Machine Learning Lab

ETCS-454

Faculty: Ms. Sudha Narang

Name: Somin Wadhwa

Roll No.: 019-6402714

Group: 8C5

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

MACHINE LEARNING LAB
PRACTICAL RECORD

Paper Code: ETCS-454

Name : Somin Wadhwa

Roll No. : 01996402714

Branch : CSE

Group : 8C5

S.No	**Experiment**	**Date**	**Signature**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

Somin Wadhwa
01996402714

Experiment-1

Aim- Introduction to WEKA and hands on machine learning tools.

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

Found only on the islands of New Zealand, the Weka is a flightless bird with an inquisitive nature. The name is pronounced like this, and the bird sounds like this.

Weka is open source software issued under the GNU General Public License.

Advantages of Weka include:

- Free availability under the GNU General Public License.
- Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.
- A comprehensive collection of data pre-processing and modelling techniques.
- Ease of use due to its graphical user interfaces.

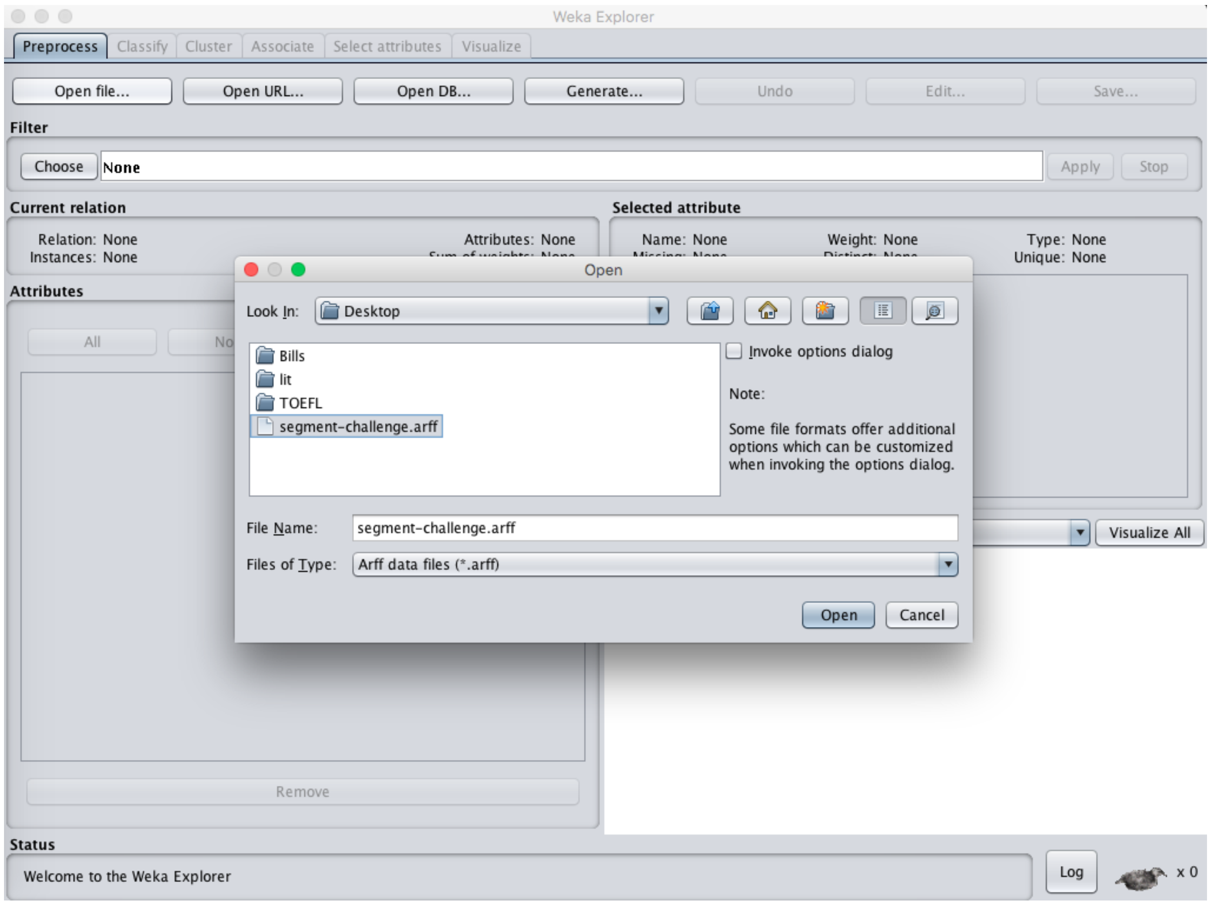
The GUI Chooser consists of four buttons:
•Explorer: An environment for exploring data with WEKA.
•Experimenter: An environment for performing experiments and conducting statistical tests between learning schemes.
•Knowledge Flow: This environment supports essentially the same functions as the Explorer but with a drag and-drop interface. One advantage is that it supports incremental learning.
•Simple CLI: Provides a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.

The Explorer interface features several panels providing access to the main components of the workbench:

- The Preprocess panel has facilities for importing data from a database, a comma-separated values (CSV) file, etc., and for preprocessing this data using a so-called filtering algorithm. These filters can be used to transform the data (e.g., turning numeric attributes into discrete ones) and make it possible to delete instances and attributes according to specific criteria.
- The Classify panel enables applying classification and regression algorithms (indiscriminately called classifiers in Weka) to the resulting dataset, to estimate the accuracy of the resulting predictive model, and to visualize erroneous predictions, receiver operating characteristic (ROC) curves, etc., or the model itself (if the model is amenable to visualization like, e.g., a decision tree).
- The Associate panel provides access to association rule learners that attempt to identify all important interrelationships between attributes in the data.
- The Cluster panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. There is also an implementation of the expectation maximization algorithm for learning a mixture of normal distributions.
- The Select attributes panel provides algorithms for identifying the most predictive attributes in a dataset.

- The Visualize panel shows a scatter plot matrix, where individual scatter plots can be selected and enlarged, and analyzed further using various selection operators.

Opening a Dataset:



Filter: None

Relation: None
Instances: None
Attributes: None

Name: None
Weight: None
Type: None
Unique: None

Look In: Desktop

Invoke options dialog

Note:

Some file formats offer additional options which can be customized when invoking the options dialog.

File Name: segment-challenge.arff

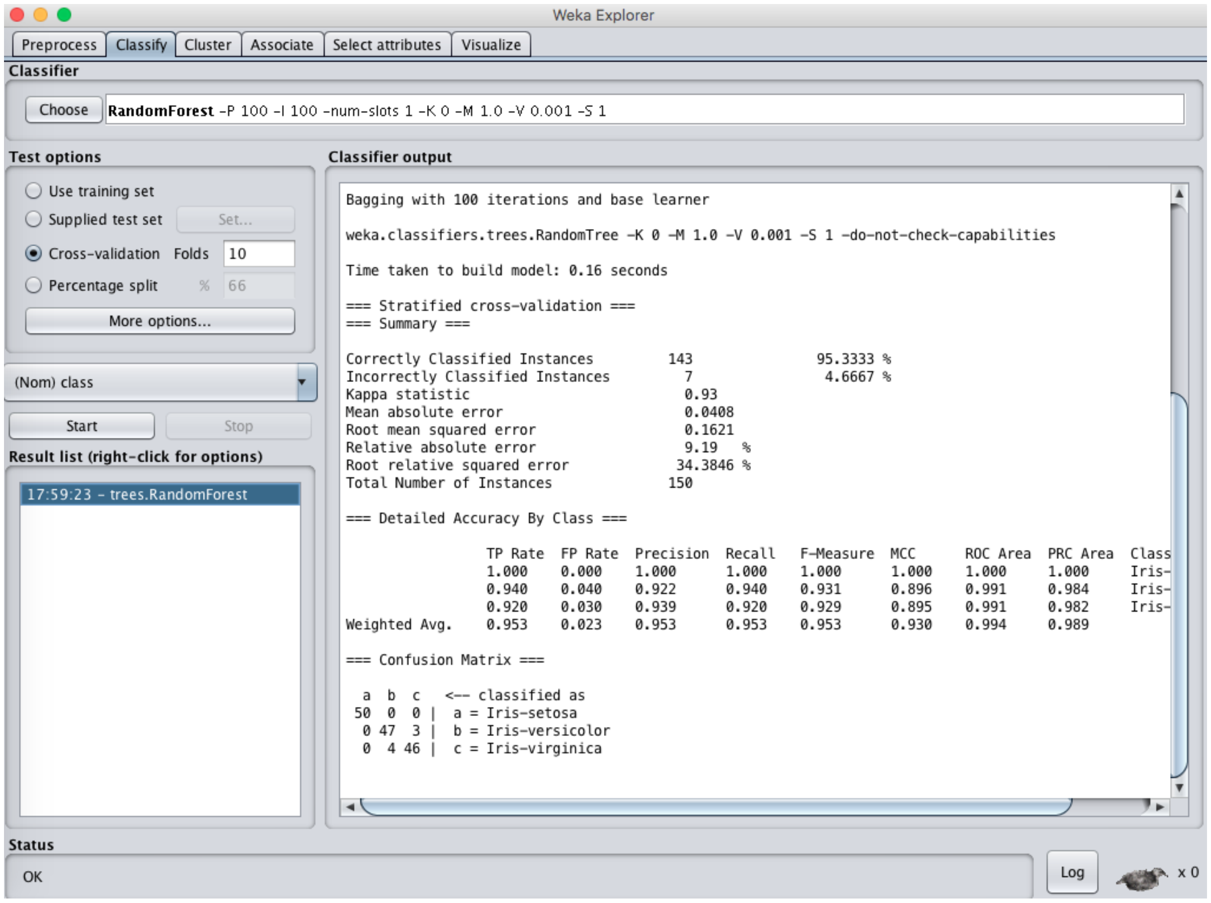
Files of Type: Arff data files (*.arff)

Welcome to the Weka Explorer

A comma-separated list. A set of data items, the dataset, is a very basic concept of machine learning. A dataset is roughly equivalent to a two-dimensional spreadsheet or database table. In WEKA,it is implemented by the weka.core.Instances class. A dataset is a collection of examples, each one of class weka.core.Instance. Each Instance consists of a number of attributes, any of which can be nominal (= one of a predefined list of values), numeric (= a real or integer number) or a string (= an arbitrary long list of characters, enclosed in ”double quotes”). Additional types are date and relational, which are not covered here but in the ARFF chapter. The external representation of an Instances class is an ARFF file, which consists of a header describing the attribute types and the data.

Selecting & Running the algorithm:

Now that you have loaded a dataset, it’s time to choose a machine learning algorithm to model the problem and make predictions. Click the “Classify” tab. This is the area for running algorithms against a loaded dataset in Weka. You will note that the “ZeroR” algorithm is selected by default. Click the “Start” button to run this algorithm.



RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1

Cross-validation [✓] Folds 10
Percentage split % 66
(Nom) class
17:59:23 - trees.RandomForest
OK

Classifier output

```
Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.16 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               95.3333 %
Incorrectly Classified Instances         7                4.6667 %
Kappa statistic                          0.93
Mean absolute error                      0.0408
Root mean squared error                  0.1621
Relative absolute error                  9.19   %
Root relative squared error             34.3846 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-
                 0.940    0.040    0.922      0.940    0.931      0.896    0.991     0.984     Iris-
                 0.920    0.030    0.939      0.920    0.929      0.895    0.991     0.982     Iris-
Weighted Avg.    0.953    0.023    0.953      0.953    0.953      0.930    0.994     0.989

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

Firstly, note the Classification Accuracy. You can see that the model achieved a result of 144/150 correct or 96%, which seems a lot better than the baseline of 33%.Secondly, look at the Confusion Matrix. You can see a table of actual classes compared to predicted classes and you can see that there was 1 error where an Iris-setosa was classified as an Iris-versicolor, 2 cases where Iris-virginica was classified as an Iris-versicolor, and 3 cases where an Iris-versicolor was classified as an Iris-setosa (a total of 6 errors). This table can help to explain the accuracy achieved by the algorithm.

Viva Voce-

Somin Wadhwa
01996402714

Experiment-2

Aim- Understanding of Machine Learning Algorithms

Theory-

Some of the popularly used Machine Learning algorithms are-

• **Support Vector Machine (SVM)** - SVM discriminates a set of high-dimension features using a or sets of hyper planes that gives the largest minimum distance to separates all data points among classes.

• **Multilayer Perceptron Neural Network (MLP)** - MLP is a non-linear feed-forward network model which maps a set of inputs x onto a set of outputs y using multi weights connections.

• **Bayesian Network (BN)** - A BN is a probabilistic graphical model for reasoning under uncertainty, where the nodes represent discrete or continuous variables and the links represent the relationships between them.

• Decision Tree (DT) - DT decides the target class of a new sample based on selected features from available data using the concept of information entropy. The nodes of the tree are the attributes, each branch of the tree represents a possible decision and the end nodes or leaves are the classes.

• **Random Forest (RF)** - RF works by constructing multiple decision tree s on various sub-samples of the datasets and output the class that appear most often or mean predictions of the decision trees.

• **Nave Bayes (NB)** - The NB classifier is a classification algorithm based on Bayes theorem with strong inde- pendent assumptions between features.

• **K-nearest Neighbour (KNN)** - KNN is an instance-based learning algorithm that store all available data points and classifies the new data points based on similarity measure such as distance. The machine learning ensemble meta-algorithms on the other hand are:

• **Boosting algorithms** - Boosting works by combining a set of weak classifier to a single strong classifier. The weak classifiers are weighted in some way from the training data points or hypotheses into a final strong classifier, thus there are a varieties of boosting algorithms.

Viva Voce-

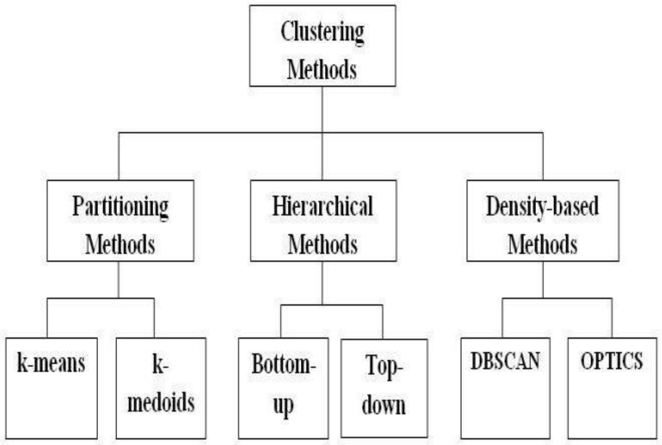
Somin Wadhwa
01996402714

Experiment-3

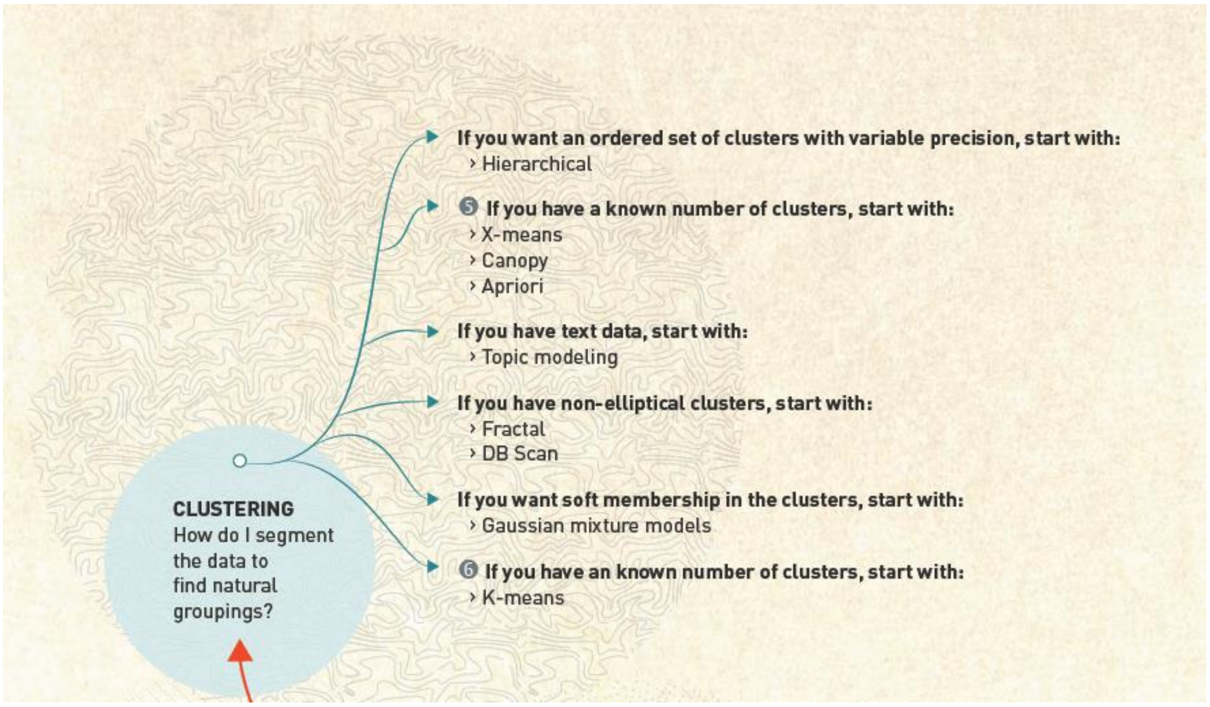
Aim: To study and implement K-Means Algorithm using WEKA.

Theory:

Clustering- Set of techniques used to segment data naturally into similar groups.



Which method to use and when?


CLUSTERING
How do I segment the data to find natural groupings?
If you want an ordered set of clusters with variable precision, start with:
› Hierarchical
5 If you have a known number of clusters, start with:
› X-means
› Canopy
› Apriori
If you have text data, start with:
› Topic modeling
If you have non-elliptical clusters, start with:
› Fractal
› DB Scan
If you want soft membership in the clusters, start with:
› Gaussian mixture models
6 If you have an known number of clusters, start with:
› K-means

Somin Wadhwa
01996402714

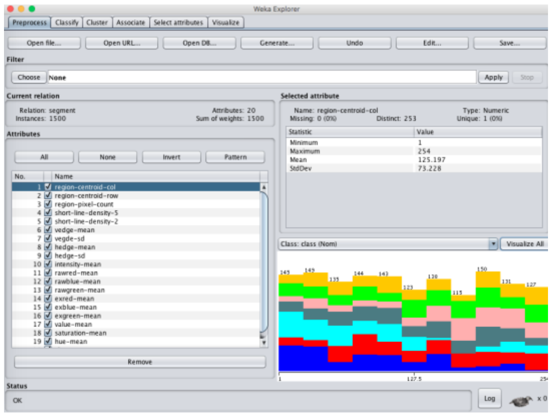
K-Means Algorithm:

Input: k: the number of clusters. D: a data set containing n objects.
Output: A set of k clusters.

Method:

1. Arbitrarily choose k objects from D as the initial cluster centers.
2. Repeat.
3. (Re) assign each object to the cluster to which the object is most similar using Eq 1, based on the mean value of the objects in the cluster.
4. Update the cluster means, i.e. calculate the mean value of the objects for each cluster until no change

Implementation:



Filter: None

Current relation
Relation: segment   Attributes: 20
Instances: 1500   Sum of weights: 1500

Selected attribute
Name: region-centroid-col   Type: Numeric
Missing: 0 (0%)   Distinct: 253   Unique: 1 (0%)

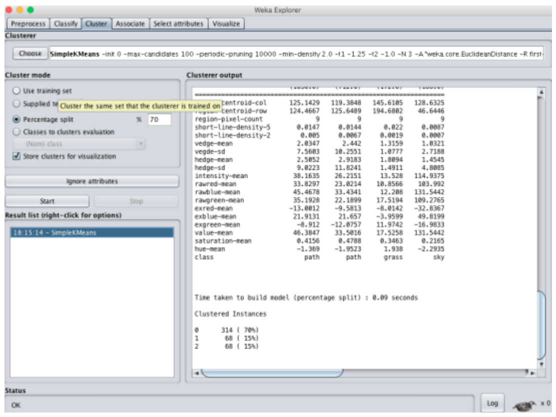
Statistic	Value
Minimum	1
Maximum	254
Mean	125.197
StdDev	73.228

Attributes

No.		Name
1	✓	region-centroid-col
2	✓	region-centroid-row
3	✓	region-pixel-count
4	✓	short-line-density-5
5	✓	short-line-density-2
6	✓	vedge-mean
7	✓	vegde-sd
8	✓	hedge-mean
9	✓	hedge-sd
10	✓	intensity-mean
11	✓	rawred-mean
12	✓	rawblue-mean
13	✓	rawgreen-mean
14	✓	exred-mean
15	✓	exblue-mean
16	✓	exgreen-mean
17	✓	value-mean
18	✓	saturation-mean
19	✓	hue-mean

Class: class (Nom)

Status
OK



Clusterer: SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-

Cluster mode
( ) Use training set
( ) Supplied te
Cluster the same set that the clusterer is trained on
(•) Percentage split % 70
( ) Classes to clusters evaluation
(Nom) class
[✓] Store clusters for visualization

Result list (right-click for options)
18:15:14 - SimpleKMeans

Clusterer output

```
==================================================================
        centroid-col       125.1429   119.3848   145.6105   128.6325
        centroid-row       124.4667   125.6489   194.6802    46.6446
region-pixel-count                9          9          9          9
short-line-density-5         0.0147     0.0144      0.022     0.0087
short-line-density-2          0.005     0.0067     0.0019     0.0007
vedge-mean                   2.0347      2.442     1.3159     1.0321
vegde-sd                     7.5603    10.2551     1.0777     2.7188
hedge-mean                   2.5052     2.9183     1.8094     1.4545
hedge-sd                     9.0223    11.8241     1.4911     4.8085
intensity-mean              38.1635    26.2151     13.528   114.9375
rawred-mean                 33.8297    23.0214    10.8566    103.992
rawblue-mean                45.4678    33.4341     12.208   131.5442
rawgreen-mean               35.1928    22.1899    17.5194   109.2765
exred-mean                 -13.0012    -9.5813    -8.0142   -32.8367
exblue-mean                 21.9131     21.657    -3.9599    49.8199
exgreen-mean                 -8.912   -12.0757    11.9742   -16.9833
value-mean                  46.3847    33.5016    17.5258   131.5442
saturation-mean              0.4156     0.4788     0.3463     0.2165
hue-mean                     -1.369    -1.9523      1.938    -2.2935
class                          path       path      grass        sky



Time taken to build model (percentage split) : 0.09 seconds

Clustered Instances

0      314 ( 70%)
1       68 ( 15%)
2       68 ( 15%)
```

Status
OK

Viva Voce-

Somin Wadhwa
01996402714

Experiment-4

Aim: To study sample ARFF files in Database.

Theory:

Commonly used files types to load data into WEKA or other tabular formats are-

1. CSV: Comma Separated Variable
2. ARFF: Attribute relation file format
3. XLSX: Excel Sheet

ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information.

The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
   % 1. Title: Iris Plants Database
   %
   % 2. Sources:
   %      (a) Creator: R.A. Fisher
   %      (b) Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)
   %      (c) Date: July, 1988
   %
   @RELATION iris

   @ATTRIBUTE sepallength  NUMERIC
   @ATTRIBUTE sepalwidth   NUMERIC
   @ATTRIBUTE petallength  NUMERIC
   @ATTRIBUTE petalwidth   NUMERIC
   @ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-
virginica}
```

The Data of the ARFF file looks like the following:

```
  @DATA
   5.1,3.5,1.4,0.2,Iris-setosa
   4.9,3.0,1.4,0.2,Iris-setosa
   4.7,3.2,1.3,0.2,Iris-setosa
   4.6,3.1,1.5,0.2,Iris-setosa
   5.0,3.6,1.4,0.2,Iris-setosa
   5.4,3.9,1.7,0.4,Iris-setosa
   4.6,3.4,1.4,0.3,Iris-setosa
   5.0,3.4,1.5,0.2,Iris-setosa
   4.4,2.9,1.4,0.2,Iris-setosa
   4.9,3.1,1.5,0.1,Iris-setosa
```

Lines that begin with a % are comments. The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.

Somin Wadhwa
01996402714

Examples:

1 Airlines:

Monthly totals of international airline passengers (in thousands) for
1949-1960.

```
@relation airline_passengers
@attribute passenger_numbers numeric
@attribute Date date 'yyyy-MM-dd'

@data
112,1949-01-01
118,1949-02-01
132,1949-03-01
129,1949-04-01
121,1949-05-01
135,1949-06-01
148,1949-07-01
148,1949-08-01
432,1960-12-01
```

2 Breast Cancer (University of Wisconsin-Madison):

This data set includes 201 instances of one class and 85 instances of another class. The instances are described by 9 attributes, some of which are linear and some are nominal.

Number of Instances: 286
Number of Attributes: 9 + the class attribute
Attribute Information:

1. Class: no-recurrence-events, recurrence-events
2. age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
3. menopause: lt40, ge40, premeno.
4. tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.
5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
6. node-caps: yes, no.
7. deg-malig: 1, 2, 3.
8. breast: left, right.
9. breast-quad: left-up, left-low, right-up, right-low, central.
10. irradiat: yes, no.

Missing Attribute Values: (denoted by "?")
Attribute #: Number of instances with missing values 8
Class Distribution:

1. no-recurrence-events: 201 instances
2. recurrence-events: 85 instances

Num Instances: 286
Num Attributes: 10

```
Num Continuous:    0 (Int 0 / Real 0)
Num Discrete:      10
Missing values:    9 /  0.3
     name                 type enum ints real     missing    distinct  (1)
   1 'age'                Enum 100    0    0      0 /  0      6 /  2    0
   2 'menopause'          Enum 100    0    0      0 /  0      3 /  1    0
   3 'tumor-size'         Enum 100    0    0      0 /  0     11 /  4    0
   4 'inv-nodes'          Enum 100    0    0      0 /  0      7 /  2    0
   5 'node-caps'          Enum  97    0    0      8 /  3      2 /  1    0
   6 'deg-malig'          Enum 100    0    0      0 /  0      3 /  1    0
   7 'breast'             Enum 100    0    0      0 /  0      2 /  1    0
   8 'breast-quad'        Enum 100    0    0      1 /  0      5 /  2    0
   9 'irradiat'           Enum 100    0    0      0 /  0      2 /  1    0
  10 'Class'              Enum 100    0    0      0 /  0      2 /  1    0
```

```
@relation breast-cancer
@attribute age {'10-19','20-29','30-39','40-49','50-59','60-
69','70-79','80-89','90-99'}
@attribute menopause {'lt40','ge40','premeno'}
@attribute tumor-size {'0-4','5-9','10-14','15-19','20-24','25-
29','30-34','35-39','40-44','45-49','50-54','55-59'}
@attribute inv-nodes {'0-2','3-5','6-8','9-11','12-14','15-
17','18-20','21-23','24-26','27-29','30-32','33-35','36-39'}
@attribute node-caps {'yes','no'}
@attribute deg-malig {'1','2','3'}
@attribute breast {'left','right'}
@attribute breast-quad
{'left_up','left_low','right_up','right_low','central'}
@attribute 'irradiat' {'yes','no'}
@attribute 'Class' {'no-recurrence-events','recurrence-events'}
@data
'40-49','premeno','15-19','0-
2','yes','3','right','left_up','no','recurrence-events'
'50-59','ge40','15-19','0-2','no','1','right','central','no','no-
recurrence-events'
'50-59','ge40','35-39','0-
2','no','2','left','left_low','no','recurrence-events'
'40-49','premeno','35-39','0-
```

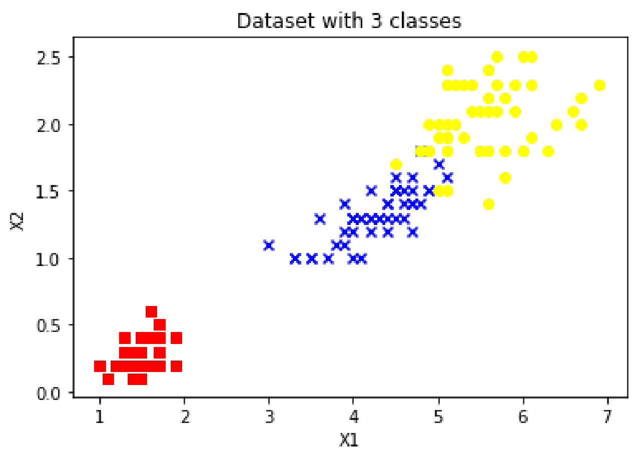
Viva Voce-

Somin Wadhwa
01996402714

Experiment-5

Aim: Study & implement major classification algorithms.

Dataset Used: IRIS



Source Code: https://github.com/sominwadhwa/ML-Homework/blob/master/major_clfs_5/major_clfs.ipynb

1. Gaussian Naïve Bayes Algorithm

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c|x).

$$\underbrace{P(c\,|\,x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\,|\,c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\,|\,X) = P(x_1\,|\,c)\times P(x_2\,|\,c)\times\cdots\times P(x_n\,|\,c)\times P(c)$$

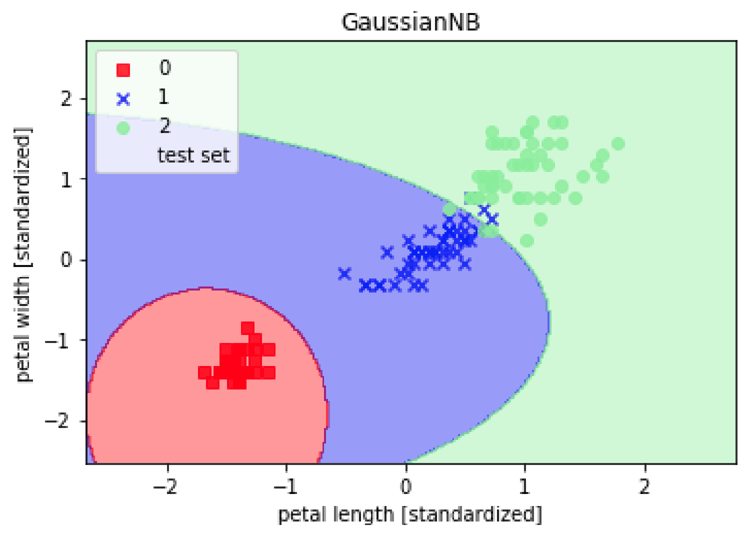
In the above above fig,

- P(c|x) is the posterior probability of class (c, target) given predictor (x,attrs).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.

- P(x) is the prior probability of predictor.

```
lr = GaussianNB()
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassfied samples: %d' % (y_test != y_pred).sum())
#print ('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=lr, test_
idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.title("GaussianNB")
plt.show()
```

```
Misclassfied samples: 0
```

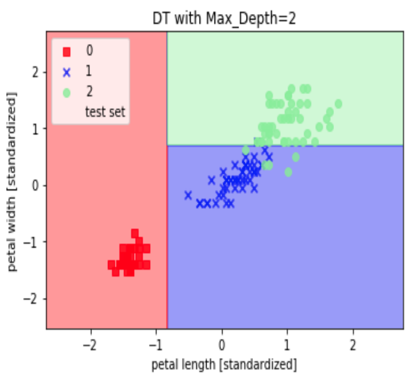


2. Decision Tree Classification

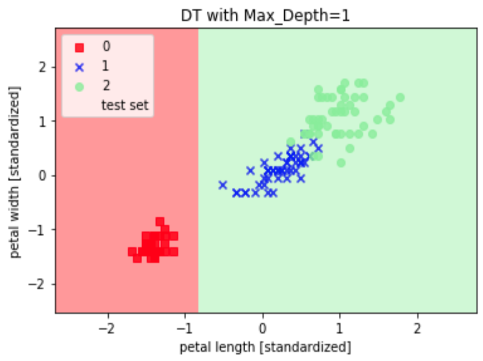
A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

```
lr = DecisionTreeClassifier(max_depth=2)
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassfied samples: %d' % (y_test != y_pred).sum())
#print ('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=lr, test_
idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.title("DT with Max_Depth=2")
plt.show()
```

```
Misclassfied samples: 0
```



```
Misclassfied samples: 13
```

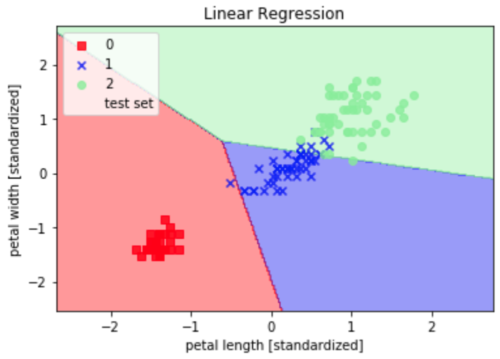


3. Logistic Regression Classifier

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

```
lr = LogisticRegression()
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassfied samples: %d' % (y_test != y_pred).sum())
#print ('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=lr, test_
idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.title("Linear Regression")
plt.show()
```

```
Misclassfied samples: 2
```

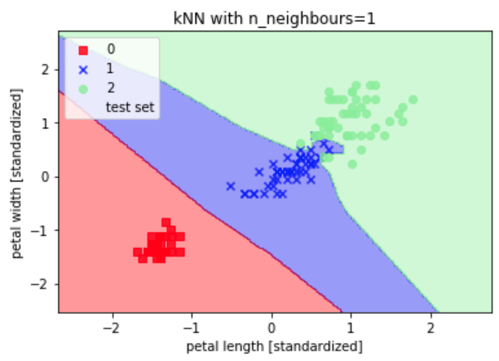


4. K-Nearest Neighbour

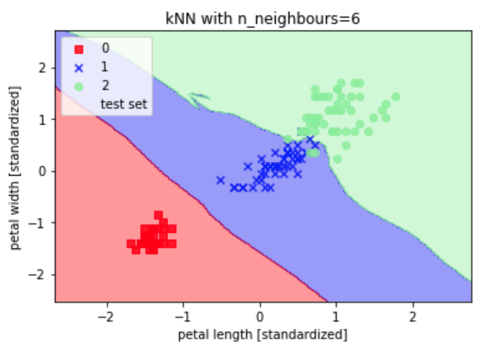
It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function. These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

```
lr = KNeighborsClassifier(n_neighbors=1, n_jobs=-1)
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassfied samples: %d' % (y_test != y_pred).sum())
#print ('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=lr, test_
idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.title("kNN with n_neighbours=1")
plt.show()
```

```
Misclassfied samples: 1
```



```
Misclassfied samples: 0
```



5. Support Vector Machine

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

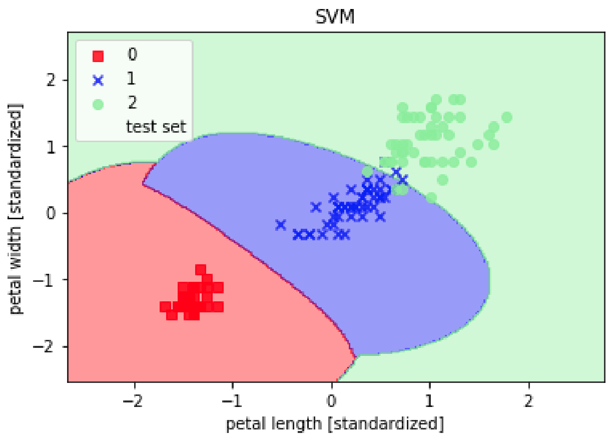
For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)

we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.

```
lr = SVC(C=1)
lr.fit(X_train_std, y_train)
```

```
y_pred = lr.predict(X_test_std)
print('Misclassfied samples: %d' % (y_test != y_pred).sum())
#print ('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=lr, test_
idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.title("SVM")
plt.show()
```

```
Misclassfied samples: 0
```



Viva Voce-

Somin Wadhwa
01996402714

Experiment-6

Aim: Supermarket Dataset on WEKA.

Here true value of interest is taken as θ and the value estimated using some algorithm as θ^. Correlation tells you how much θ and θ^ are related. It gives values between −1−1 and 11, where 00 is no relation, 11 is very strong, linear relation and −1−1 is an inverse linear relation (i.e. bigger values of θ indicate smaller values of θ^, or vice versa). Mean absolute error is:

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} |\hat{\theta}_i - \theta_i|$$

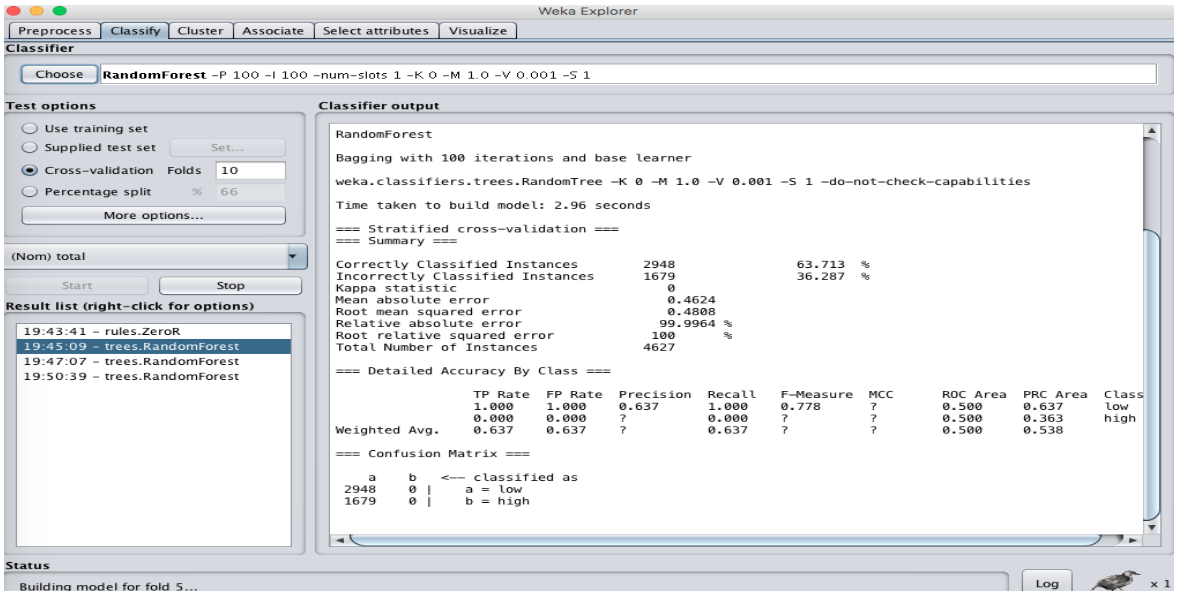
Root mean square error is:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}$$

Some terms used :

- **TP Rate**: rate of true positives (instances correctly classified as a given class)
- **FP Rate**: rate of false positives (instances falsely classified as a given class)
- **Precision**: proportion of instances that are truly of a class divided by the total instances classified as that class
- **Recall**: proportion of instances classified as a given class divided by the actual total in that class (equivalent to TP rate)
- **F-Measure**: A combined measure for precision and recall calculated as 2 * Precision * Recall / (Precision + Recall)

RandomForest



RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1

Test options: Use training set / Supplied test set / [✓ Cross-validation] Folds 10 / Percentage split % 66

(Nom) total

Result list (right-click for options)

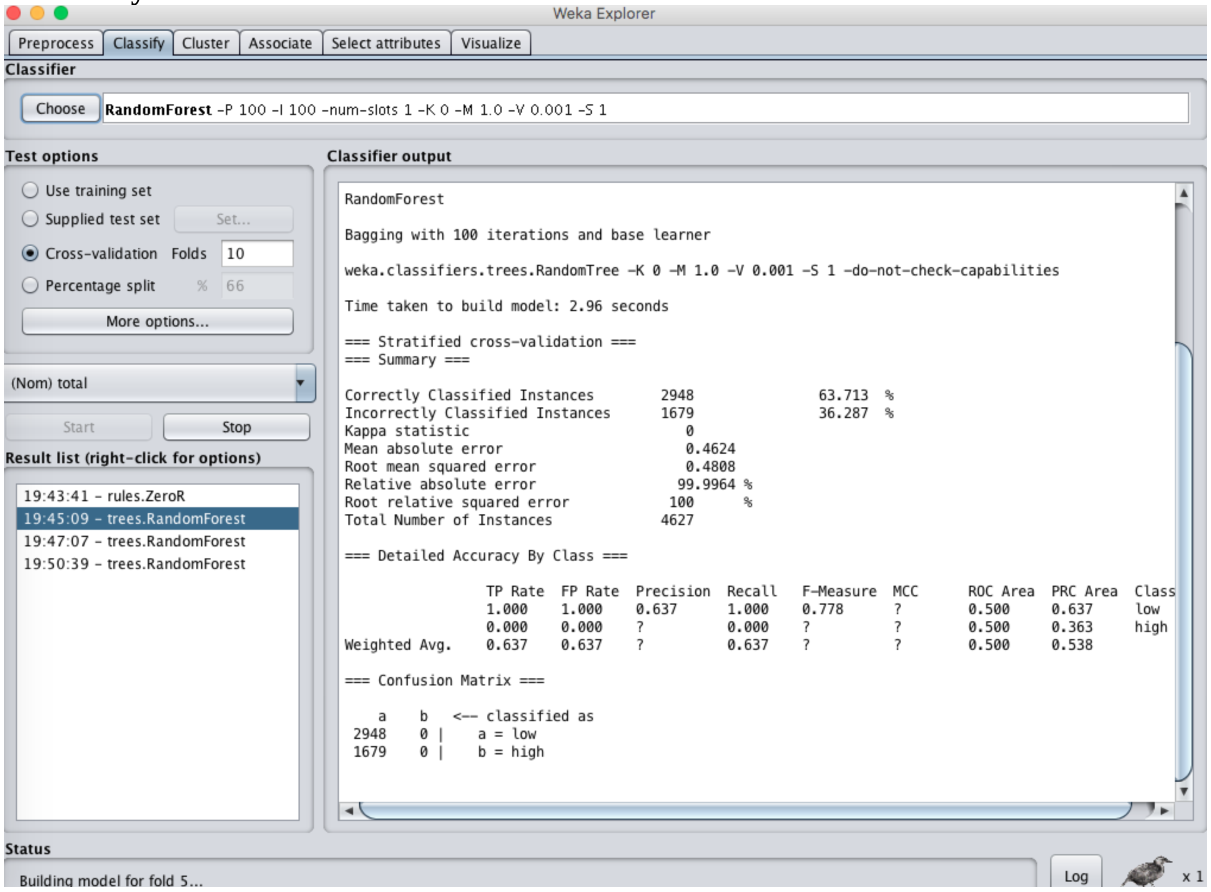
19:43:41 - rules.ZeroR
19:45:09 - trees.RandomForest
19:47:07 - trees.RandomForest
19:50:39 - trees.RandomForest

```
RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 2.96 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2948               63.713  %
Incorrectly Classified Instances      1679               36.287  %
Kappa statistic                          0
Mean absolute error                      0.4624
Root mean squared error                  0.4808
Relative absolute error                 99.9964 %
Root relative squared error            100      %
Total Number of Instances             4627

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.637      1.000    0.778      ?        0.500     0.637     low
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.363     high
Weighted Avg.    0.637    0.637    ?          0.637    ?          ?        0.500     0.538

=== Confusion Matrix ===

    a    b   <-- classified as
 2948    0 |    a = low
 1679    0 |    b = high
```

Building model for fold 5...

Somin Wadhwa
01996402714

Naïve-Bayes



RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1

19:43:41 - rules.ZeroR
19:45:09 - trees.RandomForest
19:47:07 - trees.RandomForest
19:50:39 - trees.RandomForest

```
RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 2.96 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2948               63.713  %
Incorrectly Classified Instances      1679               36.287  %
Kappa statistic                          0
Mean absolute error                      0.4624
Root mean squared error                  0.4808
Relative absolute error                 99.9964 %
Root relative squared error            100      %
Total Number of Instances             4627

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.637      1.000    0.778      ?        0.500     0.637     low
                 0.000    0.000    ?          0.000    ?          ?        0.500     0.363     high
Weighted Avg.    0.637    0.637    ?          0.637    ?          ?        0.500     0.538

=== Confusion Matrix ===

    a    b   <-- classified as
 2948    0 |    a = low
 1679    0 |    b = high
```

Building model for fold 5...

Viva Voce-

Somin Wadhwa
01996402714

Experiment-7

Aim: Implement kNN classification method and estimate different metrics like precision, recall, F-Measure and AUC-ROC.

Dataset Used: Breast Cancer by University of Wisconsin-Madison.

Source available at: https://github.com/sominwadhwa/ML-Homework/blob/master/breast_cancer_7/knn.ipynb

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970‟s as a non-parametric technique.

Algorithm: A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Source:

Train-Test Split: 80:20

```
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, stratify=data.target, random_state=66, test_size=0.2) print
(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(455, 30) (455,) (114, 30) (114,)
```

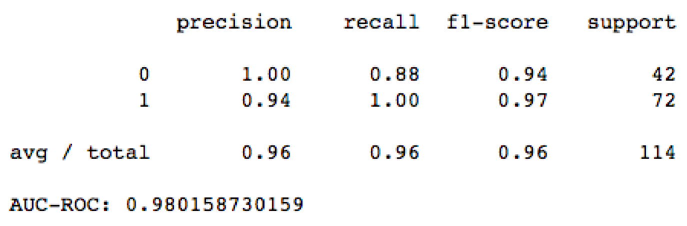
Hyper-parametrized Pipeline to train kNN-

```
pipeline = Pipeline([
    ('standardize', StandardScaler()),
    ('grid_search_lr', GridSearchCV(
        KNeighborsClassifier(),
        param_grid={'n_neighbors': [3,5,7,10],
                    'p':[1,2],
                    },
        cv=5,
        n_jobs=-1,
        scoring='roc_auc',
        verbose=2,
        refit=True
    ))
])
```

```
pipeline.fit(X_train, y_train) y_pred = pipeline.predict(X_test) y_scores
= pipeline.predict_proba(X_test)
```

----------------5 Folds of 8 Candidates totalling 40 fits-----------------------------------

```
print (classification_report(y_pred=y_pred, y_true=y_test))
print ("AUC-ROC: "+str(roc_auc_score(y_score=y_scores[:,1], y_true=y_test)
))
```



```
             precision    recall  f1-score   support

          0       1.00      0.88      0.94        42
          1       0.94      1.00      0.97        72

avg / total       0.96      0.96      0.96       114

AUC-ROC: 0.980158730159
```

Viva-Voce:

Somin Wadhwa
01996402714

Experiment-8

Aim: Understanding Python & its basics.

Source available at: https://github.com/sominwadhwa/ML-Homework/blob/master/Understanding_Python_8/intro-to-python.ipynb

1 Introduction to Python

For Machine Learning Lab (ETCS-454)

2 Python is interpreted

- Python is an *interpreted* language, in contrast to Java and C which are compiled languages.
- This means we can type statements into the interpreter and they are executed immediately.

```
In [5]: 5 + 5

Out[5]: 10

In [2]: x = 5
        y = 'Hello There'
        z = 10.5

In [3]: x + 5

Out[3]: 10
```

3 Assignments versus equations

- In Python when we write x = 5 this means something different from an equation $x = 5$.
- Unlike variables in mathematical models, variables in Python can refer to different things as more statements are interpreted.

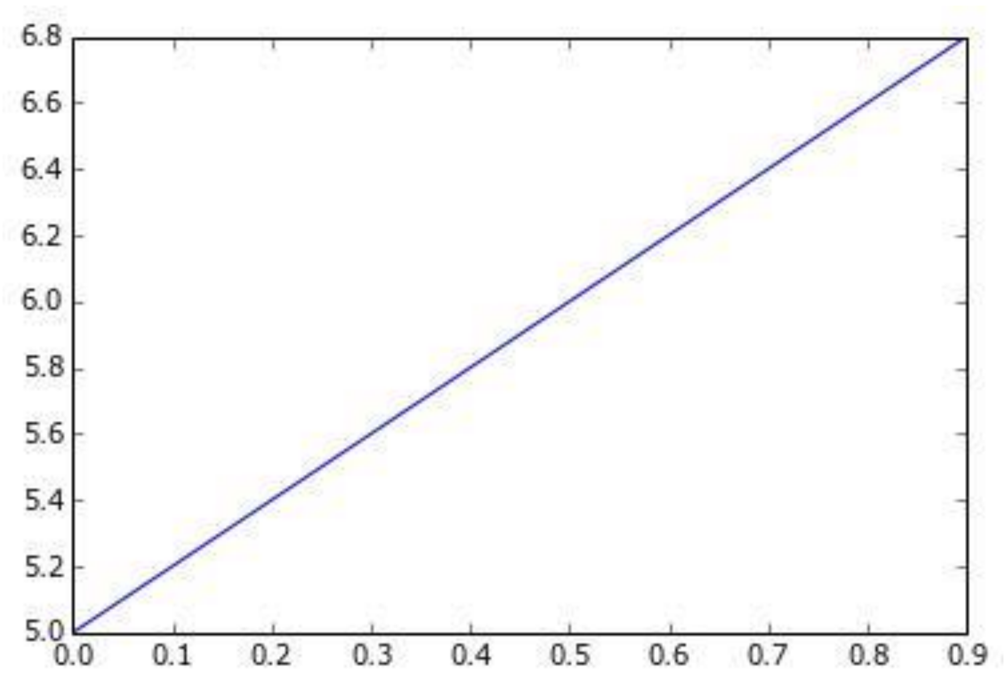
```
In [14]: x = 1
         print 'The value of x is ', x

         x = 2.5
         print 'Now the value of x is ', x

         x  = 'hello there'
         print 'Now it is ', x
```

```
The value of x is 1
Now the value of x is 2.5
Now it is hello there
```

7 Basic Plotting

1. We will use a module called matplotlib to plot some simple graphs.
2. This module provides functions which are very similar to MATLAB plotting commands.

In [12]:
```
import matplotlib.pyplot as plt
%matplotlib inline

y = x*2 + 5
plt.plot(x, y
```



More such functions are available at: https://github.com/sominwadhwa/ML-Homework/blob/master/Understanding_Python_8/intro-to-python.ipynb

Viva Voce:

Somin Wadhwa
01996402714

Experiment-9

Aim: Using python to implement Linear Model for Regression Analysis.

Dataset: Housing Price Prediction

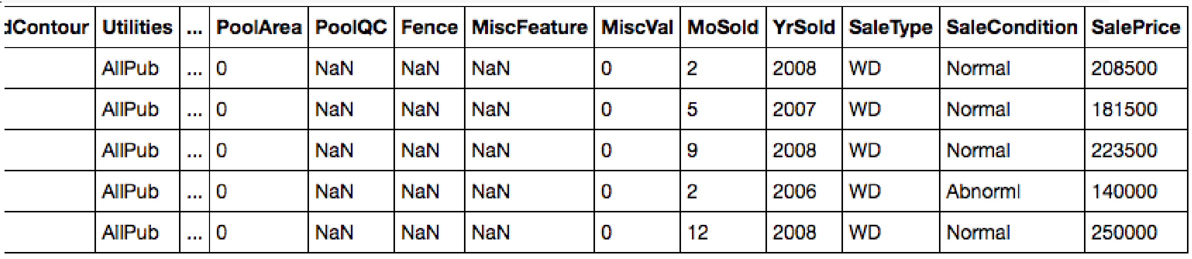
Source available at https://github.com/sominwadhwa/ML-Homework/blob/master/linear_regression_models_9/reg.ipynb

Steps followed:

Data Loading

```
train_DF = pd.read_csv('train.csv')

test_DF = pd.read_csv('test.csv')
train_DF.head()
```

Out[3]:



dContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

Retaining only Numeric Types while Fixing Skewed Ones

```
numeric_features_train = concat.dtypes[train_DF.dtypes != 'object'].ind
ex

skewed_features_train = concat[numeric_features_train].apply(lambda x:
skew(x.dropna()))

skewed_features_train = skewed_features_train[skewed_features_train > 0
.75]
skewed_features_train = skewed_features_train.index

concat[skewed_features_train] = np.log1p(concat[skewed_features_train])
```

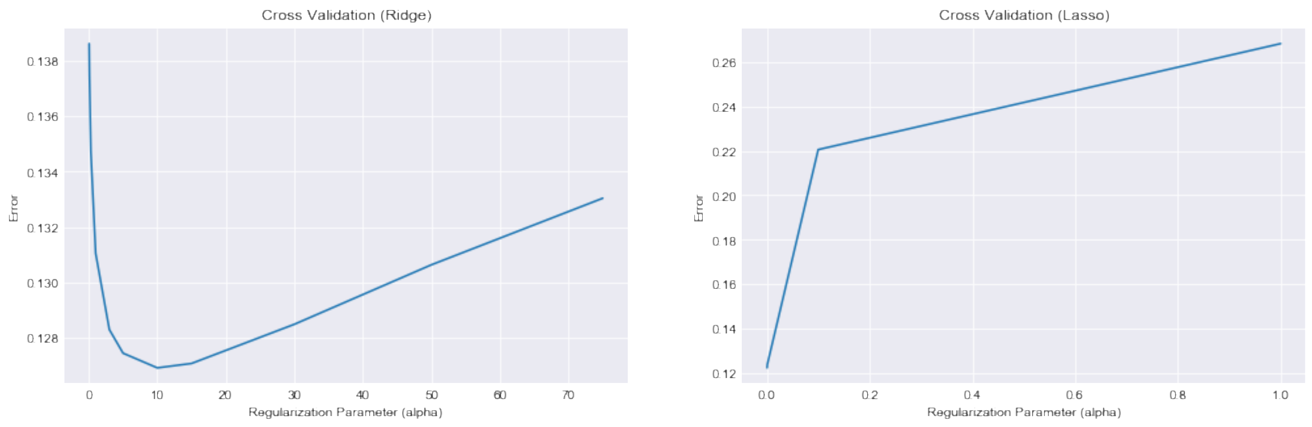
```
concat = pd.get_dummies(concat)
#Fill in empty values with mean of each column
concat = concat.fillna(train_DF.mean())
```

DEFINE CROSS VAL ERROR

```
def cv_error(model):
    cve= np.sqrt(-cross_val_score(model, X, Y, scoring="neg_mean_square
d_error", cv = 5))
    return(cve)
```

RIDGE & LASSO REGRESSION IMPLEMENTATION

```
#Ridge
a_ridge = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cvScores_ridge = [cv_error(Ridge(alpha = alpha)).mean() for alpha in a_
ridge]
cvScores_ridge = pd.Series(cvScores_ridge, index = a_ridge)
minimum_error = cvScores_ridge.min()

#Lasso
a_lasso = [1, 0.1, 0.001, 0.0005]
cvScores_lasso = [cv_error(Lasso(alpha = alpha)).mean() for alpha in a_
lasso]
cvScores_lasso = pd.Series(cvScores_lasso, index = a_lasso)
minimum_error_lasso = cvScores_lasso.min()

#Plots
figure, (ax1, ax2) = plt.subplots(1,2,figsize = (17,5))
ax1.plot(cvScores_ridge)
ax1.set_title("Cross Validation (Ridge)")
ax1.set_xlabel("Regularization Parameter (alpha)")
ax1.set_ylabel("Error")
ax2.plot(cvScores_lasso)
ax2.set_title("Cross Validation (Lasso)")
ax2.set_xlabel("Regularization Parameter (alpha)")
ax2.set_ylabel("Error")
```



```
print("Minimum Error for Ridge Model: ", minimum_error)
print("Minimum Error for Lasso Model: ", minimum_error_lasso)
```

```
Minimum Error for Ridge Model: 0.126915218001
Minimum Error for Lasso Model: 0.122423391373
```

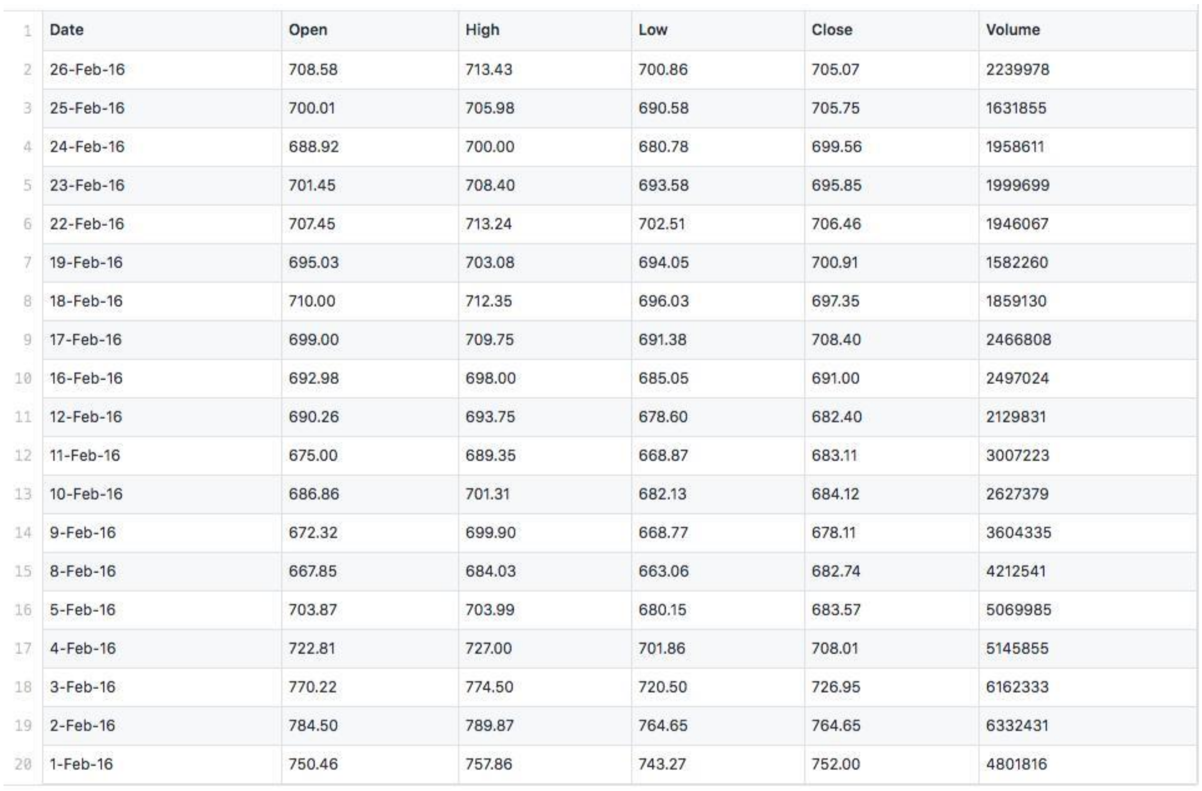
Viva-Voce

Somin Wadhwa
01996402714

Experiment – 10

Aim: Prediction of stock prices based on their previous value using a linear model

Dataset: (Google Stock Prices)



Date	Open	High	Low	Close	Volume
26-Feb-16	708.58	713.43	700.86	705.07	2239978
25-Feb-16	700.01	705.98	690.58	705.75	1631855
24-Feb-16	688.92	700.00	680.78	699.56	1958611
23-Feb-16	701.45	708.40	693.58	695.85	1999699
22-Feb-16	707.45	713.24	702.51	706.46	1946067
19-Feb-16	695.03	703.08	694.05	700.91	1582260
18-Feb-16	710.00	712.35	696.03	697.35	1859130
17-Feb-16	699.00	709.75	691.38	708.40	2466808
16-Feb-16	692.98	698.00	685.05	691.00	2497024
12-Feb-16	690.26	693.75	678.60	682.40	2129831
11-Feb-16	675.00	689.35	668.87	683.11	3007223
10-Feb-16	686.86	701.31	682.13	684.12	2627379
9-Feb-16	672.32	699.90	668.77	678.11	3604335
8-Feb-16	667.85	684.03	663.06	682.74	4212541
5-Feb-16	703.87	703.99	680.15	683.57	5069985
4-Feb-16	722.81	727.00	701.86	708.01	5145855
3-Feb-16	770.22	774.50	720.50	726.95	6162333
2-Feb-16	784.50	789.87	764.65	764.65	6332431
1-Feb-16	750.46	757.86	743.27	752.00	4801816

Source Code:

Source available at: https://github.com/sominwadhwa/ML-Homework/blob/master/stock_price_predictor/goog.csv

1. Data Preprocessing:

```
dates = list(data['Date'].apply(lambda x: int(x.split('-')[0])))
prices = list(data['Open'])

print (dates, prices)
```

2. Model Initialization & Data Loading

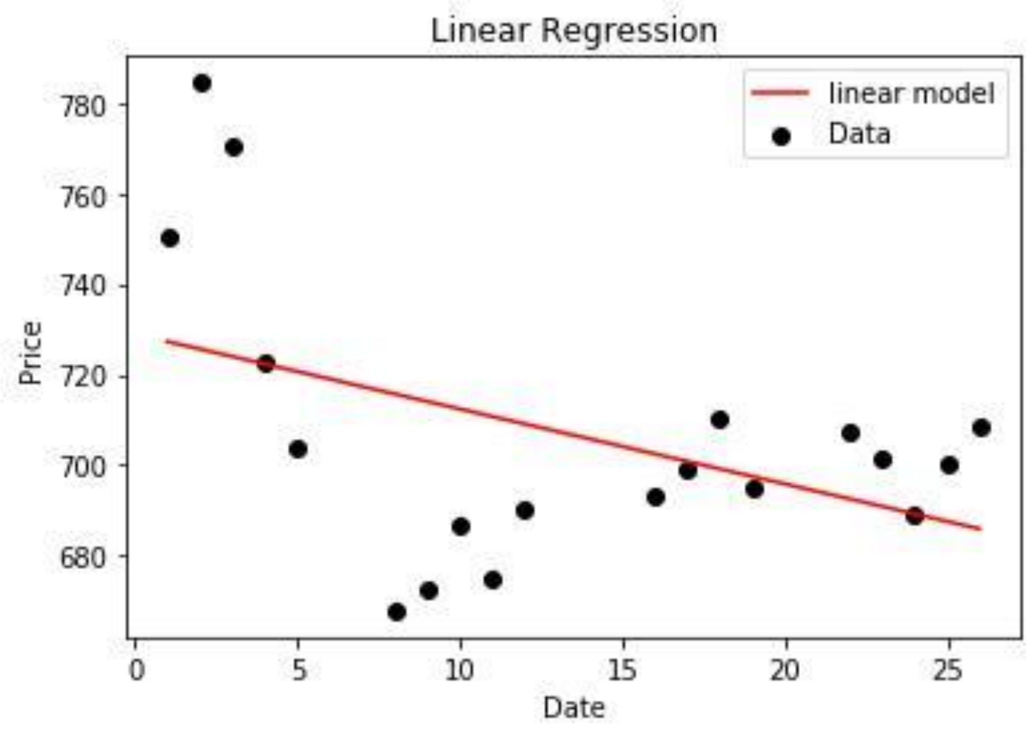
```
linear = LinearRegression()
dates = np.reshape(dates,
(len(dates),1)) prices =
np.reshape(prices, (len(prices),1))

linear.fit(dates,prices)
```

3. Results Plotting

```
plt.plot(dates, linear.predict(dates), color='red', label='linear
model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

Observation:

Residual Plot is shown below for the trained linear model.



Somin Wadhwa
01996402714

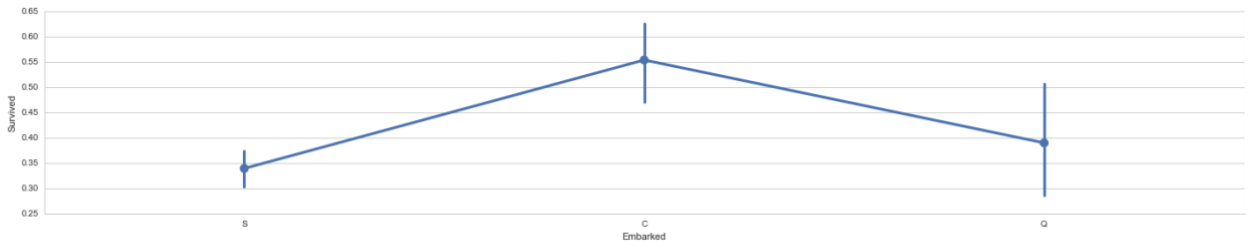
Experiment-11

Aim: Case Study: RMS Titanic.

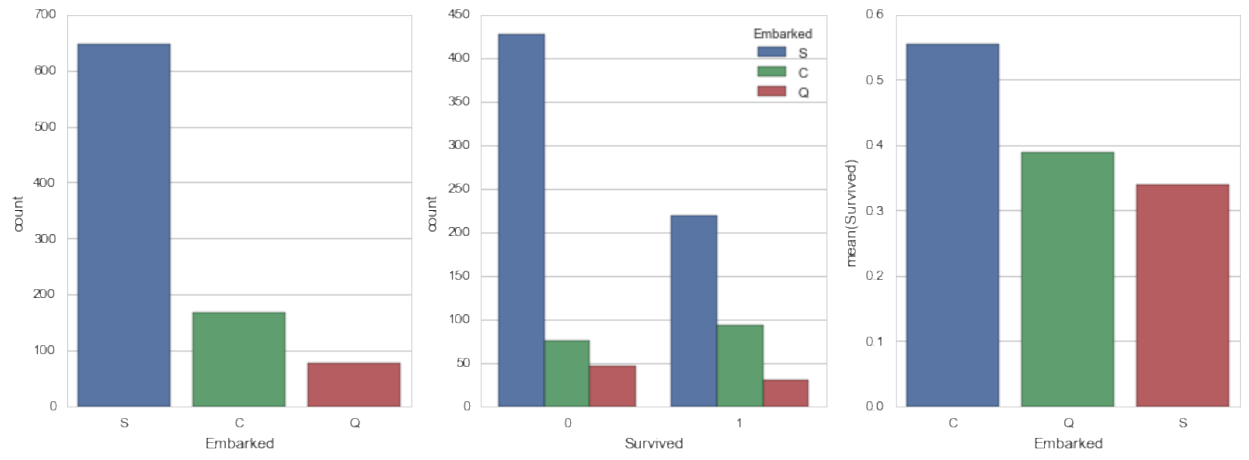
Source Code: https://github.com/sominwadhwa/ML-Homework/blob/master/11_titanic_case_study/ Titanic.ipynb

```
#Dropping data irrelevant to analysis
train_DF = train_DF.drop(['PassengerId','Name','Ticket'], axis = 1, inplac
e = False)
```

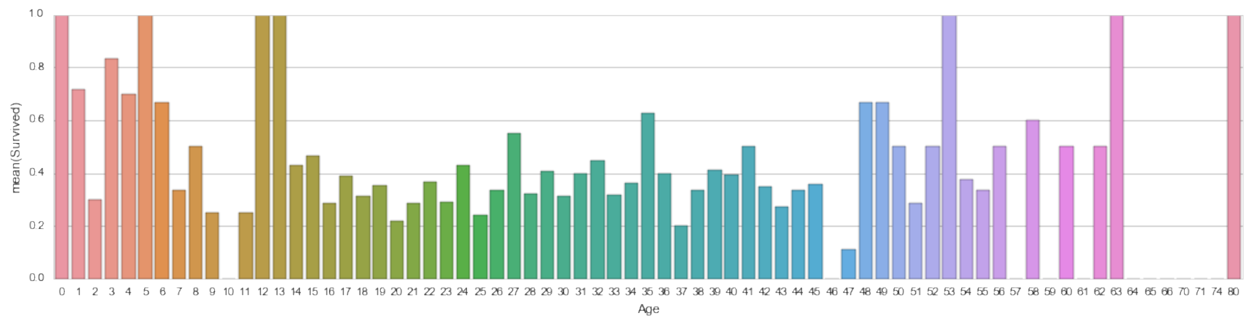
```
train_DF['Embarked'] = train_DF['Embarked'].fillna('S')
sns.factorplot(x = 'Embarked', y = 'Survived', data=train_DF, size = 4, as
pect = 5)
```

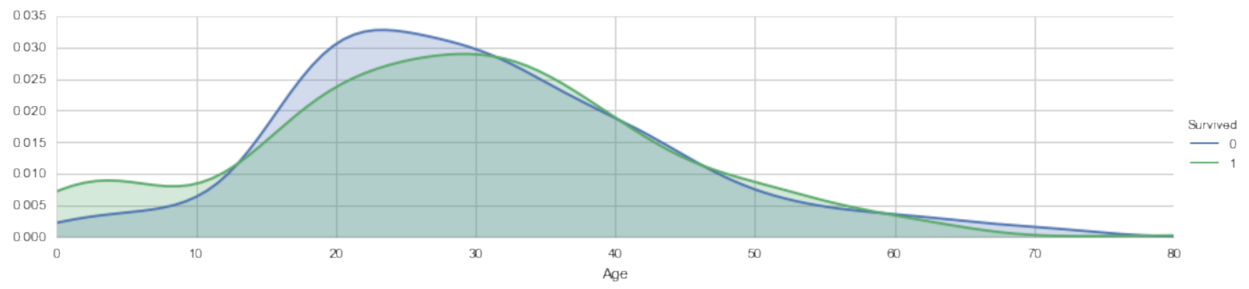


```
figure, (ax1,ax2, ax3) = plt.subplots(1,3,figsize=(15,5))
sns.countplot(x = 'Embarked', data = train_DF, ax = ax1)
sns.countplot(x = 'Survived', hue = 'Embarked', data = train_DF, ax = ax2)
embark_perc = train_DF[["Embarked", "Survived"]].groupby(['Embarked'],as_i
ndex=False).mean()
sns.barplot(x='Embarked', y='Survived', data=embark_perc,ax=ax3)
embark_perc.head()
```



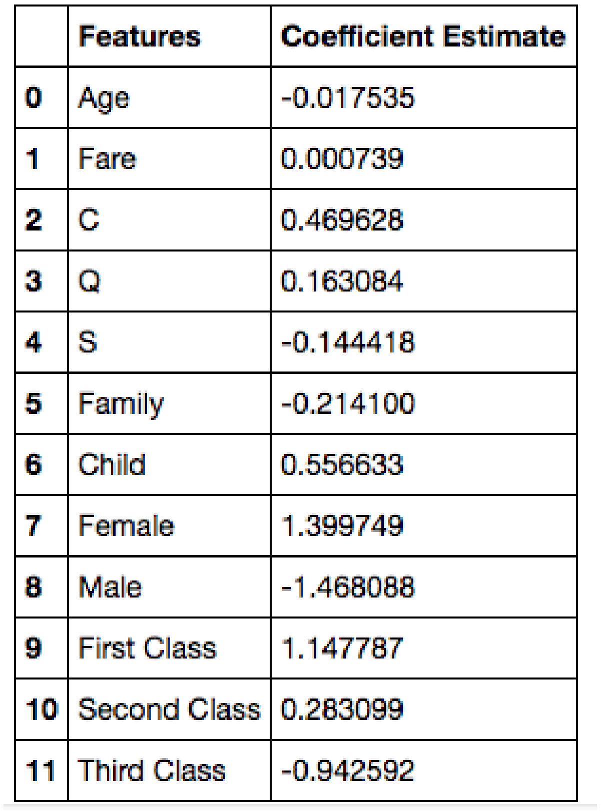
```
facet = sns.FacetGrid(train_DF, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train_DF['Age'].max()))
facet.add_legend()
fig, axis1 = plt.subplots(1,1,figsize=(18,4))
average_age = train_DF[["Age", "Survived"]].groupby(['Age'],as_index=False
).mean()
sns.barplot(x='Age', y='Survived', data=average_age)
```





```
X = train_DF.drop(['Survived'], axis = 1)
Y = train_DF['Survived']
X_test = test_DF.drop(['PassengerId'], axis = 1).copy()
```

```
logr = LogisticRegression()
logr.fit(X,Y)
Y_test = logr.predict(X_test)
logr.score(X,Y)

coeff = DataFrame(train_DF.columns.delete(0))
coeff.columns = ['Features']
coeff['Coefficient Estimate'] = pd.Series(logr.coef_[0])

coeff
```



	Features	Coefficient Estimate
0	Age	-0.017535
1	Fare	0.000739
2	C	0.469628
3	Q	0.163084
4	S	-0.144418
5	Family	-0.214100
6	Child	0.556633
7	Female	1.399749
8	Male	-1.468088
9	First Class	1.147787
10	Second Class	0.283099
11	Third Class	-0.942592

```
logr.score(X,Y)
Out[55]: 0.8080808080808080807
```

Somin Wadhwa
01996402714

Experiment-12

Aim: Text classification with 4-classes using Multinomial Naïve Bayes.

Source Available at: https://github.com/sominwadhwa/ML-Homework/blob/master/12_text-classification/fetch20newcats.ipynb

```
categories = ['alt.atheism', 'soc.religion.christian',
              'comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',
    categories=categories, shuffle=True, random_state=42)
```

```
twenty_train.target_names
```

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
X_train_counts.shape
```

```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
X_train_tf.shape
Out[14]: (2257, 35788)
```

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
```

```
docs_new = ['God is love', 'OpenGL on the GPU is fast']
X_new_counts = count_vect.transform(docs_new)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)

predicted = clf.predict(X_new_tfidf)

for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
```

```
'God is love' => soc.religion.christian
'OpenGL on the GPU is fast' => comp.graphics
```

Somin Wadhwa
01996402714

Experiment-13

Aim: Understanding of dataset of contact patterns among students collected in National University of Singapore.

Source Available at: https://github.com/sominwadhwa/ML-Homework/blob/master/13_NUS_SMS_Corpus/basic_analysis.ipynb

```
with open('SMSSpamCollection') as f:
    content = f.readlines()
# you may also want to remove whitespace characters like `\n` at the end o
f each line
dataset_lines = [x.strip() for x in content]
```

Mapping Texts against their spam labels-

```
labels, texts = zip(*map(lambda line: line.split('\t', 1), dataset_lines))
#print (texts)
labels = np.asarray(list((map(lambda label: (label == 'spam'), labels))))
#print (labels)
texts = np.asarray(texts)
```

```
len(texts), len(labels)
```

```
Out[75]: (5574, 5574)
```

```
count_vectorizer = CountVectorizer()
X_data = count_vectorizer.fit_transform(texts)
```

X_data here is a sparse matrix with ‘5574’ textual CountVectorized inputs.

----------------------------------Build a Baseline--------------------------------------

```
log_reg = LogisticRegression()
log_reg.fit(X_data, labels)
```

```
Out[81]: LogisticRegression(C=1.0, 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)
```

```
test_messages = [
    "FreeMsg: Txt: CALL to No: 86888 & claim your reward of 3 hours talk t
ime to use from your phone now! Subscribe6GB",
    "FreeMsg: Txt: claim your reward of 3 hours talk time",]
```

```
print (' '.join(map(lambda b: str(int(b)), log_reg.predict(count_vectorizer.transform(test_messages)))))
```

```
1 1 --> Correctly Classified
```

```
test_messages2=[
    "Only 99$"
]
```

```
print (' '.join(map(lambda b: str(int(b)), log_reg.predict(count_vectorizer.transform(test_messages2)))))
0 --> Correctly Classified
```

-------------------------Hyper Parameter Optimization---------------------------------------

```
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': np.logspace(-2, 5, num=10),
    'class_weight': ['balanced'],
    'max_iter': [10, 100]
}
```

```
gscv = GridSearchCV(estimator=LogisticRegression(), param_grid=param_grid, scoring='f1', cv=10, verbose=2)
gscv.fit(X_data, labels)
gscv.best_params_
```

-----10 Folds of 40 Candidates totalling 400 fits----------------------

```
Out[99]: {'C': 77.426368268112697,
        'class_weight': 'balanced',
 'max_iter': 10,
 'penalty': 'l2'}
```

Somin Wadhwa
01996402714