Machine Learning Report

Assignment-2

Name : S.Hemanth Sai

Roll No. : AIE22171

1. **Pseudo Code**:

*function Euclidean\_Distance(V1, V2):*

*if length(V1) != length(V2):*

*print "Both vectors are not of same length"*

*distance = 0*

*for i = 0 to (length(V1) – 1) :*

*distance += (V1[i] - V2[i])^2*

*return sqrt(distance)*

**Euclidean\_Distance(V1, V2):**  
- Calculate euclidean distance between two vectors.  
- Checks if the length of V1 and V2 is same.  
- Adds the square of difference of same index elements and applies the sqrt in the end .

*function Manhattan\_Distance(V1, V2):*

*if length(V1) != length(V2):*

*print "Both vectors are not of same length"*

*distance = 0*

*for i = 0 to (length(V1) – 1):*

*distance += absolute value of (V1[i] - V2[i])*

*return distance*

**Manhattan\_Distance(V1, V2):**  
- Calculate manhattan distance between two vectors.  
- Checks if the length of V1 and V2 is same .  
- Adds the absolute values of difference of same index elements .

1. **Pseudo Code**:

*function calculate\_euclidean\_distance(v1, v2):*

*if length(v1) is not equal to length(v2):*

*return positive infinity*

*sum\_of\_squares = 0*

*for i from 0 to length(v1) - 1:*

*sum\_of\_squares += (v1[i] - v2[i]) squared*

*return square root of sum\_of\_squares*

*function find\_neighbors(training\_data, test\_instance):*

*distances = empty list*

*for each train\_data in training\_data:*

*distance = calculate\_euclidean\_distance(train\_data[0], test\_instance)*

*append (distance, train\_data[1]) to distances*

*return distances*

*function k\_NN\_classifier(training\_data, test\_instance, k\_value):*

*neighbors = find\_neighbors(training\_data, test\_instance)*

*sort neighbors*

*nearest\_neighbors = first k\_value elements from neighbors*

*classes = list of classes from nearest\_neighbors*

*class\_counter = Counter of classes*

*most\_common\_class = most common class in class\_counter*

*return most\_common\_class*

**find\_meighbours() and k-NN Classifier():**

- find\_neighbours() function calculates the distance between the all training data points

- k-NN\_Classifier() function classify the test instance ,based on its k nearest neighbours’ classes.

- It ranks the neighbours by its distance and picks the majority class among the k nearest neighbours (here, k=3).

1. **Pseudo Code**:

*function Label\_Encoding(data):*

*unique\_values = empty list*

*for each val in data:*

*if val is not in unique\_values:*

*append val to unique\_values*

*print unique\_values*

*encoding = empty dictionary*

*index = 0*

*for each val in unique\_values:*

*encoding[val] = index*

*increment index by 1*

*encoded\_data = empty list*

*for each val in data:*

*append encoding[val] to encoded\_data*

*return encoded\_data*

**Label\_Encoding(data):**  
- Creates an empty list and add the unique values from input to the list.  
- Creates a dictionary and stores the unique value and assigned unique digit to it.   
- Creates an empty list to encode the input data using the above dictionary.

1. **Pseudo Code**:

*function One\_Hot\_Encoding (data):*

*unique\_values = empty list*

*for each val in data:*

*if val is not in unique\_values:*

*append val to unique\_values*

*print unique\_values*

*encoding = empty dictionary*

*index = 0*

*for each val in unique\_values:*

*encoding[val] = list of length of unique\_values filled with zeros*

*encoding[val][index] = 1*

*increment index by 1*

*encoded\_data = empty list*

*for each val in data:*

*append encoding[val] to encoded\_data*

*return encoded\_data*

**One-Hot Encoding (data):**- Creates an empty list and add the unique values from input to the list.  
- Creates a dictionary and stores the unique value and assigned unique list to it.   
- Creates an empty list to encode the input data using the above dictionary