**K Nearest Neighbor**

K-Nearest Neighbors, or KNN for short, is one of the simplest machine learning algorithms and is used in a wide array of institutions. KNN is a **non-parametric, lazy**learning algorithm. When we say a technique is non-parametric, it means that it does not make any assumptions about the underlying data. In other words, it makes its selection based on proximity to other data points regardless of what feature the numerical values represent. Being a lazylearning algorithmimplies that there is little to no training phase. Therefore, we can immediately classify new data points as they present themselves.

# Some pros and cons of KNN

**Pros**:

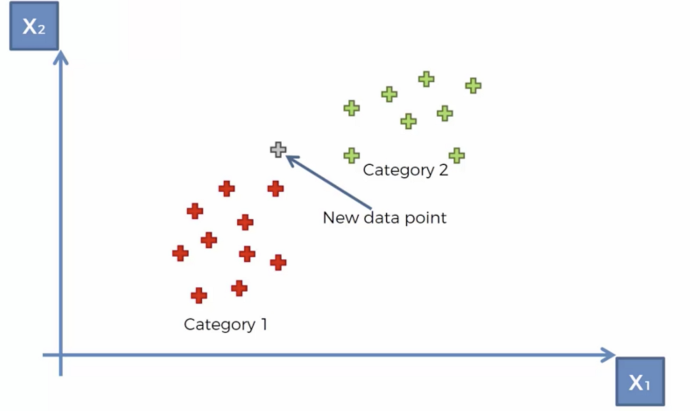
* No assumptions about data
* Simple algorithm — easy to understand
* Can be used for classification and regression

**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

# Algorithm

1. Pick a value for **K**(i.e. 5).



2. Take the **K** nearest neighbors of the new data point according to their Euclidean distance.



In mathematics, the Euclidean distance between two points in Euclidean space is **the length of a line segment between the two points**. It can be calculated from the Cartesian coordinates of the points using the Pythagorean theorem, therefore occasionally being called the Pythagorean distance.

3. Among these neighbors, count the number of data points in each category and assign the new data point to the category where you counted the most neighbors.



## **Get the data**

The dataset contains **569 rows** and **32 columns** of tumor shape and specifications. The tumor is classified as benign or malignant based on its geometry and shape. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass, which is type of biopsy procedure. They describe characteristics of the cell nuclei present in the image.

The features of the dataset include:

1. tumor radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter² / area — 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension

The mean, standard error and “worst” or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

# Code

Let’s take a look at how we could go about classifying data using the K-Nearest Neighbors algorithm in Python. For this case study, we’ll be using the breast cancer dataset from the sklearn.datasets module. We need to start by importing the proceeding libraries.

import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.metrics import confusion\_matrix  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import train\_test\_split  
import seaborn as sns  
sns.set()

#We can set the style by calling Seaborn's set() method

The dataset classifies tumors into two categories (malignant and benign) and contains something like 30 features. In the real world, you’d look at the correlations and select a subset of features that plays the greatest role in determining whether a tumor is malignant or not. However, for the sake of simplicity, we’ll pick a couple at random. We must encode categorical data for it to be interpreted by the model (i.e. malignant = 0 and benign = 1).

breast\_cancer = load\_breast\_cancer()  
X = pd.DataFrame(breast\_cancer.data, columns=breast\_cancer.feature\_names)  
X = X[['mean area', 'mean compactness']]  
y = pd.Categorical.from\_codes(breast\_cancer.target, breast\_cancer.target\_names)  
y = pd.get\_dummies(y, drop\_first=True)

#Make a Categorical type from codes and categories or dtype. This constructor #is useful if you already have codes and categories/dtype and so do not need #the factorization step

As mentioned earlier, the point of building a model, is to classify new data with undefined labels. Therefore, we need to put aside data to verify whether our model does a good job at classifying the data. By default, train\_test\_split sets aside 25% of the samples in the original dataset for testing.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

The sklearn library has provided a layer of abstraction on top of Python. Therefore, in order to make use of the KNN algorithm, it’s sufficient to create an instance of KNeighborsClassifier. By default, the KNeighborsClassifier looks for the **5**nearest neighbors. We must explicitly tell the classifier to use Euclidean distance for determining the proximity between neighboring points.

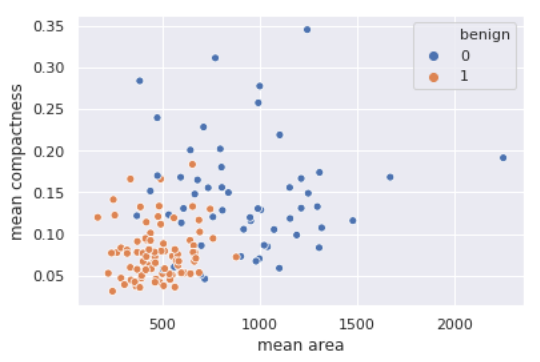
knn = KNeighborsClassifier(n\_neighbors=5, metric='euclidean')  
knn.fit(X\_train, y\_train)

Using our newly trained model, we predict whether a tumor is benign or not given its mean compactness and area.

y\_pred = knn.predict(X\_test)

We visually compare the predictions made by our model with the samples inside the testing set.

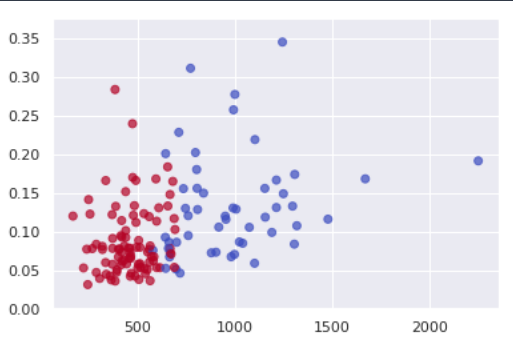
sns.scatterplot(  
 x='mean area',  
 y='mean compactness',  
 hue='benign',  
 data=X\_test.join(y\_test, how='outer')  
)



plt.scatter(  
 X\_test['mean area'],  
 X\_test['mean compactness'],  
 c=y\_pred,  
 cmap='coolwarm',  
 alpha=0.7  
)

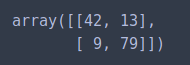
#c : color, sequence, or sequence of color

#Matplotlib allows you to adjust the transparency of a graph plot using the #alpha attribute.By default, alpha=1. If you want to make the graph plot more #transparent, then you can make alpha less than 1, such as 0.5 or 0.25



Another way of evaluating our model is to compute the confusion matrix. The numbers on the diagonal of the confusion matrix correspond to correct predictions whereas the others imply false positives and false negatives.

confusion\_matrix(y\_test, y\_pred)



Given our confusion matrix, our model has an accuracy of 121/143 = 84.6%.

# Conclusion

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 new found 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.