TELECOM CHURN PREDICTION USING VARIOUS ML ALGORITHMS
(EEN-366: Course Project)

Sachin Agrawal
sachin_a@ee.iitr.ac.in
Enrolment: 19115108

Abstract

This project assesses several machine learning algorithms to evaluate how they perform on a tabular database. The tabular database used here has 26 columns (i.e., features), and each row is a sample. We must predict whether the company will lose the customer or not. There are two classes – yes and no, with yes meaning the customer will leave the company. I have used 70% of the samples for training, 20% for validation and 10% for testing. In my tests, Naïve Bayes came out as the best model for this problem.

1. Main Objectives

- Predicting whether the company will lose or keep the customer using different ML algorithms
- Comparing the results obtained by these models and deciding the best model for such a dataset

2. Introduction

For companies, it is crucial to retain customers. Customers always try to look for the best service for their hard-earned money; hence they are not loyal to a single company. There are also so many competitors of each company as well. So, it has become increasingly important for companies, like telecom providers, to retain as many customers as possible. When customers leave the company, it is known as churn.

As a solution, we can use machine learning to pre-determine which customers have a high probability of leaving the company. The company can improve the service of these customers and provide special offers to them to retain them. However, many algorithms are available to us, like neural networks, SVM, KNN, Logistic Regression, decision trees, etc. So, in this project, I am trying to find out the best algorithm for this task by comparing them using metrics like accuracy on test set, f1 score, area under ROC curve, etc.

For this project, I have used a tabular dataset collected by a telecom company which consists of various features about each customer (like age, gender, monthly charges, payment method, etc.) and whether the customer left the company in the last quarter. I have split this data into training, validation, and test set (70% training, 20% validation and 10% test).

After this, the training set is used to train various ML models: ANN, Decision Tree, Random Forest, Decision Tree with AdaBoost, SVM, KNN, Logistic Regression and Naïve Bayes. The hyperparameters of these models are tuned using the validation dataset. Finally, various comparison metrics are calculated on the test set to find out the performance of each model.

#	Column
0	gender
1	SeniorCitizen
2	Partner
3	Dependents
4	tenure
5	PhoneService
6	MultipleLines
7	OnlineSecurity
8	OnlineBackup
9	DeviceProtection
10	TechSupport
11	StreamingTV
12	StreamingMovies
13	PaperlessBilling
14	MonthlyCharges
15	TotalCharges
16	Churn
17	InternetService_0
18	InternetService_DSL
19	InternetService_Fiber optic
20	Contract_Month-to-month
21	Contract_One year
22	Contract_Two year
23	PaymentMethod_Bank transfer (automatic)
24	PaymentMethod_Credit card (automatic)
25	PaymentMethod_Electronic check
26	PaymentMethod_Mailed check

Table 1: Features of each customer

3. Dataset and Pre-processing

The dataset, available on Kaggle [1], has 7032 samples of customers. Each customer has 26 features, which are given in the table above. The column 'churn' is the value to be predicted. If it is 'Yes', the customer leaves the company; otherwise, they stay with the company. Now, non-numerical values must be converted to numerical to train our models. For gender, male and female are converted to 0 and 1, respectively. Yes and No are converted to 1 and 0. The columns' Total Charges', 'Tenure' and 'Monthly Charges' already consist of numerical values.

3.1 Scaling the data

During scaling, we bring all the features under the same scale. Scaling helps our model understand the data. If the feature values are on a similar scale, our models can understand the data better and give better results. [2] Most of the features consist of just 0 and 1 as their values, so I have scaled all the columns between 0 and 1. The only columns that need to be scaled are 'Monthly Charges', 'Tenure' and 'Total Charges' as their values vary greatly. I have used a min-max scaler, assigning 0 to the minimum value and 1 to the maximum value and scaling the rest of the data accordingly.

3.2 Imbalanced Data

The below chart shows the imbalance in the dataset. Only 27% of data belongs to class 'Yes'. An unbalanced dataset is not good because most ML algorithms try to maximise the accuracy. Here, we are concerned with customers that are leaving, i.e., those who belong to the 'Yes' class.

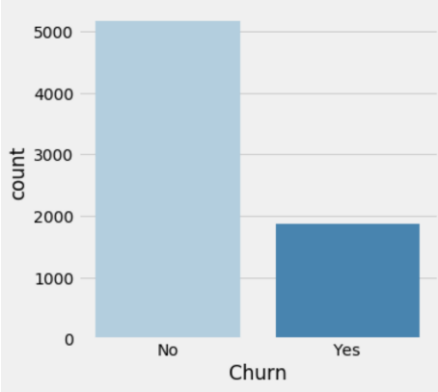

Figure 1: Visualisation of imbalance in data

3.3 Visualising the data in two dimensions

I have used Principal Component Analysis (PCA) [3] to change the basis of this dataset into two dimensions. This will help us to visualise the data in two dimensions. However, we must note that PCA selects the two most significant components of all (here, 26), so there is some loss of data. I have only used PCA for visualisation. From the Figure, we can see a loose boundary between the classes.

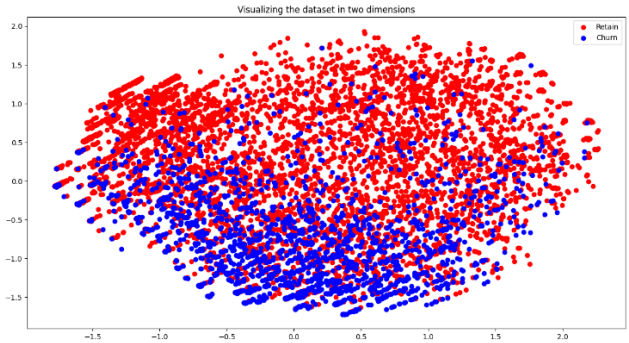

Figure 2: Visualizing the data in two dimensions

3.4 Split into train, validation, and test sets

The training set (70%, 4922 samples) is used to train a model, the Validation set (20%, 1406 samples) is used to tune hyperparameters, and the Test set (10%, 704 samples) is used to calculate the final metrics for comparison.

4. Artificial Neural Network (ANN)

In the ANN model [4], the input layer has 26 neurons (equal to the number of features). The output layer has just one neuron, whose activation determines the probability of class 'Yes' or 'No'. After testing the validation set, I used two hidden layers with 64 and 32 neurons, respectively. The hidden layers have 'relu' activation function while the output layer has sigmoid activation function. After training for 100 epochs, the accuracy achieved on the validation set is 75.71%. For class 'Yes', precision = 0.60, recall = 0.42, f1-score = 0.50 and area under ROC = 0.66.

In the following Figure, the training curve for this model is given. As the model trains, the training accuracy increases; however, validation accuracy decreases. This is a clear sign that our model is overfitting on the training set, i.e., it is trying to give the best result on the training set rather than generalising. The best way to reduce overfitting in neural networks is to use Dropout, which I have done in the next section.

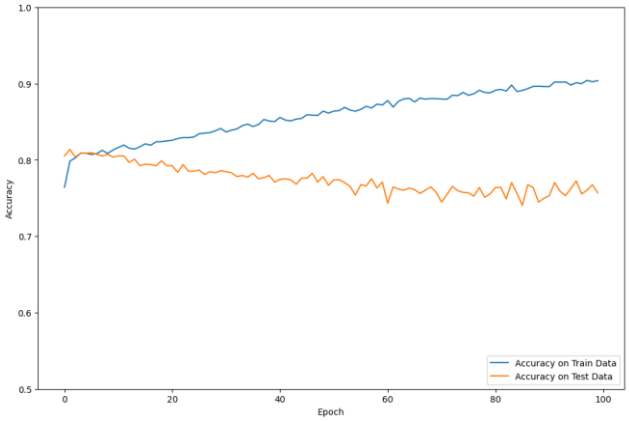

Figure 3: Training curve of ANN model

4.1 ANN with Dropout

I have added two dropout layers [5], one after each hidden layer. The rate of both is set to 0.5. The dropout layer will randomly drop neurons from the hidden layers to reduce overfitting. After training for 100 epochs, the accuracy achieved on the validation set is 77.56%. For class 'Yes', precision = 0.65, recall = 0.46, f1-score = 0.54 and area under ROC = 0.68. The Dropout model has performed better on every metric. The curve of training and validation accuracies with the number of epochs is given below.

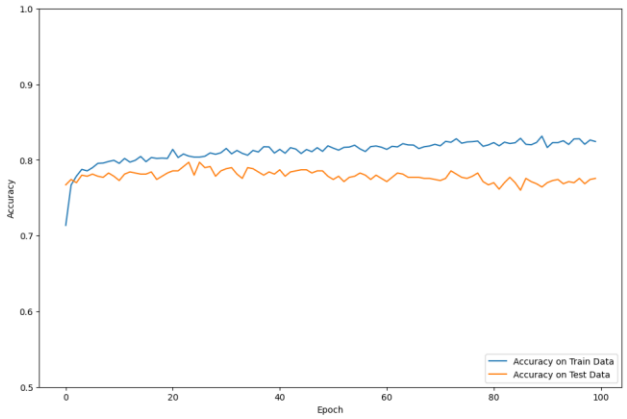

Figure 4: Training curve of ANN model with Dropout

4.2 ANN on Test Set

As the Dropout model performed better on the validation set, we should use it for testing on the test set. Results: Accuracy = 79.16%, precision = 0.61, recall = 0.51, f1-score = 0.55, area under ROC = 0.70, and training time = 143 sec.

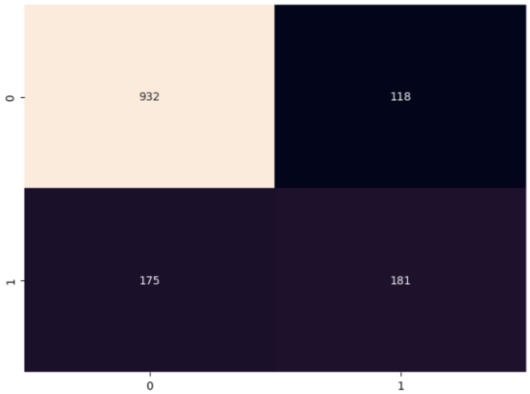

Figure 5: Confusion Matrix for ANN

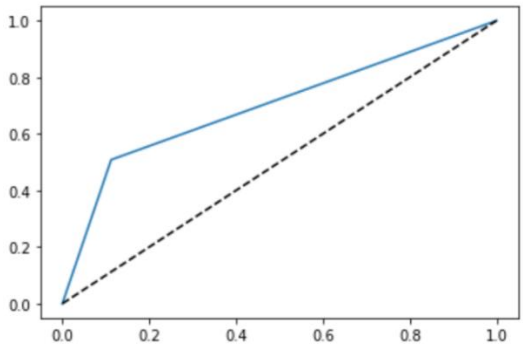

Figure 6: ROC Curve for ANN

4.3 Critical Points on ANN

- ANN takes a significant amount of time to train.
- Overfitting can be reduced by using Dropout.
- ANN has a lot of hyperparameters, so finding a good model can take a lot of time.
- ANN can give excellent accuracy if proper consideration is given to selecting hyperparameters.

5. Decision Tree

For decision tree [6], I have used entropy and information gain as the criteria for splitting. A decision tree can go up to a depth equal to the number of features of the dataset (26 in our case); however, we can stop the decision tree from spreading early. This helps reduce overfitting as otherwise, the decision tree will try to fit every sample in the training set, including noise. I trained a decision tree over several values of max_depth and calculated the accuracy of the validation set.

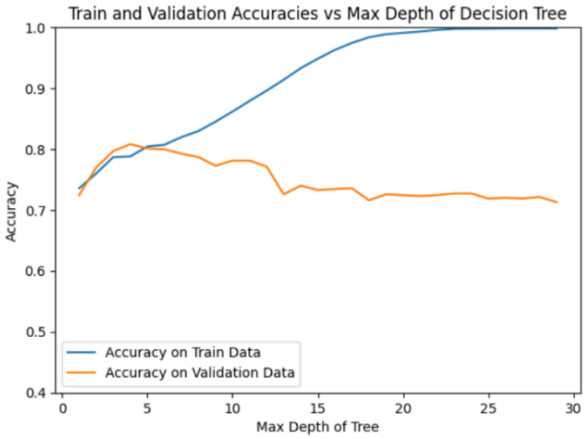

Figure 7: Training and Validation Accuracies vs Depth

From the graph, we can see that as the depth of the tree increases, our tree starts overfitting as it tries to fit every training example instead of generalising. The maximum accuracy on the validation set occurs at max_depth = 4, which we can use for final testing on the test set. This method is known as **early stopping**.

5.1 Decision Tree on Test Set

Results: Accuracy = 79.66%, precision = 0.65, recall = 0.52, f1-score = 0.58, area under ROC = 0.71 and training time = 0.02 sec.

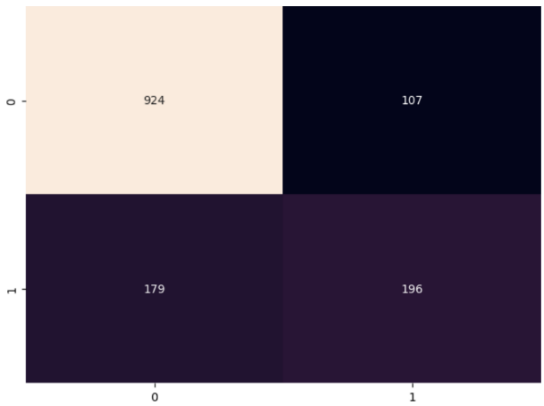

Figure 8: Confusion Matrix for Decision Tree

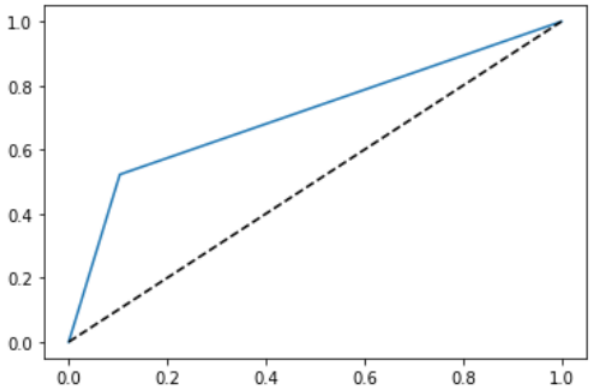

Figure 9: ROC Curve for Decision Tree

5.2 Critical Points on Decision Trees

- Decision tree takes significantly less time to train.
- Decision tree tries to fit every single example in the training set, including the noise, if it is allowed to branch out completely.
- Early stopping helps in reducing overfitting.

6. Decision Tree with AdaBoost

AdaBoost [7] is an ensemble learning method which combines several weak learners and takes a weighted sum of their outputs. I have used decision trees with max_depth = 4 as the classifier used by AdaBoost. We can vary the number of trees AdaBoost uses. I have used different numbers of trees and calculated the validation accuracies, as shown in the graph below.

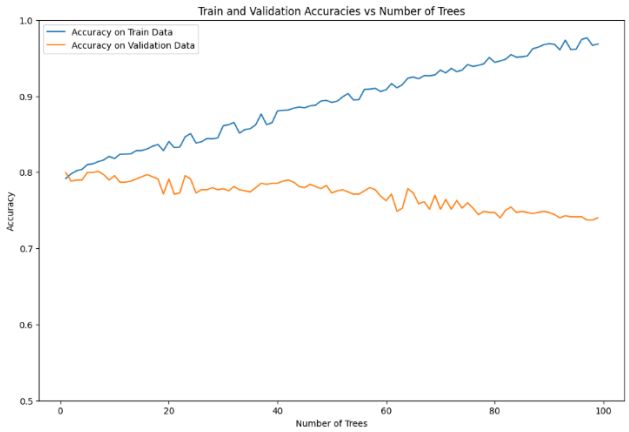

Figure 10: Train and Val Accuracy vs No. of Trees

Our AdaBoost classifier is also overfitting because more weights are given to misclassified examples in the training set, which can be noisy. Maximum validation accuracy occurs at number of trees = 7, so we will use this value to generate results on the test set.

6.1 AdaBoost on Test Set

Results: Accuracy = 79,87%, precision = 0.61, recall = 0.60, f1-score = 0.61, area under ROC = 0.73 and training time = 0.11 sec.

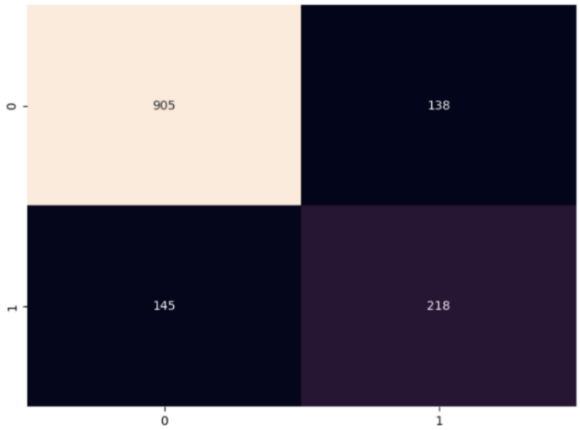

Figure 11: Confusion Matrix for AdaBoost

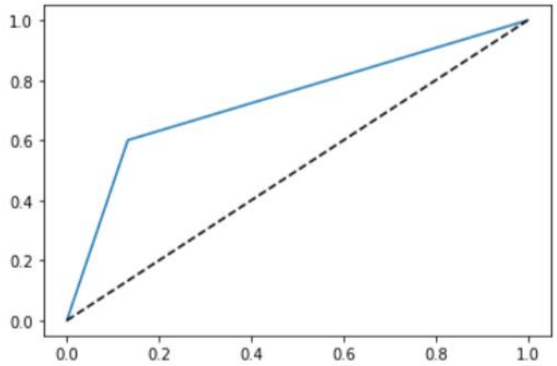

Figure 12: ROC Curve for AdaBoost

6.2 Critical Points on AdaBoost

- AdaBoost increases the accuracy of weak classifiers. Here, AdaBoost gave better accuracy than its underlying classifier, Decision Tree.
- AdaBoost can overfit as it gives more weight to incorrectly classified data, which may be noise.
- AdaBoost can take a lot of time to train if the number of classifiers is high,

7. Random Forests

Random Forest [8] uses bagging to select samples to train several trees. I have taken the max_depth of each tree as 4. After varying the number of trees in the forest, we get the following graph for validation accuracy. There are a lot of fluctuations because of randomness. Maximum validation accuracy is achieved at number of trees = 17. I have taken this to calculate the results on the test set.

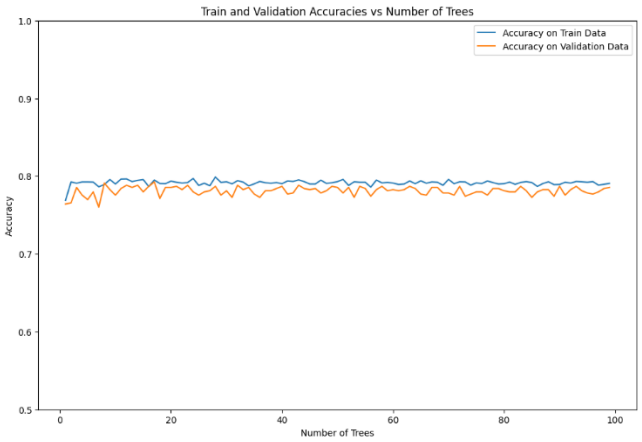

Figure 13: Train and Val Accuracy vs No. of Trees

7.1 Random Forest on Test Set

Results: Accuracy = 80.94%, precision = 0.72, recall = 0.43, f1-score = 0.54, area under ROC = 0.68 and training time = 0.06 sec.

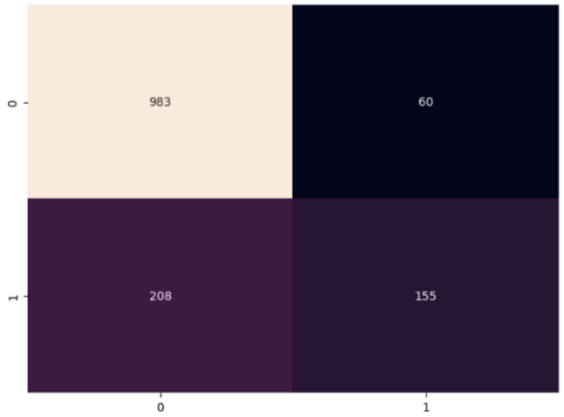

Figure 14: Confusion Matrix for Random Forest

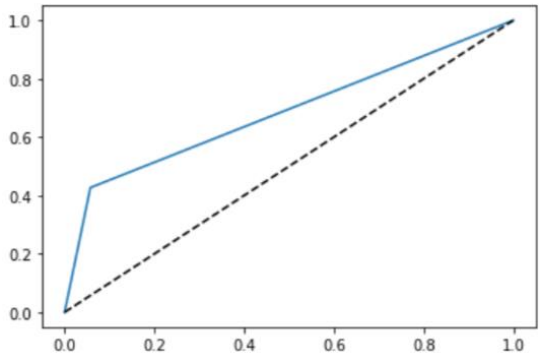

Figure 15: ROC Curve for Random Forest

7.2 Critical Points on Random Forest

- Random forests take little more time for training than decision trees.
- Random forests reduce overfitting as the samples are randomly selected using bagging for each tree.
- Due to randomness during bagging and each split in each tree, the results vary widely in each model run with the same hyperparameters.

8. Support Vector Machines (SVM)

In the SVM model [9], I have varied the penalty term (C) value and calculated the validation accuracies. Also, the kernel is taken as linear. C is the penalty applied for every misclassification. If C has an extremely high value, the SVM model cannot misclassify any point (hard-margin SVM). However, we can only use soft-margin SVM on this dataset (see Figure 2). From the graph below, we can see that validation accuracy decreases as the value of C increases. This may be because the margin of SVM shrinks as C increases. I have selected C = 1 for the test set as we get the highest validation accuracy at C = 1.

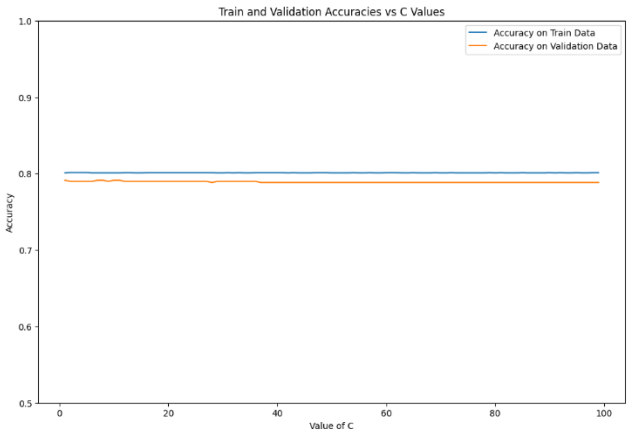

Figure 16: Train and Val Accuracy vs Value of C

8.1 SVM on Test Set

Results: Accuracy = 81.22%, precision = 0.65, recall = 0.59, f1-score = 0.62, area under ROC = 0.74 and training time = 0.85 sec.

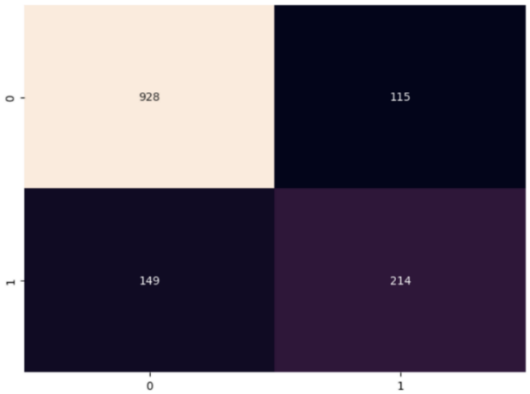

Figure 17: Confusion Matrix for SVM

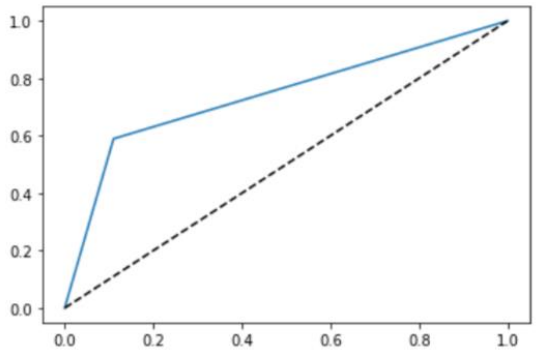

Figure 18: ROC Curve for SVM

8.2 Critical Points on SVM

- SVM works well in a higher dimensional dataset because it can select the features along which there is the highest variance.
- SVM takes more training time than Decision Trees and KNN. Training time will further increase with the number of features and samples.
- SVM does not work well when there is a lot of overlap between classes.
- In our dataset, there are many features and some overlap, as seen in Figure 2.

9. K Nearest Neighbours (KNN)

In KNN [10], there are two hyperparameters – the first is K, the number of neighbours, and the second is weights. We can either use majority voting or use distance-weighted KNN, which gives more weight to closer samples. The below graph shows the accuracy of the validation set of uniform KNN and distance weighted KNN. Maximum validation accuracy is achieved on uniform KNN at K = 13. So, I have used this on the test set to get results for KNN.

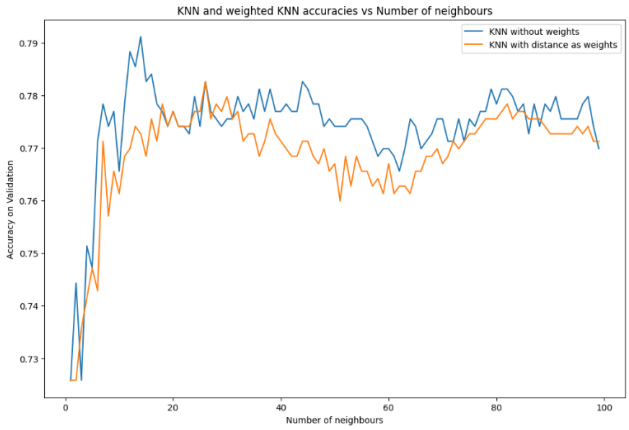

Figure 19: Accuracy of KNN and distance-weighted KNN vs value of K

9.1 KNN on Test Set

Results: Accuracy = 77.81%, precision = 0.57, recall = 0.59, f1-score = 0.58, area under ROC = 0.72 and training time = 0 sec.

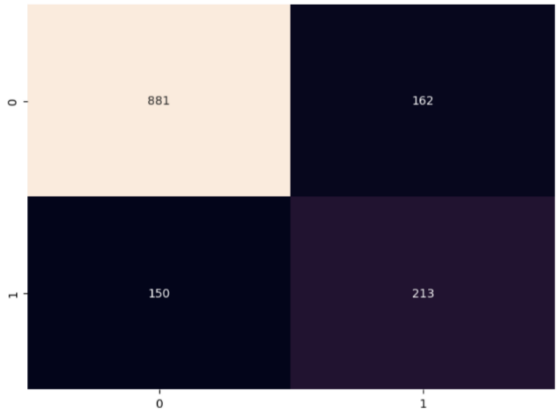

Figure 20: Confusion Matrix for KNN

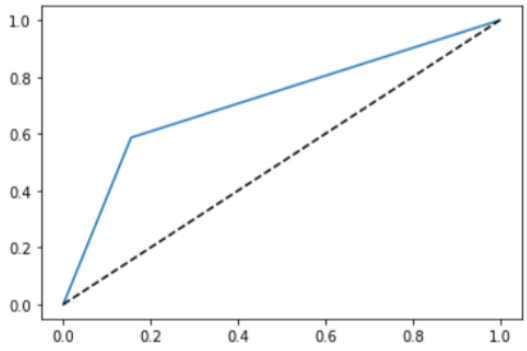

Figure 21: ROC Curve for KNN

9.2 Critical Points on KNN

- KNN takes no training time, so that we can add data to the training set anytime.
- KNN gives equal weightage to all features, so it works worse in higher dimensions than SVM because some features may be redundant or less useful.
- KNN is the most straightforward algorithm to implement and does not need many calculations.

10. Logistic Regression

Logistic Regression [11] is used in binary classification. I have used a simple logistic regression model trained using the training set and calculated the metrics using the test set.

10.1 Logistic Regression on Test Set

Results: Accuracy = 81.01%, precision = 0.69, recall = 0.53, f1-score = 0.60, area under ROC = 0.72 and training time = 0.27 sec.

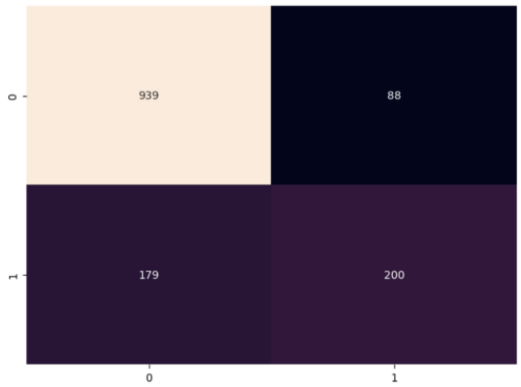

Figure 22: Confusion Matrix for Logistic Reg.

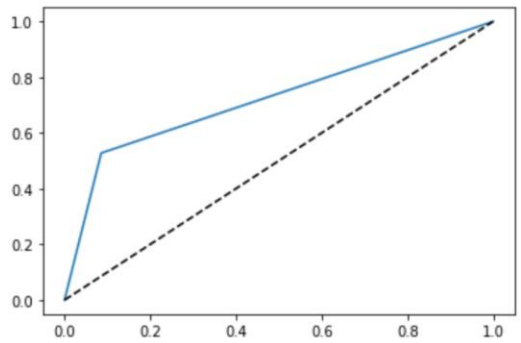

Figure 23: ROC Curve for Logistic Reg.

10.2 Critical Points on Logistic Regression

- Logistic Regression gives weights to features to determine how much a feature affects the output.
- If there are too many features, Logistic Regression can overfit.
- Logistic Regression cannot be used in multi-class problems.

11. Gaussian Naive Bayes

Naïve Bayes [12] uses the Bayes theorem to determine the probability of each sample being in a specific class. As our dataset has continuous-valued features (like 'Total Charges'), I have used Gaussian Naïve Bayes. The metrics are calculated on the test set after training on the train set.

11.1 Gaussian Naïve Bayes on Test Set

Results: Accuracy = 75.46%, precision = 0.53, recall = 0.78, f1-score = 0.63, area under ROC = 0.76 and training time = 0 sec.

11.2 Critical Points on Naïve Bayes

- Recall on the test set is higher than other models, which means it has identified the highest proportion of positive examples.
- Naïve Bayes can easily be used for multi-class problems, and it takes no training time.
- Naïve Bayes assumes all features are independent of each, which is rarely the case.

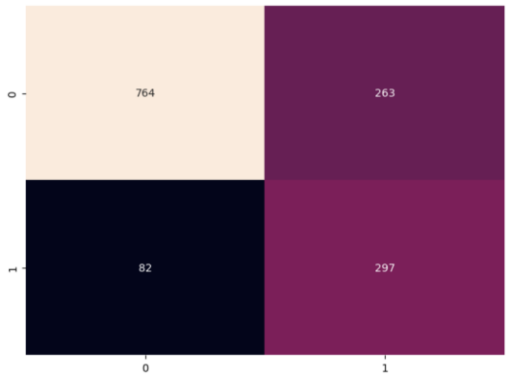

Figure 24: Confusion Matrix for Naïve Bayes

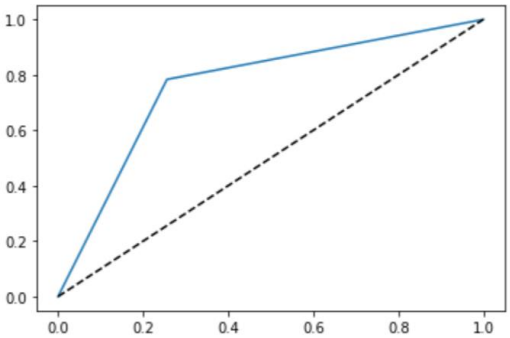

Figure 25: ROC Curve for Naïve Bayes

12. Results and Conclusions

After simulations, we have been able to achieve the results shown in Table 2. Different metrics suggest different models are the best. It is essential to know which metric should be used to find the best model. Every problem has different goals for which appropriate metrics must be used.

In our case, we want to find out the customers who will leave the customer. The company's goal is to successfully find out as many such people as possible so they can plan to convince them to stay. This suggests that recall is the best metric in this case. Recall is the ratio of true positive to (true positive + false negative), i.e., the proportion of positive samples we were able to find out. However, the company may also check the precision of the models so that they don't waste their offers on customers who are not going to leave. Training time is also essential here. The company regularly updates the data, so they need to train again and again.

Going by the recall, Naïve Bayes is the best model for the company as it found out that 78% of the customers are about to leave, the highest of any model. The training time of Naïve Bayes is negligible, so the company can regularly update the model. F1-score is also highest for Naïve Bayes.

Model	Accuracy	Precision	Recall	f1-score	Area under ROC	Training time (s)
ANN	79.16	0.61	0.51	0.55	0.70	143
DECISION TREE	79.66	0.65	0.52	0.58	0.71	0.02
ADABOOST	79.87	0.61	0.60	0.61	0.73	0.11
RANDOM FOREST	80.94	0.72	0.43	0.54	0.68	0.06
SVM	81.22	0.65	0.59	0.62	0.74	0.85
KNN	77.81	0.57	0.59	0.58	0.72	0.00
LOGISTIC	81.01	0.69	0.53	0.60	0.72	0.27
NAÏVE BAYES	75.46	0.53	0.78	0.63	0.76	0.00

Table 2: Results of ML models on the test set

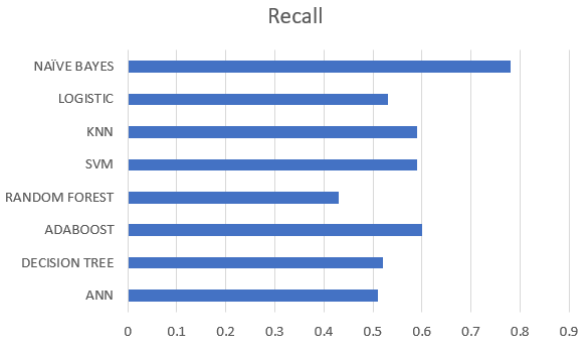

Figure 26: Comparison of recalls of ML models

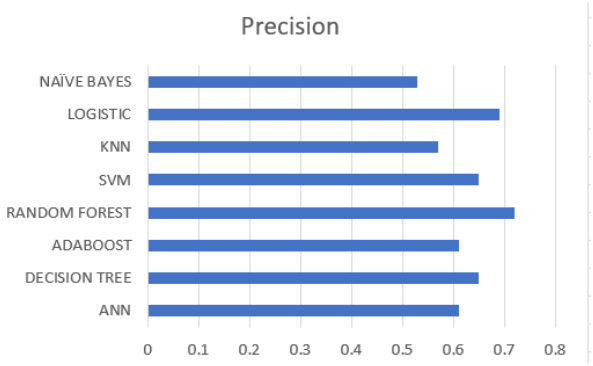

Figure 27: Comparison of precisions of ML models

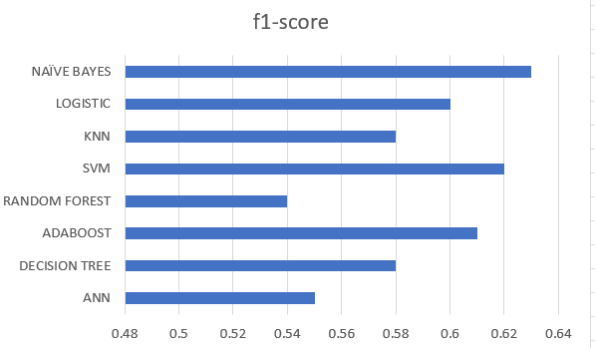

Figure 28: Comparison of f1-scores of ML models

Every dataset is different, and these models may give completely different metrics for them. It is essential to analyse all algorithms properly and select the best metrics for the need of the problem.

13. References

1. *Telco Customer Churn, available on Kaggle: https://www.kaggle.com/datasets/blastchar/telco-customer-churn*
2. *On Feature Normalization and Data Augmentation, Boyi Li, Felix Wu, Ser-Nam Lim, Serge Belongie, Kilian Q. Weinberger, arXiv:2002.11102*
3. *Principal component analysis: a review and recent developments, Ian T. Jolliffe and Jorge Cadima, 10.1098/rsta.2015.0202*
4. *Introduction to artificial neural networks, Enzo Grossi and Massimo Buscema, MEG.0b013e3282f198a0*
5. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(56):1929−1958, 2014.*
6. *Learning and Testing Decision Tree, Nader H. Bshouty, Haddad-Zaknoon, arXiv:2108.04587*
7. *The AdaBoost Flow, A. Lykov, S. Muzychka, K. Vaninsky, arXiv:1110.6228*
8. *Understanding Random Forests, Gilles Louppe, 2014, arXiv:1407.7502*
9. *Support Vector Machine Classifier via Soft-Margin Loss, Huajun Wang, Yuanhai Shao, Shenglong Zhou, Ce Zhang, Naihua Xiu, arXiv:1912.07418*
10. *k-Nearest Neighbour Classifiers: 2nd Edition, Padraig Cunningham, Sarah Jane Delany, arXiv:2004.04523*
11. *Introduction to logistic Regression, Moo K. Chung, arXiv:2008.13567*
12. *Bayes and Naive Bayes Classifier, Vikramkumar, Vijaykumar B, Trilochan, 2014, arXiv:1404.0933*