ML0101EN-Proj-Loan-py-v1

December 17, 2018

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
        !pip install -U scikit-learn scipy matplotlib
        from sklearn import preprocessing
        %matplotlib inline
```

```
Requirement already up-to-date: scikit-learn in /home/jupyterlab/conda/lib/python3.6/site-packag
Requirement already up-to-date: scipy in /home/jupyterlab/conda/lib/python3.6/site-packages (1.1
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/71/07/16d781df15be30df4acfd536c479268f1208
    100% || 12.9MB 3.4MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.8.2 in /home/jupyterlab/conda/lib/pyth
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /home/jupyterlab/conda/
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /ho
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /home/jupyterlab/conda/lib/pyth
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /home/jupyterlab/conda/lib
Requirement already satisfied, skipping upgrade: six>=1.5 in /home/jupyterlab/conda/lib/python3.
Requirement already satisfied, skipping upgrade: setuptools in /home/jupyterlab/conda/lib/python
Installing collected packages: matplotlib
  Found existing installation: matplotlib 3.0.0
    Uninstalling matplotlib-3.0.0:
      Successfully uninstalled matplotlib-3.0.0
Successfully installed matplotlib-3.0.2
```

0.0.1 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-dat
        print('Download Successful')
```

```
--2018-12-17 13:28:29--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cogni
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net).
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.n
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

2018-12-17 13:28:30 (1.05 MB/s) - loan_train.csv saved [23101/23101]

Download Successful
```

0.0.2 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  \
        0           0             0     PAIDOFF       1000     30       9/8/2016
        1           2             2     PAIDOFF       1000     30       9/8/2016
        2           3             3     PAIDOFF       1000     15       9/8/2016
        3           4             4     PAIDOFF       1000     30       9/9/2016
        4           6             6     PAIDOFF       1000     30       9/9/2016

             due_date  age             education  Gender
```

```
   0  10/7/2016   45  High School or Below    male
   1  10/7/2016   33              Bechalor  female
   2  9/22/2016   27               college    male
   3  10/8/2016   28               college  female
   4  10/8/2016   29               college    male
```

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

```
Out[4]: (346, 10)
```

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

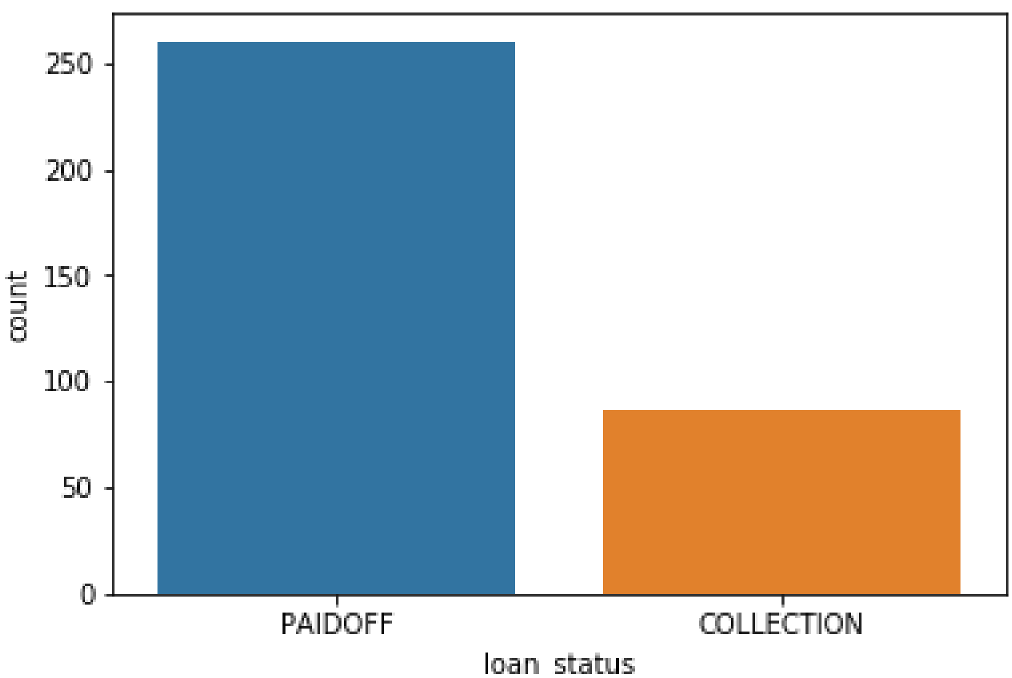
```
Out[6]: loan_status
        COLLECTION     86
        PAIDOFF       260
        dtype: int64
```

```
In [10]: !conda install -c anaconda seaborn -y
```

```
Solving environment: done

# All requested packages already installed.
```

```
In [9]: import seaborn as sns
        sns.countplot(df['loan_status'],label="Count")
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3ab05dfbe0>
```



0.0.3 Convert to date time object

```
In [11]: df['due_date'] = pd.to_datetime(df['due_date'])
         df['effective_date'] = pd.to_datetime(df['effective_date'])
         df.head()
```

```
Out[11]:    Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
         0           0             0     PAIDOFF       1000     30     2016-09-08
         1           2             2     PAIDOFF       1000     30     2016-09-08
         2           3             3     PAIDOFF       1000     15     2016-09-08
         3           4             4     PAIDOFF       1000     30     2016-09-09
         4           6             6     PAIDOFF       1000     30     2016-09-09

             due_date  age             education  Gender
         0 2016-10-07   45  High School or Below    male
         1 2016-10-07   33              Bechalor  female
         2 2016-09-22   27               college    male
         3 2016-10-08   28               college  female
         4 2016-10-08   29               college    male
```

1 Data visualization and pre-processing

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

```
In [12]: df['loan_status'].value_counts()
```

```
Out[12]: PAIDOFF       260
         COLLECTION     86
         Name: loan_status, dtype: int64
```

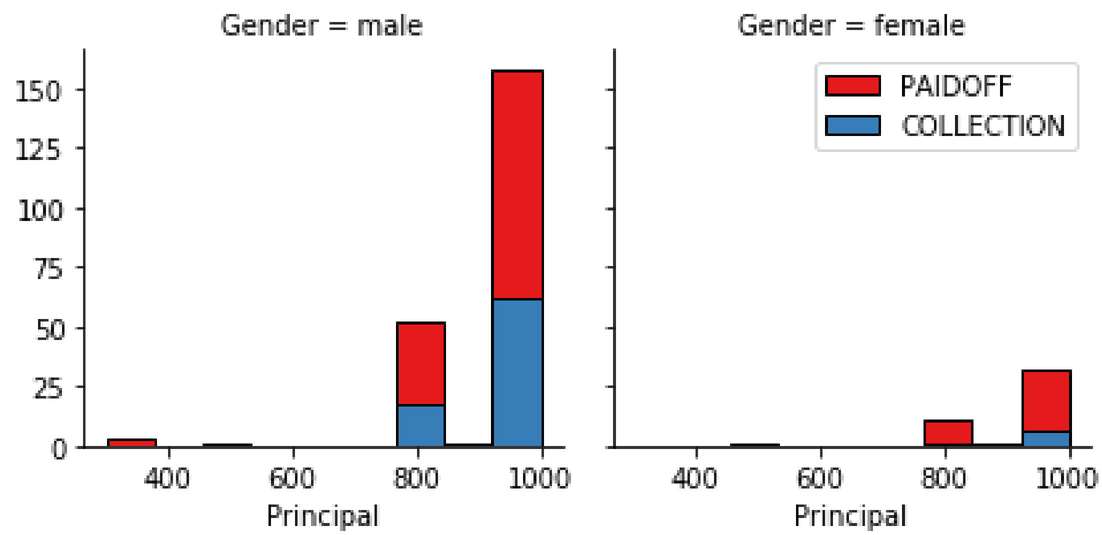
260 people have paid off the loan on time while 86 have gone into collection
Lets plot some columns to underestand data better:

```
In [8]: # notice: installing seaborn might takes a few minutes
        !conda install -c anaconda seaborn -y
```

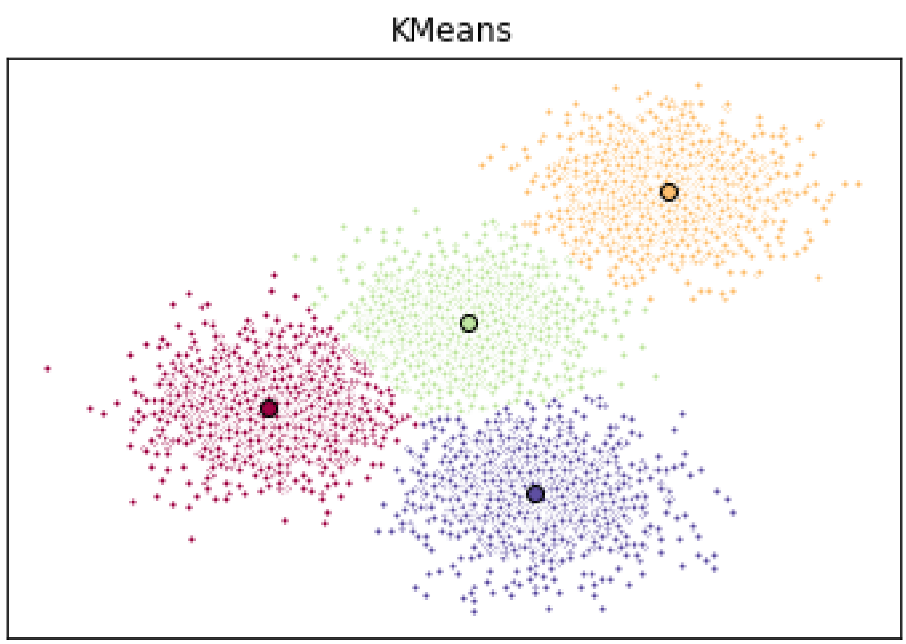
```
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
```

```
    ---------------------------|-----------------
    ca-certificates-2018.03.07 |                0         124 KB  anaconda
    seaborn-0.9.0              |           py36_0         379 KB  anaconda
    pandas-0.23.4              |   py36h04863e7_0        10.1 MB  anaconda
    patsy-0.5.1                |           py36_0         380 KB  anaconda
    statsmodels-0.9.0          |   py36h035aef0_0         9.0 MB  anaconda
    ------------------------------------------------------------
                                           Total:        19.9 MB

The following packages will be UPDATED:

    pandas:          0.23.4-py37h04863e7_0             --> 0.23.4-py36h04863e7_0 anaconda
    patsy:           0.5.0-py37_0                      --> 0.5.1-py36_0          anaconda
    seaborn:         0.9.0-py37_0                      --> 0.9.0-py36_0          anaconda
    statsmodels:     0.9.0-py37h035aef0_0              --> 0.9.0-py36h035aef0_0  anaconda

The following packages will be DOWNGRADED:

    ca-certificates: 2018.11.29-ha4d7672_0 conda-forge --> 2018.03.07-0          anaconda
    certifi:         2018.11.29-py36_1000  conda-forge --> 2018.10.15-py36_0     anaconda
    conda:           4.5.11-py36_1000      conda-forge --> 4.5.11-py36_0         anaconda
    openssl:         1.0.2p-h470a237_1     conda-forge --> 1.0.2p-h14c3975_0     anaconda


Downloading and Extracting Packages
ca-certificates-2018 | 124 KB    | ##################################### | 100%
seaborn-0.9.0        | 379 KB    | ##################################### | 100%
pandas-0.23.4        | 10.1 MB   | ##################################### | 100%
patsy-0.5.1          | 380 KB    | ##################################### | 100%
statsmodels-0.9.0    | 9.0 MB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [13]:
```
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()
```



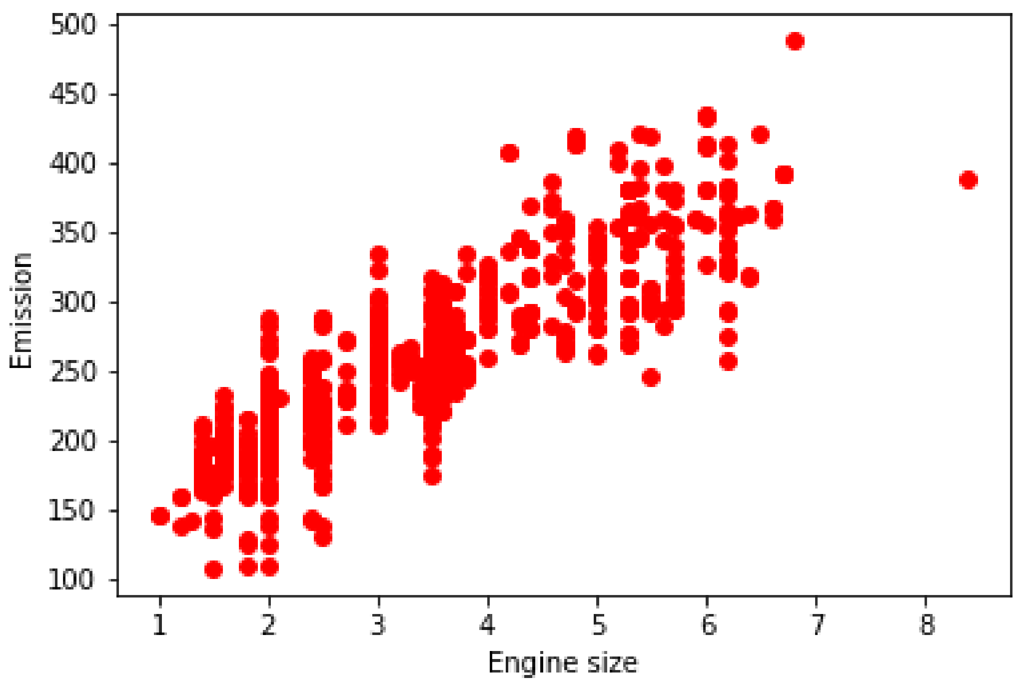

```
In [14]: bins = np.linspace(df.age.min(), df.age.max(), 10)
         g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
         g.map(plt.hist, 'age', bins=bins, ec="k")

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



2 Pre-processing: Feature selection/extraction

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

```
In [15]: 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_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 [16]: df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
         df.head()
```

```
Out[16]:    Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
         0           0             0     PAIDOFF       1000     30     2016-09-08
         1           2             2     PAIDOFF       1000     30     2016-09-08
         2           3             3     PAIDOFF       1000     15     2016-09-08
         3           4             4     PAIDOFF       1000     30     2016-09-09
         4           6             6     PAIDOFF       1000     30     2016-09-09

             due_date  age             education  Gender  dayofweek  weekend
         0 2016-10-07   45  High School or Below    male          3        0
```

```
1 2016-10-07   33              Bechalor  female          3        0
2 2016-09-22   27               college    male          3        0
3 2016-10-08   28               college  female          4        1
4 2016-10-08   29               college    male          4        1
```

2.1 Convert Categorical features to numerical values

Lets look at gender:

```
In [17]: df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

```
Out[17]: 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 [18]: df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
         df.head()
```

```
Out[18]:    Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
         0           0             0     PAIDOFF       1000     30     2016-09-08
         1           2             2     PAIDOFF       1000     30     2016-09-08
         2           3             3     PAIDOFF       1000     15     2016-09-08
         3           4             4     PAIDOFF       1000     30     2016-09-09
         4           6             6     PAIDOFF       1000     30     2016-09-09

              due_date  age             education  Gender  dayofweek  weekend
         0 2016-10-07   45  High School or Below       0          3        0
         1 2016-10-07   33              Bechalor       1          3        0
         2 2016-09-22   27               college       0          3        0
         3 2016-10-08   28               college       1          4        1
         4 2016-10-08   29               college       0          4        1
```

2.2 One Hot Encoding

How about education?

```
In [19]: df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

```
Out[19]: 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 [20]: df[['Principal','terms','age','Gender','education']].head()
```

```
Out[20]:    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 convert categorical varables to binary variables and append them to the feature Data Frame

```
In [21]: 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[21]:    Principal  terms  age  Gender  weekend  Bechalor  High School or Below  \
         0       1000     30   45       0        0         0                     1
         1       1000     30   33       1        0         1                     0
         2       1000     15   27       0        0         0                     0
         3       1000     30   28       1        1         0                     0
         4       1000     30   29       0        1         0                     0

            college
         0        0
         1        0
         2        1
         3        1
         4        1
```

2.2.1 Feature selection

Lets defind feature sets, X:

```
In [84]: X = Feature
         X[0:5]
```

```
Out[84]:    Principal  terms  age  Gender  weekend  Bechalor  High School or Below  \
         0       1000     30   45       0        0         0                     1
         1       1000     30   33       1        0         1                     0
```

```
2        1000      15    27         0          0          0                      0
3        1000      30    28         1          1          0                      0
4        1000      30    29         0          1          0                      0

   college
0        0
1        0
2        1
3        1
4        1
```

What are our lables?

```
In [85]: y = df['loan_status'].values
         y[0:5]
```

```
Out[85]: array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
               dtype=object)
```

2.3 Normalize Data

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

```
In [24]: #from sklearn import preprocessing
         X= preprocessing.StandardScaler().fit(X).transform(X)
         X[0:5]
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/preprocessing/data.py:625: DataConver
  return self.partial_fit(X, y)
/home/jupyterlab/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: DataConversionWarnin
```

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

3 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

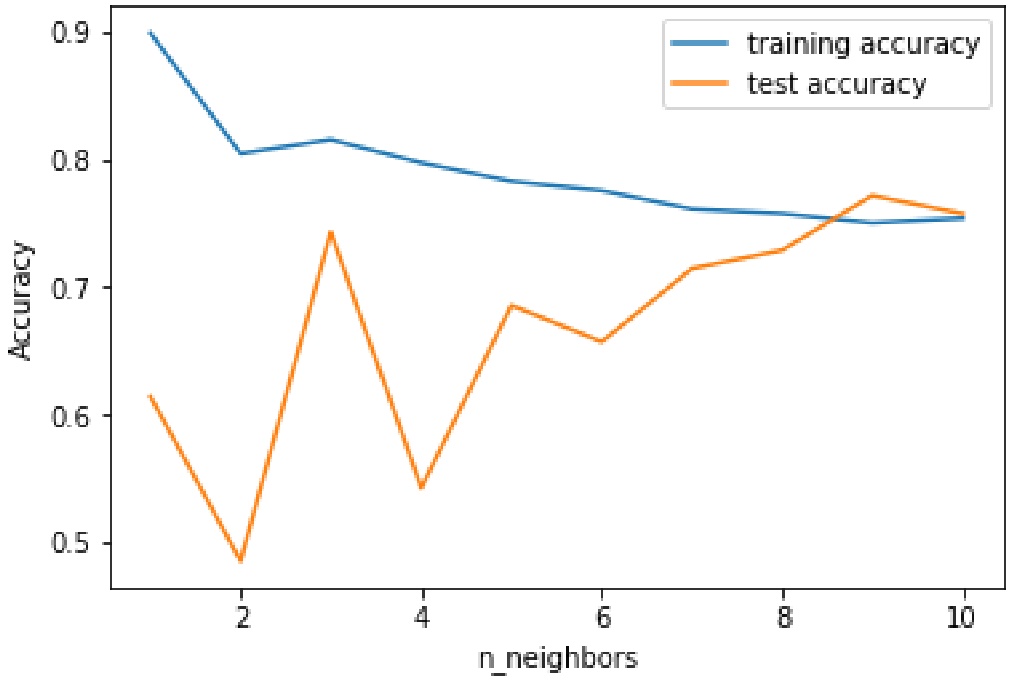
__ Notice:__ - You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model. - You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms. - You should include the code of the algorithm in the following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

4 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 [178]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state

from sklearn.neighbors import KNeighborsClassifier
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

Out[178]:
```
<matplotlib.legend.Legend at 0x7f3aaed17278>
```




```
In [183]: knn = KNeighborsClassifier(n_neighbors=8)
          knn.fit(X_train, y_train)
          print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train,
          print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_tes
```

```
Accuracy of K-NN classifier on training set: 0.76
Accuracy of K-NN classifier on test set: 0.73
```

```
In [ ]:
```

5 Decision Tree

```
In [184]: from sklearn.tree import DecisionTreeClassifier

          # Prune the tree to account for overfitting
          # Max depth of 3 levels

          tree = DecisionTreeClassifier(max_depth=3, random_state=0)
          tree.fit(X_train, y_train)
          print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
          print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.732
Accuracy on test set: 0.829
```

In [ ]:

6 Support Vector Machine

```
In [185]: from sklearn.svm import SVC
          svc = SVC(C=0.01)
          svc.fit(X_train, y_train)
          print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
          print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 0.73
Accuracy on test set: 0.83
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The d
  "avoid this warning.", FutureWarning)
```

7 Logistic Regression

```
In [192]: from sklearn.metrics import classification_report,jaccard_similarity_score, log_loss

          # Using C = 0.01 to improve better scoring

          logreg = LogisticRegression(C=0.01).fit(X_train, y_train)
          print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
          print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.732
Test set score: 0.829
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureW
  FutureWarning)
```

```
In [199]: log_reg_predict = logreg.predict(X_test)
          log_reg_predict_proba = logreg.predict_proba(X_test)[:, 1]
          print('Classification report:\n\n', classification_report(y_test, log_reg_predict))
          print('\n')
          print('jaccard_similarity_score:\n\n', jaccard_similarity_score(y_test, log_reg_predic
```

```
Classification report:

               precision    recall  f1-score   support

  COLLECTION       0.00      0.00      0.00        12
     PAIDOFF       0.83      1.00      0.91        58
```

```
   micro avg       0.83      0.83      0.83        70
   macro avg       0.41      0.50      0.45        70
weighted avg       0.69      0.83      0.75        70



jaccard_similarity_score:

 82.85714285714286 %


/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
```

8 Model Evaluation using Test set

```
In [111]: 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 [82]: !wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-dat

--2018-12-17 14:50:10--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cogni
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net).
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.n
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

2018-12-17 14:50:10 (53.9 MB/s) - loan_test.csv saved [3642/3642]
```

8.0.1 Load Test set for evaluation

```
In [257]: test_df = pd.read_csv('loan_test.csv')
          test_df.head()
```

```
Out[257]:    Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
          0           1             1     PAIDOFF       1000     30       9/8/2016
          1           5             5     PAIDOFF        300      7       9/9/2016
          2          21            21     PAIDOFF       1000     30      9/10/2016
          3          24            24     PAIDOFF       1000     30      9/10/2016
          4          35            35     PAIDOFF        800     15      9/11/2016

              due_date  age             education  Gender
          0  10/7/2016   50              Bechalor  female
          1  9/15/2016   35        Master or Above    male
          2  10/9/2016   43  High School or Below  female
          3  10/9/2016   26               college    male
          4  9/25/2016   29              Bechalor    male
```

```
In [258]: df['due_date'] = pd.to_datetime(df['due_date'])
          df['effective_date'] = pd.to_datetime(df['effective_date'])
          df.head()
```

```
Out[258]:    Unnamed: 0  Unnamed: 0.1  loan_status  Principal  terms effective_date  \
          0           0             0            0       1000     30     2016-09-08
          1           2             2            0       1000     30     2016-09-08
          2           3             3            0       1000     15     2016-09-08
          3           4             4            0       1000     30     2016-09-09
          4           6             6            0       1000     30     2016-09-09

              due_date  age             education  Gender  dayofweek  weekend
          0 2016-10-07   45  High School or Below       0          3        0
          1 2016-10-07   33              Bechalor       1          3        0
          2 2016-09-22   27               college       0          3        0
          3 2016-10-08   28               college       1          4        1
          4 2016-10-08   29               college       0          4        1
```

```
In [214]: df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
          df.head()
```

```
Out[214]:    Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
          0           0             0     PAIDOFF       1000     30     2016-09-08
          1           2             2     PAIDOFF       1000     30     2016-09-08
          2           3             3     PAIDOFF       1000     15     2016-09-08
          3           4             4     PAIDOFF       1000     30     2016-09-09
          4           6             6     PAIDOFF       1000     30     2016-09-09

              due_date  age             education  Gender  dayofweek  weekend
          0 2016-10-07   45  High School or Below       0          3        0
          1 2016-10-07   33              Bechalor       1          3        0
          2 2016-09-22   27               college       0          3        0
          3 2016-10-08   28               college       1          4        1
          4 2016-10-08   29               college       0          4        1
```

In [215]:
```python
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
#df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
```

Out[215]:
```
Gender  loan_status
0       PAIDOFF        0.731293
        COLLECTION     0.268707
1       PAIDOFF        0.865385
        COLLECTION     0.134615
Name: loan_status, dtype: float64
```

In [261]:
```python
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)
X = Feature
y = df['loan_status'].values
```

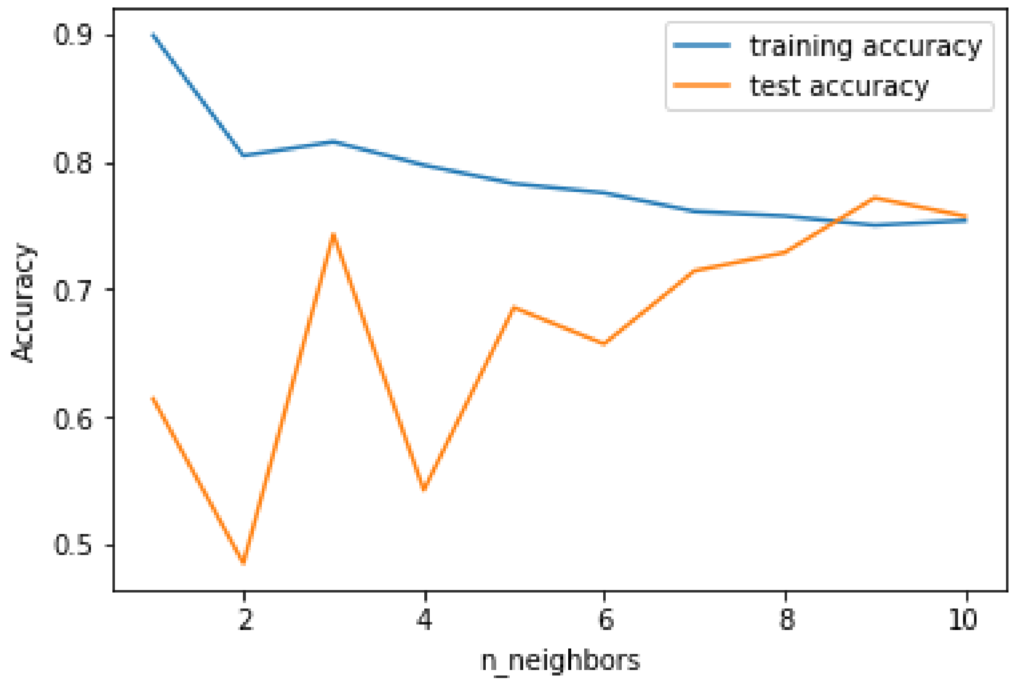
9 Train-Test-Split

In [262]:
```python
#Test_Train_Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
```

10 K Nearest Neighbor(KNN)

In [218]:
```python
from sklearn.neighbors import KNeighborsClassifier
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

Out[218]:
```
<matplotlib.legend.Legend at 0x7f3aaebaa978>
```




```
In [267]: knn = KNeighborsClassifier(n_neighbors=8)
          knn.fit(X_train, y_train)
          print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train,
          print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_tes
```

```
Accuracy of K-NN classifier on training set: 0.75
Accuracy of K-NN classifier on test set: 0.79
```

11 Decision Tree

```
In [225]: from sklearn.tree import DecisionTreeClassifier

          # Prune the tree to account for overfitting
          # Max depth of 3 levels

          tree = DecisionTreeClassifier(max_depth=3, random_state=0)
          tree.fit(X_train, y_train)
          print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
          print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.732
Accuracy on test set: 0.829
```

12 Support Vector Machine

```
In [231]: from sklearn.svm import SVC
          svc = SVC(C=0.01)
          svc.fit(X_train, y_train)
          print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
          print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 0.73
Accuracy on test set: 0.83


/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The d
  "avoid this warning.", FutureWarning)
```

13 Logistic Regression

```
In [264]: from sklearn.metrics import classification_report,jaccard_similarity_score, log_loss

          # Using C = 0.01 to improve better scoring

          logreg = LogisticRegression(C=0.01).fit(X_train, y_train)
          print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
          print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.732
Test set score: 0.829


/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureW
  FutureWarning)
```

```
In [233]: log_reg_predict = logreg.predict(X_test)
          log_reg_predict_proba = logreg.predict_proba(X_test)[:, 1]
          print('Classification report:\n\n', classification_report(y_test, log_reg_predict))
          print('\n')
          print('jaccard_similarity_score:\n\n', jaccard_similarity_score(y_test, log_reg_predic
```

```
Classification report:

               precision    recall  f1-score   support

  COLLECTION       0.00      0.00      0.00        12
     PAIDOFF       0.83      1.00      0.91        58

   micro avg       0.83      0.83      0.83        70
```

```
   macro avg       0.41      0.50      0.45        70
weighted avg       0.69      0.83      0.75        70



jaccard_similarity_score:

 82.85714285714286 %


/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
```

In [ ]:

In [ ]:

In [ ]:

14 Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	?	?	NA
Decision Tree	?	?	NA
SVM	?	?	NA
LogisticRegression	?	?	?

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

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

Thanks for completing this lesson!

Author: Saeed Aghabozorgi

Saeed Aghabozorgi, 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.