(https://www.bigdatauniversity.com)

Classification with Python

In this notebook we try to practice all the classification algorithms that we learned in this course.

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

Lets first load required libraries:

In [1]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About dataset

This dataset is about past loans. The **Loan_train.csv** data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

Field	Description
Loan_status	Whether a loan is paid off on in collection
Principal	Basic principal loan amount at the
Terms	Origination terms which can be weekly (7 days), biweekly, and monthly payoff schedule
Effective_date	When the loan got originated and took effects
Due_date	Since it's one-time payoff schedule, each loan has one single due date
Age	Age of applicant
Education	Education of applicant
Gender	The gender of applicant

Lets download the dataset

In [2]:
```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2019-07-09 22:30:28--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: ‘loan_train.csv’

loan_train.csv      100%[===================>]  22.56K  --.-KB/s    in 0.02s

2019-07-09 22:30:29 (1.04 MB/s) - ‘loan_train.csv’ saved [23101/23101]
```

Load Data From CSV File

In [3]:
```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[3]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bechalor
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college

In [4]:
```
df.shape
```

Out[4]: (346, 10)

Convert to date time object

In [5]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[5]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college

Data visualization and pre-processing

Let’s see how many of each class is in our data set

In [6]:
```
df['loan_status'].value_counts()
```

Out[6]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

260 people have paid off the loan on time while 86 have gone into collection

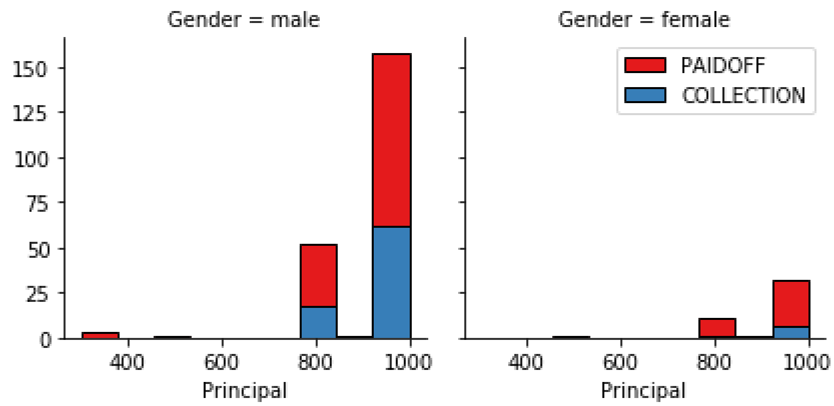
Lets plot some columns to underestand data better:

In [7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

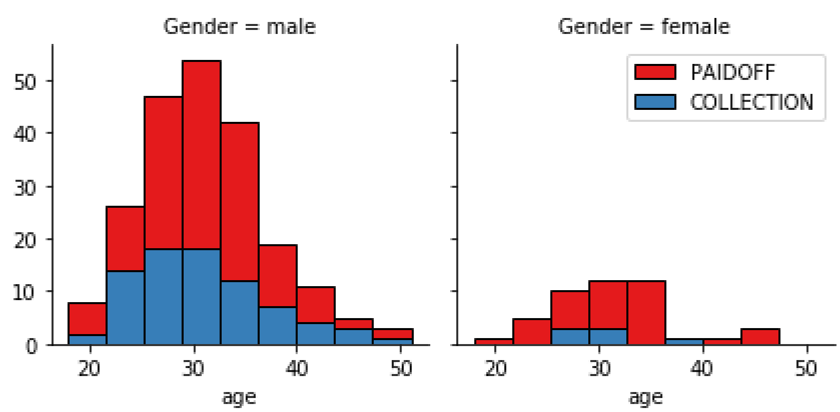
```
Collecting package metadata: done
Solving environment: -
The environment is inconsistent, please check the package plan carefully
The following packages are causing the inconsistency:

  - anaconda/linux-64::conda-build==3.17.8=py36_0
  - anaconda/linux-64::grpcio==1.16.1=py36hf8bcb03_1
  - anaconda/linux-64::keras==2.1.5=py36_0
  - anaconda/linux-64::libarchive==3.3.3=h5d8350f_5
  - anaconda/linux-64::python-libarchive-c==2.8=py36_6
  - anaconda/linux-64::tensorboard==1.8.0=py36hf484d3e_0
  - anaconda/linux-64::tensorflow==1.8.0=h57681fa_0
  - anaconda/linux-64::tensorflow-base==1.8.0=py36h5f64886_0
  - defaults/linux-64::anaconda==5.3.1=py37_0
  - defaults/linux-64::astropy==3.0.4=py37h14c3975_0
  - defaults/linux-64::bkcharts==0.2=py37_0
  - defaults/linux-64::blaze==0.11.3=py37_0
  - defaults/linux-64::bokeh==0.13.0=py37_0
  - defaults/linux-64::bottleneck==1.2.1=py37h035aef0_1
  - defaults/linux-64::dask==0.19.1=py37_0
  - defaults/linux-64::datashape==0.5.4=py37_1
  - defaults/linux-64::mkl-service==1.1.2=py37h90e4bf4_5
  - defaults/linux-64::numba==0.39.0=py37h04863e7_0
  - defaults/linux-64::numexpr==2.6.8=py37hd89afb7_0
  - defaults/linux-64::odo==0.5.1=py37_0
  - defaults/linux-64::pytables==3.4.4=py37ha205bf6_0
  - defaults/linux-64::pytest-arraydiff==0.2=py37h39e3cac_0
  - defaults/linux-64::pytest-astropy==0.4.0=py37_0
  - defaults/linux-64::pytest-doctestplus==0.1.3=py37_0
  - defaults/linux-64::pywavelets==1.0.0=py37hdd07704_0
  - defaults/linux-64::scikit-image==0.14.0=py37hf484d3e_1
done

## Package Plan ##

  environment location: /home/jupyterlab/conda

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2019.5.15  |                0         133 KB  anaconda
    certifi-2019.6.16          |           py36_0         154 KB  anaconda
    conda-4.7.5                |           py36_0         3.0 MB  anaconda
    conda-package-handling-1.3.10|           py36_0         259 KB  anaconda
    libtiff-4.0.10             |       h2733197_2         604 KB  anaconda
    python-libarchive-c-2.8    |           py36_9          22 KB  anaconda
    zstd-1.3.7                 |       h0b5b093_0         887 KB  anaconda
    ------------------------------------------------------------
                                           Total:         5.0 MB

The following NEW packages will be INSTALLED:
```

```
  conda-package-han~ anaconda/linux-64::conda-package-handling-1.3.10-py36_0

The following packages will be UPDATED:

  conda                                      4.6.14-py36_0 --> 4.7.5-py36_0
  openssl            conda-forge::openssl-1.1.1b-h14c3975_1 --> anaconda::ope
nssl-1.1.1-h7b6447c_0
  python-libarchive~                            2.8-py36_6 --> 2.8-py36_9

The following packages will be SUPERSEDED by a higher-priority channel:

  ca-certificates    conda-forge::ca-certificates-2019.6.1~ --> anaconda::ca-
certificates-2019.5.15-0
  certifi                                      conda-forge --> anaconda
  libtiff            conda-forge::libtiff-4.0.10-h57b8799_~ --> anaconda::lib
tiff-4.0.10-h2733197_2
  zstd                   conda-forge::zstd-1.4.0-h3b9ef0a_0 --> anaconda::zst
d-1.3.7-h0b5b093_0



Downloading and Extracting Packages
python-libarchive-c- | 22 KB     | ##################################### | 10
0%
libtiff-4.0.10       | 604 KB    | ##################################### | 10
0%
certifi-2019.6.16    | 154 KB    | ##################################### | 10
0%
zstd-1.3.7           | 887 KB    | ##################################### | 10
0%
conda-4.7.5          | 3.0 MB    | ##################################### | 10
0%
ca-certificates-2019 | 133 KB    | ##################################### | 10
0%
conda-package-handli | 259 KB    | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [8]:
```
import seaborn as sns

bins = np.linspace(df.Principal.min(), df.Principal.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wra
p=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```



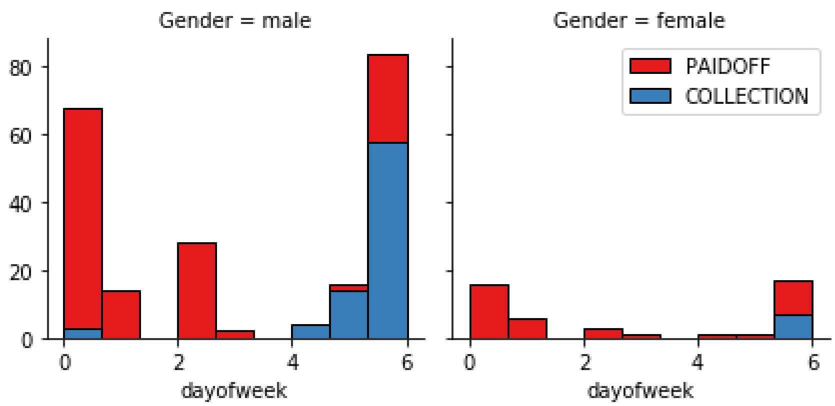
In [9]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wra
p=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```



Pre-processing: Feature selection/extraction

Lets look at the day of the week people get the loan

In [10]:
```
df['dayofweek'] = df['effective_date'].dt.dayofweek
bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wra
p=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



We see that people who get the loan at the end of the week dont pay it off, so lets use Feature binarization to set a threshold values less then day 4

In [11]:
```
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[11]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college

Convert Categorical features to numerical values

Lets look at gender:

In [12]:
```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[12]:
```
Gender  loan_status
female  PAIDOFF        0.865385
        COLLECTION     0.134615
male    PAIDOFF        0.731293
        COLLECTION     0.268707
Name: loan_status, dtype: float64
```

86 % of female pay there loans while only 73 % of males pay there loan

Lets convert male to 0 and female to 1:

In [13]:
```
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df.head()
```

Out[13]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college

One Hot Encoding

How about education?

In [14]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[14]:
```
education             loan_status
Bechalor              PAIDOFF        0.750000
                      COLLECTION     0.250000
High School or Below  PAIDOFF        0.741722
                      COLLECTION     0.258278
Master or Above       COLLECTION     0.500000
                      PAIDOFF        0.500000
college               PAIDOFF        0.765101
                      COLLECTION     0.234899
Name: loan_status, dtype: float64
```

Feature befor One Hot Encoding

In [15]:
```
df[['Principal','terms','age','Gender','education']].head()
```

Out[15]:

	Principal	terms	age	Gender	education
0	1000	30	45	0	High School or Below
1	1000	30	33	1	Bechalor
2	1000	15	27	0	college
3	1000	30	28	1	college
4	1000	30	29	0	college

Use one hot encoding technique to conver categorical varables to binary variables and append them to the feature Data Frame

In [16]:
```
Feature = df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(df['education'])], axis=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

Out[16]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

Feature selection

Lets defind feature sets, X:

In [17]:
```
X_train=Feature
X = Feature
X[0:5]
```

Out[17]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

What are our lables?

In [18]:
```
y_train = df['loan_status'].values
y = df['loan_status'].values
y[0:5]
```

Out[18]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

Normalize Data

Data Standardization give data zero mean and unit variance (technically should be done after train test split)

In [19]:
```
X_train= preprocessing.StandardScaler().fit(X).transform(X)
X_train[0:5]
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/preprocessing/dat
a.py:625: DataConversionWarning: Data with input dtype uint8, int64 were all
converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/jupyterlab/conda/lib/python3.6/site-packages/ipykernel_launcher.py:1: D
ataConversionWarning: Data with input dtype uint8, int64 were all converted t
o float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

Out[19]:
```
array([[ 0.51578458,  0.92071769,  2.33152555, -0.42056004, -1.20577805,
        -0.38170062,  1.13639374, -0.86968108],
       [ 0.51578458,  0.92071769,  0.34170148,  2.37778177, -1.20577805,
         2.61985426, -0.87997669, -0.86968108],
       [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.20577805,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.48739188,  2.37778177,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.3215732 , -0.42056004,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679]])
```

Classification

In [ ]:

Now, it is your turn, use the training set to build an accurate model. Then use the test set to report the accuracy of the model You should use the following algorithm:

- K Nearest Neighbor(KNN)
- Decision Tree
- Support Vector Machine
- Logistic Regression

Notice:

- You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model.
- You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms.
- You should include the code of the algorithm in the following cells.

K Nearest Neighbor(KNN)

Notice: You should find the best k to build the model with the best accuracy.
warning: You should not use the **loan_test.csv** for finding the best k, however, you can split your train_loan.csv into train and test to find the best **k**.

In [20]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [21]:
```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[21]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
```

In [ ]:

Decision Tree

In [22]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [23]:
```
loanTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
loanTree # it shows the default parameters
```

Out[23]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```

In [24]:
```
loanTree.fit(X_train,y_train)
```

Out[24]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```

Support Vector Machine

In [25]:
```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

Out[25]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [ ]:

In [ ]:

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[26]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
                   tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:

In [ ]:

Model Evaluation using Test set

In [27]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

First, download and load the test set:

In [28]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2019-07-09 22:46:39--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3642 (3.6K) [text/csv]
Saving to: ‘loan_test.csv’

loan_test.csv       100%[===================>]   3.56K  --.-KB/s    in 0s

2019-07-09 22:46:39 (57.1 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [36]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[36]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	1	1	PAIDOFF	1000	30	9/8/2016	10/7/2016	50	Bechalor
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	35	Master or Above
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	43	High School or Below
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	26	college
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	29	Bechalor

Subset the same variables, convert to dummy values and Normalize Data

In [37]:
```
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df.head()
```

Out[37]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07	50	Bechalor
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15	35	Master or Above
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09	43	High School or Below
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09	26	college
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25	29	Bechalor

In [38]:
```
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)

test_df.head()
```

Out[38]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07	50	Bechalor
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15	35	Master or Above
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09	43	High School or Below
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09	26	college
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25	29	Bechalor

In [39]:
```
Feat_t= test_df[['Principal','terms','age','Gender','weekend']] #'weekend'
Feat_t = pd.concat([Feat_t,pd.get_dummies(test_df['education'])], axis=1)
Feat_t.drop(['Master or Above'], axis = 1,inplace=True)
Feat_t.head()
```

Out[39]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	50	1	0	1	0	0
1	300	7	35	0	1	0	0	0
2	1000	30	43	1	1	0	1	0
3	1000	30	26	0	1	0	0	1
4	800	15	29	0	1	1	0	0

In [40]:
```
X_test=Feat_t
X_test= preprocessing.StandardScaler().fit(X_test).transform(X_test)
X_test[0:5]
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/preprocessing/dat
a.py:625: DataConversionWarning: Data with input dtype uint8, int64 were all
converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/jupyterlab/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: D
ataConversionWarning: Data with input dtype uint8, int64 were all converted t
o float64 by StandardScaler.
```

Out[40]:
```
array([[ 0.49362588,  0.92844966,  3.05981865,  1.97714211, -1.30384048,
         2.39791576, -0.79772404, -0.86135677],
       [-3.56269116, -1.70427745,  0.53336288, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404, -0.86135677],
       [ 0.49362588,  0.92844966,  1.88080596,  1.97714211,  0.76696499,
        -0.41702883,  1.25356634, -0.86135677],
       [ 0.49362588,  0.92844966, -0.98251057, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404,  1.16095912],
       [-0.66532184, -0.78854628, -0.47721942, -0.50578054,  0.76696499,
         2.39791576, -0.79772404, -0.86135677]])
```

In [41]:
```
y_test = test_df['loan_status'].values
y_test[0:5]
```

Out[41]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

KNN

In [42]:
```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[42]:
```
array(['PAIDOFF', 'COLLECTION', 'COLLECTION', 'COLLECTION', 'COLLECTION'],
      dtype=object)
```

In [45]:
```
print("Jaccar: ",jaccard_similarity_score(y_test, yhat))
print("F1 Score: ",f1_score(y_test, yhat, average='weighted'))
```

```
Jaccar:  0.6296296296296297
F1 Score:  0.6430311890838205
```

Decision Tree

In [47]:
```
print("Jaccar: ",jaccard_similarity_score(y_test,loanTree.predict(X_test)))
print("F1 Score: ",f1_score(y_test,loanTree.predict(X_test), average='weighte
d'))
```

```
Jaccar:  0.7777777777777778
F1 Score:  0.7283950617283951
```

SVM

In [48]:
```
print("Jaccar: ",jaccard_similarity_score(y_test,clf.predict(X_test)))
print("F1 Score: ",f1_score(y_test,clf.predict(X_test), average='weighted'))
```

```
Jaccar:  0.7222222222222222
F1 Score:  0.6212664277180406
```

In [ ]:

Logistic Regression

In [51]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob[0:5]
```

Out[51]:
```
array([[0.25256814, 0.74743186],
       [0.40233132, 0.59766868],
       [0.42774804, 0.57225196],
       [0.47276992, 0.52723008],
       [0.44726818, 0.55273182]])
```

In [52]:
```
print("Jaccar: ",jaccard_similarity_score(y_test,LR.predict(X_test)))
print("F1 Score: ",f1_score(y_test,LR.predict(X_test), average='weighted'))
print("Log Losss: ",log_loss(y_test, yhat_prob))
```

```
Jaccar:  0.7407407407407407
F1 Score:  0.6304176516942475
Log Losss:  0.5566084946309207

/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classifica
tion.py:1143: UndefinedMetricWarning: F-score is ill-defined and being set to
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Report

You should be able to report the accuracy of the built model using different evaluation metrics:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.62962	0.64303	NA
Decision Tree	0.77777	0.72839	NA
SVM	0.72727	0.62126	NA
LogisticRegression	0.74074	0.63041	0.55660

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler (http://cocl.us/ML0101EN-SPSSModeler)

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio (https://cocl.us/ML0101EN_DSX)

Thanks for completing this lesson!

Author: Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi)

Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi), PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients' ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.
