CS6140: Machine Learning

Homework Assignment # 6

Assigned: *Due: 04/23/2019, 11:59pm, through Blackboard*

SANTANDER CUSTOMER TRANSACTION PREDICTION (KAGGLE COMPETITION)

Tushar Sharma (sharma.tu@husky.neu.edu) , Rong Su (su.r@husky.neu.edu)

April 27, 2019

1 Objectives and significance:

The objective of this project is to identify which customer will make a specific transaction in the future given the transaction information. This is a binary classification problem based on an anonymized, labeled dataset which contains the features of the transactions for each of the customers. Our goal is to train a model of existing data, then make predictions on future transactions. Further, by exploring different model and tuned parameters, evaluate the accuracy and performance for each model on this dataset.

During lecture on Naive Bayes Classifier towards the end, professor talked about modelling Naive Bayes with continuous features using kernel density estimation for estimating feature distributions given class. This gave a good reason to think whether features can be estimated as mixtures of various distributions and eventually used in binary classification problems. Obective of this project will be to implement a Hybrid/Flexible Naive Bayes classification on "Santander Customer Transaction Prediction" dataset (an ongoing competition on kaggle). Exploring the kernels gave insights into how people have implemented Gaussian-approximated Naive Bayes and achieved an area under receiver operating characteristic curve (ROC curve) of 0.889. Although, features will follow a complex mixture of distribution and single gaussian approximation will induce errors in prediction. By implementing non-parametric kernel density estimation, aim is to fit the features to a better model and eventually increase the area under ROC curve.

2 Background:

Concepts introduction

- Data normalization: scaling attribute values to fall within a specified range.
- Data reduction: reducing the number of features.
- Training data: labeled data to train an algorithm.
- Classification: predicting the class(targets/ labels or categories) of given data points, a approximately mapping process.
- Binary classification: classifying the elements of a given set into two groups.

- Overfitting: a model that models the training data too well.
- Validation: the process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data.

Distinction of our work and Previous work done

Follow valid machine learning procedure: data preprocessing, model development, validation, and evaluation.For each of the procedure, compare the performance and accuracy of each methods. Non Parametric Naive Bayes is something uniques and not available on any kernels on kaggle. We will discuss it in detail below:

Naive Bayes classifier is a classification technique which gives a linear decision boundary within the data under certain conditions. The uber aim is to estimate the probability of class given the set of features. This can be represented as follows:

$p(c|x) = \frac{p(x|c).p(c)}{p(x)}$

where c $\in\{0,1\}$, p(x|c) is the combined probability of each feature given class and p(x) is the combination if distribution of 'd' features.

Now, generally p(x|c) say for each feature is modeled as a single gaussian as a good approximate which can be represented as follows.

$p(x|c) = \mathcal{N}(\mu_c, \sigma_c)$ [$\mathcal{N}$ represents gaussian distribution]

Believing in this approximation is particularly good when the plot of features show normal distribution given each class, one at at time. Although, in cases where the feature distribution is different from gaussian, may it be bumpy in between or unexpected troughs in the ends, we may use a more complex method of modelling these features to reduce error in predictions. One of these methods is called non-paramteric kernel density estimation as phrased in a paper titled "Estimating Continuous Distributions in Bayesian Classifiers" [Ref-2]. Using all gaussian kernels and averaging over all datapoints in a class, we will get the probability of feature given class as follows:

$p(x|c) = \frac{1}{n}\sum_i \mathcal{N}(\mu_c, \sigma_c)$

where i is the number of datapoints in set D.

Standard kernel density estimator is given by following formula,

$p(x|c) = (nh)^{-1}\sum_j K(\frac{x-\mu_i}{h})$, where h=$\sigma$ is the KDE parameter defines the width of the kernel and K=$\mathcal{N}(\mu_c,\sigma_c)$ is the distribution within the kernel.

Exploring a few kernels, gave good insights on Features, Test data and Gaussian Naive Bayes classifier. Kernel in reference [6] shows that features in test and training dataset follow very similar patterns. This suggests that if features given class are not following a strict gaussian then it is worthwhile to model them using KDE for probability density functions to better implement the learner on test dataset. Also, reference [6] suggests that gaussian Naive Bayes works pretty well with AUC of 0.889 on test data.

Evaluation of our learner is Area Under Receiver Operating Characteristic curve (AUC), an approach which checks true positive cases vs false positive cases. It is essentially a plot of function of true positive

rate(sensitivity function) which is dependent on false-positive cases or fallouts. More detailed explaination can be found of reference[7].

One more addition to this approach could be to use KDE to model mixture of different distributions (Gaussian+Uniform+Exponential) within the class to learn features more accurately and henceforth get better results on test dataset. This idea has been discussed in reference [4] and shall be looked into if time permits.

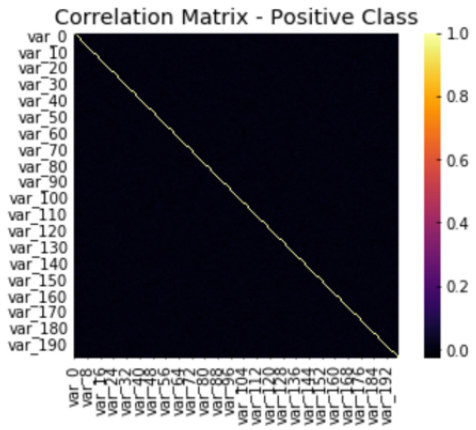
3 Method:

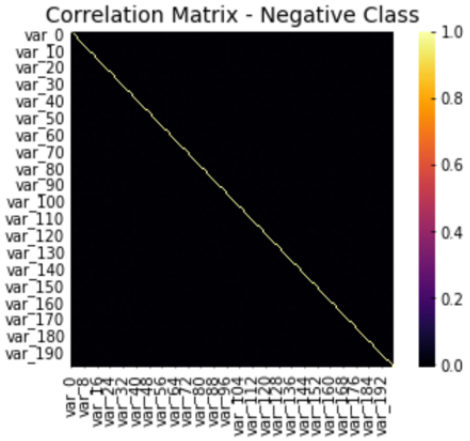
The dataset we will be using is "Santander customer transaction prediction" dataset available on the kaggle.com link provided in references. This is a pre-processed dataset with generic predictor variable names as the information in these fields is classified. There are 200 features and 200,000 rows with no missing values in the training dataset. The predicted class is 0 when customer did not make a particular transaction and 1 when customer made the transaction. Almost all these variables follow a gaussian or mixture of gaussian distribution with certain discrepancies (which we will most likely address in our solution). Most of the data processing and implementation required in this approach will be done using kaggle kernels in Python and/or R depending on the availability of funtions and libraries.

For this project, we have input data in csv format. Training data(labeled) has 200000 records, each with 202 features. Among these features, we have "ID_code" which is a unique index string for each of the entry. 200 numeric features named as "var_0, var_1, ... , var_199" which describe this record. And we have a binary column "target" in 0s and 1s which represents if this transaction is made or not. We need to use the value of "target" in training data to train model, then to predict value of "target" in testing dataset. To have a closer look of the features, Figure 1 shows the visualization of first five variables in first 20 rows. We observe that these five numeric variables show some combination of Gaussian distribution.

Figure 1: data distribution in first 20 rows of var_0, var_1, ... , var_4 in train.csv. By filter the "null" values among the data and numeric check, we know there's no missing values or mismatched data. Also we learn from simple plotting that this is an unbalanced target with target 0 vs. target 1 = 9:1. All these results are very useful information that will influence the analysis of models. Further, by learning the features, we got the graph of correlations among 200 numeric features as Figure 2. We can see from the graph that there's very low linear relationship between any features.

By filtering the "null" values among the data and numeric check, we know there's no missing values or mismatched data. Also we learn from simple plotting that this is an unbalanced target with target 0 vs. target 1 = 9:1. All these results are very useful information that will influence the analysis of models. Further, by learning the features, we got the graph of correlations among 200 numeric features as Figure 2. We can see from the graph that there's very low linear relationship between any features.





Models Used

- Logistic classification (sklearn.linear_model.LogisticRegression)
- Perceptron learning algorithm (sklearn.linear_model.Perceptron)
- Pocket algorithm
  - Pseudo-code
  - w = zero_vector
  - w_pocket = zero_vector
  - run = 0
  - run_pocket = 0
  - for iter in MAX_ITERS: # terminate by max iterations
    - for index in range(row_train):
      - xrow = xTrain[index, :]
      - if (xrow.dot(w) >= 0 and yTrain[index] == '0') or (xrow.dot(w) < 0 and yTrain[index] == '1'):
      - if run > run_pocket: w_pocket = w run_pocket = run
      - if yTrain[index] == '1': w += eta * xrow
      - else: w -= eta * xrow
      - run = 0
      - else: run += 1
  - return w_pocket
- Gaussian Naive Bayes (sklearn.naive_bayes.GaussianNB)
- Non Parametric Naive Bayes
  - import numpy as np from sklearn.base
  - import BaseEstimator, ClassifierMixin from sklearn.neighbors
  - import KernelDensity import multiprocessing as mp
  - class KDEClassifier(BaseEstimator, ClassifierMixin):
    - def __init__(self, bandwidth=1.0, kernel='gaussian'):

        - self.bandwidth = bandwidth
        - self.kernel = kernel
    - def fit(self, X, y):
        - self.classes_ = np.sort(np.unique(y))
        - training_sets = [X[y == yi]
        - for yi in self.classes_] self.models_ = []
        - self.models_.append([KernelDensity(
        - bandwidth=self.bandwidth, kernel=self.kernel).fit(np.asarray(i[j]).reshape(-1,1)) for j in i])
        - self.logpriors_ = [np.log(Xi.shape[0] / X.shape[0]) for Xi in training_sets]
        - return self
    - def predict_proba_pool(self, X, mdlpool):
        - return(mdlpool.score_samples(X))
    - def predict_proba(self, X):
        - logprobs = []
        - for mdl in self.models_:
        - logprobs.append(np.array(pool.starmap(self.predict_proba_pool,
        - [(X.iloc[:,(mdl.index(mdlclass)):(mdl.index(mdlclass)+1)],mdlclass) for mdlclass in mdl])).T)
        - logprobs_ = np.array([np.sum(i, axis = 1) for i in logprobs]).T
        - result = np.exp(logprobs_ + self.logpriors_)
        - return result / result.sum(1, keepdims=True)
    - def predict(self, X):
        - return self.classes_[np.argmax(self.predict_proba(X), 1)]

- Neural Network (sklearn.neural_network.MLPClassifier)

Evaluation Strategy

For most models(expect pocket), we are using 10-fold validation (sklearn.model_selection.StratifiedKFold) on the entire training data. Plot ROC and calculate AUC for each of the model. Below is pseudo code of plotting ROC. We will show the result in next chapter.

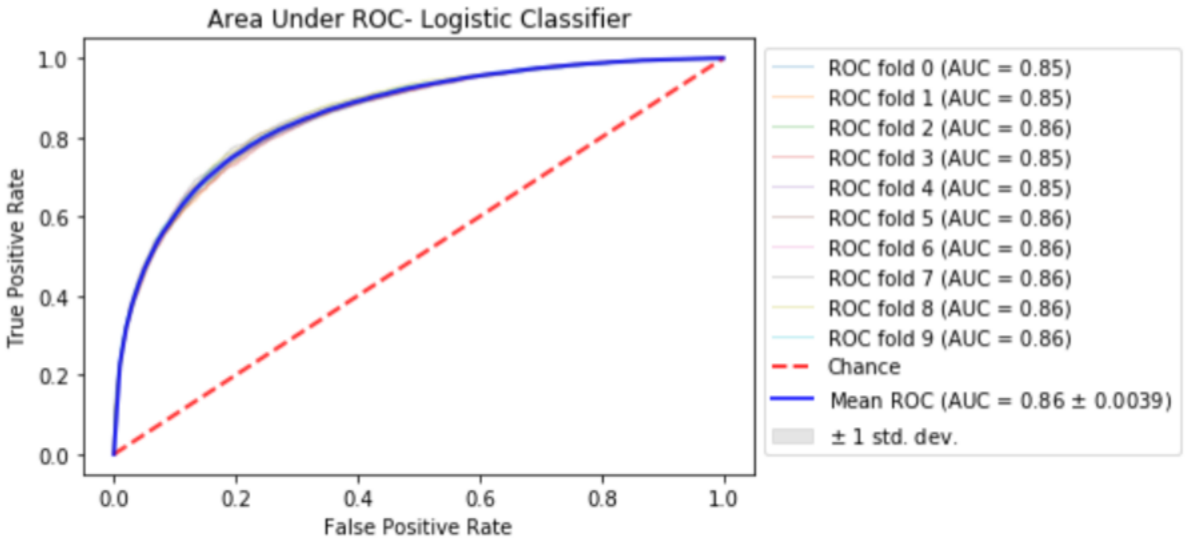
- cv = StratifiedKFold(n_splits=10)
- for train, test in cv.split(variables, target):
    - probas_ = classifier.fit(variables[train], target[train]).decision_function(variables[test])
        - # Compute ROC curve and area the curve
        - fpr, tpr, thresholds = roc_curve(target[test], probas_)
        - tprs.append(scipy.interp(mean_fpr, fpr, tpr))
        - tprs[-1][0] = 0.0
        - roc_auc = auc(fpr, tpr)
        - aucs.append(roc_auc)
        - plt.plot(fpr, tpr, lw=1, alpha=0.3, label='ROC fold %d (AUC = %0.2f)' % (i++, roc_auc))

4 Results

We start with a few basic linear classifiers and then go in depth into Naive Bayes.

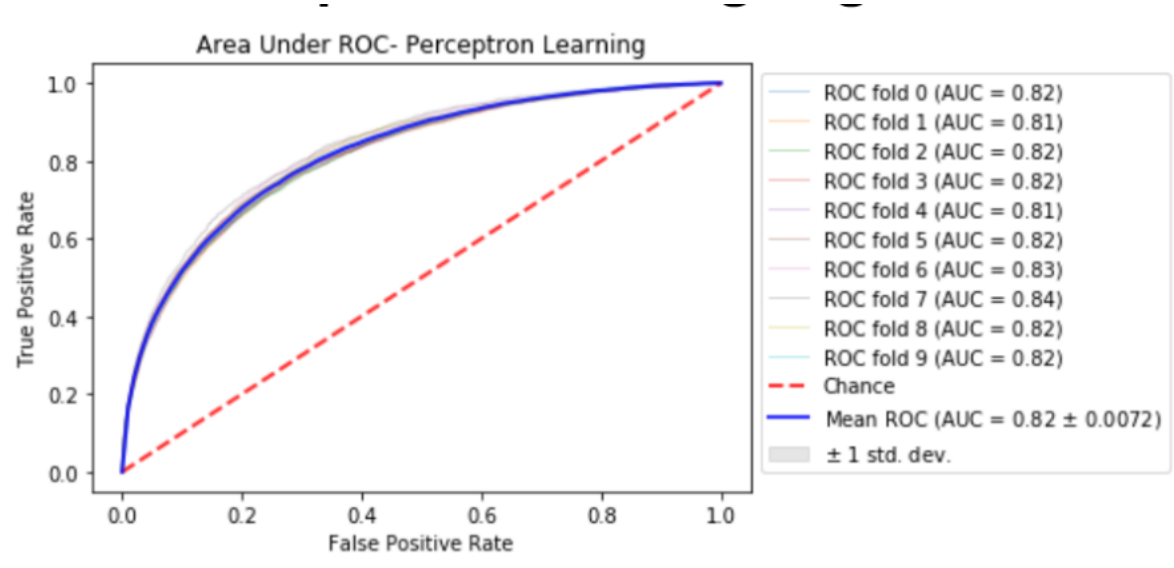
- **Logistic Classifier**

Learning the linear boundary $w^t x + w_0$:



- **Perceptron learning algorithm**

One more way to learn $w^t x + w_0$:



- **Pocket algorithm** (AUC was not possible using our implementation so rather presenting the accuracy and confusion matrix)

We are still learning $w^t x + w_0$.

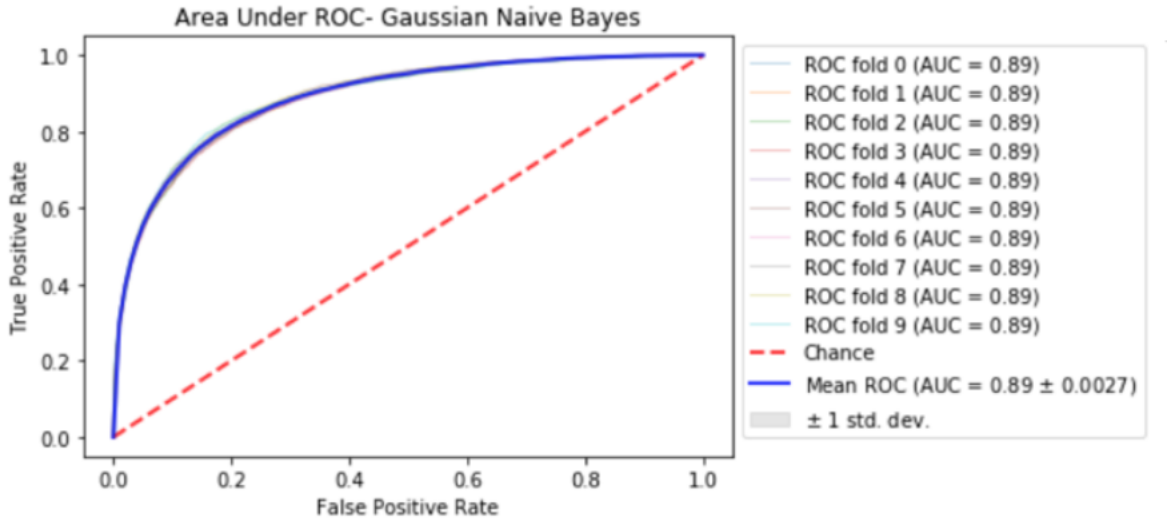
Confusion Matrix:

True Negative 16585	False Positive 896
False Negative 1706	True Positive 2294

Accuracy: 90%

- **Gaussian Naive Bayes**

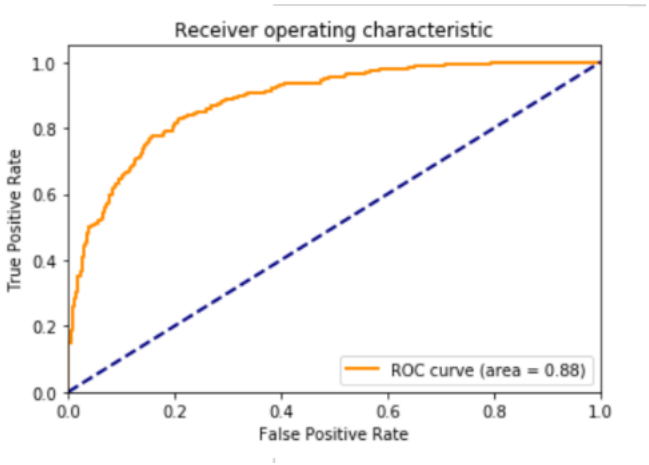
We know our features are somewhat independent and somewhat gaussian so modelling them using Gaussian Naive Bayes should give us better results than previously seen linear classifiers. Running a 10-Fold Cross Validation test on GNB gave us the following results:



- **Non parametric Naive Bayes (Only 1 fold for 80:20 split on training data)**

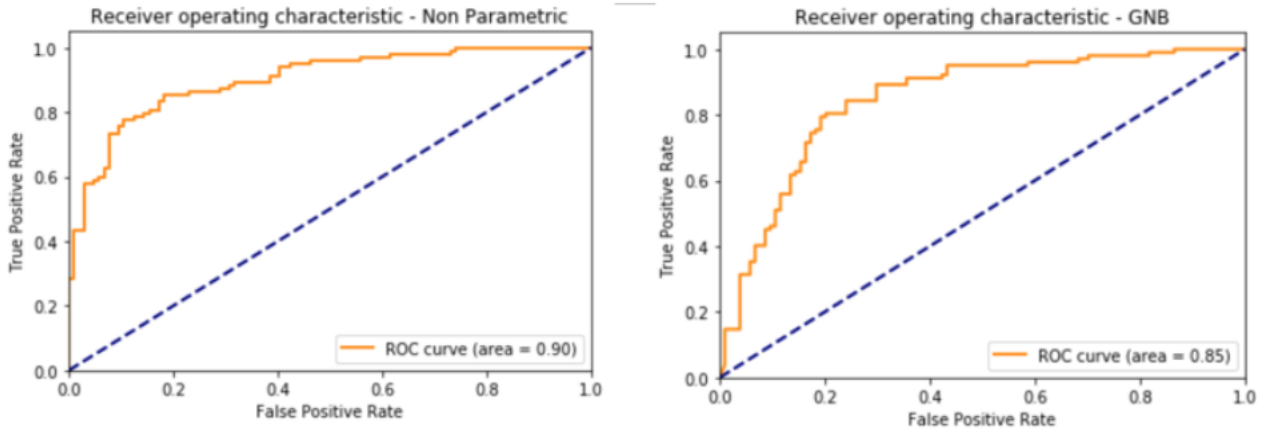
Not quite satisfied with the Gaussian Naive Bayes performance, we dig deeper into how to model continuous features, found the research paper mentioned in background and try to recreate the Non parametric process described in this paper.

Initial run: We faced some issues with run time due to the complexity of the model training and predictions. Due to this issue we could nnot perform either 10-Fold cross validation or grid search for best parameters efficiently. Initially we could get only AUC = 0.88 which is a bit shy of Gaussian Naive Bayes.

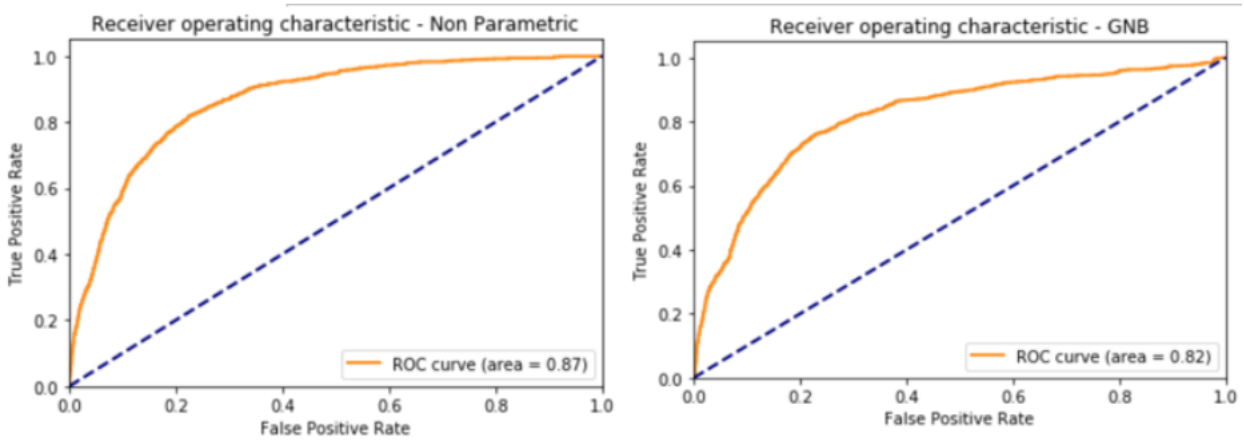


We ran a few tests on smaller data sets to understand the bandwidth parameter setting and comparison with Gaussian Naive bayes. The results are shown below:

Non Parametric vs Gaussian NB on concrete dataset from UCI ML repository:

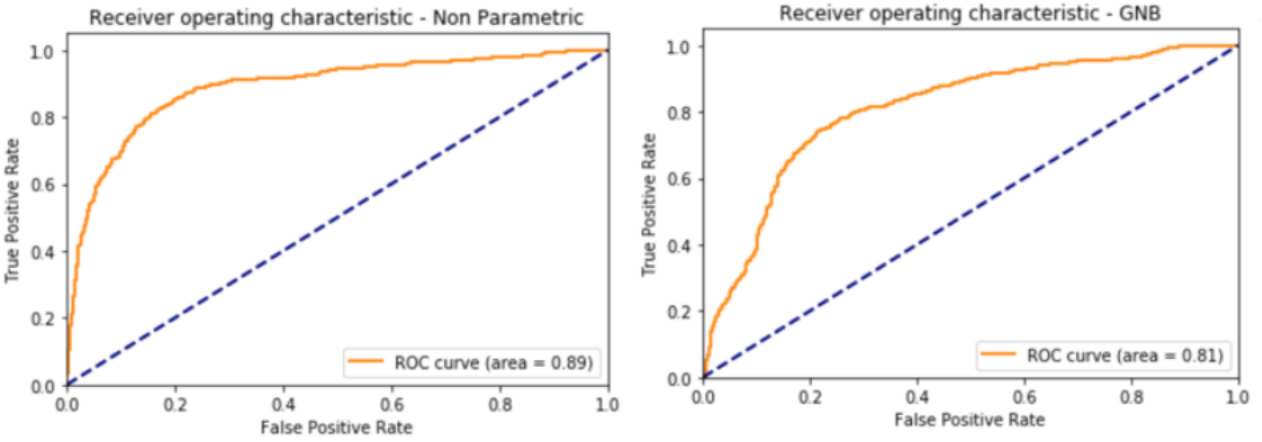


Non Parametric vs Gaussian NB on bank dataset from UCI ML repository:



ACCURACY - 88.5% **ACCURACY - 86.3%**

Non Parametric vs Gaussian NB on pendigit dataset from UCI ML repository:

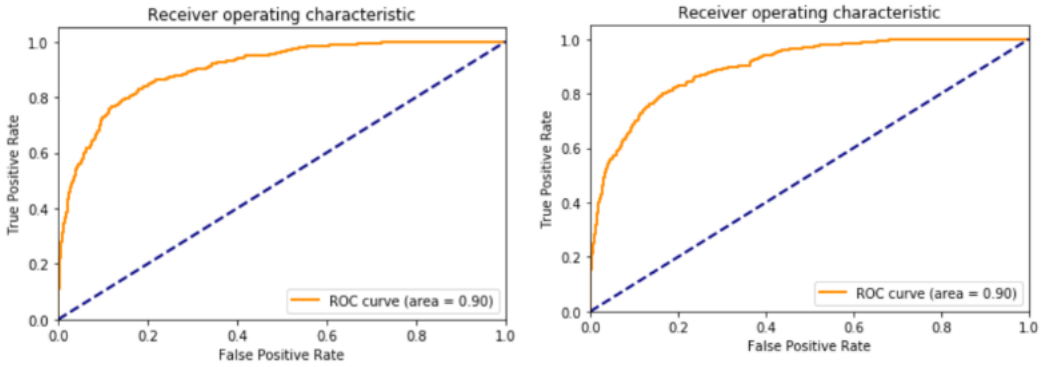


ACCURACY - 83.26% **ACCURACY - 77.77%**

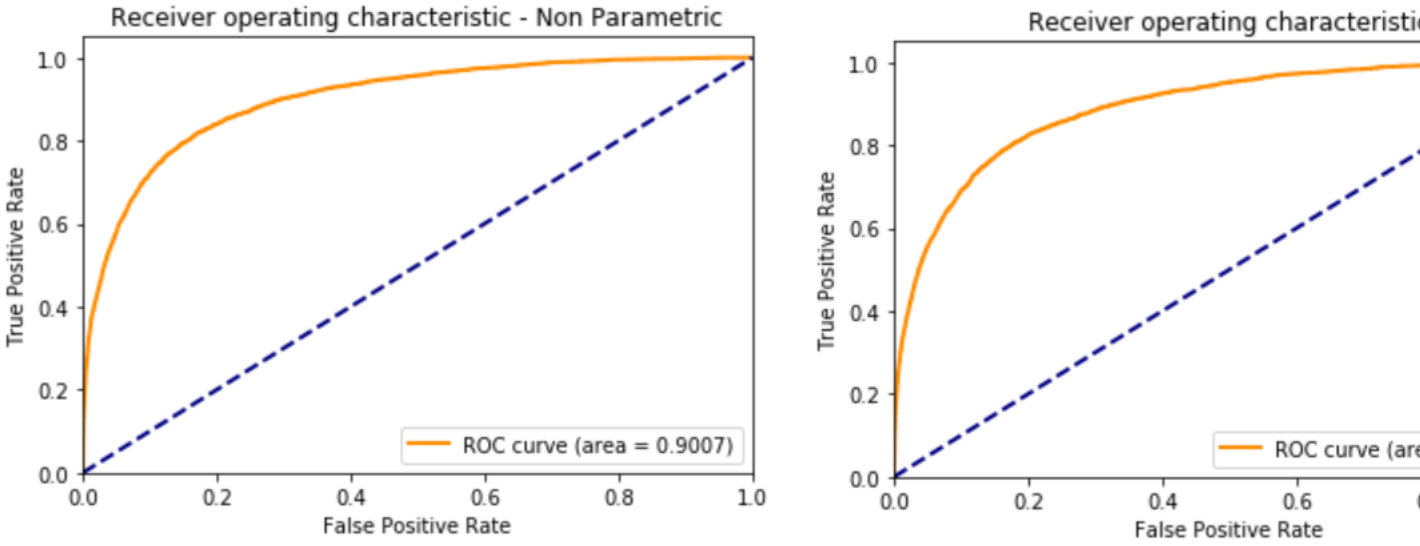
In all 3 comparisons above we concluded that Non Parametric Naive Bayes always wins. This could be due to non-gaussian nature of features present in these datasets.

This experimentation gave us more experience and understanding of setting bandwidth parameters. One thing we realised was we should start with a smaller dataset first and use the optimal parmaters for larger dataset. So here we present the results on 1/10th of santander transactions dataset:

PLEASE NOTE: THE LEFT IS NON PARAMETRIC



We can see how for a smaller dataset we could see similar performance on AUC. With a better understanding of Kernel parameter setting and some belief left in Non Parametric methods we went on to the full dataset in order to check the comparison again and luckily got the following results:



Now we see the AUC for Non Parametric method is higher than Gaussian Naive Bayes. Now we will move on to the conclusions.

5 Conclusions

This project is being done based on the basic supervised machine learning procedure. Since we know "no quality data, no quality mining result", we pay much attention to the data cleaning first. Thanks to Kaggle and data source, the input data we got is well-formatted and with no missing value. We can directly jump to observe the distribution of the data to find some interesting patterns. By plotting some basic distributions to get some initial intuition, we first find out this is an unbalanced data with different target ratio as 9:1 which we can get more understanding about the prediction accuracy later when we train models. By plotting correlation matrix, we find out there's nearly no linear relationship between any features. This leads us to put our most efforts on exploring Naive Bayes related models due to the pre-assumption of NB model is the independency of each features. Further, by observing the distributions of each feature on single target, we

notice that for target 0, most features are shown like normal distribution while distributions of each feature on target 1 are shown far from normal distribution.

- Working on linear classfiers made us realize that logistic classification works the best on this dataset in terms of better AUC (0.86)
- Perceptron works good although most probably gets stuck in the local minima and results in the AUC of (0.82)
- Since Pocket algorithm is proposed to solve the minor weight updated problem in Perceptron and it's most likely to reduce misclassification error. We tried Pocket algorithm and we got accuracy of 0.90.
- Gaussian naive bayes seems to work pretty well as features are near-independent and near-gaussian and gave an AUC of 0.89
- Initial runs on non parametric naive bayes gave abit less results than Gaussian Naive Bayes
  - The paper claimed that Kernel Density Estimated distributions coverges to the true nature of distribution given a large enough dataset.
  - We were able to confirm the same using our initial tests on since the nature of true distribution is gaussian
- Next we tested the non parametric naive bayes on some smaller datasets with not necessarily normal features and got better performance than Naive Bayes
  - The paper does claims that Non parametric will outperform the gaussian naive bayes in case the features are not normal and dataset is large enough
  - We could agree from our results with this claim
- By selecting the bandwidth parameters optimally on a small dataset we would finally get better AUC for Non Parametric Naive Bayes (0.9007) for Non Parametric Naive Bayes which is 0.01 more than Gaussian NB.

We can say that we were better able to model the discrepancies in the normality of features using Non Parametric Estimates and slightly imporve the performance of Naive Bayes Classifier.

6 Individual Tasks

Rong-

- Data cleaning: checking mismatched data format, checking missing value, checking the distribution of target labels
- Implemented pocket algorithm from scratch and got prediction accuracy of 90%.
- Explored Neural Network in sklearn with different parameters. Tuned model with stratified model selection.

Tushar-

- Worked on basic EDA, correlation matrix and normality checks on features.

- Started with linear classifiers such as logistic classifier and perceptron learning
- Implemented gaussian Naive Bayes by exploring on Kaggle
- Explored for non parametric methods to estimate continuous features
- Implemented non parametric methods using sklearn baseestimator class and self implemented prediction concepts
- Worked on the basic pipeline for AUC 10-Fold cross valildation and visualization

References:

[1]Lecture Notes: Chapter 8 Linear Classifiers and Naive Bayes

[2]kaggle dataset: https://www.kaggle.com/c/santander-customer-transaction-prediction/data

[3]paper on non-parametric KDE: http://web.cs.iastate.edu/~honavar/bayes-continuous.pdf

[4]GitHub documentation on KDE: https://github.com/ashkonf/HybridNaiveBayes/blob/master/src/distributions.py

[5]paper on Flexible NB: https://www-sciencedirect-com.ezproxy.neu.edu/science/article/pii/S0888613X08001400

[6]kaggle kernel with implementation of Gaussian NB: https://www.kaggle.com/blackblitz/gaussian-naive-bayes

[7]kaggle kernel with feature comparison plots: https://www.kaggle.com/magf46/eda-naive-bayes-dt-lgbm-polyfeatures

[8]wikipedia page on ROC curve: https://en.wikipedia.org/wiki/Receiver_operating_characteristic