# Classification with Python

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

--2019-01-04 13:17:13-- 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’

100%[======================================>] 23,101 --.-K/s in 0.002s

2019-01-04 13:17:13 (14.0 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** | **Gender** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 45 | High School or Below | male |
| **1** | 2 | 2 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 33 | Bechalor | female |
| **2** | 3 | 3 | PAIDOFF | 1000 | 15 | 9/8/2016 | 9/22/2016 | 27 | college | male |
| **3** | 4 | 4 | PAIDOFF | 1000 | 30 | 9/9/2016 | 10/8/2016 | 28 | college | female |
| **4** | 6 | 6 | PAIDOFF | 1000 | 30 | 9/9/2016 | 10/8/2016 | 29 | college | male |

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** | **Gender** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male |
| **1** | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female |
| **2** | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male |
| **3** | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female |
| **4** | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male |

# 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

Solving environment: done

## Package Plan ##

environment location: /Users/Saeed/anaconda/envs/python3.6

added / updated specs:

- seaborn

The following packages will be downloaded:

package | build

---------------------------|-----------------

openssl-1.0.2o | h26aff7b\_0 3.4 MB anaconda

ca-certificates-2018.03.07 | 0 124 KB anaconda

------------------------------------------------------------

Total: 3.5 MB

The following packages will be UPDATED:

ca-certificates: 2018.03.07-0 --> 2018.03.07-0 anaconda

openssl: 1.0.2o-h26aff7b\_0 --> 1.0.2o-h26aff7b\_0 anaconda

Downloading and Extracting Packages

openssl-1.0.2o | 3.4 MB | ####################################### | 100%

ca-certificates-2018 | 124 KB | ####################################### | 100%

Preparing transaction: done

Verifying transaction: done

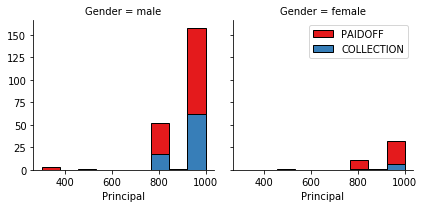
Executing transaction: done

In [7]:

**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\_wrap**=**2)g**.**map(plt**.**hist, 'Principal', bins**=**bins, ec**=**"k")

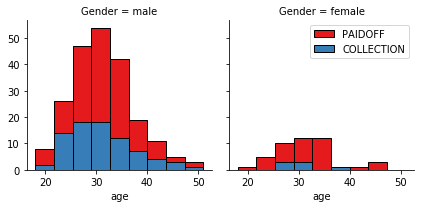
g**.**axes[**-**1]**.**legend()plt**.**show()



In [8]:

bins **=** np**.**linspace(df**.**age**.**min(), df**.**age**.**max(), 10)g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**2)g**.**map(plt**.**hist, 'age', bins**=**bins, ec**=**"k")

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

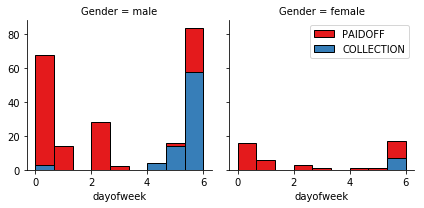


# Pre-processing: Feature selection/extraction

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

In [9]:

df['dayofweek'] **=** df['effective\_date']**.**dt**.**dayofweekbins **=** np**.**linspace(df**.**dayofweek**.**min(), df**.**dayofweek**.**max(), 10)g **=** sns**.**FacetGrid(df, col**=**"Gender", hue**=**"loan\_status", palette**=**"Set1", col\_wrap**=**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 [10]:

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

Out[10]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** | **dayofweek** | **weekend** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | male | 3 | 0 |
| **1** | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | female | 3 | 0 |
| **2** | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | male | 3 | 0 |
| **3** | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | female | 4 | 1 |
| **4** | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | male | 4 | 1 |

## Convert Categorical features to numerical values

Lets look at gender:

In [11]:

df**.**groupby(['Gender'])['loan\_status']**.**value\_counts(normalize**=True**)

Out[11]:

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 [12]:

df['Gender']**.**replace(to\_replace**=**['male','female'], value**=**[0,1],inplace**=True**)df**.**head()

Out[12]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** | **dayofweek** | **weekend** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 45 | High School or Below | 0 | 3 | 0 |
| **1** | 2 | 2 | PAIDOFF | 1000 | 30 | 2016-09-08 | 2016-10-07 | 33 | Bechalor | 1 | 3 | 0 |
| **2** | 3 | 3 | PAIDOFF | 1000 | 15 | 2016-09-08 | 2016-09-22 | 27 | college | 0 | 3 | 0 |
| **3** | 4 | 4 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 28 | college | 1 | 4 | 1 |
| **4** | 6 | 6 | PAIDOFF | 1000 | 30 | 2016-09-09 | 2016-10-08 | 29 | college | 0 | 4 | 1 |

## One Hot Encoding

#### How about education?

In [13]:

df**.**groupby(['education'])['loan\_status']**.**value\_counts(normalize**=True**)

Out[13]:

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 [14]:

df[['Principal','terms','age','Gender','education']]**.**head()

Out[14]:

|  | **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 [15]:

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[15]:

|  | **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 [16]:

X **=** FeatureX[0:5]

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 |

What are our lables?

In [17]:

y **=** df['loan\_status']**.**valuesy[0:5]

Out[17]:

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 [18]:

X**=** preprocessing**.**StandardScaler()**.**fit(X)**.**transform(X)X[0:5]

Out[18]:

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

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 [22]:

**from** sklearn.model\_selection **import** train\_test\_splitX\_train, X\_test, y\_train, y\_test **=** train\_test\_split( X, y, test\_size**=**0.2, random\_state**=**4)print ('Train set:', X\_train**.**shape, y\_train**.**shape)print ('Test set:', X\_test**.**shape, y\_test**.**shape)

Train set: (276, 8) (276,)

Test set: (70, 8) (70,)

In [23]:

**from** sklearn.neighbors **import** KNeighborsClassifierk **=** 6

neighK6 **=** KNeighborsClassifier(n\_neighbors **=** k)**.**fit(X\_train,y\_train)neighK6

yhat **=** neighK6**.**predict(X\_test)yhat[0:5]

**from** sklearn **import** metricsprint("Train set Accuracy: ", metrics**.**accuracy\_score(y\_train, neighK6**.**predict(X\_train)))print("Test set Accuracy: ", metrics**.**accuracy\_score(y\_test, yhat))

Train set Accuracy: 0.800724637681

Test set Accuracy: 0.714285714286

In [25]:

Ks **=** 10mean\_acc **=** np**.**zeros((Ks**-**1))std\_acc **=** np**.**zeros((Ks**-**1))

ConfustionMx **=** [];**for** n **in** range(1,Ks):

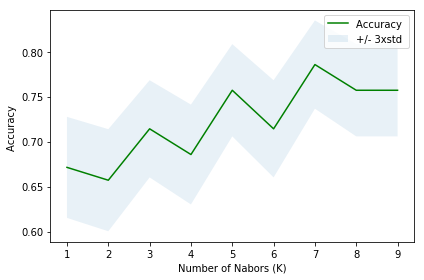
neigh **=** KNeighborsClassifier(n\_neighbors **=** n)**.**fit(X\_train,y\_train)

yhat**=**neigh**.**predict(X\_test)

mean\_acc[n**-**1] **=** metrics**.**accuracy\_score(y\_test, yhat)

std\_acc[n**-**1]**=**np**.**std(yhat**==**y\_test)**/**np**.**sqrt(yhat**.**shape[0])

plt**.**plot(range(1,Ks),mean\_acc,'g')plt**.**fill\_between(range(1,Ks),mean\_acc **-** 1 **\*** std\_acc,mean\_acc **+** 1 **\*** std\_acc, alpha**=**0.10)plt**.**legend(('Accuracy ', '+/- 3xstd'))plt**.**ylabel('Accuracy ')plt**.**xlabel('Number of Nabors (K)')plt**.**tight\_layout()plt**.**show()print( "Best accuracy:", mean\_acc**.**max(), "k=", mean\_acc**.**argmax()**+**1)



Best accuracy: 0.785714285714 k= 7

# Decision Tree

In [26]:

**from** sklearn.tree **import** DecisionTreeClassifier**from** sklearn.model\_selection **import** train\_test\_splitX\_trainset, X\_testset, y\_trainset, y\_testset **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**3)*#Modelling*Tree **=** DecisionTreeClassifier(criterion**=**"entropy", max\_depth **=** 6)Tree

Out[26]:

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=6,

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 [27]:

Tree**.**fit(X\_trainset,y\_trainset)

Out[27]:

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=6,

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 [28]:

predTree **=** Tree**.**predict(X\_testset)print (predTree [0:5])print (y\_testset [0:5])

**from** sklearn **import** metrics**import** matplotlib.pyplot **as** plt

print("Accuracy: ", metrics**.**accuracy\_score(y\_testset, predTree))

**!**conda install -c conda-forge pydotplus -y**!**conda install -c conda-forge python-graphviz -y**from** sklearn.externals.six **import** StringIO**import** pydotplus**import** matplotlib.image **as** mpimg**from** sklearn **import** tree**%matplotlib** inline dot\_data **=** StringIO()filename **=** "loan.png"featureNames **=** df**.**columns[0:8]targetNames **=** df['loan\_status']**.**unique()**.**tolist()out**=**tree**.**export\_graphviz(Tree,feature\_names**=**featureNames, out\_file**=**dot\_data, class\_names**=** np**.**unique(y\_trainset), filled**=True**, special\_characters**=True**,rotate**=False**) graph **=** pydotplus**.**graph\_from\_dot\_data(dot\_data**.**getvalue()) graph**.**write\_png(filename)img **=** mpimg**.**imread(filename)plt**.**figure(figsize**=**(100, 200))plt**.**imshow(img,interpolation**=**'nearest')

['PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF']

['PAIDOFF' 'PAIDOFF' 'COLLECTION' 'COLLECTION' 'PAIDOFF']

Accuracy: 0.701923076923

Fetching package metadata .............

Solving package specifications: .

Package plan for installation in environment /opt/conda/envs/DSX-Python35:

The following NEW packages will be INSTALLED:

pydotplus: 2.0.2-py\_2 conda-forge

pydotplus-2.0. 100% |################################| Time: 0:00:00 18.80 MB/s

Fetching package metadata .............

Solving package specifications: .

Package plan for installation in environment /opt/conda/envs/DSX-Python35:

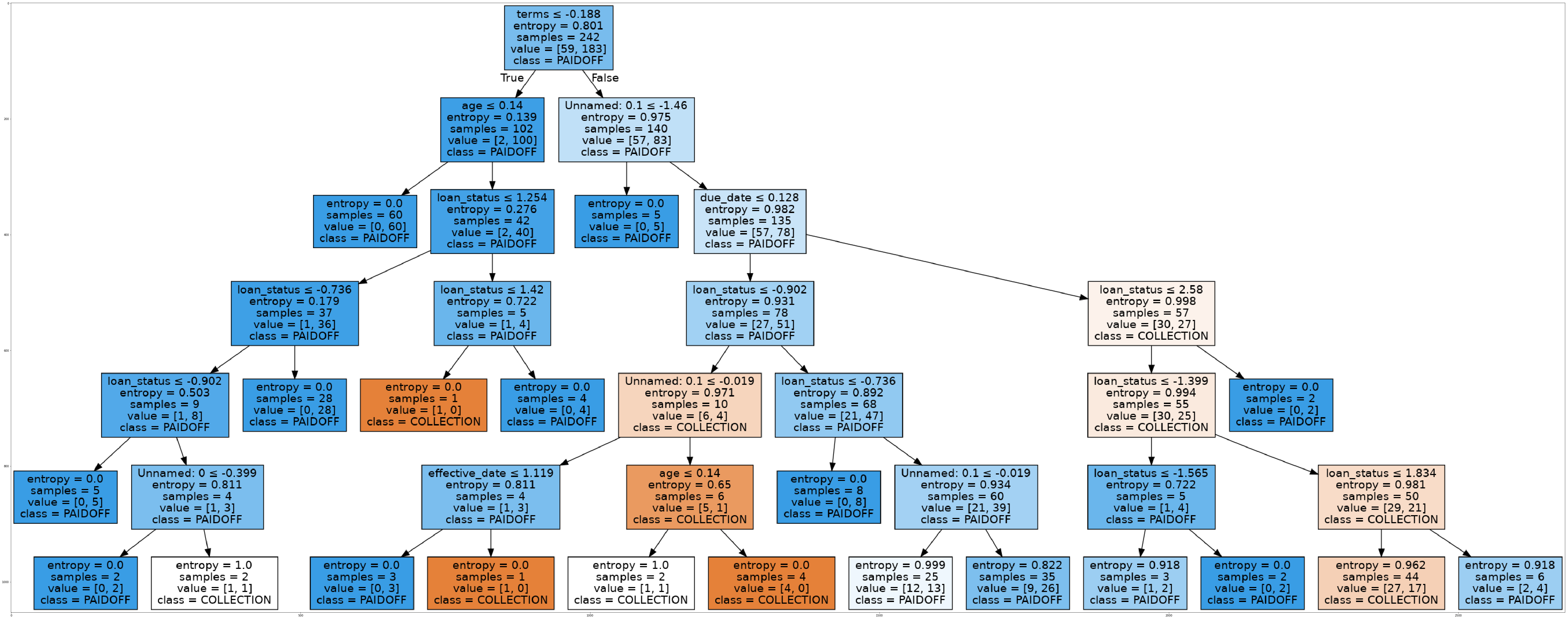
The following NEW packages will be INSTALLED:

python-graphviz: 0.8.4-py35\_2 conda-forge

python-graphvi 100% |################################| Time: 0:00:00 18.89 MB/s

Out[28]:

<matplotlib.image.AxesImage at 0x2af2b9463320>



# Support Vector Machine

In [29]:

df**.**dtypesdf **=** df[pd**.**to\_numeric(df['education'], errors**=**'coerce')**.**notnull()]df['education'] **=** df['education']**.**astype('int')df**.**dtypes

**from** sklearn **import** svmclf **=** svm**.**SVC(kernel**=**'rbf')clf**.**fit(X\_train, y\_train)

Out[29]:

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

In [30]:

yhat **=** clf**.**predict(X\_test)yhat [0:5]**from** sklearn.metrics **import** classification\_report, confusion\_matrix**import** itertools

In [34]:

**def** plot\_confusion\_matrix(cm, classes,

normalize**=False**,

title**=**'Confusion matrix',

cmap**=**plt**.**cm**.**Blues):

**if** normalize:

cm **=** cm**.**astype('float') **/** cm**.**sum(axis**=**1)[:, np**.**newaxis]

plt**.**imshow(cm, interpolation**=**'nearest', cmap**=**cmap)

plt**.**title(title)

plt**.**colorbar()

tick\_marks **=** np**.**arange(len(classes))

plt**.**xticks(tick\_marks, classes, rotation**=**45)

plt**.**yticks(tick\_marks, classes)

fmt **=** '.2f' **if** normalize **else** 'd'

thresh **=** cm**.**max() **/** 2.

**for** i, j **in** itertools**.**product(range(cm**.**shape[0]), range(cm**.**shape[1])):

plt**.**text(j, i, format(cm[i, j], fmt),

horizontalalignment**=**"center",

color**=**"white" **if** cm[i, j] **>** thresh **else** "black")

plt**.**tight\_layout()

plt**.**ylabel('True label')

plt**.**xlabel('Predicted label')cnf\_matrix **=** confusion\_matrix(y\_test, yhat, labels**=**['PAIDOFF','COLLECTION'])np**.**set\_printoptions(precision**=**2)

print (classification\_report(y\_test, yhat))

plt**.**figure()plot\_confusion\_matrix(cnf\_matrix, classes**=**['PAIDOFF','COLLECTION'],normalize**=** **False**, title**=**'Confusion matrix')

**from** sklearn.metrics **import** f1\_scoref1\_score(y\_test, yhat, average**=**'weighted')

**from** sklearn.metrics **import** jaccard\_similarity\_scorejaccard\_similarity\_score(y\_test, yhat)

precision recall f1-score support

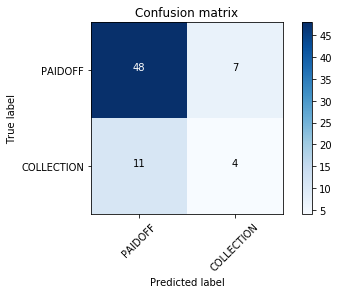
COLLECTION 0.36 0.27 0.31 15

PAIDOFF 0.81 0.87 0.84 55

avg / total 0.72 0.74 0.73 70

Out[34]:

0.74285714285714288



# Logistic Regression

In [35]:

df **=** df[['loan\_status', 'Principal', 'terms', 'effective\_date', 'due\_date', 'age', 'education', 'Gender']]df['loan\_status'] **=** df['loan\_status']**.**astype('int')

**from** sklearn **import** preprocessingX **=** preprocessing**.**StandardScaler()**.**fit(X)**.**transform(X)X[0:5]

**from** sklearn.model\_selection **import** train\_test\_splitX\_train, X\_test, y\_train, y\_test **=** train\_test\_split( X, y, test\_size**=**0.2, random\_state**=**4)print ('Train set:', X\_train**.**shape, y\_train**.**shape)print ('Test set:', X\_test**.**shape, y\_test**.**shape)

**from** sklearn.linear\_model **import** LogisticRegression**from** sklearn.metrics **import** confusion\_matrixLogR **=** LogisticRegression(C**=**0.01, solver**=**'liblinear')**.**fit(X\_train,y\_train)LogR

Train set: (276, 8) (276,)

Test set: (70, 8) (70,)

Out[35]:

LogisticRegression(C=0.01, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,

penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,

verbose=0, warm\_start=False)

In [36]:

yhat **=** LogR**.**predict(X\_test)yhatyhat\_prob **=** LogR**.**predict\_proba(X\_test)yhat\_prob

Out[36]:

array([[ 0.5 , 0.5 ],

[ 0.45, 0.55],

[ 0.31, 0.69],

[ 0.34, 0.66],

[ 0.32, 0.68],

[ 0.32, 0.68],

[ 0.49, 0.51],

[ 0.48, 0.52],

[ 0.34, 0.66],

[ 0.49, 0.51],

[ 0.34, 0.66],

[ 0.5 , 0.5 ],

[ 0.25, 0.75],

[ 0.34, 0.66],

[ 0.44, 0.56],

[ 0.26, 0.74],

[ 0.52, 0.48],

[ 0.3 , 0.7 ],

[ 0.5 , 0.5 ],

[ 0.32, 0.68],

[ 0.44, 0.56],

[ 0.49, 0.51],

[ 0.51, 0.49],

[ 0.47, 0.53],

[ 0.41, 0.59],

[ 0.51, 0.49],

[ 0.51, 0.49],

[ 0.37, 0.63],

[ 0.5 , 0.5 ],

[ 0.25, 0.75],

[ 0.47, 0.53],

[ 0.46, 0.54],

[ 0.46, 0.54],

[ 0.48, 0.52],

[ 0.39, 0.61],

[ 0.46, 0.54],

[ 0.5 , 0.5 ],

[ 0.29, 0.71],

[ 0.46, 0.54],

[ 0.45, 0.55],

[ 0.51, 0.49],

[ 0.32, 0.68],

[ 0.45, 0.55],

[ 0.51, 0.49],

[ 0.31, 0.69],

[ 0.5 , 0.5 ],

[ 0.47, 0.53],

[ 0.5 , 0.5 ],

[ 0.46, 0.54],

[ 0.46, 0.54],

[ 0.28, 0.72],

[ 0.47, 0.53],

[ 0.31, 0.69],

[ 0.49, 0.51],

[ 0.28, 0.72],

[ 0.25, 0.75],

[ 0.32, 0.68],

[ 0.43, 0.57],

[ 0.47, 0.53],

[ 0.34, 0.66],

[ 0.42, 0.58],

[ 0.33, 0.67],

[ 0.46, 0.54],

[ 0.53, 0.47],

[ 0.32, 0.68],

[ 0.49, 0.51],

[ 0.33, 0.67],

[ 0.42, 0.58],

[ 0.45, 0.55],

[ 0.32, 0.68]])

In [37]:

**from** sklearn.metrics **import** jaccard\_similarity\_scorejaccard\_similarity\_score(y\_test, yhat)**from** sklearn.metrics **import** log\_losslog\_loss(y\_test, yhat\_prob)

Out[37]:

0.57722876094796538

# Model Evaluation using Test set

In [38]:

**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 [39]:

**!**wget -O loan\_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan\_test.csv

--2019-01-04 13:28:15-- 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’

100%[======================================>] 3,642 --.-K/s in 0s

2019-01-04 13:28:15 (491 MB/s) - ‘loan\_test.csv’ saved [3642/3642]

### Load Test set for evaluation

In [40]:

test\_df **=** pd**.**read\_csv('loan\_test.csv')test\_df**.**head()

Out[40]:

|  | **Unnamed: 0** | **Unnamed: 0.1** | **loan\_status** | **Principal** | **terms** | **effective\_date** | **due\_date** | **age** | **education** | **Gender** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | PAIDOFF | 1000 | 30 | 9/8/2016 | 10/7/2016 | 50 | Bechalor | female |
| **1** | 5 | 5 | PAIDOFF | 300 | 7 | 9/9/2016 | 9/15/2016 | 35 | Master or Above | male |
| **2** | 21 | 21 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 43 | High School or Below | female |
| **3** | 24 | 24 | PAIDOFF | 1000 | 30 | 9/10/2016 | 10/9/2016 | 26 | college | male |
| **4** | 35 | 35 | PAIDOFF | 800 | 15 | 9/11/2016 | 9/25/2016 | 29 | Bechalor | male |

In [41]:

X**=** preprocessing**.**StandardScaler()**.**fit(X)**.**transform(X)X[0:5]Y **=** test\_df['loan\_status']**.**valuesY[0:5]

Out[41]:

array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'], dtype=object)

In [44]:

*#test the KNN algorithm already trained with K=6*yhatKNN**=**neigh**.**predict(X)KNNJaccard **=** jaccard\_similarity\_score(y, yhatKNN)KNNF1 **=** f1\_score(y, yhatKNN, average**=**'weighted')print("Avg F1-score: %.2f" **%** KNNF1 )print("KNN Jaccard Score: %.2f" **%** KNNJaccard)

yhatDEC **=** Tree**.**predict(X)DTJaccard **=** jaccard\_similarity\_score(y, yhatDEC)DTF1 **=** f1\_score(y, yhatDEC, average**=**'weighted')print("Avg F1-score: %.2f" **%** DTF1 )print("Decision Tree Jaccard Score: %.2f" **%** DTJaccard)

yhatSVM**=**clf**.**predict(X)SVMJaccard **=** jaccard\_similarity\_score(y, yhatSVM)SVMF1 **=** f1\_score(y, yhatSVM, average**=**'weighted')print("Avg F1-score: %.2f" **%** SVMF1)print("SVM Jaccard score: %.2f" **%** SVMJaccard)

yhatLOG **=** LogR**.**predict(X)yhatLOGproba **=** LogR**.**predict\_proba(X)LogRJaccard **=** jaccard\_similarity\_score(y, yhatLOG)LogRF1 **=** f1\_score(y, yhatLOG, average**=**'weighted')Logloss **=** log\_loss(y, yhatLOGproba)print("LogLoss: : %.2f" **%** Logloss)print("Avg F1-score: %.4f" **%** LogRF1)print("LOG Jaccard score: %.4f" **%** LogRJaccard)

Avg F1-score: 0.78

KNN Jaccard Score: 0.79

Avg F1-score: 0.78

Decision Tree Jaccard Score: 0.79

Avg F1-score: 0.76

SVM Jaccard score: 0.77

LogLoss: : 0.56

Avg F1-score: 0.7199

LOG Jaccard score: 0.7428

In [ ]:

# Report

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

| **Algorithm** | **Jaccard** | **F1-score** | **LogLoss** |
| --- | --- | --- | --- |
| KNN | 0.79 | 0.78 | NA |
| Decision Tree | 0.79 | 0.78 | NA |
| SVM | 0.77 | 0.76 | NA |
| LogisticRegression | 0.7428 | 0.7199 | 0.56 |

## 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.

Copyright © 2018 [Cognitive Class](https://cocl.us/DX0108EN_CC). This notebook and its source code are released under the terms of the [MIT License](https://bigdatauniversity.com/mit-license/).