**Crack the Machine Learning Coding Questions**

Review these Algorithms and Their Code Before Your Next Machine Learning Interview

[[Youssef Hosni](https://medium.com/@yousefhosni?source=post_page-----ac8ecc2af90f--------------------------------)](https://medium.com/@yousefhosni?source=post_page-----ac8ecc2af90f--------------------------------)

[[Geek Culture](https://medium.com/geekculture?source=post_page-----ac8ecc2af90f--------------------------------)](https://medium.com/geekculture?source=post_page-----ac8ecc2af90f--------------------------------)

[Youssef Hosni](https://medium.com/@yousefhosni?source=post_page-----ac8ecc2af90f--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F859af34925b7&operation=register&redirect=https%3A%2F%2Fmedium.com%2Fgeekculture%2Fcrack-the-machine-learning-coding-questions-ac8ecc2af90f&user=Youssef+Hosni&userId=859af34925b7&source=post_page-859af34925b7----ac8ecc2af90f---------------------post_header-----------)

Published in

[Geek Culture](https://medium.com/geekculture?source=post_page-----ac8ecc2af90f--------------------------------)

·

17 min read

·

Feb 1, 2023

258

2

Machine learning questions are an essential part of every data science and AI-related position interview. These questions can be categorized into 4 categories. These categories are:

* Machine learning basics
* Resume-Based Machine Learning Questions
* Machine Learning Coding Questions
* Applied Machine Learning Problems

Machine learning coding questions will not only evaluate whether you understand the theory of a certain machine learning algorithm but also whether you are able to code up an algorithm from scratch in a short amount of time or not. Typically the interviewer will ask you to implement a certain machine learning algorithm using an online ide or on a whiteboard during the interview.

This might seem overwhelming because there are so many machine learning algorithms and each has a unique implementation but don’t worry there are only a limited number of algorithms that appear in interviews. Since there are algorithms that are too complicated to implement within one hour and it does not make much sense to test them during interviews which are

* **Supervised learning algorithms:** linear and logistic regression, and the k nearest neighbors.
* **Unsupervised learning algorithms:**the only algorithm that is asked frequently to implement is k means clustering.

In this article, we will go through these algorithms and implement them from scratch so you can review them before your next interview.

**[Types of Machine Learning Questions In A Data Science Interview & How To Prepare for Them](https://medium.com/geekculture/types-of-machine-learning-questions-in-a-data-science-interview-how-to-prepare-for-them-1bc815c9421?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)**

[What To Expect In The Machine Learning Interview](https://medium.com/geekculture/types-of-machine-learning-questions-in-a-data-science-interview-how-to-prepare-for-them-1bc815c9421?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)

[medium.com](https://medium.com/geekculture/types-of-machine-learning-questions-in-a-data-science-interview-how-to-prepare-for-them-1bc815c9421?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)



Photo by [AltumCode](https://unsplash.com/@altumcode?utm_source=medium&utm_medium=referral" \t "_blank) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

**Table of Content:**

1. Linear Regression
2. Logistic Regression
3. K-Nearest Neighbors (KNN)
4. K-Means Clustering

**If you want to study Data Science and Machine Learning for free, check out these resources:**

* Free interactive roadmaps to learn Data Science and Machine Learning by yourself. Start here: <https://aigents.co/learn/roadmaps/intro>
* The search engine for Data Science learning resources (FREE). Bookmark your favorite resources, mark articles as complete, and add study notes. <https://aigents.co/learn>
* Want to learn Data Science from scratch with the support of a mentor and a learning community? Join this Study Circle for free: <https://community.aigents.co/spaces/9010170/>

**If you want to start a career in data science & AI and do not know how. I offer data science mentoring sessions and long-term career mentoring:**

* Long-term mentoring: <https://lnkd.in/dtdUYBrM>
* Mentoring sessions: <https://lnkd.in/dXeg3KPW>

**Join the**[**Medium membership**](https://youssefraafat57.medium.com/membership)**program for only 5$ to continue learning without limits. I’ll receive a small portion of your membership fee if you use the following link, at no extra cost to you.**

**[Join Medium with my referral link - Youssef Hosni](https://youssefraafat57.medium.com/membership?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)**

[Read every story from Youssef Hosni (and thousands of other writers on Medium). Your membership fee directly supports…](https://youssefraafat57.medium.com/membership?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)

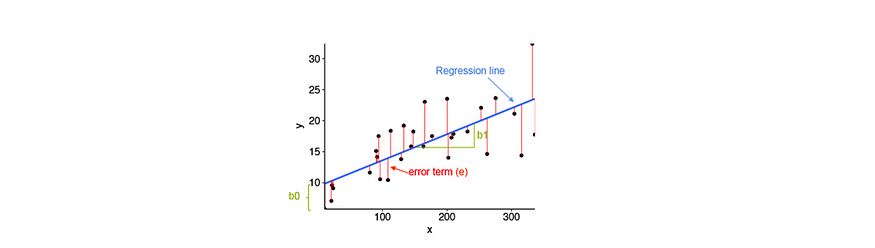
[youssefraafat57.medium.com](https://youssefraafat57.medium.com/membership?source=post_page-----ac8ecc2af90f--------------------------------" \t "_blank)

**1. Linear Regression**

**1.1. Linear Regression Basics**

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It is used to predict the value of the dependent variable based on the values of the independent variables. Linear regression assumes a linear relationship between the variables, and the model is represented by a straight-line equation. Linear regression can be used for both simple and multiple regression analysis. It is widely used in many fields such as economics, social sciences, and natural sciences. Practically linear regression is used due to its simplicity and interoperability. For example, it could be used to predict the demand for certain products.

In linear regression, a linear function is used to model the relationship between the dependent variable (y) and one or more independent variables (x).



The general form of the linear function for simple linear regression is:

* y = b0 + b1\*x

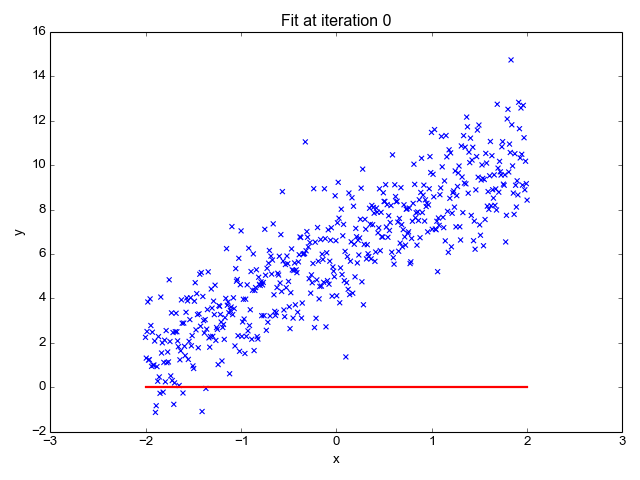
where y is the dependent variable, x is the independent variable, b0 is the y-intercept, and b1 is the slope.

For multiple linear regression, the function is similar but with multiple independent variables:

* y = b0 + b1*x1 + b2*x2 + … + bn\*xn

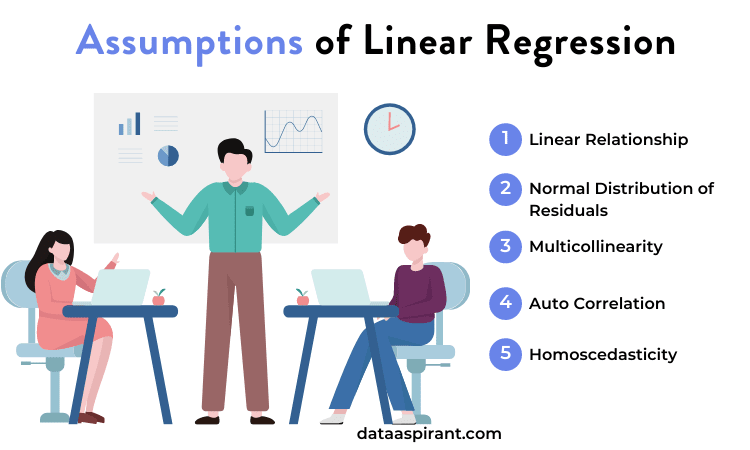
where x1, x2, …, xn are the independent variables, and b1, b2, …, bn are the respective regression coefficients.

The goal of linear regression is to find the values of the coefficients (b0, b1, b2, …, bn) that minimize the difference between the predicted values of y and the actual values of y. This is typically done using the method of least squares, which minimizes the sum of the squares of the differences between the predicted and actual values of y.



**1.2. Linear Regression Assumption**

Linear regression makes several assumptions about the data that are being used to fit the model. These assumptions are important to keep in mind in order to ensure that the model is a good fit for the data and that the results are valid.



Assumption of Linear Regression / Photo by [dataaspirant](https://dataaspirant.com/assumptions-of-linear-regression-algorithm/" \t "_blank)

The main assumptions are:

1. Linearity: Linear regression assumes a linear relationship between the dependent variable and the independent variable(s). This means that the relationship can be represented by a straight line.
2. Independence: Linear regression assumes that the observations are independent of each other. This means that there should not be any correlation among the observations.
3. Homoscedasticity: Linear regression assumes that the variance of the errors (residuals) is constant across the different values of the independent variable(s).
4. Normality: Linear regression assumes that the errors (residuals) are normally distributed.
5. No multicollinearity: Linear regression assumes that the independent variables are not highly correlated with each other.

Violation of these assumptions can lead to inaccurate or unreliable results. It is important to check for these assumptions and address any problems before interpreting the results of a linear regression analysis.

**1.3. Building Linear Regression in Python**

Let's implement the linear regression in python using the code below:

class LinearRegression() :  
   
 def \_\_init\_\_( self, learning\_rate, iterations ) :  
   
 self.learning\_rate = learning\_rate  
   
 self.iterations = iterations  
   
 # Function for model training  
   
 def fit( self, X, Y ) :  
   
 # no\_of\_training\_examples, no\_of\_features  
   
 self.m, self.n = X.shape  
   
 # weight initialization  
   
 self.W = np.zeros( self.n )  
   
 self.b = 0  
   
 self.X = X  
   
 self.Y = Y  
   
   
 # gradient descent learning  
   
 for i in range( self.iterations ) :  
   
 self.update\_weights()  
   
 return self  
   
 # Helper function to update weights in gradient descent  
   
 def update\_weights( self ) :  
   
 Y\_pred = self.predict( self.X )  
   
 # calculate gradients   
   
 dW = - ( 2 \* ( self.X.T ).dot( self.Y - Y\_pred ) ) / self.m  
   
 db = - 2 \* np.sum( self.Y - Y\_pred ) / self.m   
   
 # update weights  
   
 self.W = self.W - self.learning\_rate \* dW  
   
 self.b = self.b - self.learning\_rate \* db  
   
 return self  
   
 # Hypothetical function h( x )   
   
 def predict( self, X ) :  
   
 return X.dot( self.W ) + self.b

The Python class is used for implementing linear regression using gradient descent. The class has several methods:

* \_\_init\_\_(self, learning\_rate, iterations): This is the constructor method that gets called when an object of the class is created. It sets the learning rate and the number of iterations for the gradient descent algorithm.
* fit(self, X, Y): This method is used to train the linear regression model. It takes in the independent variable(s) X and the dependent variable Y as input. It initializes the weights (self.W) and the bias term (self.b) to zero, and the number of training examples (self.m) and the number of features (self.n) to the shape of X. Then, it performs the specified number of iterations of gradient descent by calling the update\_weights() method.
* update\_weights(self): This is a helper method that updates the weights and bias terms in each iteration of gradient descent. It first predicts the dependent variable value (Y\_pred) using the current weights and bias term by calling the predict() method. Then, it calculates the gradients of the cost function with respect to the weights and the bias term. Finally, it updates the weights and bias term by subtracting the product of the learning rate and the gradients.
* predict(self, X): This method is used to make predictions on the new independent variable(s) X. It returns the dot product of X and the weights plus the bias term.

This implementation is for a simple linear regression (one independent variable), if you want to implement it for multiple independent variables you would need to adjust the implementation accordingly. Also, gradient descent is an optimization method used to minimize the loss function, it’s important to note that there are other optimization methods that could be used such as batch gradient descent, mini-batch gradient descent, and stochastic gradient descent.

**2. Logistic Regression**

**2.1. Logistic Regression Basics**

Logistic regression is a type of statistical analysis used for predicting the outcome of a categorical dependent variable based on one or more independent variables. Logistic regression is similar to linear regression, but the output variable is a probability (a value between 0 and 1) rather than a continuous value. The goal of logistic regression is to find the best-fitting model to describe the relationship between the dependent variable and the independent variable(s).

In logistic regression, the probability of a certain class or event existing is modeled using a logistic function. The logistic function, also called the sigmoid function, is an S-shaped curve that maps any input to a value between 0 and 1. The curve is defined by a logistic equation, which has a parameterized set of coefficients (also called weights) that are learned from the data.

The logistic regression algorithm starts by assuming a random set of weights for the independent variables and then uses a method called maximum likelihood estimation to find the set of weights that best explain the data. The maximum likelihood estimation method calculates the likelihood that the observed data belongs to the modeled distribution. The algorithm iteratively adjusts the weights to maximize the likelihood of the data.

Once the model is trained, it can be used to predict the probability of a certain class or event for new data. The predicted probability can then be transformed into a binary prediction (e.g. class 1 or class 0) by applying a threshold value, usually 0.5.

Logistic regression is commonly used in many fields, such as healthcare, finance, and marketing, to analyze data and make predictions about events or outcomes. It can be used for both binary classification (two possible outcomes) and multi-class classification (more than two possible outcomes).

**2.2. Logistic Regression Mathematical equations**

The main equation used in logistic regression is the logistic function, also known as the sigmoid function. It is defined as:

* p(x) = 1 / (1 + e^(-wx))

where x is the input vector, w is the weight vector, and p(x) is the probability of the event of interest. The function maps any input x to a value between 0 and 1.

In logistic regression, the goal is to find the weight vector w that maximizes the likelihood of the observed data. The likelihood is defined as the probability of the data given the parameters (weights) of the model. It is calculated by multiplying the probability of each data point belonging to the modeled distribution.

The logistic regression algorithm uses a method called maximum likelihood estimation to find the best-fitting model. The method is based on the principle of finding the weight vector w that maximizes the likelihood of the data. This can be done by solving the following optimization problem:

* argmax (w) L(w) = ∏ p(x)^y \* (1-p(x))^(1-y)

where y is the binary dependent variable, and p(x) is the probability of y = 1 given x.

The above optimization problem is non-linear and non-convex, so it’s impossible to find an analytical solution. Therefore, the gradient descent algorithm is often used to find the local optimal solution.

In summary, the logistic regression equation is a function of the inputs(x) and weights(w) which is the sigmoid function, and the goal of logistic regression is to find the weight vector w that maximizes the likelihood of the observed data.

**2.3. Building Logistic Regression in Python**

import numpy as np  
  
class LogisticRegression:  
 def \_\_init\_\_(self, learning\_rate=0.01, num\_iterations=100000, fit\_intercept=True):  
 self.learning\_rate = learning\_rate  
 self.num\_iterations = num\_iterations  
 self.fit\_intercept = fit\_intercept  
   
 def sigmoid(self, z):  
 return 1 / (1 + np.exp(-z))  
   
 def fit(self, X, y):  
 if self.fit\_intercept:  
 X = np.c\_[np.ones((X.shape[0], 1)), X]  
   
 self.theta = np.zeros(X.shape[1])  
   
 for i in range(self.num\_iterations):  
 z = np.dot(X, self.theta)  
 h = self.sigmoid(z)  
 gradient = np.dot(X.T, (h - y)) / y.size  
 self.theta -= self.learning\_rate \* gradient  
   
 def predict\_prob(self, X):  
 if self.fit\_intercept:  
 X = np.c\_[np.ones((X.shape[0], 1)), X]  
   
 return self.sigmoid(np.dot(X, self.theta))  
   
 def predict(self, X, threshold=0.5):  
 return self.predict\_prob(X) >= threshold

This is a basic implementation of logistic regression that can handle binary classification problems. The fit method takes in a training set X and target variable y and uses the sigmoid function and gradient descent to find the optimal values for the model's parameters (theta). The predict\_prob and predict methods can then be used to make predictions on new data. The predict method uses a threshold of 0.5 by default, but this can be adjusted as needed.

Here are more details on the methods of the LogisticRegression class:

* \_\_init\_\_(self, learning\_rate=0.01, num\_iterations=100000, fit\_intercept=True): This is the constructor of the class that sets the initial values for the learning rate, number of iterations, and whether to fit an intercept for the model. These can be changed by passing in different values when creating an instance of the class.
* sigmoid(self, z): This method defines the sigmoid function, which is used to map any real-valued number to a value between 0 and 1. This is used in the logistic regression algorithm to model the probability of a certain outcome.
* fit(self, X, y): This method is used to train the logistic regression model on the given dataset, represented by the input variables X and the target variable y. It first adds a column of ones to X if the fit\_intercept attribute is True, which is used to fit an intercept term in the model. Then, it initializes theta (model's parameters) with zeros and updates the values of theta through multiple iterations by taking the dot product of X and theta, passing it through the sigmoid function, and then updating theta based on the gradient of the cost function.
* predict\_prob(self, X): This method is used to predict the probability of a certain outcome for a given input X. It first adds a column of ones to X if the fit\_intercept attribute is True and then returns the predicted probability by passing X and theta through the sigmoid function.
* predict(self, X, threshold=0.5): This method is used to predict the class (0 or 1) for a given input X by thresholding the predicted probability. It first calls the predict\_prob method and then checks if the probability is greater than or equal to the threshold value (0.5 by default). If it is, the class is predicted as 1, otherwise, it is predicted as 0.

This is a basic implementation, it will work for simple problems but it may not work well for more complex problems. Also, this code is not optimized and other optimization techniques like regularization can be added for better performance.

**3. K-Nearest Neighbors (KNN)**

**3.1. KNN Basics**

The k-nearest neighbor (k-NN) algorithm is a type of instance-based learning or non-parametric method used for classification and regression. The basic idea behind the algorithm is to find the k number of closest instances in the training dataset for a new instance and have them “vote” on the class or value of the new instance. The class or value with the most votes is then assigned to the new instance.

The k-NN algorithm is considered a “lazy learning” algorithm because it doesn’t build a model until a new instance is given for classification or regression. This also means that the algorithm doesn’t require any training data in advance.

In k-NN classification, the value of k is typically small, such as k=1 or k=3. A k-value of 1 means that the new instance is assigned to the class of its nearest neighbor. A larger k-value, such as k=3, means that the new instance is assigned to the class that is most common among its 3 nearest neighbors.

In k-NN regression, the value of k is typically set to an odd number to avoid tiebreakers. The value for the new instance is the average of the k nearest neighbors’ values.

One of the main advantages of the k-NN algorithm is its simplicity. It is easy to understand and implement. However, it can be computationally expensive when the dataset is large, and the performance may degrade when the number of features is high.

**3.2. KNN Mathematical Equations**

The k-nearest neighbor (k-NN) algorithm uses a distance metric to measure the similarity between a new instance and the instances in the training dataset. The most commonly used distance metric for k-NN is the Euclidean distance, which is calculated as the square root of the sum of the squared differences between the coordinates of the two instances. The equation for the Euclidean distance between a new instance x and a training instance y is:

d(x,y) = sqrt( (x1 — y1)² + (x2 — y2)² + … + (xd — yd)² )

Where x1, x2, …, xd are the coordinates of the new instance and y1, y2, …, yd are the coordinates of the training instance. The coordinates are also known as features or attributes, which are the variables of the data.

Once the distances are calculated, the algorithm selects the k training instances that are closest to the new instance. In k-NN classification, the new instance is assigned to the class that is most common among the k nearest neighbors. In k-NN regression, the value for the new instance is the average of the k nearest neighbors’ values.

In summary, the k-NN algorithm uses the distance metric to measure the similarity between a new instance and the instances in the training dataset, and it selects the k closest instances to “vote” on the class or value of the new instance.

**3.3. KNN Implementation in Python**

Here is how to implement the k-nearest neighbor (k-NN) algorithm from scratch in Python:

import numpy as np  
  
def euclidean\_distance(x, y):  
 """Calculates the Euclidean distance between two instances"""  
 return np.sqrt(np.sum((x - y) \*\* 2))  
  
def k\_nearest\_neighbors(X\_train, y\_train, x\_test, k):  
 """  
 Finds the k nearest neighbors of a test instance in the training dataset  
 and returns their indices and distances  
 """  
 distances = [euclidean\_distance(x\_test, x) for x in X\_train]  
 nearest\_neighbors = np.argsort(distances)[:k]  
 return nearest\_neighbors, [distances[i] for i in nearest\_neighbors]  
  
def k\_nearest\_neighbors\_classification(X\_train, y\_train, x\_test, k):  
 """  
 Finds the k nearest neighbors of a test instance in the training dataset  
 and returns the class that is most common among them  
 """  
 nearest\_neighbors, \_ = k\_nearest\_neighbors(X\_train, y\_train, x\_test, k)  
 classes = [y\_train[i] for i in nearest\_neighbors]  
 return max(set(classes), key = classes.count)  
  
def k\_nearest\_neighbors\_regression(X\_train, y\_train, x\_test, k):  
 """  
 Finds the k nearest neighbors of a test instance in the training dataset  
 and returns the average value among them  
 """  
 nearest\_neighbors, \_ = k\_nearest\_neighbors(X\_train, y\_train, x\_test, k)  
 return np.mean([y\_train[i] for i in nearest\_neighbors])

This example uses the Euclidean distance as the distance metric and the numpy library to perform the calculations. The function k\_nearest\_neighbors finds the k nearest neighbors of a test instance in the training dataset and returns their indices and distances. The function k\_nearest\_neighbors\_classification finds the k nearest neighbors of a test instance in the training dataset and returns the class that is most common among them. The function k\_nearest\_neighbors\_regression finds the k nearest neighbors of a test instance in the training dataset and returns the average value among them.

Note that this is a very simple implementation of k-NN and it does not include any optimization or handling for missing data, categorical features, and scaling features.

To use these functions, you will need to provide the training dataset X\_train and y\_train and the test instance x\_test and the value of k to use. The functions return the class or value of the test instance, depending on the problem type.

**4. K-Means Clustering**

**4.1. K-Means Clustering Basics**

K-means is a popular unsupervised machine learning algorithm that is used for clustering. Clustering is the process of grouping similar data points together. The goal of K-means is to divide a dataset into K clusters, where each cluster contains similar data points. The number of clusters, K, is a hyperparameter that is specified by the user.

The algorithm works by first initializing K centroids at random positions in the feature space. Each centroid represents the center of a cluster. Then, the algorithm iteratively performs the following steps:

1. Assign each data point to the cluster whose centroid is closest to it. This is done by calculating the distance between the data point and each centroid and assigning the data point to the cluster whose centroid is closest.
2. Recalculate the centroid of each cluster by taking the mean of all data points assigned to that cluster.
3. Repeat steps 1 and 2 until the centroids no longer move or a maximum number of iterations is reached.

The algorithm stops when the centroids stop moving or when a maximum number of iterations is reached. The final clusters are determined by the final positions of the centroids.

One of the main benefits of k-means is its computational efficiency. The algorithm is linear in the number of data points and the number of dimensions. However, a major drawback is that the final clusters may not be optimal for the dataset and the number of clusters, K, must be specified in advance.

**4.2. K-Means Clustering Mathematical Equations**

The mathematical equation for the K-Means Clustering algorithm is as follows:

1. The first step is to initialize K centroids, denoted as C = {c1, c2, …, ck}. These centroids are chosen randomly from the dataset.
2. The next step is to assign each data point x\_i to the closest centroid. This is done by calculating the distance between the data point x\_i and each centroid c\_j. The data point is assigned to the cluster whose centroid is closest. Mathematically, this can be represented as:

c(i) = argmin(||x\_i - c\_j||^2) for j = 1, 2, ..., k

3. Once all data points have been assigned to a cluster, the centroids are recalculated by taking the mean of all data points assigned to that cluster. Mathematically, this can be represented as:

c\_j = (1/n\_j) \* sum(x\_i for i in c\_j) for j = 1, 2, ..., k

Where n\_j is the number of data points in cluster j.

4. Steps 2 and 3 are repeated until the centroids no longer move or a maximum number of iterations is reached.

In this equation ||x\_i — c\_j||² is the square of Euclidean distance between the data point and centroid.

It is important to note that the K-Means algorithm is sensitive to the initial positions of the centroids and can sometimes converge to a local optimum instead of the global optimum. To overcome this issue, the algorithm is often run multiple times with different initial centroids and the best solution is chosen.

**4.3. K-Means Clustering Implementation in Python**

Here is how to implementation of the k-nearest neighbors (KNN) algorithm in Python:

import numpy as np  
  
def euclidean\_distance(x1, x2):  
 """  
 Calculates the euclidean distance between two data points  
 """  
 return np.sqrt(np.sum((x1 - x2)\*\*2))  
  
class KNN:  
 def \_\_init\_\_(self, k=5):  
 self.k = k  
  
 def fit(self, X, y):  
 """  
 X : array-like, shape (n\_samples, n\_features)  
 Training data  
 y : array-like, shape (n\_samples,)  
 Target values  
 """  
 self.X\_train = X  
 self.y\_train = y  
  
 def predict(self, X):  
 """  
 X : array-like, shape (n\_samples, n\_features)  
 Test data  
 """  
 y\_pred = []  
 for x in X:  
 distances = [euclidean\_distance(x, x\_train) for x\_train in self.X\_train]  
 # find k nearest neighbors  
 k\_neighbors = np.argsort(distances)[:self.k]  
 # predict the most common class among k nearest neighbors  
 k\_neighbors\_labels = [self.y\_train[i] for i in k\_neighbors]  
 y\_pred.append(max(set(k\_neighbors\_labels), key=k\_neighbors\_labels.count))  
 return np.array(y\_pred)

The above code defines a class KNN, which has two methods: fit() and predict(). The fit() method is used to train the model by storing the training data (X) and target values (y). The predict() method is used to make predictions on new data.

In the predict method, for each point in the input data, we first calculate the euclidean distance between it and each of the points in the training data. Then, we get the indexes of k nearest points and get the labels for these k nearest points. Finally, we return the most common label among these k points as the final predicted label for that point.

It is important to note that this implementation uses the Euclidean distance as a measure of similarity. Other distance measures such as Manhattan distance or Cosine similarity can also be used.