INDEX

Experiment 1

Aim: 1. Study and Implement the Naive Bayes learner using WEKA. (Breast Cancer data file).

Theory: 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'.

The 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) from P(c), P(x) and P(x|c). Look at the equation below:

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

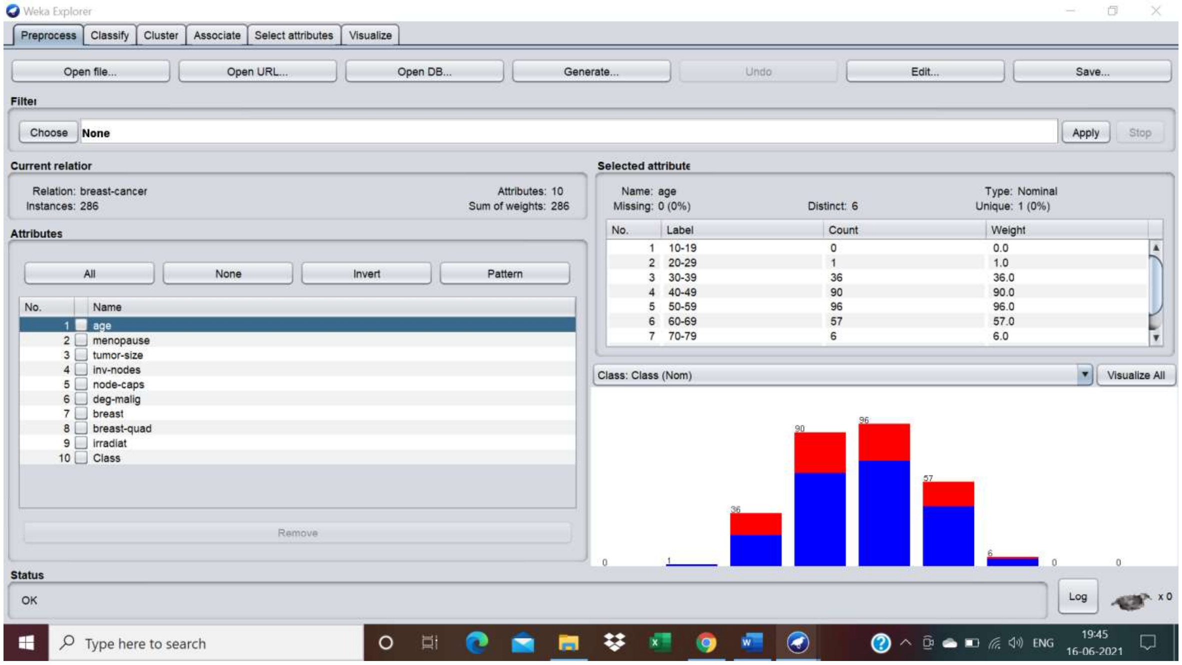
$$P(c\mid X) = P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Above,

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- 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.

First, we use the data mining tools WEKA to do the training data prediction. Here, we will use 10-fold cross validation on training data to calculate the machine learning rules and their performance. The results are as follows:

Output:



Filter
Choose None

Current relation
Relation: breast-cancer
Instances: 286
Attributes: 10
Sum of weights: 286

Attributes

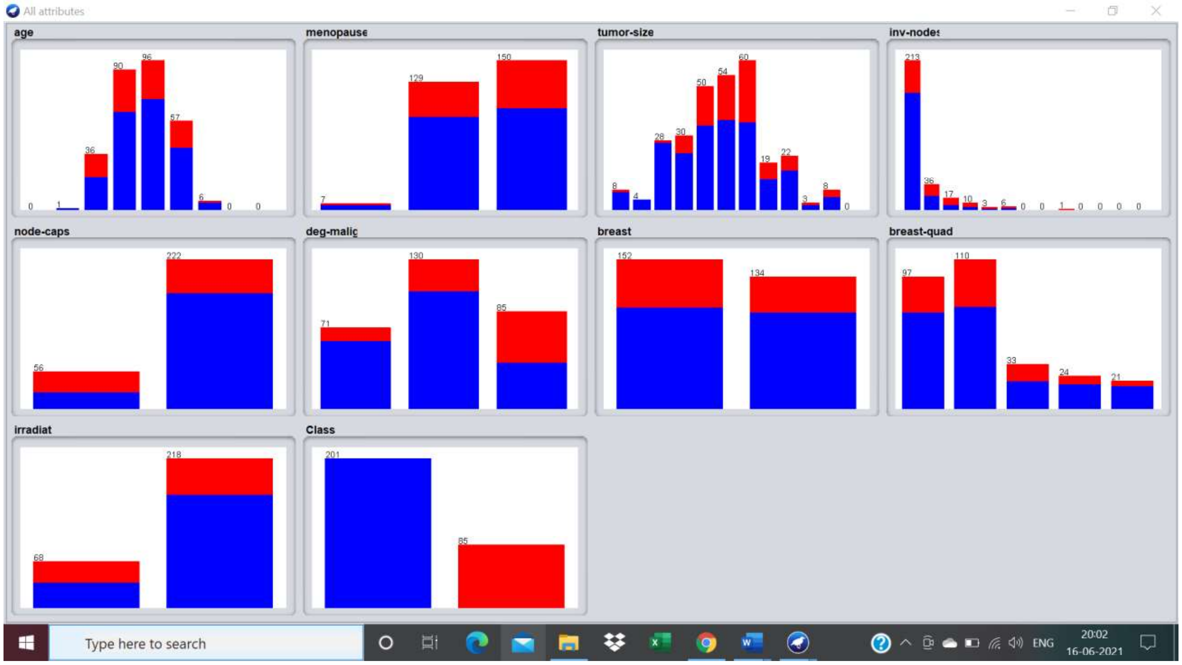
No.	Name
1	age
2	menopause
3	tumor-size
4	inv-nodes
5	node-caps
6	deg-malig
7	breast
8	breast-quad
9	irradiat
10	Class

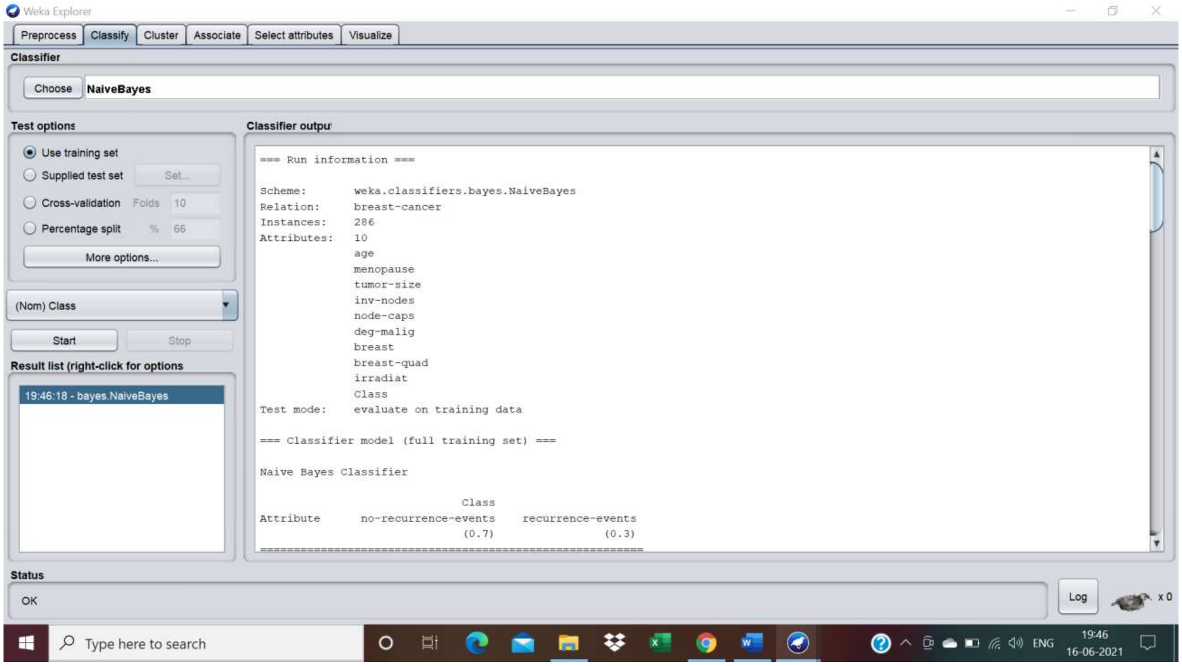
Selected attribute
Name: age
Missing: 0 (0%)
Distinct: 6
Type: Nominal
Unique: 1 (0%)

No.	Label	Count	Weight
1	10-19	0	0.0
2	20-29	1	1.0
3	30-39	36	36.0
4	40-49	90	90.0
5	50-59	96	96.0
6	60-69	57	57.0
7	70-79	6	6.0

Class: Class (Nom)

Status
OK





Classifier: NaiveBayes

Test options:
- ● Use training set
- ○ Supplied test set
- ○ Cross-validation Folds 10
- ○ Percentage split % 66

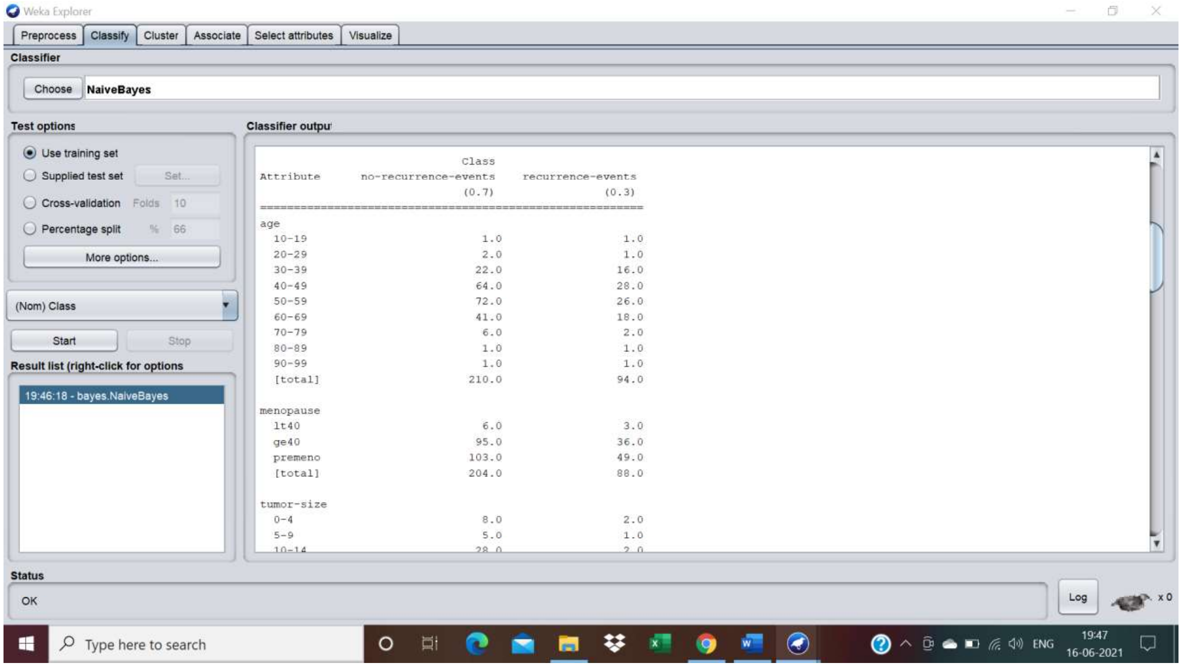
(Nom) Class

Result list: 19:46:18 - bayes.NaiveBayes

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     breast-cancer
Instances:    286
Attributes:   10
              age
              menopause
              tumor-size
              inv-nodes
              node-caps
              deg-malig
              breast
              breast-quad
              irradiat
              Class
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Naive Bayes Classifier

                              Class
Attribute      no-recurrence-events     recurrence-events
                              (0.7)                 (0.3)
=========================================================
```

Status: OK

19:46
16-06-2021



Classifier: NaiveBayes

Test options:
- ● Use training set
- ○ Supplied test set
- ○ Cross-validation Folds 10
- ○ Percentage split % 66

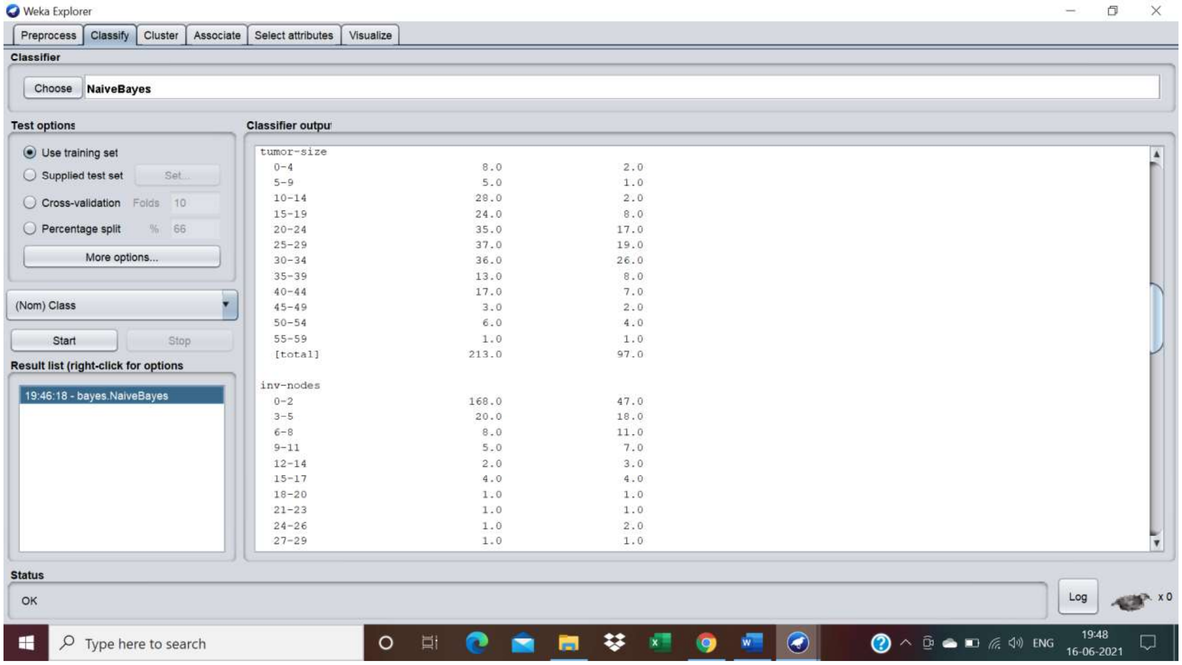
(Nom) Class

Result list: 19:46:18 - bayes.NaiveBayes

```
                              Class
Attribute      no-recurrence-events     recurrence-events
                              (0.7)                 (0.3)
=========================================================
age
  10-19                         1.0                   1.0
  20-29                         2.0                   1.0
  30-39                        22.0                  16.0
  40-49                        64.0                  28.0
  50-59                        72.0                  26.0
  60-69                        41.0                  18.0
  70-79                         6.0                   2.0
  80-89                         1.0                   1.0
  90-99                         1.0                   1.0
  [total]                     210.0                  94.0

menopause
  lt40                          6.0                   3.0
  ge40                         95.0                  36.0
  premeno                     103.0                  49.0
  [total]                     204.0                  88.0

tumor-size
  0-4                           8.0                   2.0
  5-9                           5.0                   1.0
  10-14                        28.0                   2.0
```

Status: OK

19:47
16-06-2021



NaiveBayes

● Use training set
○ Supplied test set
○ Cross-validation Folds 10
○ Percentage split % 66

(Nom) Class

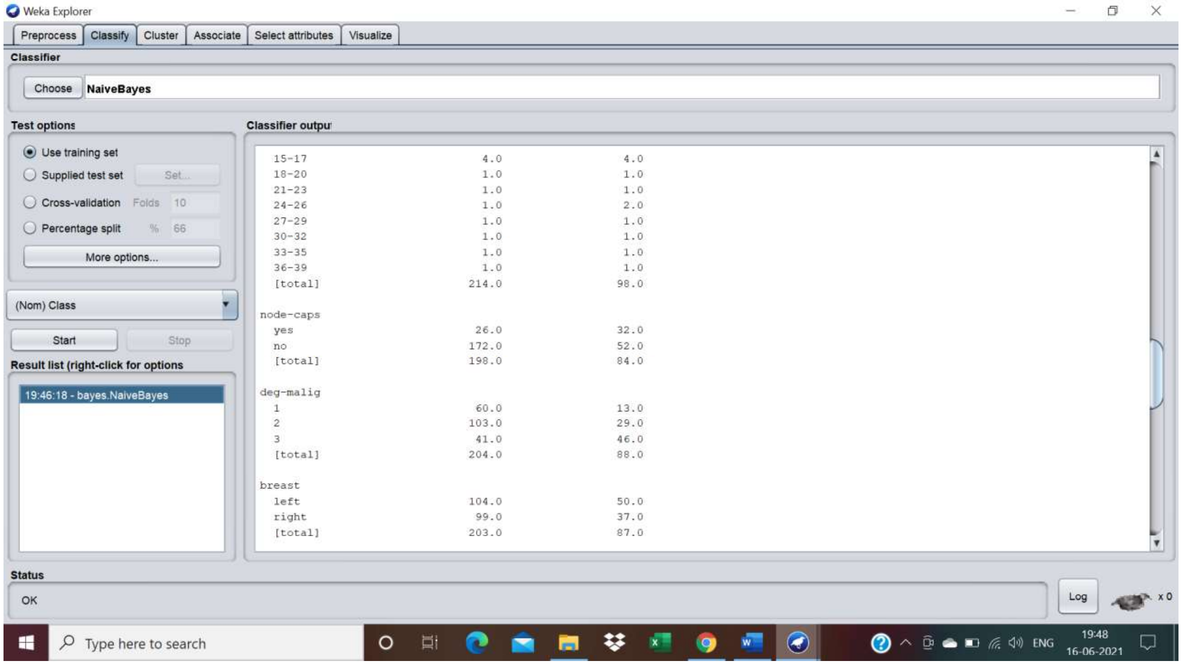
Result list (right-click for options)

19:46:18 - bayes.NaiveBayes

```
tumor-size
  0-4                        8.0          2.0
  5-9                        5.0          1.0
  10-14                     28.0          2.0
  15-19                     24.0          8.0
  20-24                     35.0         17.0
  25-29                     37.0         19.0
  30-34                     36.0         26.0
  35-39                     13.0          8.0
  40-44                     17.0          7.0
  45-49                      3.0          2.0
  50-54                      6.0          4.0
  55-59                      1.0          1.0
  [total]                  213.0         97.0

inv-nodes
  0-2                      168.0         47.0
  3-5                       20.0         18.0
  6-8                        8.0         11.0
  9-11                       5.0          7.0
  12-14                      2.0          3.0
  15-17                      4.0          4.0
  18-20                      1.0          1.0
  21-23                      1.0          1.0
  24-26                      1.0          2.0
  27-29                      1.0          1.0
```

Status

OK



NaiveBayes

● Use training set
○ Supplied test set
○ Cross-validation Folds 10
○ Percentage split % 66

(Nom) Class

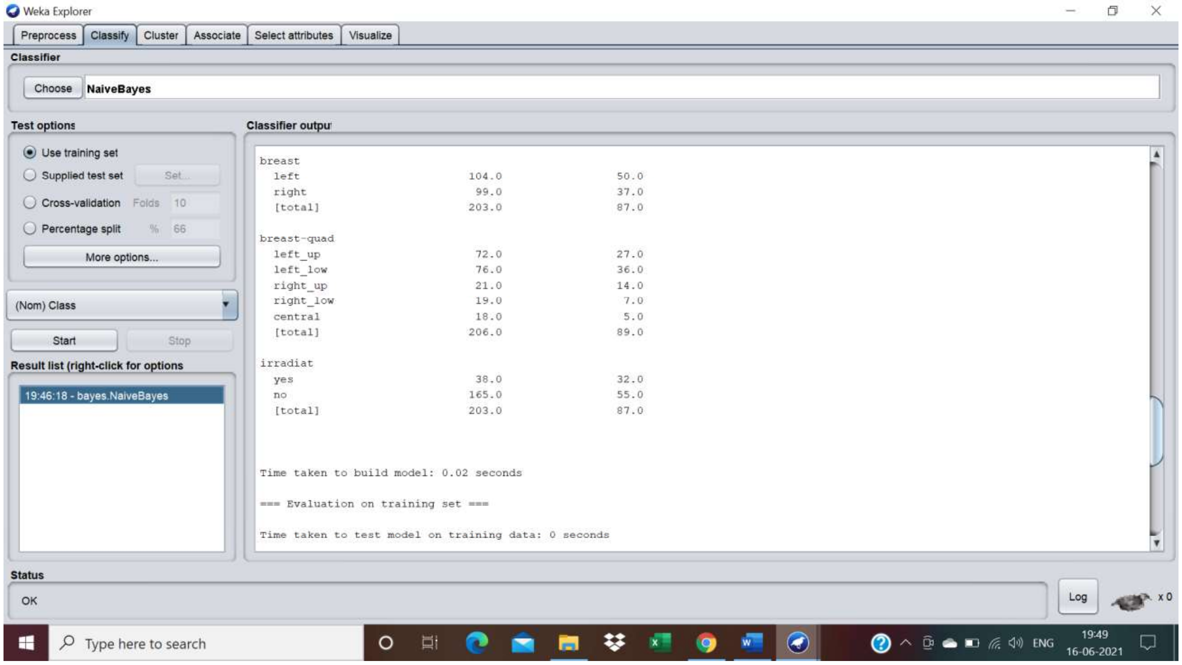
Result list (right-click for options)

19:46:18 - bayes.NaiveBayes

```
  15-17                      4.0          4.0
  18-20                      1.0          1.0
  21-23                      1.0          1.0
  24-26                      1.0          2.0
  27-29                      1.0          1.0
  30-32                      1.0          1.0
  33-35                      1.0          1.0
  36-39                      1.0          1.0
  [total]                  214.0         98.0

node-caps
  yes                       26.0         32.0
  no                       172.0         52.0
  [total]                  198.0         84.0

deg-malig
  1                         60.0         13.0
  2                        103.0         29.0
  3                         41.0         46.0
  [total]                  204.0         88.0

breast
  left                     104.0         50.0
  right                     99.0         37.0
  [total]                  203.0         87.0
```

Status

OK



Classifier: NaiveBayes

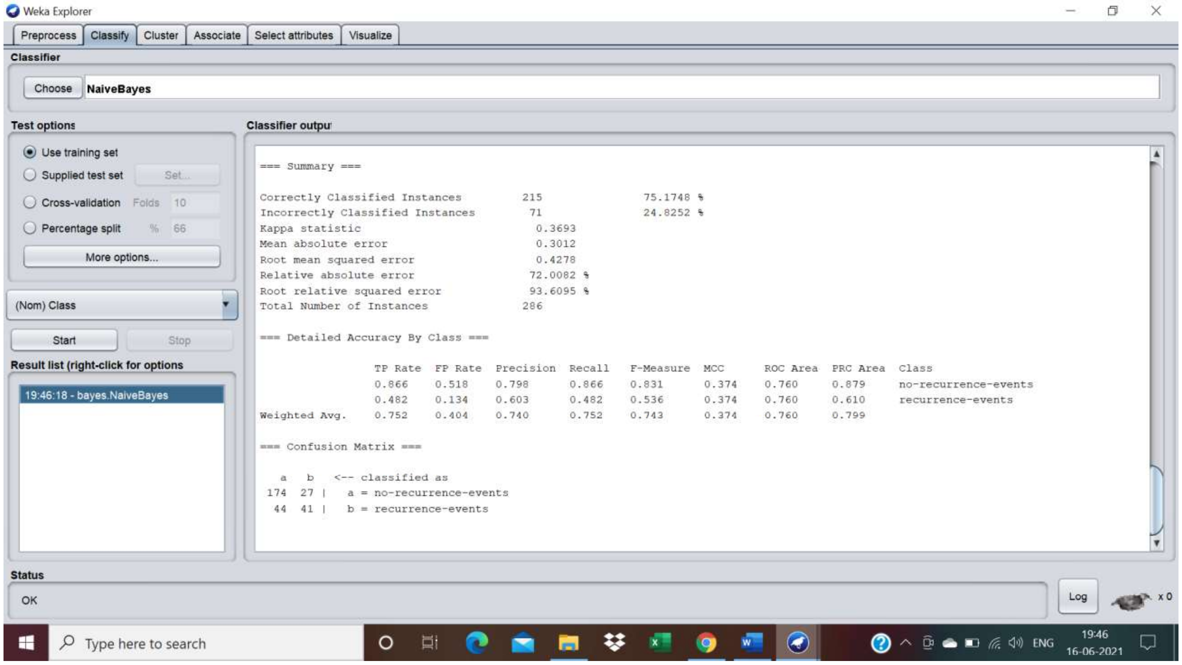
● Use training set

(Nom) Class

19:46:18 - bayes.NaiveBayes

```
breast
  left                      104.0                 50.0
  right                      99.0                 37.0
  [total]                   203.0                 87.0

breast-quad
  left_up                    72.0                 27.0
  left_low                   76.0                 36.0
  right_up                   21.0                 14.0
  right_low                  19.0                  7.0
  central                    18.0                  5.0
  [total]                   206.0                 89.0

irradiat
  yes                        38.0                 32.0
  no                        165.0                 55.0
  [total]                   203.0                 87.0



Time taken to build model: 0.02 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds
```

Status: OK



Classifier: NaiveBayes

● Use training set

(Nom) Class

19:46:18 - bayes.NaiveBayes

```
=== Summary ===

Correctly Classified Instances         215               75.1748 %
Incorrectly Classified Instances        71               24.8252 %
Kappa statistic                          0.3693
Mean absolute error                      0.3012
Root mean squared error                  0.4278
Relative absolute error                 72.0082 %
Root relative squared error             93.6095 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.866    0.518    0.798      0.866    0.831      0.374    0.760     0.879     no-recurrence-events
                 0.482    0.134    0.603      0.482    0.536      0.374    0.760     0.610     recurrence-events
Weighted Avg.    0.752    0.404    0.740      0.752    0.743      0.374    0.760     0.799

=== Confusion Matrix ===

   a   b   <-- classified as
 174  27 |   a = no-recurrence-events
  44  41 |   b = recurrence-events
```

Status: OK

Experiment 2

Aim: Study and Implement the Decision Tree learners using WEKA. (Breast Cancer dataset).

Theory: C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set { S={s_{1},s_{2},...}} S={s_{1},s_{2},...} of already classified samples. Each sample {s_{i}} s_{i} consists of a p-dimensional vector {(x_{1,i},x_{2,i},...,x_{p,i})} (x_{{1,i}},x_{{2,i}},...,x_{{p,i}}), where the {x_{j}} x_{j} represent attribute values or features of the sample, as well as the class in which {s_{i}} s_{i} falls.

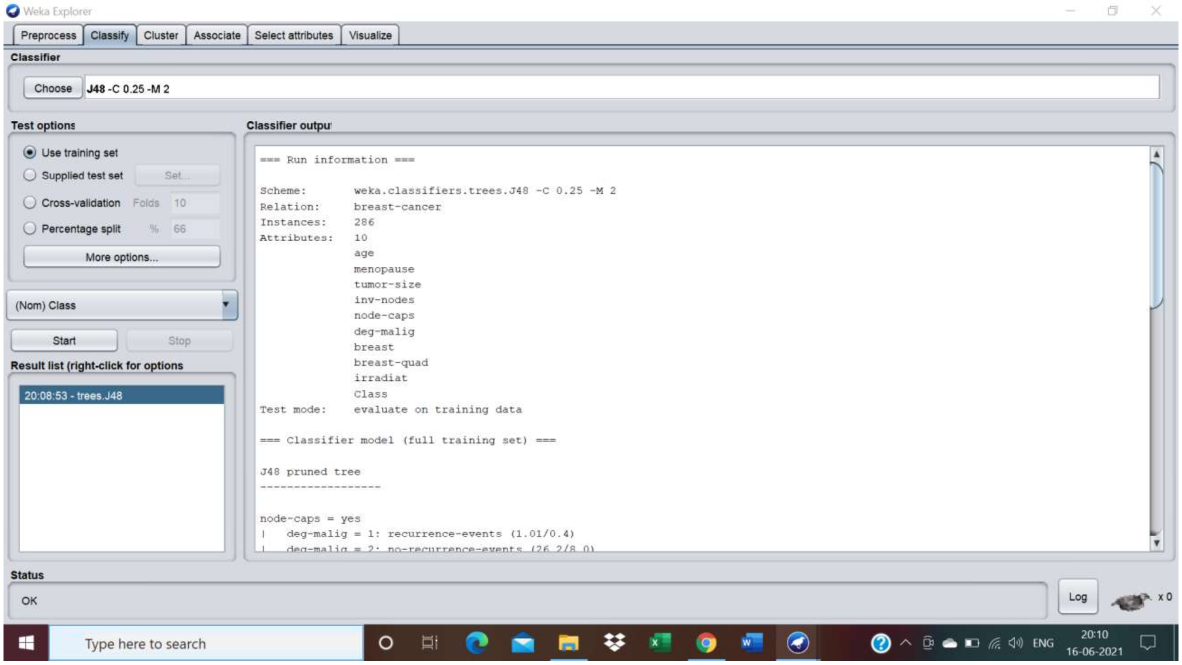
At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

This algorithm has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

J48 is an open-source Java implementation of the C4.5 algorithm in the WEKA.

Output:



J48 -C 0.25 -M 2

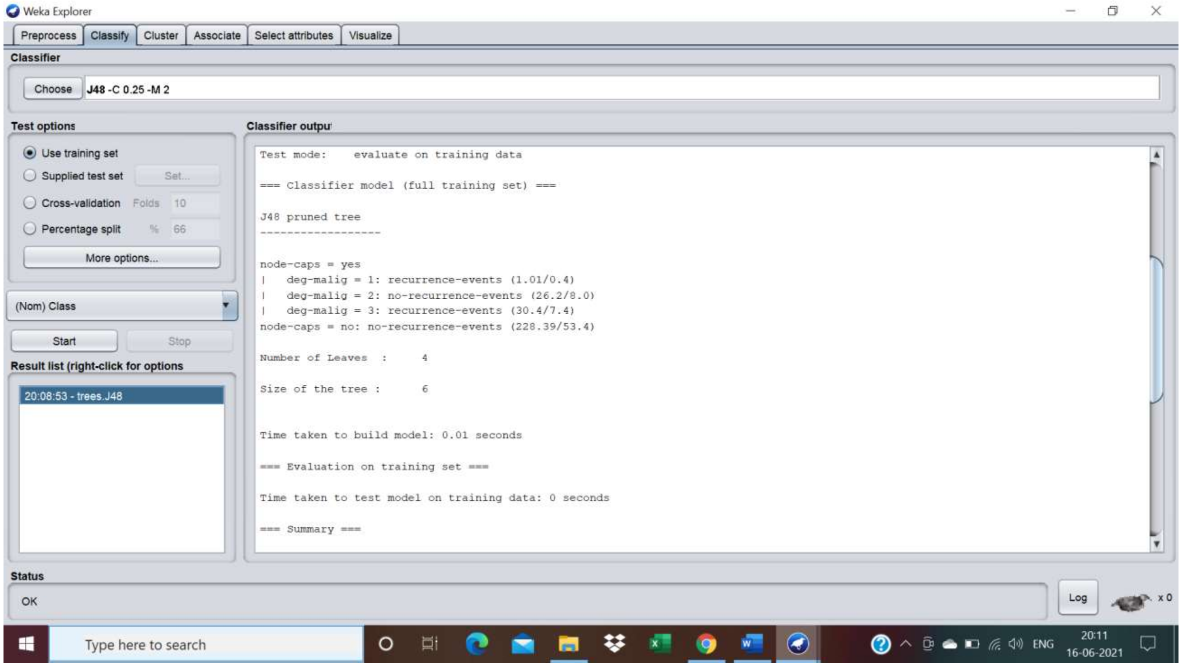
● Use training set
○ Supplied test set
○ Cross-validation Folds 10
○ Percentage split % 66

(Nom) Class

20:08:53 - trees.J48

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     breast-cancer
Instances:    286
Attributes:   10
              age
              menopause
              tumor-size
              inv-nodes
              node-caps
              deg-malig
              breast
              breast-quad
              irradiat
              Class
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
------------------

node-caps = yes
|   deg-malig = 1: recurrence-events (1.01/0.4)
|   deg-malig = 2: no-recurrence-events (26.2/8.0)
```

OK



J48 -C 0.25 -M 2

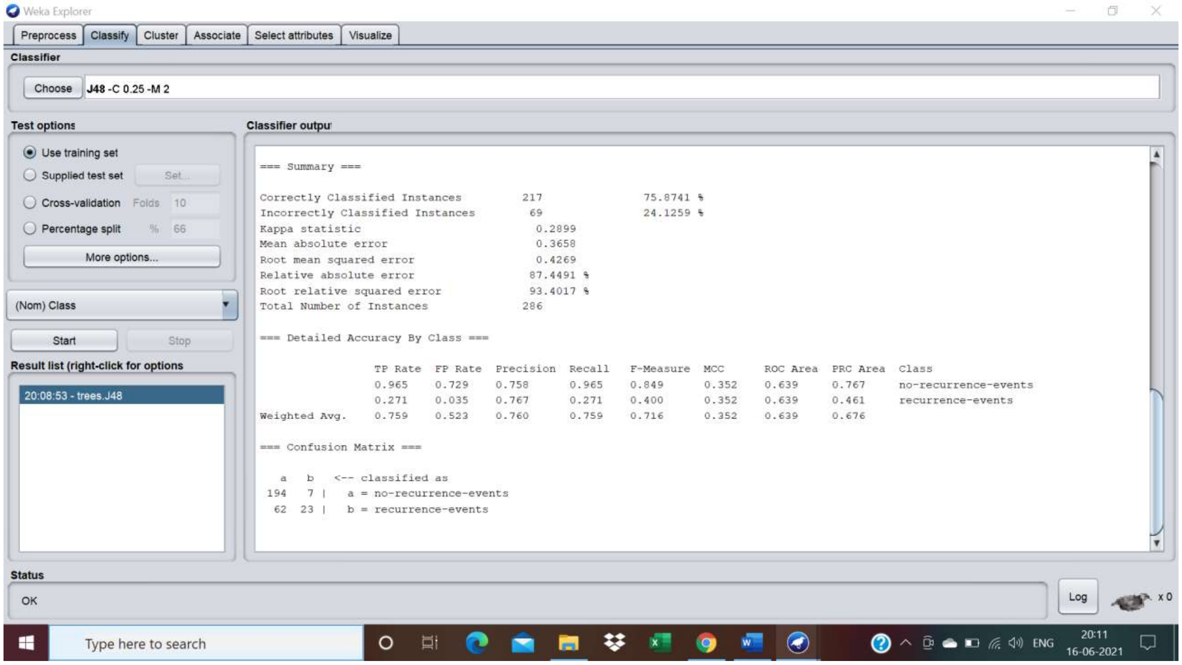
● Use training set
○ Supplied test set
○ Cross-validation Folds 10
○ Percentage split % 66

(Nom) Class

20:08:53 - trees.J48

```
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
------------------

node-caps = yes
|   deg-malig = 1: recurrence-events (1.01/0.4)
|   deg-malig = 2: no-recurrence-events (26.2/8.0)
|   deg-malig = 3: recurrence-events (30.4/7.4)
node-caps = no: no-recurrence-events (228.39/53.4)

Number of Leaves  :     4

Size of the tree :      6


Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===
```

OK



J48 -C 0.25 -M 2

● Use training set
○ Supplied test set
○ Cross-validation Folds 10
○ Percentage split % 66

(Nom) Class

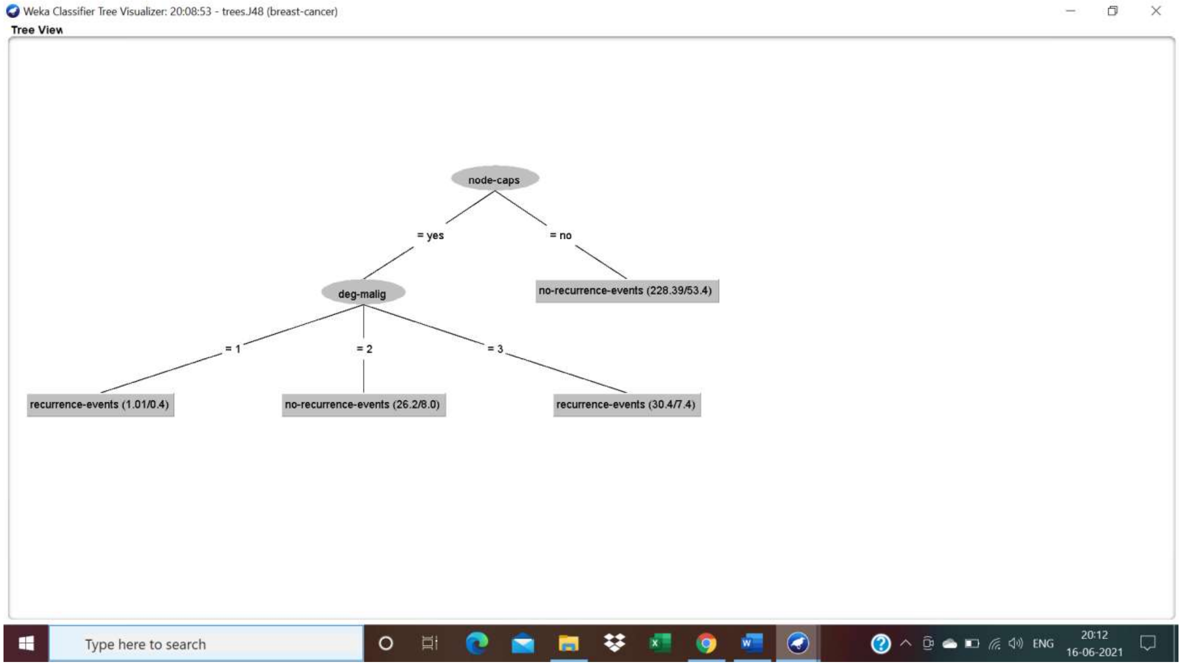
20:08:53 - trees.J48

Classifier output

```
=== Summary ===

Correctly Classified Instances         217               75.8741 %
Incorrectly Classified Instances        69               24.1259 %
Kappa statistic                          0.2899
Mean absolute error                      0.3658
Root mean squared error                  0.4269
Relative absolute error                 87.4491 %
Root relative squared error             93.4017 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.965    0.729    0.758      0.965    0.849      0.352    0.639     0.767     no-recurrence-events
                 0.271    0.035    0.767      0.271    0.400      0.352    0.639     0.461     recurrence-events
Weighted Avg.    0.759    0.523    0.760      0.759    0.716      0.352    0.639     0.676

=== Confusion Matrix ===

   a   b   <-- classified as
 194   7 |   a = no-recurrence-events
  62  23 |   b = recurrence-events
```

Status

OK

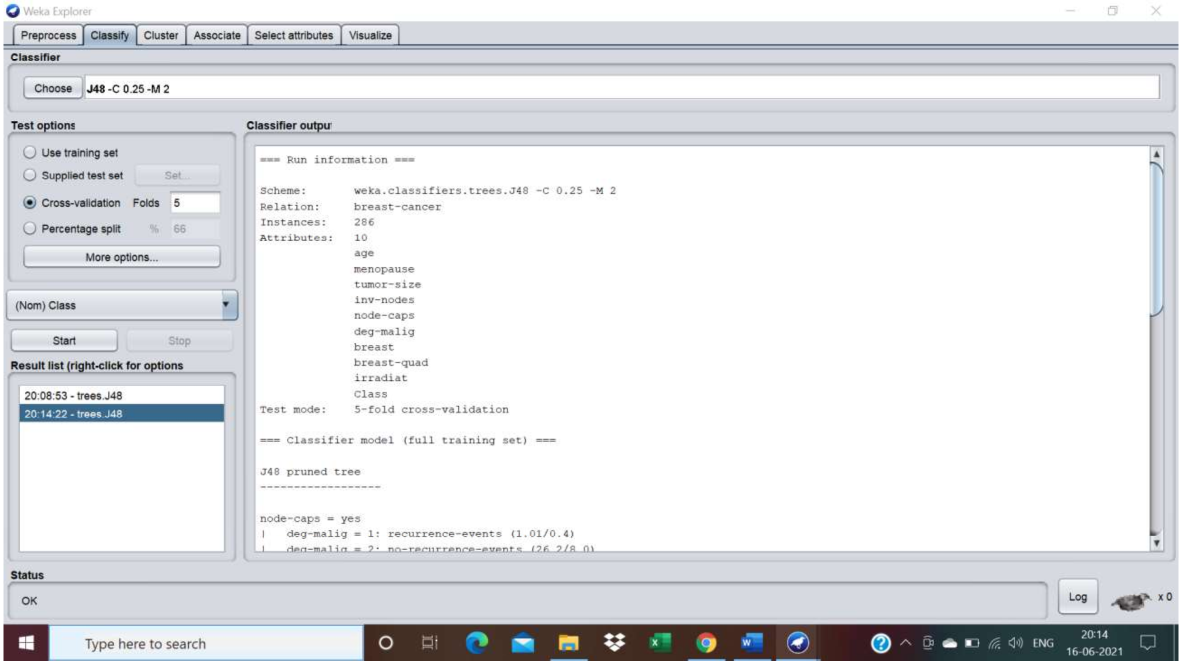

Weka Classifier Tree Visualizer: 20:08:53 - trees.J48 (breast-cancer)
Tree View
node-caps
= yes
= no
deg-malig
no-recurrence-events (228.39/53.4)
= 1
= 2
= 3
recurrence-events (1.01/0.4)
no-recurrence-events (26.2/8.0)
recurrence-events (30.4/7.4)
Type here to search
ENG
20:12
16-06-2021

Experiment 3

Aim: Estimate the accuracy of decision classifier on breast cancer dataset using 5-fold cross-validation. (You need to choose the appropriate options for missing values).

Theory: There are various approaches to determine the performance of classifiers. The performance can most simply be measured by counting the proportion of correctly predicted examples in an unseen test dataset. This value is the accuracy, which is also 1-ErrorRate. Both terms are used in literature. The simplest case is using a training set and a test set which are mutually independent. This is referred to as hold-out estimate. To estimate variance in these performance estimates, hold-out estimates may be computed by repeatedly re-sampling the same dataset – i.e. randomly reordering it and then splitting it into training and test sets with a specific proportion of the examples, collecting all estimates on test data and computing average and standard deviation of accuracy. A more elaborate method is cross-validation. Here, a number of folds n is specified. The dataset is randomly reordered and then split into n folds of equal size. In each iteration, one fold is used for testing and the other n-1 folds are used for training the classifier. The test results are collected and averaged over all folds. This gives the cross-validation estimate of the accuracy. The folds can be purely random or slightly modified to create the same class distributions in each fold as in the complete dataset. In the latter case the cross-validation is called stratified. Leave-one-out (= loo) cross-validation signifies that n is equal to the number of examples. Out of necessity, loo cross-validation has to be non-stratified, i.e. the class distributions in the test set are not related to those in the training data. Therefore loo cross-validation tends to give less reliable results. However it is still quite useful in dealing with small datasets since it utilizes the greatest amount of training data from the dataset.

Output:



Classifier: J48 -C 0.25 -M 2

Test options

○ Use training set
○ Supplied test set
● Cross-validation Folds 5
○ Percentage split % 66

(Nom) Class

Result list (right-click for options)

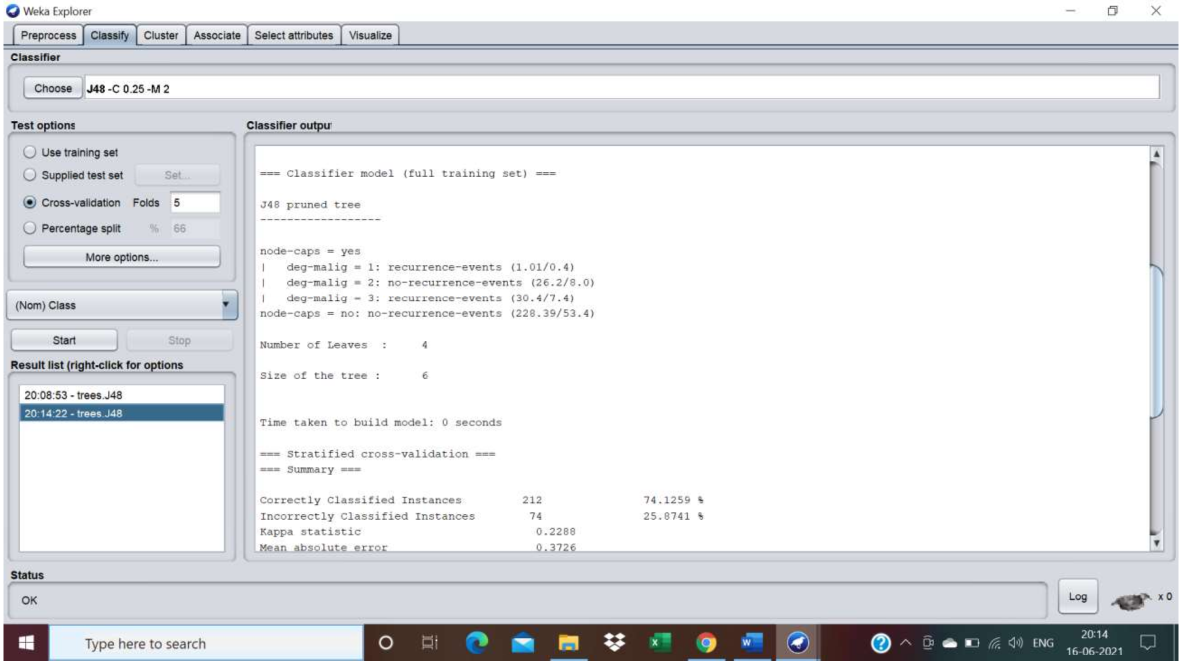
20:08:53 - trees.J48
20:14:22 - trees.J48

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     breast-cancer
Instances:    286
Attributes:   10
              age
              menopause
              tumor-size
              inv-nodes
              node-caps
              deg-malig
              breast
              breast-quad
              irradiat
              Class
Test mode:    5-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

node-caps = yes
|   deg-malig = 1: recurrence-events (1.01/0.4)
|   deg-malig = 2: no-recurrence-events (26.2/8.0)
```

Status: OK

20:14
16-06-2021



Classifier: J48 -C 0.25 -M 2

Test options

○ Use training set
○ Supplied test set
● Cross-validation Folds 5
○ Percentage split % 66

(Nom) Class

Result list (right-click for options)

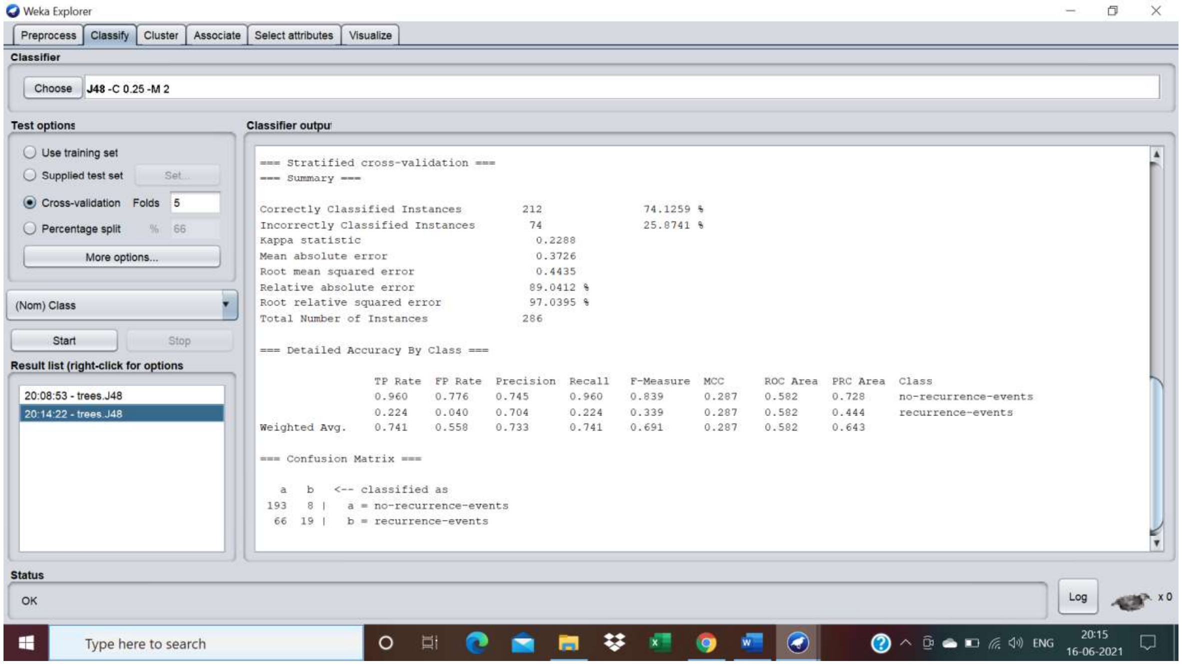
20:08:53 - trees.J48
20:14:22 - trees.J48

Classifier output

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

node-caps = yes
|   deg-malig = 1: recurrence-events (1.01/0.4)
|   deg-malig = 2: no-recurrence-events (26.2/8.0)
|   deg-malig = 3: recurrence-events (30.4/7.4)
node-caps = no: no-recurrence-events (228.39/53.4)

Number of Leaves  :     4

Size of the tree :      6


Time taken to build model: 0 seconds

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

Correctly Classified Instances         212               74.1259 %
Incorrectly Classified Instances        74               25.8741 %
Kappa statistic                          0.2288
Mean absolute error                      0.3726
```

Status: OK

20:14
16-06-2021



Classifier: J48 -C 0.25 -M 2

Cross-validation Folds 5

(Nom) Class

20:08:53 - trees.J48
20:14:22 - trees.J48

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

Correctly Classified Instances         212               74.1259 %
Incorrectly Classified Instances        74               25.8741 %
Kappa statistic                          0.2288
Mean absolute error                      0.3726
Root mean squared error                  0.4435
Relative absolute error                 89.0412 %
Root relative squared error             97.0395 %
Total Number of Instances              286

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.960    0.776    0.745      0.960    0.839      0.287    0.582     0.728     no-recurrence-events
                 0.224    0.040    0.704      0.224    0.339      0.287    0.582     0.444     recurrence-events
Weighted Avg.    0.741    0.558    0.733      0.741    0.691      0.287    0.582     0.643

=== Confusion Matrix ===

   a   b   <-- classified as
 193   8 |   a = no-recurrence-events
  66  19 |   b = recurrence-events
```

Status: OK

Experiment 4

Aim: Estimate the precision, recall, accuracy, and F-measure of the decision tree classifier on the text Classification task for each of the 10 categories using 10-fold cross-validation.

Theory: Text classification is one of the key techniques in text mining to categorize the documents in a supervised manner. The processing of text classification involves two main problems are the extraction of feature terms that become effective keywords in the training phase and then the actual classification of the document using these feature terms in the test phase. This text classification task has numerous applications such as automated indexing of scientific articles according to predefined thesauri of technical terms, routing of customer email in a customer service department, filing patents into patent directories, automated population of hierarchical catalogues of Web resources, selective dissemination of information to consumers, identification of document genre, or detection and identification of criminal activities for military, police, or secret service environments and so on. Text classification can be used for document filtering and routing to topic-specific processing mechanisms such as information extraction and machine translation.

TP = true positives: number of examples predicted positive that are actually positive

FP = false positives: number of examples predicted positive that are actually negative

TN = true negatives: number of examples predicted negative that are actually negative

FN = false negatives: number of examples predicted negative that are actually positive

Recall is referred to as the true positive rate or sensitivity. The True Positive (TP) rate is the proportion of examples which were classified as class x, among all examples which truly have class x, i.e., how much part of the class was captured.

The Precision is the proportion of the examples which truly have class x among all those which were classified as class x.

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

The F-Measure is simply 2*Precision*Recall/(Precision+Recall), a combined measure for precision and recall.

These measures are useful for comparing classifiers.

Data file:

```
@relation  textclass1

@attribute text1 {ball,goal,medals,party,poll,ministers}

@attribute text2 {wicket,ball,poll,election,performance,party}

@attribute news {politics, sports}

@data

ball, wicket, sports

goal, ball, sports

party, poll, politics

poll, election, politics

ministers, election, politics

medals, performance, sports

ball, party, sports

goal, wicket, sports

ministers, party, politics

party, election, politics

goal, election, politics

poll,performance, politics

ball,performance, sports
```

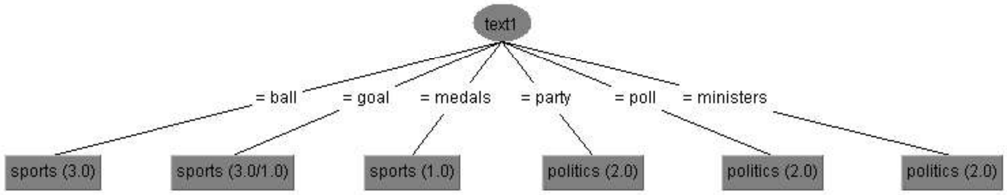
Output:

```
=== Run information ===


Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2

Relation:     textclass1

Instances:    13

Attributes:   3
```

```
              text1
              text2
              news
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

text1 = ball: sports (3.0)
text1 = goal: sports (3.0/1.0)
text1 = medals: sports (1.0)
text1 = party: politics (2.0)
text1 = poll: politics (2.0)
text1 = ministers: politics (2.0)

Number of Leaves  :     6

Size of the tree :      7


Time taken to build model: 0 seconds

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

Correctly Classified Instances           4               30.7692 %
Incorrectly Classified Instances         9               69.2308 %
```

```
Kappa statistic                    -0.4444

Mean absolute error                   0.5192

Root mean squared error                0.6517

Relative absolute error             100.5319 %

Root relative squared error           125.8013 %

Total Number of Instances              13


=== Detailed Accuracy By Class ===

  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class

    0.571    1.000     0.400     0.571     0.471       -0.507     0.524          0.688       politics

     0.000    0.429     0.000     0.000     0.000       -0.507     0.524           0.523       sports


Weighted Avg.    0.308    0.736    0.215     0.308    0.253     -0.507   0.524     0.612


=== Confusion Matrix ===


 a b   <-- classified as

 4 3 | a = politics

 6 0 | b = sports
```

Visualize Tree



Experiment 5

Aim: Develop a machine learning method to classifying your incoming mail.

Theory: Email filtering is an extremely useful and practical problem. Email has become a mainstream form of communication. We obtain useful information each day via email but there is problem of getting unwanted emails.

Many users receive numerous unwanted emails each day which is why a spam filter has been created. The spam filter needs to sort incoming mail into wanted and unwanted. This can be tricky because the filter could allow too much spam into the inbox or could label some legitimate emails as spam. Machine learning can help solve this problem. The email client can be trained to learn where to put each email.

A machine learning algorithm for email filtering will take in a set of labeled messages as the input and will output correct labels for the testing data.

Pre-processing: Not all of the information in an email is necessary or even useful information. There is a lot of unrelated information that serves no purpose in classifying the email. This information can be eliminated from the email and not processed by the machine learning algorithm. The email header can be discarded. The user rarely sees or pays attention to this information anyhow so it can be ruled out.

Before attempting to eliminate things like stopwords or meaningless words, certain sets of data need to be preserved. Email addresses are useful and shouldn't be split apart. The data needs to be tokenized.

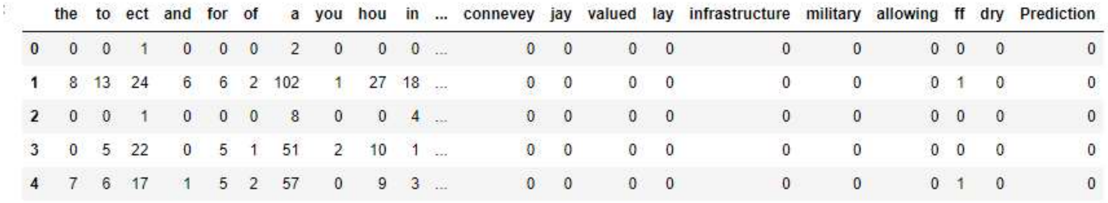
Words that appear with a high frequency can be ignored since they will appear in almost all samples. Common words, most punctuation, pronouns, simple verbs and adverbs can be thrown out since they will appear in almost all of the data so they can be ignored. The word "the" will appear multiple times in each sample; however, it adds no value. The words thrown out make up a stoplist which there are many predefined lists of stopwords that can be used.

After stop words are discarded, the remaining words are stemmed. Stemming reduces words to their root word. This allows for better comparisons between words. If the words "running" and "runner" both appeared, they would be treated as two different features. By stemming, they are both reduced to "run" which allows them to be grouped together. Stemming helps the program to correctly classify an email. The algorithm must depend on multiple words to create a classification for the email.

Techniques: There are many machine learning techniques available to filter emails. Many spam filtering techniques use text categorization methods to label the data. Using machine learning can easily be applied to this problem.

This section will give a brief overview of each algorithm how it can be used for this machine learning problem. The algorithm that will be discussed is Naive Bayes.

Data used:



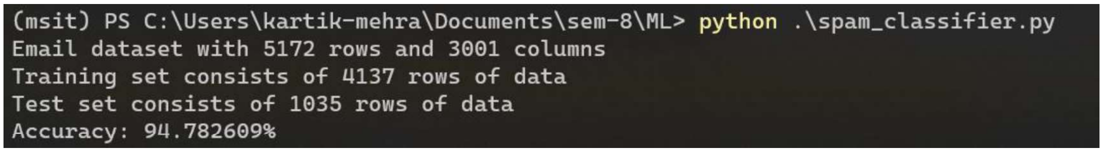
	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	0	0	1	0	0	0	2	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	0	0	0	0	1	0	0
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	0	0	0	0	0	0	0
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	0	0	0	0	0	0	0
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	0	0	0	0	1	0	0

Source code:

```
import os


import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score



data_dir = 'data'
filename = 'emails.csv'
filepath = os.path.join(data_dir, filename)


emails = pd.read_csv(filepath)
emails.drop(columns='Email No.', inplace=True)
print('Email dataset with {} rows and {} columns'.format(*emails.shape))
# emails.head()
```

```
X = emails.iloc[:, :-1]
y = emails.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)
print(f'Training set consists of {X_train.shape[0]} rows of data')
print(f'Test set consists of {X_test.shape[0]} rows of data')

## Naive Bayes
model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f'Accuracy: {acc:%}')
```

Output:



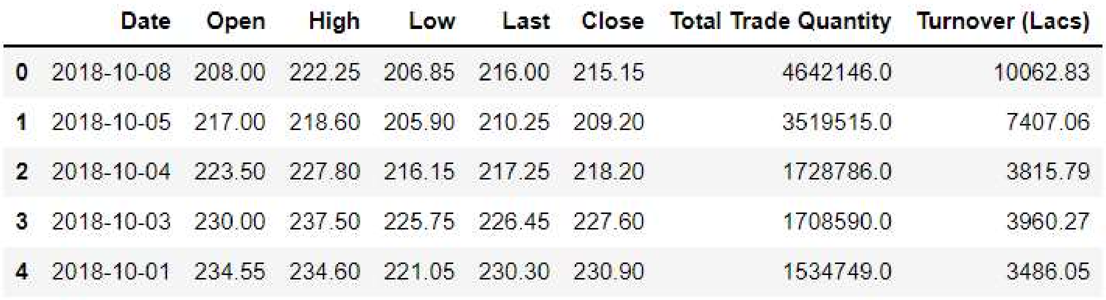
```
(msit) PS C:\Users\kartik-mehra\Documents\sem-8\ML> python .\spam_classifier.py
Email dataset with 5172 rows and 3001 columns
Training set consists of 4137 rows of data
Test set consists of 1035 rows of data
Accuracy: 94.782609%
```

Experiment 6

Aim: Develop a machine learning method to Predict stock prices based on past price variation.

Theory: Stock market prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The successful prediction of a stock's future price could yield significant profit. The efficient-market hypothesis suggests that stock prices reflect all currently available information and any price changes that are not based on newly revealed information thus are inherently unpredictable. Others disagree and those with this viewpoint possess myriad methods and technologies which purportedly allow them to gain future price information.

Data used:



	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146.0	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515.0	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786.0	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590.0	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749.0	3486.05

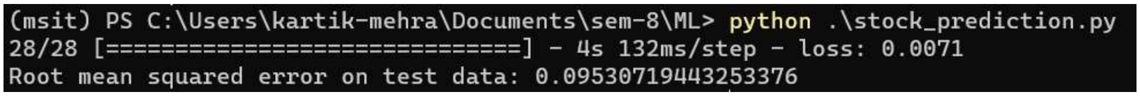
Source code:

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

import tensorflow as tf
```

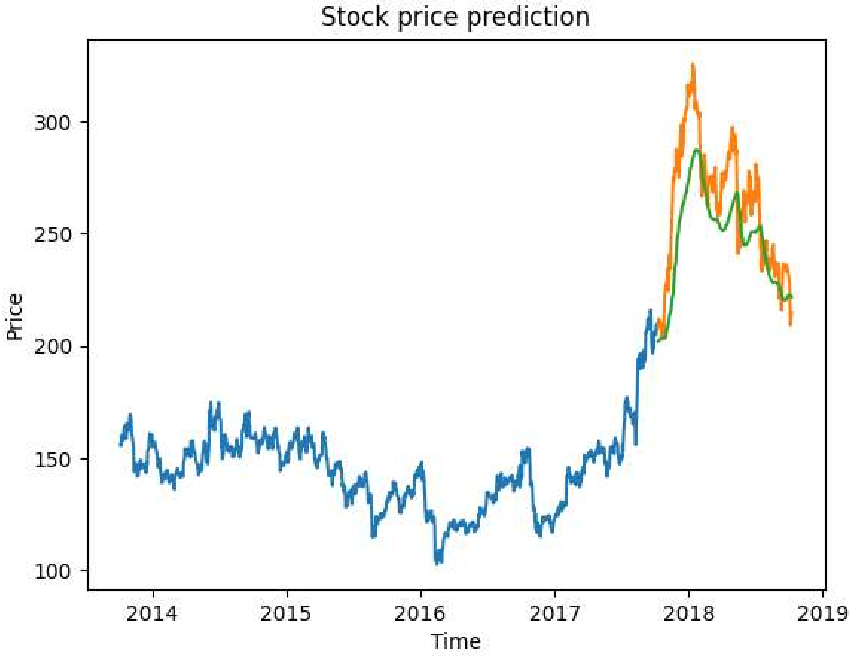
```
data_dir = 'data'

df = pd.read_csv(os.path.join(data_dir, 'NSE-TATAGLOBAL11.csv'), parse_dates=['Date'], index_col='Date', usecols=['Date', 'Close'])
df.sort_index(inplace=True)

days = 100
train_size = int(0.8 * df.shape[0])

scaler = MinMaxScaler()
dataset = scaler.fit_transform(df.values)

train = dataset[:train_size]
test = dataset[train_size - days:]

X_train, y_train = [], []
for i in range(days, train.shape[0]):
    X_train.append(train[i - days : i])
    y_train.append(train[i, 0])
X_train = np.array(X_train)
y_train = np.array(y_train)

X_test, y_test = [], []
for i in range(days, test.shape[0]):
    X_test.append(test[i - days : i])
    y_test.append(test[i, 0])
X_test = np.array(X_test)
y_test = np.array(y_test)
```

```
model = tf.keras.models.Sequential([
   tf.keras.layers.LSTM(64, return_sequences=True, input_shape=(days, 1)),
   tf.keras.layers.LSTM(64),
   tf.keras.layers.Dense(1)
])

model.compile(
   optimizer='adam',
   loss='mse'
)

model.fit(X_train, y_train, epochs=1)

y_pred = model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'Root mean squared error on test data: {rmse}')

# plt.figure(figsize=(16, 8))
plt.plot(pd.DataFrame(data=scaler.inverse_transform(train), index=df.index[:train_size]))
plt.plot(pd.DataFrame(data=scaler.inverse_transform(test[days:]),
index=df.index[train_size:]))
plt.plot(pd.DataFrame(data=scaler.inverse_transform(y_pred), index=df.index[train_size:]))

plt.xlabel('Time')
plt.ylabel('Price')
plt.title('Stock price prediction')

plt.show()
```

Output:



```
(msit) PS C:\Users\kartik-mehra\Documents\sem-8\ML> python .\stock_prediction.py
28/28 [==============================] - 4s 132ms/step - loss: 0.0071
Root mean squared error on test data: 0.09530719443253376
```



Experiment 7

Aim: Develop a machine learning method to Cluster gene expression data, how to modify existing methods to solve the problem better.

Theory: k-nearest neighbors (kNN) are a classical method for recommender systems. Ratings or items are predicted by using the past ratings/items of the k most similar users and/or items. If the influence of the similar users/items is weighted by the similarity, all users/items may be used for the prediction. Popular similarity metrics are the Pearson correlation and the cosine similarity. Using the user-item matrix to compute the similarity is often called collaborative filtering. Computing the item similarities from the item attributes leads to content-based filtering.

The k nearest neighbors (k-NN) algorithm uses an idea distance between each instance. The distance between two messages can be measured and it can be determined how close they are to each other. The algorithm attempts to classify the message be looking at the nearest neighbors.

The distance is usually measured by using the Euclidian distance. The formula for the Euclidian distance is given by

$$\mathrm{d}(\mathbf{x},\mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

The goal is to measure the distance of k closest neighbors and take the majority vote of those neighbors to classify the current message.

Training: To create the training data simply store the feature vectors and the label of each message.

Classification: Take a message m, find the k nearest neighbors and count the number of each label (spam or not) that are given from the neighbors. If there are more spam messages in the k nearest neighbors then it is classified as spam. If not, then that message is classified as legitimate mail.

One advantage of this algorithm is that there isn't really a training phase. However, to classify a message, all distances between that message and all the training examples must be calculated and the k nearest neighbors need to be found and counted.

Dataset: MovieLens

Source code:

```
import csv

import random

import math
import operator



def loadDataset(filename, split, trainingSet=[] , testSet=[]):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
            neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] is predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

# prepare data
trainingSet=[]
testSet=[]
split = 0.67
loadDataset('iris.data', split, trainingSet, testSet)
print 'Train set: ' + repr(len(trainingSet))
print 'Test set: ' + repr(len(testSet))
```

```
# generate predictions
predictions=[]
k = 3
for x in range(len(testSet)):
neighbors = getNeighbors(trainingSet, testSet[x], k)
result = getResponse(neighbors)
predictions.append(result)
print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
accuracy = getAccuracy(testSet, predictions)
print('Accuracy: ' + repr(accuracy) + '%')
```

Experiment 8

Aim: Select two datasets. Each dataset should contain examples from multiple classes. For training purposes assume that the class label of each example is unknown (if it is known, ignore it). Implement the K-means algorithm and apply it to the data you selected. Evaluate performance by measuring the sum of Euclidean distance of each example from its class center. Test the performance of the algorithm as a function of the parameter k.

Theory: Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application-specific.

Clustering analysis can be done on the basis of features where we try to find subgroups of samples based on features or on the basis of samples where we try to find subgroups of features based on samples. We'll cover here clustering based on features. Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.

k-means algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way k-means algorithm works is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.

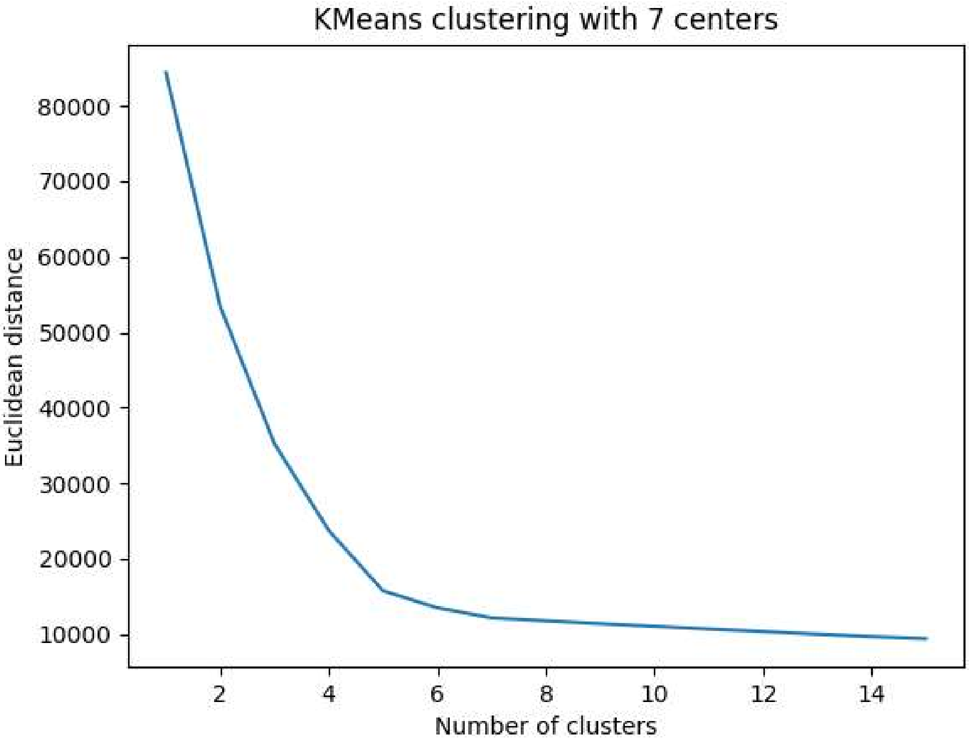
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.
    - Compute the sum of the squared distance between data points and all centroids.
    - Assign each data point to the closest cluster (centroid).
    - Compute the centroids for the clusters by taking the average of the all the data points that belong to each cluster.

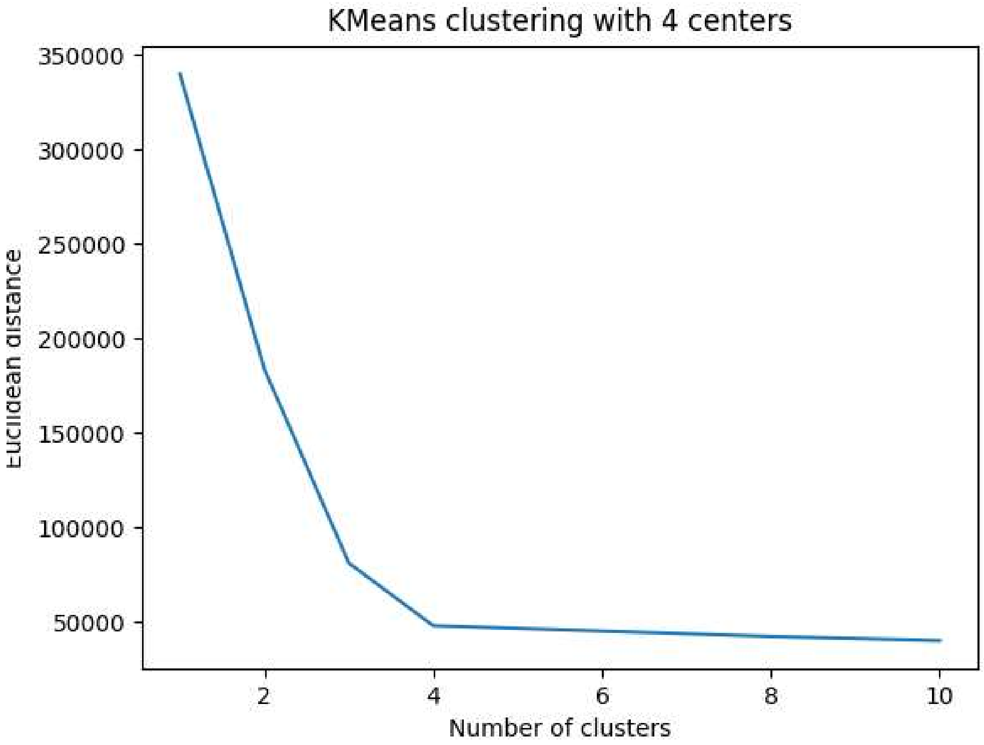
Source code:

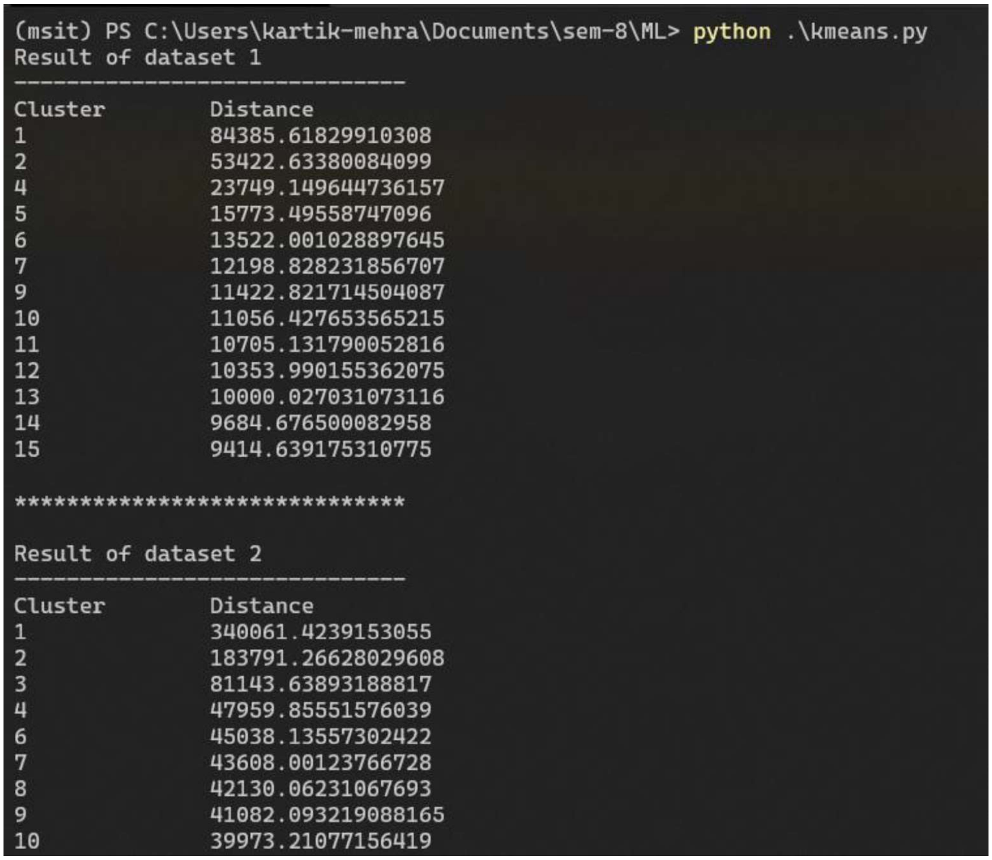
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs



def measureKMeansPerformance(X, n_samples, n_features, centers, max_clusters=10, random_state=42):

    sum_squared_distances = []

    euclidean_distances = []

    clusters = []


    for n_clusters in range(1, max_clusters + 1):

        model = KMeans(n_clusters=n_clusters, random_state=random_state)

        model.fit(X)

        assigned_centers = model.cluster_centers_[model.predict(X)]

        euclidean_distance = np.sqrt(np.square(assigned_centers - X).sum(axis=1)).sum()

        euclidean_distances.append(euclidean_distance)

        sum_squared_distances.append(model.inertia_)

        clusters.append(n_clusters)


    plt.plot(clusters, euclidean_distances)

    plt.xlabel('Number of clusters')
    plt.ylabel('Euclidean distance')
    plt.title(f'KMeans clustering with {centers} centers')
    plt.show()

    return {cluster: euclidean_distance for cluster, euclidean_distance in zip(clusters, euclidean_distances)}



row_format = "{:<15}" * 2

# Dataset 1
n_samples = 10000
n_features = 2
centers = 7

X1, _ = make_blobs(n_samples=n_samples, n_features=n_features, centers=centers, random_state=42)
result1 = measureKMeansPerformance(X1, n_samples, n_features, centers, max_clusters=15)

print('Result of dataset 1')
print('-'*30)
print(row_format.format('Cluster', 'Distance'))
for item in result1.items():
    print(row_format.format(*item))

# Dataset 2
n_samples = 30000
n_features = 3
centers = 4
```

```
X2, _ = make_blobs(n_samples=n_samples, n_features=n_features, centers=centers,
random_state=42)

result2 = measureKMeansPerformance(X2, n_samples, n_features, centers)


print()

print('*' * 30, end='\n\n')

print('Result of dataset 2')

print('-'*30)

print(row_format.format('Cluster', 'Distance'))

for item in result2.items():

   print(row_format.format(*item))
```

Output:







```
(msit) PS C:\Users\kartik-mehra\Documents\sem-8\ML> python .\kmeans.py
Result of dataset 1
------------------------------
Cluster         Distance
1               84385.61829910308
2               53422.63380084099
4               23749.149644736157
5               15773.49558747096
6               13522.001028897645
7               12198.828231856707
9               11422.821714504087
10              11056.427653565215
11              10705.131790052816
12              10353.990155362075
13              10000.027031073116
14              9684.676500082958
15              9414.639175310775

******************************

Result of dataset 2
------------------------------
Cluster         Distance
1               340061.4239153055
2               183791.26628029608
3               81143.63893188817
4               47959.85551576039
6               45038.13557302422
7               43608.00123766728
8               42130.06231067693
9               41082.093219088165
10              39973.21077156419
```