Credit Risk Analysis in Social Lending Ecosystem

Hussien Hussien
hhussien@uwo.ca
Western University
London, Ontario, Canada

KEYWORDS

Decision Trees, Credit Risk Analysis, Machine Learning.

ACM Reference Format:
Hussien Hussien. 2020. Credit Risk Analysis in Social Lending Ecosystem. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

LendingClub is a peer-to-peer lending platform. It enables borrowers to create unsecured loans between $1,000-40,000. Loan period varies but the most common is 36 months. When a loan is created on the platform; information on the borrower, loan amount, interested rates, loan grade and loan purpose are attached. Investors can then browse through loans and invest based on that information.

Investors realize returns once borrowers pay back the principle plus interest. A large reason investors are attracted to LendingClub is because of these interest rates which vary from 5.03%-30.9%, depending on the credit grade given to the loan by LendingClub. Each loan grade ranges alphabetically from A to G, with A being the highest-grade; lowest-interest loan and G being the lowest-grade; highest-interest loan. Each letter grade is given a finer subgrade from 1-5.

While lower grade loans can represent large marginal returns if paid back, default rates in the those grades tend to offset any consistent and significant returns. In this paper we attempt to tackle this problem by building a model that can predict loan outcomes and thus act as a screen for low-risk loans with high interest rates. The input to our algorithms are $y = \{1, 0\}$ which is the outcome of the loan granted and $X_1, X_2, ...X_n$ which are a variety of attributes about the loan and borrower. We then use a Logistic regression as a baseline model and experiment with a series of tree based algorithms to output a predicted outcome of the loan, $\hat{y}$.

2 DATASET AND FEATURES

The data for this project was extracted from LendingClub.com's investor portal. They allow users with investor accounts to download a log of loan information by quarter. Loan data from Apr-2017 to Dec-2019. The original statuses of each loan were Current, Fully Paid, Charged Off, Late (31-120 days), In Grace Period, Late (16-30 days) and Default. Given the fact that we aim to screen for loans that are likely to pay off the principal plus interest; we decide to drop most loans currently being paid off and assign remaining classifications as follows:

$$y = \begin{cases} 1 & \text{if loan status} \in \{\text{'Charged Off', 'Default','Late (31-120 days)'}\} \\ 0 & \text{if loan status = 'Fully Paid'} \end{cases}$$

Preprocessing

With originally over 151 columns, 1,455,489 rows and a large amount of NA values, a significant amount of data cleaning was required to prepare these test sets for modelling. Rows with over 75% Nan values were dropped. All other NA values were imputed with the median of that predictor's values, 0 or another value depending

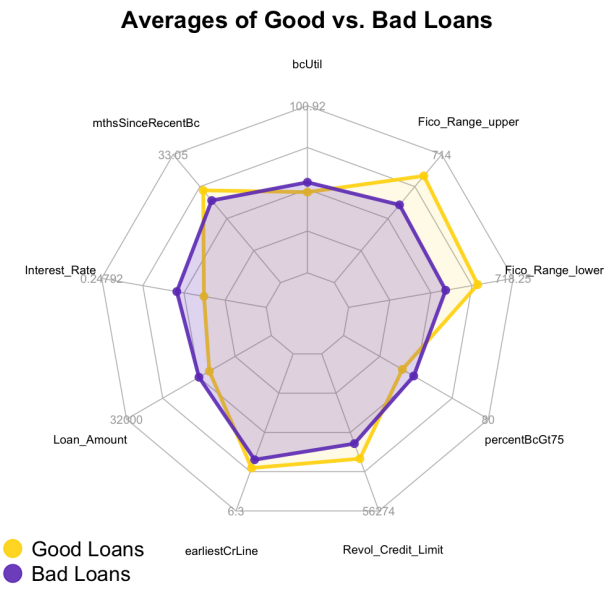


Figure 1: Comparison of averages in significant predictors for both loans that were fully repaid and loans that defaulted.

Feature Descriptions in clock wise order from top-most feature; *bcUtil*: Ratio of total current balance to high credit/credit limit for all bankcard accounts, *Fico Range Upper*: The upper boundary range the borrower's FICO at loan origination belongs to, *Fico Range Lower*: The lower boundary range the borrower's FICO at loan origination belongs to, *percent bc gt 75*: Percentage of all bankcard accounts > 75% of limit, *Revol Credit Limit*: Total revolving high credit/credit limit, *EarliestCrline*: Days since the borrower's earliest reported credit line was opened, *Loan Amnt*: Loan Amount, *Interest Rate*: Interest Rate on Loan, *mthsSinceRecentBc*: Months since most recent bankcard account opened

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

on context and distribution of that predictor's data. Certain high dimensional numeric data was bucket into groups based on 4, 8 or 10 quantile cuts. Since this data set contains a large number of conclusive loans, 27 predictors are removed that wold not be available when evaluating a new loan. IE, delinquency amount. Additional processing and reductions were done on an adhoc basis resulting in a cleaned data set of 504242 loans and 84 predictors. 25% of these loans are classified as bad loans.

In EDA.rmd, we build a simple logistic regression model for analysis to get an idea of which features are significant based on p-vlaues. Below are a few examples of significant features:

- Total amount of loan
- Loan interest rate
- Sub-grade of loan
- Employment duration
- Home ownership type (Rent, Own, etc)
- Total annual income
- Repayment period
- Debt to income ratio

The discrepancy between good and bad loans among a subset of significant predictors can be found in figure 1.

While good for data exploration, this data set was further processed for modelling. Categorical variables were encoded into dummy variables. This resulted with over 1000+ predictors. To reduce dimensions, dummy variables with a frequency of less than 500 and variables with variance of less than 0.07 were removed from the dataset. Reducing total amount of predictors back down to 86 which were then normalized and used as input for our algorithms.

3 METHODS

In this project, we analyze a variety of tree-based classification methods. All of which are applied using the popular machine learning library Scikit-learn. Gridsearch is applied to determine best set of hyper-parameters for each model via brute force testing. In order to combat over-fitting, all models are fit and chosen based on cross validated evaluation metric. Measures are taken to avoid over fitting such as decorrelation of trees, regularization and cross validation.

LOGISTIC REGRESSION

A binary classification algorithm that extends the linear regression model. It outputs probability of a given class via the logit function. Where B_i represents coefficient of the ith predictor.
Logit Function

$$P(X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}}$$

We choose the l2 penalty parameter, denoted by λ and set it to 0.01. After choosing the regularization method, logistic regression aims to fit it's coefficients to maximize the penalized likelihood function.

Decision Trees

A decision tree is a very intuitive algorithm to classify data points and is the building block to the Random Forest and Bagging Tree algorithms covered in the next sub section. It segments the predictor space into a number of regions, then classifies a training observations based on the most commonly occurring class of the training data in that segment. An example of a segment in this case could be; Borrowers who have less than $15,000 total revolving credit and want to borrow a loan amount of $35,000+. Say there were 100 cases of borrowers who were in this segment in our training set and 63 of them defaulted on their loans. When we have a new observation that falls into this category, we will assign it a prediction of 'Default' by majority. The decision tree algorithms decides on how to best segment the data by doing what is called recursive binary splitting which involves splitting a predictor, X_k, into a variety of regions and chooses the split that results in the greatest reduction in the cost function. For a Decision Tree Classifier, the cost function we aim to minimize is called the Gini Index denoted as;

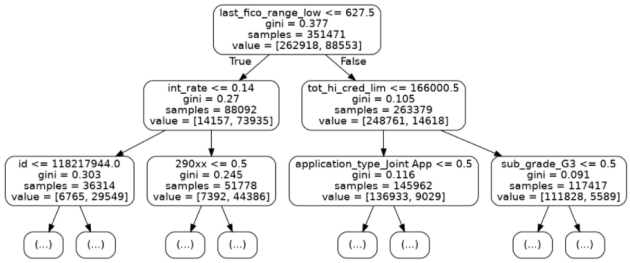


Figure 2: The top 2 levels of trained decision tree.

$$GiniIndex = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$$

where $\hat{p}_{mk}$ is the proportion of training observations from the m^{th} region that are from the k^{th} class. This measures the total variance across k-classes, thus a smaller Gini Index the better the fit. It is easy to see that if you simply split the decision tree into the same number of segments as observations, you would just end up with a nearest neighbor algorithm of K=1. This is avoided by pruning the tree, which is the process of removing splits or segments that don't make significant improvements on the tree's total fit. Hyper Parameters tuned to:

- **Max Features:** $\sqrt{p}$
- **Max Depth:** 10

Bagging Trees

Bagging Tree is a powerful ensemble method that aims to reduce the variance in decision trees. It trains k trees by training each one on a bootstrapped sample of the training data. It samples a difference portion of observations and features in each model. For each observation, we obtain a prediction from each of the k trees and assign the observation by majority vote; the final prediciton is the most commonly occuring class. This takes a wisdom of the crowds approach that the aggragate of each model will preform better than each model individually. Through hyper parameter tuning, the best evaluation metrics where obtained by creating 500 trees and sampling 80% of the data each time.

Random Forest

Random Forest is a further enhancement on the bagging tree that forces each tree to be as de-correlated as possible. A great summary of how it works can be found in the following passage from the ISLR textbook:

"Random forests provide an improvement over bagged trees by way of a small tweak that de-correlates the trees. As in bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose m $\sqrt{p}$ —that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

... bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.... Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average (p m)/p of the splits will not even consider the strong predictor, and so other predictors will have more of a chance... thereby making the average of the resulting trees less variable and hence more reliable." (James, Gareth, et al, 2017, pg. 319)

In this case, we obtained the best scores by creating 500 trees and sampling 80% of the observations and $\sqrt{p}$ predictors.

4 RESULTS

4.1 Evaluation

We the primary metric we use to evaluate model classification performance on both train and test sets is *Area Under the Receiver Operating Characteristic Curve* (ROC AUC score or AUC score). The ROC curve is created by plotting the true positive rate against the false positive rate at a range of probability threshold settings. Taking the area of this curve allows use to evaluate an estimators classification ability in a simple way.

$$\text{AUC} = \int_{x=0}^{1} TPR(FPR^{-1}(x))dx$$

where x are the range of thresholds from 0-1.0. Another reason we use AUC is because in practice a user may want to adjust the probability threshold for which loans get screened. AUC and the ROC curve allows us to easily see how our model performs under various thresholds.

We also analyze secondary metrics like accuracy (percentage of correct predictions out of all predictions) and Precision. Percision can be throught of as the portion of loans that actually defaulted out of the loans our estimators predicted would default.

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$

When analyzing the ability of our models to screen for loans, we use two metrics: Average returns and Return on Investment.

Average returns is obtained by taking the subset of loans that our models didn't predict would default, simply summing interest rates of the loans that were repaid and dividing them by the amount of loans.

Return on Investment, ROI, is a measure of the average dollar return per dollar invested. ROIs greater than 1.0 are positive returns, ROIs less than 1.0 are negative returns. It is easy to see an ROI as large as possible is desired in this case.

4.2 Training Results

Table 1 displays the cross validated AUC scores of the various estimators on the training data.

Table 1: CV Train Results

Model	AUC
Logistic Regression	0.9585
Decision Tree	0.9263
Bagging Tree	0.9583
Random Forest	0.9596

4.3 Out-Of-Sample Results

Table 3 shows the AUC scores of the various estimators on the test set data.

Table 2: Test Results

Model	AUC	Accuracy	Precision
Logistic Regression	0.9584	0.9257	0.8805
Decision Tree	0.9110	0.8855	0.8330
Bagging Tree	0.9582	0.9241	0.86523
Random Forest	0.9599	0.9264	0.8681

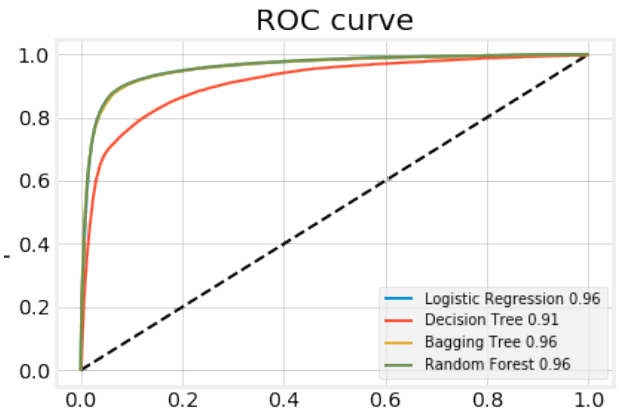


Figure 3: ROC AUC Curve
X-Axis: False positive Rate Y-Axis: True positive Rate

4.4 Discussion

Random Forest seems to perform on all training and test evaluation metrics, with a Test AUC of 0.9599 and Test Accuracy of 0.9264. The improvement appears incremental over our baseline logistic regression model. I would imagine this is a result of some combination of highly-correlated predictors and noise.

In Table 3, we examine the ability for each model to screen for bad loans in the test data set. The strategy simulate is creating a portfolio of subset of loans that our model does not predict will default, investing the entire loan amount in each loan and examining the ROI and average returns based on the ground truth outcome. Here the naive approach is to invest in each loan in that grade. We use default probability threshold of 0.5 to screen loans.

Table 3: Loan Screening Results

Subgrade	Model	Average Return	ROI
Total Returns	Naive Approach	0.897	0.874
	Logistic Regression	1.05	1.05
	Decision Tree	1.01	0.994
	Bagging Tree	1.05	1.04
	Random Forest	1.06	1.05
G Grade	Naive Approach	0.625	0.61
	Logistic Regression	1.12	1.13
	Decision Tree	0.843	0.822
	Bagging Tree	1.15	1.15
	Random Forest	1.15	1.15
F Grade	Naive Approach	0.666	0.644
	Logistic Regression	1.14	1.14
	Decision Tree	1.1	1.1
	Bagging Tree	1.17	1.16
	Random Forest	1.17	1.17
D Grade	Naive Approach	0.737	0.706
	Logistic Regression	1.08	1.07
	Decision Tree	1.01	0.997
	Bagging Tree	1.08	1.07
	Random Forest	1.09	1.08
A Grade	Naive Approach	0.969	0.957
	Logistic Regression	1.04	1.04
	Decision Tree	1.01	1.01
	Bagging Tree	1.04	1.03
	Random Forest	1.04	1.04
F & G Grade	Naive Approach	0.64	0.614
	Logistic Regression	1.13	1.14
	Decision Tree	0.868	0.844
	Bagging Tree	1.16	1.15
	Random Forest	1.16	1.16

The model that best screens for bad loans appears to be the Random Forest Model across the board with Returns as high as 17% in the F grade loan category.

5 CONCLUSION/FUTURE WORK

In this paper, we build various classification models to predict loan defaults on the LendingClub Peer-to-Peer lending platform. We train these models on 500,000 loan outcomes from 2017-2019. Random Forest is the strongest classifier with Test AUC of 0.9599 and Test Accuracy of 0.9264. We construct a series of simulated portfolios of loans that the Random Forest model predicted had less than a 50% of defaulting. This consistently gave us significant results with average Return-on-investment of 1.17 when investing in F Grade loans compared to 1.04 when investing in the higher quality A Grade loans.

In the future, I'd like to experiment with a variety of other classification algorithms. I'd also like to analyze portfolio performance when tweaking probability thresholds to be more or less risk averse. General analysis into the viability of this model as an investment screening tool will also be done.

6 REFERENCES

1. Corp., LendingClub. "Loan Statistics." Historical Loan Issuance Data, 2019, www.lendingclub.com/statistics/additional-statistics.

2. James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R. Springer, 2017.

3. Yiu, Tony. "Turning Lending Club's Worst Loans into Investment Gold." Medium, Towards Data Science, 19 June 2019, towardsdatascience.com/turning-lending-clubs-worst-loans-into-investment-gold-475ec97f58ee.

4.Fawcett, Tom. (2006). Introduction to ROC analysis. Pattern Recognition Letters. 27. 861-874. 10.1016/j.patrec.2005.10.010.

Appendix

Table of Contents

Data cleaning and preprocessing

Import libraries

```
#Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error, mean_absolute_error

import math

%matplotlib inline
```

Helper Functions

```
def mega_set(*args):
    '''
    appends datasets together
    '''
    master = pd.DataFrame()
    for i in args:
        i.drop(i.tail(2).index,inplace=True)
        if master.empty:
            # Clean i
            master = i
        else:
            # Clean i
            master = master.append(i, ignore_index = True)
    return master.reset_index()

def na_percent(data):
    '''
    Returns rows where there is a non zero percentage of missing values
    '''
    percentage_of_na = (data.isna().sum()) / data.shape[0]
    return percentage_of_na[percentage_of_na != 0.0]

def percent_to_float(x):
    '''
    Turns percentage values into floats
    '''
    if type(x) == float:
        return x / 100
    else: return float(x.strip('%')) / 100

def replace_na_with(x,rep):
    '''
    Replace a given x value with replacement value if x is NaN
    '''
    if (type(x) != str) and np.isnan(x):
        return rep
    else: return x

# Convert loan_status to 1/0
def convert_to_binary(x):
    if  x == 'Late (31-120 days)' or x == 'Charged Off' or x == 'Default':
        return 1
    elif x == 'Fully Paid':
        return 0
```

Load and merge loan data

```
# 2019
df_2019Q4 = pd.read_csv('lendingclub_data/LoanStats_securev1_2019Q4.csv', low_memory=False, header=1)
df_2019Q3 = pd.read_csv('lendingclub_data/LoanStats_securev1_2019Q3.csv', low_memory=False, header=1)
df_2019Q2 = pd.read_csv('lendingclub_data/LoanStats_securev1_2019Q2.csv', low_memory=False, header=1)
df_2019Q1 = pd.read_csv('lendingclub_data/LoanStats_securev1_2019Q1.csv', low_memory=False, header=1)

# 2018
df_2018Q4 = pd.read_csv('lendingclub_data/LoanStats_securev1_2018Q4.csv', low_memory=False, header=1)
df_2018Q3 = pd.read_csv('lendingclub_data/LoanStats_securev1_2018Q3.csv', low_memory=False, header=1)
df_2018Q2 = pd.read_csv('lendingclub_data/LoanStats_securev1_2018Q2.csv', low_memory=False, header=1)
df_2018Q1 = pd.read_csv('lendingclub_data/LoanStats_securev1_2018Q1.csv', low_memory=False, header=1)


# 2017
df_2017Q4 = pd.read_csv('lendingclub_data/LoanStats_securev1_2017Q4.csv', low_memory=False, header=1)
df_2017Q3 = pd.read_csv('lendingclub_data/LoanStats_securev1_2017Q3.csv', low_memory=False, header=1)
df_2017Q2 = pd.read_csv('lendingclub_data/LoanStats_securev1_2017Q2.csv', low_memory=False, header=1)
df_2017Q1 = pd.read_csv('lendingclub_data/LoanStats_securev1_2017Q1.csv', low_memory=False, header=1)
```

```
loan_data = mega_set(df_2019Q4, df_2019Q3,df_2019Q2, df_2019Q1, # 2019
                     df_2018Q4, df_2018Q3,df_2018Q2, df_2018Q1, # 2018
                     df_2017Q4, df_2017Q3,df_2017Q2, df_2017Q1, #2017
```

```
                    )
loan_data2 = loan_data.copy()
```

Drop bad rows and columns

```
# We will ignore all loans that aren't Late (31-120 days),
# Default, Charged Off or Fully Paid

# Get names of indexes for each irrelevant rows
indexNames = loan_data[(loan_data['loan_status'] == 'Current') | (loan_data['loan_status'] =='In Grace Period') | (loan_data['loan_s

# Delete these row indexes
loan_data.drop(indexNames , inplace=True)
```

```
# Drop columns that don't appear relevant or
# we wouldn't have at the time of evaluating a loan
drop_list = ["collection_recovery_fee",
             "funded_amnt_inv",#'issue_d',
             "installment","last_credit_pull_d",
             "last_pymnt_amnt","last_pymnt_d",
             "member_id",'url', 'title',
             "next_pymnt_d","num_tl_120dpd_2m",
             "num_tl_30dpd","out_prncp",
             "out_prncp_inv","recoveries",
             "total_pymnt","total_pymnt_inv",
             "total_rec_int","total_rec_late_fee",
             "total_rec_prncp","url", 'hardship_flag',
             'debt_settlement_flag', 'acc_now_delinq',
             'delinq_amnt', 'pymnt_plan', 'grade']

loan_data.drop(drop_list, inplace=True,  axis=1)
```

```
# Delete Rows with over 0.75% Nans, I checked and they have nothing to do with the loan status
loan_data.dropna(thresh=(loan_data.shape[0] * 0.75), axis = 1, inplace=True)
```

```
# There is a an large overlap between data that is nan for revol_util, bc_open_to_buy, bc_util, percent_bc_gt_75
# There is very little charged off account and 0 default accounts o this data set so I will just remove these rows
temp = loan_data[loan_data['revol_util'].isna()][loan_data['percent_bc_gt_75'].isna()][loan_data['bc_open_to_buy'].isna()][loan_data
cond = loan_data['id'].isin(temp['id'])
loan_data.drop(loan_data[cond].index, inplace = True)
```

Fill NA values

```
# Replace the following column's NA values with 'None'
replace_w_nones = ['emp_title','emp_length','zip_code']

for i in replace_w_nones:
    loan_data[i] = loan_data[i].map(lambda x: replace_na_with(x,'None'))
```

```
# Replace the following column's NA values with 0
loan_data['inq_last_6mths'] = loan_data['inq_last_6mths'].map(lambda x: replace_na_with(x,0))
```

```
# Replace the following column's NA values with the mean of each column
replace_w_median = ['dti', 'mths_since_recent_bc',
                  'mths_since_recent_inq','mo_sin_old_il_acct',
                  'revol_util', 'bc_open_to_buy', 'all_util','il_util',
                  'percent_bc_gt_75', 'bc_util',
                  'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op','mths_since_rcnt_il',
                 'avg_cur_bal','num_rev_accts', 'pct_tl_nvr_dlq']

loan_data['revol_util'] = loan_data['revol_util'].map(lambda x: percent_to_float(x))

for j in replace_w_median:
    median_j = loan_data[j].median()
    loan_data[j] = loan_data[j].map(lambda x:replace_na_with(x,median_j))
```

Bin Items

```
# Bin 'Months since' items
bin_items = ['mths_since_recent_inq', 'mo_sin_old_rev_tl_op',
             'mths_since_recent_bc','mo_sin_old_il_acct','mo_sin_rcnt_rev_tl_op','mths_since_rcnt_il']

for k in bin_items:
    print(k)
    loan_data2[k] = pd.qcut(loan_data[k], 4, labels=['0', '1', '2', '3'])
```

```
bin_items = ['percent_bc_gt_75']

for k in bin_items:
    print(k)
    loan_data2[k] = pd.qcut(loan_data[k], 5, labels=['0', '1', '2', '3'],duplicates='drop')
```

```
# Convert earliest credit line date into number of days between earliest credit line and loan issue date, then bucket them into 10 q
loan_data['days_since_earliest_cr_line'] = pd.to_datetime(loan_data['issue_d']) - pd.to_datetime(loan_data['earliest_cr_line'])
loan_data['days_since_earliest_cr_line'] = loan_data['days_since_earliest_cr_line'].dt.days
loan_data['days_since_earliest_cr_line'] = pd.qcut(loan_data['days_since_earliest_cr_line'],
                                           q=10,
                                           labels=['0', '1', '2', '3','4','5','6','7','8','9'],
                                           duplicates='drop')
loan_data.drop(['earliest_cr_line'], axis=1, inplace=True)
```

Encode y variables into binary

```
loan_data['loan_status'] = loan_data['loan_status'].map(lambda x:convert_to_binary(x))
```

Misc

```
# Convert interest rate from percentage to float value
loan_data['int_rate'] = loan_data['int_rate'].map(lambda x: percent_to_float(x))
```

Exploratory Data Analysis

```
knitr::opts_chunk$set(echo = TRUE)
suppressMessages(library(dplyr))
# Libraries
library(tidyverse)
library(viridis)
library(patchwork)
#library(hrbrthemes)
library(fmsb)
library(colormap)
library(BBmisc)
```

Load Data

```
df <- read.delim("data/loan_data.csv", sep = ",", dec = ".") #Read Data

df.num <- dplyr::select_if(df, is.numeric)

drops <- c('index','id')
y <- c('loan_status')

df.num <- df.num[ , !(names(df.num) %in% drops)]
#df.num <- df.num[ , !(names(df.num) %in% y)]


#df.num <- cbind(df.num,factor(df$loan_status, levels=c(1,0),labels=c(1,0)))

levels(df.num$loan_status)

colnames(df)[-'loan_status']
```

Fit logistic regression tree to look at importance features.

```
fit_glm <- glm(loan_status ~ ., data=df.num,family=binomial)
summary(fit_glm)
```

Output:

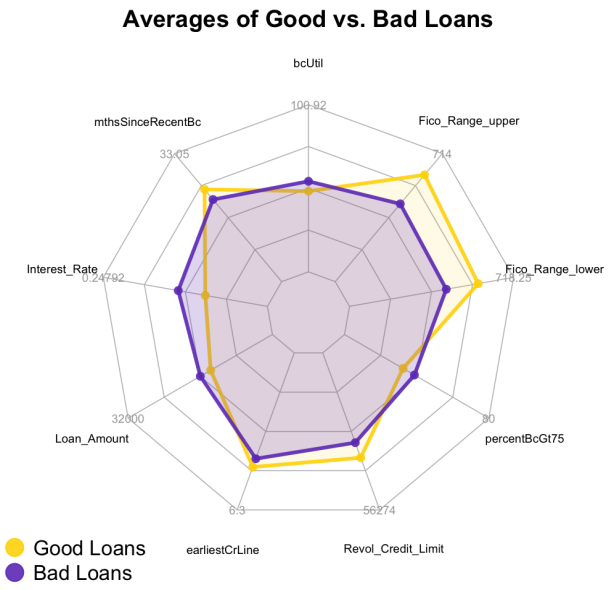
```
glm.fit: fitted probabilities numerically 0 or 1 occurred
Call:
glm(formula = loan_status ~ ., family = binomial, data = df.num)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-7.0696  -0.3165  -0.1473   0.0740   8.4904

Coefficients: (2 not defined because of singularities)
                             Estimate Std. Error  z value Pr(>|z|)
(Intercept)                 5.831e+00  1.316e+00    4.430 9.42e-06 ***
X                          -1.386e-06  1.733e-08  -79.970  < 2e-16 ***
loan_amnt                   4.838e-05  6.627e-07   73.013  < 2e-16 ***
funded_amnt                        NA         NA       NA       NA
int_rate                    3.400e+00  1.171e-01   29.045  < 2e-16 ***
annual_inc                 -3.787e-07  1.004e-07   -3.772 0.000162 ***
dti                         3.553e-03  2.430e-04   14.621  < 2e-16 ***
delinq_2yrs                -5.698e-02  9.294e-03   -6.131 8.71e-10 ***
fico_range_low             -2.374e+00  3.282e-01   -7.234 4.70e-13 ***
fico_range_high             2.386e+00  3.282e-01    7.269 3.63e-13 ***
inq_last_6mths              2.107e-02  8.690e-03    2.424 0.015336 *
open_acc                    3.095e-02  2.322e-02    1.333 0.182577
pub_rec                     9.616e-03  3.852e-02    0.250 0.802860
revol_bal                  -5.228e-06  1.217e-06   -4.295 1.75e-05 ***
revol_util                 -1.234e-01  5.740e-02   -2.150 0.031541 *
total_acc                  -2.322e-02  1.107e-02   -2.098 0.035942 *
last_fico_range_high       -4.206e-02  1.694e-04 -248.330  < 2e-16 ***
last_fico_range_low         2.722e-03  8.498e-05   32.035  < 2e-16 ***
collections_12_mths_ex_med  1.462e-01  3.200e-02    4.570 4.88e-06 ***
policy_code                        NA         NA       NA       NA
```

```
tot_coll_amt                -6.303e-07  5.367e-07   -1.174 0.240216
tot_cur_bal                 -9.268e-07  2.051e-07   -4.519 6.22e-06 ***
open_acc_6m                  7.640e-03  7.258e-03    1.053 0.292448
open_act_il                 -2.606e-02  1.119e-02   -2.328 0.019894 *
open_il_12m                  3.778e-03  2.197e-02    0.172 0.863446
open_il_24m                  1.544e-02  1.591e-02    0.971 0.331647
mths_since_rcnt_il           2.100e-03  2.763e-04    7.601 2.93e-14 ***
total_bal_il                -6.486e-07  1.104e-06   -0.588 0.556859
il_util                      2.619e-03  3.914e-04    6.691 2.22e-11 ***
open_rv_12m                  2.181e-02  2.126e-02    1.026 0.304911
open_rv_24m                  2.901e-02  1.547e-02    1.875 0.060853 .
max_bal_bc                   3.059e-06  2.080e-06    1.471 0.141368
all_util                    -2.364e-03  5.487e-04   -4.309 1.64e-05 ***
total_rev_hi_lim             4.873e-06  7.073e-07    6.889 5.62e-12 ***
inq_fi                      -1.041e-02  4.361e-03   -2.388 0.016937 *
total_cu_tl                 -2.195e-03  2.228e-03   -0.985 0.324514
inq_last_12m                -5.784e-03  3.247e-03   -1.782 0.074800 .
acc_open_past_24mths        -5.093e-02  1.485e-02   -3.429 0.000605 ***
avg_cur_bal                 -5.278e-07  7.894e-07   -0.669 0.503770
bc_open_to_buy               1.855e-06  1.366e-06    1.358 0.174391
bc_util                      2.293e-03  5.293e-04    4.332 1.48e-05 ***
chargeoff_within_12_mths     1.685e-01  4.421e-02    3.811 0.000139 ***
mo_sin_old_il_acct           2.376e-04  1.338e-04    1.776 0.075710 .
mo_sin_old_rev_tl_op         1.676e-03  1.127e-04   14.874  < 2e-16 ***
mo_sin_rcnt_rev_tl_op        6.851e-04  5.208e-04    1.316 0.188324
mo_sin_rcnt_tl               3.406e-03  9.415e-04    3.618 0.000297 ***
mort_acc                    -1.143e-02  1.131e-02   -1.011 0.312206
mths_since_recent_bc         1.684e-03  2.342e-04    7.188 6.59e-13 ***
mths_since_recent_inq        3.848e-03  1.284e-03    2.997 0.002727 **
num_accts_ever_120_pd        1.938e-02  5.000e-03    3.877 0.000106 ***
num_actv_bc_tl               2.556e-02  7.374e-03    3.466 0.000528 ***
num_actv_rev_tl              4.213e-02  9.373e-03    4.495 6.95e-06 ***
num_bc_sats                 -2.278e-02  5.802e-03   -3.926 8.65e-05 ***
num_bc_tl                   -1.566e-02  3.735e-03   -4.192 2.76e-05 ***
num_il_tl                    1.903e-02  1.110e-02    1.715 0.086324 .
num_op_rev_tl               -5.374e-02  1.157e-02   -4.643 3.43e-06 ***
num_rev_accts                3.512e-02  1.124e-02    3.123 0.001789 **
num_rev_tl_bal_gt_0         -3.399e-02  9.853e-03   -3.450 0.000561 ***
num_sats                     1.828e-02  2.324e-02    0.787 0.431567
num_tl_90g_dpd_24m           3.761e-02  1.507e-02    2.496 0.012562 *
num_tl_op_past_12m          -2.316e-02  2.065e-02   -1.122 0.262065
pct_tl_nvr_dlq               9.042e-03  8.627e-04   10.481  < 2e-16 ***
percent_bc_gt_75            -1.820e-03  3.003e-04   -6.061 1.35e-09 ***
pub_rec_bankruptcies         1.387e-01  4.115e-02    3.370 0.000751 ***
tax_liens                    6.879e-02  4.227e-02    1.627 0.103667
tot_hi_cred_lim              8.296e-08  1.720e-07    0.482 0.629575
total_bal_ex_mort           -2.317e-06  9.734e-07   -2.380 0.017305 *
total_bc_limit              -3.446e-06  1.150e-06   -2.997 0.002723 **
total_il_high_credit_limit   2.776e-06  5.281e-07    5.257 1.46e-07 ***
days_since_earliest_cr_line  2.471e-02  4.165e-03    5.933 2.97e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 568415  on 504241  degrees of freedom
Residual deviance: 216546  on 504174  degrees of freedom
AIC: 216682

Number of Fisher Scoring iterations: 7
```

Spider Plot

```
# Create data
set.seed(1)

#Max, Min, Value
min_max <- data.frame(
  bcUtil= c(max(df$bc_util) * 0.40,min(df$bc_util)),
  mthsSinceRecentBc =c(max(df$mths_since_recent_bc)* 0.05,min(df$mths_since_recent_bc)),
  Interest_Rate= c(max(df$int_rate)* 0.8,min(df$int_rate)),
  Loan_Amount = c(max(df$loan_amnt)* 0.8,min(df$loan_amnt)),
  earliestCrLine = c(max(df$days_since_earliest_cr_line)* 0.7,min(df$days_since_earliest_cr_line)),
  Revol_Credit_Limit= c(max(df$total_rev_hi_lim)* 0.03,min(df$total_rev_hi_lim)),
  percentBcGt75= c(max(df$percent_bc_gt_75)* 0.8,min(df$percent_bc_gt_75)),
```

```
  Fico_Range_lower= c(max(df$fico_range_low)* 0.85,min(df$fico_range_low)),
  Fico_Range_upper=c(max(df$fico_range_high)* 0.84,min(df$fico_range_high)))


data.good <- df[df$loan_status == 0,] %>%
  select('bc_util', # Ratio of total current balance to high credit/credit limit for all bankcard accounts.
         'mths_since_recent_bc', # Months since most recent bankcard account opened.
         'int_rate', #Interest Rate on Loan
         'loan_amnt', # Loan Amount
         'days_since_earliest_cr_line', # Days since the borrower's earliest reported credit line was opened
         'total_rev_hi_lim', # MTotal revolving high credit/credit limit
         'percent_bc_gt_75', # Percentage of all bankcard accounts > 75% of limit.
         'fico_range_low', # The ulower boundary range the borrower’s FICO at loan origination belongs to.
         'fico_range_high') # The upper boundary range the borrower’s FICO at loan origination belongs to.
temp1 <- rbind(min_max, c(mean(data.good$bc_util),mean(data.good$mths_since_recent_bc),
                          mean(data.good$int_rate),mean(data.good$loan_amnt),
                          mean(data.good$days_since_earliest_cr_line),
                          mean(data.good$total_rev_hi_lim),
                          mean(data.good$percent_bc_gt_75),mean(data.good$fico_range_low),
                          mean(data.good$fico_range_low)))

# Bad

data.bad <- df[df$loan_status == 1,] %>%
  select('bc_util', # Ratio of total current balance to high credit/credit limit for all bankcard accounts.
         'mths_since_recent_bc', # Months since most recent bankcard account opened.
         'int_rate', #Interest Rate on Loan
         'loan_amnt', # Loan Amount
         'days_since_earliest_cr_line', # Days since the borrower's earliest reported credit line was opened
         'total_rev_hi_lim', # MTotal revolving high credit/credit limit
         'percent_bc_gt_75', # Percentage of all bankcard accounts > 75% of limit.
         'fico_range_low', # The ulower boundary range the borrower’s FICO at loan origination belongs to.
         'fico_range_high') # The upper boundary range the borrower’s FICO at loan origination belongs to.

temp2 <- rbind(temp1, c(mean(data.bad$bc_util),mean(data.bad$mths_since_recent_bc),
                          mean(data.bad$int_rate),mean(data.bad$loan_amnt),
                          mean(data.bad$days_since_earliest_cr_line),
                          mean(data.bad$total_rev_hi_lim),
                          mean(data.bad$percent_bc_gt_75),mean(data.bad$fico_range_low),
                          mean(data.bad$fico_range_low)))

colors_fill <- c(
                 scales::alpha("gold", 0.1),
                 scales::alpha("#6f42c1", 0.2),
                 scales::alpha("skyblue", 0.2))

# Define line colors
colors_line <- c(
                scales::alpha("gold", 0.9),
                scales::alpha("#6f42c1", 0.9),
                scales::alpha("royalblue", 0.9))

#par(mar=c(2,2,2,1))
#par(fig=c(10,70,60,100)/100)

# Custom the radarChart !
radarchart( temp2, axistype=2, #seg=10,
            title="Averages of Good vs. Bad Loans",

  #custom polygon
  pcol = colors_line, pfcol = colors_fill , plwd=3, plty=c(1,1,1,1),

  #custom the grid
  cglcol="grey", cglty=1, axislabcol="darkgrey", caxislabels=seq(0,20,5), cglwd=0.8,

  #custom labels
  vlcex=0.6, calcex=0.3, palcex=0.6,pangle=45

  )
legend(x=-1.5,
       y=-1.0,
       legend = c("Good Loans","Bad Loans"),
       bty = "n", pch=20 , col = colors_line, cex = 1.05, pt.cex = 3)
```




```
# Portion of good loans
nrow(data.good) / nrow(df.num)

# Portion of badloans
nrow(data.bad) / nrow(df.num)
```

output:

```
[1] 0.7488151
[1] 0.2511849
```

Model Fitting:

Import libraries

```
#Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score
from sklearn.metrics import recall_score, f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.cluster import KMeans
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.utils.multiclass import unique_labels
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import VarianceThreshold
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, mean_squared_error, roc_auc_score
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV, KFold, RandomizedSearchCV
import pydotplus
from IPython.display import Image
from functools import reduce
from tqdm import tqdm
import sklearn.tree as tree
import pydotplus
from sklearn.externals.six import StringIO

%matplotlib inline
plt.style.use('fivethirtyeight')
```

Load, clean, transform and Split

```
# Load data
loan_data = pd.read_csv('storage/loan_data.csv', error_bad_lines=False, low_memory=False).drop('Unnamed: 0', axis=1)
df = loan_data.copy()
df.drop(['emp_title'], axis=1,inplace=True)
df.drop_duplicates(inplace=True)
df.shape
```

>>> (502104, 82)

```
# Clean, encode categorical variables
zip_codes = pd.get_dummies( df['zip_code'])
#Drop low frequency zipcodes
for i in zip_codes.columns:
    if zip_codes[i].sum() < 500:
        zip_codes.drop(i,axis=1,inplace=True)

# Encode categorical columns
df.dtypes[df.dtypes == 'object']
# list of categorical columns
cat_cols = ['term','sub_grade','home_ownership','verification_status','purpose',
            'addr_state','initial_list_status','application_type','days_since_earliest_cr_line','emp_length']

df2 = pd.get_dummies(df[cat_cols], prefix_sep='_', drop_first=True)
# Merge
df = df.drop(cat_cols, axis=1)
df = df.drop('zip_code', axis=1)
df = pd.concat([df, df2,zip_codes], axis=1)
```

Split data into test and training sets

```
X = df.drop(['loan_status','issue_d'], axis=1)
y = df['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=30)
X_train_results = X_train.copy()
X_test_results = X_test.copy()
```

Transform

```
# We drop all columns with less than 0.08 variance and standardize them
model_pipeline_transform_dem = Pipeline([
    ('var_thresh', VarianceThreshold(0.07)),
    ('scale', StandardScaler())
])
```

```
X_test.shape
```

>>> (150631, 490)

```
X_train = model_pipeline_transform_dem.fit_transform(X_train)
X_test = model_pipeline_transform_dem.transform(X_test)
X_test.shape
```

>>> (150631, 86)

Model Fitting and CV

Logistic Regression

```
model_pipeline_logreg = Pipeline([
    ('logreg', LogisticRegression(penalty='l2',max_iter=4000)) # Stochastic Average Gradient descent
])

params = {'logreg__C': [0.01], #TESTED np.logspace(start=-5, stop=3, num=9)
         'logreg__solver': ['saga'],
         'logreg__penalty':['l2'],
         'logreg__max_iter': [4000]}

logreg_model = GridSearchCV(model_pipeline_logreg, params,n_jobs=5, cv=5,scoring='roc_auc', refit=True,verbose=3)
logreg_model.fit(X_train, y_train)
print('Best roc_auc_score: {:.4}, with best C: {}'.format(logreg_model.best_score_, logreg_model.best_params_['logreg__C']))
```

>>> Best roc_auc_score: 0.9586, with best C: 0.01

Decision Tree

```
model_pipeline_tree = Pipeline([
    ('tree', tree.DecisionTreeClassifier(max_depth=10, max_features='sqrt'))
])

params = {'tree__max_features': ['sqrt'],
          'tree__max_depth': [10], # TESTED [5, 10, 15, None]
         }
tree_model = GridSearchCV(model_pipeline_tree, params, n_jobs=5, cv=5, scoring='roc_auc', refit=True,verbose=3)
tree_model.fit(X_train, y_train)
print('Best roc_auc_score: {:.4}, with best Max Features: {} and Max Depth: {}'.format(tree_model.best_score_,
                                                          tree_model.best_params_['tree__max_features'],
                                                         tree_model.best_params_['tree__max_depth'],
                                                         ))
```

>>> Best roc_auc_score: 0.9297, with best Max Features: sqrt and Max Depth: 10

Bagging Trees

```
model_pipeline_bag = Pipeline([
    ('bag', BaggingClassifier(tree.DecisionTreeClassifier(max_features='sqrt', max_depth=10)))
])

params = {'bag__n_estimators': [500], # TESTED [250, 500]
          'bag__max_samples': [0.8]
         }

bag_model = GridSearchCV(model_pipeline_bag, params, n_jobs=5, cv=5,
                         scoring='roc_auc', refit=True,verbose=3)
```

```
bag_model.fit(X_train, y_train)
print('Best roc_auc_score: {:.4}, with best n estimators: {}'.format(bag_model.best_score_,
                                                                     bag_model.best_params_['bag__n_estimators']
                                                                    ))
```

>>> Best roc_auc_score: 0.9583, with best n estimators: 500

Random Forest

```
model_pipeline_rf = Pipeline([
    ('forest', RandomForestClassifier())
])

params = {'forest__n_estimators': [500], # [250, 500]
          'forest__max_samples': [0.8],
          'forest__max_features': ["sqrt"],
          'forest__min_samples_split': [2]
         }

rf_model = GridSearchCV(model_pipeline_rf,
                        params, cv=5, n_jobs=5,
                        scoring='roc_auc', refit=True,verbose=3)
rf_model.fit(X_train, y_train)
print('Best roc_auc_score: {:.4}, with best n estimators: {}'.format(rf_model.best_score_,
                                                                     rf_model.best_params_['forest__n_estimators']
                                                                    ))
```

>>> Best roc_auc_score: 0.9595, with best n estimators: 500

Test Results

```
# Use each model to predict on the testing data.
logreg_ypred = logreg_model.predict_proba(X_test)[:,1]
tree_ypred = tree_model.predict_proba(X_test)[:,1]
bagging_ypred = bag_model.predict_proba(X_test)[:,1]
rf_ypred = rf_model.predict_proba(X_test)[:,1]
```

Build table of results

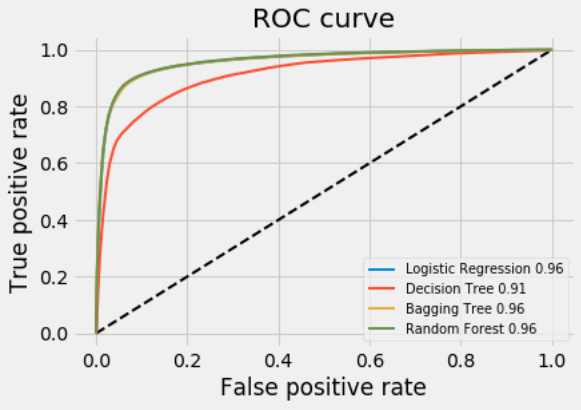
```
# Record the test accuracy
results = {'Model':['Logistic Regression',
                    'Decision Tree',
                    'Bagging Tree',
                    'Random Forest',
                   ],
           'Hypertuned Parameters':[logreg_model.best_params_,
                                    tree_model.best_params_,
                                    bag_model.best_params_,
                                    rf_model.best_params_
                                    ],
           'CV Training AUC':[logreg_model.best_score_,
                              tree_model.best_score_,
                              bag_model.best_score_,
                              rf_model.best_score_
                                ],
          'Test AUC':[roc_auc_score(y_test, logreg_model.predict_proba(X_test)[:,1]),
                      roc_auc_score(y_test, tree_model.predict_proba(X_test)[:,1]),
                      roc_auc_score(y_test, bag_model.predict_proba(X_test)[:,1]),
                      roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1])
                     ],
           'Test Accuracy':[accuracy_score(y_test, logreg_model.predict(X_test)),
                      accuracy_score(y_test, tree_model.predict(X_test)),
                      accuracy_score(y_test, bag_model.predict(X_test)),
                      accuracy_score(y_test, rf_model.predict(X_test)),
```

```
                        ],
            'Precision':[precision_score(y_test, logreg_model.predict(X_test)),
                         precision_score(y_test, tree_model.predict(X_test)),
                         precision_score(y_test, bag_model.predict(X_test)),
                         precision_score(y_test, rf_model.predict(X_test)),
                         ]}

# Save Model Results to CSV file
results_table = pd.DataFrame(results)
results_table.to_csv('Model_results.csv')
results_table
```

	Model	Hypertuned Parameters	CV Training AUC	Test AUC	Test Accuracy	Precision
0	Logistic Regression	{'logreg__C': 0.01, 'logreg__max_iter': 4000, ...	0.958582	0.958443	0.925686	0.880531
1	Decision Tree	{'tree__max_depth': 10, 'tree__max_features': ...	0.926333	0.910979	0.885488	0.832935
2	Bagging Tree	{'bag__max_samples': 0.8, 'bag__n_estimators':...	0.958311	0.958196	0.924132	0.865290
3	Random Forest	{'forest__max_features': 'sqrt', 'forest__max_...	0.959562	0.959938	0.926403	0.868110

Plot ROC Curve

```
fpr_lr, tpr_lr, _ = roc_curve(y_test, logreg_ypred)
fpr_tree, tpr_tree, _ = roc_curve(y_test, tree_ypred)
fpr_bag, tpr_bag, _ = roc_curve(y_test, bagging_ypred)
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_ypred)


plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--',lw=2.0)
plt.plot(fpr_lr, tpr_lr, lw=2.0,label=('Logistic Regression {:.2}'.format(roc_auc_score(y_test, logreg_ypred))))
plt.plot(fpr_tree, tpr_tree, lw=2.0, label='Decision Tree {:.2}'.format(roc_auc_score(y_test, tree_ypred)))
plt.plot(fpr_bag, tpr_bag,lw=2.0, label='Bagging Tree {:.2}'.format(roc_auc_score(y_test, bagging_ypred)))
plt.plot(fpr_rf, tpr_rf,lw=2.0, label='Random Forest {:.2}'.format(roc_auc_score(y_test, rf_ypred)))


plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best',fontsize='x-small')
plt.savefig('confusion_matrix.png', transparent=True)
plt.show()
```



Portfolio Selection: Returns Analysis

Analyze average return rate and ROI

- Full code can be found in Model_fitting.ipynb but here are the results.

```
-- Total Returns --
Naive Approach
Average return rate: 0.846
Total ROI: 0.816
------
Logistic Regression
Average return rate: 1.06
Total ROI: 1.05
------
Decision Tree
Average return rate: 0.981
Total ROI: 0.959
------
Bagging Tree
Average return rate: 1.06
Total ROI: 1.05
------
Random Forest
Average return rate: 1.06
Total ROI: 1.06
------
-- G Grade --
Naive Approach
Average return rate: 0.625
Total ROI: 0.61
------
Logistic Regression
Average return rate: 1.12
Total ROI: 1.13
------
Decision Tree
Average return rate: 0.843
Total ROI: 0.822
------
Bagging Tree
Average return rate: 1.15
Total ROI: 1.15
------
Random Forest
Average return rate: 1.15
Total ROI: 1.15
------
-- F Grade --
Naive Approach
Average return rate: 0.648
Total ROI: 0.624
------
Logistic Regression
Average return rate: 1.14
Total ROI: 1.15
------
Decision Tree
Average return rate: 0.88
Total ROI: 0.855
------
Bagging Tree
Average return rate: 1.16
Total ROI: 1.16
------
Random Forest
Average return rate: 1.17
Total ROI: 1.17
------
-- D Grade --
Naive Approach
Average return rate: 0.737
Total ROI: 0.706------
Logistic Regression
Average return rate: 1.08
Total ROI: 1.07------
Decision Tree
Average return rate: 0.935
```

```
Total ROI: 0.901------
Bagging Tree
Average return rate: 1.08
Total ROI: 1.07------
Random Forest
Average return rate: 1.09
Total ROI: 1.08------
-- C Grade --
Naive Approach
Average return rate: 0.806
Total ROI: 0.777
------
Logistic Regression
Average return rate: 1.06
Total ROI: 1.06
------
Decision Tree
Average return rate: 0.968
Total ROI: 0.943
------
Bagging Tree
Average return rate: 1.07
Total ROI: 1.06
------
Random Forest
Average return rate: 1.07
Total ROI: 1.06
------
-- B Grade --
Naive Approach
Average return rate: 0.897
Total ROI: 0.874
------
Logistic Regression
Average return rate: 1.05
Total ROI: 1.05
------
Decision Tree
Average return rate: 1.0
Total ROI: 0.987
------
Bagging Tree
Average return rate: 1.05
Total ROI: 1.04
------
Random Forest
Average return rate: 1.06
Total ROI: 1.05
------
-- A Grade --
Naive Approach
Average return rate: 0.969
Total ROI: 0.957
------
Logistic Regression
Average return rate: 1.04
Total ROI: 1.04
------
Decision Tree
Average return rate: 1.02
Total ROI: 1.01
------
Bagging Tree
Average return rate: 1.04
Total ROI: 1.03
------
Random Forest
Average return rate: 1.04
Total ROI: 1.04
------
-- F & G Grade --
Naive Approach
Average return rate: 0.64
Total ROI: 0.619
------
Logistic Regression
Average return rate: 1.13
Total ROI: 1.14
------
Decision Tree
Average return rate: 0.868
```

```
Total ROI: 0.844
------
Bagging Tree
Average return rate: 1.16
Total ROI: 1.15
------
Random Forest
Average return rate: 1.16
Total ROI: 1.16
------
```