EE 559 Project

Evaluation of Performance of Binary Classifiers for Credit Risk

Project Report
By

Bhavana Ganesh
bhavanag@usc.edu

02 MAY 2017

Abstract

Methods to assess the creditworthiness of loan applicants is necessary for the profitability of the banks or lending sources. With the number of loans being given, for a typical lending institution even a slight improvement in credit scoring model can translate to significant profits. This motivates in development of advanced and sophisticated models to discriminate good and bad borrowers.

This project aims at comparing few of the classification algorithms based on their performance to classify in such a scenario. The German Credit data set from the UCI data set repository is being used for this purpose. The data set classifies people described by a set of attributes as good or bad credit risk. The objective is to develop models that will classify any incoming person into one of these two categories based on their attributes with a reasonable good accuracy value.

Different approaches to the problem is evaluated such as parametric (Naïve Bayes) and non-parametric methods (Support Vector Machines, K-Nearest Neighbors). In addition to this even ensemble of classifiers (Random Forest, Boosting) is evaluated for improvement in performance.

Keywords: Binary Classifier, Naïve Bayes, Support Vector Machine, k-Nearest Neighbors, Ensemble, Random Forest, Boosting, German Credit

Approach

The project evaluates three classification techniques for rating each customer as good or bad credit. The dataset contains 9 features with 6 categorical data and 3 numerical data. The original dataset contains 20 features, but for ease of implementation these 20 features are reduced to 9 features. The categorical data are encoded into binary feature(s) and the numerical data is normalized. Principal component analysis is used for feature dimension reduction so that features that would not contribute significantly to the classification can be eliminated. The various kind of classifiers can be evaluated into two major categories probabilistic and deterministic. In this project one probabilistic method namely Naïve Bayes and two deterministic or non-parametric method namely Support Vector Machine (SVM) and k Nearest Neighbors (KNN) is compared for best performance. In attempt to improve the performance ensemble of classifiers like Boosted Naïve Bayes and Random Forest is implemented.

Preprocessing

The dataset contains a total of 1000 samples. The data set contains missing data in two of the categorical features which is replaced with zeros.

The processing of categorical data consists of two steps, first to assign integer values to different categories and then encode the categorical features into expanded and unique features. The encoding is done using *OneHotEncoder()* from *sklearn* library. The encoding will output a sparse matrix where each column represents one possible value of one feature. This leads to an expansion of feature dimension from six to twenty-six features. Below table shows how the feature values are mapped to different features.

Feature Index	Feature
0	Sex: Male
1	Sex: Female
2	Job: Category 0
3	Job: Category 1
4	Job: Category 2
5	Job: Category 3
6	Housing: Free
7	Housing: Rent
8	Housing: Own
9	Saving Accounts: NAN/Missing data
10	Saving Accounts: Little
11	Saving Accounts: Moderate
12	Saving Accounts: Quite Rich
13	Saving Accounts: Rich
14	Checking Account: NAN/Missing data
15	Checking Account: Little
16	Checking Account: Moderate
17	Checking Account: Rich
18	Purpose: Radio/TV

19	Purpose: Repairs
20	Purpose: Domestic Appliance
21	Purpose: Vacation/Other
22	Purpose: Furniture/Equipment
23	Purpose: Car
24	Purpose: Education
25	Purpose: Business
26	Age
27	Credit Amount
28	Duration

The data is split into training and testing data. This splitting is done before normalizing the numerical data as the test data should be normalized using the mean and variance of the training data. This is so that it is representative of the actual testing process. For this project the dataset is split in the ratio 80:20 for train and test respectively. After which the training data is normalized and using the same parameters the test data is normalized.

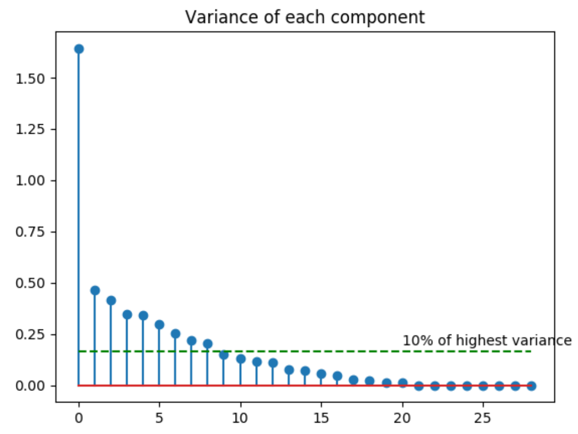
After the pre-processing, we end up with two sets of data with a total of 29 features. The training data has 800 samples and the test data has 200 samples. **All the performance of the classifiers is evaluated with the accuracy values and the F-score.**

Feature reduction/ Feature Selection

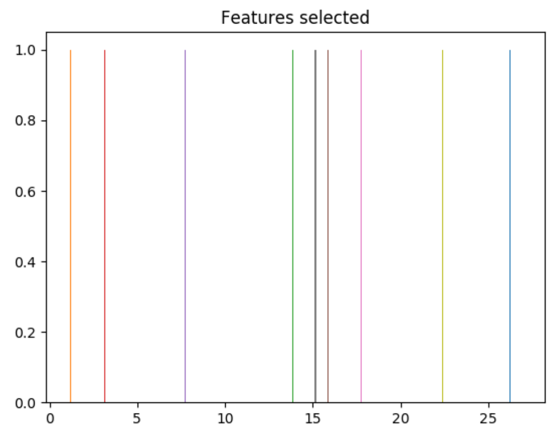
There are 29 features after the encoding and not all of them significantly contribute to the classification process. Hence the number of features can be reduced without hampering the results of the classifiers. Principal Component Analysis (PCA) is used for the process of feature selection.

PCA uses Singular Value Decomposition (SVD) of the data to project it to a lower dimensional space. The procedure returns a set of observations of values that are linearly uncorrelated i.e. the resulting vectors are uncorrelated orthogonal basis set.

The variance of each of the features are shown in the plot below.



The features with variance less than 10% of the highest variance are discarded as they wouldn't contribute very significantly to the performance of the classifier. For this data set we see that there are 9 features above this mark. Hence the feature dimension can be reduced from 29 to 9 without affecting the performance significantly. Below plot shows the features that are selected.



[27. 1. 14. 3. 8. 16. 18. 15. 22.]

From the table that maps the feature number in encoded feature set to the meaningful features the selected features are

> *Amount*, *Sex: Female*, *Checking account: NAN*, *Job Category: 1*, *Housing: own*, *Checking account: Moderate*, *Purpose: Radio/TV*, *Checking account: Little*, *Purpose: Furniture/Equipment*

The features mentioned above are in ascending order of variance. After the PCA is performed on the training set we will have a training dataset of a lower dimension. The PCA function from the *sklearn* library return the dataset projected onto the lower dimensional space. The same features are chosen from the test set as well and the samples are projected onto the lower dimension space.

Classifiers

The methods used for classification are described below along with their results in three categories: Deterministic method (SVM), Probabilistic method (Naïve Bayes and KNN) and finally the ensemble of classifiers (Random Forest, Boosted Random Forest and Boosted Naïve Bayes).

Deterministic Methods – SVM

A support vector machine constructs a hyperplane in a high or infinite-dimensional space which can be used for classification. It maximizes the distance of the hyperplane to the nearest training data. SVM can be constructed as a linear classifier or as a non-linear classifier by mapping the points to higher dimensional space using the kernel trick.

The SVM implementation for the dataset considered here is using a radial basis function kernel.

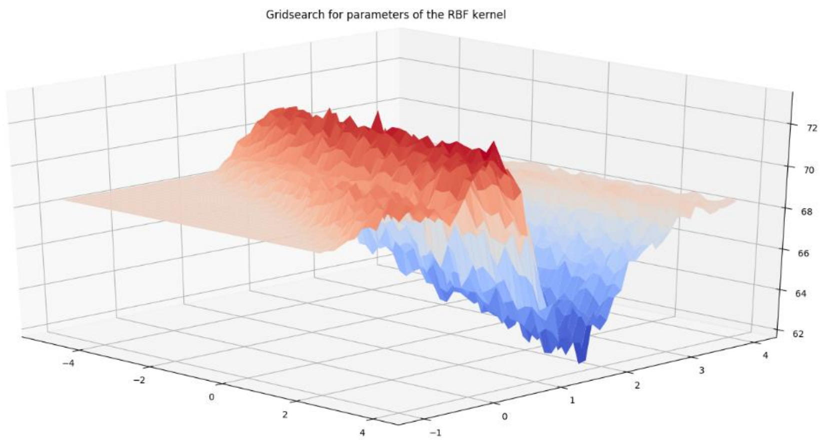
$k(\vec{x_i}, \vec{x_j}) = \exp\left(-\gamma\|\vec{x_i} - \vec{x_j}\|^2\right)$ for $\gamma > 0$

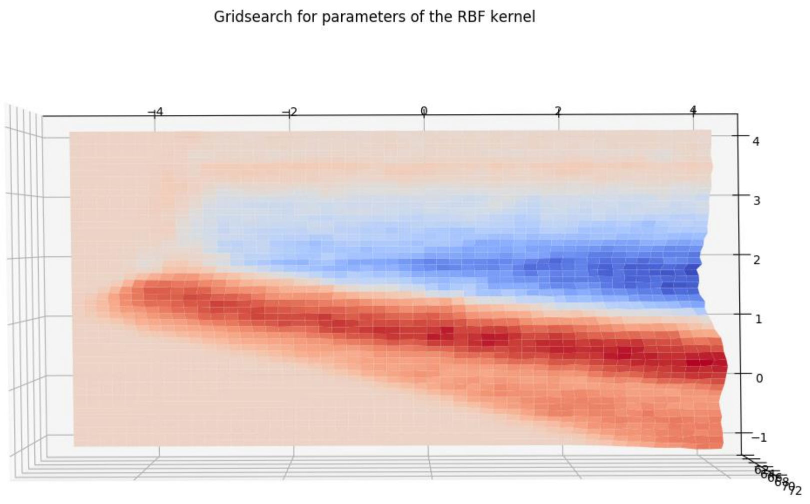
To optimize the two parameters one from the kernel function, γ and the slack variable, grid search is used to sweep through a range of both the parameters and determine the accuracy to pick the best pair for the problem of interest.

γ = [0.00001, 10000]

C = [0.1, 10000]

A total of 50 values are generated in these ranges to form the grid parameters. To perform SVM on the data set *LibSVM* library is used. The training accuracy of the classifier is calculated for every combination of the parameters. A 5 fold cross validation is used to evaluate each of the accuracy values. Also to have a smoother accuracy curve the accuracy values for the same parameters is performed five times and averaged. It is observed that the classifier gives good accuracy for only a certain range of values as observed in the form of peaks in the plot below.





The red region indicates higher accuracy values while the blue region are the values for which SVM gives lower accuracy.

The parameter giving the maximum accuracy in this case was found to be

Ɣ = 0.003727

C = 7906.043

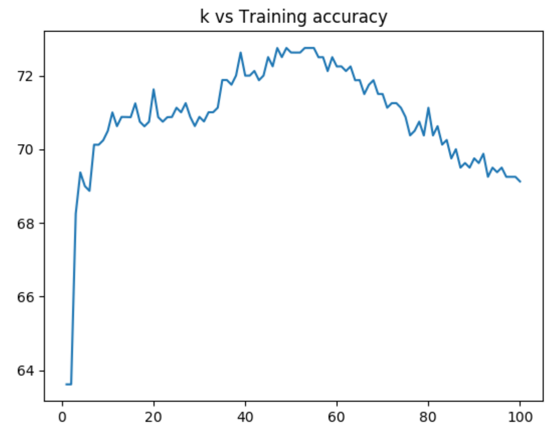
With a training accuracy of 73.25%

With these as parameters the final SVM model is developed and run on the test data to evaluate the performance of the model on unseen and new data.

```
SVM: Training accuracy -  73.25

SVM: Test accuracy -  74.5

SVM: Classification Report

precision    recall  f1-score   support

1      0.82      0.85      0.84       153

2      0.45      0.40      0.43        47

avg   0.74      0.74      0.74       200
```

<u>Deterministic Method – k Nearest Neighbors (k-NN)</u>

k-NN is a type of instance based learning algorithm where the function is only approximated locally. The sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most common among its *k* neighbors, where *k* is a positive integer.
The contribution by its neighbors are weighted for their contribution to the decision.
The implementation for this uses the inverse of the Euclidean distance between the sample and the neighbor as a weight for its contribution. The accuracy of the classifier is determined for a range of *k* values to determine the best *k* value for this data set. A 5 fold cross validation is used to determine the training accuracy.
k = [1,100]
The training accuracy for 100 values of *k* in the range was determined to find the optimal value of *k* for this data set. Also to have a smooth curve as well as more reliable data, for each k value the model is run 20 times and averaged. The trend of accuracy vs *k* is shown in the plot below.



It is observed that the accuracy is highest for *k* in the region around *k* = 45, which gives a training accuracy of 72.5%. A final model using *k* = 45 is developed and used on the new and unseen testing data to evaluate the models performance.

```
KNN: Training accuracy -  72.5007226845

KNN: Test accuracy -  73.0

KNN: Classification Report

          precision   recall  f1-score   support

      1      0.80      0.87     0.83       153

      2      0.39      0.28     0.33        47

    avg      0.70      0.73     0.71       200
```

The k-NN classifier does not perform as well as SVM but the performance of the two are comparable.

Probabilistic Method – Naïve Bayes

Naïve Bayes is based on applying Bayes theorem with the assumption of independence between every pair of features, which seems like a legit assumption for the dataset of consideration.

Bayes' theorem,

$$P(y|x_1, \ldots.., x_n) = \frac{P(y)P(x_1, \ldots., x_n|y)}{P(x_1, \ldots., x_n)}$$

Where *y* indicated the class and $x_1 \ldots . x_n$ *are the samples.*

The sample is assigned the class with the highest probability as described in the equation below,

$$\hat{y} = arg \max_{y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

The Gaussian Naïve Bayes assumes the class conditional density are Gaussian distributions

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(- \frac{(x_i - \mu_y)^2}{2\ {\sigma_y}^2})$$

The mean and the variance are determined using maximum likelihood.

For the implementation of Naïve Bayes for this dataset, a *sklearn* library is used. A 5 fold cross validation is used to determine the training accuracy and the model is used to predict the labels for a set of new values from the test set.

```
Naive Bayes: Training accuracy -  71.0006445564

Naive Bayes: Test accuracy -  71.25

Naive Bayes: Classification Report

          precision   recall  f1-score   support

     1      0.80      0.86     0.83      153

     2      0.42      0.32     0.36       47

   avg      0.71      0.73     0.72      200
```

It is observed that this classifier does not perform better than either SVM or k-NN.

Since the best performing classifier so far is SVM it is a good idea to try to use ensemble of classifiers to evaluate for any improvement in performance. An ensemble uses a divide-and-conquer approach to improve performance. The main principle behind the ensemble is that a group of "weak learners" can come together to form a "strong learner".

For this project two of the ensemble of classifiers are implemented - random forest and boosting or AdaBoost.

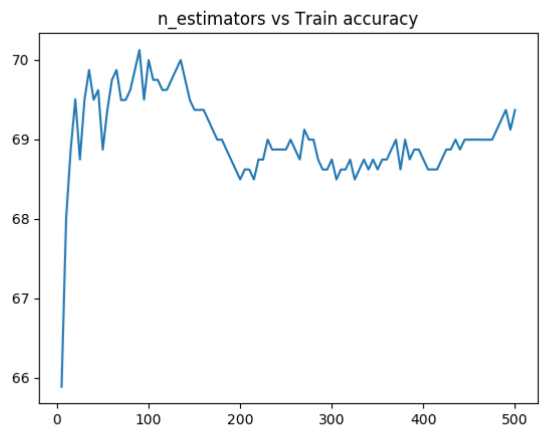
Ensemble of Classifiers – Random Forest

The random forests use a standard machine learning technique called "decision trees" which is the weak learner. The random forest algorithm uses this weak learner and by combining them with the notion of ensemble, random forest is created which is considered as a strong learner. So when the a new input enters the system it runs down all the trees and the result is the averaged to improve accuracy and control over-fitting.

This implementation of random forest is using the *sklearn* library. One of the inputs the model takes is the number of trees to be used for the ensemble. The implementation optimizes this parameter for best performance.

#estimators = [5,500]

The model is evaluated for 100 values of the parameter from the range to find the optimal value. Below is a plot of #estimators vs accuracy.



It is observed that the best performance is observed when the number of trees are around 110. Which gives an accuracy of 69.74% The final model was developed with the parameter value equal to 110 and tested on the unseen test data set.

```
Random Forest: Training accuracy -  69.7467430

Random Forest: Test accuracy -  66.5

Random Forest: Classification Report

          precision   recall  f1-score   support

      1      0.81      0.74      0.77      153

      2      0.33      0.43      0.37       47

    avg      0.70      0.67      0.68      200
```

Sadly, this ensemble classifier failed to perform better than any of the other classifiers mentioned so far. Another ensemble is Adaptive Boosting of the weak learner. As we observed earlier Naïve Bayes did not perform at par with the other classifiers hence can be considered as a weak learner and AdaBoost can be used to improve the performance.

Ensemble of Classifiers – Adaboost Naïve Bayes

The AdaBoost or Adaptive Boosting algorithm combines the weak learning algorithm into a weighted sum that represents the final output of the boosted classifier. Boosting begins with fitting the naïve bayes classifier on the original dataset and then fist additional copies of the classifier on the same dataset but weighs the incorrectly classified instances.

This implementation of AdaBoost Naïve Bayes uses the sklearn library. The model takes the parameter which determines the maximum number of estimators at which the boosting is terminated. To optimize our model for an optimal number of estimators a sweep search for the parameter giving the best accuracy is done.

# estimators = [5,100]

The accuracy values for parameter values in the above range does not seem to change significantly with the number of estimators. Hence a final model using the number of estimators to be 50 is developed and is tested on the unseen test dataset to evaluate the performance

AdaBoost Naive Bayes: Training accuracy - 71.62

AdaBoost Naive Bayes: Test accuracy - 70.75

Adaboost Naive Bayes: Classification Report

	precision	**recall**	**f1-score**	**support**
1	0.81	0.82	0.81	153
2	0.39	0.38	0.39	47
avg	0.71	0.71	0.71	200

There seems to be no significant improvement in performance of the Naïve Bayes Classifier because of boosting.

Inference

It is observed that the non-parametric models performed better than any of the others including the ensemble classifiers as well. The best performing algorithm among the ones evaluated in this project is the SVM with a RBF kernel. This may be because of the infinite dimension of the RBF kernel that the data set gives a good classification accuracy.
As summary, the classification report of the SVM is shown below:

```
SVM: Test accuracy -  74.5

SVM: Classification Report

precision   recall  f1-score   support

1      0.82     0.85     0.84      153

2      0.45     0.40     0.43       47

avg   0.74     0.74     0.74      200
```

This is the best of the results obtained from the study of various binary classifiers for classifying any person as a good or a bad credit.

Implementation Details

All the implementations used in this project is in Python 3.5 but is compatible with Python 2 version as well. Anaconda IDE - Spyder was used for scripting.
Different libraries are used for various purpose throughout the code. The *'pandas'* library is used for data extraction from its original .csv format to a form that can be processed in Python. All the matrices are defined in *'numpy'* format as all the operations are used from that library. The plots including the surface plots generated in this project are done using *'matplotlib'* library. SVM was implemented using the LibSVM library which is an open source library available [2]. Finally the rest of the classifier models are implemented using the *'scikit or sklearn'* library.

Future Work

As inferred that SVM gave the best performance in comparison with the other classifiers, hence this performance may be improved using variations to the classical techniques such as the least square SVM or using genetic algorithm for determining the parameters rather than the traditional grid search.

References

[1] SVM *https://en.wikipedia.org/wiki/Support_vector_machine*

[2] LibSVM *https://www.csie.ntu.edu.tw/~cjlin/libsvm/*

[3] k-NN *https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#Algorithm*

[4] Ensemble Algorithms *http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics*

[5] Adaptive Boosting *https://en.wikipedia.org/wiki/AdaBoost*

[6] Database *https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29*

[7] *http://pandas.pydata.org/pandas-docs/stable/missing_data.html*

[8] *http://stackoverflow.com/questions/21654635/scatter-plots-in-pandas-pyplot-how-to-plot-by-category*

[9] *http://stackoverflow.com/questions/35109113/how-to-normalize-only-certain-columns-in-scikit-learn*

[10] Scikit Learn *http://scikit-learn.org/stable/*

[11] Pandas *http://pandas.pydata.org/*

[12] Numpy *http://www.numpy.org/*

[13] Matplotlib *https://matplotlib.org/*