Laboratory Manual

(Part-1)

Data Mining

Department of Computer Science &
Information Systems

Poonam Goyal

Arshveer Kaur

Birla Institute of Technology & Science

Pilani (Rajasthan)

Contents

Getting Started with Python

To work with python, You need to use anaconda tool. Refer the given link to download anaconda on windows and linux:
https://docs.anaconda.com/anaconda/install/linux/

Using Anaconda:

1. Install anaconda on your system.
2. Open the anaconda navigator and launch jupyter-notebook.
3. Jupter-Notebook will open in your local browser.
4. Create a new jupyter-notebook file using python-3.

Pre-processing (Lab-1)

Pre-processing:

Data preprocessing refers to processing the raw data and making it suitable for performing tasks like classification, clustering, etc. Students are required to use numpy and pandas for implementing pre-processing steps. The reference material for basics of python is given in the learning material (which will be uploaded separately).

Numpy:

NumPy (or Numpy) is a Linear Algebra Library for Python It is important for Data Science with Python because all of the libraries in the PyData Ecosystem rely on NumPy.Once you've installed NumPy you can import it as a library

```
import numpy as np
```

Numpy has many built-in functions and capabilities. We will focus on some of the most important aspects of Numpy: vectors,arrays,matrices, and number generation.

Numpy Arrays:

Numpy arrays are represented in two forms: vectors and matrices. Vectors are strictly 1-d arrays and matrices are 2-d array (a matrix can still have only one row or one column).

Creating NumPy Arrays:

We can create an array by directly converting a list or list of lists as follows

```
my_list = [1,2,3]
my_list
np.array(my_list)
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
np.array(my_matrix)
```

Built-in Methods:

1. **arrange:** Returns evenly spaced values within a given interval:

```
np.arange(0,11,2)
#first argument = starting point
# second argument = ending point excluding it
# third argument = difference
```

2. **zeros and ones:** Generates arrays of zeros or ones. Run the given commands and analyze the difference in output of each command:

```
np.zeros(3)
np.zeros((5,5))
np.ones(3)
np.ones((3,3))
```

3. **Random:** Random number arrays can be generated in many ways
    a. **Rand:** Creates an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).
    b. **Randn**: Returns a sample (or samples) from the "standard normal distribution".
    c. **Randint**: Returns random integers from lower value (inclusive) to higher value(exclusive)i.e. [a,b) where a<b.

Run the following commands one by one to know the difference between the three functions:

```
np.random.rand(2)
np.random.rand(5,5)
np.random.randn(2)
np.random.randn(5,5)
np.random.randint(1,100)
```

4. **reshape:** Returns an array containing the same data with a new shape:

```
arr=np.arange(0,25)
arr.reshape(5,5)
```

5. **max and min:** These methods find max or min values, respectively.

6. **argmax and argmin:**find index locations of max and min value in the array, respectively.

```
arr = np.array([10, 12, 41, 17, 49,  2, 46,  3, 19, 39])
print(arr.max())
print(arr.argmax())
print(arr.min())
print(arr.argmin())
```

7. **Shape:**It is an attribute that describesthe shape of an array.

```
arr.shape
arr.reshape(2,5)
arr.reshape(2,5).shape
```

8. **Indexing and Selection:** Create a sample array and select some of the elements as given below and observe the difference:

```
arr = np.arange(0,11)
print(arr)
arr[8]  #Get a value at an index
arr[1:5]        #Get values in a range
arr[0:5]        #Get values in a range
```

9. **Slicing:**Slicing allows you to select a slice of 2-d array:

```
arr = np.arange(0,11)
print(arr)
slice_of_arr = arr[0:6]
slice_of_arr            #Show slice
slice_of_arr[:]=99
slice_of_arr            #Change Slice
```

10. **Copy:** Copying allows you to make multiple changes in an array keeping the original array safe:

```
arr_copy = arr.copy()
arr_copy
```

11. **Indexing a 2D array (matrices):** The general format is **arr_2d[row][col]** or **arr_2d[row,col]:**

```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr_2d
arr_2d[1]               #Indexing row
arr_2d[1,0]             # Getting individual element value
arr_2d[:2,1:]           # 2D array slicing
arr_2d[2,:]             #Shape bottom row
```

Working with Pandas:

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

```
import pandas as pd
```

Methods in pandas

1. **Creating DataFrames:** When we create a DataFrame from a dictionary keys become column names, values become data

```
pd.DataFrame({'id':[100, 101, 102], 'color':['red', 'blue', 'red']})
# create a DataFrame from a list of lists (each inner list becomes a row)
pd.DataFrame([[100, 'red'], [101, 'blue'], [102, 'red']], columns=['id', 'color'])
```

2. **Creating dataframe from array**

```
import numpy as np
arr = np.random.rand(4, 2)
arr
pd.DataFrame(arr, columns=['one', 'two'])
```

3. **Read a dataset from pandas:** You can use the followingURL for the dataset, chiporders,***http://bit.ly/chiporders***. The dataset is *tab* delimited.

```
orders = pd.read_csv('http://bit.ly/chiporders', sep='\t')
orders.head()
```

4. **Methods and attributes:** The given commands can be used to perform the listed operations:

```
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()                # displays the first 5 rows
movies.describe()            # calculate summary statistics
movies.shape                 # prints number of rows and columns
movies.dtypes                # gives data type of each column
```

5. **Renaming attributes**: The following commands are used to rename the attributes of a dataset:

```
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.columns                  # displays the columns
ufo.rename(columns={'Colors Reported':'Colors_Reported', 'Shape
Reported':'Shape_Reported'}, inplace=True)
ufo.columns
```

6. **Removing an attribute:**

```
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head()
ufo.drop('Colors Reported', axis=1, inplace=True)
ufo.head()
```

7. **Missing values:** `isnull`function is used to find null or missing values in the dataset. The function returns true if missing values are there. The function is written as: `dataframe_name.isnull().tail()`

```
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.isnull().tail()
ufo.isnull().sum()      # count the number of missing values in each Series
```

8. **Selecting rows and columns:** Run the following commands and observe how set of rows and columns are selected from a data frame:

```
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head(3)                    # first three rows
ufo.loc[0, :]                  # row 0, all columns
ufo.loc[[0, 1, 2], :]          # rows 0 and 1 and 2, all columns
ufo.loc[0:2, :]                # rows 0 through 2 (inclusive), all columns
ufo.iloc[[0, 1], [0, 3]]       # rows in at 0 and 1, columns at 0 and 3
ufo.iloc[0:2, 0:4]             # rows at positions 0 to 2, columns 0 to 4
                               (upper limit exclusive)
```

Source:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

Pre-processing steps

- Handling missing values:
  Most of the datasets contain missing values. Missing values can be handled in following ways:
  - Discard the rows having missing values
  - Replacing the missing values with mean/median/mode of the attribute.
  - Replacing the values by prediction.
- Removing Duplicate values
- Sampling: Sampling means selecting a subset from the large dataset. It can be done in many ways:
  - Sampling without replacement: The selected record is removed from the original dataset and cannot be selected again.
  - Sampling with replacement: The record selected is not removed from the original dataset and thus can be selected more than once.
  - Stratified Sampling: Stratified sampling makes sure that the same distribution of records with respect to an attribute values is maintained in the sampled dataset as that of original dataset. Stratified sampling is usually done on the basis of classes.

- Discretization: Discretization means converting the numeric data into certain set of bins. Bins can be created in two ways:
  - Equal-width: Divide the data into N intervals of equal size
  - Equal-depth: Divide the data into N intervals, each containing same number of samples.
  - Clustering Based: Divide data int N bins, each bin is a cluster obtained by a clustering algorithm e.g. k-means.
- Normalization: Normalization or standardization is done to bring each attribute in the same range to avoid any kind of biases e.g. consider a dataset having height, weight and width. Each attribute can have different range of values such as height in meters, weights in pounds and width in centimetres. This leads to the biases according to the numbers in each attributes. So to avoid this we convert all the columns in same range say [0-1]. Normalization can be done in two ways:
  - Min-max normalization: It is a transformation of the original data. It scales the data from 0 to 1. It is calculated by the following formula:

$$x_{nor.} = \frac{x - min}{max - min}$$

  - Z-score normalization: It is also called zero-mean normalization. The essence of this technique is the data transformation by the values conversation to a common scale where an average number equals zero and a standard deviation is one.

$$x_{nor.} = \frac{x - mean}{std\ dev.}$$

Questions

1. Read and show the properties of dataset "iris_data_with_missing_values.csv"
2. Print first 20 rows of the dataset
3. Find the number of zeroes in all the columns
4. Find the number of null values in the dataset
5. Replace the zeroes with Nan values in selected attribute (say attribute 1 and 3)
6. Find the number of rows having missing values
7. Discard the records with missing values
8. Find the number of records after discarding null values
9. Replace missing value with the mean/ median/mode as applicable.
10. Create a data frame and label the columns.
11. Sample the dataset with and without replacement (sample size: 20%)
12. Normalize the dataset using z-score normalization
13. Find the outliers using z-score normalization
14. Normalize the dataset using min-max normalization
15. Write the code to discretize an attribute into 6 bins (petal length)
16. Find the correlation coefficient between the attributes

Decision Trees (Lab-2)

Decision Tree Classifier:

Decision Tree (DT) is one of the most commonly used classifier. Classification basically means classifying a new data object with the help of the model built using existing labelled data. A Decision tree is a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) represents a class label.

Creating a Decision Tree:

The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. In general decision tree classifier has moderate accuracy.

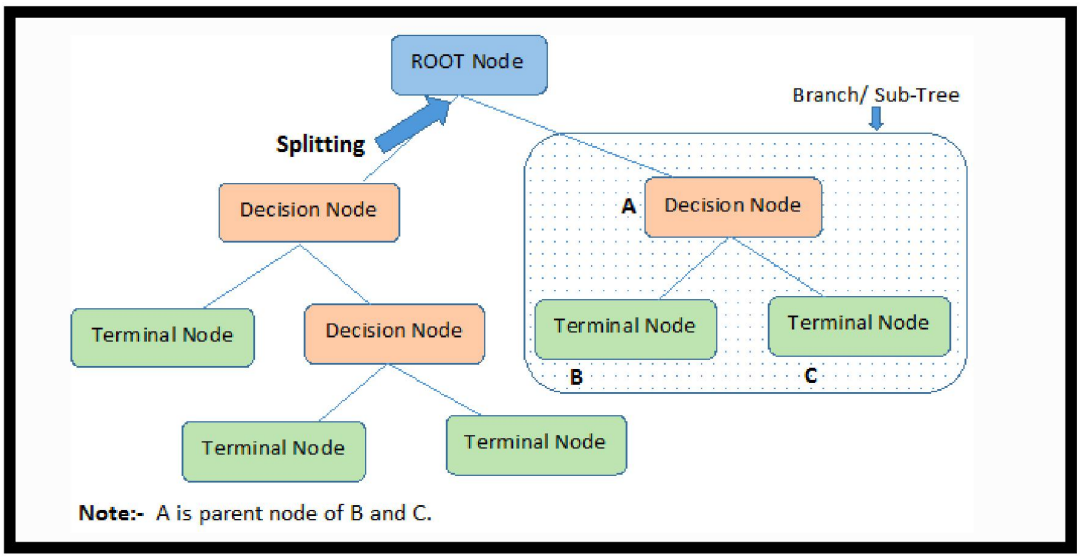


Figure 1: General structure of a decision tree

Splitting the attributes:

Many measures are used for selecting the best split. The best split is often decided based on the degree of impurity of child nodes of a node. The smaller the degree of impurity, the more skewed the class distribution. Two examples of impurity are as follows:

1. **Gini Index:**

$$Gini = 1 - \sum_{i=1}^{c} {p_i}^2$$

2. **Entropy:**

$$Entropy = \sum_{i=1}^{c} -{p_i}^2 log_2 p_i$$

Where c is the number of classes and $p_i$denotes the fraction of records belonging to class i at a given node. Given $0log_20 = 0$ for entropy calculations.

Implementing decision tree:

We will be using the Scikit-Learn for implementing a DT. The required libraries and steps are given in the following steps.

The following illustration uses spambase dataset. The dataset contains 56 attributes and 1 target attribute. The target contains two values – 1 (spam) and 0 (not a spam).

Required libraries:

1. import pandas as pd – **for loading the dataset or pre-processing**
2. import numpy as np – **for pre-processing the dataset**
3. from sklearn.model_selection import train_test_split – **train & testing**
4. from sklearn.tree import DecisionTreeClassifier – **for using decision tree**
5. from sklearn.metrics import classification_report,confusion_matrix – **evaluation**
6. from sklearn.metrics import accuracy_score – **measuring accuracy**

Steps:

1. Import the required libraries for loading the dataset
2. Load the dataset using the given command and display the first 5 rows of the dataset:

```
df = pd.read_csv('dataset_file', header = None)
```

3. Dividing the dataset for training and testing
   a. Import the libraries required for training and testing
   b. Divide the attributes into – input attributes and target attribute

```
X = df [df.columns [:-1]]

Y = df [len (df.columns)-1]
```

   c. Divide the dataset into training and testing part with ratio of (80:20) (Holdout method)

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state = 30)
```

Note: To obtain a deterministic behaviour during fitting, random state has to be fixed to an integer value. The DT function randomly permutes and selects the features at each split. This is because we can specify number of features to be considered as a parameter in the function. So, if the max_features < no_of_features in dataset, then the features selected at every execution of the function will be

different. Thus, to get the same and deterministic results we fix the random state parameter.

4. Applying the decision tree
   a. Import the required library to use decision tree function
   b. Call the decision tree function from the library and apply it on the dataset. DT function in Python uses Gini measure for split by default.

```
dtree = DecisionTreeClassifier()

dtree.fit (X_train, Y_train)
```

5. Evaluate the classifier
   a. Import the libraries required for evaluation
   b. Perform the predictions on the testing part of the dataset

```
predictions = dtree.predict (X_test)
```

   c. Perform the evaluations

```
print (classification_report (Y_test, predictions))
print ("Confusion Matrix")
print (confusion_matrix (Y_test, predictions))
print ("\n Accuracy")
print (accuracy_score (Y_test, predictions))
```

Look at all the evaluation metrics e.g., accuracy, precision, recall, f-measure, etc. and identify the best metric to evaluate the Spambase classification.

Visualization of tree:

1. Import the matplotlib library to plot the graphs in python
2. Apply the following commands to plot and visualize the decision tree

```
import matplotlib.pyplot  as plt
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (16,16), dpi=1024)
tree.plot_tree(dtree)
fig.savefig('decision_tree.png')
```

Questions

1. Create a decision stump and find the majority class, error.
2. Create a DT classifier using entropy to measure the quality of split. (random state parameter of the DT classifier can take integer values from. We will be using random state value as 30.)
3. Compare the two trees on the basis of their accuracy. Which tree gives better accuracy (using gini index or entropy).
4. Fully grown decision tree usually over-fits the data. To overcome the problem of overfitting, pruning techniques are used. One of which is restricting the height of the tree. Find the value of the height for which the decision tree model is giving the highest accuracy. (Use entropy measure)
5. Find the feature importance from the trained model and rebuild the model using top 10 important features only. The required library and function is given in the hint.

   Hint:
   ```
   import heapq
   feature_imp = dtree_entropy.feature_importances_
   print(feature_imp)
   ```

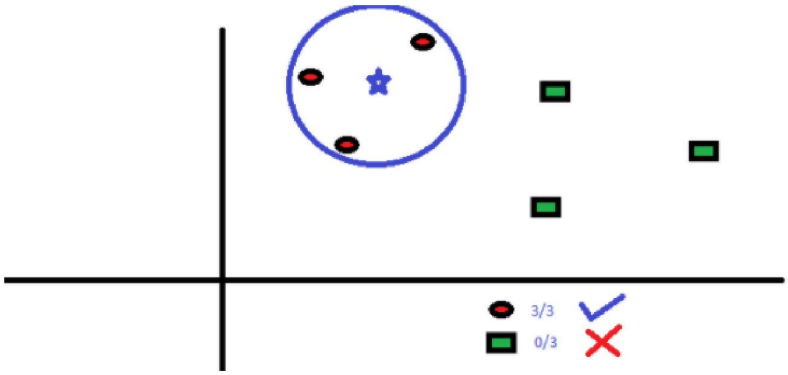
   Is there any information loss in the tree created above?

K-Nearest Neighbour (Lab-3)

K-NN Classifier:

K nearest neighbours is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions, cosine similarity, etc.). A new record is classified by a majority vote of its K neighbours, with the case being assigned to the class most common amongst its K nearest neighbours measured by a similarity measure. If K = 1, then the case is simply assigned to the class of its nearest neighbour. The K-NN classifier with K=1 is known as *rote classifier.*

Consider the example given in Figure below. Suppose you want to classify blue star (BS) in either as "red circle" or "green square". The circle in the figure shows 3(=K) nearest neighbors of BS. The three closest points to BS is all RC. Hence, BS will be labelled as "red circle". Here, the choice became very obvious as all three votes from the closest neighbor went to red circle. However, you may get different cases for different values of K. The choice of the parameter K is very crucial in this algorithm.



The distance functions used for Numeric fields are given below:

Distance functions

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

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

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

K-NN is a classification algorithm for binary (two-class) and multi-class classification problems.

Steps of K-NN:

Step 1 –Load the dataset.

Step 2 – Choose the value of K i.e. the number of nearest data points.

Step 3 – For each point in the test data do the following –

- Calculate the dissimilarity between test data and each row of training data with the help of any of the dissimilarity measure. The most commonly used method to calculate distance is Euclidean.
- Based on the dissimilarity value, sort the neighbours (data points) in ascending order.
- Choose the top K neighbors from the sorted array.
- Assign a class to the test point based on most frequent class of these neighbors.

Step 4 – End

Implementing K-NN:

We will be using the Scikit-Learn for implementing the K-NN classifier. The required libraries and steps are given as follows. We will be working with the spambase dataset. The dataset contains 56 attributes and 1 target attribute. The target contains two values – 1 (spam) and 0 (not a spam).

Required libraries:

1. import pandas as pd – **for loading the dataset or pre-processing**
2. import numpy as np – **for pre-processing the dataset**
3. from sklearn.model_selection import train_test_split – **train & testing**
4. from sklearn.neighbors import KNeighborsClassifier – **for KNN Classifier**
5. from sklearn.metrics import classification_report,confusion_matrix – **evaluation**
6. from sklearn.metrics import accuracy_score – **measuring accuracy**

Steps:

1. Import the required libraries for loading the dataset
2. Load the dataset using the given command and display the first 5 rows of the dataset:

```
df = pd.read_csv('dataset_file', header = None)
```

3. Dividing the dataset for training and testing
   a. Import the libraries required for training and testing
   b. Divide the attributes into – input attributes and target attribute

```
X = df [df.columns [:-1]]

Y = df [len (df.columns)-1]
```

   c. Divide the dataset for holdout method with ratio of (80:20).

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state = 30)
```

4. Applying the K-nearest neighbour classifier
   a. Import the required library to use KNN classifier
   b. Call the required function from the library using appropriate parameters and apply it on the dataset.

```
Knn = KNeighborsClassifier (n_neighbors=3, metric = 'euclidean')

Knn.fit (X_train, Y_train)
```

5. Evaluate the classifier
   a. Import the libraries required for evaluation
   b. Perform the predictions on the testing part of the dataset

```
predictions = Knn.predict (X_test)
```

   c. Perform the evaluations

```
print (classification_report (Y_test, predictions))
print ("Confusion Matrix")
print (confusion_matrix (Y_test, predictions))
print ("\n Accuracy")
print (accuracy_score (Y_test, predictions))
```

Applying K-fold cross validation method for training and testing:

1. Import the cross validation library for k-fold cross validation method.
2. Apply the following commands to train and test the model using cross validation method and observe the change in performance of 3-NN model
3. Here the no. of folds are taken as '5'.

```
from sklearn.model_selection import cross_val_score
knn_cv = KNeighborsClassifier (n_neighbors=3, metric = 'euclidean')
scores = cross_val_score (knn_cv, X, Y, cv=5, scoring='accuracy')
print ('scores: ', scores)
print ('mean score: ', scores.mean())
```

Weighted KNN:

The parameter used to decide the class of a data point is the majority voting, but this can be a problem if the nearest neighbors vary widely in their distance and the closest neighbors more reliably indicate the class of the data point. The weighted K-NN gives weights inversely proportional to the distances of K nearest points from the unknown point. The class having more weight is assigned to the testing data point.

Questions

1. Normalize (min-max normalization) the dataset and apply 3-NN using both Euclidean and Manhattan distance.
2. Write your observation regarding change in the performance of KNN.
3. Find the accuracy of rote learner (K=1).
4. Find the accuracy of the models by taking K from 1 to 20. (Using euclidean distance and k(5)-fold cross validation method)
5. Plot the graph between K and the accuracy/f1-measure. Find the best value of K.
6. Repeat questions 4 and 5 using Manhattan distance as dissimilarity measure.
7. Implement the weighted K-NN model.
   Use k-fold cross validation method for train-test split
   [*Hint: Use the following 'weights' parameter in the K-NN classifier function: knn_weighted = KNeighborsClassifier (n_neighbors=K, weights='distance')]*
8. What do you get as the best Kfor weighted K-NN.

Naive Bayes (Lab-4)

Naïve Bayes Classifier:

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 performs equally well as many common classifiers. Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values. However, Gaussian distribution can be used to find class-conditional probability for the continuous attributes.

It is called Naive Bayes because the calculation of the probabilities for each hypothesis is simplified to make their calculation easy. Rather than attempting to calculate the combined probability P(d1, d2, d3|h), attributes are assumed to be conditionally independent and the probability P(d1, d2, d3|h)is calculated as P(d1|h) * P(d2|h) * P(d3|h).

Implementing Naïve Bayes:

We will follow the same procedure as in previous classifiers to use Scikit-Learn for implementing Naïve Bayes classifier. The required libraries and steps are given as follows:

We will be working with the bank dataset. The dataset contains 11 attributes and 1 target attribute. The target contains two values – yes (pep) and no (not pep).

Required libraries:

1. import pandas as pd – **for loading the dataset or pre-processing**
2. import numpy as np – **for pre-processing the dataset**
3. from sklearn.model_selection import train_test_split – **train & testing**
4. from sklearn.naive_bayes import GaussianNB– **for naïve bayes Classifier**

   For Prediction and Evaluation

5. from sklearn.metrics import classification_report, confusion_matrix – **evaluation**
6. from sklearn.metrics import accuracy_score – **measuring accuracy**

Steps:

1. Import the required libraries for loading the dataset
2. Load the dataset using the given command and display the first 5 rows of the dataset:

```
df = pd.read_csv('dataset_file', header = None)
```

3. Use the following commands to perform pre-processing and transform the dataset from nominal to numeric. This is done because the classifier function accepts the dataset in numeric form only

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder ()
df.attribute = le.fit_transform (df.attribute)
* repeat this for all the required columns or attributes
df.head ()
```

4. Dividing the dataset for training and testing
   a. Import the libraries required for training and testing
   b. Divide the attributes into – set of input attributes and target attribute

```
X = df [df.columns [:-1]]
Y = df [len (df.columns)-1]
```

   c. Divide the dataset according to holdout method.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state = 30)
```

5. Applying the Naïve Bayes model
   a. Import the required library for Naïve Bayes classifier
   b. Call the required function from the library using appropriate parameters and apply it on the dataset.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

6. Evaluate the classifier
   a. Import the libraries required for evaluation
   b. Perform the predictions on the testing part of the dataset

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30,
random_state = 30)
Y_pred = gnb.fit(X_train, Y_train).predict(X_test)
```

c. Print the number of misclassified data points:

```
print("Number of mislabeled points out of a total %d points : %d" %
(X_test.shape[0], (Y_test != Y_pred).sum()))Y_pred = gnb.fit(X_train,
Y_train).predict(X_test)
```

```
print (confusion_matrix (Y_test, predictions))
print ("\n Accuracy")
print (accuracy_score (Y_test, predictions))
```

Questions

1. Consider "current_act" as an irrelevant attribute. Remove it and find the accuracy of newly created Naive Bayes classifier. Compare its accuracy with that of obtained in (1).
2. Compare the performance of K-NN with and without the irrelevant attribute considered in (2). Compare your observations of K-NN with that of NB.
3. Load "car.csv" dataset and apply NB classifier
4. Find the correlation between the attributes of the dataset and arrange them in ascending order.
5. Remove one of the highly correlated attributes and apply Naive Bayes classifier. Compare the performance of the obtained model with the performance of earlier models.

Ensemble (Lab-5)

Ensemble Classifier:

Ensemble learning helps in improving the predictive performance by combining several models into a single model.

Most ensemble methods use a single base learning algorithm i.e. learners of the same type, leading to homogeneous ensembles.
There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

Bagging:

Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different classifiers on different datasets obtained by choosing samples randomly with replacement.

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses majority voting for classification and averaging for regression.

Boosting:

Boosting is a general ensemble method that creates a strong classifier from a number of classifier weighted by their performance. It also gives higher weights to the different samples in each round of classifier set generation.

This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added.

AdaBoost is one of the most successful boosting algorithms developed for binary classification.

Implementing Ensemble of models:

We will be using Scikit-Learn for implementing Ensemble model as the previous classifier. The required libraries and steps are given as follows:

We will be working with the letter recognition dataset. The dataset contains 16 numeric attributes and 1 target attribute. The target contains 26 values – letters (A-Z). The

number of instances in dataset is 20000 and no missing values. You can refer metadata for more details.

Required libraries:

```
To load dataset:          import pandas as pd
Preprocessing:            from sklearn import preprocessing
Decision Tree:            sklearn.tree import DecisionTreeClassifier
K-NN Classifier:          from sklearn.neighbors import KNeighborsClassifier
Naive Bayes:              from sklearn.naive_bayes import GaussianNB
Train & Test Split:       from sklearn.model_selection import train_test_split
K-Fold:                   from sklearn.model_selection import cross_val_score
Bagging Algorithms:       from sklearn.ensemble import BaggingClassifier
Voting Classifier:        from sklearn.ensemble import VotingClassifier

For Prediction and Evaluation:
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

Steps:

1. Import the required libraries for loading the dataset
2. Load the dataset using the given command and display the first 5 rows of the dataset:

```
df = pd.read_csv('dataset_file', header = None)
```

3. Dividing the dataset for training and testing
   a. Import the libraries required for training and testing
   b. Divide the attributes into – input attributes and target attribute

```
X = df [df.columns [:-1]]

Y = df [len (df.columns)-1]
```

   c. Apply holdout method for training and testing with ratio of (70:30).

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state = 30)
```

4. Apply all the models to be used for ensemble
   c. Import the required library for these classifiers
   d. Call the required functions from the library using appropriate parameters and apply it on the dataset as done in the previous exercises.

5. Evaluate the individual classifiers
   a. Import the libraries required for evaluation
   b. Perform the predictions on the testing part of the dataset

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30,
random_state = 30)

Y_pred = xyz.fit(X_train, Y_train).predict(X_test)
```

   c. Perform the evaluations

```
print (classification_report (Y_test, predictions))
print ("Confusion Matrix")
print (confusion_matrix (Y_test, predictions))
print ("\n Accuracy")
print (accuracy_score (Y_test, predictions))
```

Questions

1. Ensemble Method by manipulation of Dataset (Bagged Decision Trees)

   Bagging performs best with algorithms that have high variance. A popular example is fully grown decision trees, often used without pruning. We will create decision tree classifiers with and without bagging ensemble method and compare their performance.

   Steps:

   a. Implement the Decision Tree classifier
   b. Implement Bagging Classifier using base classifiers of the same type i.e. Decision tree (no. of classifiers = 5)

```
from sklearn.ensemble import BaggingClassifier
dtree = DecisionTreeClassifier(criterion='entropy', random_state = 30)
model = BaggingClassifier(base_estimator=dtree, n_estimators=num_trees,
random_state=seed)
```

   c. Train and view scores using k-fold cross validation with k=5
   d. Test on the testing part of the dataset

2. Ensemble Method by manipulation of Classifiers (using Voting Classifier)

   The Voting Classifier takes in a list of different estimators as arguments and a voting method. The **hard** voting method uses the predicted labels and a majority rules system, while the **soft** voting method predicts a label based on the argmax/largest predicted value of the sum of the predicted probabilities.

After we provide the desired classifiers, we need to fit the resulting ensemble classifier object. We can then get predictions and use accuracy metrics.

Steps:

a. Implement different base classifiers. Classifiers to be used are: Decision tree, 3-NN with Euclidean distance, 5-NN with euclidean distance, 5-NN with Manhattan distance and Naive Bayes
b. Create an ensemble of the classifiers using Voting Classifier with hard voting method

```
from sklearn.ensemble import VotingClassifier
#Build Voting Classifier using above estimators and hard voting method
voting_clf = VotingClassifier (estimators=[('model_1', variable),
('model_2',variable).....('model_n', variable), voting='hard']
voting_clf.fit(X_train, Y_train)
scores = cross_val_score(voting_clf, X_train, Y_train, cv=5, scoring = 'accuracy')
```

c. Test on the testing part of the dataset

3. Manipulating the features
   By manipulating the features we mean taking different set of features for training different models (or same base model).

   Steps:

   a. Generate five feature sets using random vector (including 10 features each)
   b. Apply Decision Tree with same hyper-parameters on these five different feature sets
   c. Test the obtained model on the testing dataset
      (Note: The ensemble classifier would be evaluated on the entire dataset)

4. Manipulating the classes
   By manipulating the features we mean converting the multi-class problem into a two class problem and then using this manipulated data for training different models (or same base model).

   Steps:

   a. Convert the problem into two-class problem. Create 5 sets of two classes using random vector
   b. Apply five Decision tree with same hyper-parameters on these five sets.
   c. Test the obtained model on the testing dataset
      (Note: The ensemble classifier would be evaluated on the entire dataset)

5. Find which method performs the best