# Machine Learning Engineer Nanodegree

## Capstone Project

# Credit Approval Prediction

Medehal Priyanka
February 3rd 2019

## I. Definition

### Project overview

In these days the domain of banking is stepping to get engaged with artificial intelligence and machine learning. This project is based on domain of banking. This gives accurate results regarding the classification of credit card approval. It takes the data from the UCI machine learning
https://archive.ics.uci.edu/ml/datasets/Credit+Approval
The aim of this project is to predict whether the credit card can be approved for the individual or not.

### Problem Statement

The main aim of my project is to predict the credit card approval. For doing this I selected the dataset from UCI (https://archive.ics.uci.edu/ml/datasets/Credit+Approval). So my goal is to predict the credit card approvals. Here I am using classification models to find the accuracy of each model and select the best model with high accuracy to predict the approvals. Here the input parameters are training data that we took and the output will be whether to approve the credit card or not.

The tasks involved in it are:

1. Download the data
2. Cleaning the data and removing the null data points, duplicated data rows.
3. Visualizing the data.
4. Split the data and train and test which classifier performs good on the dataset based on the evaluation metric we have chosen.
5. Choose the best classifier that gives the best accuracy scores.

## Metrics

I want to use accuracy score as an evaluation metric for the prediction of credit approval. In the data set the class labels (+,-) are very closely balanced so we can use accuracy score as the evaluation metric. Here I am predicting the accuracy score of the selected models. We will select a model whose accuracy score is greater than all the other models and we treat it as the best.
For finding out the accuracy we have the following formula:
Accuracy Score= TP+TN/TP+FP+FN+TN
True Positives: are the values which are correctly predicted as positives
True Negatives: are the values which are correctly classified as negatives.
False Positives: are the values which are wrongly classified as positives. These are also type-1 errors.
False Negatives: are the values which are wrongly classified as negatives. These are also called as type-2 errors.

# II. Analysis

### Data Exploration

This file concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. This dataset is interesting because there is a good mixture of attributes – Categorical, integer and real valued attributes. There are also few missing values. On a total there are 15 attributes. The total number of instances in the datasets is 690. The characteristics of the dataset are multivariate. Here there are mainly two classes: + (approving credit card) and – (not approving credit card). There are 307 +'s and 383 –'s which are closely balanced.

For the best result we will split the data into training set and testing set. On a whole we will assign 70% of the data to training set and 30% of the data to testing set. The information about the data is a follows

| | 1 | 2 | 7 | 10 | 13 | 14 |
|---|---|---|---|---|---|---|
| **count** | 680.000000 | 680.000000 | 680.000000 | 680.000000 | 680.000000 | 680.000000 |
| **mean** | 31.309824 | 4.797515 | 2.255184 | 2.435294 | 182.864706 | 1024.198529 |
| **std** | 11.718569 | 4.986291 | 3.360643 | 4.889825 | 174.085346 | 5244.642160 |
| **min** | 13.750000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 22.670000 | 1.000000 | 0.165000 | 0.000000 | 70.000000 | 0.000000 |
| **50%** | 28.170000 | 2.855000 | 1.000000 | 0.000000 | 160.000000 | 5.000000 |
| **75%** | 37.562500 | 7.500000 | 2.750000 | 3.000000 | 274.500000 | 400.000000 |
| **max** | 76.750000 | 28.000000 | 28.500000 | 67.000000 | 2000.000000 | 100000.000000 |

Attributes:

A1: b, a.
A2: continuous.
A3: continuous.
A4: u, y, l, t.
A5: g, p, gg.
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.
A7: v, h, bb, j, n, z, dd, ff, o.
A8: continuous.
A9: t, f.
A10: t, f.
A11: continuous.
A12: t, f.
A13: g, p, s.
A14: continuous.
A15: continuous.
A16: +,- (class attribute)

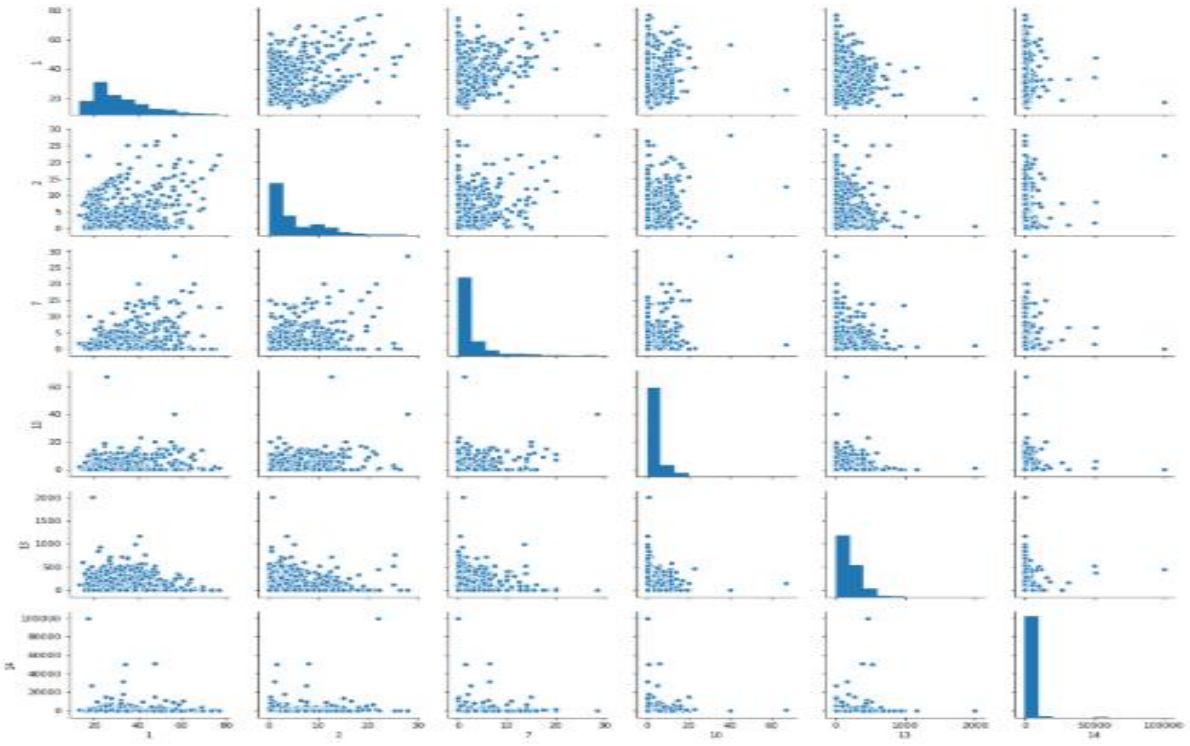
## Exploratory Visualization

### Pair plot:

Pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters. It also helps to form some simple classification models by drawing some simple lines or make linear separation in our dataset.

A Simple 2D plots is used to understand the relationship or pattern between two variables or dimensions in our dataset. A 3D plot will be used for three variables or dimensions. However, what would be do if we have more than 3 dimensions or features in our dataset as we humans do have the capability to visualize more than 3 dimensions? One solution to this problem is pair plots. It is one of the most effective starting tools.

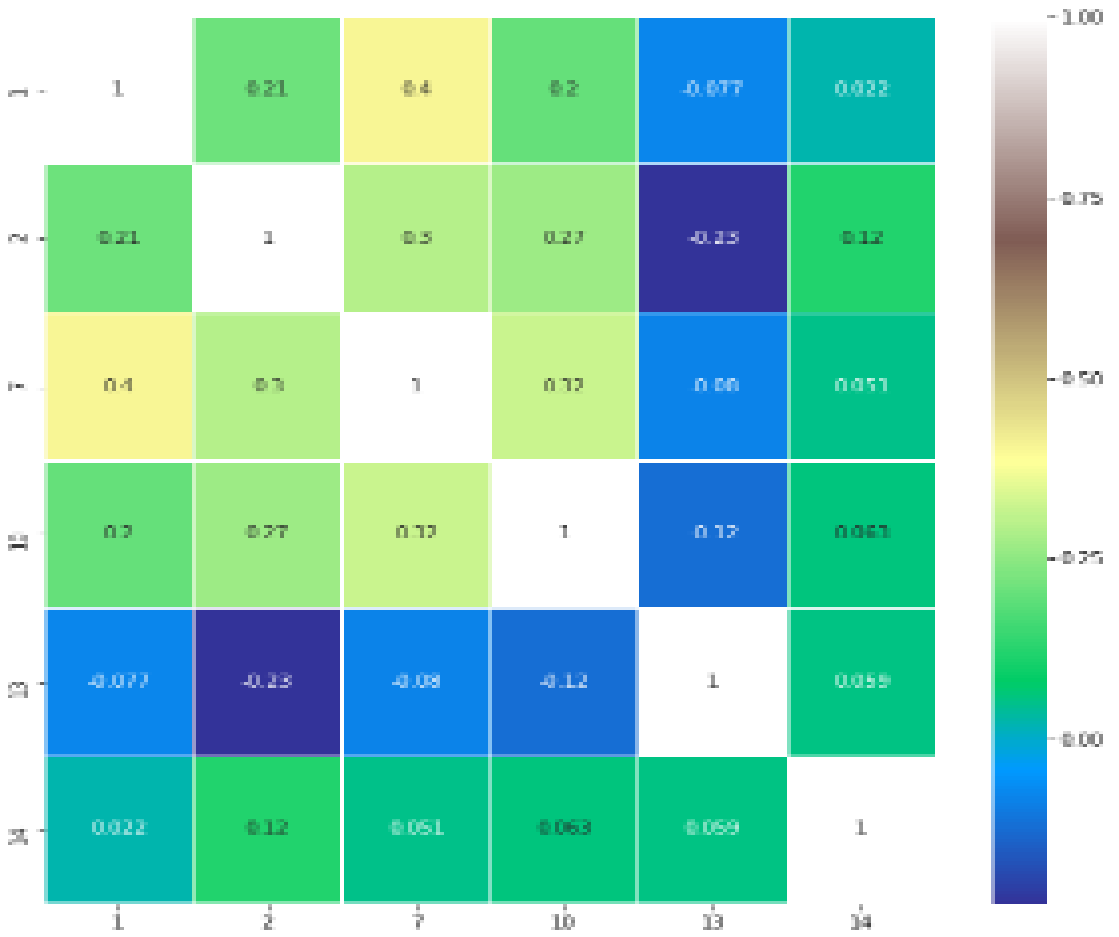
Here we will understand the relation between the numerical attributes
(1, 2, 7, 10, 13, 14)
The pair plot we got by doing so is as follows:



From the above pair plot we cannot clearly visualize the correlation between the numerical attributes so we will go with the heatmap to know clearly about the correlation.

**Heatmap:**

The heatmap is a 2-D representation of data in which values are represented by colours. A simple heatmap provides the immediate visual summary of information. More elaborate heatmaps allow the user to understand complex data.




From the above heatmap we will represent the correlation of the numerical attributes in the data. The above heatmap clearly visualize the correlation between the numerical attributes. We can say that there is no much correlation between them.

## Algorithms and techniques:

**Decision Trees:** Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predict the value.

Some advantages of decision trees are:

Simple to understand and to interpret. Trees can be visualized. Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of

variable. See algorithms for more information. Able to handle multi-output problems. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Parameters:

Class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False

**Random Forest**:

Tree models are known to be high variance, low bias models. In consequence, they are prone to overfit the training data. This is catchy if we recapitulate what a tree model does if we do not prune it or introduce early stopping criteria like a minimum number of instances per leaf node. Well, it tries to split

the data along the features until the instances are pure regarding the value of the target feature, there are no data left, or there are no features left to spit the dataset on. If one of the above holds true, we grow a leaf node. The consequence is that the tree model is grown to the maximal depth and therewith tries to reshape the training data as precise as possible which can easily lead to over fitting. Another drawback of classical tree models like the (ID3 or CART) is that they are relatively unstable. This instability can lead to the situation that a small change in the composition of the dataset leads to a completely different tree model.

Parameters:

Class sklearn.ensemble.RandomForestClassifier (n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None

Advantages:

1.Reduction in over fitting: by averaging several trees, there is a significantly lower risk of over fitting.
2.Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn't perform well because of the relationship between the train and test data.

Disadvantages :

1. It takes more time to train samples.

**Logistic Regression**:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Advantages:

Because of its efficient and straightforward nature, doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist. Also, it doesn't require scaling of features. Logistic regression provides a probability score for observations.

Disadvantages:

Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to over fitting. Also, can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

Parameters:

Class sklearn.linear_model.LogisticRegression (penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)

The application is classification oriented . So, techniques that are used are taken from Classification techniques.

**Naive bayes:**

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

contribution to the outcome.

Advantages:

- Very simple, easy to implement and fast.
- If the NB conditional independence assumption holds, then it will converge quicker than discriminative models like logistic regression.
- Even if the NB assumption doesn't hold, it works great in practice.
- Need less training data.
- Highly scalable. It scales linearly with the number of predictors and data points.
- Can be used for both binary and multiclass classification problems.
- Can make probabilistic predictions.
- Handles continuous and discrete data.
- Not sensitive to irrelevant features.

Disadvantages:

1 .The first disadvantage is that the Naive Bayes classifier makes a very strong assumption on the shape of your data distribution, i.e. any two features are independent given the output class. Due to this, the result can be potentially very bad - hence, a “naive” classifier. This is not as terrible as people generally think, because the NB classifier can be optimal even if the assumption is violated, and its results can be good even in the case of sub-optimality.

2. Another problem happens due to data scarcity. For any possible value of a feature, you need to estimate a likelihood value by a frequents approach. This can result in probabilities going towards 0 or 1, which in turn leads to numerical instabilities and worse results. In this case, you need to smooth in some way your probabilities, or to impose some prior on your data, however you may argue that the resulting classifier is not naive anymore.

3. A third problem arises for continuous features. It is common to use a binning procedure to make them discrete, but if you are not careful you can throw away a lot of information.

Parameters:

alpha : float, optional (default=1.0), fit_prior : boolean, optional (default=True), class_prior : array-like, size (n_classes,), optional (default=None), X : {array-like, sparse matrix}, shape = [n_samples, n_features], y : array-like, shape = [n_samples], sample_weight : array-like, shape = [n_samples], (default=None), deep : boolean, optional, classes : array-like, shape = [n_classes] (default=None)

**KNN:**

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output

2. Calculation time

3. Predictive Power

Advantages:
The cost of the learning process is zero No assumptions about the characteristics of the concepts to learn have to be done Complex concepts can be learned by local approximation using simple procedures

Disadvantages:
The model can not be interpreted (there is no description of the learned concepts) It is computationally expensive to find the k nearest neighbours when the dataset is very large Performance depends on the number of dimensions that we have (curse of dimensionality ) ⟹ Attribute Selection

Parameters:
*class* sklearn.neighbors.KNeighborsClassifier(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*, ***kwargs*)

**Support vector machine (svm):**

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.
To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Advantages:
SVM's are very good when we have no idea on the data. Works well with even unstructured and semi structured data like text, Images and trees. The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem. Unlike in neural networks, SVM is not solved for local optima. It scales relatively well to high dimensional data. SVM models have generalization in practice; the risk of overfitting is less in SVM.

Disadvantages:

Choosing a good kernel function is not easy. Long training time for large datasets. Difficult to understand and interpret the final model, variable weights and individual impact. Since the final model is not so easy to see, we can not do small calibrations to the model hence it's tough to incorporate our business logic.

Parameters:
*class* sklearn.svm.**SVC**(*C=1.0*, kernel='rbf', degree=*3*, gamma='auto_deprecated', *coef0=0.0*, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=*'ovr'*, random_state=*None*) ¶

**Benchmark Model:**

For this problem we will choose Naive bayes classification model as the benchmark model.
By applying this Gaussian Naive bayes we achieved an accuracy of 84.8039%
Now we will try and achieve better accuracy than this model by using the above mentioned classification models.

## III. Methodology

**Data Pre-processing:**
In this step of data pre-processing we will pre-process the data. We will read the data from dataset and replace the null values.
We will know the information about all the data types. We will know the mean standard deviation and various metrics regarding to the numerical data. To know the correlation between the numerical attributes we will plot the graphs to visualize the data (this context we will use pair plot and heatmap).
We will now remove the true outliers. We will consider the data which is three standard deviations away from the mean as the outliers and remove them.
Now we need to perform a One Hot Encoding of the categorical variables to prepare the data for classification. We can do this easily by using OneHotEncoder from the sklearn.preprocessing module or simple call get_dummies on a pandas data frame. For simplicity, we will use the later approach.
After that the whole dataset is divided into training and testing data using train_test_split from sklearn.model_selection.

Standardization of a dataset is common dataset for many machine learning estimators. They might behave badly if the individual features do not more or less look like standard normally distributed data. So we use StandardScalar()

**Implementation:**

Here we will write a function train_model which helps us to calculate the accuracy for every classification model.
We will find out the not training and testing accuracy for the chosen models.
We will now choose the best model out of svm, knn, logistic regression, random forest, decision tree whose test accuracy will be more than that of the benchmark model.

Knn: accuracy is 83.8235%
Decision tree: accuracy is 83.3333%
Logistic regression: accuracy is 88.2353%
Svm: accuracy is 87.2549%
Random forest: accuracy is 88.7255%
Among the above reports random forest seems to be performing well.
While building the Random Forest Classifier I faced a complication while tuning the parameters. Firstly we should come to a conclusion about the parameters we are going to tune. Then by varying the range of values of the chosen parameter we should be able to get the parameter value which gives us the better accuracy than the untuned value. So it is complicated to choose the parameter that is to be tuned in this case n_estimator and criterion.

**Refinement**

I found out random forest as the best classifier out of the chosen classifiers. Now we will perform tuning of random forest classifier in order to achieve the better accuracy. We will assign n_estimators=[40,50] and criterion: ['gini', 'entropy'] now we will find out the accuracy. Here we will use GridSearchCV()
The accuracy of tuned random forest is 88.73%, which is better than the untuned random forest.

## IV. Result

## Model evaluation and validation

The final model we have chosen is tuned random forest which gave us more accuracy that is 88.73%. In order to achieve this accuracy we assigned

n_estimator=[40,50] and criterion: ['gini', 'entropy'] but without using the n_estimators we achieved the accuracy of only 88.7255% which is a bit less than the tuned value. Here we can say that the solution is reasonable because we are getting much less accuracy while using other models.

The final model that is tuned random forest has been tested with various inputs to evaluate whether the model generalises well. This model is also robust enough for the given problem. We tested the random forest for various random states and we can clearly see that there is no big change in the accuracy. We have tested for the random states3000, 3100, 3200 and we have achieved the mean which is close to the accuracy of random state =3000. So from that we can say that Small changes in the training data will not affect the results greatly. So the results found from this model can be trusted.

**Justification**

My final model's solution is better than the benchmark model.

| | Tuned random forest | Benchmark model |
|---|---|---|
| Accuracy | 88.73% | 84.8039% |

From the above we can conclude that the results for the final model are stronger than the benchmark model.

Hence we can say that tuned random forest provides the significant to solve the problem of credit card approval.

## V. Conclusion

**Free-form Visualization**

After studying the data and applying all the classification models we can visualize the following

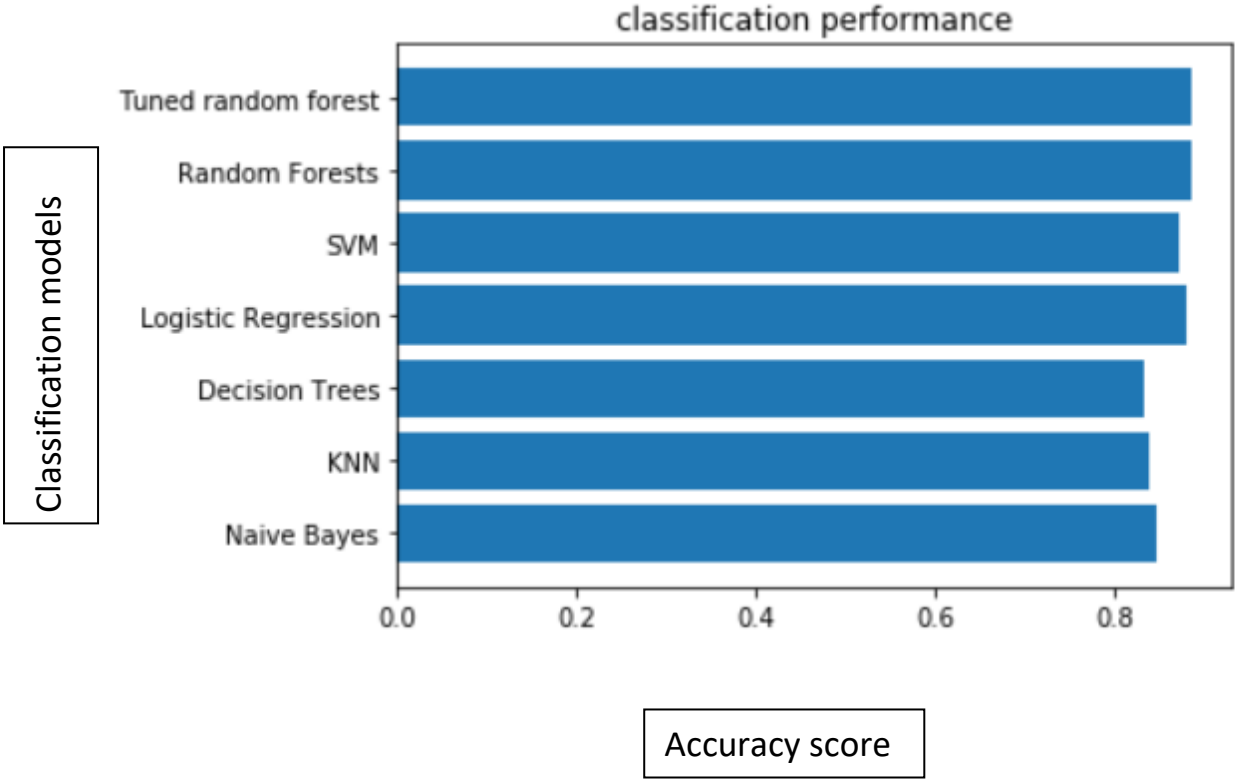
In [231]:
```
# create a dataframe from accuracy results
summary = pd.DataFrame({'accuracy':accuracy}, index=classifiers)
summary
```

Out[231]:

| | accuracy |
|---|---|
| Naive Bayes | 0.848039 |
| KNN | 0.838235 |
| Decision Trees | 0.833333 |
| Logistic Regression | 0.882353 |
| SVM | 0.872549 |
| Random Forests | 0.887255 |

Plotting the above in the form of a bar graph taking accuracy score on x-axis and classification model names on y-axis we will get the following bar graph.

Plot between accuracy and classifier models




From the above bar graph we can visualize that tuned random forest will give us the good quality of output than the other.
Here accuracy score is the important metric through with we will say that the tuned random forest will give the good quality of output.

**Reflection**:

1. I have learnt how to visualize and understand the data.
2. I have learnt that the data cleaning place a very vital role in data analytics.
3. Removing the data features which are not necessary in evaluating model is very important.
4. I got to know how to use the best technique for the data using appropriate ways

5. I got to know how to tune the parameters in order to achieve the best score.
6. On a whole I learnt how to graph a dataset and applying cleaning techniques on it and to fit the best techniques to get best score.

**Improvement:**

The process which i have followed can be improved to classify not only credit approvals but also the other card approvals like debit card and also for the approving of loans by banks. This model is a part of my researching in banking sector. This application can be taken to next level with the many more applications. This model also helps us to reduce the human errors while approving the credit cards to the customers and also it will help in reducing of the corruption in banking sector.