**KNN Algos**

**(**[**https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/**](https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/) **)**

# **Some pros and cons of KNN**

Pros:

* No assumptions about data
* Simple algorithm — easy to understand
* Can be used for classification and regression

Cons:

* High memory requirement — All of the training data must be present in memory in order to calculate the closest K neighbors
* Sensitive to irrelevant features
* Sensitive to the scale of the data since we’re computing the distance to the closest K points

The K Nearest Neighbors algorithm doesn’t require any additional training when new data becomes available. Rather it determines the K closest points according to some distance metric (the samples must reside in memory).

Then, it looks at the target label for each of the neighbors and places the newfound data point into the same category as the majority.

Given that KNN computes distance, it’s imperative that we scale our data. In addition, since KNN disregards the underlying features, it’s our responsibility to filter out any features that are deemed irrelevant.

#### **Pros**

1. It is extremely easy to implement
2. As said earlier, it is [a lazy learning](https://en.wikipedia.org/wiki/Lazy_learning) algorithm and therefore requires no training prior to making real-time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, [linear regression](https://stackabuse.com/linear-regression-in-python-with-scikit-learn/), etc.
3. Since the algorithm requires no training before making predictions, new data can be added seamlessly.
4. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

#### **Cons**

1. The KNN algorithm doesn't work well with high dimensional data because, with a large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
2. The KNN algorithm has a high prediction cost for large datasets. This is because in large datasets the cost of the calculating distance between the new points and each existing point becomes higher.
3. Finally, the KNN algorithm doesn't work well with categorical features since it is difficult to find the distance between dimensions with categorical features.

Note:

* Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated
* Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization.
* For example, the majority of classifiers calculate the distance between two points by the Euclidean distance.
* If one of the features has a broad range of values, the distance will be governed by this particular feature.
* Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

**Example:**

**from sklearn.preprocessing import StandardScaler**

**scaler = StandardScaler()**

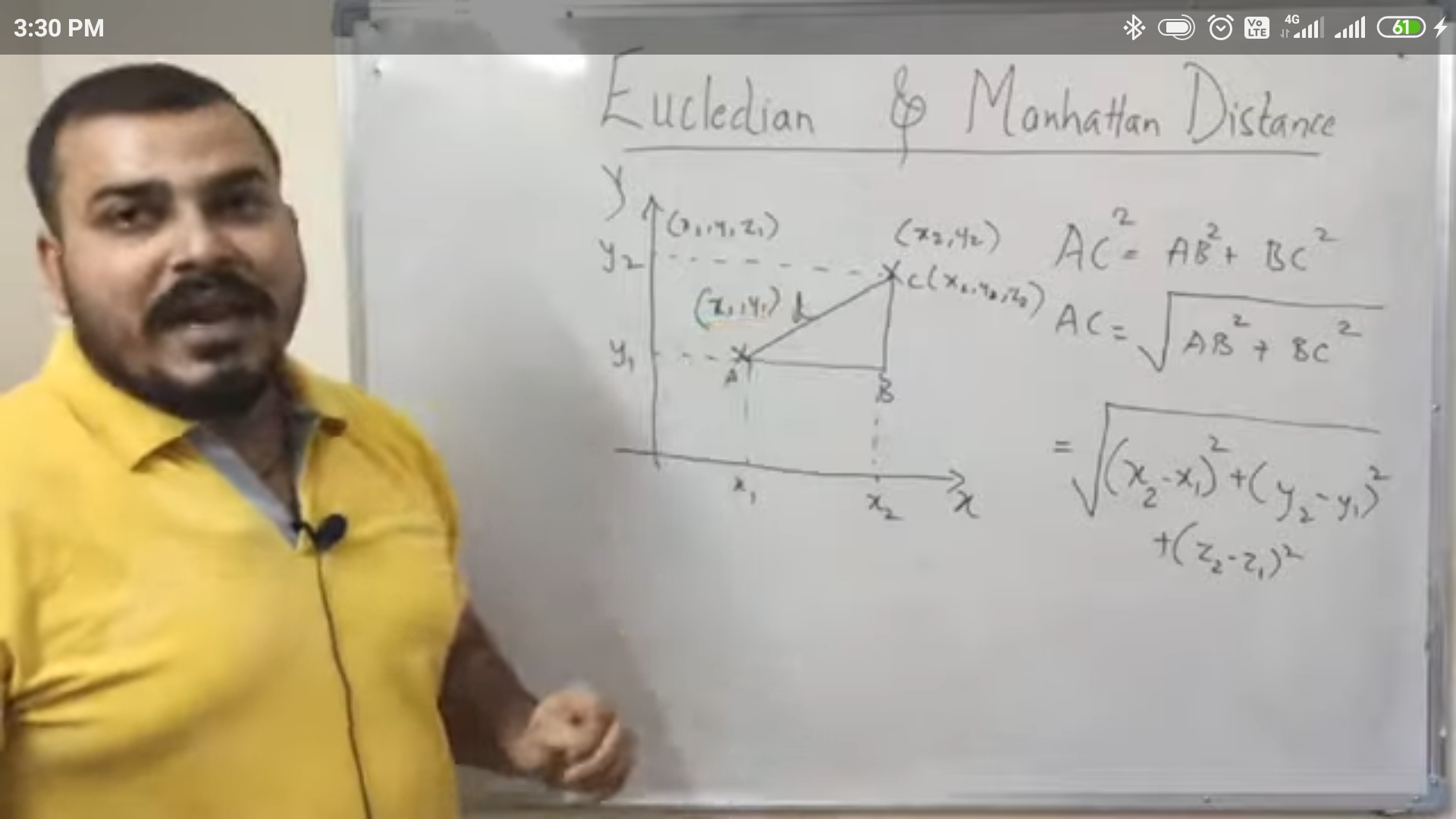
**scaler.fit(X\_train)**

**X\_train = scaler.transform(X\_train)**

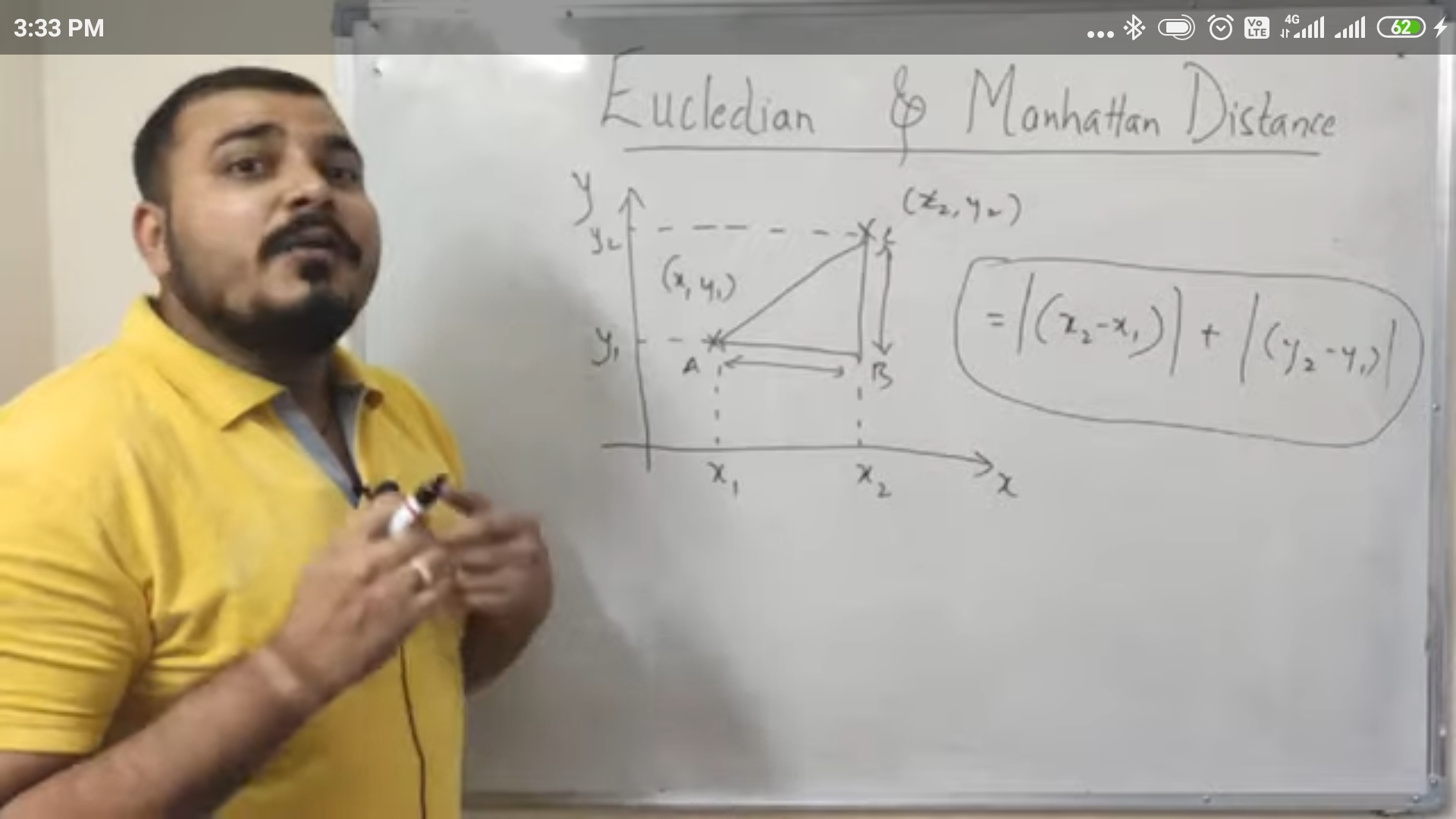
**X\_test = scaler.transform(X\_test)**

**Note:**

* We need to use the **“cross-validation”** technique in order to choose the value of k with the help of accuracy. We need to choose the odd value of k always.
* **Euclidean and Manhattan Distance:**
* **Euclidean Distance:**

****

**-Manhattan Distance:**

****

* **Jupyter Notebook:**

K Nearest Neighbors with Python