[(https://www.bigdatauniversity.com)](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
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

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()
```

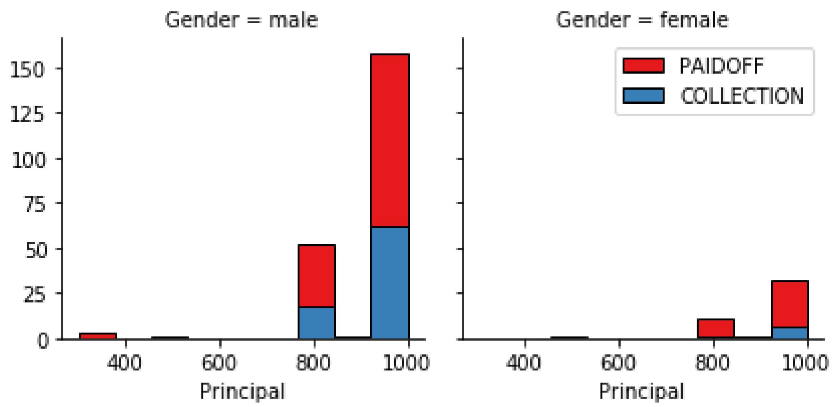
Out[6]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

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

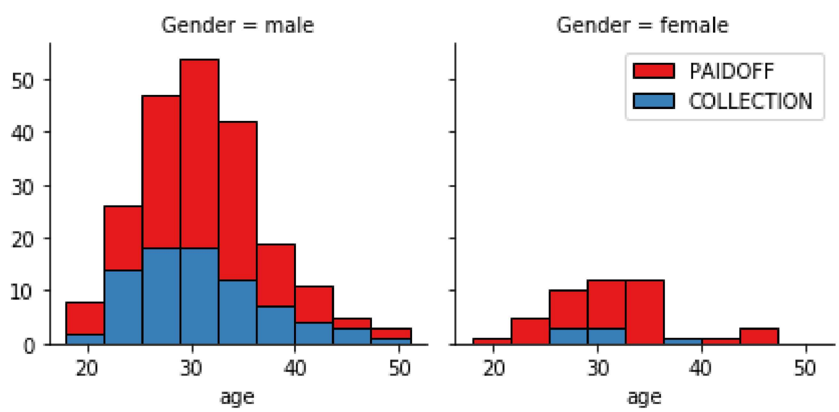
Lets plot some columns to underestand data better:

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

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()
```



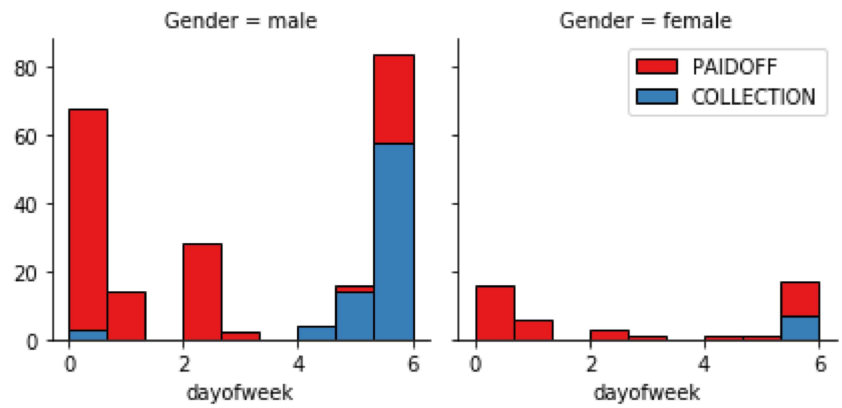
In [9]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wra
p=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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

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

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**.

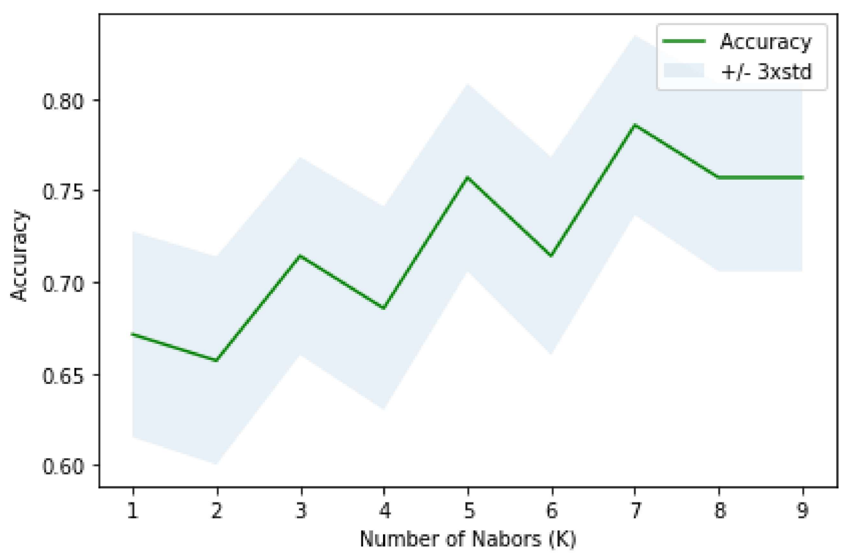
In [20]:
```
# split train_loan
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, rand
om_state=4 )
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

In [21]:

```
# import library
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# try with 10 different values of k to find the best one
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):
    #Train Model and Predict
    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])

# accuracy
print(mean_acc)

# Plot model accuracy for Different number of Neighbors
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, al
pha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()

# result
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argma
x()+1 )
```

```
[0.67142857 0.65714286 0.71428571 0.68571429 0.75714286 0.71428571
 0.78571429 0.75714286 0.75714286]
```




```
The best accuracy was with 0.7857142857142857 with k= 7
```

In [22]:
```
# train model with k=7
neigh = KNeighborsClassifier(n_neighbors = 7).fit(X_train,y_train)
neigh
```

Out[22]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

Decision Tree

In [23]:
```
# import library
from sklearn.tree import DecisionTreeClassifier
# create an instance of the DecisionTreeClassifier called loanTree
loanTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
# training
loanTree.fit(X_train,y_train)
loanTree
```

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')
```

Support Vector Machine

In [24]:
```
# import library
from sklearn import svm
# training
clf = svm.SVC()
clf.fit(X_train, y_train)
```

```
C:\Users\niping1\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureW
arning: The default value of gamma will change from 'auto' to 'scale' in vers
ion 0.22 to account better for unscaled features. Set gamma explicitly to 'au
to' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[24]:
```
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)
```

Logistic Regression

In [25]:
```
# import library
from sklearn.linear_model import LogisticRegression
# training
LR = LogisticRegression(C=0.01).fit(X_train,y_train)
LR
```

```
C:\Users\niping1\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Spec
ify a solver to silence this warning.
  FutureWarning)
```

Out[25]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

Model Evaluation using Test set

In [26]:
```
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 [27]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-co
urses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

Load Test set for evaluation

In [28]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-28-5998c8396b46> in <module>
----> 1 test_df = pd.read_csv('loan_test.csv')
      2 test_df.head()

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(filepath_or_bu
ffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, man
gle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinit
ialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter,
verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col,
date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousan
ds, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, com
ment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, l
ow_memory, memory_map, float_precision)
    683         )
    684 
--> 685         return _read(filepath_or_buffer, kwds)
    686 
    687     parser_f.__name__ = name

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffe
r, kwds)
    455 
    456     # Create the parser.
--> 457     parser = TextFileReader(fp_or_buf, **kwds)
    458 
    459     if chunksize or iterator:

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, engin
e, **kwds)
    893             self.options["has_index_names"] = kwds["has_index_names"]
    894 
--> 895         self._make_engine(self.engine)
    896 
    897     def close(self):

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, engi
ne)
   1133     def _make_engine(self, engine="c"):
   1134         if engine == "c":
-> 1135             self._engine = CParserWrapper(self.f, **self.options)
   1136         else:
   1137             if engine == "python":

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, **k
wds)
   1915         kwds["usecols"] = self.usecols
   1916 
-> 1917         self._reader = parsers.TextReader(src, **kwds)
   1918         self.unnamed_cols = self._reader.unnamed_cols
   1919 

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_sou
rce()
```

```
FileNotFoundError: [Errno 2] File b'loan_test.csv' does not exist: b'loan_tes
t.csv'
```

In [ ]:
```python
# Pre-processing loan_test
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df['dayofweek'] = 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=Tr
ue)
Feature_test = test_df[['Principal','terms','age','Gender','weekend']]
Feature_test = pd.concat([Feature_test,pd.get_dummies(test_df['education'])],
axis=1)
Feature_test.drop(['Master or Above'], axis = 1,inplace=True)
test_X = Feature_test
test_X = preprocessing.StandardScaler().fit(test_X).transform(test_X)
test_X[0:5]
```

In [ ]:
```python
# Pre-processing loan_test (cont)
test_y = test_df['loan_status'].values
test_y[0:5]
```

KNN

In [ ]:
```python
# predicted y
yhat_knn = neigh.predict(test_X)

# jaccard
jaccard_knn = jaccard_similarity_score(test_y, yhat_knn)
print("KNN Jaccard index: ", jaccard_knn)

# f1_score
f1_score_knn = f1_score(test_y, yhat_knn, average='weighted')
print("KNN F1-score: ", f1_score_knn)
```

Decision Tree

In [ ]:
```python
# predicted y
yhat_dt = loanTree.predict(test_X)

# jaccard
jaccard_dt = jaccard_similarity_score(test_y, yhat_dt)
print("DT Jaccard index: ", jaccard_dt)

# f1_score
f1_score_dt = f1_score(test_y, yhat_dt, average='weighted')
print("DT F1-score: ", f1_score_dt)
```

SVM

In [ ]:
```
# predicted y
yhat_svm = clf.predict(test_X)

# jaccard
jaccard_svm = jaccard_similarity_score(test_y, yhat_svm)
print("SVM Jaccard index: ", jaccard_svm)

# f1_score
f1_score_svm = f1_score(test_y, yhat_svm, average='weighted')
print("SVM F1-score: ", f1_score_svm)
```

Logistic regression

In [ ]:
```
# predicted y
yhat_lg = LR.predict(test_X)
yhat_lg_prob = LR.predict_proba(test_X)

# jaccard
jaccard_lg = jaccard_similarity_score(test_y, yhat_lg)
print("LR Jaccard index: ", jaccard_lg)

# f1_score
f1_score_lg = f1_score(test_y, yhat_lg, average='weighted')
print("LR F1-score: ", f1_score_lg)

# logloss
logloss_lg = log_loss(test_y, yhat_lg_prob)
print("LR log loss: ", logloss_lg)
```

Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.685	0.645	NA
Decision Tree	0.519	0.539	NA
SVM	0.815	0.786	NA
LogisticRegression	0.741	0.630	0.604

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.
