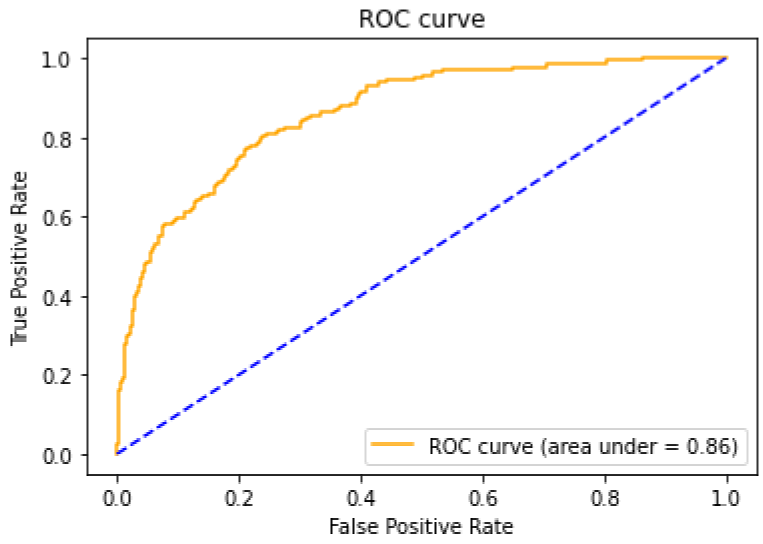




Loss and the accuracy of the above model on the test set are 0.40, and 85.31%, respectively. We also calculated the below metrics for the unseen data.

	Precision	Recall	F1-score	Support
On label 0	0.87	0.96	0.91	617
On label 1	0.75	0.47	0.58	167

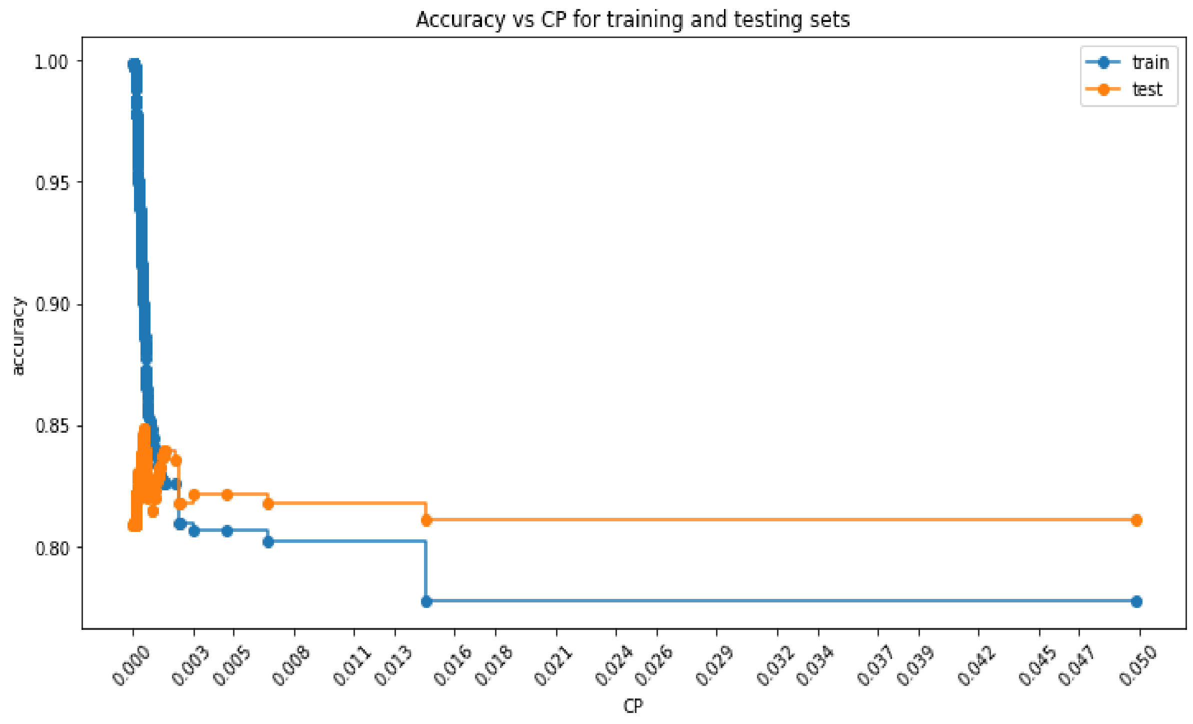




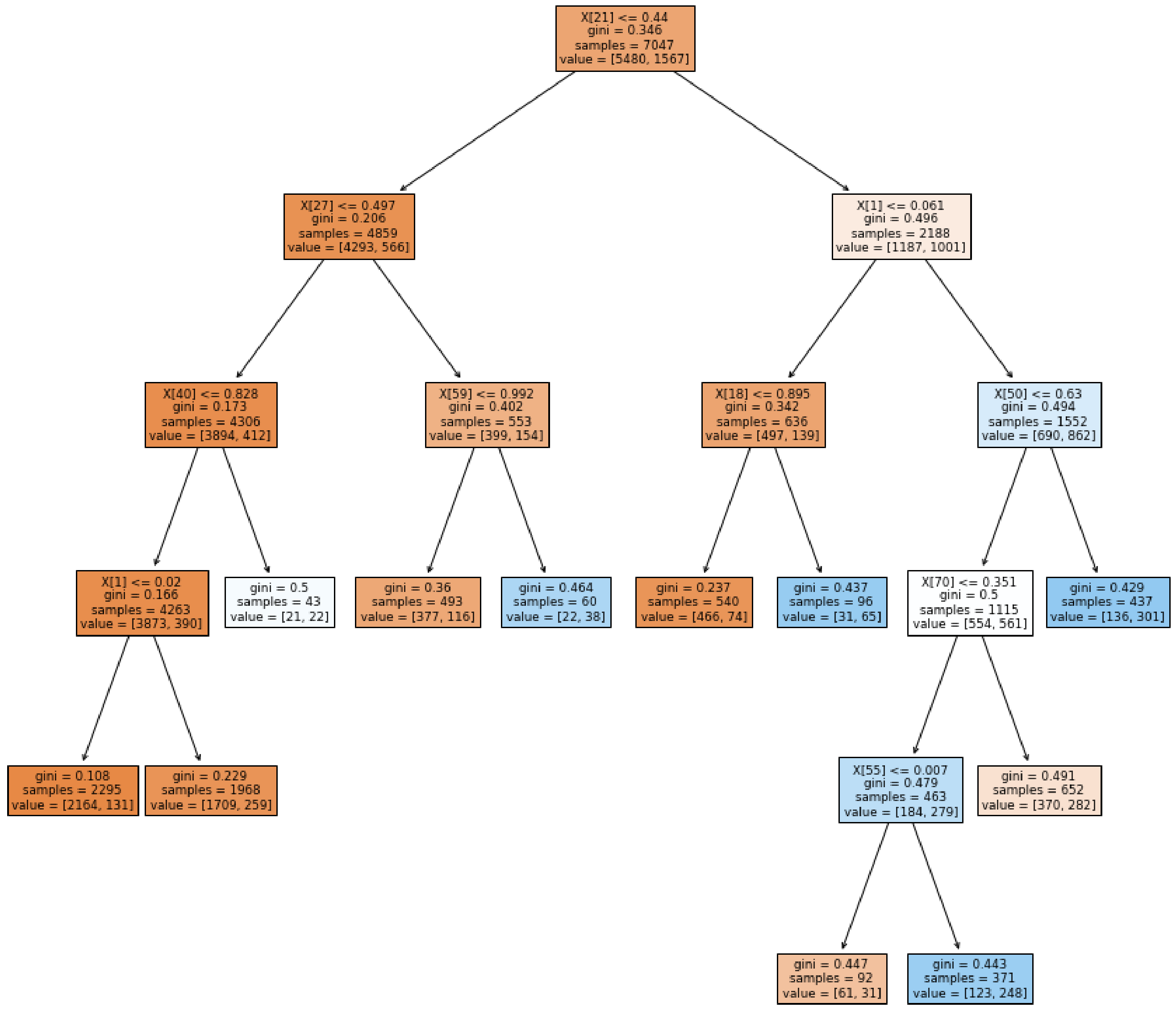
3.5 Decision Tree

Here, we are going to address the toxicity by fitting a decision tree model to our data.
We allocated 10% of the data to the test set, and with the rest, we are going to build a prediction model.
We chose the Gini impurity measure as the optimum split criteria.
One approach to prevent overfitting is using Complexity Parameter (CP) to prune the excessive leaves.

It is time to choose the most suitable value for CP. We will pick the one that has the best accuracy on the test set.

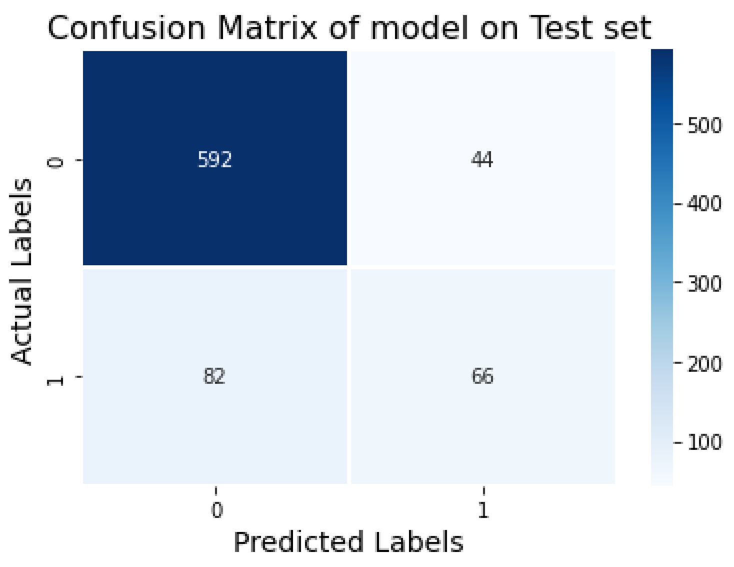


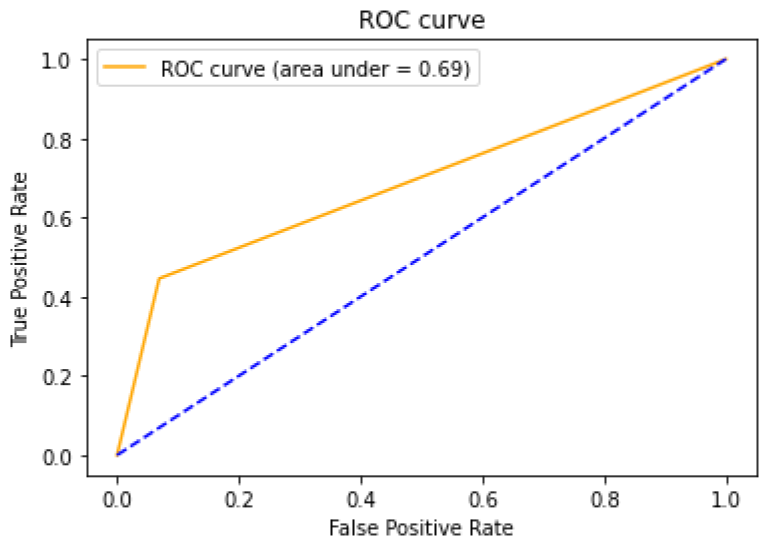
Above graph demonstrates the accuracy of different decision trees with different CP values.
According to the previous figure, we will fit a decision tree with Gini Criteria, and $\alpha_{CP} = 0.00167$, we have a tree as the following:


X[21] <= 0.44 gini = 0.346 samples = 7047 value = [5480, 1567]
X[27] <= 0.497 gini = 0.206 samples = 4859 value = [4293, 566]
X[1] <= 0.061 gini = 0.496 samples = 2188 value = [1187, 1001]
X[40] <= 0.828 gini = 0.173 samples = 4306 value = [3894, 412]
X[59] <= 0.992 gini = 0.402 samples = 553 value = [399, 154]
X[18] <= 0.895 gini = 0.342 samples = 636 value = [497, 139]
X[50] <= 0.63 gini = 0.494 samples = 1552 value = [690, 862]
X[1] <= 0.02 gini = 0.166 samples = 4263 value = [3873, 390]
gini = 0.5 samples = 43 value = [21, 22]
gini = 0.36 samples = 493 value = [377, 116]
gini = 0.464 samples = 60 value = [22, 38]
gini = 0.237 samples = 540 value = [466, 74]
gini = 0.437 samples = 96 value = [31, 65]
X[70] <= 0.351 gini = 0.5 samples = 1115 value = [554, 561]
gini = 0.429 samples = 437 value = [136, 301]
gini = 0.108 samples = 2295 value = [2164, 131]
gini = 0.229 samples = 1968 value = [1709, 259]
X[55] <= 0.007 gini = 0.479 samples = 463 value = [184, 279]
gini = 0.491 samples = 652 value = [370, 282]
gini = 0.447 samples = 92 value = [61, 31]
gini = 0.443 samples = 371 value = [123, 248]

This tree has an accuracy of 82.92% on the test set. We also calculated the below metrics for the test data.

	Precision	Recall	F1-score	Support
On label 0	0.88	0.93	0.90	636
On label 1	0.60	0.45	0.51	148

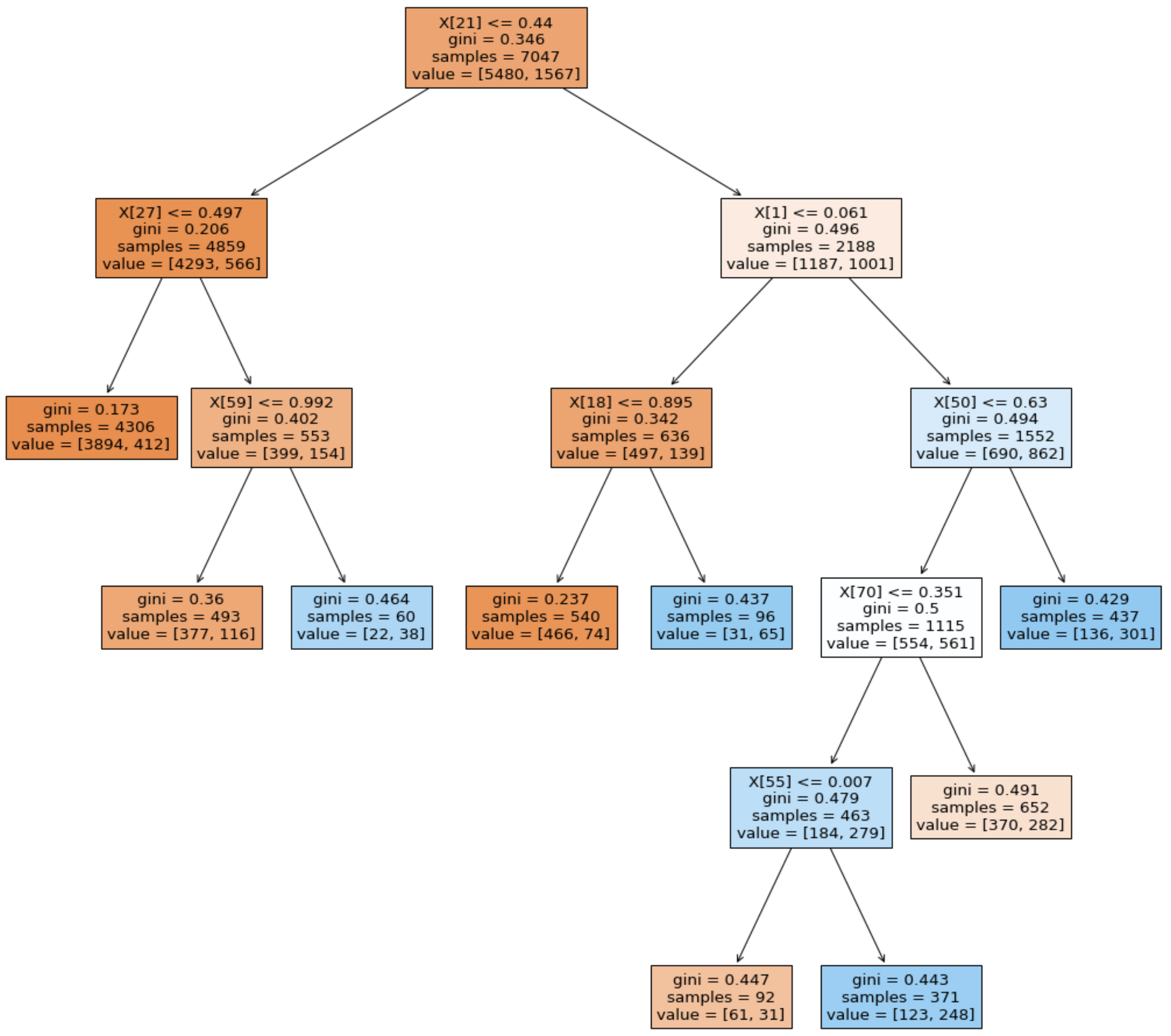




Another way to prevent overfitting is to test different tree configurations with a wide variety of depths and leaf nodes, and choose the setting with the most accurate result.
While we care about the most accurate results, we aim to choose the less complicated settings.
After testing trees with depth 3 to 19, and number of leaf nodes from 3 to 19, we got the below results.

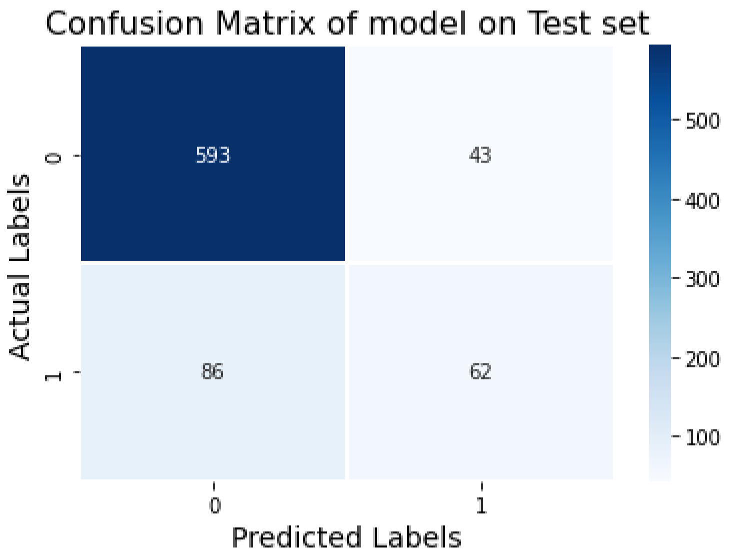
	Accuracy
Number of leaf nodes = 9	81.11%
Depth = 7	81.63%

We decided to go with max_leaf_nodes parameter, because we prefer a simpler model rather than a complicated one with a slight boost.

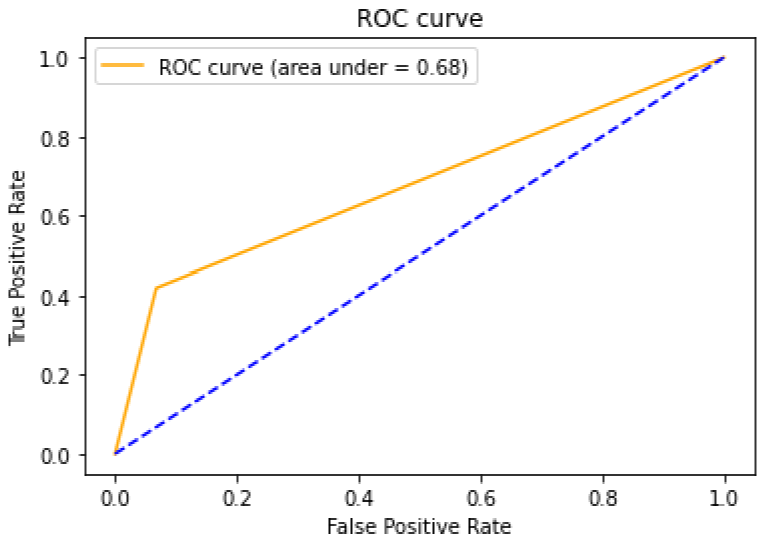


This tree has an accuracy of 83.54% on the test set. We also calculated the below metrics for the test data.

	Precision	Recall	F1-score	Support
On label 0	0.87	0.93	0.90	636
On label 1	0.59	0.42	0.49	148



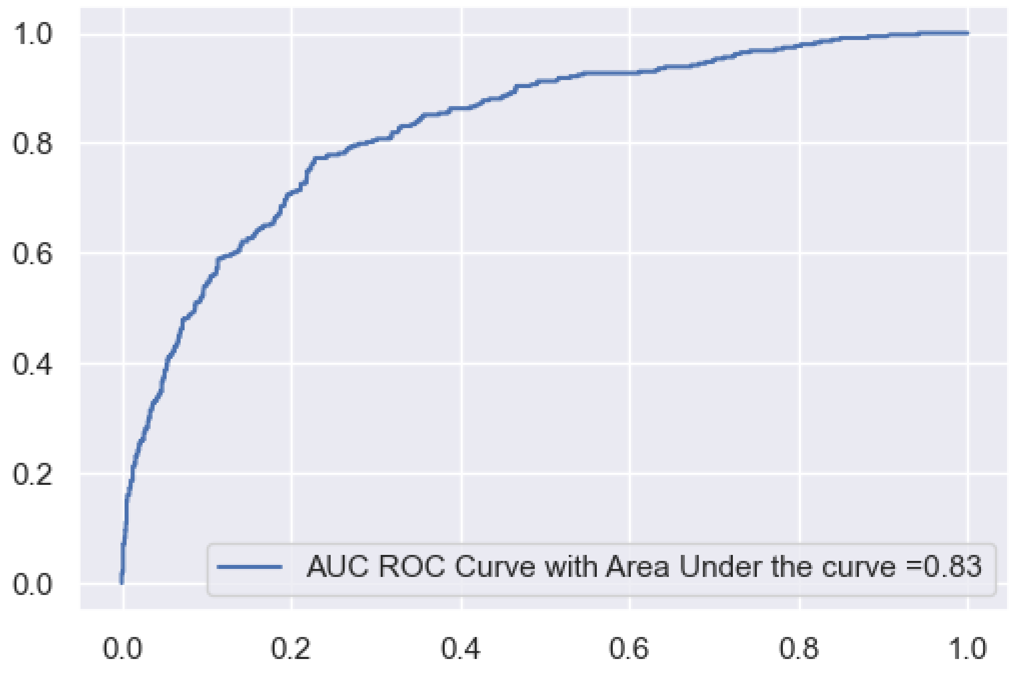
Confusion Matrix of model on Test set
Actual Labels
Predicted Labels
0
1
593
43
86
62
500
400
300
200
100



ROC curve
ROC curve (area under = 0.68)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

3.6 Adaboost

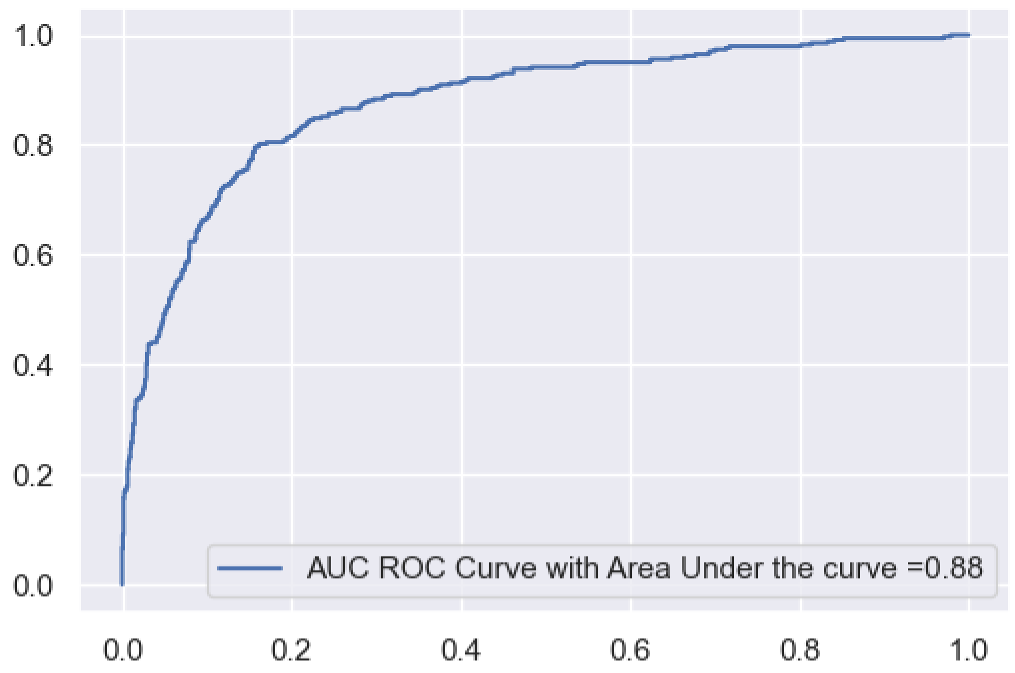
As we have seen in the tox21, the Adaboost had overfitting on training data but in this model without
Tuning anything the model works pretty good on both training and testing data.



For testing we get to 83 and in training we get to 86, and our model is precisely generalized.

3.7 Random Forest

Exactly like the toxcast data, we start tuning our model we use grid search to get to the best model.



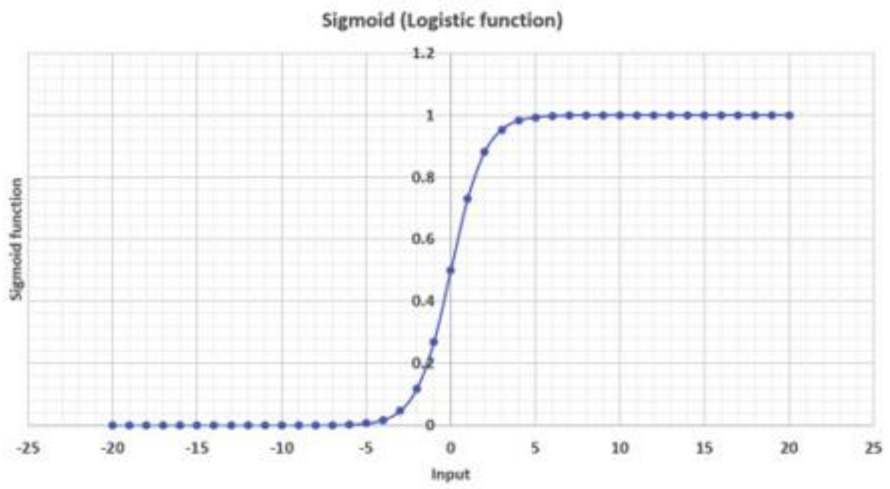
With training score 99 we can see this model over-fitted as well
So, we should change the base trees in random Forest to get over this issue.
We chose 1 for the depth of our tree we get to 84, 87.
For our training and testing data. It is true that we get to less score but it is finer to choose this because the training score is near to tests.

3.8 Logistic Regression

Logistic regression is a powerful supervised ML algorithm which we use for predicting the binary labels of Tox-Cast dataset.

Logistic function:

$$Sigmoid\ Logistic\ function = \frac{1}{1+e^{-x}}$$

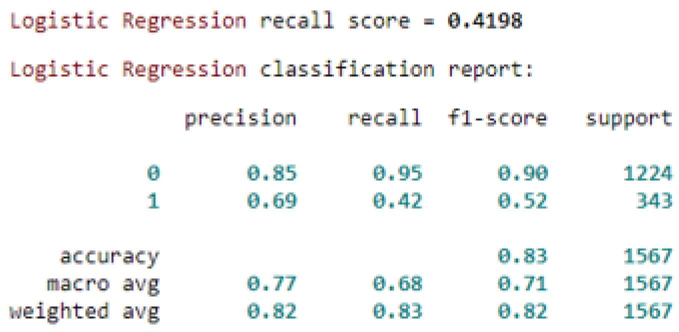


Logistic regression loss function cross entropy:

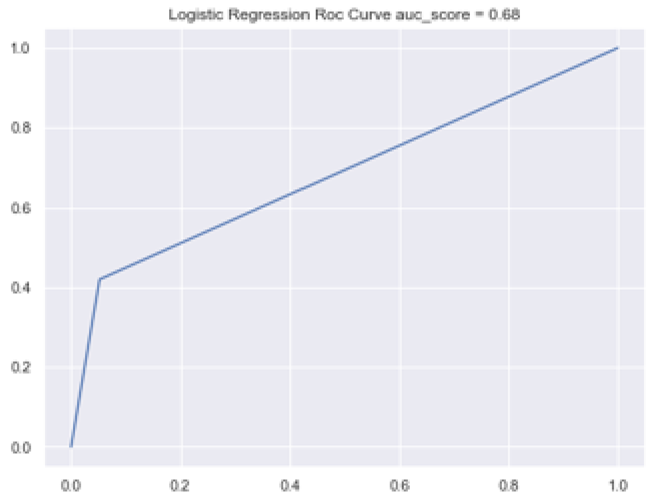
$$-\frac{1}{N}\sum_{i=1}^{N} y_i\, log\big(P(y_i)\big) + (1 - y_i)\, log\big(1 - P(y_i)\big)$$

There are 135 features in tox-cast dataset, and we are going to work with a single label.

First, we will apply logistic regression model, for train-test split 80 percent for train and 20 percent for test.

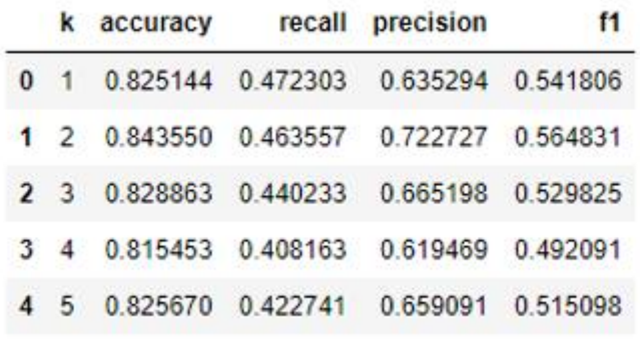


```
Logistic Regression recall score = 0.4198

Logistic Regression classification report:
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      1224
           1       0.69      0.42      0.52       343

    accuracy                           0.83      1567
   macro avg       0.77      0.68      0.71      1567
weighted avg       0.82      0.83      0.82      1567
```

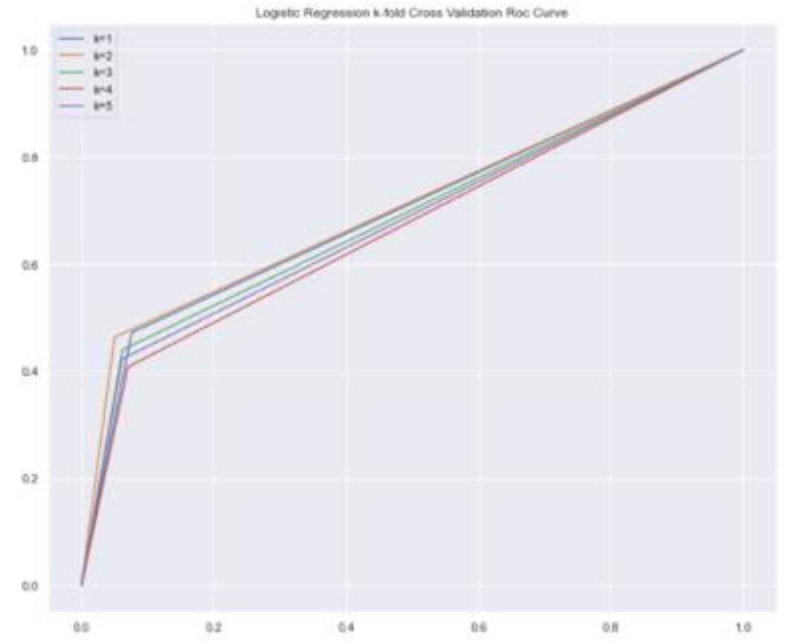


Another way of approach is to use k-fold cross-validation (splits = 5)

The results are summarized below.



	k	accuracy	recall	precision	f1
0	1	0.825144	0.472303	0.635294	0.541806
1	2	0.843550	0.463557	0.722727	0.564831
2	3	0.828863	0.440233	0.665198	0.529825
3	4	0.815453	0.408163	0.619469	0.492091
4	5	0.825670	0.422741	0.659091	0.515098



3.9 Perceptron

The Perceptron algorithms is a two-class (binary) classification machine learning algorithms.

It is a very simple type of neural network model (single neuron) that takes a row of data as input and predicts a class label.

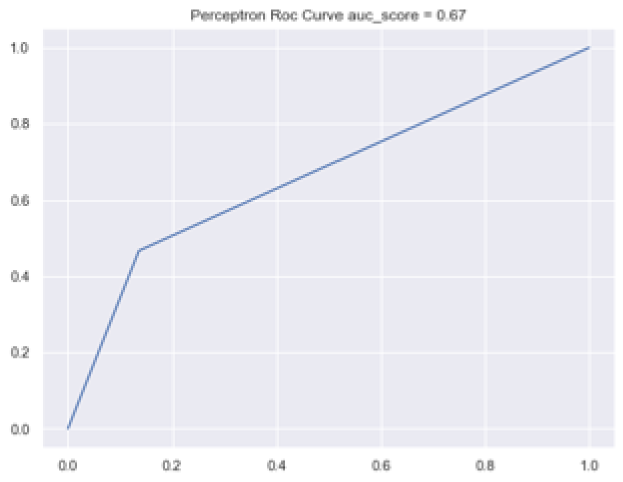
*Activation = (Weights*Inputs) + Bias*

if Activation > 0 : Predict 1

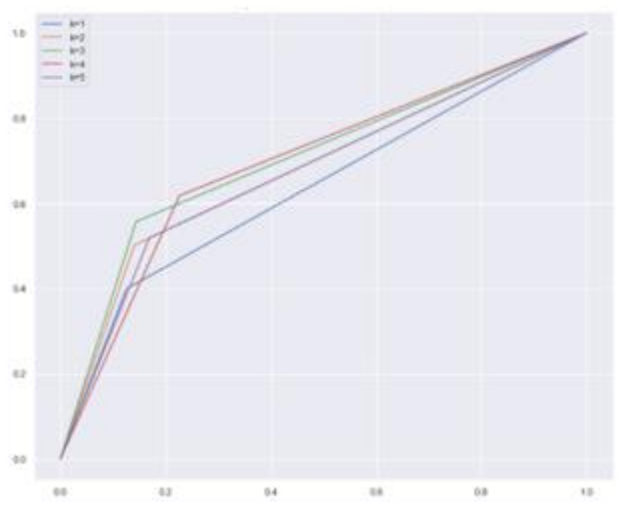
if Activation < 0 Predict 0

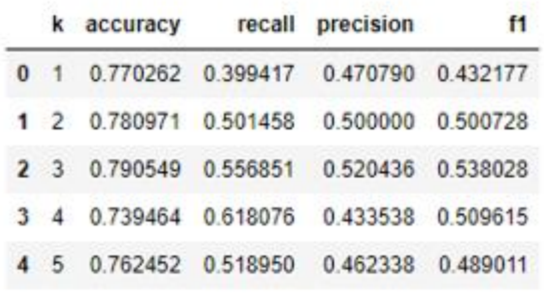
Note: given that the inputs are multiplied by model coefficients, like linear and logistic regression, it is a good idea to standardize data prior to using the model.

```
Perceptron recall score = 0.4665

Perceptron classification report:
              precision    recall  f1-score   support

           0       0.85      0.86      0.86      1224
           1       0.49      0.47      0.48       343

    accuracy                           0.78      1567
   macro avg       0.67      0.67      0.67      1567
weighted avg       0.77      0.78      0.77      1567
```



Perceptron with cross validation (split = 5):



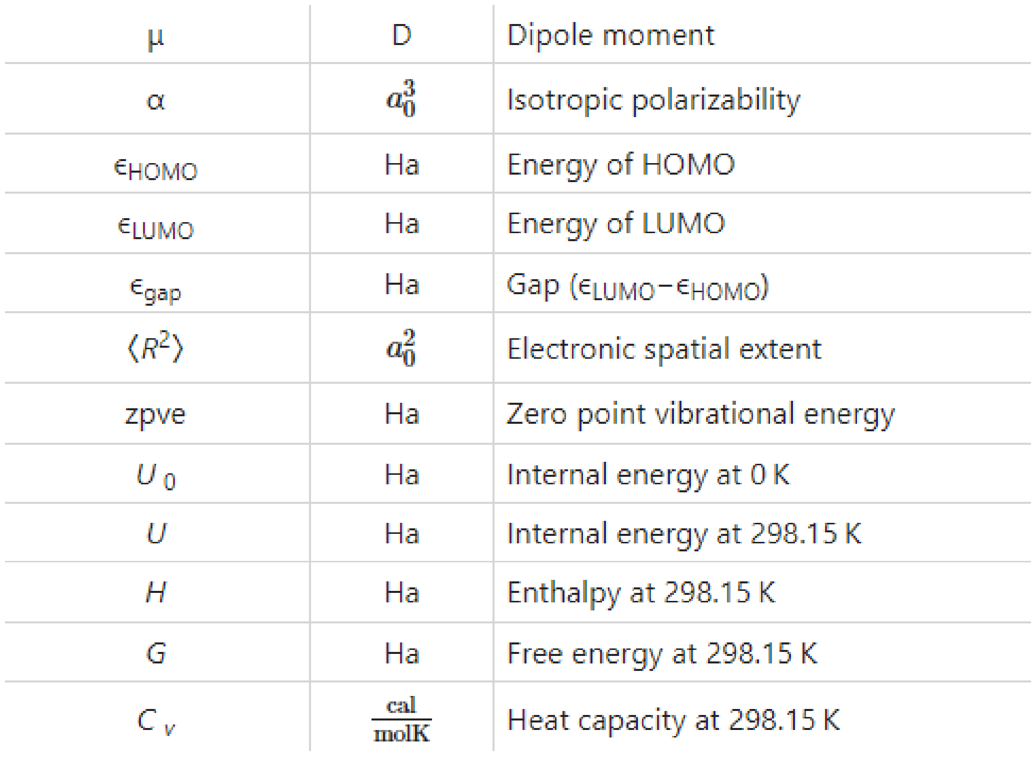


	k	accuracy	recall	precision	f1
0	1	0.770262	0.399417	0.470790	0.432177
1	2	0.780971	0.501458	0.500000	0.500728
2	3	0.790549	0.556851	0.520436	0.538028
3	4	0.739464	0.618076	0.433538	0.509615
4	5	0.762452	0.518950	0.462338	0.489011

4. QM9 Dataset

Computational Design of new drugs and materials requires exploration of chemical compound space, which grows combinatorially with molecular size. The QM9 dataset gathers geometric, energetic, and thermodynamic properties of 134k stable small organic molecules along with their SMILES representation [2].

This dataset consists of 12 regression tasks (molecular properties) which are presented in the following table.



μ	D	Dipole moment
α	a_0^3	Isotropic polarizability
ϵ_{HOMO}	Ha	Energy of HOMO
ϵ_{LUMO}	Ha	Energy of LUMO
ϵ_{gap}	Ha	Gap ($\epsilon_{LUMO} - \epsilon_{HOMO}$)
$\langle R^2 \rangle$	a_0^2	Electronic spatial extent
zpve	Ha	Zero point vibrational energy
U_0	Ha	Internal energy at 0 K
U	Ha	Internal energy at 298.15 K
H	Ha	Enthalpy at 298.15 K
G	Ha	Free energy at 298.15 K
C_v	$\frac{\text{cal}}{\text{molK}}$	Heat capacity at 298.15 K

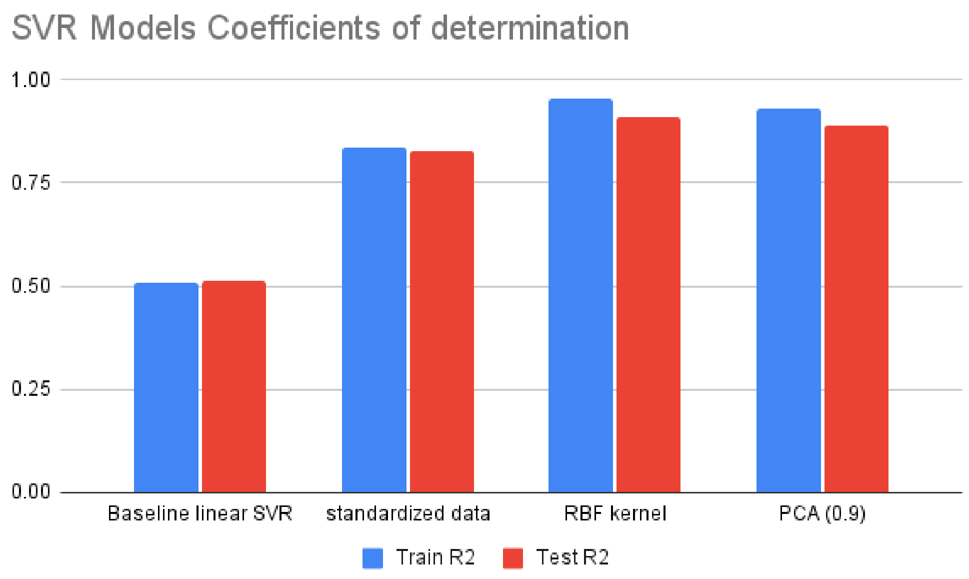
4.1 Preprocessing

To extract features from the SMILES strings, descriptastorus [3] was used to compute 200 numeric features and then normalized to fit a cumulative distribution function. This type of normalization comes along with nice qualities, for example, a feature with a value of 0.3 sits roughly above 30 percent of the data.
Next, we dropped features with a constant value for the entire dataset and then searched for missing values, which were not present in the case of the QM9 dataset.

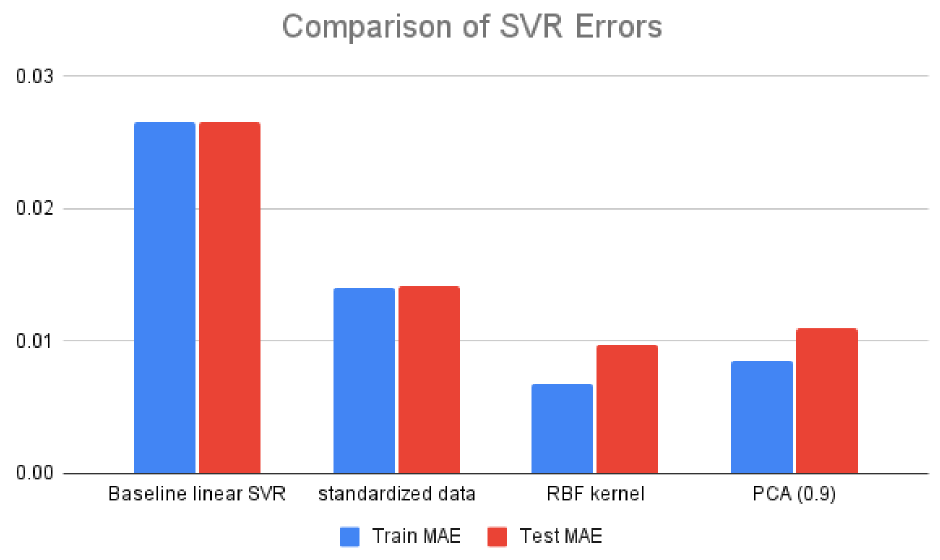
4.2 Support Vector Regression

Support vector machines are typically used for small datasets, in fact, they might struggle with very large datasets because SVMs typically hold the kernel matrix in memory which grows quadratically with the size of the training set. The QM9 datasets with over 130k data points face the issue, as a regular 80/20 split for train/test sets does not converge. To combat this issue, a smaller training set is employed (10% of data) to train an SVR. We then make progressive improvements to this method by standardizing features and labels, introducing an RBF kernel function, and using PCA to reduce the dimensionality of our data, speeding up evaluation while losing only a little model score.

We use the energy gap property in the QM9 dataset for this demonstration, reporting train and test MAE and Coefficient of determination for different models.

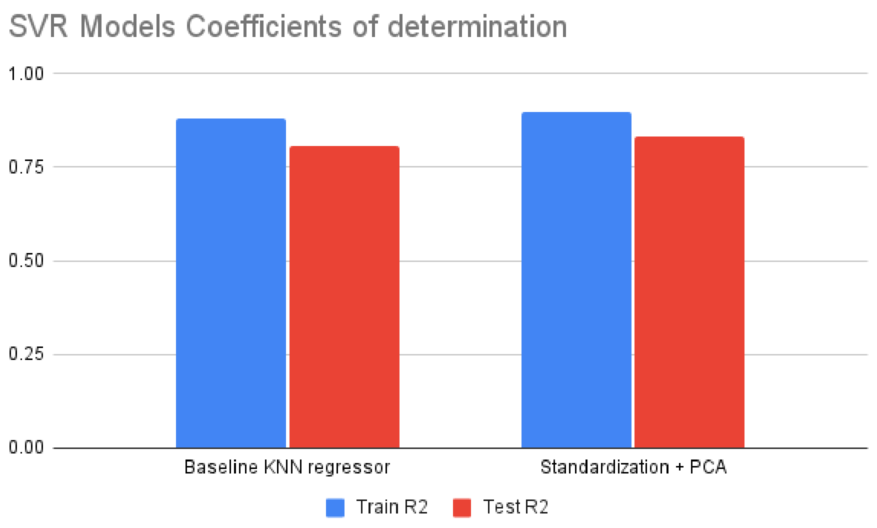


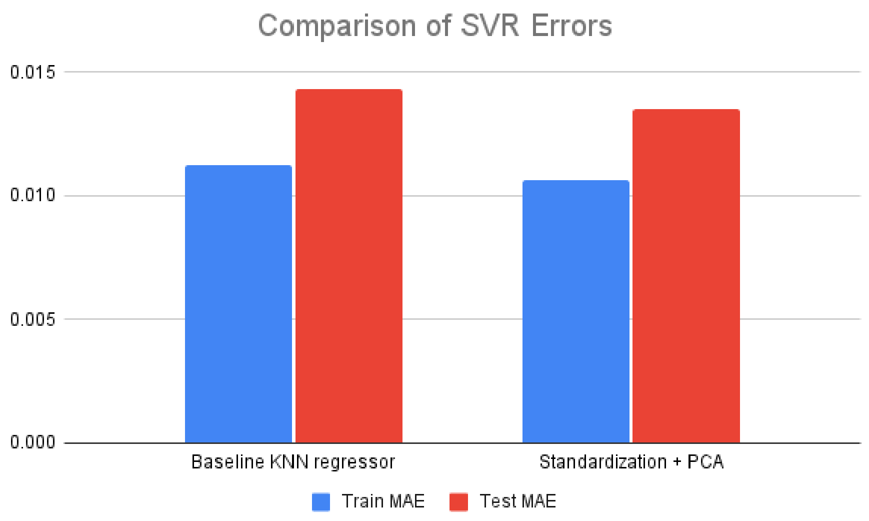
Even though we use only 10% of data for training, our model generalizes well to the other 90%. The first noticeable improvement is achieved by standardizing features and labels. Difference in the range of labels compared with features can be the cause of this outcome. A radial basis kernel enables the SVR to learn non-linear predictions leading to further improvement. The large test set meant that testing took a long time, using dimensionality reduction with PCA not only sped up testing but also resulted in very minimal performance loss.



4.3 K-Nearest Neighbors

Nearest Neighbor methods can also be used for regression tasks, we compare the results of two models, a baseline and a KNN regressor with standardization of features and labels, projected using a PCA to contain 90% of the original variance. The data split is 10/90 to get comparable results with support vector regression results on the "gap" task of QM9.





The performance of the two models is similar, however, the PCA model is considerably faster in prediction and the performance is comparable with the tuned SVR model.

4.4 Neural Network

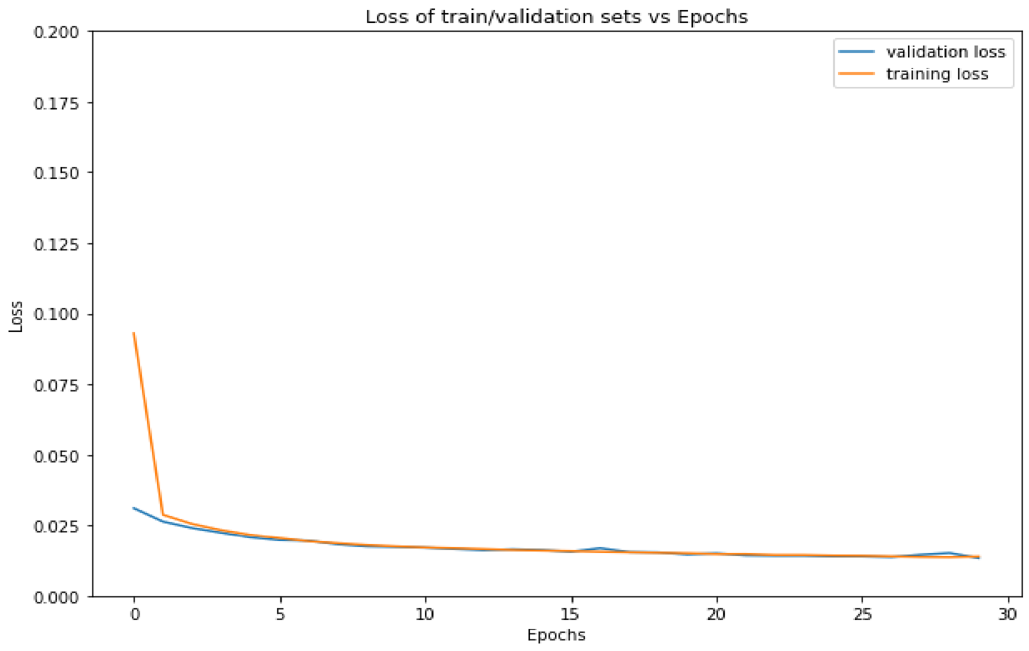
In this section, we are going to implement a Multi-Layer Neural Networks algorithm to perform a regression algorithm on QM9. First, we scale the labels to assist our optimization algorithm to converge faster.

We divide each column to its maximum value.

The Neural Network architecture which we used:

	Number of Neurons	**Activation Function**
Layer 1	**64**	**Relu**
Layer 2	**32**	**Relu**
Layer 3	**16**	**Relu**
Layer 4	**12**	**linear**

You can see the performance of the above model in the below figure.



Mean absolute error of the above model on the test set is 0.0133.

You can see the error of the model on each individual task in the below table.

Task	Mean Absolute Error
mu	0.026
alpha	0.006
homo	0.016
lumo	0.040
gap	0.016
r2	0.018
zpve	0.009
cv	0.011
u0	0.003
u298	0.003
h298	0.003
g298	0.003

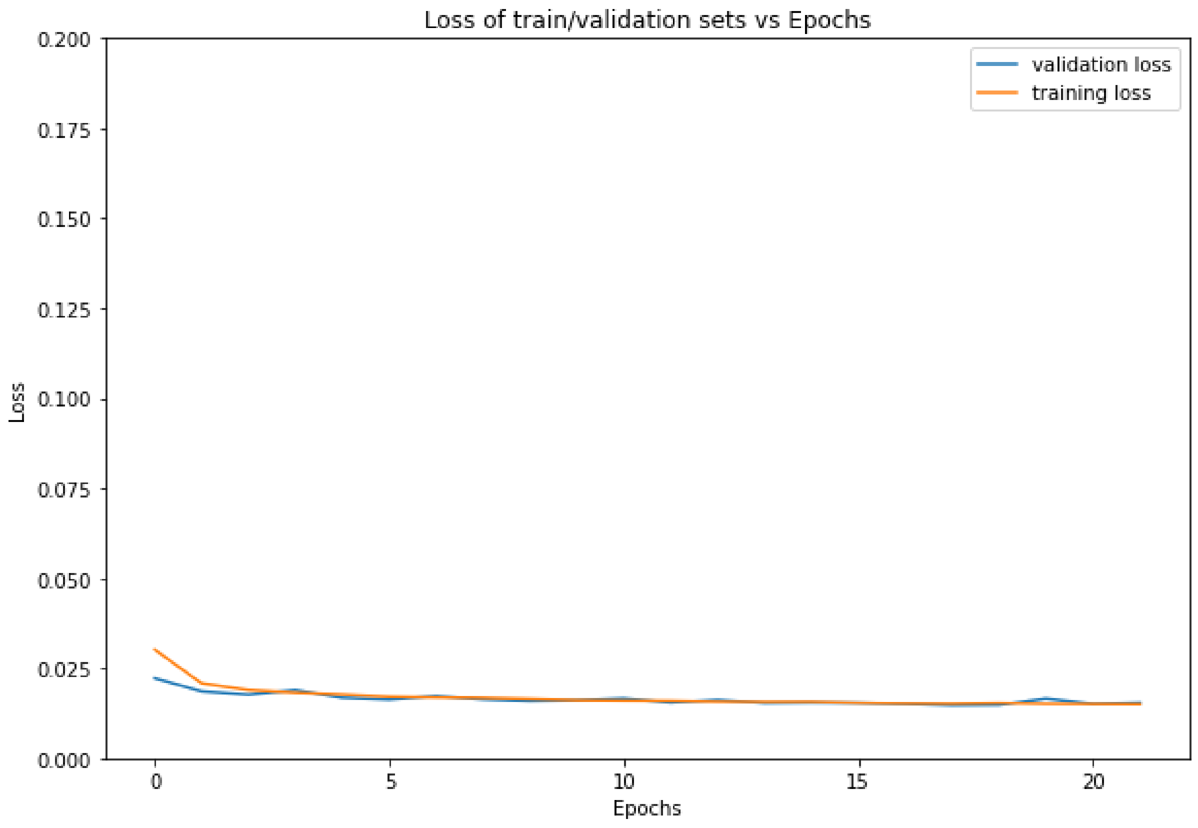
The R-square metric is as follows:

	R-Square
On Train	90.57%
On Test	90.57%

Now we will use a single class Neural Network to perform a regression task on the "gap" task.
The Neural Network architecture which we used:

	Number of Neurons	**Activation Function**
Layer 1	**64**	**Relu**
Layer 2	**32**	**Relu**
Layer 3	**16**	**Relu**
Layer 4	**1**	**linear**

You can see the performance of the above model in the below figure.



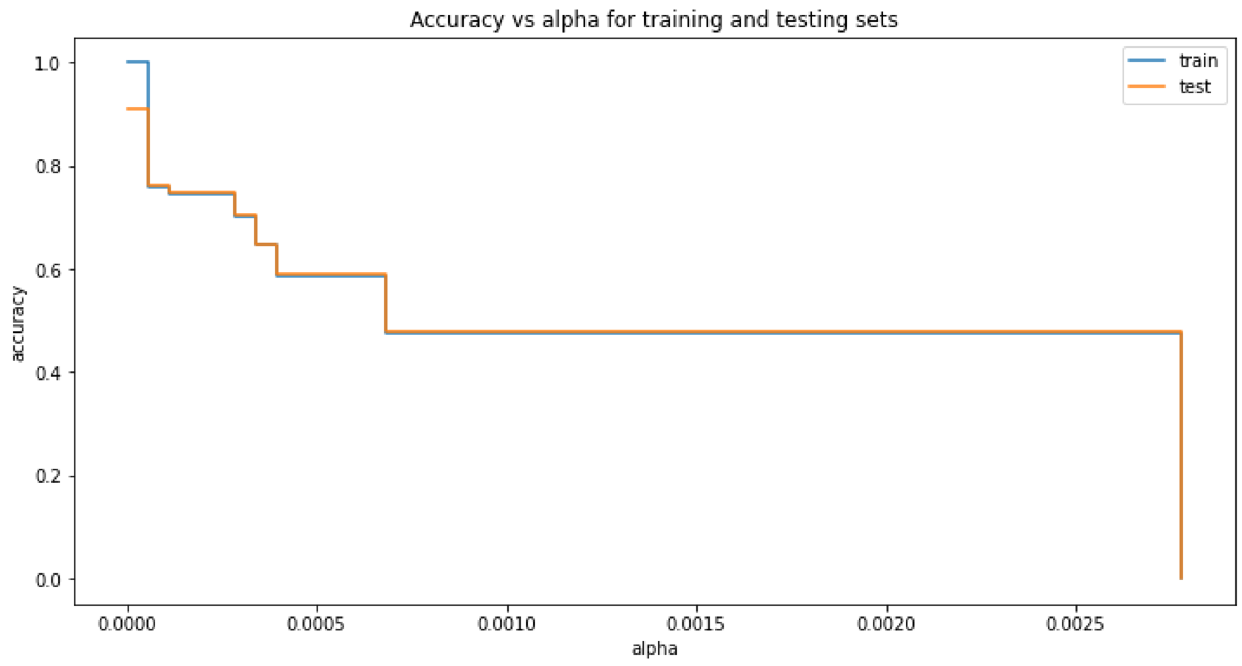
Mean absolute error of the above model on the test set is 0.015.

The R-square metric is as follows:

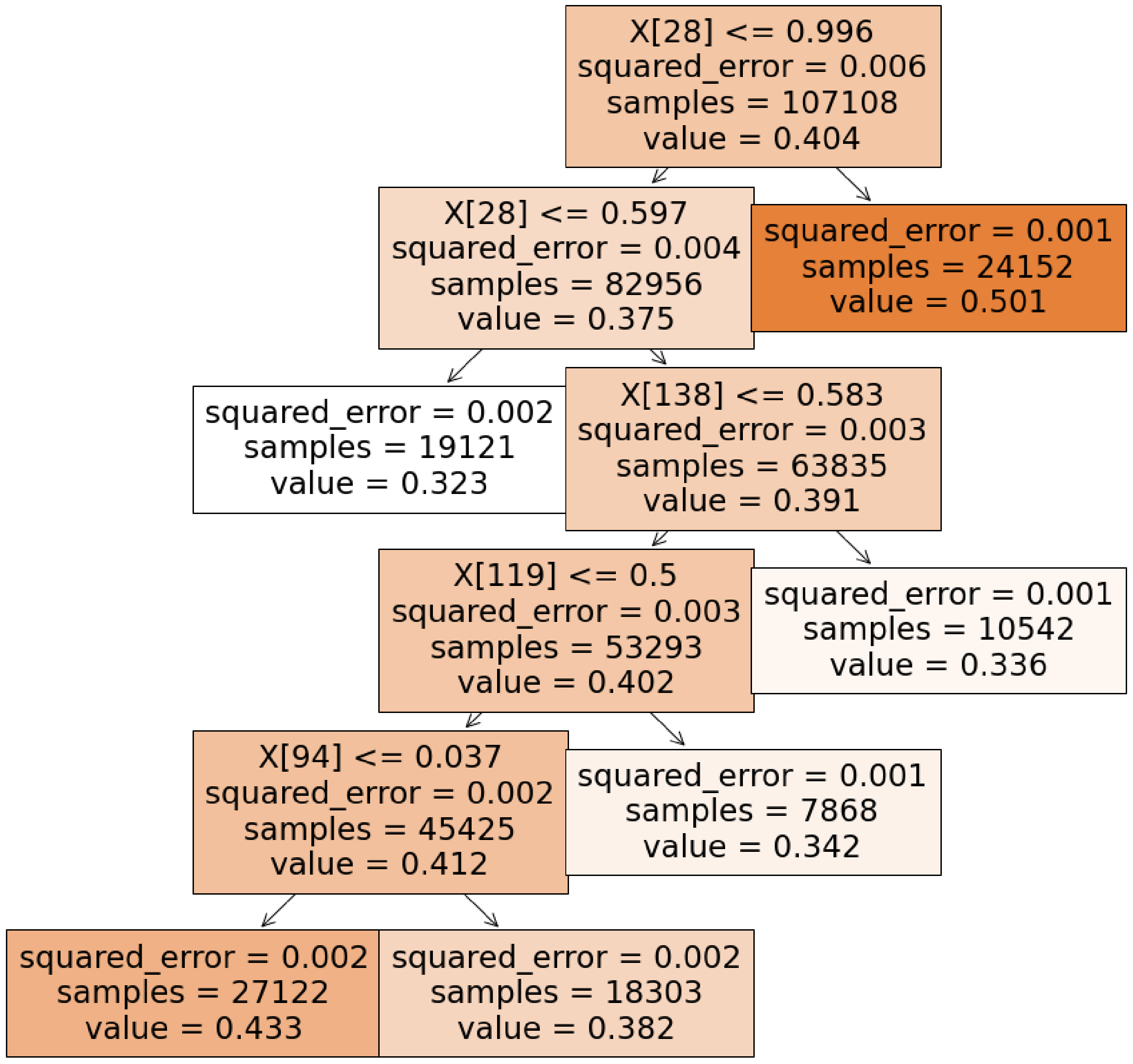
	R-Square
On Train	91.91%
On Test	91.70%

4.5 Decision Tree

Here, we are going to predict the “gap” property of Quantum Mechanic features of molecules by fitting a decision tree model to our data. According to the below graph, we chose the most suitable CP.



Below figure is a tree with $\alpha_{cp} = 0.00025$



Mean Absolute Error of the above graph on test set is 0.0288.

The R-square metric is as follows:

	R-Square
On Train	74.57%
On Test	74.99%

4.6 Linear Regression

We are going to predict the gap label from QM9 data using regression modeling by the features available to us.

After some pre-processing and data cleansing, we gathered some descriptive statistics from both of available features and the label.

It appears the label is normally distributed (which is not essential), but there seems to be some outliers at the higher end of the label causing a little skewness to the right. Therefore, we eliminated a very small part of the data.

Since we are using machine learning, there are some differences with traditional statistical methods. Because of standardizing our label variable, removing features after modeling does not seem to be so important (the coefficients [weights] tend to be very small and won't affect our model significantly).

But multicollinearity is still a big issue, if it ever happens. To prevent this, we use a traditional metric called Variance Inflation Factor.

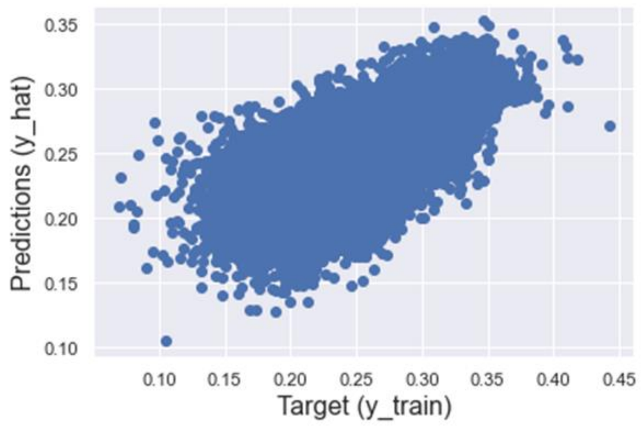
We calculated the VIF for every feature available. There is debate how large VIF is tolerable before we eliminate a feature. But since we have a tremendous number of features (163), we can easily eliminate any feature with VIF>5.

After doing this, 49 features remain, which are not linearly correlated.

We use 20 percent of our data for training (data is so big it might be enough).

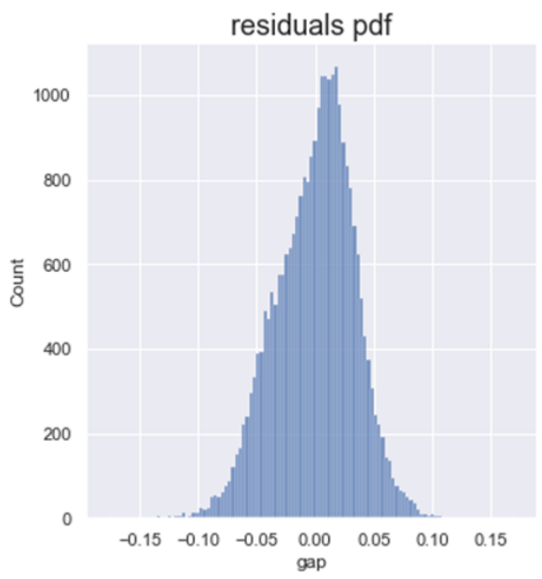
Now it's a good time to apply the regression model using sklearn.

Before looking into any statistics, we visualize our result by using a scatter plot with X axis being Target (output) and Y axis being predictions.



After a perfect modeling, all dots should be around a 45 degree line (Y=X), which rarely happens.

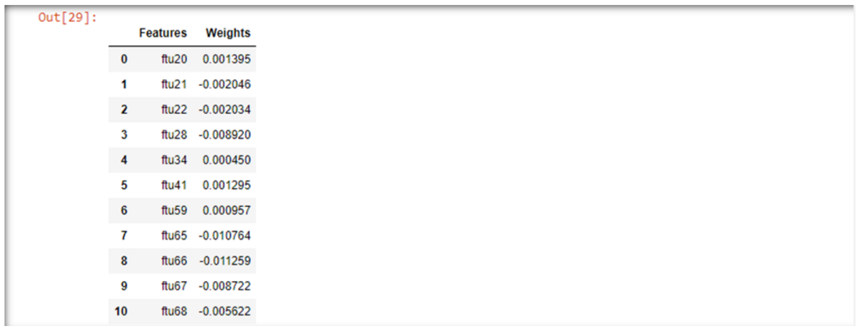
Another way to evaluate goodness of fit is to examine the probability density function of residuals by plotting a histogram. Since we are using OLS method for regression, as expected, the distribution of residuals seems to be normal with mean of zero (the peak of the histogram does not represent mean, because it is not a perfectly normal distribution.



Now let’s look at some statistics for the results. R-squared is reported to be 0.50, which means our model predicts 50 percent of variability, which is not perfect and our adjusted R-squared is almost the same as our R-squared which is perfect. This means there is almost no colinearity in our model.

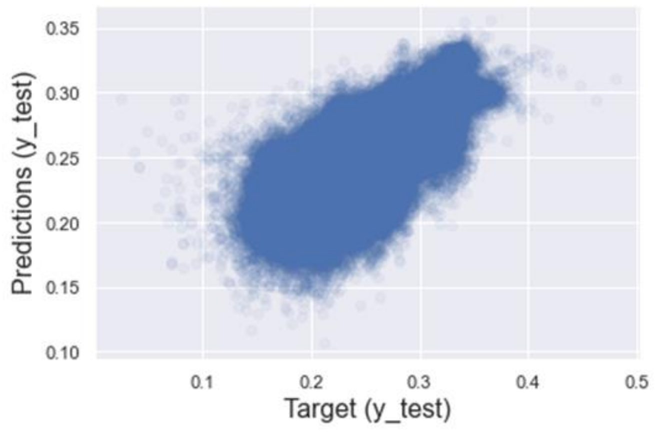
Since we have no prior knowledge of our label and features, we can’t say if 50% is good or not.

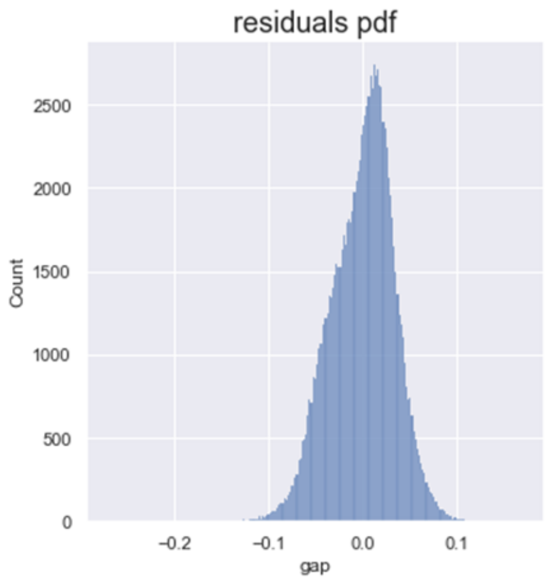
As shown in this table, some features can be removed but since we're using weights and not coefficients, the impact of small weights is so small that we can ignore them.


Out[29]:

	Features	Weights
0	ftu20	0.001395
1	ftu21	-0.002046
2	ftu22	-0.002034
3	ftu28	-0.008920
4	ftu34	0.000450
5	ftu41	0.001295
6	ftu59	0.000957
7	ftu65	-0.010764
8	ftu66	-0.011259
9	ftu67	-0.008722
10	ftu68	-0.005622

We use the visualizing methods used before on our test data and as expected, because we have a very large data sample, our model works similarly on both test part and train part of the data.





4.7 Adaboost

This approach of using AdaBoost for regression problems does not make much difference. The only thing is the base model is a linear

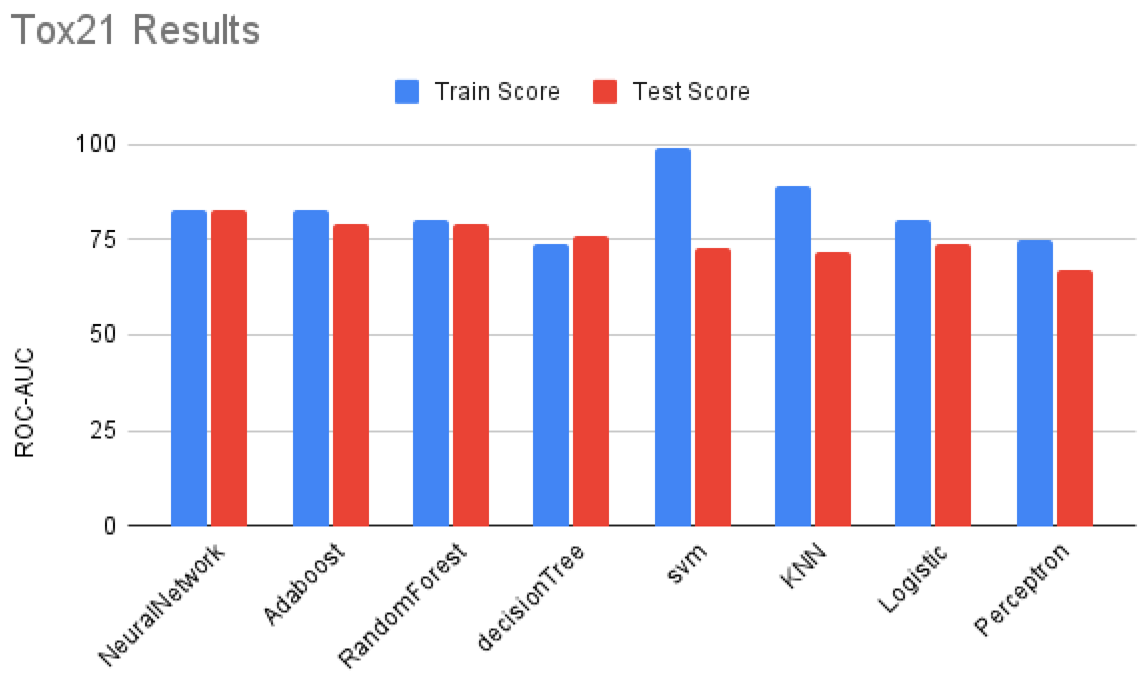
regression, and the bad points with chosen distance get higher weight for the next iteration.
We have done this on qm9 without tuning, because of the volume of our data and we get 95 for both training and testing r2 score.

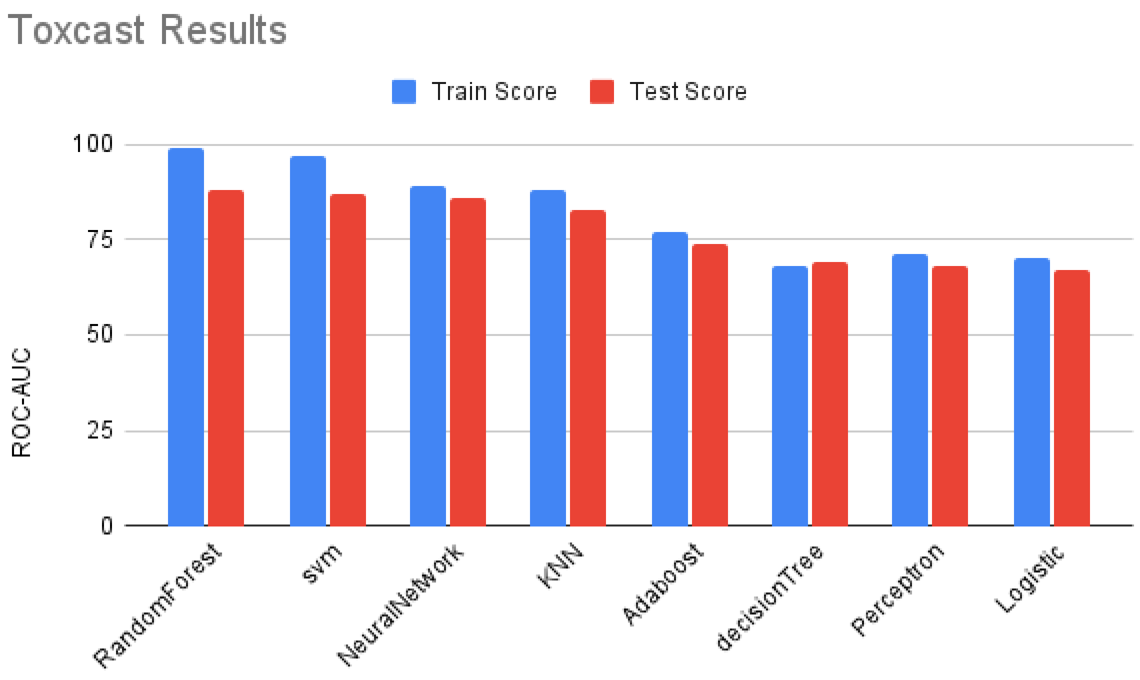
4.8 Random forest

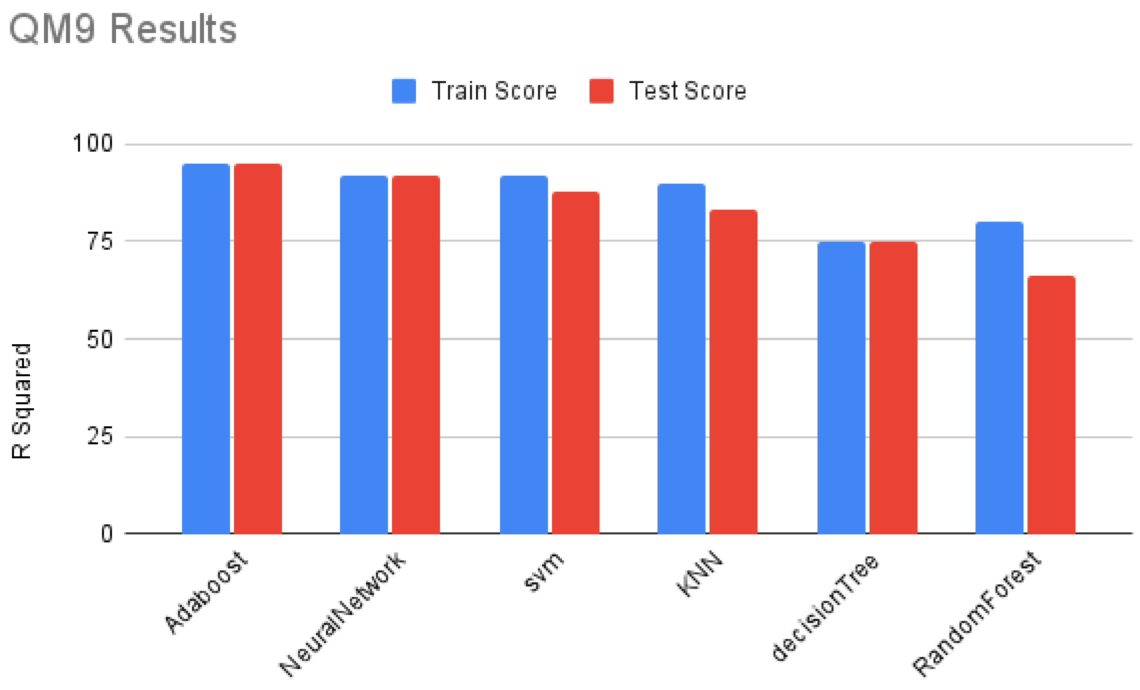
The data has large number of rows and we do not want our model take too much time so we start do this without tuning and amazingly we get to 99 for both training and testing data with 2 Score we can see how our model works perfect.

Conclusion:

Here, we will compare the final scores of each model for our datasets.




Toxcast Results
Train Score
Test Score
ROC-AUC
100
75
50
25
0
RandomForest
svm
NeuralNetwork
KNN
Adaboost
decisionTree
Perceptron
Logistic


QM9 Results
Train Score
Test Score
R Squared
100
75
50
25
0
Adaboost
NeuralNetwork
svm
KNN
decisionTree
RandomForest

References

[1] T.Unterthiner, A.Mayr, G.Klambauer, S.Hochreiter, 2016, “DeepTox: Toxicity Prediction using Deep Learning”, *Frontiers in Environmental Science* Vol.3

[2] Raghunathan Ramakrishnan, et al. Quantum chemistry structures and properties of 134-kilo molecules. *Scientific Data*, 1(1):140022, Aug 2014.

[3] Brian Kelley, Descriptastorus, https://github.com/bp-kelley/descriptastorus