Loan Payment Prediction Model by Vishal Paul

Given data about loans, let's try to predict whether a given loan will be paid off or not.

We will use six different models to make our predictions.

Getting Started

In [18]:

```
import numpy as np
import os
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

In [19]:

```
pwd
```

Out[19]:

```
'C:\\Users\\as'
```

In [21]:

```
data = pd.read_csv( "Loan payments data.csv")
```

In [22]:

```
data
```

Out[22]:

	Loan_ID	loan_status	Principal	terms	effective_date	due_date	paid_off_time	past_due_days	age	education	Gender
0	xqd20166231	PAIDOFF	1000	30	9/8/2016	10/7/2016	9/14/2016 19:31	NaN	45	High School or Below	male
1	xqd20168902	PAIDOFF	1000	30	9/8/2016	10/7/2016	10/7/2016 9:00	NaN	50	Bechalor	female
2	xqd20160003	PAIDOFF	1000	30	9/8/2016	10/7/2016	9/25/2016 16:58	NaN	33	Bechalor	female
3	xqd20160004	PAIDOFF	1000	15	9/8/2016	9/22/2016	9/22/2016 20:00	NaN	27	college	male
4	xqd20160005	PAIDOFF	1000	30	9/9/2016	10/8/2016	9/23/2016 21:36	NaN	28	college	female
...	...	...	...	...	...	...	...	...	...	...	...
495	xqd20160496	COLLECTION_PAIDOFF	1000	30	9/12/2016	10/11/2016	10/14/2016 19:08	3.0	28	High School or Below	male
496	xqd20160497	COLLECTION_PAIDOFF	1000	15	9/12/2016	9/26/2016	10/10/2016 20:02	14.0	26	High School or Below	male
497	xqd20160498	COLLECTION_PAIDOFF	800	15	9/12/2016	9/26/2016	9/29/2016 11:49	3.0	30	college	male
498	xqd20160499	COLLECTION_PAIDOFF	1000	30	9/12/2016	11/10/2016	11/11/2016 22:40	1.0	38	college	female
499	xqd20160500	COLLECTION_PAIDOFF	1000	30	9/12/2016	10/11/2016	10/19/2016 11:58	8.0	28	High School or Below	male

500 rows × 11 columns

In [23]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Loan_ID         500 non-null    object
 1   loan_status     500 non-null    object
 2   Principal       500 non-null    int64
 3   terms           500 non-null    int64
 4   effective_date  500 non-null    object
 5   due_date        500 non-null    object
 6   paid_off_time   400 non-null    object
 7   past_due_days   200 non-null    float64
 8   age             500 non-null    int64
 9   education       500 non-null    object
 10  Gender          500 non-null    object
dtypes: float64(1), int64(3), object(7)
memory usage: 43.1+ KB
```

Preprocessing

In [24]:

```
data.isna().sum()
```

Out[24]:

```
Loan_ID            0
loan_status        0
Principal          0
terms              0
effective_date     0
due_date           0
paid_off_time    100
past_due_days    300
age                0
education          0
Gender             0
dtype: int64
```

In [25]:

```
data['loan_status'].unique()
```

Out[25]:

```
array(['PAIDOFF', 'COLLECTION', 'COLLECTION_PAIDOFF'], dtype=object)
```

In [26]:

```
{column: len(data[column].unique()) for column in data.columns}
```

Out[26]:

```
{'Loan_ID': 500,
 'loan_status': 3,
 'Principal': 6,
 'terms': 3,
 'effective_date': 7,
 'due_date': 25,
 'paid_off_time': 321,
 'past_due_days': 34,
 'age': 33,
 'education': 4,
 'Gender': 2}
```

In [27]:

```
def binary_encode(df, column, positive_value):
    df = df.copy()
    df[column] = df[column].apply(lambda x: 1 if x == positive_value else 0)
    return df

def ordinal_encode(df, column, ordering):
    df = df.copy()
    df[column] = df[column].apply(lambda x: ordering.index(x))
    return df
```

In [28]:

```
def preprocess_inputs(df):
    df = df.copy()

    # Drop Loan_ID column
    df = df.drop('Loan_ID', axis=1)

    # Create date/time columns
    for column in ['effective_date', 'due_date', 'paid_off_time']:
        df[column] = pd.to_datetime(df[column])

    df['effective_day'] = df['effective_date'].apply(lambda x: x.day)

    df['due_month'] = df['due_date'].apply(lambda x: x.month)
    df['due_day'] = df['due_date'].apply(lambda x: x.day)

    df['paid_off_month'] = df['paid_off_time'].apply(lambda x: x.month)
    df['paid_off_day'] = df['paid_off_time'].apply(lambda x: x.day)
    df['paid_off_hour'] = df['paid_off_time'].apply(lambda x: x.hour)

    df = df.drop(['effective_date', 'due_date', 'paid_off_time'], axis=1)

    # Fill missing values with column means
    for column in ['past_due_days', 'paid_off_month', 'paid_off_day', 'paid_off_hour']:
        df[column] = df[column].fillna(df[column].mean())

    # Binary encode the Gender column
    df = binary_encode(df, 'Gender', positive_value='male')

    # Ordinal encode the education column
    education_ordering = [
        'High School or Below',
        'college',
        'Bechalor',
        'Master or Above'
    ]
    df = ordinal_encode(df, 'education', ordering=education_ordering)

    # Encode the label (loan_status) column
    label_mapping = {'COLLECTION': 0, 'PAIDOFF': 1, 'COLLECTION_PAIDOFF': 2}
    df['loan_status'] = df['loan_status'].replace(label_mapping)

    # Split df into X and y
    y = df['loan_status'].copy()
    X = df.drop('loan_status', axis=1).copy()

    # Scale X with a standard scaler
    scaler = StandardScaler()
    X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

    return X, y
```

In [29]:

```
X, y = preprocess_inputs(data)
```

In [30]:

```
X
```

Out[30]:

	Principal	terms	past_due_days	age	education	Gender	effective_day	due_month	due_day	paid_off_month	paid_off_day	paid_off_hour
0	0.493377	0.897891	0.000000	2.284043	-1.022825	0.426653	-3.126073	0.664986	-1.303142	-1.035098	-0.463997	1.339835
1	0.493377	0.897891	0.000000	3.106587	1.771779	-2.343823	-3.126073	0.664986	-1.303142	0.690066	-1.475829	-1.072109
2	0.493377	0.897891	0.000000	0.309935	1.771779	-2.343823	-3.126073	0.664986	-1.303142	-1.035098	1.126025	0.616252
3	0.493377	-0.978972	0.000000	-0.677119	0.374477	0.426653	-3.126073	-1.094236	0.724148	-1.035098	0.692382	1.581030
4	0.493377	0.897891	0.000000	-0.512610	0.374477	-2.343823	-2.209336	0.664986	-1.167989	-1.035098	0.836930	1.822224
...	...	...	...	...	...	...	...	...	...	...	...	...
495	0.493377	0.897891	-1.780899	-0.512610	-1.022825	0.426653	0.540875	0.664986	-0.762531	0.690066	-0.463997	1.339835
496	0.493377	-0.978972	-1.187446	-0.841628	-1.022825	0.426653	0.540875	-1.094236	1.264758	0.690066	-1.042187	1.581030
497	-1.243866	-0.978972	-1.780899	-0.183592	0.374477	0.426653	0.540875	-1.094236	1.264758	-1.035098	1.704214	-0.589721
498	0.493377	0.897891	-1.888799	1.132480	0.374477	-2.343823	0.540875	2.424209	-0.897684	2.415229	-0.897640	2.063419
499	0.493377	0.897891	-1.511147	-0.512610	-1.022825	0.426653	0.540875	0.664986	-0.762531	0.690066	0.258740	-0.589721

500 rows × 12 columns

In [31]:

```
y
```

Out[31]:

```
0      1
1      1
2      1
3      1
4      1
      ..
495    2
496    2
497    2
498    2
499    2
Name: loan_status, Length: 500, dtype: int64
```

Training

In [32]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=123)
```

In [33]:

```
models = [
    LogisticRegression(),
    SVC(),
    DecisionTreeClassifier(),
    MLPClassifier(),
    RandomForestClassifier(),
    XGBClassifier()
]

for model in models:
    model.fit(X_train, y_train)
```

```
c:\users\as\appdata\local\programs\python\python37\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:617: ConvergenceWarning: Stochastic Opti
mizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
c:\users\as\appdata\local\programs\python\python37\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier is deprec
ated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier
object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[15:35:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [34]:

```
model_names = [
    "   Logistic Regression",
    "Support Vector Machine",
    "         Decision Tree",
    "        Neural Network",
    "         Random Forest",
    "               XGBoost"
]

for model, name in zip(models, model_names):
    print(name + ": {:.4f}%".format(model.score(X_test, y_test) * 100))
```

```
   Logistic Regression: 98.6667%
Support Vector Machine: 98.6667%
         Decision Tree: 100.0000%
        Neural Network: 100.0000%
         Random Forest: 100.0000%
               XGBoost: 100.0000%
```

In [ ]:

In [ ]: