BUSINESS INTELLIGENCE TERM PROJECT

Ekin Silahlioglu (116200078)

İbrahim Ethem Karalı (115200058)

İbrahim Doğan (11512112)

Computer Engineering

CMPE 343 - Spring 2020

Course Teacher: Savaş Yıldırım

Faculty of Engineering and Natural Sciences

Istanbul Bilgi University

Loan Prediction Problem

Aim

There are lots of Business Intelligence problem that needs to be solved. And one of them is taking a loan from a bank. Lots of people taking a loan for various reasons such as health, education, accommodation,… Thus banks need to decide whether they should give loans to a particular person according to some criteria for instance education level, income, debt amount. Not all people are eligible for taking a loan because when you take the loan you need to repay it.

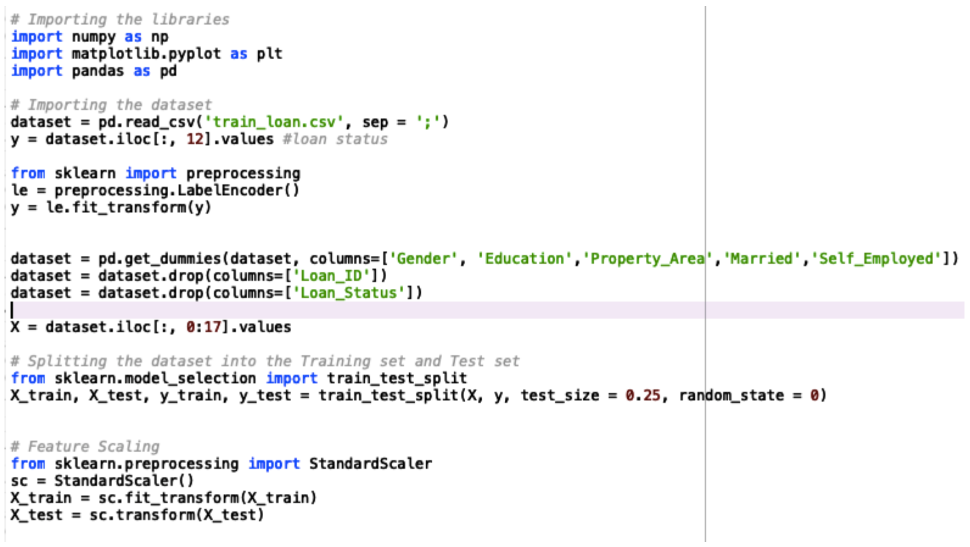
We are going to use the Loan Prediction Problem dataset. We aim to make the machine understand in which conditions can a person take a loan from a bank.

Dataset

Our Loan Prediction dataset consists of 13 attributes. These are loan id, gender, married status, dependents, education, self-employed, applicant income, co-applicant income, loan amount, loan amount term, credit history, property area, loan status. Loan status has two classes; Yes or No so in the end machine is going to predict whether it is Yes (can take a loan) or No (can not take a loan).

Preprocessing Part

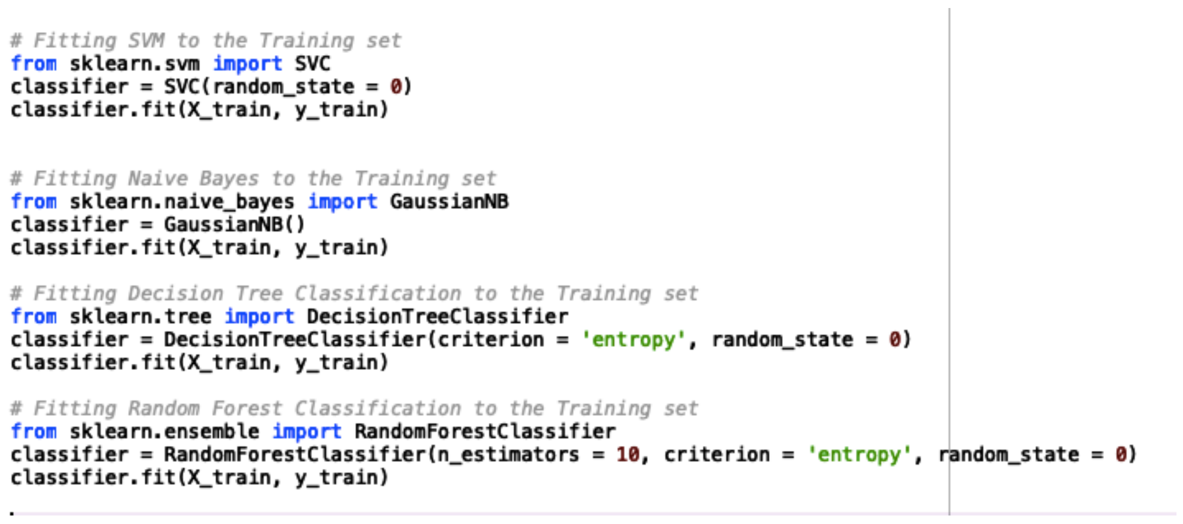
To begin with, we preprocessed our loan prediction data. We transformed all columns with a string value to an integer value by creating dummy variables. Also as our output consists of two classes, we used Label Encoder to turn Yes and No values into 1s and 0s. Moreover, most of the integer values in various columns have a different range so, in order to prevent particular columns more significant, we used Standard Scaler to put all our values in a particular range.



```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('train_loan.csv', sep = ';')
y = dataset.iloc[:, 12].values #loan status

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)


dataset = pd.get_dummies(dataset, columns=['Gender', 'Education','Property_Area','Married','Self_Employed'])
dataset = dataset.drop(columns=['Loan_ID'])
dataset = dataset.drop(columns=['Loan_Status'])

X = dataset.iloc[:, 0:17].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)


# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Model Training

We used four different machine learning algorithms to assess our problem. These are Support Vector Machine (SVM), Naive Bayes, Decision Tree, and Random Forest Classification. We used a 20% ratio for train and test split data. As we are going to see in the next section the Support Vector Machine gave the highest score among the four of them. We tried to use PCA and Grid Search Algorithms to find better accuracies. However, the main objective of PCA is to create new attributes according to the importance of attributes and reduced information, and because of the lack of information, our accuracy decreases drastically. Moreover, we used Grid Search, we realized that we had already used optimal parameters and the parameters that Grid Search algorithms provide us actually lowering our accuracy.



```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(random_state = 0)
classifier.fit(X_train, y_train)


# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Evaluation

We see that the Support Vector Machine exceeds all four algorithms in every score metric. The worst accuracy comes from the Decision Tree. The accuracy of Naive Bayes and SVM is very close but SVM classifies our problem in a better way.

ALGORTIHMS	Precision Score	Recall Score	F1 Score	Accuracy
Support Vector Machine	0.8192	0.9855	0.8947	0.8383
Naive Bayes	0.8227	0.9420	0.8783	0.8181
Decision Tree	0.8064	0.7246	0.7633	0.6868
Random Forest Classification	0.8550	0.8550	0.8550	0.7979

We created a confusion matrix for our Support Vector Machine model. As seen below we can see that for the test data our model has 16 wrong predictions and 83 correct predictions.

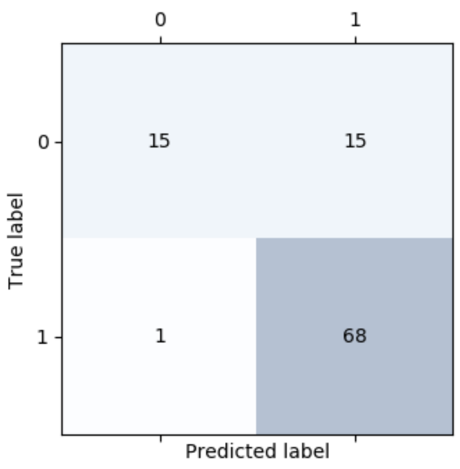


Table 1: Confusion Matrix

Data Visualization

In Table 2 below red ones are the predictions of the model and the blue ones are the real test values. On the y coordinate 1 is equal to Yes (can take Loan) and 0 is equal to No (can not take Loan). As mentioned in the evaluation part, there is little difference between predictions and real test values.

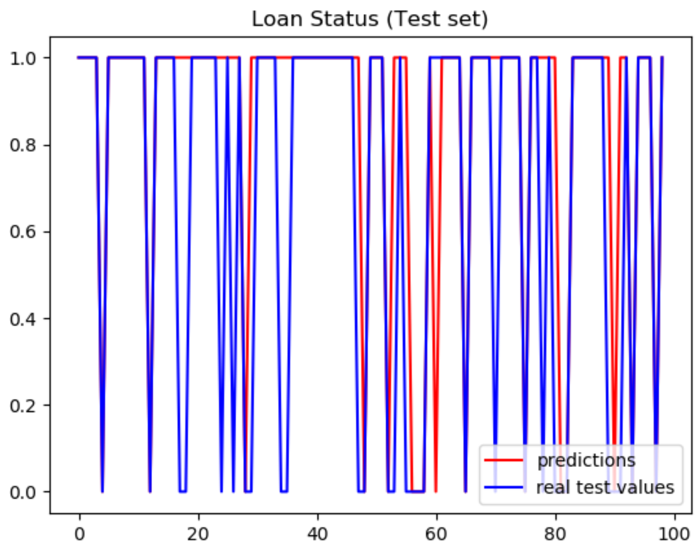


Table 2: Loan Status (Test Set)

In Table 3 we chose two attributes (Loan Amount and Applicant Income) to visualize our data. We can see that person who has low income, has a higher tendency to want to get loan. And some people who has low income, apply for a big loan amount.

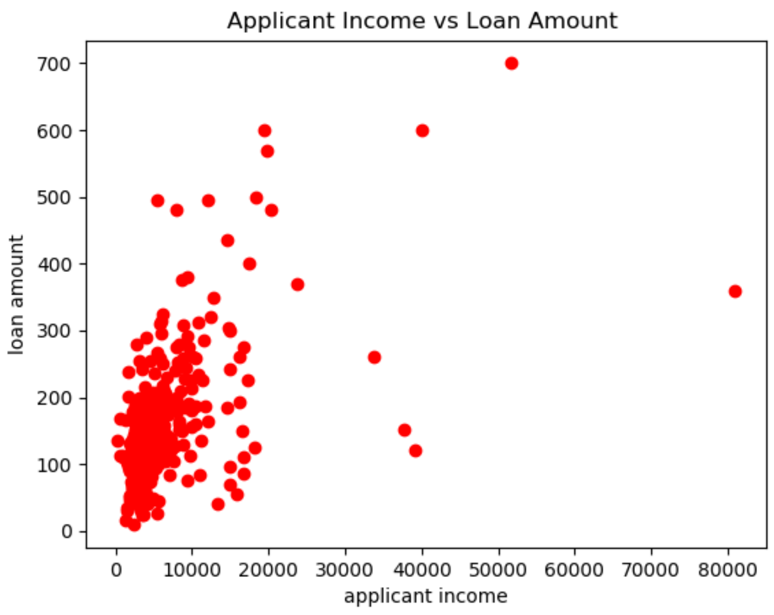
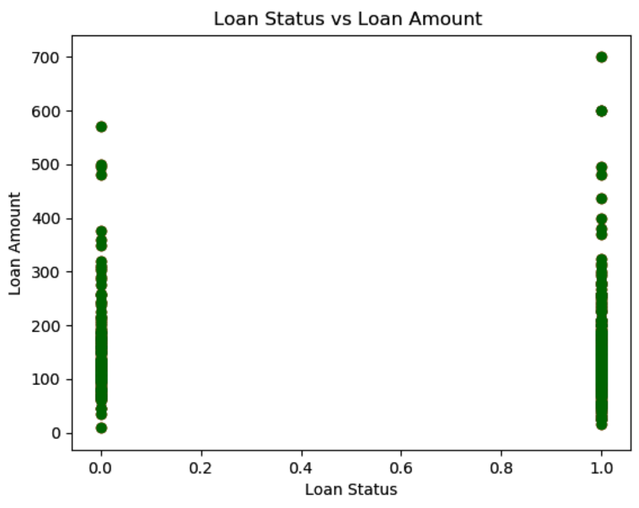
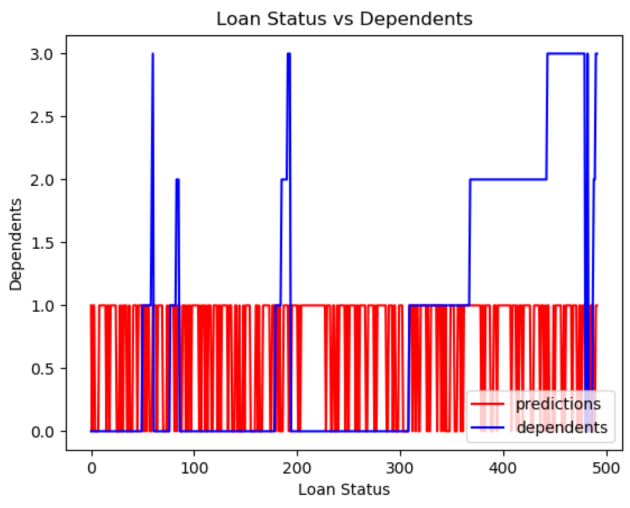


Table 3: Applicant Income vs Loan Amount

In below there are some visualizations for better understanding the data.





Conclusion

In conclusion, we tried four different algorithms and saw their results. We tested that the Support Vector Machine gave the best accuracy. We visualized our data to easily understand it. For future works, we can try to find more acceptable data to fit our model in order to get a higher accuracy.