DHRUV GARG 16 BCE 1190

K-MEANS CLUSTERING AND KNN USING SKLEARN

LAB 8

K-means clustering theory:

K-means clustering is a type of **unsupervised learning**, which is used when you have unlabelled 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. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on **feature similarity**. The results of the K-means clustering algorithm are:

1. The **centroids** of the K clusters, which can be used to label new data
2. **Labels** for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyse the groups that have formed organically. The cost function is given as follows:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\|x_i - v_j\| \right)^2$$

KNN theory:

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). A label is classified by a **majority vote of its neighbours**, with the label being assigned to the class most common amongst its K nearest neighbours, which are measured by a distance function, often, the **Euclidean distance measure**.

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

In the case of KNN, there is no training phase. The distances are measured from the test point to each and every train-point, and then K nearest neighbours are chosen. For **classification**, a majority vote amongst the nearest neighbours is taken. While for **regression**, the mean value of the K nearest neighbours is taken.

Choosing an optimal value of K:

It is very important for both the algorithms. In KNN, the boundary becomes smoother with increasing value of K. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increases with increasing K. To get the optimal value of K, we use the **Elbow plot**, **Silhouette coefficient** and **GridSearchCV**. This value of K should be used for all predictions.

PART 1: Lab implementation

First, we created 50 sample points in two clusters using **make_blobs**, with 3 centres. To fit our KMeans cluster centroids, we used the KMeans class from sklearn.cluster. The three clusters were well separated and hence they were clustered well. Next, we verified the optimal value of K using the distortion Elbow plot. This also gave an inflection point at K=3. Next, we also used the Silhoutte coefficient to see how tightly or loosely grouped the clusters are. Finally, we also used an online KMeans clustering webpage to understand how the centroids are updated for each iteration and how the initialization of the cluster centres also plays a significant role.

PART 2: ABSENTEEISM FROM WORK Dataset

IMPLEMENTATION 1

For KMeans clustering I took the parameters "Distance from Residence to work" and

"Absenteeism time in hours" to try and obtain **two clusters** (although skewed in terms of the number of samples per group). First I used the init='random' method of **KMeans** from **sklearn.cluster**. It did a pretty bad job as it had many misclassified cluster labels. To improve its performance, I used the **Elbow method** to find the optimal value of K from the distortion values. The **optimal value of K** came out to be 4. This is in contrast to the actual number of clusters = 2. Finally, on implementing the **Silhoutte coefficient** method for finding the optimal K, we get K=9. This stark contrast is due to the fact that the two methods work on two different metrics. One works on **distortion**, while the other works on the **tightness of the clusters**. However, since K=4 is closer to the actual value of K=2, we go with the Elbow method, and plot the cluster patterns. Although its performance is below-par in this example, it can be **attributed to** the fact that the given data itself was skewed.

IMPLEMENTATION 2

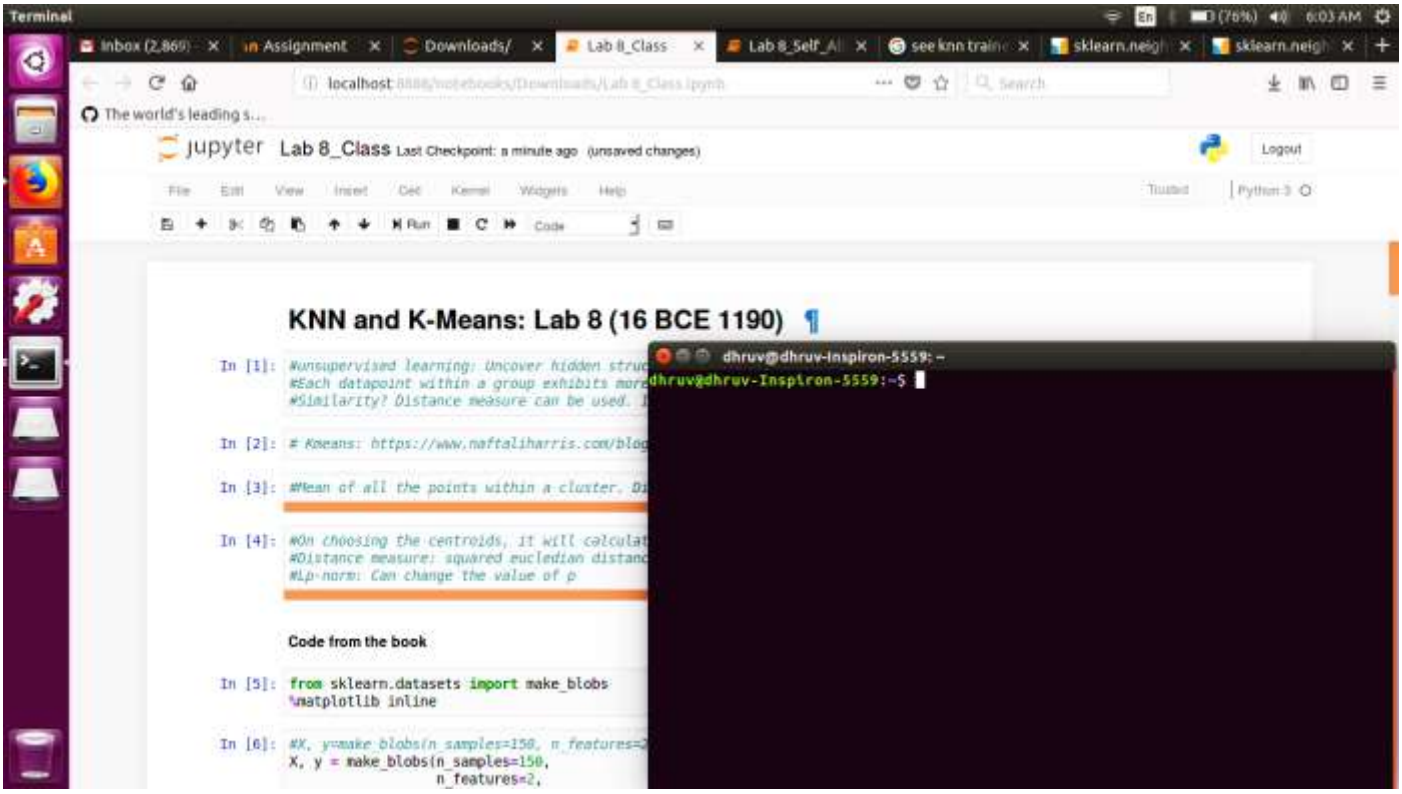
With **KNN**, I implemented KNN Regression **KNeighboursRegressor** from **sklearn.neighbours**. I used the regression model to predict the service time given the age of the employee. This is the same parameter on which I had implemented SVR, so that a comparison is possible. Using the default parameters except the n_neigbours=4, the **mean squared error** gave a very good score (low score = minimal error) of 0.708. This was similar to that obtained using SVR. Next, **tuning** of the **K** parameter was done using the **GridSearchCV** method from **sklearn.model_selection**. The parameters were used to vary the number of neighbours, the weights (distance or uniform), leaf size and p (power parameter). With the best parameters from GridSearchCV, the **minimum mean squared error** reached **0.41**, which is an improvement over the previous 0.70. The GridSearchCV suggested k_neighbours = 10, weights = 'distance' and leaf_size as 30.

Comparison of KNN for regression with SVR:

While **KNN** was thought of as being an ordinary, simple algorithm, it has shown that in many situations, it **performs equally or even better**. In this case it performed at par with **SVR** without the tuning of K. **After tuning**, it **performed better than SVR** which gave a mean squared error of 0.6. With no training phase, KNN saves time. However, a significant amount of time is spent while computing the distance measures.

SCREENSHOTS

Lab implementation



Chosen dataset – Absenteeism at work