Name: Nachiket Erlekar
Roll: 41434 Batch: Q4

ML Assignment No. 3

3.1 Title

Assignment based on k-NN Classification

3.2 Problem Definition:

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If k=3, find-the class of the point (6,6). Extend the same example for Distance-Weighted k-NN and Locally weighted Averaging.

3.3 Prerequisite:

Basic of Python, Data Mining Algorithm, Concept of KNN Classification

3.4 Software Requirements:

Anaconda with Python 3.7

3.5 Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD, Lenovo A13-4089Model.

3.6 Learning Objectives:

Learn How to Apply KNN Classification for Classify Positive and Negative Points in given example.

3.7 Outcomes:

After completion of this assignment students are able Implement code for KNN Classification for Classify Positive and Negative Points in given example also Extend the same example for Distance-Weighted k-NN and locally weighted Averaging

3.8 Theory Concepts:

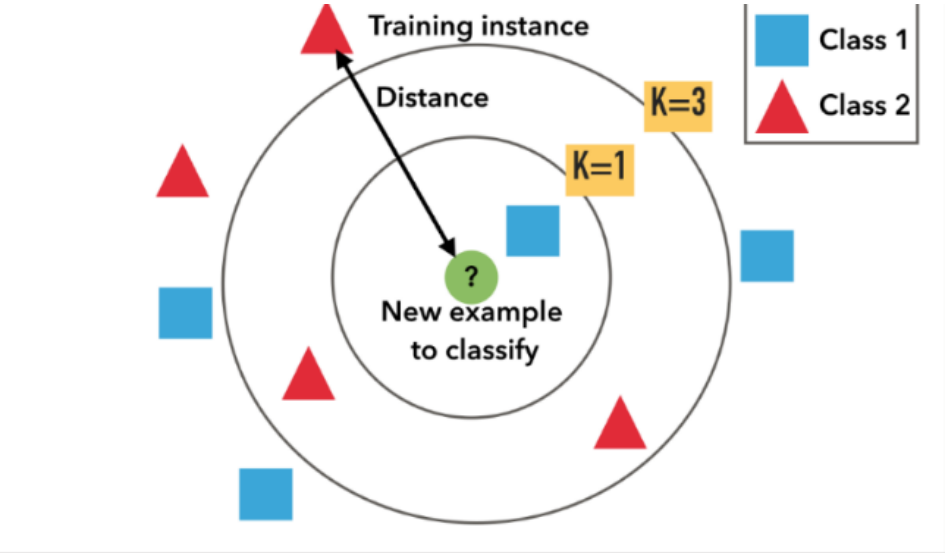
3.8.1 Motivation

K-Nearest Neighbors (kNN) Algorithm-
KNN is an *non parametric lazy learning* algorithm. That is a pretty concise statement. When you say a technique is non parametric , it means that it does not make any assumptions on the underlying data distribution. This is pretty useful , as in the real world , most of the practical data does not obey the typical theoretical assumptions made (eg gaussian mixtures, linearly separable etc) . Non parametric algorithms like KNN come to the rescue here.

It is also a lazy algorithm. What this means is that it does not use the training data points to do any *generalization*. In other words, there is *no explicit training phase* or it is very minimal. This means the training phase is pretty fast . Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase. (Well this is an exaggeration, but not far from truth). This is in contrast to other techniques like SVM where you can discard all non support vectors without any problem. Most of the lazy algorithms – especially KNN – makes decision based on the entire training data set (in the best case a subset of them).

The dichotomy is pretty obvious here – There is a non existent or minimal training phase but a costly testing phase. The cost is in terms of both time and memory. More time might be needed as in the worst case, all data points might take point in decision. More memory is needed as we need to store all training data.

KNN Algorithm is based on **feature similarity**: How closely out-of-sample features resemble our training set determines how we classify a given data point:



Example of k-NN classification. The test sample (inside circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If, for example k = 5 it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle).

KNN can be used for **classification**—the output is a class membership (predicts a class—a discrete value). An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. It can also be used for **regression**—output is the value for the object (predicts continuous values). This value is the average (or median) of the values of its k nearest neighbors.

Assumptions in KNN

Before using KNN, let us revisit some of the assumptions in KNN.

KNN assumes that the data is in a *feature space*. More exactly, the data points are in a metric space. The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance – This need not necessarily be Euclidean distance although it is the one commonly used.

Each of the training data consists of a set of vectors and class label associated with each vector. In the simplest case , it will be either + or – (for positive or negative classes). But KNN , can work equally well with arbitrary number of classes.

We are also given a single number "k" . This number decides how many neighbors (where neighbors is defined based on the distance metric) influence the classification. This is usually a odd number if the number of classes is 2. If k=1 , then the algorithm is simply called the nearest neighbor algorithm.

KNN for Classification

Lets see how to use KNN for classification. In this case, we are given some data points for training and also a new unlabelled data for testing. Our aim is to find the class label for the new point. The algorithm has different behavior based on k.

Case 1 : k = 1 or Nearest Neighbor Rule

This is the simplest scenario. Let x be the point to be labeled . Find the point closest to x . Let it be y. Now nearest neighbor rule asks to assign the label of y to x. This seems too simplistic and some times even counter intuitive. If you feel that this procedure will result a huge error , you are right – but there is a catch. This reasoning holds only when the number of data points is not very large.

If the number of data points is very large, then there is a very high chance that label of x and y are same. An example might help – Lets say you have a (potentially) biased coin. You toss it for 1 million time and you have got head 900,000 times. Then most likely your next call will be head. We can use a similar argument here.

Let me try an informal argument here - Assume all points are in a D dimensional plane . The number of points is reasonably large. This means that the density of the plane at any point is fairly high. In other words , within any subspace there is adequate number of points. Consider a point x in the subspace which also has a lot of neighbors. Now let y be the nearest neighbor. If x and y are sufficiently close, then we can assume that probability that x and y belong to same class is fairly same – Then by decision theory, x and y have the same class.

The book "Pattern Classification" by Duda and Hart has an excellent discussion about this Nearest Neighbor rule. One of their striking results is to obtain a fairly tight error bound to the Nearest Neighbor rule. The bound is

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1}P^*\right)$$

Where P^* is the Bayes error rate, c is the number of classes and P is the error rate of Nearest Neighbor. The result is indeed very striking (atleast to me) because it says that if the number of points is fairly large then the error rate of Nearest Neighbor is less that twice the Bayes error rate. Pretty cool for a simple algorithm like KNN.

Case 2 : k = K or k-Nearest Neighbor Rule

This is a straightforward extension of 1NN. Basically what we do is that we try to find the k nearest neighbor and do a majority voting. Typically k is odd when the number of classes is 2. Lets say k = 5 and there are 3 instances of C1 and 2 instances of C2. In this case , KNN says that new point has to labeled as C1 as it forms the majority. We follow a similar argument when there are multiple classes.

One of the straight forward extension is not to give 1 vote to all the neighbors. A very common thing to do is *weighted kNN* where each point has a weight which is typically calculated using its distance. For eg under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified. This means that neighboring points have a higher vote than the farther points.

It is quite obvious that the accuracy *might* increase when you increase k but the computation cost also increases.

Some Basic Observations

1. If we assume that the points are d-dimensional, then the straight forward implementation of finding k Nearest Neighbor takes O(dn) time.
2. We can think of KNN in two ways – One way is that KNN tries to estimate the posterior probability of the point to be labeled (and apply bayesian decision theory based on the posterior probability). An alternate way is that KNN calculates the decision surface (either implicitly or explicitly) and then uses it to decide on the class of the new points.

3. There are many possible ways to apply weights for KNN – One popular example is the Shephard's method.
4. Even though the naive method takes O(dn) time, it is very hard to do better unless we make some other assumptions. There are some efficient data structures like **KD-Tree** which can reduce the time complexity but they do it at the cost of increased training time and complexity.
5. In KNN, k is usually chosen as an odd number if the number of classes is 2.
6. Choice of k is very critical – A small value of k means that noise will have a higher influence on the result. A large value make it computationally expensive and kinda defeats the basic philosophy behind KNN (that points that are near might have similar densities or classes) .A simple approach to select k is set $k = \sqrt{n}$
7. There are some interesting data structures and algorithms when you apply KNN on graphs – See **Euclidean minimum spanning tree** and **Nearest neighbor graph** .

8. There are also some nice techniques like condensing, search tree and partial distance that try to reduce the time taken to find the k nearest neighbor.

Applications of KNN

KNN is a versatile algorithm and is used in a huge number of fields. Let us take a look at few uncommon and non trivial applications.

1. Nearest Neighbor based Content Retrieval
This is one the fascinating applications of KNN – Basically we can use it in Computer Vision for many cases – You can consider handwriting detection as a rudimentary nearest neighbor problem. The problem becomes more fascinating if the content is a video – given a video find the video closest to the query from the database – Although this looks abstract, it has lot of practical applications – Eg :
Consider **ASL** (American Sign Language) . Here the communication is done using hand gestures.
So lets say if we want to prepare a dictionary for ASL so that user can query it doing a gesture. Now the problem reduces to find the (possibly k) closest gesture(s) stored in the database and show to user. In its heart it is nothing but a KNN problem. One of the professors from my dept , Vassilis Athitsos , does research in this interesting topic – See **Nearest Neighbor Retrieval and Classification** for more details.
2. Gene Expression
This is another cool area where many a time, KNN performs better than other state of the art techniques . In fact a combination of KNN-SVM is one of the most popular techniques there. This is a huge topic on its own and hence I will refrain from talking much more about it.

3. Protein-Protein interaction and 3D structure prediction

Graph based KNN is used in protein interaction prediction. Similarly KNN is used in structure prediction.

4. Credit ratings—collecting financial characteristics vs. comparing people with similar financial features to a database. By the very nature of a credit rating, people who have similar financial details would be given similar credit ratings. Therefore, they would like to be able to use this existing database to predict a new customer's credit rating, without having to perform all the calculations.

5. Should the bank give a loan to an individual? Would an individual default on his or her loan? Is that person closer in characteristics to people who defaulted or did not default on their loans?

6. In political science—classing a potential voter to a "will vote" or "will not vote", or to "vote Democrat" or "vote Republican".

7. More advance examples could include handwriting detection (like OCR), image recognition and even video recognition.

Pros:

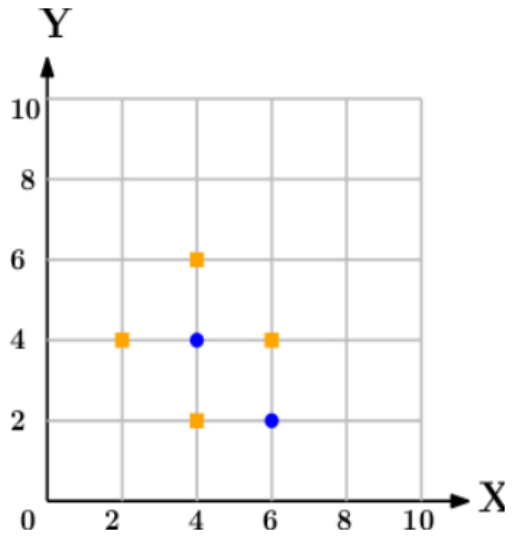
- No assumptions about data—useful, for example, for nonlinear data
- Simple algorithm—to explain and understand/interpret
- High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
- Versatile—useful for classification or regression

Cons:

- Computationally expensive—because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

3.9 Given Diagram Represent Positive and Negative Point with Color

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If k=3, find-the class of the point (6,6). Extend the same example for Distance-Weighted k-NN and Locally weighted Averaging



3.10 Algorithm

1. Import the Required Packages
2. Read Given Dataset
3. Import KNeighborshood Classifier and create object of it.
4. Predict the class for the point(6,6) w.r.t to General KNN.
5. Predict the class for the point(6,6) w.r.t to Distance Weighted KNN.

3.11 Conclusion

In this way we learn KNN Classification to predict the General and Distance Weighted KNN for Given data point in term of Positive or Negative.