BITS F464 - Machine Learning

Assignment Report: Yash Bhisikar (2021A7PS0483G)

Data Insights

The training dataset contains 400k data samples. It consists of 30 features which are “kinematic properties measured by the detectors and functions of these measured properties”. The target is a binary variable which indicates whether the data point corresponds to a Higgs Boson signal detection or not.

- No null values are found in the dataset, i.e. the data is complete. No missing data imputation is required.
- Histogram Plots -

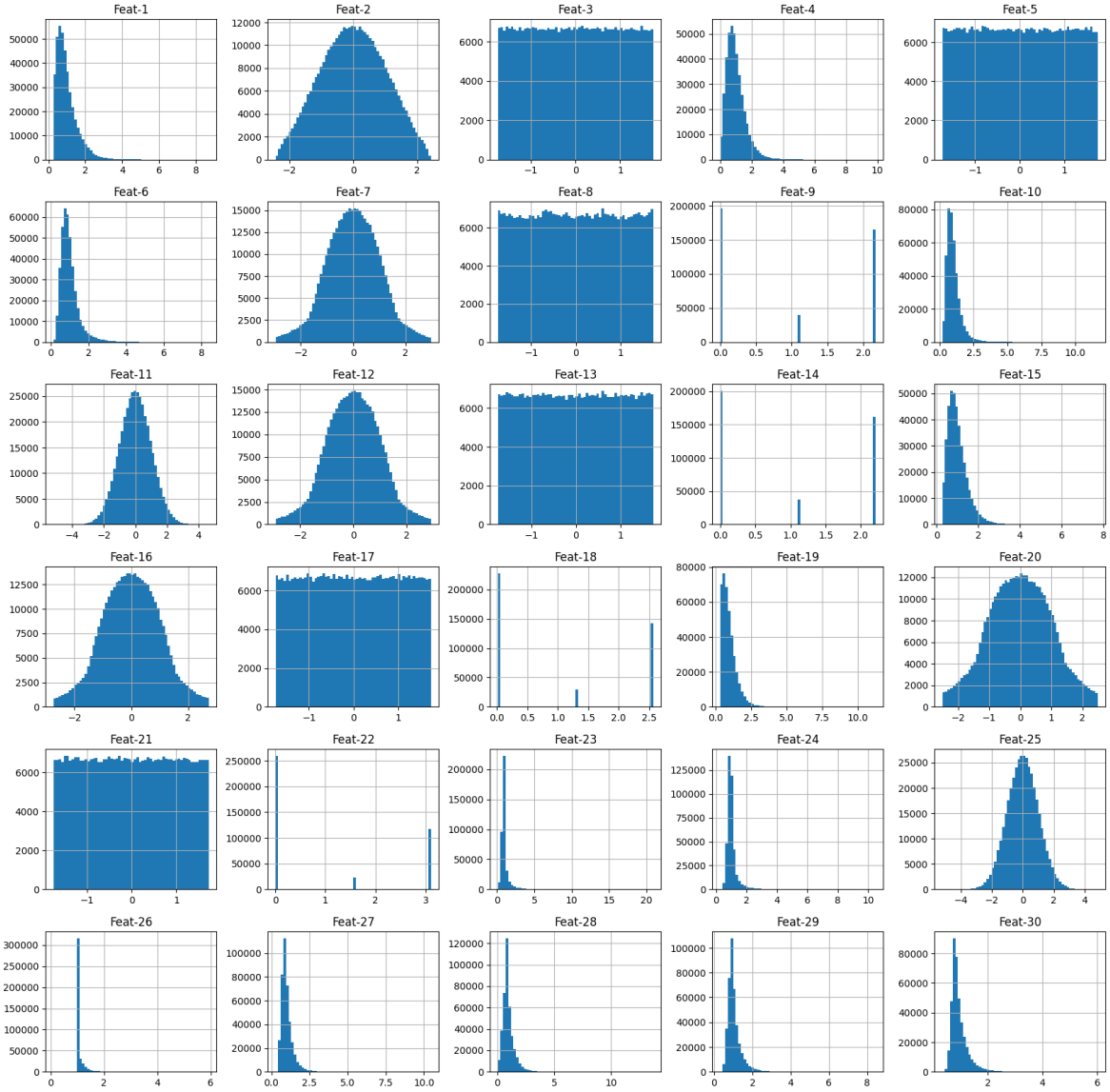

Figure 1: Histogram plots for the features. Note that the feature values had been divided into 60 bins. Adding more bins may make the graphs smoother in continuous random variable cases and noisier with sparsely distributed variables.

Observations:

Category 1: A lot of the features follow Gaussian distribution (Examples: 2, 13, 17). All of these are centred around 0

Category 2: Some of the features are very skewed and look very similar to the F-distribution (Examples: 1, 4, 10). All of them are skewed away from 0. Their skewness was found to be > 0.6 in all of the cases

Category 3: Some of the features are uniformly distributed. (Examples: 3, 5, 8)

Category 4: A few categorical features can also be found (Examples: 9, 14, 22). All of them happen to be tri-modal

- Box and Whisker Plots -

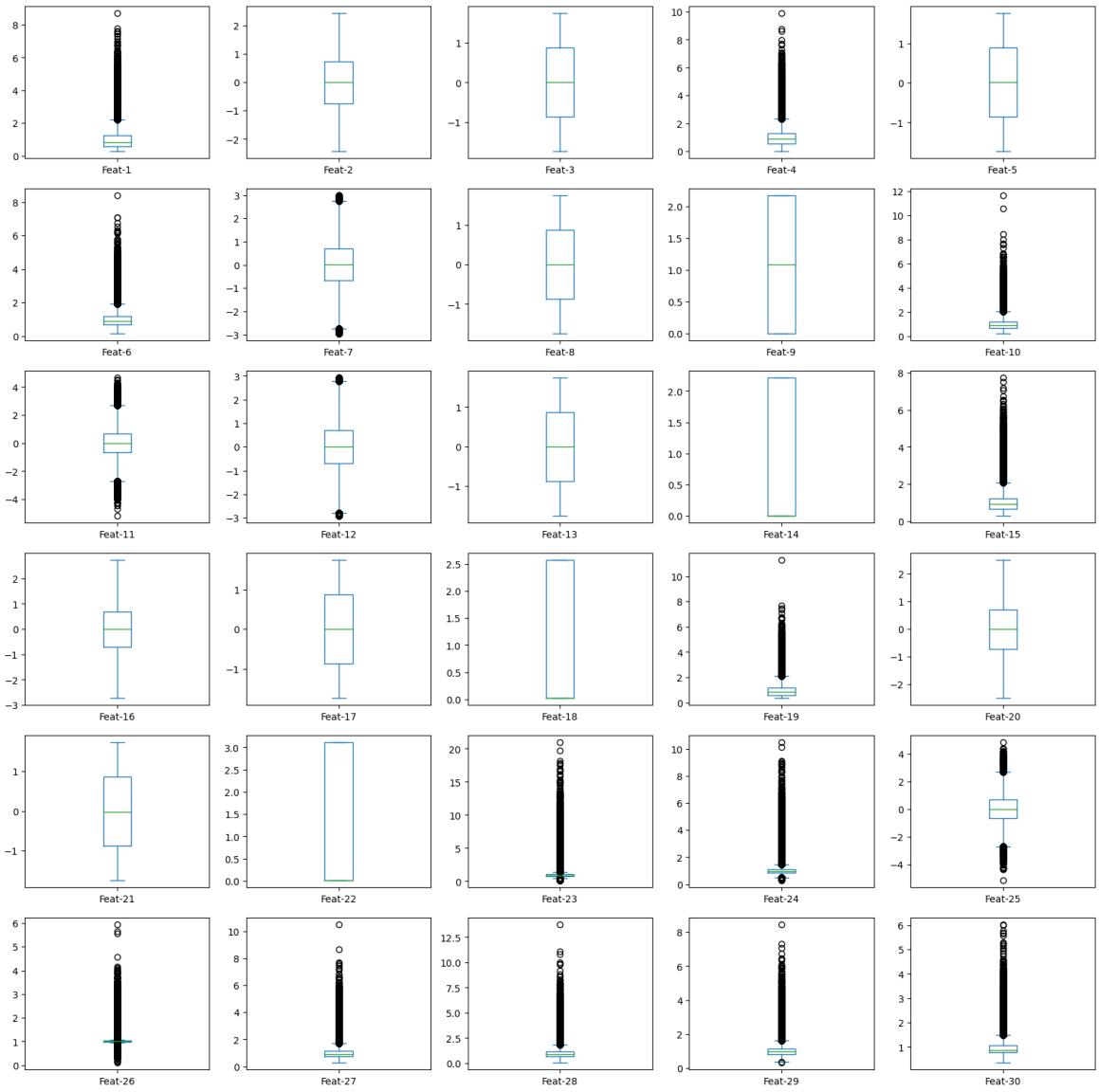

Figure 2: Box and Whisker Plots. Black circles correspond to outliers in the feature values. These values are 1.5 times greater (lesser) than the upper (lower) quartile.

Observations:

1. Notice that plots with outliers(black circles) predominantly belong to features which belong to either Category 2 or Category 3, and a few from Category 1.
2. Two approaches can be tried from here:

a. Apply a quantile transformation on features from Category-2 to convert them to Normal Distribution (Category-1). However, it will only preserve the relative ranks of the values and might change the correlation with other variables.

b. Choosing a normalisation technique that is robust to the outliers. This has to be taken care of while training.

- Correlation Matrix

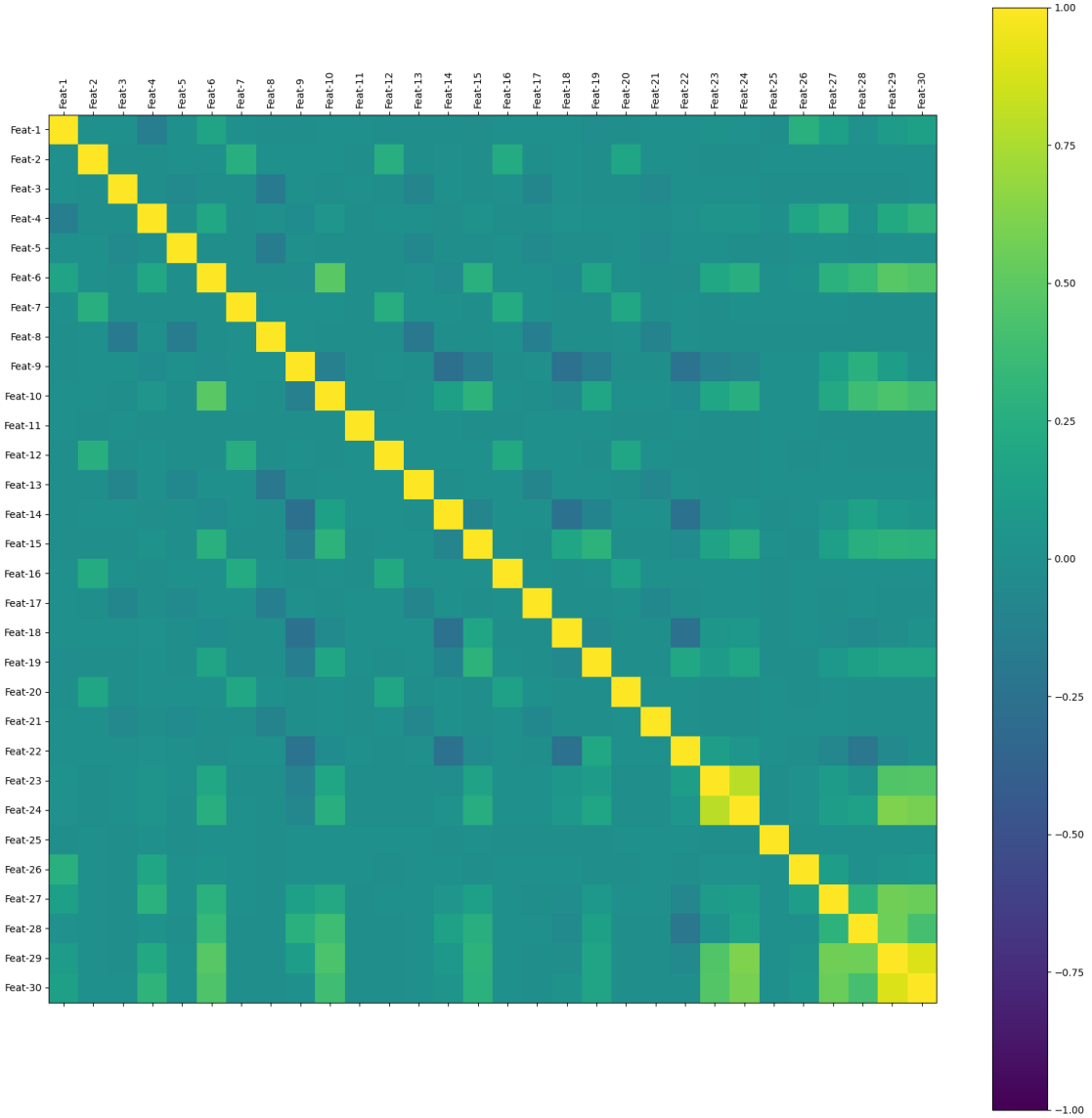

Figure 3: Correlation Matrix for the 30 features. Light yellow corresponds to a positive correlation, and Purple corresponds to a negative correlation.

Observations:

1. Positive Correlations between features are prevalent in the dataset.
2. The following is the list of features with their co-efficients

Feature (i)	Feature (j)	Correlation Co-efficient
Feat-23	Feat-24	0.7953563120204956
Feat-24	Feat-29	0.611498159974583
Feat-24	Feat-30	0.588871999811396

Feat-27	Feat-29	0.5691329190140545
Feat-27	Feat-30	0.5492615090028847
Feat-28	Feat-29	0.5691329190140545
Feat-29	Feat-30	0.8946638804393859

3. How to deal with this?
    a. Apply PCA to the dataset before training any model.
    b. Use some feature-selection technique such as Variance Threshold, Information Gain or KL-Divergence can be used. Apart from that, selection of k-best features on basis of statistical tests is also possible.

- Mutual Information Scores

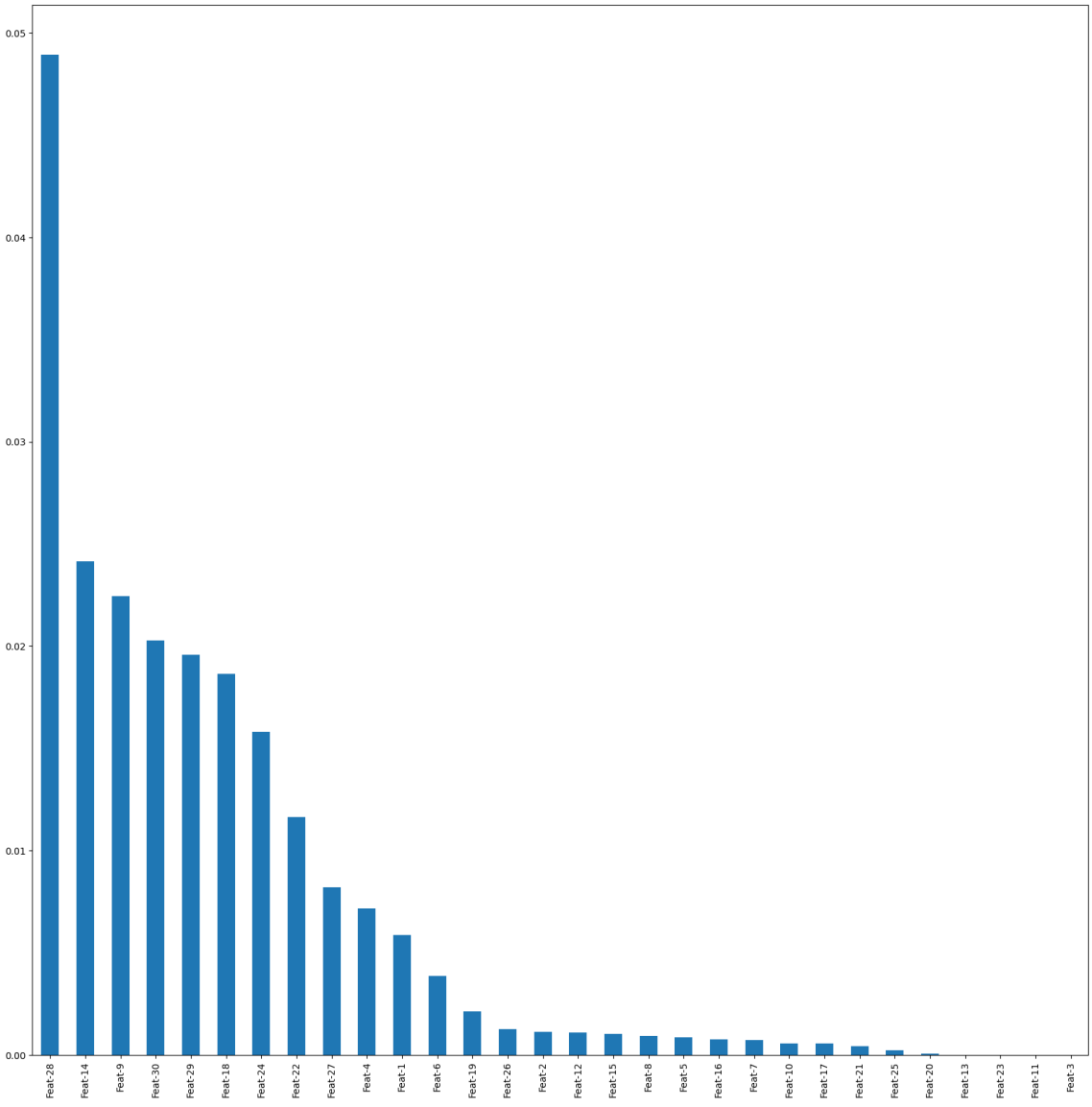

Figure 4: Mutual Information plot for the features

Observations:

1. The highest score is of Feat-28. But that too is around 0.05
2. This means, these variables alone are not conclusive enough to make predictions on the target
3. Each one of them is almost independent of the target variable.

Data Pre-Processing

- The dataset was split into train and validation sets with their in the ratio 0.7 : 0.3. This was done to ensure that the models do not overfit to the training set.
- All the variables were z-score normalised. I tried to first apply a quantile transformation on the skewed features (Category - 2) and then apply z-score normalisation to the complete dataset. I found out that this particularly didn't help with ANNs (as expected), but showed some improvements when using regressors. However, the absoute accuracies of the regressors were very low nonetheless, so I chose to discard the values. Normalisation ensures that the values belong to a common scale, without distorting the difference in ranges.

Experiment Results

The experiments can be found in the `experiments` folder within the submission. Each experiment has a different notebook. The results are stored in the cell outputs on the respective notebooks.

Logistic Regression

Name	val-accuracy	train-accuracy	val-loss	train-loss	val-precision	val-recall	val-f1-score
Logistic-Regressor	0.642158	0.64105	0.63730	0.63820	0.53, 0.74	0.58, 0.74	0.58, 0.69
Logistic-Regressor-PCA(n_comp = 0.8)	0.602875	0.602675	0.65839	0.65895	0.59, 0.61	0.49, 0.7	0.54, 0.65

Decision Tree

Name	train-accuracy	train-loss	val-accuracy	val-loss	val-precision	val-recall	val-f1-score
Decision-Tree(depth=5)	0.66723	0.60282	0.66435	0.60463	0.66, 0.67	0.59, 0.73	0.62, 0.7
Decision-Tree(depth=10)	0.721125	0.54300	0.69766	0.63534	0.68, 0.71	0.67, 0.73	0.67, 0.72
Decision-Tree(depth=15)	0.81516	0.39769	0.68335	2.69780	0.66, 0.7	0.67, 0.7	0.66, 0.7
Decision-Tree(depth=20)	0.92654	0.18085	0.66054	7.70775	0.64, 0.68	0.64, 0.68	0.64, 0.68

Notice how increasing the max depth leads to over-fitting on the train dataset

Random Forest

Name	train-accuracy	train-loss	val-accuracy	val-loss	val-precision	val-recall	val-f1-score
Random-Forest(n=100, depth=10)	0.731375	0.55893	0.711775	0.57429	0.71, 0.72	0.66, 0.76	0.68, 0.74
Random-Forest(n=200, depth=10)	0.73194	0.55814	0.712325	0.57336	0.7, 0.72	0.67, 0.75	0.69, 0.74
Random-Forest(n=100,	0.84496	0.44434	0.72473	0.54885	0.71, 0.73	0.69, 0.76	0.7, 0.74

depth=15)							
Random-Forest(n=200, depth=15)	0.84735	0.44290	0.72555	0.54780	0.72, 0.73	0.69, 0.76	0.70, 0.75
Random-Forest(n=300, depth=15)	0.84626	0.44408	0.72658	0.54762	0.72, 0.74	0.69, 0.76	0.70, 0.75

Increasing the depth as well as number of estimators lead to a small improvement in performance, at the cost of increased training time

Gaussian Naive Bayes

Since a lot of the features were distributed , I thought it was a good idea to try out Gaussian Naive Bayes so that I could somehow incorporate the priors. However, as evident from the results below, it didn't quite work out

1. Train Accuracy: 0.60087
2. Train Loss: 0.79289
3. Validation Accuracy : 0.59999
4. Validation Loss: 0.79126
5. Precision: 0.64, 0.59
6. Recall: 0.34, 0.83
7. f1-score: 0.44, 0.69

Artificial Neural Network

I tried out a bunch of architectures, varying the number of hidden layers, learning rates, early stopping and normalisation/scaling techniques. However, I did not record results for all of them. Here are the results for the submission I have selected on Kaggle

1. Train Accuracy: 0.74718
2. Train Loss: 25.28178
3. Validation Accuracy : 0.74678
4. Validation Loss: 25.32038
5. Precision: 0.46, 0.53
6. Recall: 0.47, 0.52
7. f1-score: 0.46, 0.53

ANNs seem to be achieving the max accuracy.

Observations

1. Logistic Regression performs well with linearly separable data. However, often times this is not the case with most of the datasets. Here, even after applying PCA on the dataset and then using Logistic Regression, we don't see a major change in performance. The curse of dimensionality also comes into the picture when there are many random variables.
2. The improvement of Random Forest over Decision Trees was expected. Increasing the max depth for decision trees increased overfitting, but Random Forest was robust to the change and even showed increased performance. Since different trees in Random Forest use a different set of features at each level, this tends to have a regularizing effect. This, along with the increased number of estimators, decreases overfitting as observed in the results.
3. Naive Bayes performs poorly on the dataset, but this is can be attributed to my training setup for GNB. Some of the features have high correlation, which weren't removed from training. The continuous variables can be binned to get categories, since Naive Bayes performs better in that case. All of this constitutes for future work.
4. ANNs seem to be performing decently well enough from the above experiments. This can be attributed to multiple factors:
    a. Universal Function Approximators: Their structure allows them to model and learn complex non-linear relationships and patterns within data. It's clear from the results of logistic regression that the relationships within the data is not trivially linear.
    b. Ability to extract meaning information from the features: Some of the features were highly correlated, as seen from the correlation matrix. ANNs reduce the need of manual feature engineering in many cases, and can figure out more comprehensive representations of input data.
    c. Even with the increasingly complex structures of neural networks, techniques such as regularization, learning rate scheduling and optimizers allow for better solutions at a faster convergence rate.

This is the reason why I chose to go ahead with ANNs, given their versatility and the data insights I obtained from other methods. Even at the expense of more training time as compared to others, the increase in accuracy was a tradeoff I believed was valid to make.

Evaluation Metrics

- Loss: A numerical measure of how well the predictions match with the actual output. The criterion used was Binary Cross Entropy, which is well-suited for binary classification tasks
- Accuracy: Percentage of correct predictions out of the total predictions
- Precision: Proportion of correctly predicted positive instances out of all instances predicted as positive. It gives an indication of the model's ability to avoid false positives.
- Recall: measures the proportion of correctly predicted positive instances out of all actual positive instances in the dataset. It indicates the model's ability to identify all relevant instances, avoiding false negatives.
- F1-score: harmonic mean of precision and recall. It provides a balanced measure that considers both precision and recall. Particularly helpful with imbalanced class distributions within the dataset.
- Confusion Matrix: (Not shown in the results) a matrix representation of the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions made by the model.
- Relevant Formulae:
  - Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  - Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

  - Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

  - F1-Score

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

  - Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Future Work

1. Try out *XGBoost* and *SVM*s on the dataset. Although I don't expect SVMs to perform very well given results of other methods, but still can be included for completeness.
2. More manual feature engineering. Leaving out variables(on metrics like information gain, correlation values, etc.), feature rankings, variable transformation(logarithmic, reciprocal, square-root etc.), encoding and discretizing variables, etc.
3. Decision Trees: Try changing parameters like split method(best, random) and class weights

4. Random Forest: Change split criterion (gini, entropy, log loss), max-depth, number of estimators and boostrapping
5. ANNs: Vary number of hidden layers, size of hidden layers, different initializations (kaiming, xavier, etc), learning rates, schedulers, etc.
6. Evaluation Metrics such as F-Beta score, MCC, ROC curves for all the above methods