Department of Computer Science & Engineering
PESIT_BSC
Bangalore
Machine Learning Laboratory
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2016 -2017)
SEMESTER – VII

Subject Code: 15CSL76	**IA Marks : 20**
No of Lecture Hrs/Week : 01I + 02P	**Exam hours : 3**
Total No of Lecture Hours : 40	**Exam Marks : 80**

Faculty: Pooja Agarwal, Vandana M L

AIM: This course will enable students to
1. Make use of Data sets in implementing the machine learning algorithms
2. Implement the machine learning concepts and algorithms in any suitable language of choice

DESCRIPTION:
1. The programs can be implemented in either JAVA or Python.
2. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.
3. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students

S. No	Program Title
1	Implement and demonstrate the **FIND-S** algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2	For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.
3	Write a program to demonstrate the working of the decision tree based **ID3 algorithm.** Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.
4	Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.
5	Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
6	Assuming a set of documents that need to be classified, use **the naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7	Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

8	Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using **k-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
9	Write a program to implement **k-Nearest Neighbor algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
10	Implement the non-parametric **Locally Weighted Regression algorithm** to fit data points. Select appropriate data set for your experiment and draw graphs.

Program 1: Implement and demonstrate the "find-s" algorithm for finding the most specific hypothesis based on a given set of training data samples: read the training data from csv file.

Solution:

Find-S algorithm is a searching algorithm.

Purpose: To find the maximally specific hypothesis from the set of hypothesis space.

Method: Begin with most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.

Notations:

- D - Training data set
- X - Set of instances with in training data set
- x - particular instance in training example
- H - Set of possible hypothesis
- h - particular hypothesis described by conjunction of constraints on the attributes
- a_i - constraint attribute of hypothesis, a_i can have a value of θ (no value), or any value(a_i =sunny), or ?(any value)
- c - target concept

Algorithm:

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
        • For each attribute constraint $a_i$ is satisfied by x
        • Then do nothing
        • Else replace $a_i$ in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Example:

Positive and negative training data set for the target concept *"Enjoy sport"*

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**Enjoy sport**
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes

2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- *Step 1*: Initialize h to most specific hypothesis

  h (θ ,θ ,θ ,θ ,θ ,θ , θ)
- *Step 2: 1st positive training example*

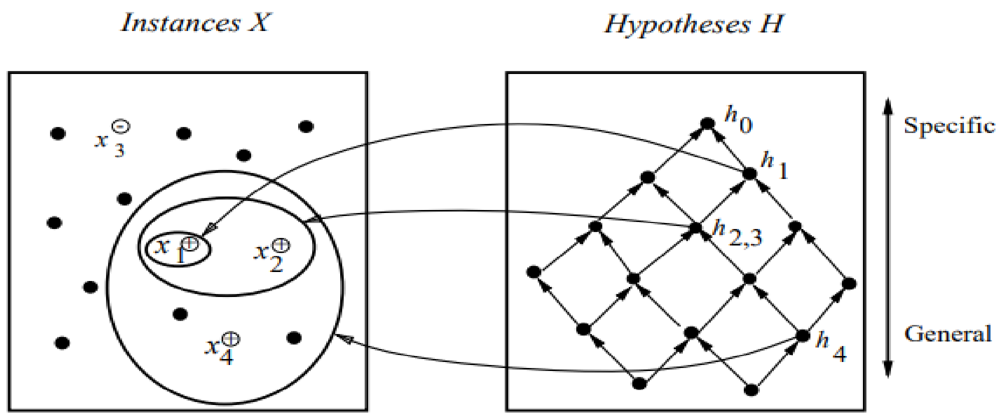
  h (Sunny, Warm, Normal, Strong, Warm, Same)
- *Step 3: 2nd Positive training example*

  h (Sunny, Warm, ? , Strong, Warm, Same)
- *Step 4: 3rd negative training example*

  h (Sunny, Warm, ? , Strong, Warm, Same)
- *Step 5: 4th positive training example*

  h (Sunny, Warm, ? , Strong, ?, ?)

Hypothesis Space Search by Find-S



h_0 = <∅, ∅, ∅, ∅, ∅, ∅>

x_1 = *<Sunny Warm Normal Strong Warm Same>*, + h_1 = *<Sunny Warm Normal Strong Warm Same>*

x_2 = *<Sunny Warm High Strong Warm Same>*, + h_2 = *<Sunny Warm ? Strong Warm Same>*

x_3 = *<Rainy Cold High Strong Warm Change>*, - h_3 = *<Sunny Warm ? Strong Warm Same>*

x_4 = *<Sunny Warm High Strong Cool Change>*, + h_4 = *<Sunny Warm ? Strong ? ? >*

Properties:

- Find-S is guaranteed to output the most specific hypothesis within H that is consistent with positive training examples.
- Find-S algorithm ignores the negative examples.

Drawbacks of Find-S:

- Algorithm can’t tell whether it has learned the right concept because it picks one hypothesis out of many possible ones
- Can’t tell when training data inconsistent because it ignores the negative examples: doesn’t account for noise
- Picks a maximally specific h, depending on H, there might be several correct hypothesis!

Gender	Age	Student	PreviouslyDeclined	HairLength	Employed	Typeofcolateral	Firstloan	LifeInsurence	Risk
Male	Young	No	Yes	Short	No	House	No	No	Low
Male	Young	No	No	Long	Yes	Car	Yes	Yes	High
Male	Young	No	No	Short	Yes	Car	Yes	No	High
Male	Old	No	No	Short	No	House	No	Yes	High
Male	Old	No	No	Long	No	House	No	Yes	Low
Female	Young	Yes	No	Long	Yes	Car	No	Yes	High
Male	Old	No	Yes	Short	No	Car	Yes	No	Low
Male	Young	Yes	Yes	Long	No	House	No	No	High
Female	Old	No	Yes	Short	Yes	House	No	Yes	Low
Male	Old	Yes	Yes	Long	No	House	No	Yes	High
Male	Young	No	No	Short	Yes	Car	Yes	No	Low
Female	Old	No	No	Long	No	House	No	No	High
Male	Young	No	Yes	Short	No	Car	Yes	No	low

Male	Old	No	No	Long	Yes	Car	No	No	High

Example of training data set:

Task: To determine whether the risk of defaulting on a loan is "high" or "low," based on a set of binary attributes. The task is to learn a hypothesis that best fits the data.

Implementation of the FIND-S algorithm is to find the maximally specific hypothesis.

- The ‘inupt.scv’ including examples with different values for 9 binary attributes and a "high" or "low" label for each example.
- The following are the attributes values are:
  - ✓ Gender (Male, Female)
  - ✓ Age (Young, Old)
  - ✓ Student? (Yes, No)
  - ✓ PreviouslyDeclined? (Yes, No)
  - ✓ HairLength (Long, Short)
  - ✓ Employed? (Yes , No) •
  - ✓ TypeOfColateral (House, Car)
  - ✓ FirstLoan (Yes, No)
  - ✓ LifeInsurance (Yes, No)
- The hypothesis space H should consist of all possible conjunctions over the nine attributes, where each conjunct is of the form ATTR=[value], ATTR='?', or ATTR='null'.
- In the program, you can choose to exclude the possibility “ATTR=null”, and find an alternative way to represent the single hypothesis that always returns “False”.

The program Prints (to standard output),

1. The size of the input space (number of possible unique inputs).
2. Let |C| = the size of the concept space (the space of all possible concepts; i.e., Boolean functions of the input). Prints, by itself on the next line, the number of decimal digits in |C|.
3. Prints, by itself on the next line, the size of the hypothesis space H (the space of all semantically distinct conjunctions of the type described above).
4. Runs the FIND-S algorithm on '9Cat-Train.labeled', using the hypothesis space H described above, and prints to the file "output.txt" the current hypothesis after every 30 training instances (namely, after 30, 60, 90,... training instances, counting both positive and negative instances). A hypothesis is displayed as a (tab) delimited list of attribute values enclosed in angle brackets. The current hypothesis after every 30 instances should be printed on a separate line.

Code: Finds:

```
import java.io.BufferedReader; //class reads text from a character input stream
import java.io.BufferedWriter; //class writes text to a character output stream
import java.io.FileReader; //class is a convenience class for reading character files
import java.io.FileWriter; // class is a convenience class for writing character files
```

Program 2: For a given set of training data example stored in .csv file, implement and demonstrate the Candidate-Elimination Algorithm to output and describes the set of all hypotheses consistent with training example.

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set D of examples. The final output hypothesis are consistent with the given the given training example. To find such hypotheses the simplest way is as follow:

a. List all the hypotheses in H
b. Get each training example d from D
    i. Remove all the hypotheses which are not consistent with the training example d.

After execution of all the examples, the remaining hypotheses are all consistent with the given training example set. This procedure is only useful when the hypotheses set H is finite. This procedure is called as List-Then-Eliminate Algorithm.

The other algorithm is Candidate-Elimination Algorithm. The Candidate-Elimination Algorithm works on the same principle as the above List-Then-Eliminate Algorithm. However, it uses more compact representation of the version space. The version space is represented in term of most general and most specific members. To understand this following definitions are used.

General Boundary G, with respect to hypothesis space H and training data D, is the set of maximally general hypotheses consistent with D.

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg\exists g' \in H)[(g' >g\ g) \wedge Consistent(g', D)]\}$$

Specific Boundary S, with respect to hypothesis space H and training data D, is the set of maximally specific hypotheses consistent with D.

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg\exists s' \in H) [(s >g\ s') \wedge Consistent(s', D)]\}$$

<u>Candidate-Elimination Learning Algorithm</u>

The Candidate-Elimination algorithm computes the version space containing all hypothesis from H that are consistent with an observed sequence of training examples. It begins by initializing the version space to the set of all hypotheses in H; that is by initializing the G boundary set to contain most general hypothesis in H

$$G0 = \{(?, ?, ?, ?)\}$$

Then initialize the S boundary set to contain most specific hypothesis in H

$$S0 = \{(\Theta, \Theta, \Theta, \Theta)\}$$

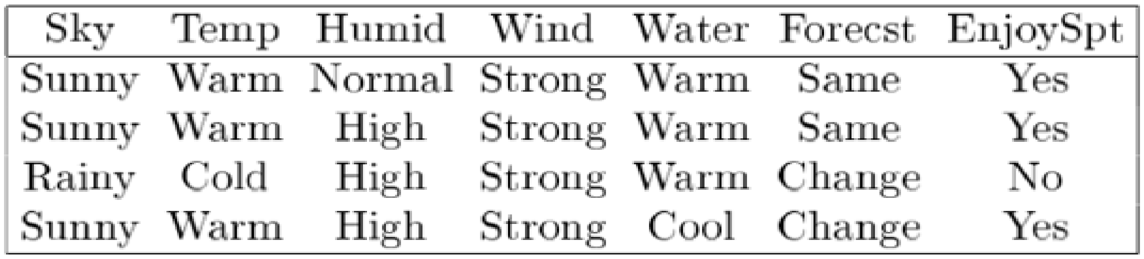
For each training example, these S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypothesis found inconsistent with the new training examples. After execution of all the training examples, the computed version space contain all the hypotheses consistent with these training examples. The algorithm is summarized as below:

Candidate-Elimination Algorithm

```
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For every learning example d, do
        - If d is a positive example
                 Remove from G any hypothesis inconsistent with d
                 For each hypothesis s in S that is not consistent with d
                         Remove s from S
                         Add to s all minimal generalization h of s such that
                                 H is consistent with d, and some member of G is more general than h
                         Remove from s any hypothesis that is more general than another hypothesis is S
        - If d is negative example
                 Remove from S any hypothesis inconsistent with d
                 For each hypothesis g in G that is not consistent with d
                         Remove g from G
                         Add to G all minimal specializations h of g such that
                                 H is consistent with d, and some member of S is more specific than h
                         Remove from G any hypothesis that is less general than another hypothesis in G
```

For the implementation of Candidate-Elimination Algorithm the EnjoySport data set can be used from UCI repository. The data set contains around 700 data samples. Few of the samples are showed in Table 1.

Table 1. Example data sample of EnjoySport data set



Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Implementation Details:

- For the implementation of the Candidate-Elimination algorithm, python programing language has been used.
- As a part of pre-build library only the csv library has been used. This is required to read the dataset from the .csv file.
- The algorithm starts with loading the dataset into the variable.

- Using the initialization function, **CandidateElimination(datasetShort, f)** the full dataset along with the attribute vales (stored in f) is loaded into the class named **CandidateElimination**
- The first step is to initialize the maximally general hypotheses G in H and maximally specific hypotheses S in H using the functions **initializeG()** and **initializeS()** repectively. The G will be initialized to **{?, ?, ?, ?, ?, ?}** and S to **{-, -, -, -, -, -}**.
- For each of the training sample, the first requirement is to check whether the sample is positive or negative. To do this, two functions named **is_positive** and **is_negative** is implemented.
- If the sample is positive then,
    - Remove the hypothesis from G which are inconsistent for the given sample. This can be done by the function **remove_inconsistent_G**
    - Now for each hypothesis s in S that is not consistent with d, checked by **consistent(s, sample),** do the following:
        - Remove s from S
        - Generalize the inconsistent s using **generalize_inconsistent_S**
            - Get the general hypothesis using **get_general** and add it the S
    - Remove from s any hypothesis that is more general than another hypothesis is S using the function **remove_more_general**
- If the sample is negative then,
    - Remove the hypothesis from S which are inconsistent for the given sample by using the function **remove_inconsistent_S**
    - Now for each hypothesis g in G that is not consistent with d, check using **consistent(g, sample),** do the following,
        - Remove g from G
        - Specialize the inconsistent g using **specialize_inconsistent_G**
            - Get the specific hypothesis using **get_specific** and add it the G
        - Remove from G any hypothesis that is less general than another hypothesis in G using the function **remove_more_specific**

Inputs and outputs:

The input to the program is the set of training examples having the structure shown in the table 1. This can be directly given to the program in a sting tuple or can be read from the csv file.

Example,

- Give set of examples directly to program in the form of sting tuple. This is easy if the given samples are less.

```
datasetShort=[(('sunny','warm','normal','strong','warm','same'),'yes'),(('sunny','warm','high','strong','warm','same'),'yes'),(('rainy','cold','high','strong','warm','change'),'no'),(('sunny','warm','high','strong','cool','change'),'yes')]
```

Here, one tuple consist of one training example. For each tuple, the first value of the tuple is the value of the corresponding attributes. For example, in the first training example, the value of the 'Sky' attribute is 'sunny' while the value of 'Temp' attribute is 'warm' and so on.

If the input to the program is the one given above then the output will be as below:

Final S:: [('sunny', 'warm', '?', 'strong', '?', '?')]
Final G:: [('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?')]
Final Version Space: [('?', 'warm', '?', 'strong', '?', '?'), ('sunny', '?', '?', 'strong', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('sunny', 'warm', '?', 'strong', '?', '?'), ('sunny', '?', '?', '?', '?', '?'), ('sunny', 'warm', '?', '?', '?', '?')]

The final version space contains the general and specific boundaries and all the hypothesis in between this two boundaries.

- Read the training examples from the csv file. This is needed if the training examples are more.

If you wish to read the raining examples from you need to import the csv library using the command **import** csv. Then read all the samples one by one and save it the appropriate data structure. Here is the python code to do it.

```
with open('candidate_elem_dataSet.csv') as csvfile:
   readCSV = csv.reader(csvfile, delimiter=',')
   for row in readCSV:
      dataset.append(((row[0], row[1], row[2], row[3], row[4], row[5]), row[6]))
```

The output for the sample set provided in this csv file is as follow:

No need to continue.
The version space has converged - Specific and General boundaries are merged.
Final S:: [('?', '?', '?', 'weak', 'warm', '?')]
Final G:: [('?', '?', '?', 'weak', 'warm', '?')]
Final Version Space: [('?', '?', '?', 'weak', 'warm', '?')]

The interesting thing to observe here is that, the General and Specific boundaries are merged and thus the version space has converged and so there is no need to continue further. The final version space output will be this only one hypothesis.

<u>Program 3:</u> Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Following terminologies are used in this algorithm

- **<u>Entropy</u> :** Entropy is a measure of impurity

  It is defined for a binary class with values a/b as:

Entropy = - p(a)*log(p(a)) - p(b)*log(p(b))

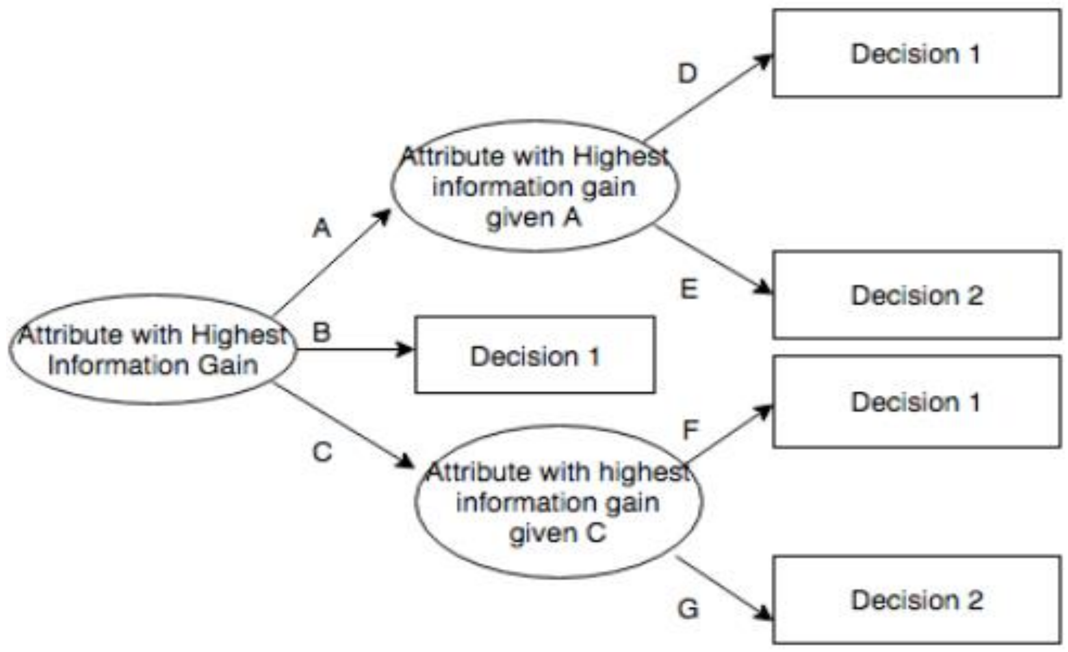
- **<u>Information Gain</u> :** measuring the expected reduction in Entropy

Gain(S,A)= Entropy(S) - Sum for v from 1 to n of (|Sv|/|S|) * Entropy(Sv)

THE PROCEDURE

1) In the ID3 algorithm, begin with the original set of attributes as the root node.
2) On each iteration of the algorithm, iterate through every unused attribute of the remaining set and calculates the entropy (or information gain) of that attribute.
3) Then, select the attribute which has the smallest entropy (or largest information gain) value.
4) The set of remaining attributes is then split by the selected attribute to produce subsets of the data.
5) The algorithm continues to recurse on each subset, considering only attributes never selected before.

<u>Flow of Decision Tree</u>


Attribute with Highest Information Gain
A
B
C
Attribute with Highest information gain given A
Decision 1
Attribute with highest information gain given C
D
E
F
G
Decision 1
Decision 2
Decision 1
Decision 2

Dataset Details

playtennis dataset which has following structure
Total number of instances=15
Attributes=Outlook, Temperature, Humidity, Wind, Answer
Target Concept=Answer

ID3 (Learning Sets S, Attributes Sets A, Attributes values V) Return Decision Tree

```
Begin
        Load learning sets S first, create decision tree root node 'rootNode', add
learning        set S into root node as its subset
        For rootNode,
        1)  Calculate entropy of every attribute using the dataset
        2)  Split the set into subsets using the attribute for which entropy is minimum (or
            information gain is maximum)
        3)  Make a decision tree node containing that attribute
        4)  Recurse on subsets using renaming attributes

End
```

- This approach employs a top-down, greedy search through the space of possible decision trees.
- Algorithm starts by creating root node for the tree
- If all the examples are positive then return node with positive label
- If all the examples are negative then return node with negative label
- I f Attributes is empty, Return the single-node tree Root, with label = most common value of Targetattribute in Example
- Otherwise -

1. Calculate the entropy of every attribute using the data set S using formula

   Entropy = - p(a)*log(p(a)) - p(b)*log(p(b))

2. Split the set S into subsets using the attribute for which the resulting entropy (after splitting) is minimum (or, equivalently, information gain is maximum) using formula

   Gain(S,A)= Entropy(S) - Sum for v from 1 to n of (|Sv|/|S|) * Entropy(Sv)

3. Make a decision tree node containing that attribute
4. Recurring on subsets using remaining attributes.

Example

Consider a dataset in which target concept has 9 positive and 5 negative examples. Then entropy is calculated as

Entropy(9+,5-)= (-9/14) log(-9/14)-(5/14)log(5/14)

=0.940

SOFTWARE, PACKAGES AND LIBRARY REQUIREMENT

Software packages- python2.7

Library – numpy, math

Numpy- Stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. You can import this library as

Import numpy

Math- It provides access to the mathematical functions defined.you can import this library as

Import math

INPUTS AND OUTPUTS

Input- Input to the decision algorithm is a dataset stored in .csv file which consists of attributes, examples, target concept.

Output- For the given dataset decision tree algorithm produces the decision tree starting with rootnode which has highest information gain.

Program 4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Artificial neural networks (ANNs) are powerful tools for machine learning with applications in many areas including speech recognition, image classification, medical diagnosis, and spam filtering. It has been shown that ANNs can approximate any function to any degree of accuracy given enough neurons and training time. However, there is no guarantee on the number of neurons required or the time it will take to train them. These are the main disadvantages of using ANNs. Here we develop the BackPropogation algorithm which learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and target values for these outputs.

BACK PROPAGATION ALGORITHM:

Multiple layer perceptron are effectively applied to handle tricky problems if trained with a vastly accepted algorithm identified as the back-propagation algorithm (error) in a supervised manner. It functions on learning law with error-correction. It is also a simplified version for the least mean square (LMS) filtering algorithm which is equally popular to error back-propagation algorithm.

In Error back-propagation training there are two computational passes via several network layers:

 A forward pass A backward pass.

In forward pass, vector input is applied to the nodes of the system propagating each layer‘s outcome to the next layer via network. To get the accurate response of the network, these outputs pass on from several layers and arrive at a set of outputs. In forward pass network weights are permanent. On other hand in the backward pass, weights are adjusted according to rule for error correction. Error signal is the actual response of the network minus the desired response.

The propagation of this error signal through the network is towards backward in direction opposite to the connections of synaptic. The move the real response of network closer to the favored response, tuning of weights is to be done. There are three unique features of a multilayer perception:

1) For each neuron in any system, its illustration has an activation function that is non-linear. The logistical function is used to define a function which is sigmoid.
2) There are layer(s) of hidden neurons not contained in the input or the output present in the neural network. The study over complex tasks is facilitated by these hidden neurons.

3) Connectivity degree is high in network. Weight's population should be changed if there is a requirement to alter the connectivity of the network.

The stochastic gradient descent version of the BACKPROPAGATION algorithm for feed forward networks containing two layers of sigmoid units.

Step 1: begins by constructing a network with the desired number of hidden and output units and initializing all network weights to small random values. . For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, then updates all weights in the network. This gradient descent step is iterated (often thousands of times, using the same training examples multiple times) until the network performs acceptably well.

Step 2: The gradient descent weight-update rule is similar to the delta training rule The only difference is that the error (t - o) in the delta rule is replaced by a more complex error term aj.

Step 3: updates weights incrementally, following the Presentation of each training example. This corresponds to a stochastic approxi- mation to gradient descent. To obtain the true gradient of E one would sum the Sj, xji values over all training examples before altering weight values.

Step 4: The weight-update loop in BACKPROPAGATION may be iterated thousands of times in a typical application. A variety of termination conditions can be used to halt the procedure.

One may choose to halt after a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold.

Dataset Detail:
Wheat seeds classification is an important agriculture process. Wheat classification system based on Artificial Neural Network (ANN) is presented. The aims is to classify three different wheat seeds into their corresponding classes. There are 210 instances with 7 continuous, realvalued, numerical attributes. The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agro physics of the Polish Academy of Sciences in Lublin. Data is collected from UCI website repository.

Number of Instances: 210

Number of Attributes: 7
To construct the data, seven geometric parameters of wheat kernels were measured:

1. area A, 2. perimeter P, 3. compactness C = 4*pi*A/P^2, 4. length of kernel, 5. width of kernel, 6. asymmetry coefficient 7. Length of kernel groove. All of these parameters were real-valued continuous

Output:

```
Neural network model:
n_hidden_nodes = [5]
 l_rate = 0.6
 n_epochs = 800
 n_folds = 4

Reading 'data/seeds_dataset.csv'...
 X.shape = (210, 7)
 y.shape = (210,)
  n_classes = 3

Training and cross-validating...
Fold 1/4: train acc = 94.94%, test acc = 96.15% (n_train = 158, n_test = 52)
Fold 2/4: train acc = 92.41%, test acc = 90.38% (n_train = 158, n_test = 52)
Fold 3/4: train acc = 98.10%, test acc = 92.31% (n_train = 158, n_test = 52)
Fold 4/4: train acc = 95.57%, test acc = 94.23% (n_train = 158, n_test = 52)

Avg train acc = 95.25%
Avg test acc = 93.27%
```

Program 5: Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

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'.

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:

1) Handling Of Data:

• load the data from the CSV file and split in to training and test data set.
• Training data set can be used to by Naïve Bayes to make predictions.
• And Test data set can be used to evaluate the accuracy of the model.

2) Summarize Data:

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.
• These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.
• summary data can be break down into the following sub-tasks:

a) Separate Data By Class: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.
b) Calculate Mean: We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our gaussian distribution when calculating probabilities.
3) Calculate Standard Deviation: We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.
4) Summarize Dataset: For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.

The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.
5) Summarize Attributes By Class: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

3) Make Predictions:

• Making predictions involves calculating the probability that a given data instance belongs to each class,
• then selecting the class with the largest probability as the prediction.
• Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) Evaluate Accuracy: The predictions can be compared to the class values in the test dataset and a classification\ accuracy can be calculated as an accuracy ratio between 0& and 100%.

Dataset- This problem is comprised of 768 observations of medical details for Pima indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.

Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0).
The attributes are
1) Number of times pregnant
2) Glucose level
3) Blood Pressure
4) Skin Thickness
5) Insulin Level
6) BMI
7) DPF
8) Age
9) outcome

software packages and library requirements-
The IDE used for implementing is jupyter notebook.

The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Libraries and functions used :

- Import Csv: The Import-Csv cmdlet creates table-like custom objects from the items in CSV files. Each column in the CSV file becomes a property of the custom object and the items in rows become the property values.
- diabetes_dataset.shape: displays number of rows and columns.
- diabetes_dataset.head(15): displays first 15 dataset columns
- diabetes_dataset.describe(): describes the mean ,median many other values of the dataset.
- def splitDataset(dataset, splitRatio) : splits dataset according to the given ratio in to train dataset and test dataset.
- def mean(numbers):calculates mean of the column values.
- def stdev(numbers): calculates standard deviation of the column values.

Outcome -This program predicts whether the given patient has a diabetes or not. The accuracy is calculated by using the test dataset.

The accuracy obtained is 66.66 %

Program 6: Assuming a set of documents that need to be classified, use **the naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set

<u>Naïve bayes Classifier</u>

This classifier can be used when moderate to large training set is available.

- **Problem statement:**
  - **Given features $X_1,X_2,\ldots,X_n$**
  - **Predict a label Y**

<u>Bayes Classifier</u>

$$P(c\,|\,\mathbf{x}) \propto P(\mathbf{x}\,|\,c)P(c) = P(x_1,\cdots,x_n\,|\,c)P(c) \text{ for } c = c_1,\ldots,c_L.$$

Determining the joint distribution is infeasible so Naïve Bayes classifier works on the assumption that features are independent
Naïve Bayes Classifier

$$\begin{aligned} P(x_1,x_2,\cdots,x_n\,|\,c) &= \underline{P(x_1\,|\,x_2,\cdots,x_n,c)}P(x_2,\cdots,x_n\,|\,c) \\ &= \underline{P(x_1\,|\,c)}P(x_2,\cdots,x_n\,|\,c) \\ &= P(x_1\,|\,c)P(x_2\,|\,c)\cdots P(x_n\,|\,c) \end{aligned}$$

<u>Apply the MAP classification rule: assign to *c if</u>**

$$[P(a_1\,|\,c^*)\cdots P(a_n\,|\,c^*)]P(c^*) > [P(a_1\,|\,c)\cdots P(a_n\,|\,c)]P(c), \quad c \neq c^*, c = c_1,\cdots,c_L$$

<u>Learning Phase</u>

For each target value of $c_i\ (c_i = c_1,\cdots,c_L)$
  $\hat{P}(c_i) \leftarrow$ estimate $P(c_i)$ with examples in S;
  For every feature value x_{jk} of each feature $x_j\ (j=1,\cdots,F;\ k=1,\cdots,N_j)$
    $\hat{P}(x_j = x_{jk}\,|\,c_i) \leftarrow$ estimate $P(x_{jk}\,|\,c_i)$ with examples in S;

<u>Test Phase</u>

$$[\hat{P}(a'_1\,|\,c^*)\cdots \hat{P}(a'_n\,|\,c^*)]\hat{P}(c^*) > [\hat{P}(a'_1\,|\,c_i)\cdots \hat{P}(a'_n\,|\,c_i)]\hat{P}(c_i), \quad c_i \neq c^*, c_i = c_1,\cdots,c_L$$

Algorithm to train and derive inference from the Naïve Bayes model

```
TRAINMULTINOMIALNB($\mathbb{C}, \mathbb{D}$)
 1  $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
 2  $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
 3  for each $c \in \mathbb{C}$
 4  do $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}, c$)
 5      $prior[c] \leftarrow N_c/N$
 6      $text_c \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS($\mathbb{D}, c$)
 7      for each $t \in V$
 8      do $T_{ct} \leftarrow$ COUNTTOKENSOFTERM($text_c, t$)
 9      for each $t \in V$
10      do $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11  return $V, prior, condprob$
```

```
APPLYMULTINOMIALNB($\mathbb{C}, V, prior, condprob, d$)
1  $W \leftarrow$ EXTRACTTOKENSFROMDOC($V, d$)
2  for each $c \in \mathbb{C}$
3  do $score[c] \leftarrow \log prior[c]$
4      for each $t \in W$
5      do $score[c] += \log condprob[t][c]$
6  return $\arg\max_{c \in \mathbb{C}} score[c]$
```

$$\hat{P}(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B'},$$

Note :To eliminate zeros, we use
where
$B = |V|$
is the number of terms in the vocabulary.
add-one or *Laplace smoothing*, which simply adds one to each count
Accuracy = No of test samples correctly classified /Total no of samples*100
Precision = true positives true positives + false positives
Recall = true positives true positives + false negatives
Dataset: https://archive.ics.uci.edu/ml/datasets/DBWorld+e-mails
Dataset Description:
Data Set Contains 64 e-mails from DBWorld newsletter
For classification two categories can be used
1. Announces of conferences
2. Everything Else
Preprocessing: removing stop words
Features : Bag of words can be used as the features
Libraries to use:
scikit-learn
NLTK

Program 7: Write a program to construct a**Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

A Bayesian network – also called a belief network or causal probabilistic network- is a graphical representation of probabilistic information: It is a directed acyclic graph in which nodes represent random (stochastic) variables, and links between nodes represent direct probabilistic influences between the variables. In this formalism, propositions are given numerical probability values signifying the degree of belief accorded them, and the values are combined and manipulated according to the rules of probability theory. Typically, the direction of a connection between nodes indicates a causal influence or class-property relationship
Bayesian statistical inference uses probabilities for both prior and future events to estimate the uncertainty that is inevitable with prediction. The fundamental concept in Bayesian networks is that probabilities can be assigned to parameter values,

Bayes theorem, these probabilities can be updated given new data. In Bayesian models the parameter is viewed as a domain variable, with a probability distribution, since the actual value of the parameter is unknown. The causal links between the variables are represented by arrows in the model. The model is strong if the arrows can be interpreted as causal mechanisms. The relationships can, alternatively, be considered an association and this type of model would be viewed more cautiously. Bayesian networks can express the relationships between diagnoses, physical findings, laboratory test results, and imaging study findings. Physicians can determine the a priori ("pre-test") probability of a disease, and then incorporate laboratory and imaging results to calculate the a posteriori ("post-test") probability

Dataset : https://archive.ics.uci.edu/ml/datasets/Heart+Disease

Tool boxes http://bayespy.org/
BayesPy provides tools for Bayesian inference with Python. The user constructs a model as a Bayesian network, observes data and runs posterior inference. The goal is to provide a tool which is efficient, flexible and extendable enough for expert use but also accessible for more casual users.

Bayesian Belief Network is a specific type of Causal belief network.Nodes represent Stochastic Variables(features) and arcs identity direct causal influences between linked variables.Bayesian Calculus is used to determine state probabilities of each node or variable from conditional and prior probabilities

<u>Following steps are used to build the Bayesian network</u>

Step 1: Identify the variables which is set of attributes specified in the dataset(ex Medical Dataset)

Step2: Determine the domain of each variable that is set of values a variable may take

Step3: Create a directed graph network of nodes where each node represents the attribute and edges represent parent child relationship. Edge represents that the child variable is conditionally dependent on the parent.

Step4 : determine the prior and conditional probability for each attribute

Step5 : perform the inference on the model and determine the marginal probabilities

<u>Tool: PyBBN</u>

Description : PyBBN is a python library for Bayesian Belief Networks .It uses junction tree algorithm to perform inference in a Bayesian Net

Junction tree Algorithm consists of the following steps

1.Moralize

2.Triangulate

3.Form Junction Tree

4. Assign the potentials to the junction tree cliques and initialize the separator potentials to unity

5.Select a root clique

6. Carry out the message passing absorption to and from the root clique until the updates passed along both the directions of every link on the junction

7.read of the clique marginal potentials from the junction tree

PyBBN functions

BbnNode : to create a node for each attribute

Bbn().add_node :to create network structure

InferenceController.apply(): to convert BBN to join tree

EvidenceBuilder: to insert an evidence

Get_bbn_potential : to get the marginal probabilities

.

Program 8: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. Add Python ML library classes/API in the program.

Unsupervised Learning:

In machine learning, unsupervised learning is a class of problems in which one seeks to determine how the data are organized. It is distinguished from supervised learning (and reinforcement learning) is that the learner is given only unlabeled examples.

Dataset:

➢ **Iris dataset**

➢ **Number of Attributes:2 1. sepal length 2. sepal width**

➢ **Number of instances:150**

Clustering Algorithms -

1. K-means clustering:

- It is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups).
- The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.
- Data points are clustered based on feature similarity.
- The results of the K-means clustering algorithm are:
    - The centroids of the K clusters, which can be used to label new data
    - Labels for the training data (each data point is assigned to a single cluster)

  Each centroid of a cluster is a collection of feature values which define the resulting groups.

  Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

The k-means is a partitional clustering algorithm.

Let the set of data points (or instances) be as follows:

D = {x1, x2, …, xn}, where
x = (xi1, xi2, …, xir), is a vector in a real-valued space X ⊆ Rr, and r is the number of attributes in the data.

The k-means algorithm partitions the given data into k clusters with each cluster having a center called a centroid.

k is specified by the user.

Given k, the k-means algorithm works as follows:

Algorithm K-means(k, D)

1. Identify the k data points as the initial centroids (cluster centers).
2. Repeat step 1.
3. For each data point x ϵ D do.
4. Compute the distance from x to the centroid.
5. Assign x to the closest centroid (a centroid represents a cluster).
6. Re-compute the centroids using the current cluster memberships until the stopping criterion is met.

2. Expectation–maximization

➢ EM algorithm is an iterative method to find maximum likelihood estimates of parameters in statistical models, where the model depends on unobserved latent variables.

➢ Iteratively learn probabilistic categorization model from unsupervised data.

➢ Initially assume random assignment of examples to categories “Randomly label” data

➢ Learn initial probabilistic model by estimating model parameters θ from randomly labeled data

➢ Iterate until convergence:

1. Expectation (E-step): Compute P(ci | E) for each example given the current model, and probabilistically re-label the examples based on these posterior probability estimates.

2. Maximization (M-step): Re-estimate the model parameters, θ, from the probabilistically re-labeled data.
3.

The EM Algorithm for Gaussian Mixtures

➢ The probability density function for multivariate_normal is

where mu is the mean, Sigma the covariance matrix, and k is the dimension of the space where x takes values.

Algorithm:

An arbitrary initial hypothesis h=<μ1, μ2 ,.., μk> is chosen.

The EM Algorithm iterates over two steps:

Step 1 (Estimation, E): Calculate the expected value E[zij] of each hidden variable zij, assuming that the current hypothesis h=<μ1, μ2 ,.., μk> holds.

Step 2 (Maximization, M): Calculate a new maximum likelihood hypothesis h'=<μ1', μ2' ,.., μk'>, assuming the value taken on by each hidden variable zij is its expected value E[zij] calculated in step 1. Then replace the hypothesis h=<μ1, μ2 ,.., μk> by the new hypothesis h'=<μ1', μ2' ,.., μk'> and iterate.

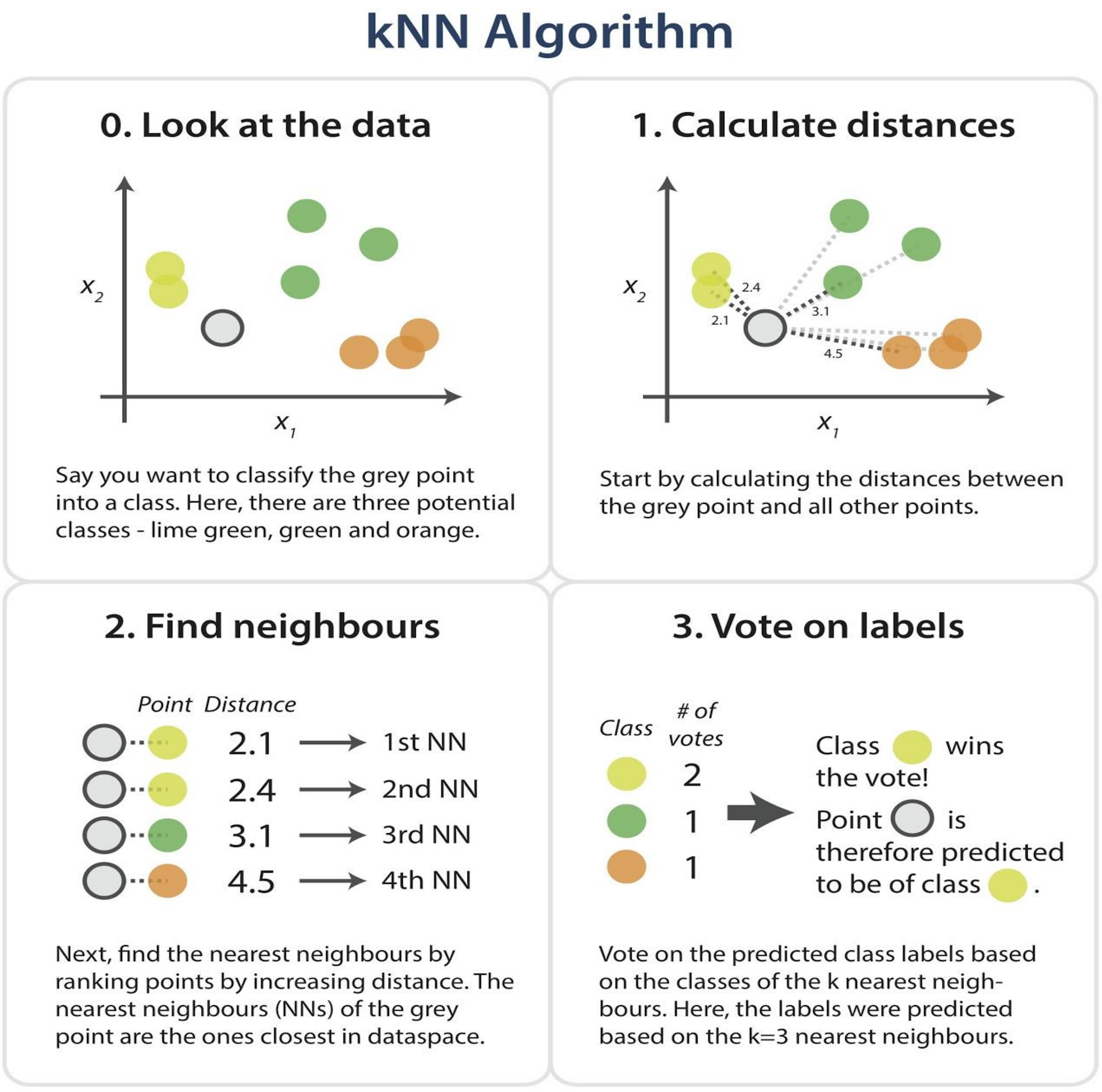
Program 9. Write a program to implement K-nearest neighbor algorithm to classify iris dataset. Print both correct and wrong predication using python machine learning.

k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

The kNN task can be broken down into writing 3 primary functions:

1. Calculate the distance between any two points
2. Find the nearest neighbours based on these pair wise distances
3. Majority vote on a class labels based on the nearest neighbour list



kNN Algorithm
0. Look at the data
x_2
x_1
Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.
1. Calculate distances
x_2
2.4
2.1
3.1
4.5
x_1
Start by calculating the distances between the grey point and all other points.
2. Find neighbours
Point
Distance
2.1
1st NN
2.4
2nd NN
3.1
3rd NN
4.5
4th NN
Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.
3. Vote on labels
Class
# of votes
2
1
1
Class wins the vote!
Point is therefore predicted to be of class .
Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

Dataset

Iris dataset, consists of flower measurements for three species of iris flower. Our task is to predict the species labels of a set of flowers based on their flower measurements. Since you'll be building a predictor based on a set of known correct classifications

The data set contains 3 classes of 151 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: class of iris plant

Attribute Information:

1.sepal length in cm

2. sepal width in cm

3. petal length in cm

4. petal width in cm

Class:

- Iris Setosa
- Iris Versicolour
- Iris Virginica

Algorithm

1. Load the data and split into train and test sets.
2. Need a measure of similarity.
3. Compute the distance function d(a,b), where a,b are the scenarios composed of N features, such that a={a1,…..,an}, b={b1,….,bn}.
4. Euclidean distance measuring: where distance d between two points (a1, a2) and (b1, b2) is given by d = sqrt((a1-b1)^2 + (a2-b2)^2). Each flower in the iris dataset has 4 dimensions, need to find the distance between each flower.

   D=sqrt((a1-b1)^2+(a2-b2)^2+(a3-b3)^2+(a4-b4)^2).
5. The zip function aggregates elements from lists to return a list of tuples. List comprehensions are a powerful Pythonic construct that facilitate quick computations on lists.
6. Iterating over values from the corresponding dimensions in the two data points, calculating the differences squared, and storing each dimension's. These are then summed and returned.
7. This pair wise calculation is done for every train instance and the given test instance.
8. Next, the distances are sorted in order to find the k closest neighbours to the test instance.

9. Then the training instances ranked from closest to furthest from our test instance, as desired. The function takes the k parameter, which controls how many nearest neighbours are returned.
10. Finally, using the nearest neighbours just identified, to get a prediction for the class of the test instance by majority voting - simply tally up which class comes up the most often among xthe nearest neighbours.

Expected Outcome

Iris dataset is a classified dataset . Using K-NN algorithm we need to predict the correct or wrong predictions, and need to get get the accuracy.

Program 10: Implement the non-parametric **Locally Weighted Regression algorithm** to fit data points. Select appropriate data set for your experiment and draw graphs.

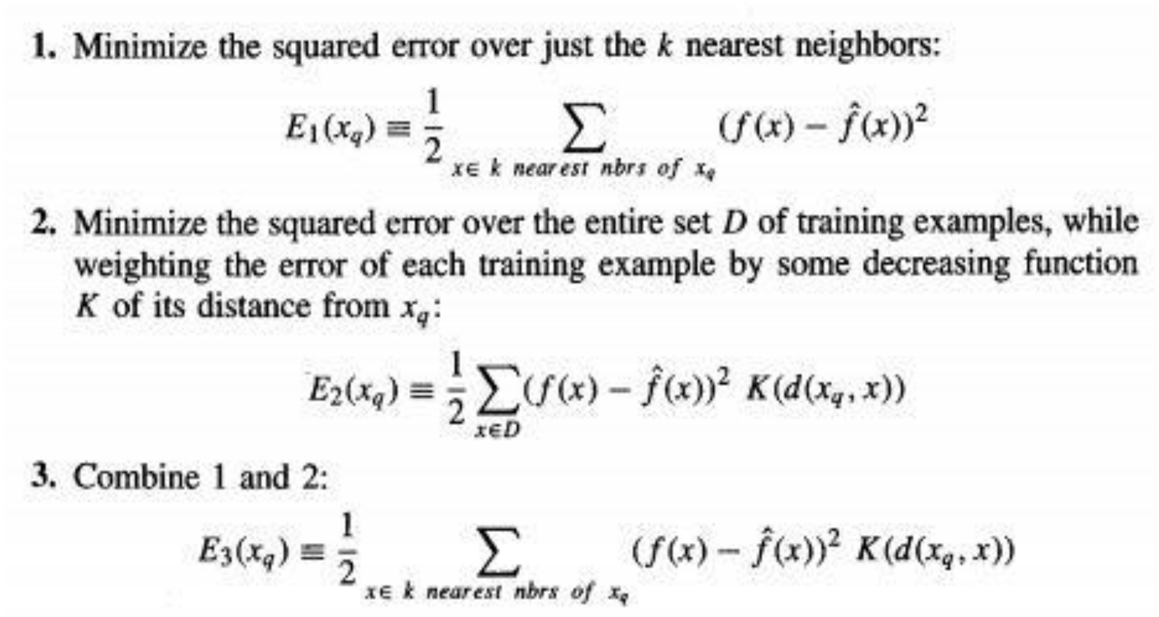
Locally Weighted Regression –

- Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data(training examples).
- Nonparametric regression requires larger sample sizes than regression based on parametric models. Because larger the data available ,accuracy will be high.

Locally Weighted Linear Regression –

- Locally weighted regression is called local because the function is approximated based a only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point.
- Query point is nothing but the point nearer to the target function , which will help in finding the actual position of the target function.

Let us consider the case of locally weighted regression in which the target function f is approximated near x, using a linear function of the form



1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \text{ k nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \text{ k nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Dataset

https://www.kaggle.com/c/locally-weighted-linear-regression-for-f16-autopilot-cruise/data

Dataset Description

Data is collected from expert pilot flying the F16 under medium-to-high turbulence conditions for normal flight operation (no combat conditions). The task is to device a model to predict the input signals provided to the Fly-by-wire flight control system, as controlled by our expert pilot.