K-Nearest Neighbors

All you need to know about KNN.

Sangeet Aggarwal Jun 8, 2020 · 6 min read

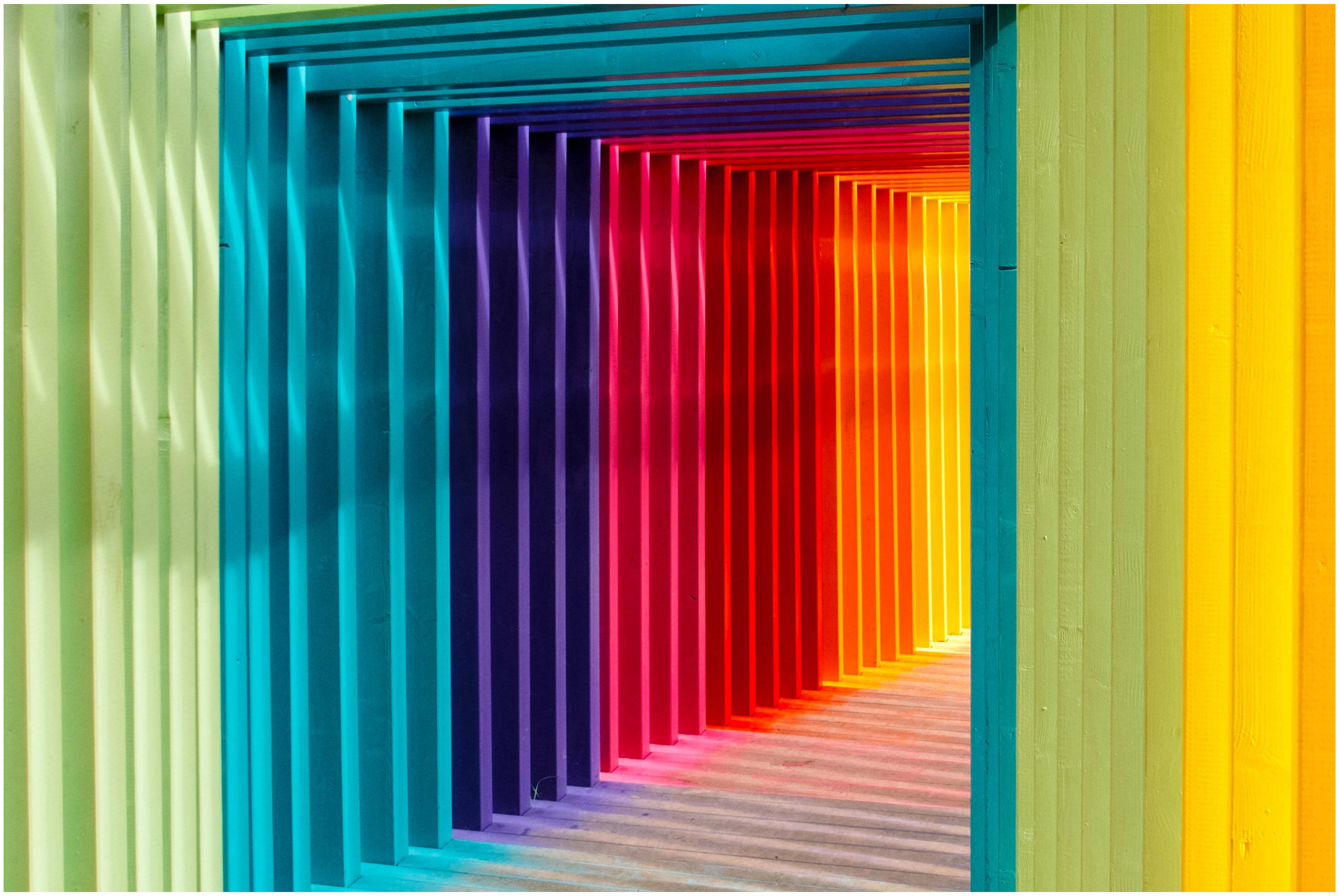


Photo by Robert Katzki on Unsplash

“A man is known for the company he keeps.”

A perfect opening line I must say for presenting the K-Nearest Neighbors. Yes, that's how simple the concept behind KNN is. It just classifies a data point based on its few nearest neighbors. How many neighbors? That is what we decide.

Looks like you already know a lot of there is to know about this simple model. Let’s dive in to have a much closer look.

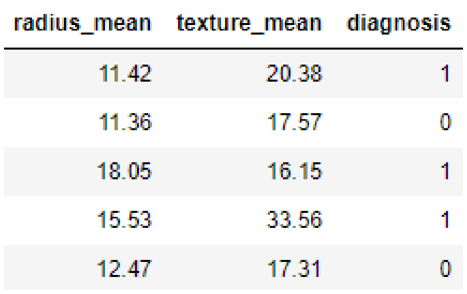
Before moving on, it’s important to know that KNN can be used for both classification and regression problems. We will first understand how it works for a classification problem, thereby making it easier to visualize regression.

KNN Classifier

The data we are going to use is the *Breast Cancer Wisconsin(Diagnostic) Data Set*. There are 30 attributes that correspond to the real-valued features computed for a cell nucleus under consideration. A total of 569 such samples are present in this data, out of which 357 are classified as *‘benign’* (harmless) and the rest 212 are classified as *‘malignant’* (harmful).

The *diagnosis* column contains ‘M’ or ‘B’ values for malignant and benign cancers respectively. I have changed these values to 1 and 0 respectively, for better analysis.

Also, for the sake of this post, I will only use two attributes from the data → ‘mean radius’ and ‘mean texture’. This will later help us visualize the decision boundaries drawn by KNN. Here’s how the final data looks like (after shuffling):



radius_mean	texture_mean	diagnosis
11.42	20.38	1
11.36	17.57	0
18.05	16.15	1
15.53	33.56	1
12.47	17.31	0

Let's code the KNN:

```
# Defining X and y
X = data.drop('diagnosis',axis=1)
y = data.diagnosis

# Splitting data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.25,random_state=42)

# Importing and fitting KNN classifier for k=3
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

# Predicting results using Test data set
pred = knn.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(pred,y_test)
```

The above code should give you the following output with a slight variation.

```
0.8601398601398601
```

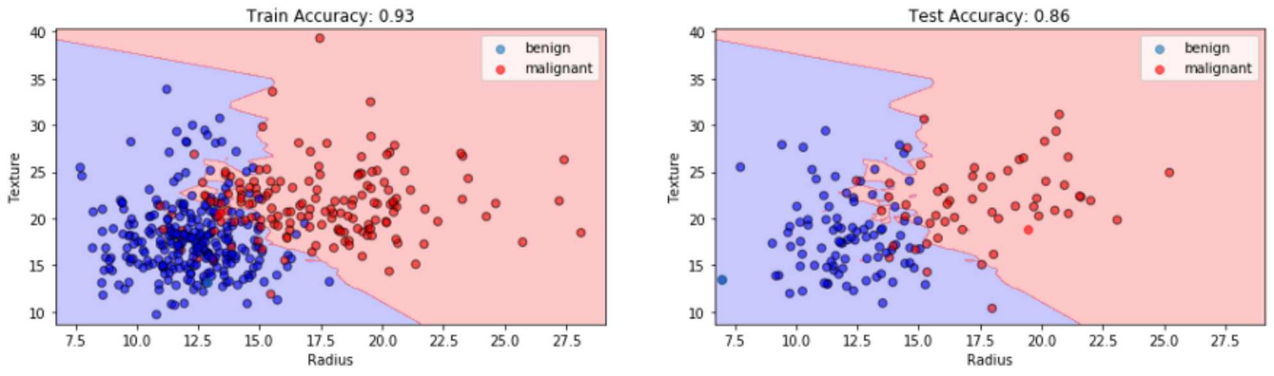
What just happened? When we trained the KNN on training data, it took the following steps for each data sample:

1. Calculate the distance between the data sample and every other sample

with the help of a method such as Euclidean.

2. Sort these values of distances in ascending order.
3. Choose the top K values from the sorted distances.
4. Assign the class to the sample based on the most frequent class in the above K values.

Let's visualize how KNN drew a decision boundary on the train data set and how the same boundary is then used to classify the test data set.



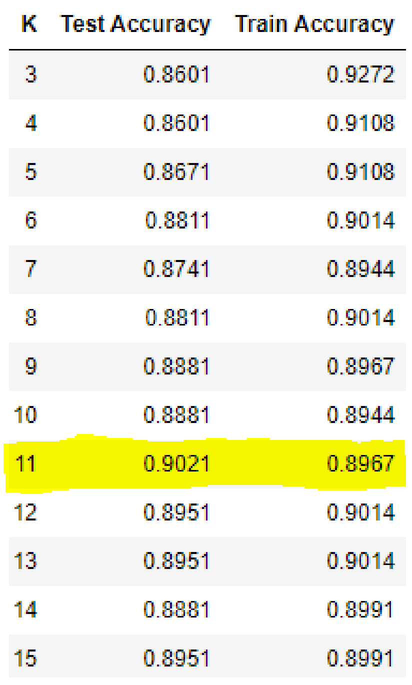
KNN Classification at K=3. Image by Sangeet Aggarwal

With the training accuracy of 93% and the test accuracy of 86%, our model might have shown overfitting here. Why so?

When the value of K or the number of neighbors is too low, the model picks only the values that are closest to the data sample, thus forming a very complex decision boundary as shown above. Such a model fails to generalize well on the test data set, thereby showing poor results.

The problem can be solved by tuning the value of *n_neighbors* parameter. *As we increase the number of neighbors, the model starts to generalize well, but increasing the value too much would again drop the performance.*

Therefore, it's important to find an optimal value of K, such that the model is able to classify well on the test data set. Let's observe the train and test accuracies as we increase the number of neighbors.



K	Test Accuracy	Train Accuracy
3	0.8601	0.9272
4	0.8601	0.9108
5	0.8671	0.9108
6	0.8811	0.9014
7	0.8741	0.8944
8	0.8811	0.9014
9	0.8881	0.8967
10	0.8881	0.8944
11	0.9021	0.8967
12	0.8951	0.9014
13	0.8951	0.9014
14	0.8881	0.8991
15	0.8951	0.8991

Best results are observed at k=11

The above result can be best visualized by the following plot.

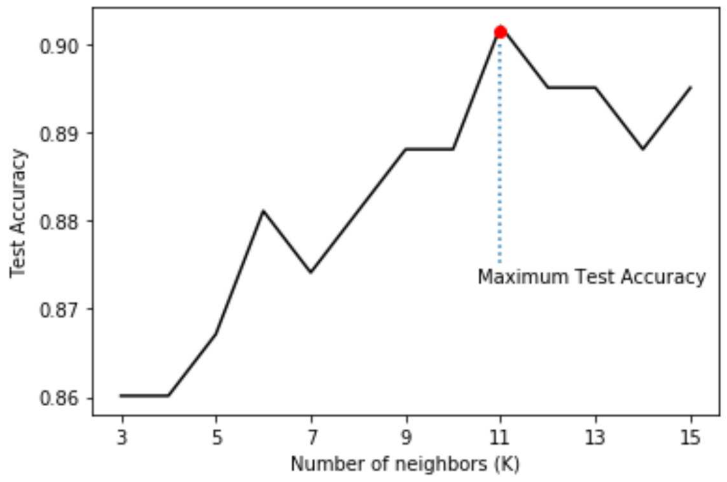
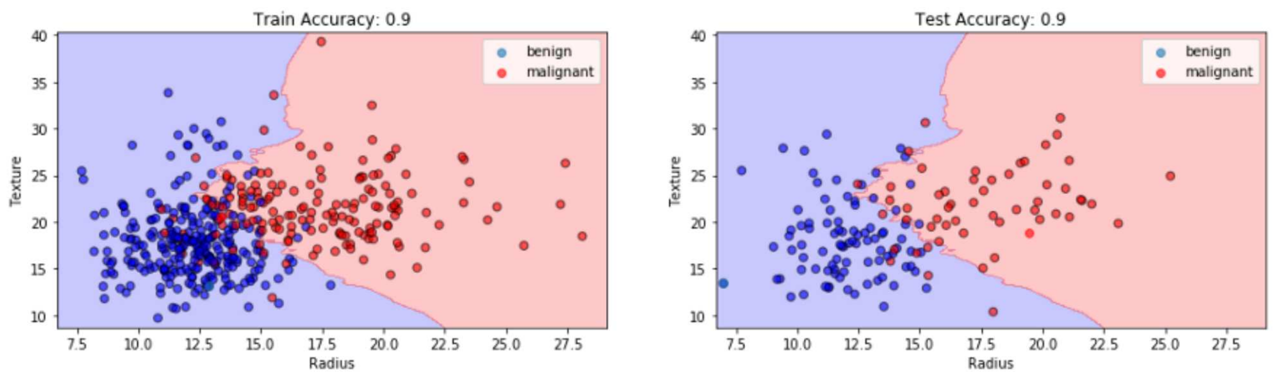


Image by Sangeet Aggarwal

The plot shows an overall upward trend in test accuracy up to a point, after which the accuracy starts declining again. This is the optimal number of nearest neighbors, which in this case is 11, with a test accuracy of 90%.

Let's plot the decision boundary again for k=11, and see how it looks.



KNN Classification at K=11. Image by Sangeet Aggarwal

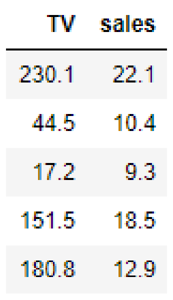
We have improved the results by fine-tuning the number of neighbors. Also, the decision boundary by KNN now is much smoother and is able to generalize well on test data.

Let's now understand how KNN is used for regression.

KNN Regressor

While the KNN classifier returns the mode of the nearest K neighbors, the KNN regressor returns the mean of the nearest K neighbors.

We will use advertising data to understand KNN's regression. Here are the first few rows of TV budget and sales.

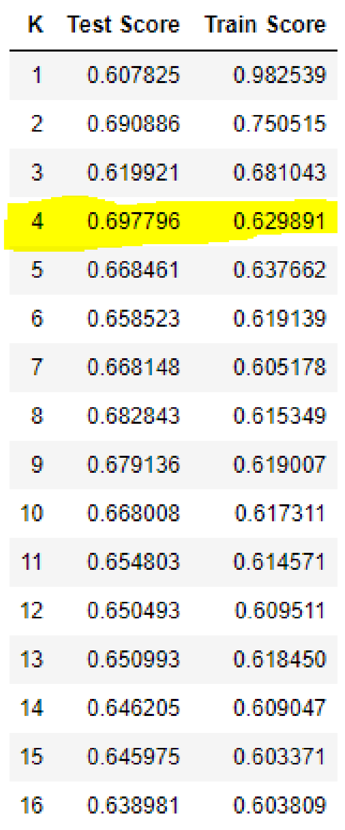


TV	sales
230.1	22.1
44.5	10.4
17.2	9.3
151.5	18.5
180.8	12.9

```
# Defining X and Y
X_ad = ad.TV.values.reshape(-1,1)
y_ad = ad.sales


# Splitting data into train and test
```

```
train_x, test_x, train_y, test_y = train_test_split(X_ad, y_ad,
test_size=0.25, random_state=42)

# Running KNN for various values of n_neighbors and storing results
knn_r_acc = []

for i in range(1,17,1):
    knn = KNeighborsRegressor(n_neighbors=i)
    knn.fit(train_x,train_y)

    test_score = knn.score(test_x,test_y)
    train_score = knn.score(train_x,train_y)

    knn_r_acc.append((i, test_score ,train_score))

df = pd.DataFrame(knn_r_acc, columns=['K','Test Score','Train
Score'])
print(df)
```

The above code will run KNN for various values of K (from 1 to 16) and store the train and test scores in a Dataframe. Let's see how these scores vary as we increase the value of *n_neighbors (or K)*.

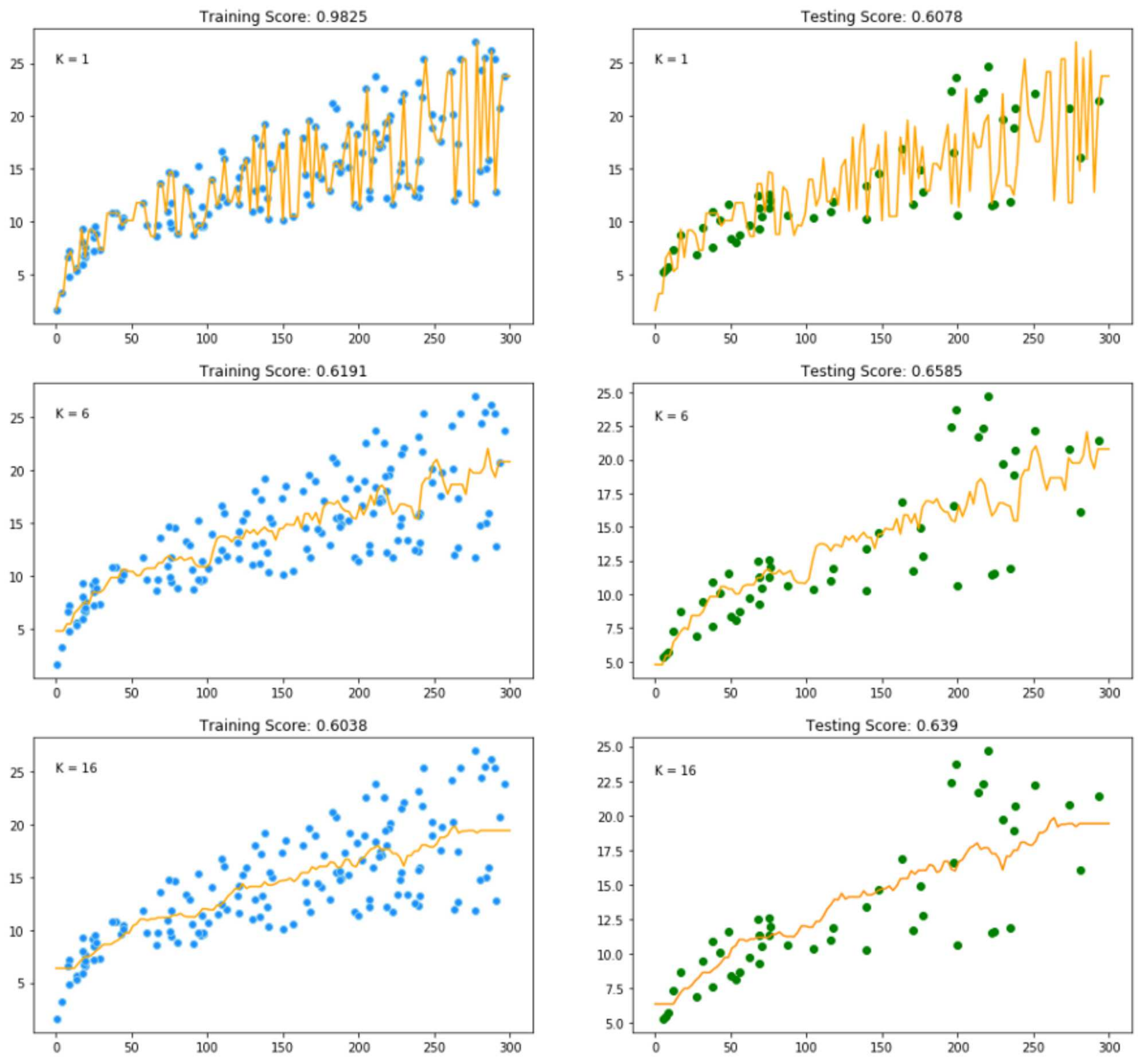


K	Test Score	Train Score
1	0.607825	0.982539
2	0.690886	0.750515
3	0.619921	0.681043
4	0.697796	0.629891
5	0.668461	0.637662
6	0.658523	0.619139
7	0.668148	0.605178
8	0.682843	0.615349
9	0.679136	0.619007
10	0.668008	0.617311
11	0.654803	0.614571
12	0.650493	0.609511
13	0.650993	0.618450
14	0.646205	0.609047
15	0.645975	0.603371
16	0.638981	0.603809

Best results at K=4

At K=1, the KNN tends to closely follow the training data and thus shows a high training score. However, in comparison, the test score is quite low, thus indicating overfitting.

Let's visualize how the KNN draws the regression path for different values of K.



Left: Training dataset with KNN regressor **Right:** Testing dataset with same KNN regressors. Image by Sangeet Aggarwal

As K increases, the KNN fits a smoother curve to the data. This is because a higher value of K reduces the edginess by taking more data into account, thus reducing the overall complexity and flexibility of the model.

As we saw earlier, increasing the value of K improves the score to a certain point, after which it again starts dropping. This can be better understood by the following plot.

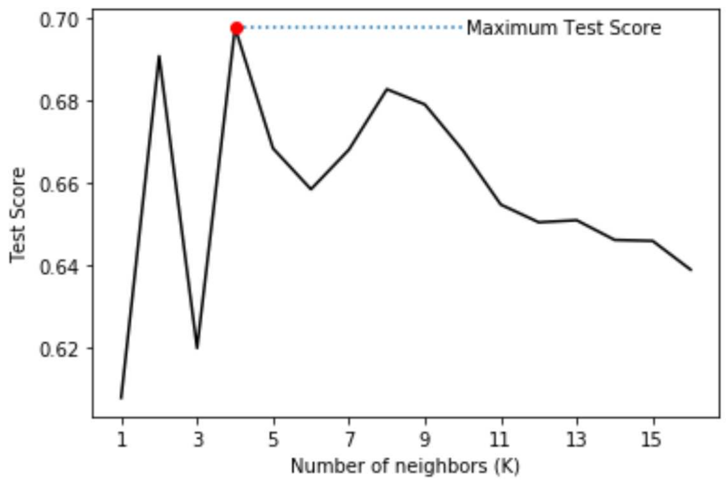


Image by Sangeet Aggarwal

As we see in this figure, the model yields the best results at K=4. I have used R^2 to evaluate the model, and this was the best we could get. This is because our dataset was too small and scattered.

Some other points are important to know about KNN are:

- KNN classifier does not have any specialized training phase as it uses all the training samples for classification and simply stores the results in memory.
- KNN is a non-parametric algorithm because it does not assume anything about the training data. This makes it useful for problems having non-linear data.
- KNN can be computationally expensive both in terms of time and storage, if the data is very large because KNN has to store the training data to work. This is generally not the case with other supervised learning models.
- KNN can be very sensitive to the scale of data as it relies on computing

an be v
efore

stay tur

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Machine Learning Knn Knn Algorithm Data Science Nearest Neighbors