# K-Nearest Neighbor Deconstructed

Application of Nearest Neighbor Search

As beginning Python developers and data scientists, we are introduced to the bevy of[[libraries](https://stackoverflow.com/questions/18474784/what-exactly-is-a-python-library) that support our work like [Numpy](https://numpy.org/doc/), [SciPy](https://www.scipy.org/docs.html), [Pandas](https://pandas.pydata.org/docs/), and [SciKit-Learn](https://scikit-learn.org/stable/). These libraries and their packages allow us to fast-track our learning and workflow in parallel. They remove barriers to entry into the technology industry and improve productivity for the beginner and experienced engineer.

However, these out-of-the-box solutions can be challenging to adapt and apply to individual business problems, especially. One issue that only experience can solve is understanding what is happening under the hood when interfacing with a library - what can't we see on the screen.

An excellent way to gain insight is to build your own [algorithm](https://legacy.python.org/workshops/2002-02/papers/15/index.htm) from scratch and then compare it to the library version. This will allow you to deconstruct the algorithm:

* identify what data is and is not in it
* how data structures are manipulated within
* logical sequence
* what mathematical calculations are involved

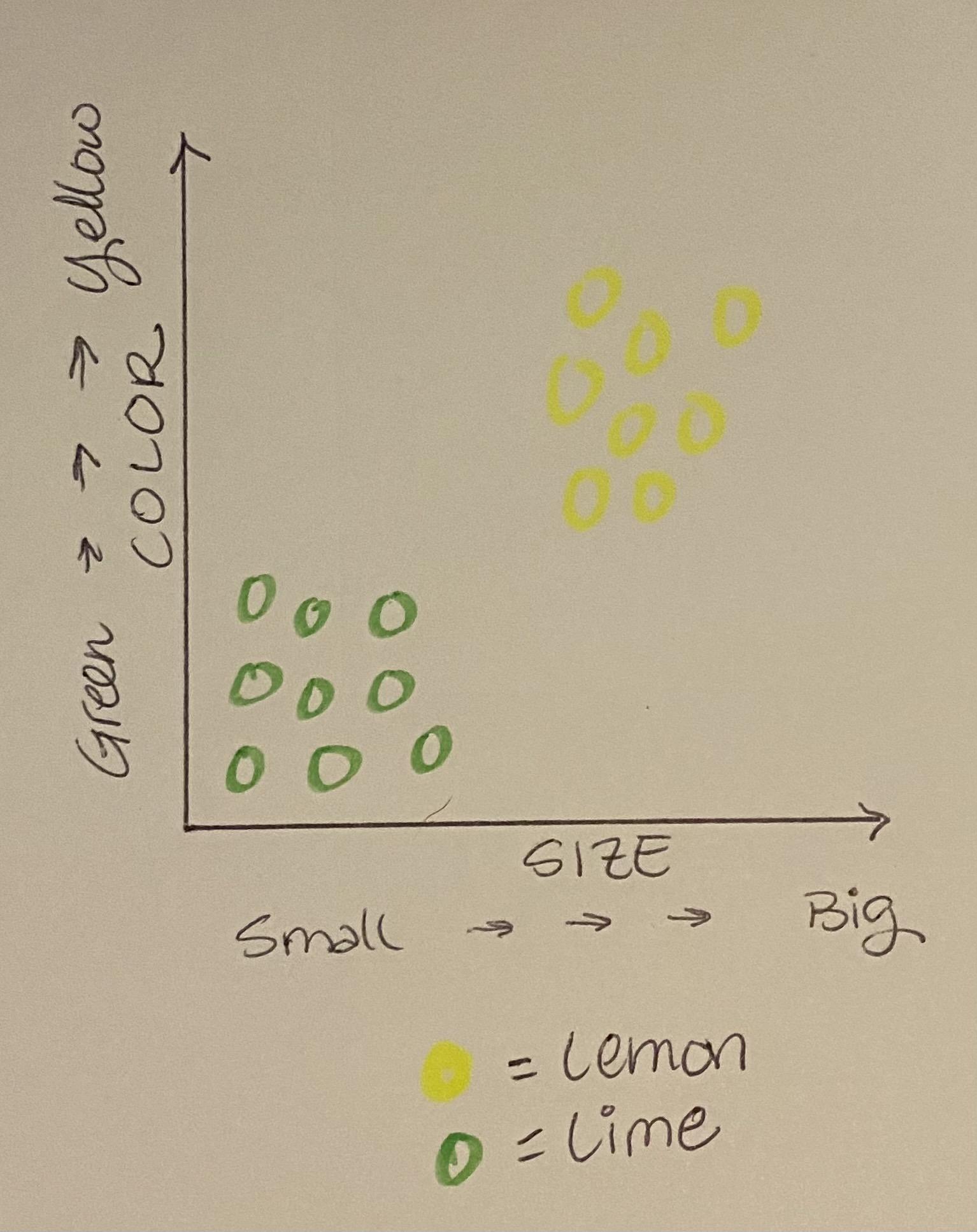
An additional benefit to building your algorithm is that you have complete control over the code! As your understanding of the algorithm, dataset, data structures, and python itself grows during the exercise (...and, it will), you can adjust the algorithm code itself to reflect your optimization of knowledge and skill.

## Build a Classification System Using K-NN

#### Understanding Classification

Imagine you have a mystery fruit in front of you. You think it could either be a lemon or a lime, but your not too sure. You know limes are small, round, and green, while lemons are usually large, oval-ish, and bright yellow. You haven't cut your mystery fruit open, so the taste isn't an observable feature right now.

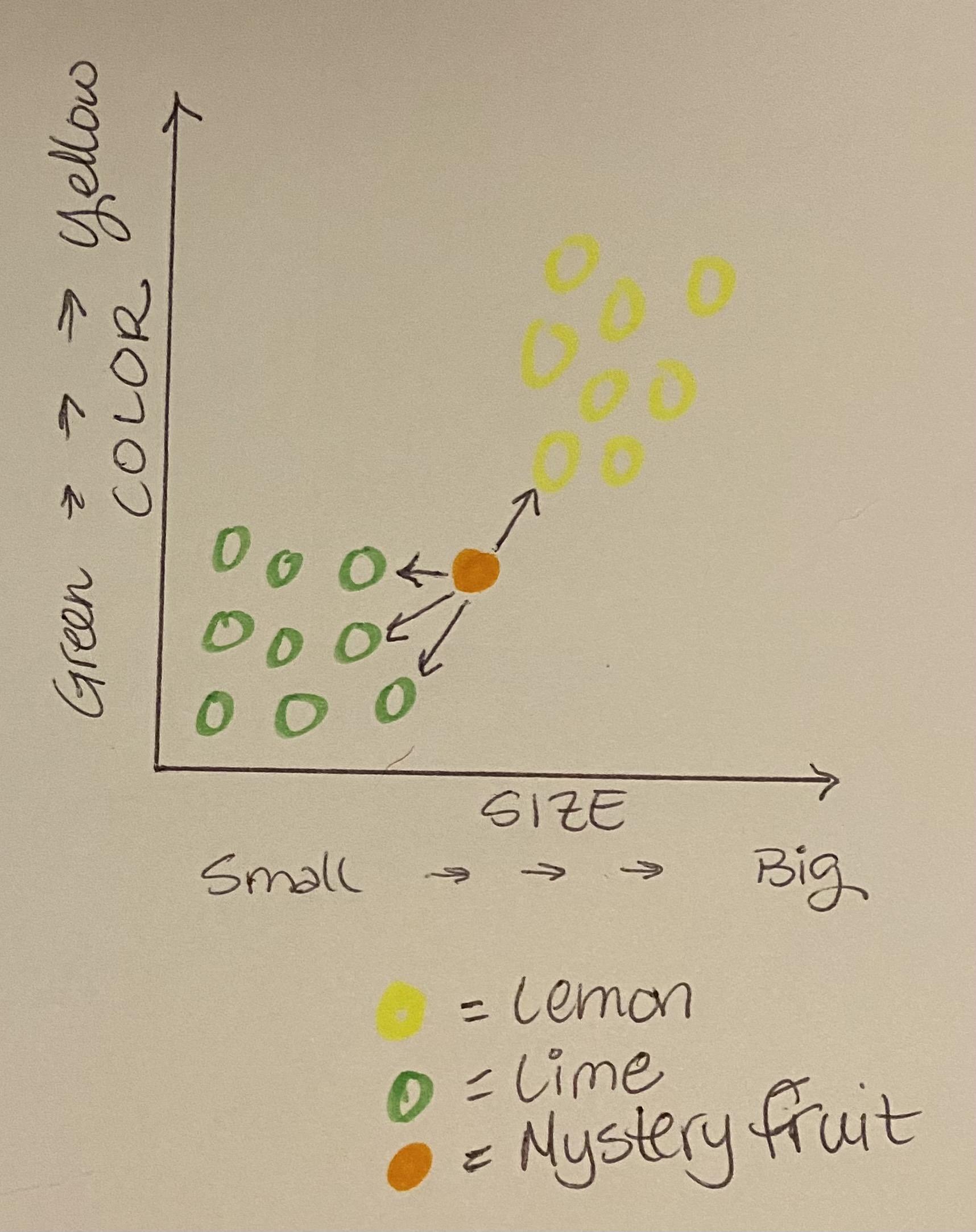
Let's just look at color and size to see how K-NN helps us determine what color your mystery fruit is.



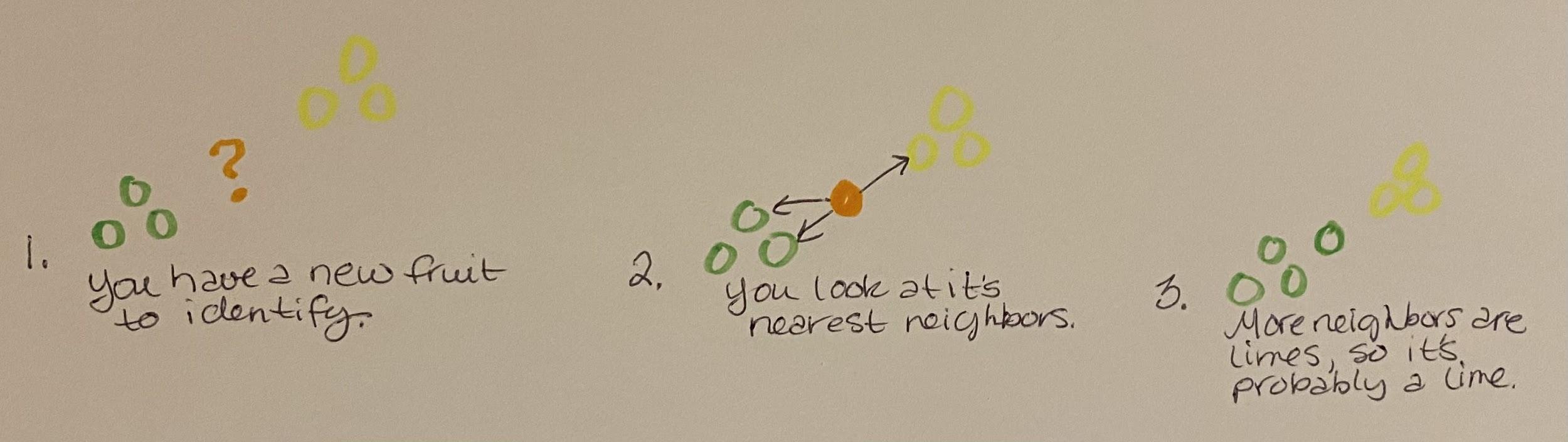
If it's small and green, the map tells us the fruit is a lime. If it is big and yellow, it is a lemon. What else do we see on the graph? You're right! All the limes are clustered together, and all the lemons are clustered together.



Now we take a look at your mystery fruit. Can you tell by looking at the graph whether it's a lemon or a lime? Try looking at its nearest neighbors, the closest fruits in straight line distance from its spot on the chart.



Notice that more of the neighbors are limes than lemons. So, your mystery fruit is probably a lime. Whoo Hoo! you just used the K-Nearest Neighbors to classify your mystery fruit.



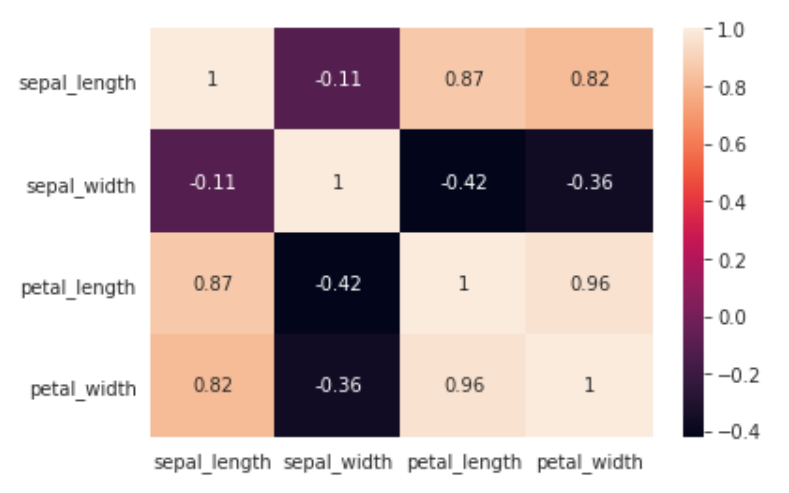
In theory, the algorithm is pretty simple. It is also handy.

#### Build a K-NN Algorithm from Scratch Using a Python Class

**Exploratory Data Analysis**

Before we begin, let’s take a quick look at the [Iris](https://archive.ics.uci.edu/ml/datasets/iris) dataset we use. The Iris dataset contains descriptive information on three species of Irises (the plant). We use this dataset because we can use multiple measurements to solve the taxonomic problem: *"Given the measurements of a mystery iris, can we classify it?"*

Let's see if we can identify some relationships or patterns in the measurement data before we begin.

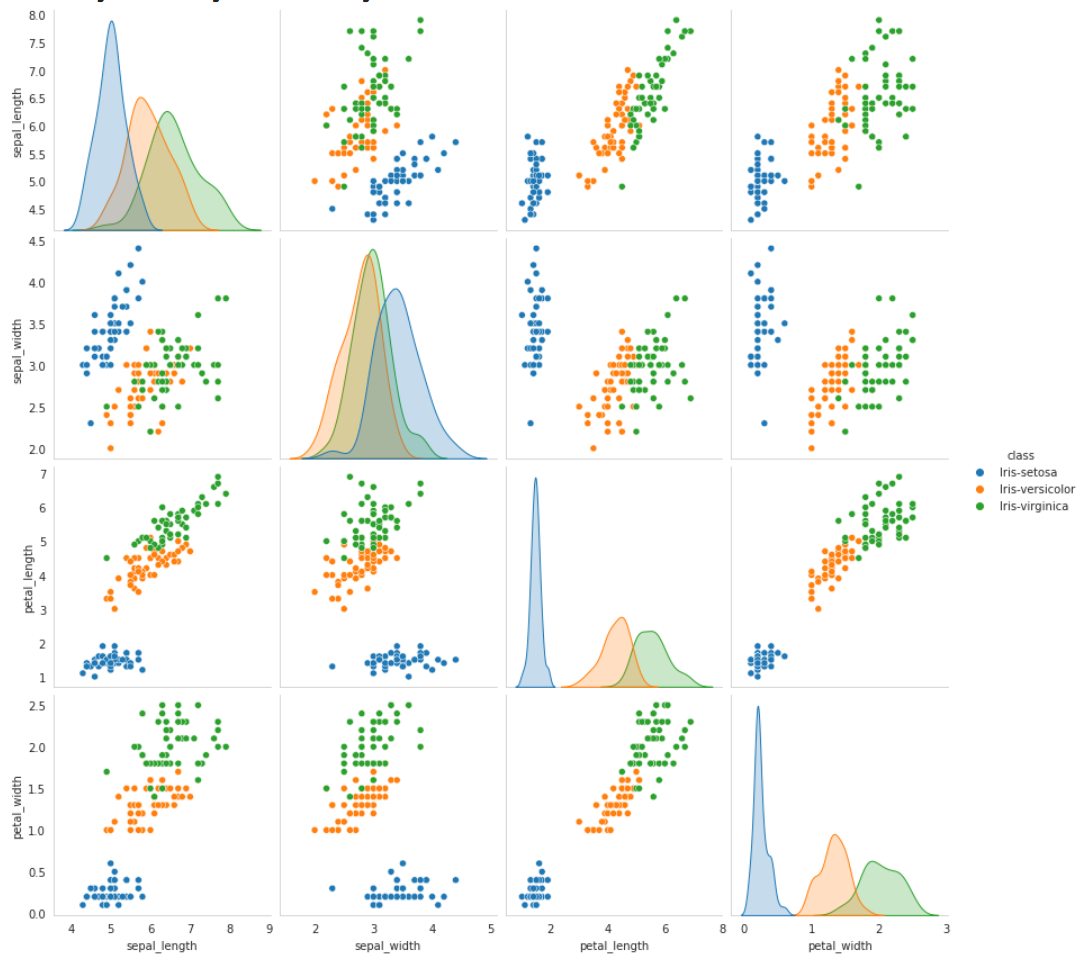


We use the Seaborn library to map the correlations between features. A positive correlation tells us that the two elements have a positive linear relationship. So, as one measurement increases, the other measurement also increases. If a negative correlation exists, there is an inverse relationship where one measurement increases and the other decreases. The closer to 1 or -1 a number is, the stronger the correlation (linear relationship) between the two elements, regardless of whether it's positive or negative.

We observe the strongest correlations to be (in order of greatest strength):

1. petal\_width & petal\_length
2. sepal\_length & petal\_length
3. petal\_width & sepal\_length
4. sepal\_length & petal\_width

We note that petal length and width are the most highly correlated with the class. This means that irises in the Iris Virginica class have the longest and widest petals of the three classes of irises in the dataset. Let's take a look at a broad overview of the inter-correlation of all of these features.



Pair plots allow you to see distributions of individual variables (across the diagonal) and the relationships between two variables within a multi-classification dataset. This is extremely useful in framing our solution to the business problem since we can't visualize in 4 dimensions or higher.

What we observe in the pair plot:

* a separation of the class Iris Setosa in distribution
* The distinction between classes in petal\_length and petal\_width

What we infer from the pair plot:

* if 0 <= petal\_length <= 2 and 0 <= petal\_width <= 0.7, the iris is identified as Iris Setosa
* if 2 <= petal\_length <= 5.2 and 1 <= petal\_width <= 1.7, the iris is identified as Iris Versicolor
* if None of these is true, the iris is identified as Iris Virginica

Now that we have a good idea of what our data looks like and what it represents, we can begin to develop our model.

**Modeling**

As we noted in our exploration of the data, the features are strongly linearly correlated yet have differences in distributions among classes. The K-NN algorithm makes predictions by calculating similarities between each instance, in this case, each flower and its measurements, making it the ideal first attempt algorithm to use to solve the mystery flower problem.

Let's build our algorithm.

To reproduce the From Scratch model, will need to import the following libraries:

* [import pandas as pd](https://pandas.pydata.org/docs/user_guide/index.html#user-guide)
* [import numpy as np](https://numpy.org/doc/1.20/)
* [import seaborn as sns](https://seaborn.pydata.org/)

Note: The loading of the dataset using the Pandas library is not displayed in this article.

We first build a KNearestNeigbors [Python class object](https://docs.python.org/3/tutorial/classes.html?highlight=inheritance#classes) (K-NN)that will hold all of our code. Think of it as the ultimate organizer and top level of hierarchy. We load our K-NN instance object using with fields using the \_\_init\_\_ method.

class KNearestNeighbors(object):

def \_\_init\_\_(self, k):

self.k = k

We create a K-NN instance method to split our dataset into our training dataset and our testing dataset. The benefit of using the Python class structure is that the instance method behaves just like a function and has access to class and instance scope through self.

def train\_test\_split(self, dataset, test\_size=0.33):

n\_test = int(len(dataset) \* test\_size)

test\_set = dataset(n\_test)

train\_set = []

for idx in dataset.index:

if idx in test\_set.index:

continue

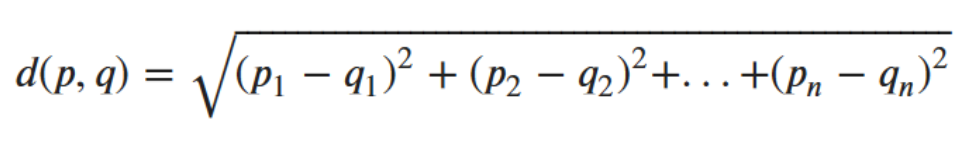
train\_set.append(dataset.iloc[idx])

train\_set = pd.DataFram(train\_set).astype(float).values.tolist()

test\_set = test\_set.astype(float).values.tolist()

return train\_set, test\_set

Because the classification of our mystery iris is unknown, we need to calculate the distances between it and other flowers in the training set. The easiest way (as long as you know the number of dimensions) is by using Euclidean distance or straight-line distance as seen in this formula:



Luckily in Python, you don't have to know the number of dimensions beforehand:

def euclidean\_dist(v1, v2):

v1, v2 = np.array(v1), np.array(v2)

distance = 0

for i in range(len(v1) -1):

distance += (v1[i] - v2[i]) \*\*2

return np.sqrt(distance)

In our predict() method, we implement the algorithm's ability to classify. You may notice there are a few more steps in this method than in our previous ones. Inside the predict method we:

1. Utilize our euclidean\_dist() method to calculate the distance between the test\_instance and each row of the train\_set
2. Sort the distances from lowest to highest by value
3. Keep the k smallest distance
4. Get the values of a class variable for train\_set rows with k smallest distance - these are the nearest neighbors’ values to our mystery flower and have the most similar features.
5. Determine majority class among train\_set rows with k smallest - this is our classification prediction.
6. Return prediction of sorted classes - hands the value back to the caller.

def predict(self, train\_set, test\_instance):

distances = []

for i in range(len(train\_set)):

dist = self.euclidean\_dist(train\_set[i][:-1], test\_instance)

distances.append((train\_set[i], dist))

distances.sort(key=lambda x: x[1])

neighbors = []

for i in range(self.k):

neighbors.append(distances[i][0])

classes = {}

for i in range(len(neighbors)):

response = neighbors[i][-1]

if response in classes:

classes[response] +=1

else:

classes[response] =1

sorted\_classes = sorted(classes.items(), key=lambda x: x[1], reverse=True)

return sorted\_classes[0][0]

Let's test our algorithm to see how well it performed on the training data.

def evaluate(y\_true, y\_pred):

correct = 0

for actual, pred in zip(y\_true, y\_pred):

if actual == pred:

correct +=1

return correct / len(y\_true)

train\_set, test\_set = train\_test\_split(dataset)

print(len(train\_set), len(test\_set))

knn = KNearestNeighbors(k=3)

preds = []

for row in test\_set:

preds\_only = row[:-1]

prediction = knn.predict(train\_set, preds\_only)

preds.append(prediction)

actual = np.array(test\_set)[:, -1]

knn.evaluate(actual, preds)

print(knn.evaluate)

k\_evaluations = []

for k in range(1, 26, 2):

knn = KNearestNeighbors(k=k)

preds = []

for row in test\_set:

preds\_only = row[:-1]

prediction = knn.predict(train\_set, preds\_only)

preds.append(prediction)

curr\_accuracy = knn.evaluate(actual, preds)

k\_evaluations.append((k, curr\_accuracy))

print(k\_evaluations)

# 0.967 is a good score for this dataset

Accuracy Score: 0.9459459459459459

Cross-validation Score: 0.972972972972973

The From Scratch model performs well with an accuracy score of 0.9459 and a cross-validation score of 0.9729. This means the model is predicting the correct class for the flower about 96% of the time. At this point, we would typically iterate on the model, hyper-tuning parameters until we optimize efficiencies within the model to achieve optimal performance. However, today we will look at the same algorithm implemented using the SciKit-Learn library.

#### Test From Scratch K-NN Algorithm Against SciKit Learn K-NN Algorithm

SciKit-Learn is a free software machine learning library for the Python Programming language. It contains efficiency tools for machine learning and statistical modelings such as classification, regression, clustering, and dimensionality reduction.

To reproduce the SkiKit-Learn model, will need to import the following packages:

* [from sklearn.datasets import load\_iris](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)
* [from sklearn.pipeline import make\_pipeline](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html)
* [from sklearn.model\_selection import train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
* [from sklearn.preprocessing import MinMaxScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html)
* [from sklearn.neighbors import NeighborhoodComponentsAnalysis](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NeighborhoodComponentsAnalysis.html)
* [from sklearn.neighbors import KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)
* [from sklearn.metrics import accuracy\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)
* [from sklearn.model\_selection import cross\_val\_score](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

Note: The dataset’s loading using the SkiKit-Learn package is not displayed in this article.

The following model is operating within a K-NN Python class. It is set up just like we did in our From Scratch algorithm.

n\_neighbors = 3

random\_state = 42

# load dataset

X = dataset

y = dataframe["class"]

# split into train/test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.67, random\_state=42, stratify = y)

# reduce dimensions to 2 with neighborhood component analysis

nca = make\_pipeline(

MinMaxScaler(),

NeighborhoodComponentsAnalysis(n\_components=2,

random\_state=random\_state)

)

# use nearest neighbor classifier to evaluate the methoda

knn = KNeighborsClassifier(n\_neighbors=n\_neighbors)

# fit the method's model

nca.fit(X\_train, y\_train)

# fit nearest neighbor classifier on training set

knn.fit(nca.transform(X\_train), y\_train)

# predict the class labels for the provided data

knn.predict(nca.transform(X\_train))

# return probability estimates for the test data

knn.predict\_proba(nca.transform(X\_test))

# calculate the nearest neighbor accuracy on the test set

knn\_accuracy\_sklearn = knn.score(nca.transform(X\_test), y\_test)

# calculate the nearest neighbor cross validation score on test set

scores = cross\_val\_score(knn, X\_train, y\_train, cv=5)

print(f'Accuracy Score: {knn\_accuracy\_sklearn}')

print(f'Cross-validation Score: {np.mean(scores)}')

# 0.967 is a good score for this dataset

Accuracy Score: 0.9702970297029703

Cross-validation Score: 0.9800000000000001

The SciKit-Learn model is performing well with an accuracy score of 0.9702 and a cross-validation score of 0.98. This means the model is predicting the correct class for the flower about 97.5% of the time.

## K-NN Machine Learning Use Cases

#### ...And Limitations

As you observed in the demonstration above, K-NN is a reasonably straight-forward algorithm that makes your model more intelligent. This is the magic of machine learning! You may be wondering, "What are some real-world applications for K-NN other than taxonomy?"

#### **Recommender Systems**

Think about [Netflix](https://medium.com/@springboard_ind/how-netflixs-recommendation-engine-works-bd1ee381bf81#:~:text=For%20every%20new%20subscriber%2C%20Netflix,they%20would%20like%20to%20watch.&text=Netflix's%20machine%20learning%20based%20recommendations,the%20scenes%20and%20refreshes%20it.), [Spotify](https://towardsdatascience.com/how-spotify-recommends-your-new-favorite-artist-8c1850512af0#:~:text=Well%2C%20Spotify's%20recommender%20system%20provides,it%20deems%20%E2%80%9Csimilar%E2%80%9D%20users.), or [The New York Times App](https://open.nytimes.com/how-the-new-york-times-is-experimenting-with-recommendation-algorithms-562f78624d26). What do they all have in common? They recommend movies, songs, or news articles to you. How do they figure out which ones to recommend to you? This is a business problem with a K-NN component to the solution. They collect information on your preferences, group you with people with similar tastes, make recommendations to you based on the closest users. K-NN may be part of the overall algorithm and used only as a jumping-off point, and the logic is at the core of the solution process.

#### **OCR**

[OCR](https://docparser.com/blog/what-is-ocr/) stands for optical character recognition. It allows your computer to read a text page as an image and then translate it to text. While OCR is a complex task, it's built on simple logic. Part of the logic includes measuring and extracting features like lines, points, and curves from two characters, like a 1 and 3. You then compare each character’s features to know the characters and each other to classify the characters. This, too, involves the K-NN logic.

#### **Limitations**

Predicting the weather or the stock market are highly challenging because so many variables are involved. There is no linear correlation in either sphere that would allow us to use past data to predict future performance. Any such industry or sphere heavily influenced by randomness or ambiguity is not a candidate for K-NN logic.

## K-NN Challenge

Complete this exercise on your own:

* using cosine distance instead of Euclidean distance
* standardize the data (for practice for real-world data sets where you will need to do this more often than not)
* post link(s) to your project(s) in the comments that tell me about your experience, what you find easy/challenging, what you learn, your results, etc.

I hope this gives you a foundation or fills in a few gaps regarding K-Nearest Neighbors and Machine Learning.

My Papaw always told me I'd be judged by the company I kept. I wonder what he would think about K-NN in Machine Learning!