Spark Project

Introduction:

Powerful systems are needed to handle huge amounts of data which costs a lot. Distributed systems can prevent the cost of getting such a single high-end machine. Spark can help you achieve that.
Apache Spark has its architectural foundation in the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.
https://en.wikipedia.org/wiki/Apache_Spark

Problem statement:

In this project, we will implement the KNN (***k*-nearest neighbors**) algorithm. We are going to run this algorithm on Iris Dataset. The Iris Flower Dataset involves predicting the flower species given measurements of iris flowers. The given problem is a multiclass classification problem. The number of observations for each class is the same. The dataset has 150 rows with 4 input variables and 1 output variable.

The 4 features are as follows:

- Sepal-Length (cm)
- Sepal-Width (cm)
- Petal-Length (cm)
- Petal-Width (cm)

Prerequisites:

- Download Iris Dataset: https://archive.ics.uci.edu/ml/datasets/iris
    - https://en.wikipedia.org/wiki/Iris_flower_data_set#Use_of_the_data_set
- KNN Algorithm: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
    - For every unknown data sample, calculate its distance with the known samples. From these calculated distances you select the minimum **k** distances. After this you select the most frequent class label from the class labels of the corresponding data samples of the **k** samples you selected. In other words, you pick the **k** known data samples which gives the minimum distance from the unknown data sample and take the label which the majority of **k** known data samples have.

Let’s start:

Apply all the below steps to Iris-dataset:

- Write a function that reads the data using Spark data frames.
- You can retrieve the class column from the train, validation and test sets and store it in separate lists, which will help you in making predictions later.
- Normalize the data:
    - Subtract the mean of the respective column from each cell, and divide that with the standard deviation of that column. Check the below url on how to do that:
    - Mean: https://sparkbyexamples.com/spark/spark-sql-aggregate-functions/#mean
    - Standard Deviation: https://sparkbyexamples.com/spark/spark-sql-aggregate-functions/#stddev
- Randomly divide the data into three sets such that one set has around 60% data sample and the other two have 20% data sample each. Let's call the 60% set as the train set and the 20% set as the validation set and the other 20% as the test set. (Note: This has to be a random division)
- Write a function which calculates the Euclidean Distance between two rows: https://en.wikipedia.org/wiki/Euclidean_distance
    - **Note:** the **class** column is ignored while calculating the distance.
    - **Optional:** You can write this function in such a way that you can calculate the distance of one row with all the other rows at once.
- Fetching **k** nearest neighbor: Randomly guess a value for **k**, let’s say **k = 5**. For each row in the validation data set, repeatedly use your euclidean function to calculate the distance with each row of the train set. Once you have all the distances, select the **k** minimum distances and get the corresponding training data samples for these **k** distances.
- Making a prediction: From the above **k** train samples, get the most frequent **class label.** This class label is the prediction for one validation data sample/row. Repeat the above two points for each data sample in the validation data sample/row.
- Reporting Accuracy: Once you have predicted class labels for all the validation data samples. Compare them with the actual class labels and report the accuracy. Accuracy is defined as the percentage of correctly predicted class labels.

- Optimizing KNN: Tweak the value of **k**. You can try **k** = 10, 15, 20, 25 etc. For each value check calculate the accuracy using the above points and finally pick the value which gives you the max accuracy on the validation set.

- Once you have finalized the value of **k**, run your algorithm one last time on the test set and report its accuracy.