SYSTEMS SOFTWARE

ASSIGNMENT

Predicting The Loan Approval Status

FALL 2018

Professor
Kaikai Liu

Date of Submission
23rd October 2018

Submitted by
Vineela Velicheti (Student ID: 012445186)

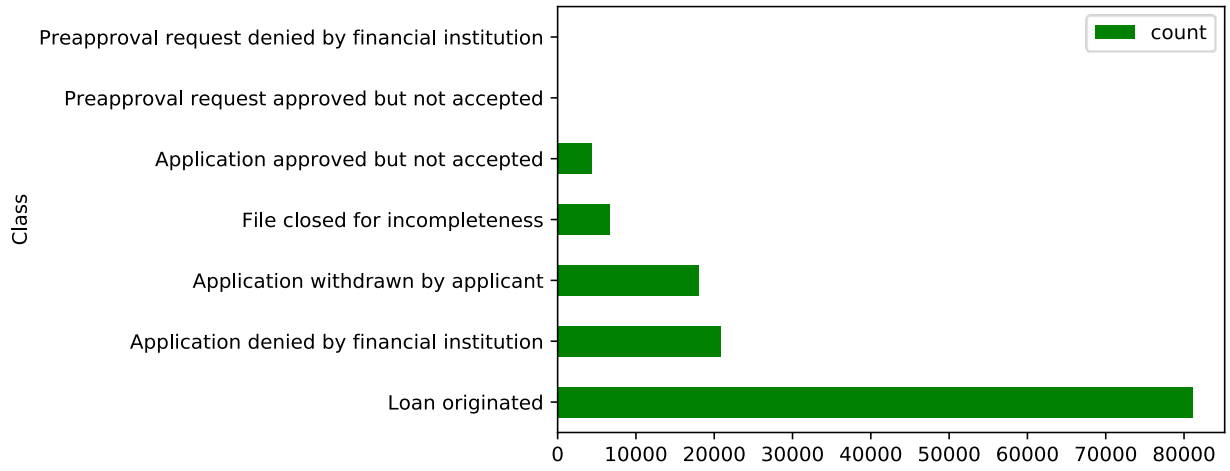
Objective:

The objective of this assignment is to analyze various loan applications from Home Mortgage Disclosure Act (HMDA) and predict the loan approval status. Various data mining techniques are used to carefully analyze each application and forecast the top features affecting the loan status.

Dataset:

The data used for this assignment is obtained from the Consumer Financial Protection Bureau (CFPB) website. This website provides various types of data like mortgage loan, credit card etc. for years 2007 to 2017. CFPB provides an API to extract subsets of the whole data using various filters like year, state, county etc.

For the purpose of this assignment, the HMDA data of San Francisco, Santa Clara and Sacramento counties for the year 2017 is considered. It consists of total 130,955 records with 78 columns. The target column i.e. “Action_taken” has 7 classes. However, more than 50% of the records belong to “Loan originated” class. Below file shows the imbalance in the dataset.

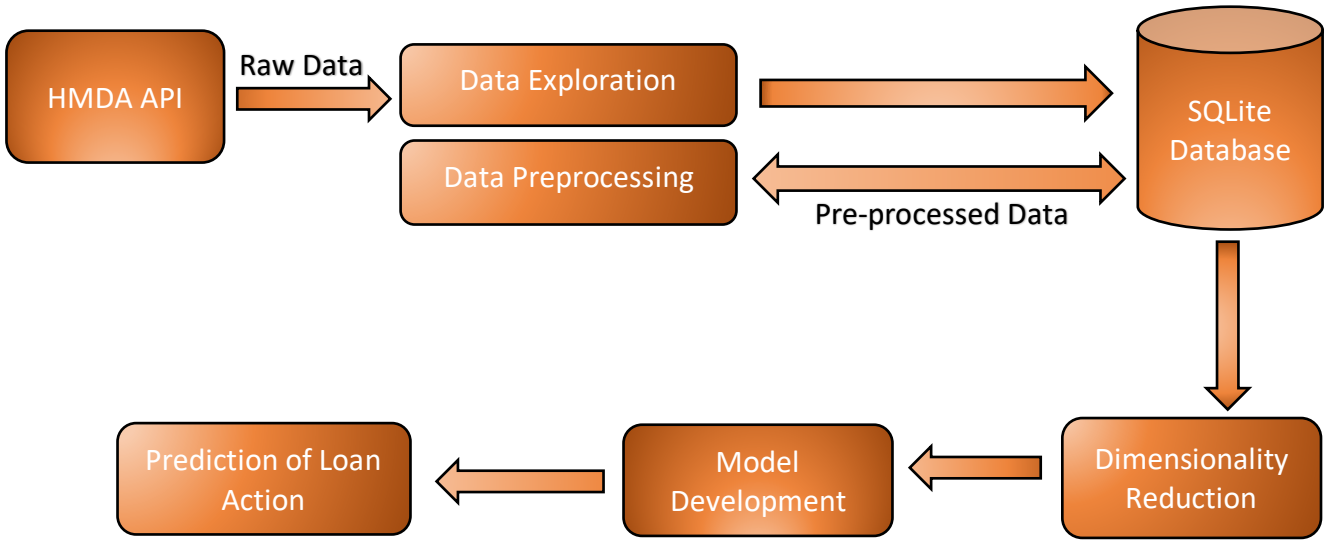


Tools and Packages used:

- Python (Version 2.7): Can be installed from www.python.org.
- Anaconda: Jupyter-Notebook for python programming
- SQLite: This comes inbuilt with python. The SQLite helps in storing the data internally in the database instead of saving them as files on local machine.

System Design & Implementation:

Architecture:



Data Ingestion: The HMDA website provides an API to get the data in various formats. For this assignment, the data is extracted in the form of a csv. The query language used in this API is based on Socrata's SoQL language. The below code uses the API link to extract the data in a csv format and stores it in a data frame.

```
mortgage_df =
pd.read_csv("https://api.consumerfinance.gov:443/data/hmda/slice/hmda_lar.csv?$where=as_of_year%3D2017+and+property_type+in+(1%2C2)+and+owner_occupancy%3D1+and+lien_status%3D1+and+state_abbr%3D'CA'+and+county_name+in+('San+Francisco+County'%2C'Santa+Clara+County'%2C'Sacramento+County')&$limit=10000000&$offset=0")
```

Data Exploration: In this step, the data is explored to understand the basic details like data type, missing values etc. Pandas provides a function "describe" to understand the descriptive statistics of all numeric features in a data frame. Also, the percentage of missing values is obtained for each feature and a total of 27 features are dropped as their missing value percentage is more than 50%. The modified data is then stored in the SQLite database using the below code snippet.

```
conn = sqlite3.connect("mortgage.db")
mortgage_df.to_sql("loan_data", conn, if_exists="replace")
```

Pre-Processing: The data stored in the SQLite database is retrieved into a data frame for further pre-processing.

- *Categorical to Numerical*: All the categorical features in the data frame are converted to numerical with the help of the one hot encoding technique. In this technique, new columns are created for each unique value in a categorical feature.
  For example: The agency_name is a categorical feature with 6 unique values. For each unique value, a new column will be added to the data frame leading to 6 new columns. Each row will have a 1 or 0 depending on their respective value.

  ```
  # Prints unique values in a column.
  unique_agency = loan_df_new['agency_name'].value_counts()
  print(unique_agency)

  # One – hot encoding to create new columns
  agency_dummy = pd.get_dummies(loan_df_new['agency_name'], prefix = 'agency')
  loan_df_new = pd.concat([loan_df_new,agency_dummy],axis=1)
  loan_df_new.shape
  ```

- Missing Values: In the exploration step, features with more than 50% missing values are dropped. However, the data frame still consists of features with lot of missing values. These values are now imputed with the mean of their respective attributes. For example, the below code can be used to impute the missing values in "applicant_income_000s" feature

  ```
  loan_df_fin['applicant_income_000s'] =
  loan_df_fin['applicant_income_000s'].fillna(loan_df_fin.applicant_income_000s.mean())
  ```

  After performing the pre-processing steps, the data frame is split into 80:20 ratio for train and test purposes. These data frames are stored into the database as four separate tables: X_train, Y_train, X_test, Y_test.

Dimensionality Reduction: To perform the dimensionality reduction step, 3 algorithms: Principal Component Analysis (PCA), Singular Value Decomposition (SVD), Variance Threshold were chosen. Three separate python scripts (mentioned below) are implemented to see how the performance differs in each case.

- **Prediction_PCA.py**: PCA with 10 classification Algorithms
- **Prediction_SVD.py:** SVD with 10 classification Algorithms
- **Prediction_Variance_Threshold.py:** Variance Threshold with 10 classification Algorithms.

Model Training: As part of this step, normalization and standardization techniques are applied to the data and then it is used to train the model. When PCA is used as the dimensionality reduction technique, only normalization is applied. This is because PCA implicitly standardizes the data. Below is the list of the classification algorithms chosen to implement with each dimensionality reduction technique.

Decision Tree	**Gradient Boost**
K-Nearest Neighbors (KNN)	**Random Forest**
Stochastic Gradient Descent (SGD)	**Extra Trees**
Naive Bayes	**Ada Boost**
Neural Networks (MLP)	**XGBoost**

Sklearn provides an easy way to train these classifiers. For example, to train a decision tree model the following steps of code can be used:

```
from sklearn import tree
dec = tree.DecisionTreeClassifier()
dec.fit(Train_PCA,Y_train)
pred_dec = dec.predict(Test_PCA)
```

Experimental Evaluation

Performance Measures:

Below performance measures are used for the experimental evaluation. The Sklearn.metrics module provides functions to calculate measures like accuracy, F-1 score, loss etc. for each classification model.

- ***Accuracy:*** The accuracy for a decision tree classifier can be determines using the below python code:

```
Acc_Dec = accuracy_score(Y_test, pred_dec)
print("Decision Tree Accuracy = ", Acc_Dec)
```

- ***F-1 Score:*** However, accuracy is not always a good measure to determine the performance of a model, especially when the data is imbalanced. In these cases, F-1 score can be used as a better evaluation technique. The F-1 score in sklearn has a parameter "average" which determines the type of averaging performed on the data. In this assignment, two types of averages: micro and weighted are implemented.

```
F1_Dec = f1_score(Y_test,pred_dec, average = 'micro')
print("Decision Tree F-1 score(micro) = ", F1_Dec)
F1W_Dec = f1_score(Y_test,pred_dec, average = 'weighted')
print("Decision Tree F-1 score(weighted) = ",F1W_Dec )
```

- ***Latency Time:*** The time required to train each model can be calculated using the below script:

```
start_time = time.clock()

dec = tree.DecisionTreeClassifier()
dec.fit(Train_PCA,Y_train)
pred_dec = dec.predict(Test_PCA)

Time_Dec = time.clock() - start_time
print("Time Taken in seconds:", Time_Dec)
```

- ***CPU Utilization:*** "psutil (python system and process utilities) is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in python". psutil.cpu_percent() is a function that returns a float representing the current system-wide CPU utilization as a percentage.

```
Cpu_Dec = psutil.cpu_percent()
print("Decision Tree Cpu Percent: ", Cpu_Dec)
```

- ***Memory Usage:*** The memory used for each process is retrieved with the help of the below python script.

```
def memory_usage_psutil():
        import psutil
        process = psutil.Process(os.getpid())
        mem = process.memory_info()[0] / float(2 ** 20)
        return mem
```

Algorithm Evaluation: As previously mentioned 3 dimensionality reduction algorithms are implemented across 10 classification algorithms leading to a total of 30 combinations. Below screenshot provides an idea on how the performance measures vary from technique to technique.

1. *Principal Component Analysis (PCA)*:

DR	PCA				
Classifier	F1 Score(Micro)	F1 score(Weighted)	Time(Secs)	CPU_Per	Mem_Usage
Decision Tree	0.98	0.98	10.03	41.6	513.49
KNN	0.62	0.48	31.06	43	543.89
SGD	0.62	0.48	1	49.9	528.46
Naïve Bayes	0.8	0.82	0.49	52.5	605.16
Neural Network (MLP)	0.62	0.48	9.9	42.7	608.7
Gradient Boost	0.68	0.6	192.52	38	576.13
Random Forest	0.98	0.98	120.32	95.2	624.73
Extra Trees	0.99	0.99	26.2	85.4	828.85
Adaboost	0.99	0.98	121.8	92.5	948.96
XGBoost	0.98	0.98	212.68	35.3	1062.9

2. *Singular Value Decomposition (SVD)*:

DR	SVD				
Classifier	F1 Score(Micro)	F1 score(Weighted)	Time(Secs)	CPU_Per	Mem_Usage
Decision Tree	0.98	0.98	10.66	36	588.17
KNN	0.62	0.48	30.7	46.2	522.13
SGD	0.62	0.48	0.95	42.1	522.89
Naïve Bayes	0.79	0.81	0.45	43.2	613.29
Neural Network (MLP)	0.62	0.48	9.08	42.9	622.74
Gradient Boost	0.68	0.61	218.8	34.8	591.19
Random Forest	0.99	0.99	117.53	95.9	635.85
Extra Trees	0.99	0.99	25.78	85.6	835.79
Adaboost	0.98	0.98	120.47	95.2	872.55
XGBoost	0.98	0.98	220.14	34.7	829.97

3. *Variance Threshold*:

DR	VarianceThreshold				
Classifier	F1 Score(Micro)	F1 Score(Weighted)	Time(Secs)	CPU_Per	Mem_Usage
Decision Tree	0.99	0.99	1.35	35.3	591.16
KNN	0.62	0.48	57.7	37.6	542.51
SGD	0.62	0.48	1.49	39.4	628.26
Naïve Bayes	0.77	0.78	0.6	41.6	682.53
Neural Network (MLP)	0.62	0.48	11.27	37.6	694.26
Gradient Boost	1	1	98.73	33.7	613.63
Random Forest	0.99	0.99	37.64	91	765.66
Extra Trees	0.99	0.99	25.79	84	1152.03
Adaboost	0.99	0.99	40.13	89.9	1276.18
XGBoost	0.99	0.99	215.9	34.7	1152.88

Result Analysis: Considering a trade-off between all the performance measures, Decision tree classifier with Variance Threshold as dimensionality Reduction technique can be considered as the best model. Following observations can also be made from the above results.

- Ensemble methods like Adaboost, Extra trees etc. have high latency time and memory usage.
- With variance threshold as DR technique, the memory used by XGBoost is almost double the memory used by Decision tree.
- Simple classifiers like Naïve Bayes, SGD exhibit least latency time. But, as a compromise the F-1 scores in these cases are very low compared to other algorithms.

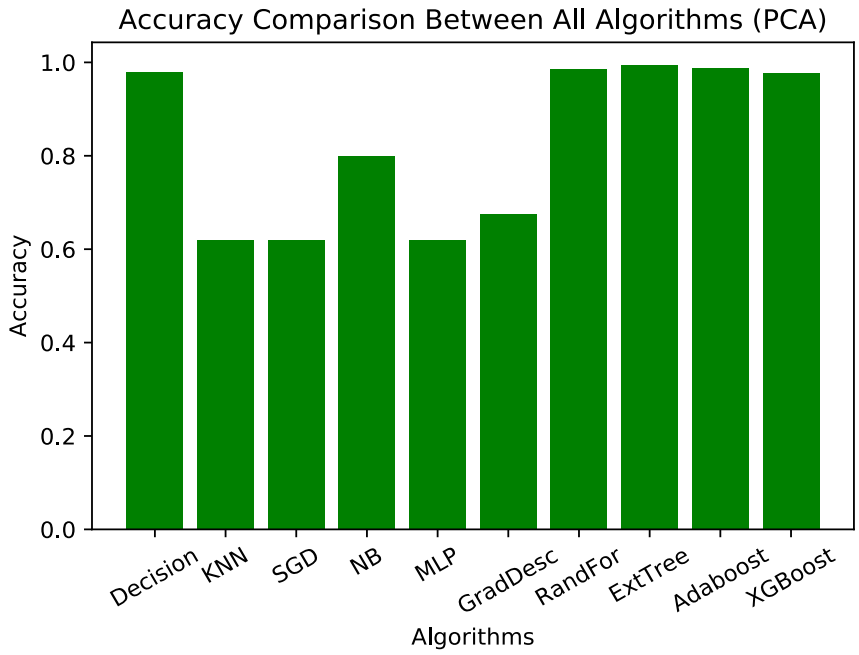
The top 10 features effecting the classifier are also extracted in the python script. Nevertheless, the features always differ with the classifier and the DR algorithm.

Visualizations:

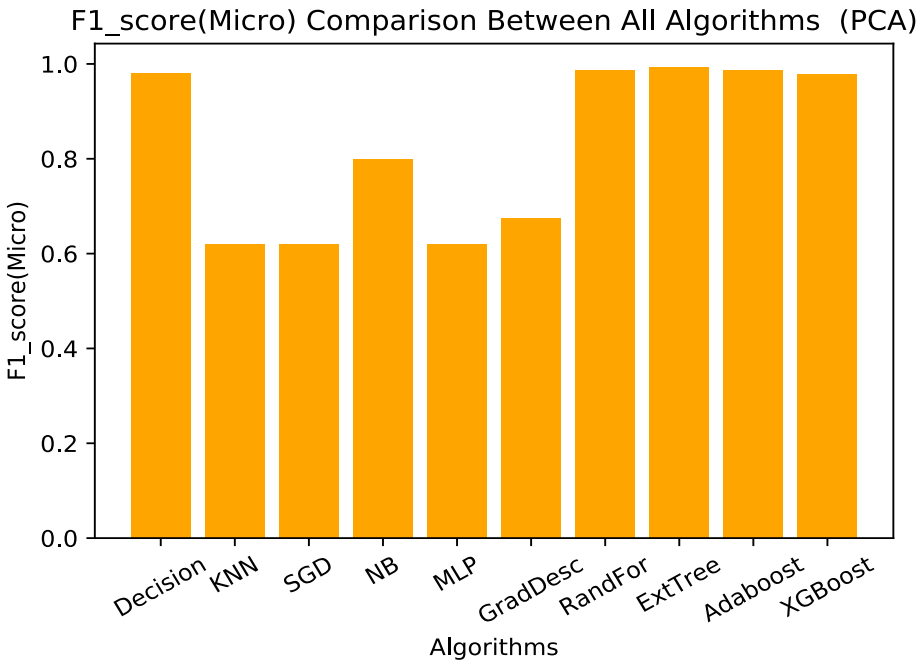
Below are the graphs showing various performance measures (for PCA) that are generated in python.

Principal Component Analysis (PCA):

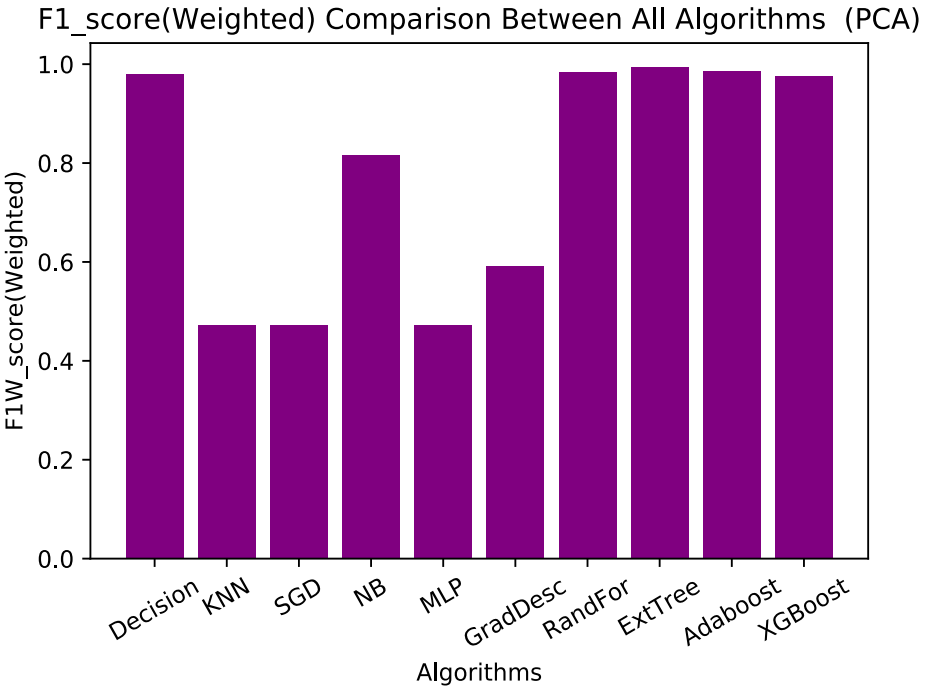
a. Accuracy:



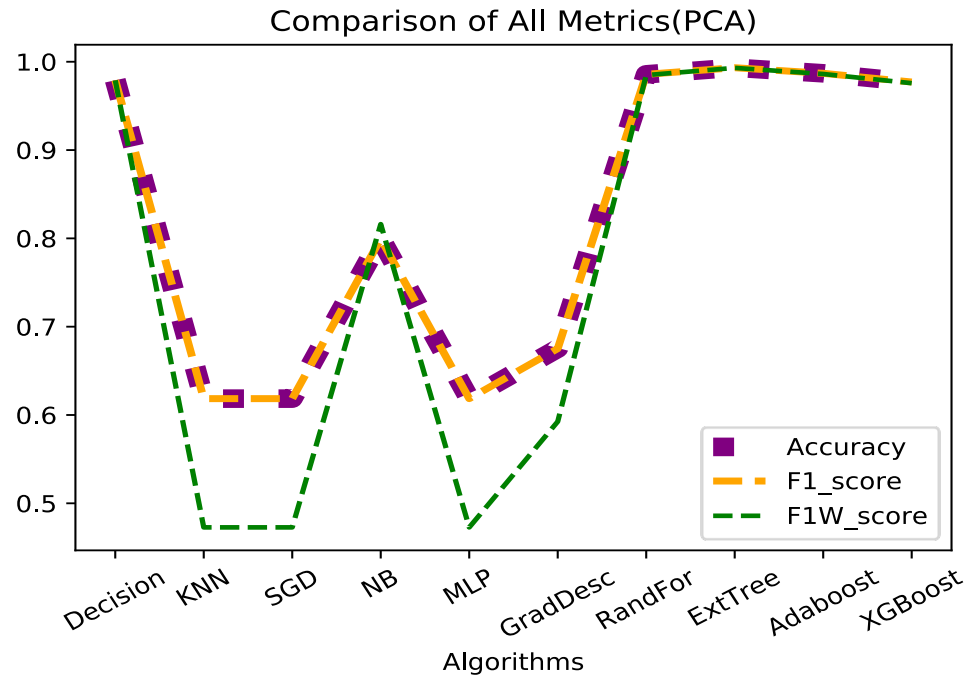
b. F-1 Score: (Micro)



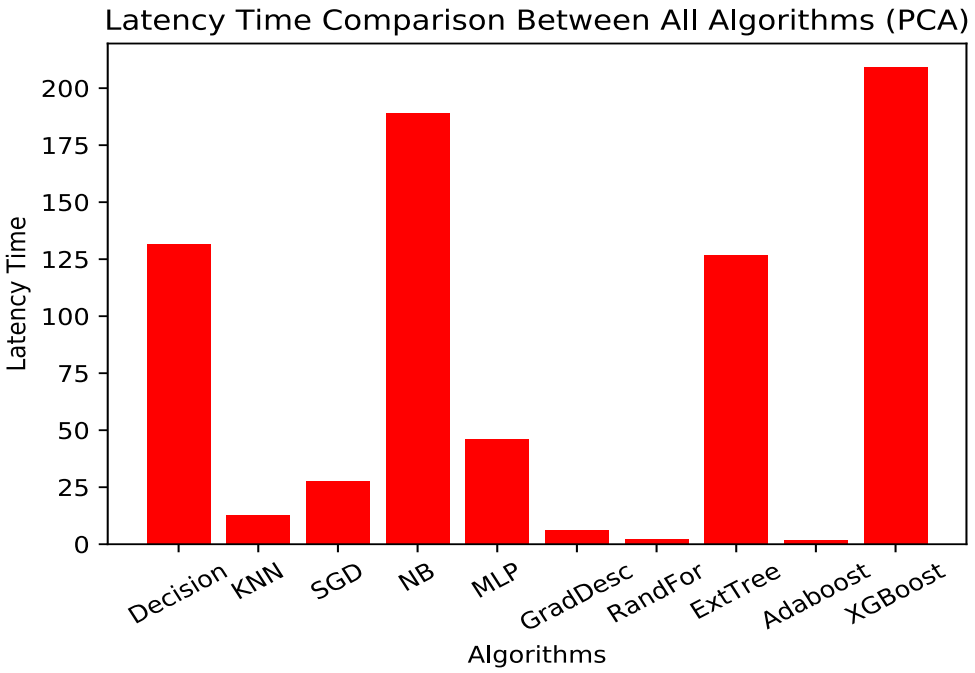
c. F-1 Score: (Weighted)



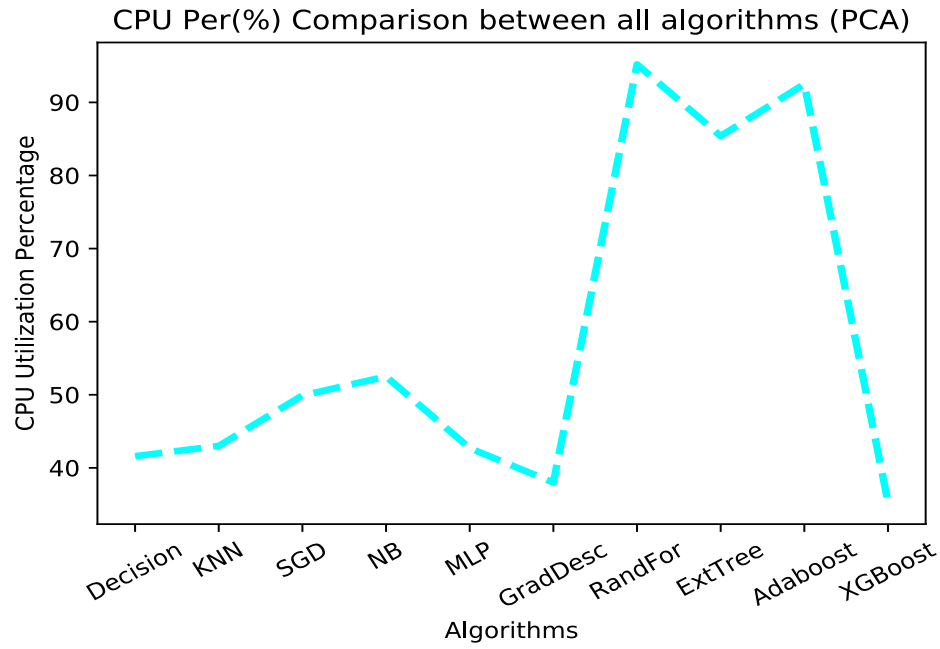
d. Comparison of All Together:



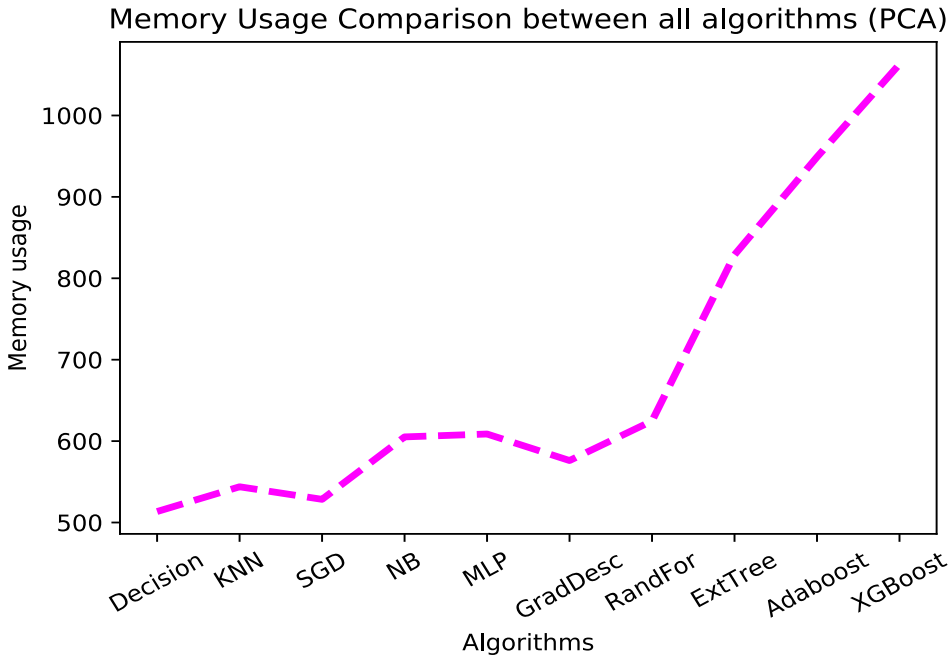
e. Latency Time:



f. CPU Utilization:



g. Memory Usage:



For Variance threshold and SVD scripts, similar images are generated and stored in the “Images” folder. They are not provided here as they are very similar to the above images.

References:

1. *https://www.consumerfinance.gov/data-research/hmda/explore*
2. *https://api.consumerfinance.gov/data/hmda/slice/hmda_lar.html?*
3. *https://sqlitebrowser.org/*
4. *https://psutil.readthedocs.io/en/latest/*
5. *http://fa.bianp.net/blog/2013/different-ways-to-get-memory-consumption-or-lessons-learned-from-memory_profiler/*
6. *https://python-graph-gallery.com/122-multiple-lines-chart/*
7. *http://scikit-learn.org/stable/modules/classes.html*