# Laboratory #1 Classification

## Table of Contents



The steps for you to follow to do this include:

1. Using the datasets provided run each of the three different machine learning techniques as described in this laboratory documentation. This will provide you with familiarity of each technique. This counts for roughly 50% of the points available for this laboratory.
2. Next, understanding how each of these techniques works use each of the three techniques to evaluate the online news popularity dataset. Note that this will involve some significant data "munging" or pre-processing to get the dataset in a form that will work. That is, each technique may require you to pre-process the dataset in a different way.
3. Generate your written laboratory report on your work. Use Python notebook to generate the result and submit the downloaded file (html, word or pdf). Don't submit the pure code without any result.

## Method #1. Tree-based classification

### Step 1: Collecting the data

The data set used for this method is the credit.csv file.This data set is available for you on Moodle.In addition, you can check it out at the UCI Data Repository at http://archive.ics.uci.edu/ml/.Before you use a data set you should check on its description to see how big the data set is, what format it is in, what processing has already been done, and to determine if there is anything "quirky" about it that you need to know in order to conduct your analysis.For example, the class variable "default" is the 17$^{th}$ column, or variable, out of the 17 in

this dataset.We'll use this knowledge when we create our model later.For now, use the following command to read it into Python.

```
import pandas as pd
# import the csv file
input_file = "address/to/file/credit_for_Python.csv"
credit = pd.read_csv(input_file)
```

If you want to see how the dataset is organized, you may use command head()

```
credit.head(4)
```

| | Creditability | Account Balance | Duration of Credit (month) | Payment Status of Previous Credit | Purpose | Credit Amount | Value Savings/Stocks | Length of current employment | Instalment per cent | Sex & Marital Status | ... | Duration in Current address | Most valuable available asset | Age (years) | Concurrent Credits |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 18 | 4 | 2 | 1049 | 1 | 2 | 4 | 2 | ... | 4 | 2 | 21 | 3 |
| **1** | 1 | 1 | 9 | 4 | 0 | 2799 | 1 | 3 | 2 | 3 | ... | 2 | 1 | 36 | 3 |
| **2** | 1 | 2 | 12 | 2 | 9 | 841 | 2 | 4 | 2 | 2 | ... | 4 | 1 | 23 | 3 |
| **3** | 1 | 1 | 12 | 4 | 0 | 2122 | 1 | 3 | 3 | 3 | ... | 2 | 1 | 39 | 3 |

4 rows × 21 columns

Use the *describe()* command to check what is in the credit object created to hold the data in Python.The describe command is shown below with the first few lines returned.

```
credit.describe()
```

| | Creditability | Account Balance | Duration of Credit (month) | Payment Status of Previous Credit | Purpose | Credit Amount | Value Savings/Stocks | Length of current employment | Instalment per cent | Sex & Marital Status | ... | Duration in Current address |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.000000 | 1000.00000 | 1000.000000 | 1000.00000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.00000 | ... | 1000.000000 |
| **mean** | 0.700000 | 2.577000 | 20.903000 | 2.54500 | 2.828000 | 3271.24800 | 2.105000 | 3.384000 | 2.973000 | 2.68200 | ... | 2.845000 |
| **std** | 0.458487 | 1.257638 | 12.058814 | 1.08312 | 2.744439 | 2822.75176 | 1.580023 | 1.208306 | 1.118715 | 0.70808 | ... | 1.103718 |
| **min** | 0.000000 | 1.000000 | 4.000000 | 0.00000 | 0.000000 | 250.00000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 | ... | 1.000000 |
| **25%** | 0.000000 | 1.000000 | 12.000000 | 2.00000 | 1.000000 | 1365.50000 | 1.000000 | 3.000000 | 2.000000 | 2.00000 | ... | 2.000000 |
| **50%** | 1.000000 | 2.000000 | 18.000000 | 2.00000 | 2.000000 | 2319.50000 | 1.000000 | 3.000000 | 3.000000 | 3.00000 | ... | 3.000000 |
| **75%** | 1.000000 | 4.000000 | 24.000000 | 4.00000 | 3.000000 | 3972.25000 | 3.000000 | 5.000000 | 4.000000 | 3.00000 | ... | 4.000000 |
| **max** | 1.000000 | 4.000000 | 72.000000 | 4.00000 | 10.000000 | 18424.00000 | 5.000000 | 5.000000 | 4.000000 | 4.00000 | ... | 4.000000 |

8 rows × 21 columns

### Step 2:Exploring the data

There are many ways you can further check the data in this object, e.g. info or describe.For example, you can use the *info()* command to get information about properties of all the variables

```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
Creditability                          1000 non-null int64
Account Balance                        1000 non-null int64
Duration of Credit (month)             1000 non-null int64
Payment Status of Previous Credit      1000 non-null int64
Purpose                                1000 non-null int64
Credit Amount                          1000 non-null int64
Value Savings/Stocks                   1000 non-null int64
Length of current employment           1000 non-null int64
Instalment per cent                    1000 non-null int64
Sex & Marital Status                   1000 non-null int64
Guarantors                             1000 non-null int64
Duration in Current address            1000 non-null int64
Most valuable available asset          1000 non-null int64
Age (years)                            1000 non-null int64
Concurrent Credits                     1000 non-null int64
Type of apartment                      1000 non-null int64
No of Credits at this Bank             1000 non-null int64
Occupation                             1000 non-null int64
No of dependents                       1000 non-null int64
Telephone                              1000 non-null int64
Foreign Worker                         1000 non-null int64
dtypes: int64(21)
memory usage: 164.1 KB
```

What is target variable? Let's store it in a new variable.

```
target = credit['Creditability']
```

If you want to see the number of each category of target variable you can use following command:

```
target.value_counts()
```

```
yes    700
no     300
Name: Creditability, dtype: float64
```

(You will see 1 and 0 which refer to yes/no)

In order to develop the tree-based classification model you need to split the data into training and test records. There are many ways that we can make this split. We introduce two methods here:

*Method 1:*

```
# Generate random numbers
import random
random.seed(12345)
indx = random.sample(range(0, 1000), 1000)
credit_rand = credit.iloc[indx]
target_rand = target.iloc[indx]
```

We can see the result of randomization using:

```
credit_rand.head(5)
```

| | Creditability | Account Balance | Duration of Credit (month) | Payment Status of Previous Credit | Purpose | Credit Amount | Value Savings/Stocks | Length of current employment | Instalment per cent | Sex & Marital Status | ... | Duration in Current address | Most valuable available asset | Age (years) | Concurrer Credit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **426** | 1 | 2 | 39 | 3 | 6 | 11760 | 2 | 4 | 2 | 3 | ... | 3 | 4 | 32 | |
| **750** | 1 | 2 | 36 | 3 | 0 | 2862 | 2 | 5 | 4 | 3 | ... | 3 | 4 | 30 | |
| **10** | 1 | 1 | 11 | 4 | 0 | 3905 | 1 | 3 | 2 | 3 | ... | 2 | 1 | 36 | |
| **839** | 0 | 4 | 12 | 2 | 0 | 1386 | 3 | 3 | 2 | 2 | ... | 2 | 2 | 26 | |
| **845** | 0 | 2 | 27 | 4 | 3 | 2520 | 3 | 3 | 4 | 3 | ... | 2 | 2 | 23 | |

5 rows × 21 columns

You can and should check to make sure that randomizing your data has not made any substantive changes to it, i.e. the means should still be the same and so on. You can check the summary on the original data to the summary using the new object you created to do this.

```
credit_rand.describe()
```

Compare this result to what we had earlier.

You'll need to subset the observations (records) to establish the training set and the test set. Sort of a rule of thumb is to use between 75% and 90% of the records for the training set. There are many ways to do this in Python. For example, you can use:

```
credit_train = credit_rand.iloc[0:700]
credit_test = credit_rand.iloc[700:1000]
target_train = target_rand.iloc[0:700]
target_test = target_rand.iloc[700:1000]
```

If the randomization went well then the percentages between splits should be close.Before we checked on the number of defaults for both training and test.This time we want to see the percentage of each:

```
target_train.value_counts()/700
```

```
1    0.691429
0    0.308571
Name: Creditability, dtype: float64
```

```
target_test.value_counts()/300
```

```
1    0.72
0    0.28
Name: Creditability, dtype: float64
```

*Method 2:*

We can use following code to split data:

```
# Method 2 of splitting the model
from sklearn.model_selection import train_test_split

y = target
X = credit.drop(['Creditability'],axis=1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=52)
```

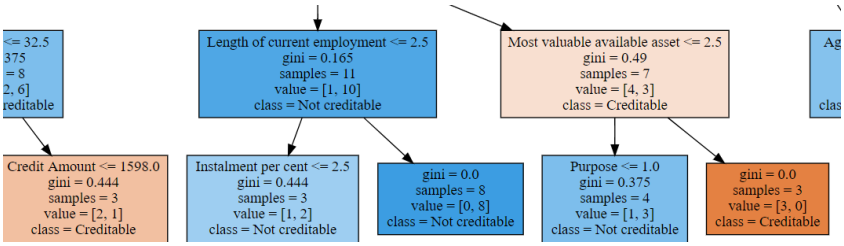
**Step 3: Training a model on the data**

You only need one command line to create the decision tree model. However, now you need to set-up this command so that the algorithm knows that the class or response variable is in the 17th column. The command and the model created are:

```
# Design decision tree

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

model = tree.DecisionTreeClassifier()
model = model.fit(X_train, y_train)
```

To visualize the model, we can use:

```
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

graph = Source(tree.export_graphviz(model, out_file=None
 , feature_names=X.columns, class_names=['default', 'no default']
 , filled = True))
display(SVG(graph.pipe(format='svg')))
```




To see the visualized version of decision tree, it is required to install Graphviz package which is sometimes tricky. The best way is to go to "anaconda command prompt" and type:

```
conda install python-graphviz
```

### Step 4: Evaluating Model Performance

We still need to use our test set to evaluate/validate the model's overall performance. To do this we'll use the predict() command as follows:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

y_predict = model.predict(X_test)

print(confusion_matrix(y_test, y_predict))
print(accuracy_score(y_test, y_predict)*100)
```

Q1- If you see the accuracy 0f 100%, what does it mean? Does this mean that we design a perfect model? This is some thing that needs more discussion. Write a few sentences about accuracy of 100%.

## Method #2. Random forest

Let's use the same dataset but this time for random forest. Use the similar strategy described in course to see the accuracy.

```
# Part 2 Random forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

y_predict = clf.predict(X_test)
print(confusion_matrix(y_test, y_predict))

print(accuracy_score(y_test, y_predict)*100)
```

```
[[ 41  49]
 [ 32 178]]
73.0
```

Q2- What are the three most important features in this model.

Now, Change the random seed to 23458 and find the new accuracy of random forest.

## Method #3. Adding regression to trees

### Step 1: Collecting the Data

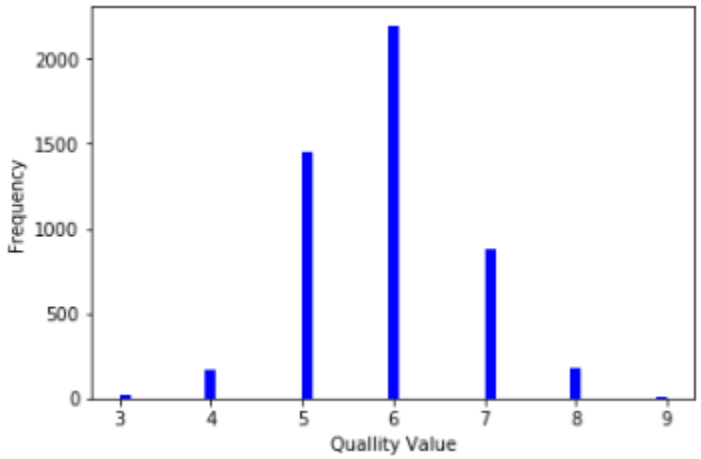
Our last method will use the whitewines.csv file available for you on Moodle. There is nothing quirky about reading this data into Python.

```
input_file = "address/to/file/whitewines.csv"
wine = pd.read_csv(input_file)
wine.head(5)
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 6.7 | 0.62 | 0.24 | 1.10 | 0.039 | 6.0 | 62.0 | 0.99340 | 3.41 | 0.32 | 10.400000 | 5 |
| 1 | 5.7 | 0.22 | 0.20 | 16.00 | 0.044 | 41.0 | 113.0 | 0.99862 | 3.22 | 0.46 | 8.900000 | 6 |
| 2 | 5.9 | 0.19 | 0.26 | 7.40 | 0.034 | 33.0 | 123.0 | 0.99500 | 3.49 | 0.42 | 10.100000 | 6 |
| 3 | 5.3 | 0.47 | 0.10 | 1.30 | 0.036 | 11.0 | 74.0 | 0.99082 | 3.48 | 0.54 | 11.200000 | 4 |
| 4 | 6.4 | 0.29 | 0.21 | 9.65 | 0.041 | 36.0 | 119.0 | 0.99334 | 2.99 | 0.34 | 10.933333 | 6 |

Again, there is not anything particularly quirky about this data on the surface. However, because we will use regression we should check to see if the class variable, quality, follows a normal distribution or is nearly normal.

```
# Plot the target variable
import matplotlib.pyplot as plt

# An "interface" to matplotlib.axes.Axes.hist() method
n, bins, patches = plt.hist(x=wine['quality'], bins='auto', color='b', )
plt.xlabel('Quallity Value')
plt.ylabel('Frequency')
plt.show()
```




### Step 2: Exploring and Preparing the Data

Essentially all we need to do this time is to subset our data into training and test sets. Let's use an approximately 75% for training and 25% for testing split of the observations.

### Step 3: Training a Model on the Data

Design the model similar the way you did earlier.

```
model = tree.DecisionTreeClassifier()
model = model.fit(X_train, y_train)
```

To get the basic information about the tree, you can export the information of the model to a *.dot* file.

```
from sklearn.tree import export_graphviz

# export the decision tree to a tree.dot file
# for visualizing the plot easily anywhere
export_graphviz(model, out_file ='tree.dot', feature_names =X.columns)
```

If you open the generated file (tree.dot), you should see a result similar to this:

```
digraph Tree {
node [shape=box] ;
0 [label="alcohol <= 10.75\ngini = 0.674\nsamples = 2938\nvalue = [14, 104, 890, 1322,
502, 104, 2]"] ;
1 [label="volatile acidity <= 0.287\ngini = 0.622\nsamples = 1780\nvalue = [10, 83,
757, 774, 131, 24, 1]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="volatile acidity <= 0.227\ngini = 0.618\nsamples = 1088\nvalue = [8, 32,
339, 568, 116, 24, 1]"] ;
1 -> 2 ;
3 [label="alcohol <= 8.75\ngini = 0.59\nsamples = 533\nvalue = [3, 14, 113, 313, 73,
17, 0]"] ;
2 -> 3 ;
4 [label="sulphates <= 0.425\ngini = 0.667\nsamples = 21\nvalue = [1, 0, 6, 3, 10, 1,
0]"] ;
3 -> 4 ;
5 [label="volatile acidity <= 0.205\ngini = 0.375\nsamples = 8\nvalue = [0, 0, 6, 2,
0, 0, 0]"] ;
4 -> 5 ;
6 [label="gini = 0.0\nsamples = 2\nvalue = [0, 0, 0, 2, 0, 0, 0]"] ;
5 -> 6 ;
7 [label="gini = 0.0\nsamples = 6\nvalue = [0, 0, 6, 0, 0, 0, 0]"] ;
5 -> 7 ;
8 [label="alcohol <= 8.55\ngini = 0.391\nsamples = 13\nvalue = [1, 0, 0, 1, 10, 1,
0]"] ;
4 -> 8 ;
9 [label="volatile acidity <= 0.198\ngini = 0.667\nsamples = 3\nvalue = [1, 0, 0, 1,
0, 1, 0]"] ;
8 -> 9 ;
10 [label="pH <= 3.11\ngini = 0.5\nsamples = 2\nvalue = [0, 0, 0, 1, 0, 1, 0]"] ;
```

You may see the regression graph using following commands:

```
dot_data = tree.export_graphviz(model, out_file=None,
```

```
                        feature_names=X.columns,
                        filled=True, rounded=True,
                        special_characters=True)

graph = graphviz.Source(dot_data)
graph
```

**Step 4: Evaluating Model Performance**

Start with predicting the output using the same methods as before. Call the output y_predict.

The fact that the model's range is much smaller than the data's actual range suggests that the model is not adequately capturing the extremes of the range of quality, i.e. the very good or very bad wines. We can also check the correlation:

```
np.corrcoef(y_test,y_predict)
```

A 54% correlation is ok, but not great. Last thing that we can try is to see the amount of RMSE (Root Mean Square Error) for all the test instances. You will find out an RMSE of around 0.84.

Q3- What is your interpretation about this amount of RMSE?

## Method #4. News Popularity

The Online News Popularity data set from the University of California – Irvine Machine Learning Data Repository is provided for your use as well as two related journal articles, "Predicting and Evaluating the Popularity of Online News" by He Ren and Quan Yang, and "A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News" by Kelwin Fernandes, Pedro Vinagre and Paulo Cortez.The URL for the data set description is: http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity#.Ultimately, your assignment is to use this one dataset and compare the results of these three different machine learning techniques to evaluate which of those provides the best results.Because you are assessing the "market share" based on different measures of popularity, in this case the best results will be the results that provide the best understanding of what drives market share. This is different than determining exactly what the market share is.

### Step 1: Collecting the Data

Again, start with loading the data and see how the output looks like.

| | url | timedelta | n_tokens_title | n_tokens_content | n_unique_tokens | n_non_stop_words | n_non_stop_unique_tokens |
|---|---|---|---|---|---|---|---|
| 0 | http://mashable.com/2013/01/07/amazon-instant-... | 731.0 | 12.0 | 219.0 | 0.663594 | 1.0 | 0.815385 |
| 1 | http://mashable.com/2013/01/07/ap-samsung-spon... | 731.0 | 9.0 | 255.0 | 0.604743 | 1.0 | 0.791946 |
| 2 | http://mashable.com/2013/01/07/apple-40-billio... | 731.0 | 9.0 | 211.0 | 0.575130 | 1.0 | 0.663866 |
| 3 | http://mashable.com/2013/01/07/astronaut-notre... | 731.0 | 9.0 | 531.0 | 0.503788 | 1.0 | 0.665635 |
| 4 | http://mashable.com/2013/01/07/att-u-verse-apps/ | 731.0 | 13.0 | 1072.0 | 0.415646 | 1.0 | 0.540890 |

The goal is to see how popular the articles are. There are 61 features here but this makes analysis so time consuming while we don't need some of the features anyways.

### Step 2: Pre-processing

We want to make this problem a classification one. One approach is to make any piece of article more than 1400 likes as a favorite one.

```
# handle goal attrubte to binary
popular = news.shares >= 1400
unpopular = news.shares < 1400
news.loc[popular,'shares'] = 1
news.loc[unpopular,'shares'] = 0
```

**Step 3: Modeling and evaluation**

Q4- Try decision tree and random forest and evaluate the model.