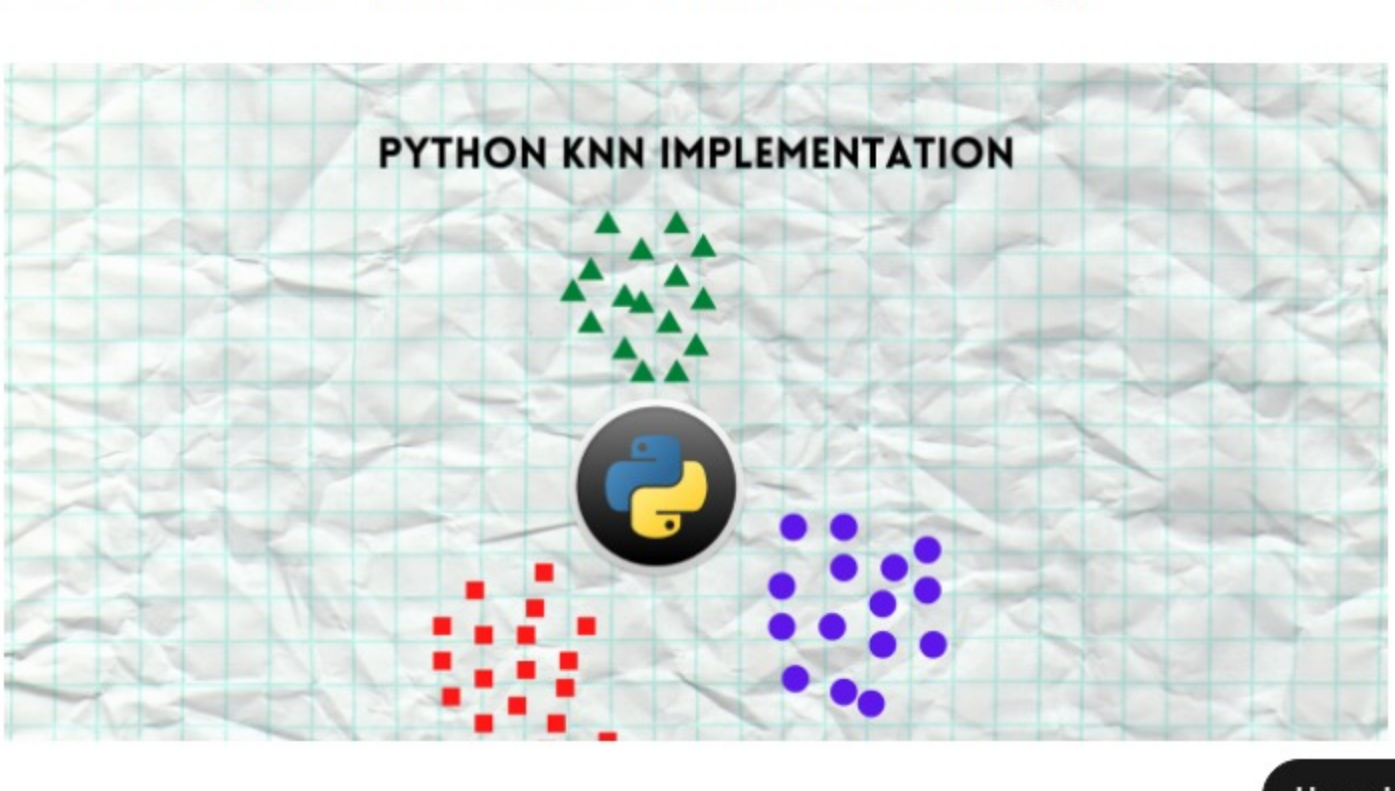
Moosa Ali Follow
Apr 30, 2021 · 4 min read · Listen

KNN- Implementation from scratch (96.6% Accuracy)| Python | Machine Learning

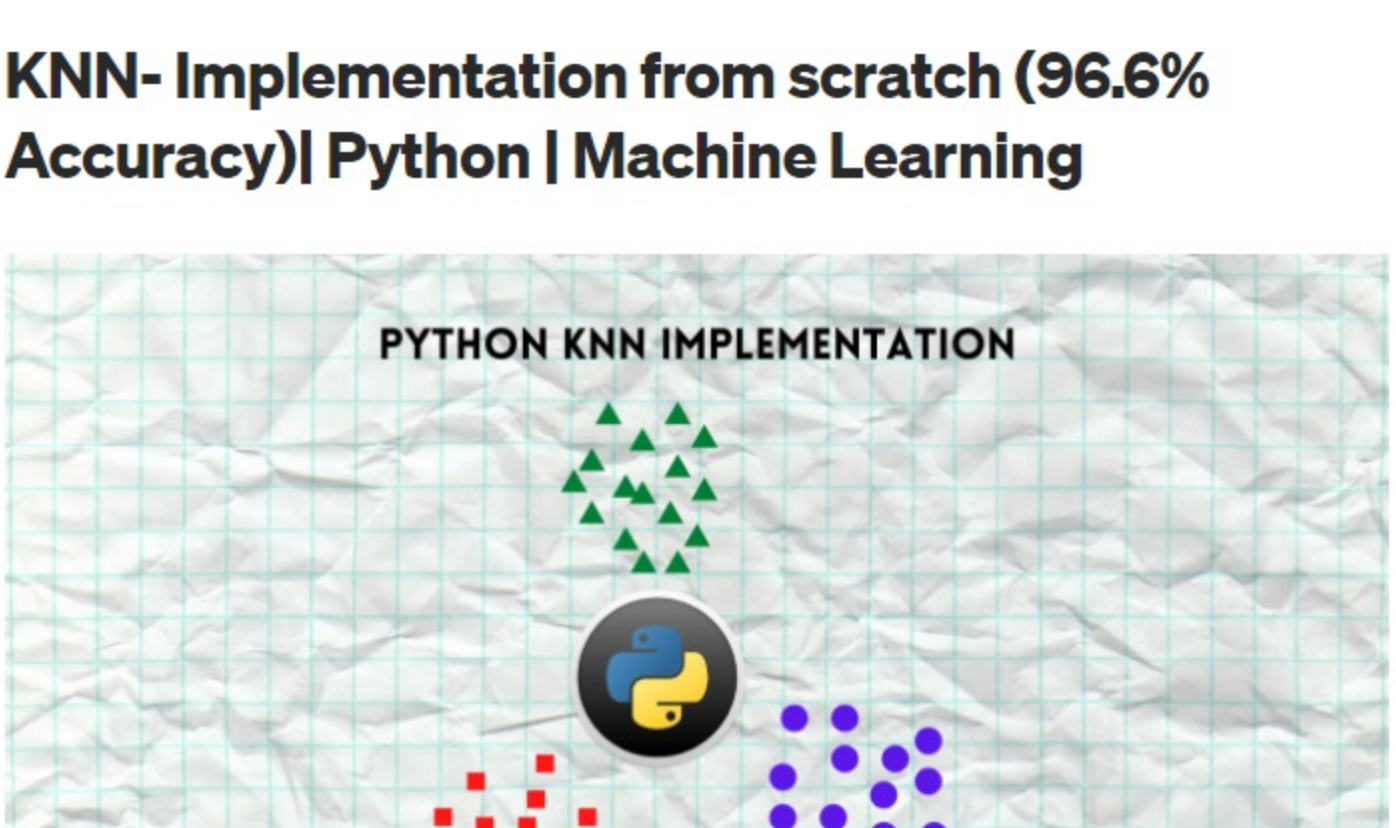


Upgrade Open in app

Published in Analytics Vidhya

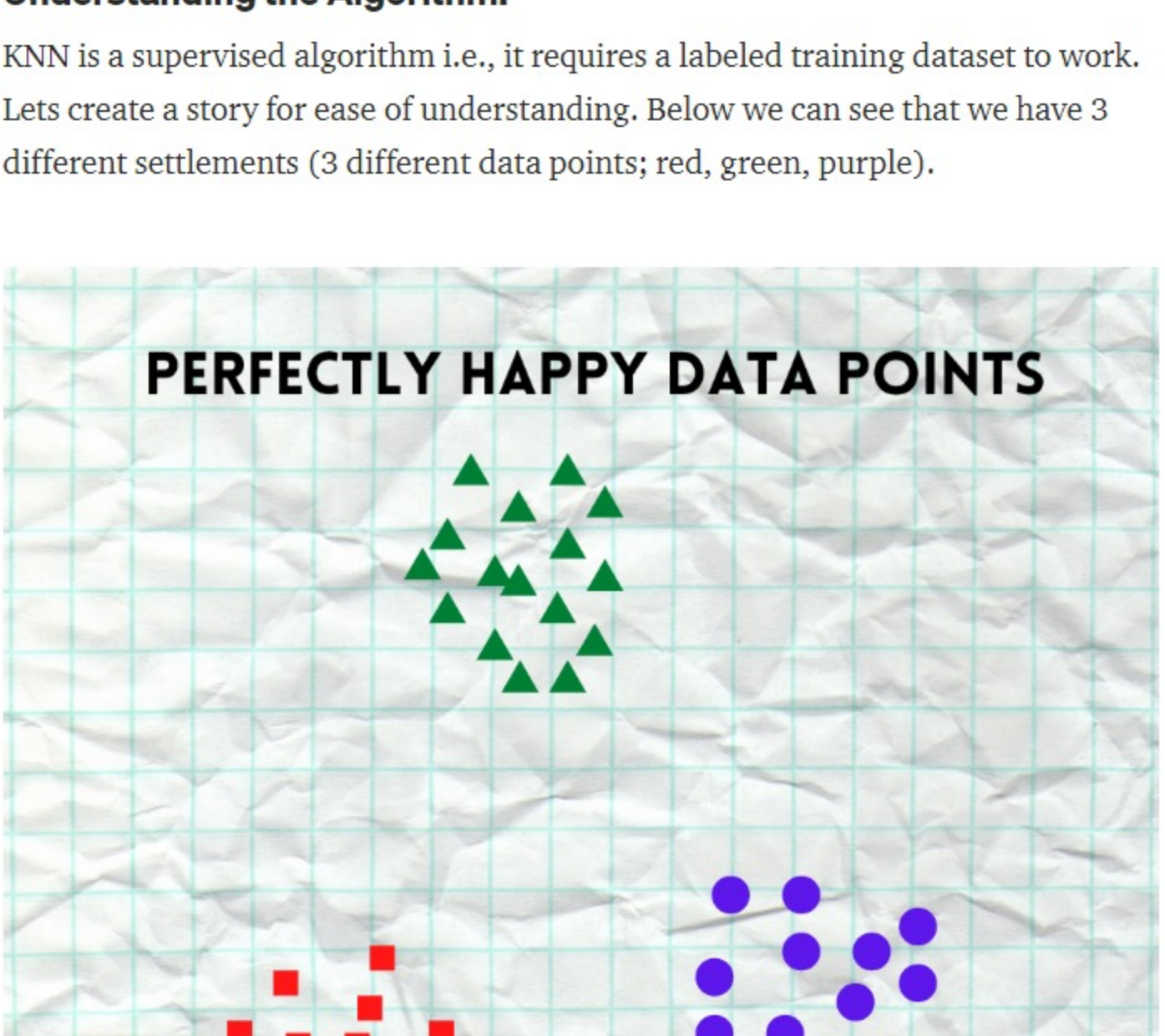
Moosa Ali Follow
Apr 30, 2021 · 4 min read · Listen

KNN- Implementation from scratch (96.6% Accuracy)| Python | Machine Learning



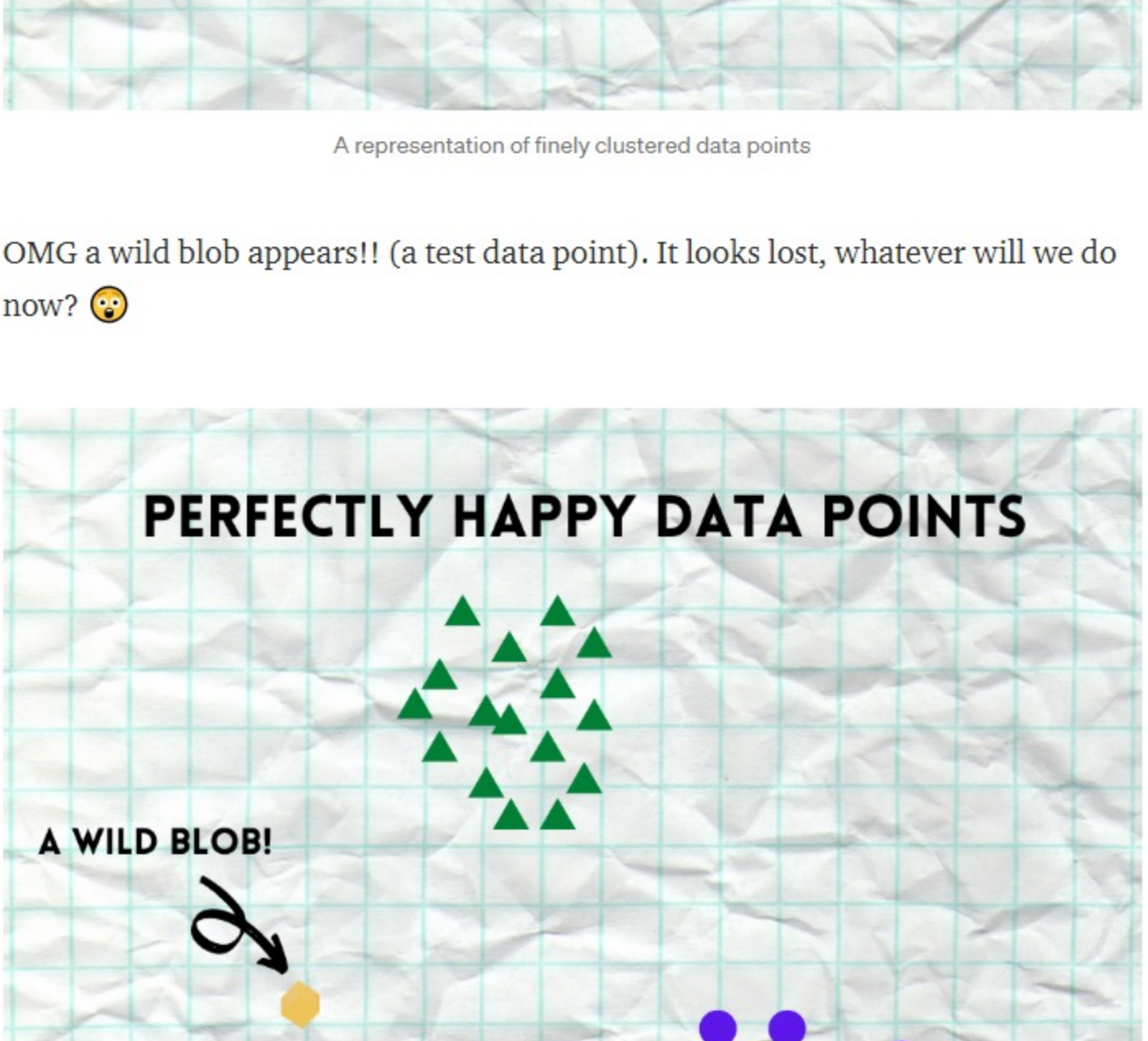
Understanding the Algorithm:

KNN is a supervised algorithm i.e., it requires a labeled training dataset to work. Lets create a story for ease of understanding. Below we can see that we have 3 different settlements (3 different data points; red, green, purple).



A representation of finely clustered data points

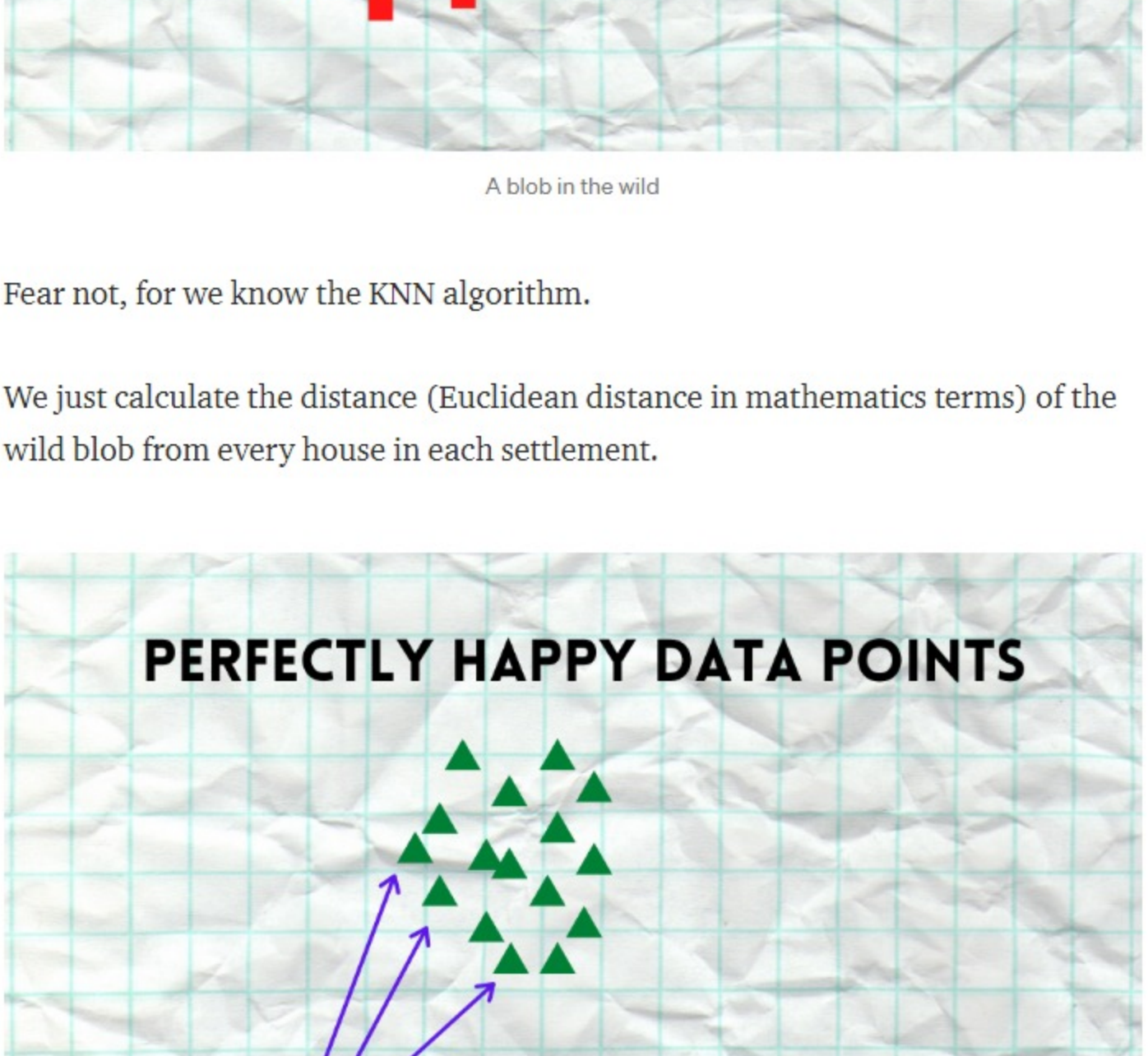
OMG a wild blob appears!! (a test data point). It looks lost, whatever will we do now? 😲



A blob in the wild

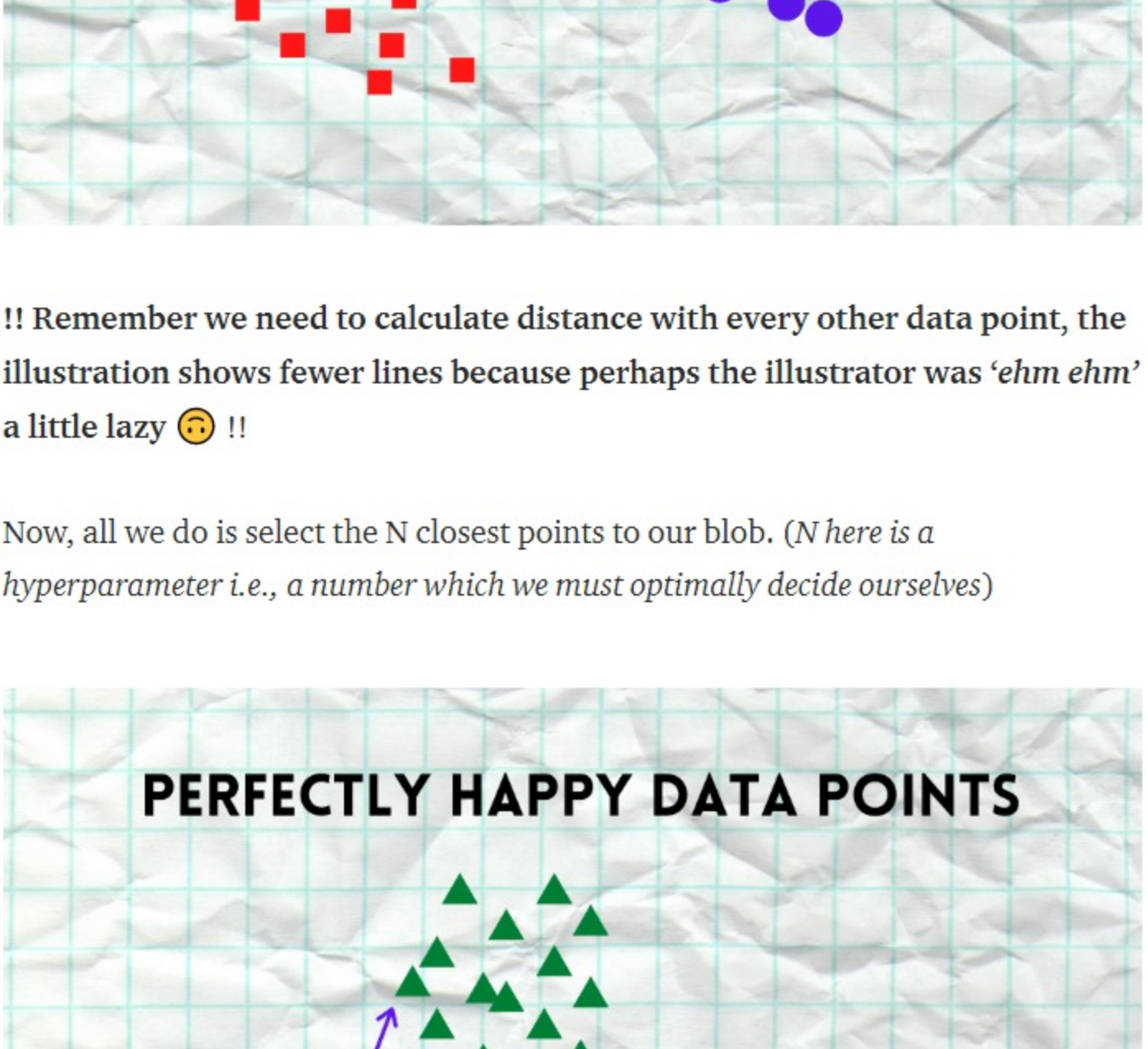
Fear not, for we know the KNN algorithm.

We just calculate the distance (Euclidean distance in mathematics terms) of the wild blob from every house in each settlement.

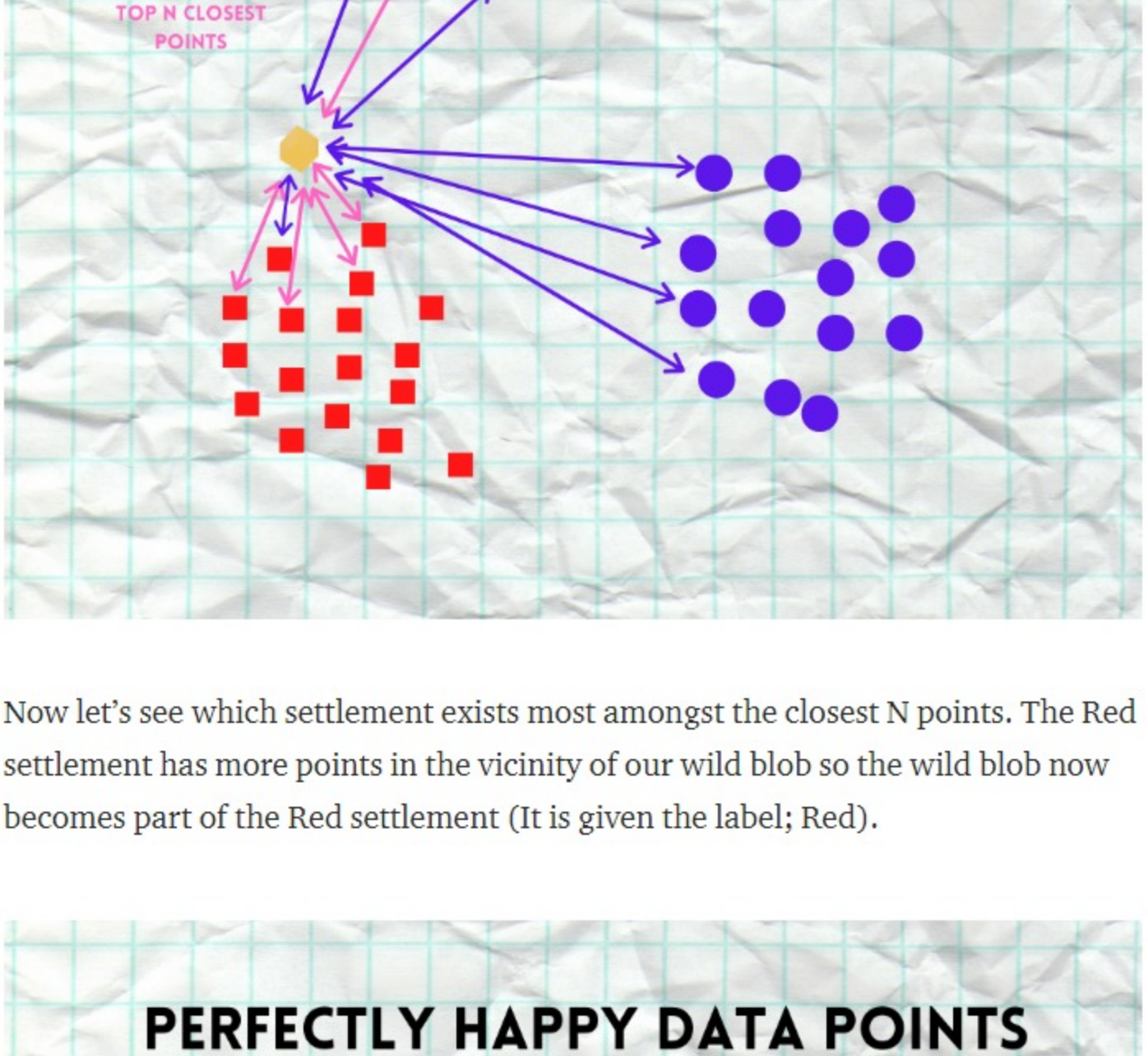


!! Remember we need to calculate distance with every other data point, the illustration shows fewer lines because perhaps the illustrator was *'ehm ehm'* a little lazy 🙃 !!

Now, all we do is select the N closest points to our blob. (*N here is a hyperparameter i.e., a number which we must optimally decide ourselves*)



Now let's see which settlement exists most amongst the closest N points. The Red settlement has more points in the vicinity of our wild blob so the wild blob now becomes part of the Red settlement (It is given the label; Red).



We do this for every test data point. And that is it, that is the algorithm.

Okay story time is over, let's get to coding.

Let us import the necessary libraries.

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Data sample

Convert the output text label to numeric representation.

```
#Seperating the input features and output labels
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

#converting text labels to numeric form
labels, unique = pd.factorize(y)
```

Let's code our simple algorithm.

As always, we need to split our data into test and train samples.

```
#splitting data in test and train segments
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size = 0.40)
```

We have used a 60–40 split for the total data.

With all the steps in place, it's time to test the accuracy.

```
        #return label with majority occurence
        max_idx = np.argmax(counts)
        return values[max_idx]
```

KNN.py hosted with ❤ by GitHub view raw

We have used a helper function in the above code which is below.

```
#Creating a helper function
def takeSecond(elem):
   return elem[1]
```

With all the steps in place, it's time to test the accuracy.

```
#getting predicted values using our algorithm
predictions = list(map(KNNClassify, X_test))
```

```
#calling the accuracy function
accuracy(predictions, y_test)
```

```
Accuracy = 96.66666666666667 %
```

96.67%!! That's a very good number.

Conclusion

This might just be the smallest code we have to write for a machine learning algorithm but it yields an accuracy of 96.67%. Our initial statement stands, ML algorithms don't need to be complex, they can be as simple as the KNN we just learnt in this article. It would be fun to try this algorithm on multiple datasets and see how it performs on those.

Thank you for reading.