Data Mining the Lending Club Loan Dataset

Kevalkumar Vegad

2020

Contents

[Abstract: 2](#_Toc40182930)

[Part1: Data Dictionary 3](#_Toc40182931)

[Part2: Data Pre-Processing 5](#_Toc40182932)

[2.1 Eliminate columns with missing data : 5](#_Toc40182933)

[2.2 Duplicate values: 6](#_Toc40182934)

[2.3 Remove column which has identical value: 6](#_Toc40182935)

[2.4 Bifurcate data-frame into numerical and categorical: 6](#_Toc40182936)

[2.5 Label Encoding: 7](#_Toc40182937)

[2.6 Split data into training set and test set 8](#_Toc40182938)

[2.7 Feature Scaling: 8](#_Toc40182939)

[Part3: Applying Different Algorithms: 9](#_Toc40182940)

[3.1 K-Nearest Neighbors (KNN) 9](#_Toc40182941)

[3.2 Support-vector machine (SVM): 10](#_Toc40182942)

[3.3 Decision Tree ALGO: 11](#_Toc40182943)

[3.4 RandomForest: 12](#_Toc40182944)

[3.5 Naive Bayes: 13](#_Toc40182945)

[Part4: Supervised Clustering: 15](#_Toc40182946)

[Conclusion: 17](#_Toc40182947)

[Reference: 18](#_Toc40182948)

# Abstract:

In KDD, fraud detection is usually framed as classification problems —predicting a discrete class label output given a data observation. Examples of classification problems that can be thought of are Spam Detectors, Recommender Systems and Loan Default Prediction.

The data used for this coursework is the Lending Club Loan Data which is publicly available here: https://www.lendingclub.com/info/download-data.action. The information contained in this database relates to the loans applications Lending Club approved in Quarter 2 of the year 2017. The file is a matrix of about 105451 observations and 135 variables (memory usage: 110.2+ MB). In the given data file, features include credit scores, number of finance inquiries, address including zip codes, annual income, number of collections over last 12 months, loan description, debt-to-income ratio, employment length, employee title, fico ranges, loan amount, loan grade, homeownership status, interest rate, last payment amount and date, current loan status, number of times creditors asked for payment, months since last delinquency, remaining principal, interest received to date, late fees received to date, and principal received to date among others.

The main objective of this course work is to predict the loan status by creating models that have enough intelligence in order to properly classify transactions as either Current, Late, Fully Paid etc. from the given fields and accurately identify the loan defaulters.

# Part1: Data Dictionary

**Data Dictionary after basic pre-processing**

|  |  |
| --- | --- |
| **Column name** | **Description** |
| loan\_amnt | The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value. |
| installment | The monthly payment owed by the borrower if the loan originates. |
| annual\_inc | The self-reported annual income provided by the borrower during registration. |
| dti | A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower’s self-reported monthly income. |
| mths\_since\_last\_delinq | The number of months since the borrower's last delinquency. |
| revol\_bal | Total credit revolving balance |
| total\_acc | The total number of credit lines currently in the borrower's credit file |
| total\_bal\_il | Total current balance of all installment accounts |
| max\_bal\_bc | Maximum current balance owed on all revolving accounts |
| total\_rev\_hi\_lim | Total revolving high credit/credit limit |
| bc\_open\_to\_buy | Total open to buy on revolving bankcards. |
| mo\_sin\_old\_il\_acct | Months since oldest bank installment account opened |
| mo\_sin\_rcnt\_rev\_tl\_op | Months since most recent revolving account opened |
| percent\_bc\_gt\_75 | Percentage of all bankcard accounts > 75% of limit. |
| total\_bal\_ex\_mort | Total credit balance excluding mortgage |
| total\_il\_high\_credit\_limit | Total installment high credit/credit limit |
| emp\_title | The job title supplied by the Borrower when applying for the loan.\* |
| home\_ownership | The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER |
| loan\_status | Current status of the loan |
| title | The loan title provided by the borrower |
| addr\_state | The state provided by the borrower in the loan application |
| out\_prncp | Remaining outstanding principal for total amount funded |
| total\_pymnt | Payments received to date for total amount funded |
| total\_rec\_prncp | Principal received to date |
| total\_rec\_int | Interest received to date |
| last\_pymnt\_amnt | Last total payment amount received |
| tot\_cur\_bal | Total current balance of all accounts |
| mths\_since\_rcnt\_il | Months since most recent installment accounts opened |
| il\_util | Ratio of total current balance to high credit/credit limit on all install acct |
| all\_util | Balance to credit limit on all trades |
| avg\_cur\_bal | Average current balance of all accounts |
| bc\_util | Ratio of total current balance to high credit/credit limit for all bankcard accounts. |
| mo\_sin\_old\_rev\_tl\_op | Months since oldest revolving account opened |
| mths\_since\_recent\_bc | Months since most recent bankcard account opened. |
| tot\_hi\_cred\_lim | Total high credit/credit limit |
| total\_bc\_limit | Total bankcard high credit/credit limit |
| **Column name** | **Description** |
| grade | LC assigned loan grade |
| emp\_length | Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. |
| verification\_status | Indicates if income was verified by LC, not verified, or if the income source was verified |
| purpose | A category provided by the borrower for the loan request. |
| zip\_code | The first 3 numbers of the zip code provided by the borrower in the loan application. |
| earliest\_cr\_line | Earliest credit line at time of application for the secondary applicant |

**Final Data Dictionary after Backward Elimination:**

|  |  |
| --- | --- |
| **Column name** | **Description** |
| loan\_amnt | The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value. |
| installment | The monthly payment owed by the borrower if the loan originates. |
| revol\_bal | Total credit revolving balance |
| total\_acc | The total number of credit lines currently in the borrower's credit file |
| out\_prncp | Remaining outstanding principal for total amount funded |
| total\_pymnt | Payments received to date for total amount funded |
| total\_rec\_prncp | Principal received to date |
| total\_rec\_int | Interest received to date |
| last\_pymnt\_amnt | Last total payment amount received |
| mths\_since\_rcnt\_il | Last total payment amount received |
| max\_bal\_bc | Maximum current balance owed on all revolving accounts |
| avg\_cur\_bal | Average current balance of all accounts |
| total\_il\_high\_credit\_limit | Total installment high credit/credit limit |
| grade | LC assigned loan grade |
| verification\_status | Indicates if income was verified by LC, not verified, or if the income source was verified |
| purpose | A category provided by the borrower for the loan request. |

**Code for backward elimination[1]:**

X1 = df.loc[:,df.columns != 'loan\_status']

y1 = df['loan\_status']

#Adding constant column of ones, mandatory for sm.OLS model

import statsmodels.api as sm

X\_1 = sm.add\_constant(X1)

#Fitting sm.OLS model

model = sm.OLS(y1,X\_1).fit()

model.pvalues

#Backward Elimination

cols = list(X1.columns)

pmax = 1

while (len(cols)>0):

p= []

X\_1 = X1[cols]

X\_1 = sm.add\_constant(X\_1)

model = sm.OLS(y1,X\_1).fit()

p = pd.Series(model.pvalues.values[1:],index = cols)

pmax = max(p)

feature\_with\_p\_max = p.idxmax()

if(pmax>0.05):

cols.remove(feature\_with\_p\_max)

else:

break

selected\_features\_BE = cols

print(selected\_features\_BE)

# ['loan\_amnt', 'installment', 'revol\_bal', 'total\_acc', 'out\_prncp', 'total\_pymnt', 'total\_rec\_prncp',

# 'total\_rec\_int', 'last\_pymnt\_amnt', 'mths\_since\_rcnt\_il', 'il\_util', 'max\_bal\_bc', 'avg\_cur\_bal',

# 'bc\_util', 'mo\_sin\_rcnt\_rev\_tl\_op', 'percent\_bc\_gt\_75', 'tot\_hi\_cred\_lim', 'total\_bal\_ex\_mort',

# 'total\_il\_high\_credit\_limit', 'grade', 'verification\_status', 'purpose']

# Part2: Data Pre-Processing

The main challenge when it comes to modeling fraud detection as a classification problem comes from the fact that in real world data, the majority of transactions is not fraudulent. But the data received is quite imbalanced which requires lot of pre-processing.

There is lots of missing value in the data. It may have happened during data collection, or maybe due to some data validation rule, but regardless missing values must be taken into consideration.

## Eliminate columns with missing data :

Simple and sometimes effective strategy. If a column has mostly missing values, then that column itself can be eliminated by setting some percentage value, for instance 60%. So here, we have removed column which has empty value more than 60% which is done by following line of code.

*dataset1.loc[:, dataset1.isin([' ','NULL',0]).mean() < 0.6]*

*#Count the Null Columns*

*null\_columns = dataset2.columns[dataset2.isnull().any()]*

*sum\_num\_missing = dataset2[null\_columns].isnull().sum()*

*len(dataset2)*

*percentage\_missing\_values = (sum\_num\_missing/len(dataset2))\*100*

*print(percentage\_missing\_values>60)*

*percentage\_missing\_values[percentage\_missing\_values>60]*

*need\_to\_delete = percentage\_missing\_values>60*

*need\_to\_delete[need\_to\_delete==True]*

*#deleting the column which has null values greater than 60%*

*dataset4 = dataset2.drop(['mths\_since\_last\_record', 'mths\_since\_last\_major\_derog', 'annual\_inc\_joint','dti\_joint','verification\_status\_joint','mths\_since\_recent\_bc\_dlq','mths\_since\_recent\_revol\_delinq','revol\_bal\_joint','sec\_app\_earliest\_cr\_line','sec\_app\_inq\_last\_6mths','sec\_app\_mort\_acc','sec\_app\_open\_acc','sec\_app\_revol\_util','sec\_app\_open\_il\_6m','sec\_app\_num\_rev\_accts','sec\_app\_chargeoff\_within\_12\_mths','sec\_app\_collections\_12\_mths\_ex\_med','sec\_app\_mths\_since\_last\_major\_derog'],axis=1)*

## Duplicate values:

This data has data objects which are duplicates of one another which also need to be removed. Duplicate rows can be found using duplicated() function. In our case, there are total four rows which has duplicate rows so we need to delete these rows using below code.

*#Now need to find rows with duplicate values*

*duplicate\_rows\_df = dataset4[dataset4.duplicated()]*

*dataset4.drop\_duplicates().shape*

*dataset4 = dataset4.drop([105451, 105452, 105453, 105454])*

Do the same thing for column which has duplicate values.

*#Now need to find rows with duplicate values*

*dataset4.T.drop\_duplicates().T*

*dataset4 = dataset4.drop(['funded\_amnt', 'funded\_amnt\_inv'], axis=1) # drope these column with same values*

*dataset4 = dataset4.drop(['out\_prncp\_inv'], axis=1)*

*dataset4 = dataset4.drop(['total\_pymnt\_inv'], axis=1)*

## Remove column which has identical value:

Some of the columns in data-frame has identical values. Because of this identical values, these columns won’t make any difference while training this dataset. Hence, we need to remove the below mentioned columns.

1. issue\_d
2. pymnt\_plan
3. next\_pymnt\_d
4. last\_pymnt\_d
5. last\_credit\_pull\_d
6. policy\_code
7. application\_type
8. term
9. initial\_list\_status

## Bifurcate data-frame into numerical and categorical:

To deal with missing value, we need to divide data-frame into numerical and categorical data-frame. We can do that by dropping all the categorical columns and save the data-frame as numeric data-frame.

*dataset4\_numeric = dataset4.drop(['grade','emp\_title','emp\_length','home\_ownership','verification\_status','loan\_status',*

*'purpose','title','zip\_code','addr\_state','earliest\_cr\_line'],axis = 1)*

*dataset4\_categorical = dataset4[['grade','emp\_title','emp\_length','home\_ownership','verification\_status','loan\_status',*

*'purpose','title','zip\_code','addr\_state','earliest\_cr\_line']]*

Now, we can replace missing values with mean value for numerical columns. Finally drop columns which has standard deviation less than 10. The pyhton code for this task is as below.

*drop column which has standard deviation less than 10*

*dataset4\_numeric = dataset4\_numeric.drop(dataset4\_numeric.std()[dataset4\_numeric.std() < threshold].index.values, axis=1)*

*# dealing with the missing values and replacing it by mean*

*from sklearn.impute import SimpleImputer*

*imp = SimpleImputer(missing\_values=np.nan, strategy='mean')*

*imp.fit\_transform(dataset4\_numeric)*

*dataset4\_numeric.fillna(dataset4\_numeric.mean(), inplace=True)*

Dealing with the missing values is slightly different for categorical columns data-frame. Here, we need to replace missing categorical data with respected column’s mod (here, mode of a set of data values is the value that appears most often) values. As the total number of null values are 0 for categorical data-frame, we do not require to deal with missing values here.

*# dealing with the missing values and replacing it by mod for categorical data*

*print(dataset4\_categorical.isnull().values.sum()) #output: 0*

## Label Encoding:

We need to convert all the categorical features into some kind of numbers as machine learning algorithm cannot process string values. We can use label encoding to convert string values into numerical values. Next task is to perform label encoding by using pandas to convert a column into a category, then use those category values for your label encoding. At the end, assign the encoded variable to a new column using the cat.codes accessor.

*#change column type into category*

*for col in ['grade', 'emp\_title', 'emp\_length', 'home\_ownership', 'verification\_status', 'loan\_status','purpose','title','zip\_code','addr\_state',*

*'earliest\_cr\_line']:*

*dataset4\_categorical[col] = dataset4\_categorical[col].astype('category')*

*dataset4\_categorical['grade'] = dataset4\_categorical['grade'].cat.codes*

*dataset4\_categorical['emp\_title'] = dataset4\_categorical['emp\_title'].cat.codes*

*dataset4.emp\_length.unique()*

*dataset4\_categorical['emp\_length'] = dataset4\_categorical['emp\_length'].cat.codes*

*dataset4\_categorical['home\_ownership'] = dataset4\_categorical['home\_ownership'].cat.codes*

*dataset4\_categorical['verification\_status'] = dataset4\_categorical['verification\_status'].cat.codes*

*dataset4\_categorical['loan\_status'] = dataset4\_categorical['loan\_status'].cat.codes*

*dataset4\_categorical['purpose'] = dataset4\_categorical['purpose'].cat.codes*

*dataset4\_categorical['title'] = dataset4\_categorical['title'].cat.codes*

*dataset4\_categorical['zip\_code'] = dataset4\_categorical['zip\_code'].cat.codes*

*dataset4\_categorical['addr\_state'] = dataset4\_categorical['addr\_state'].cat.codes*

*dataset4\_categorical['earliest\_cr\_line'] = dataset4\_categorical['earliest\_cr\_line'].cat.codes*

{0: 'Charged Off',

1: 'Current',

2: 'Fully Paid',

3: 'In Grace Period',

4: 'Late (16-30 days)',

5: 'Late (31-120 days)'}

Now, we just need to simply merge both numerical and categorical data-frame into single data-frame.

*df = pd.concat([dataset4\_numeric, dataset4\_categorical], axis=1)*

## Split data into training set and test set

Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when we separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to ensure that the testing and training sets are similar. By using similar data for training and testing, we can minimize the effects of data discrepancies and better understand the characteristics of the model. scikit-learn provides train\_test\_split library to split data into training set and test set.

X = df.loc[:,df.columns != 'loan\_status'].values

y = df['loan\_status'].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

## Feature Scaling:

Our dataset contains variables that are different in scale. For instance, num\_sats has values like 4,9,16 and total\_bc\_limit has values like 50,000, 60,000, 100,000 et al. In machine learning, algorithm works on Euclidian distance which is distance between two points calculated by using square of sqrt(4^2 16^2) and sqrt of (50,000^2-100000^2). In this case, columns which has comparatively small values have overpower by columns which have higher values. This will cause problem in training values and predicting values for different algorithms. To overcome this issue, feature scaling is carried out.

Here, in this project we have used ‘StandardScaler’ to perform feature scalling. It transforms the data in such a manner that it has mean as 0 and standard deviation as 1. In short, it standardizes the data.

Apply feature scaling techniques to dataset:

*from sklearn.preprocessing import StandardScaler*

*sc\_X = StandardScaler()*

*X\_train = sc\_X.fit\_transform(X\_train)*

*X\_test = sc\_X.transform(X\_test)*

Here, we have also applied backward elimination techniques to find which columns are vital in predating loan status. Python code is shown in section 1(part1:data dictionary).

# Part3: Applying Different Algorithms:

## K-Nearest Neighbors (KNN)

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[2] In both cases, the input consists of the k closest training examples in the feature space; the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The implementation of the same is describe below.

# Fitting K-NN to the Training set

*from sklearn.neighbors import KNeighborsClassifier*

*classifier\_KNN = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)*

*classifier\_KNN.fit(X\_train, y\_train)*

*# Predicting the Test set results*

*y\_pred\_KNN = classifier\_KNN.predict(X\_test)*

*# Making the Confusion Matrix*

*from sklearn.metrics import confusion\_matrix*

*cm\_KNN = confusion\_matrix(y\_test, y\_pred\_KNN)*

*#IMPLEMENTING CROSS-VALIDATION using cross\_val\_score() function*

*from sklearn.model\_selection import cross\_val\_score*

*cross\_val\_score(classifier\_KNN, X\_train, y\_train, cv=3, scoring="accuracy")*

*#Out[31]: array([0.97411095, 0.97368421, ]) approx 97% of accuracy*

**Confusion matrix:**

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score, classification\_report, confusion\_matrix

confusion\_matrix(y\_test,y\_pred\_KNN)

# Out[14]:

# array([[ 0, 9, 0, 0, 0, 0],

# [ 0, 19943, 21, 0, 0, 0],

# [ 0, 174, 601, 0, 0, 0],

# [ 0, 191, 0, 0, 0, 0],

# [ 0, 55, 0, 0, 0, 0],

# [ 0, 96, 1, 0, 0, 0]], dtype=int64)

from sklearn.metrics import classification\_report

# print(classification\_report(y\_test,y\_pred\_KNN))

# **precision recall f1-score support**

# 0 0.00 0.00 0.00 9

# 1 0.97 1.00 0.99 19964

# 2 0.96 0.78 0.86 775

# 3 0.00 0.00 0.00 191

# 4 0.00 0.00 0.00 55

# 5 0.00 0.00 0.00 97

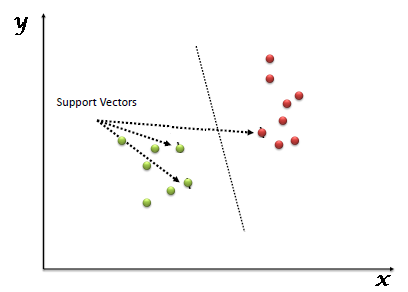
# accuracy 0.97 21091

# macro avg 0.32 0.30 0.31 21091

# weighted avg 0.96 0.97 0.97 21091

## Support-vector machine (SVM):

In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate[3]. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).

****

*# Fitting SVM to the Training set*

*from sklearn.svm import SVC*

*classifier\_SVM = SVC(kernel = 'linear', random\_state = 0)*

*classifier\_SVM.fit(X\_train, y\_train)*

*# Predicting the Test set results*

*y\_pred\_SVM = classifier\_SVM.predict(X\_test)*

*#IMPLEMENTING CROSS-VALIDATION using cross\_val\_score() function*

*from sklearn.model\_selection import cross\_val\_score*

*cross\_val\_score(classifier\_SVM, X\_train, y\_train, cv=3, scoring="accuracy")*

*#Out[53]: array([0.98374822, 0.98385491, 0.98406828])*

*from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score, classification\_report, confusion\_matrix*

***confusion\_matrix(y\_test,y\_pred\_SVM)***

*# Out[45]:*

*# array([[ 5, 4, 0, 0, 0, 0],*

*# [ 0, 19949, 15, 0, 0, 0],*

*# [ 0, 0, 775, 0, 0, 0],*

*# [ 0, 191, 0, 0, 0, 0],*

*# [ 0, 55, 0, 0, 0, 0],*

*# [ 0, 79, 1, 0, 0, 17]], dtype=int64)*

*from sklearn.metrics import classification\_report*

*# print(classification\_report(y\_test,y\_pred\_SVM))*

*#* ***precision recall f1-score support***

*# 0 1.00 0.56 0.71 9*

*# 1 0.98 1.00 0.99 19964*

*# 2 0.98 1.00 0.99 775*

*# 3 0.00 0.00 0.00 191*

*# 4 0.00 0.00 0.00 55*

*# 5 1.00 0.18 0.30 97*

*# accuracy 0.98 21091*

*# macro avg 0.66 0.46 0.50 21091*

*# weighted avg 0.97 0.98 0.98 21091*

## Decision Tree ALGO:

The motive of Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data(training data)[4]. The decision tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label. The practical implementation is illustrated below:

*# Fitting Decision Tree Classification to the Training set*

*from sklearn.tree import DecisionTreeClassifier*

*classifier\_DT = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)*

*classifier\_DT.fit(X\_train, y\_train)*

*# Predicting the Test set results*

*y\_pred\_DT = classifier\_DT.predict(X\_test)*

*# Making the Confusion Matrix*

*from sklearn.metrics import confusion\_matrix*

*cm\_DT = confusion\_matrix(y\_test, y\_pred\_DT)*

*#IMPLEMENTING CROSS-VALIDATION using cross\_val\_score() function*

*from sklearn.model\_selection import cross\_val\_score*

*cross\_val\_score(classifier\_DT, X\_train, y\_train, cv=3, scoring="accuracy")*

*#array([0.9688478 , 0.96881223, 0.96984353])*

*from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score, classification\_report, confusion\_matrix*

***confusion\_matrix(y\_test,y\_pred\_DT)***

*# Out[49]:*

*# array([[ 9, 0, 0, 0, 0, 0],*

*# [ 0, 19616, 2, 206, 75, 65],*

*# [ 0, 7, 766, 2, 0, 0],*

*# [ 0, 181, 0, 3, 2, 5],*

*# [ 0, 53, 0, 1, 0, 1],*

*# [ 0, 62, 1, 2, 2, 30]], dtype=int64)*

*from sklearn.metrics import classification\_report*

*print(classification\_report(y\_test,y\_pred\_DT))*

*#* ***precision recall f1-score support***

*# 0 1.00 1.00 1.00 9*

*# 1 0.98 0.98 0.98 19964*

*# 2 1.00 0.99 0.99 775*

*# 3 0.01 0.02 0.01 191*

*# 4 0.00 0.00 0.00 55*

*# 5 0.30 0.31 0.30 97*

*# accuracy 0.97 21091*

*# macro avg 0.55 0.55 0.55 21091*

*# weighted avg 0.97 0.97 0.97 21091*

## RandomForest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees[5][6]. Here, Random decision forests correct for decision trees' habit of overfitting to their training set which can be seen in the accuracy below.[7]

*# Fitting Random Forest Classification to the Training set*

*from sklearn.ensemble import RandomForestClassifier*

*classifier\_RF = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)*

*classifier\_RF.fit(X\_train, y\_train)*

*# Predicting the Test set results*

*y\_pred\_RF = classifier\_RF.predict(X\_test)*

*# Making the Confusion Matrix*

*from sklearn.metrics import confusion\_matrix*

*cm\_RF = confusion\_matrix(y\_test, y\_pred\_RF)*

*#IMPLEMENTING CROSS-VALIDATION using cross\_val\_score() function*

*from sklearn.model\_selection import cross\_val\_score*

*cross\_val\_score(classifier\_RF, X\_train, y\_train, cv=3, scoring="accuracy")*

*#array([0.98495733, 0.9851707 , 0.98538407])*

*from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score, classification\_report, confusion\_matrix*

***confusion\_matrix(y\_test,y\_pred\_RF)***

*# Out[53]:*

*# array([[ 4, 0, 5, 0, 0, 0],*

*# [ 0, 19962, 2, 0, 0, 0],*

*# [ 0, 0, 775, 0, 0, 0],*

*# [ 0, 191, 0, 0, 0, 0],*

*# [ 0, 55, 0, 0, 0, 0],*

*# [ 0, 69, 1, 0, 0, 27]], dtype=int64)*

*from sklearn.metrics import classification\_report*

*print(classification\_report(y\_test,y\_pred\_RF))*

***# precision recall f1-score support***

*# 0 1.00 0.44 0.62 9*

*# 1 0.98 1.00 0.99 19964*

*# 2 0.99 1.00 0.99 775*

*# 3 0.00 0.00 0.00 191*

*# 4 0.00 0.00 0.00 55*

*# 5 1.00 0.28 0.44 97*

*# accuracy 0.98 21091*

*# macro avg 0.66 0.45 0.51 21091*

*# weighted avg 0.97 0.98 0.98 21091*

## Naive Bayes:

Naïve Bayes Theorem is based on assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature[8].

*# Fitting Naive Bayes to the Training set*

*from sklearn.naive\_bayes import GaussianNB*

*classifier\_NB = GaussianNB()*

*classifier\_NB.fit(X\_train, y\_train)*

*# Predicting the Test set results*

*y\_pred\_NB = classifier\_NB.predict(X\_test)*

*# Making the Confusion Matrix*

*from sklearn.metrics import confusion\_matrix*

*cm\_NB = confusion\_matrix(y\_test, y\_pred\_NB)*

*#IMPLEMENTING CROSS-VALIDATION using cross\_val\_score() function*

*from sklearn.model\_selection import cross\_val\_score*

*cross\_val\_score(classifier\_NB, X\_train, y\_train, cv=3, scoring="accuracy")*

*#array([0.9413229 , 0.91987909, 0.93602418])*

*from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score, classification\_report, confusion\_matrix*

***confusion\_matrix(y\_test,y\_pred\_NB)***

*# Out[56]:*

*# array([[ 7, 1, 1, 0, 0, 0],*

*# [ 0, 19047, 2, 238, 102, 575],*

*# [ 2, 4, 767, 2, 0, 0],*

*# [ 0, 173, 0, 3, 0, 15],*

*# [ 0, 51, 0, 1, 0, 3],*

*# [ 0, 81, 1, 1, 0, 14]], dtype=int64)*

*from sklearn.metrics import classification\_report*

*print(classification\_report(y\_test,y\_pred\_NB))*

***# precision recall f1-score support***

*# 0 0.78 0.78 0.78 9*

*# 1 0.98 0.95 0.97 19964*

*# 2 0.99 0.99 0.99 775*

*# 3 0.01 0.02 0.01 191*

*# 4 0.00 0.00 0.00 55*

*# 5 0.02 0.14 0.04 97*

*# accuracy 0.94 21091*

*# macro avg 0.47 0.48 0.47 21091*

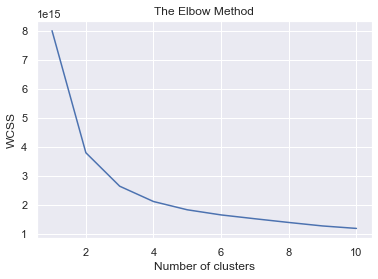
*# weighted avg 0.97 0.94 0.95 21091*

# Part4: Supervised Clustering:

The idea is to cluster the database using k-means to obtain different clusters for the various types of loan status (Current, Late, Fully Paid,etc.).

**Elbow curve:**

In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

**

*Figure: elbow curve*

The cluster value where this decrease in inertia value(WCSS) becomes constant can be chosen as the right cluster value for our data. Here, we can choose any number of clusters between 5 and 8. Lets select 6 as different loan status are also 6 (‘Current', 'Fully Paid', 'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)' and 'Charged Off').

*# Using the elbow method to find the optimal number of clusters*

*from sklearn.cluster import KMeans*

*wcss = []*

*for i in range(1, 11):*

*kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)*

*kmeans.fit(X)*

*wcss.append(kmeans.inertia\_)*

*plt.plot(range(1, 11), wcss)*

*plt.title('The Elbow Method')*

*plt.xlabel('Number of clusters')*

*plt.ylabel('WCSS')*

*plt.show()*

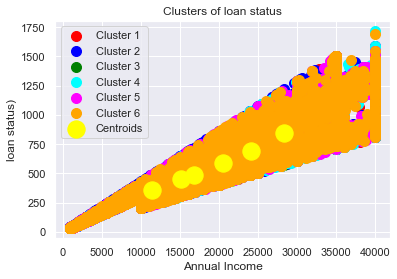
Next step is to train the K-Means model on the dataset:

*from sklearn.cluster import KMeans*

*kmeans = KMeans(n\_clusters = 6, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)*

*y\_kmeans = kmeans.fit\_predict(X)*

If we plot all these 6 clusters for loan status then it would come out as per figure 2.



*Figure2*

Here, it’s not much visible to identify each cluster but we can statistically identify how accurate this cluster method is by using classification report.

*from sklearn.metrics import classification\_report*

*print(classification\_report(y,y\_kmeans))*

*# precision recall f1-score support*

*# 0 0.00 0.04 0.00 25*

*# 1 0.95 0.46 0.62 99850*

*# 2 0.04 0.00 0.01 3896*

*# 3 0.01 0.10 0.02 932*

*# 4 0.00 0.23 0.01 312*

*# 5 0.00 0.15 0.01 436*

*# accuracy 0.43 105451*

*# macro avg 0.17 0.16 0.11 105451*

*# weighted avg ::0.90 0.43 0.58 105451*

As per above matrix accuracy of overall cluster is 43% which is not good but accuracy of identifying cluster 1(‘current’) is 62%.

# Conclusion:

**Accuracy Comparison of Different Algorithms:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Score1** | **Score2** | **Score3** |
| KNN | 0.97411095 | 0.97368421 | 0.97496444 |
| SVM | 0.98374822 | 0.98385491 | 0.98406828 |
| Decision Tree | 0.9688478 | 0.96881223 | 0.96984353 |
| Random Forest | 0.98495733 | 0.9851707 | 0.98538407 |
| Naive Bayes | 0.9413229 | 0.91987909 | 0.93602418 |

Table1: Accuracy matrix of different algorithms

**f1-score Comparison of Different Algorithms:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **F1-score** | | | | |
| **KNN** | **SVM** | **Decision Tree** | **Random Forest** | **Naïve Bayes** |
| 0: 'Charged Off' | 0.00 | 0.71 | 1.00 | 0.62 | 0.78 |
| 1: 'Current' | 0.99 | 0.99 | 0.98 | 0.99 | 0.97 |
| 2: 'Fully Paid' | 0.86 | 0.99 | 0.99 | 0.99 | 0.99 |
| 3: 'In Grace Period' | 0.00 | 0.00 | 0.01 | 0.00 | 0.01 |
| 4: 'Late (16-30 days)' | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5: 'Late (31-120 days)' | 0.00 | 0.30 | 0.30 | 0.44 | 0.04 |

Table2: f1-score matrix of different algorithms

As per above table Random Forrest has the highest accuracy in predicting current loan status among different algorithms. As far as the f1-scores are concern, all the algorithms have accurately identified loan status ‘current’ with almost 98% of accuracy. Individual f1-score is described in table 2. For supervised learning methods, we cannot use it for this project as the accuracy of the cluster is less than 50%.

# Reference:

[1] Abhini Shetye, (2019, February 13). Feature Selection with sklearn and Pandas, towards data science. URL: <https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b>

[2] Altman, Naomi S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression" (PDF). The American Statistician. 46 (3): 175–185. doi:10.1080/00031305.1992.10475879. hdl:1813/31637

[3] Ray Sunil, (2017, SEPTEMBER 13). Understanding Support Vector Machine(SVM) algorithm from examples (along with code), Analytics Vidhya. URL: <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>

[4] Saxena Rahul, (2017, January 2017), How Decision Tree Algorithm Works, Dataaspirant, URL: <https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/>

[5]Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.

[6] Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601.

[7]Hastie Trevor; Tibshirani, Robert; Friedman, Jerome(2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

[8] Ray Sunil, (2017, SEPTEMBER 11), 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, Analytics Vidhya.URL: <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>