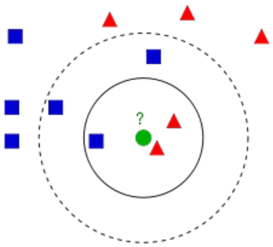
K-Nearest Neighbour

Tushar B. Kute,
http://tusharkute.com



What sort of Machine Learning?

- An idea that can be used for machine learning—as does another maxim involving poultry: "birds of a feather flock together."
- In other words, things that are alike are likely to have properties that are alike.
- We can use this principle to classify data by placing it in the category with the most similar, or "nearest" neighbors.

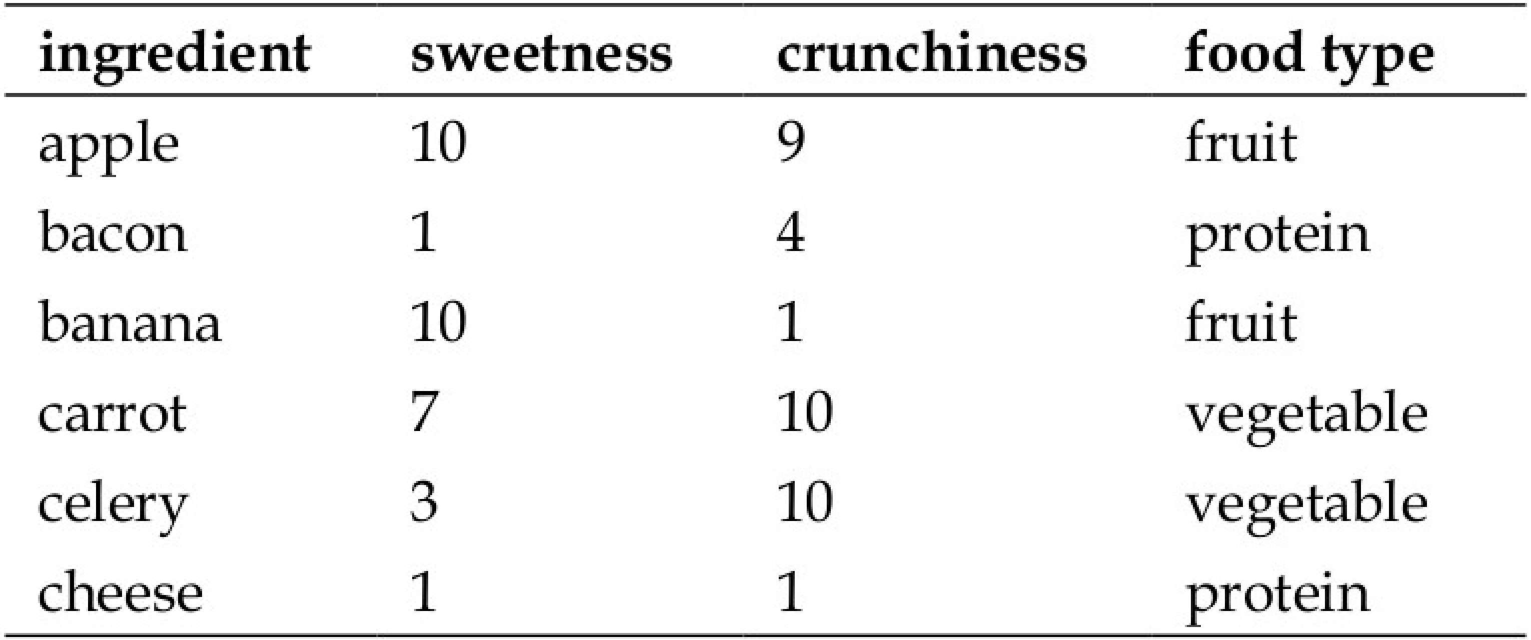
Nearest Neighbor Classification

- In a single sentence, nearest neighbor classifiers are defined by their characteristic of classifying unlabeled examples by assigning them the class of the most similar labeled examples. Despite the simplicity of this idea, nearest neighbor methods are extremely powerful. They have been used successfully for:
  - Computer vision applications, including optical character recognition and facial recognition in both still images and video
  - Predicting whether a person enjoys a movie which he/she has been recommended (as in the Netflix challenge)
  - Identifying patterns in genetic data, for use in detecting specific protein or diseases

The kNN Algorithm

- The kNN algorithm begins with a training dataset made up of examples that are classified into several categories, as labeled by a nominal variable.
- Assume that we have a test dataset containing unlabeled examples that otherwise have the same features as the training data.
- For each record in the test dataset, kNN identifies k records in the training data that are the "nearest" in similarity, where k is an integer specified in advance.
- The unlabeled test instance is assigned the class of the majority of the k nearest neighbors

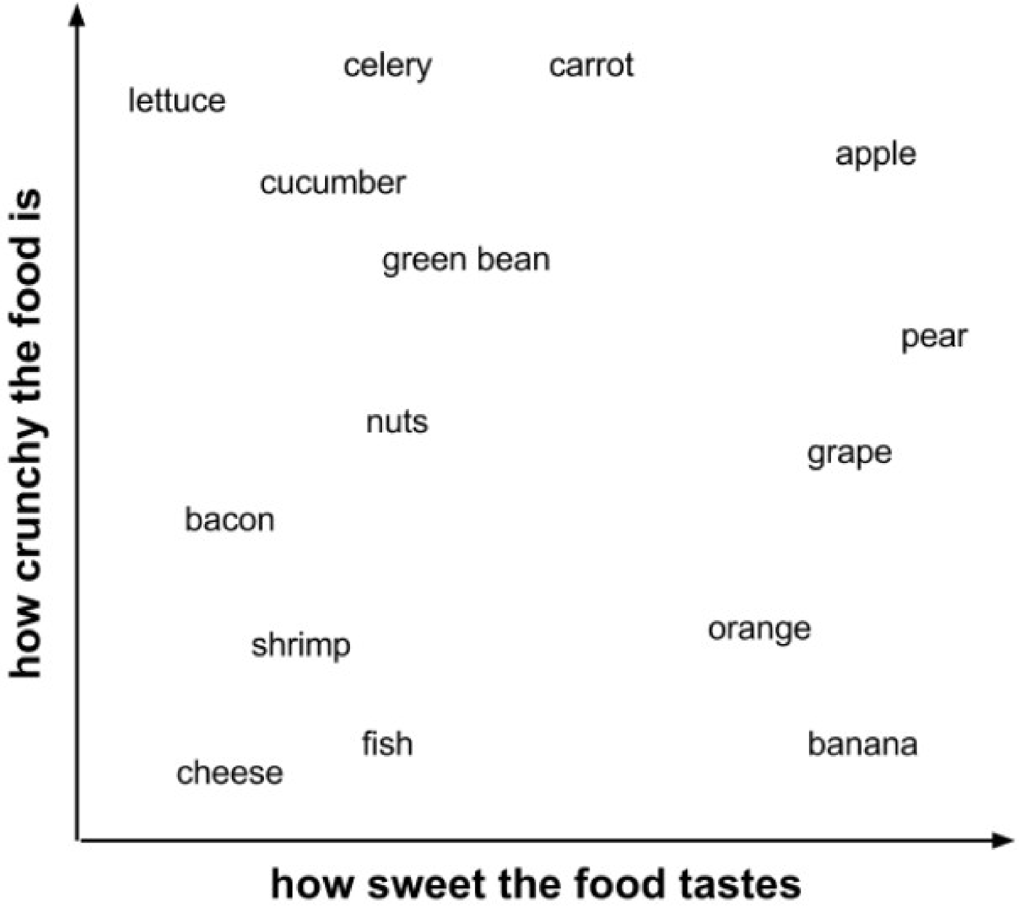
Example:



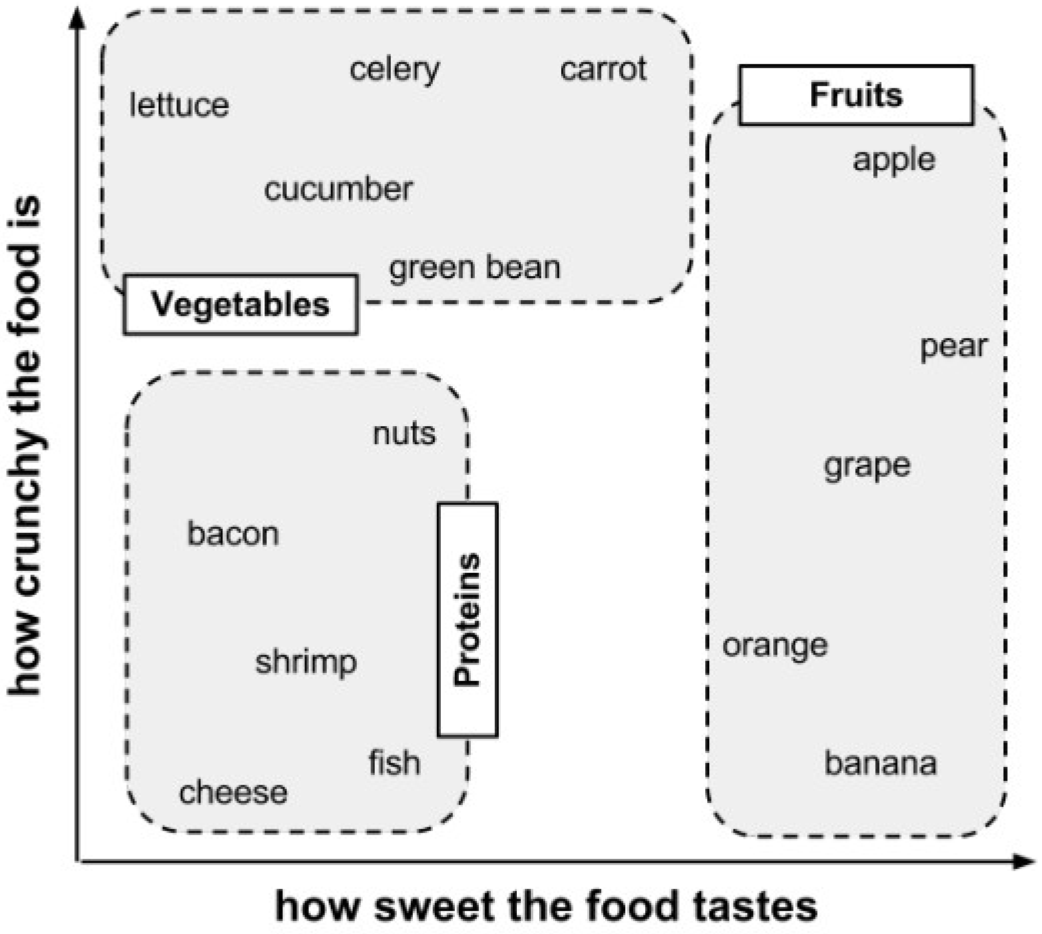
ingredient	sweetness	crunchiness	food type
apple	10	9	fruit
bacon	1	4	protein
banana	10	1	fruit
carrot	7	10	vegetable
celery	3	10	vegetable
cheese	1	1	protein

Reference: Machine Learning with R, Brett Lantz, Packt Publishing

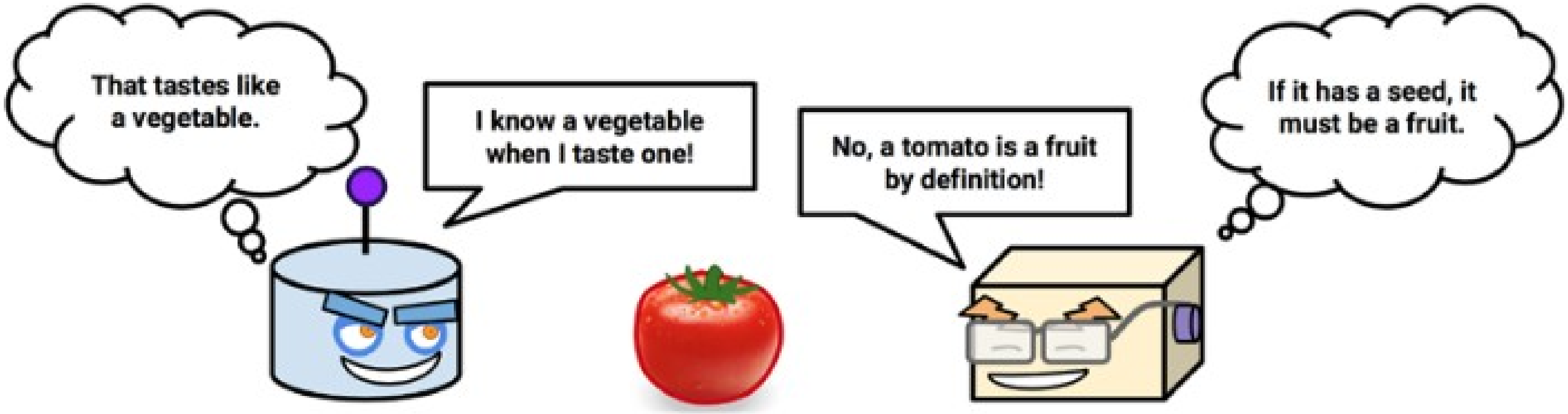
Example:


celery
carrot
lettuce
apple
cucumber
green bean
pear
nuts
grape
bacon
orange
shrimp
fish
banana
cheese
how crunchy the food is
how sweet the food tastes

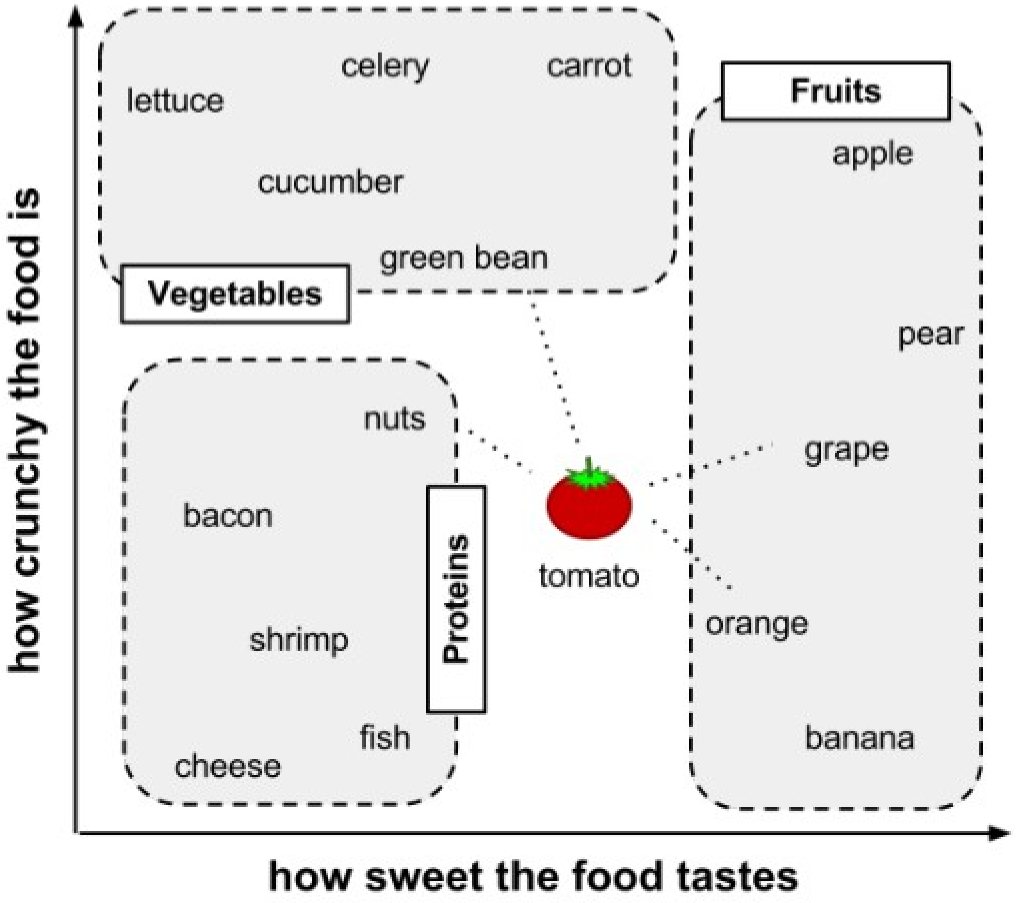
Example:


celery
carrot
lettuce
cucumber
green bean
Vegetables
Fruits
apple
pear
grape
orange
banana
nuts
bacon
Proteins
shrimp
fish
cheese
how crunchy the food is
how sweet the food tastes

Classify me now!


That tastes like a vegetable.
I know a vegetable when I taste one!
No, a tomato is a fruit by definition!
If it has a seed, it must be a fruit.

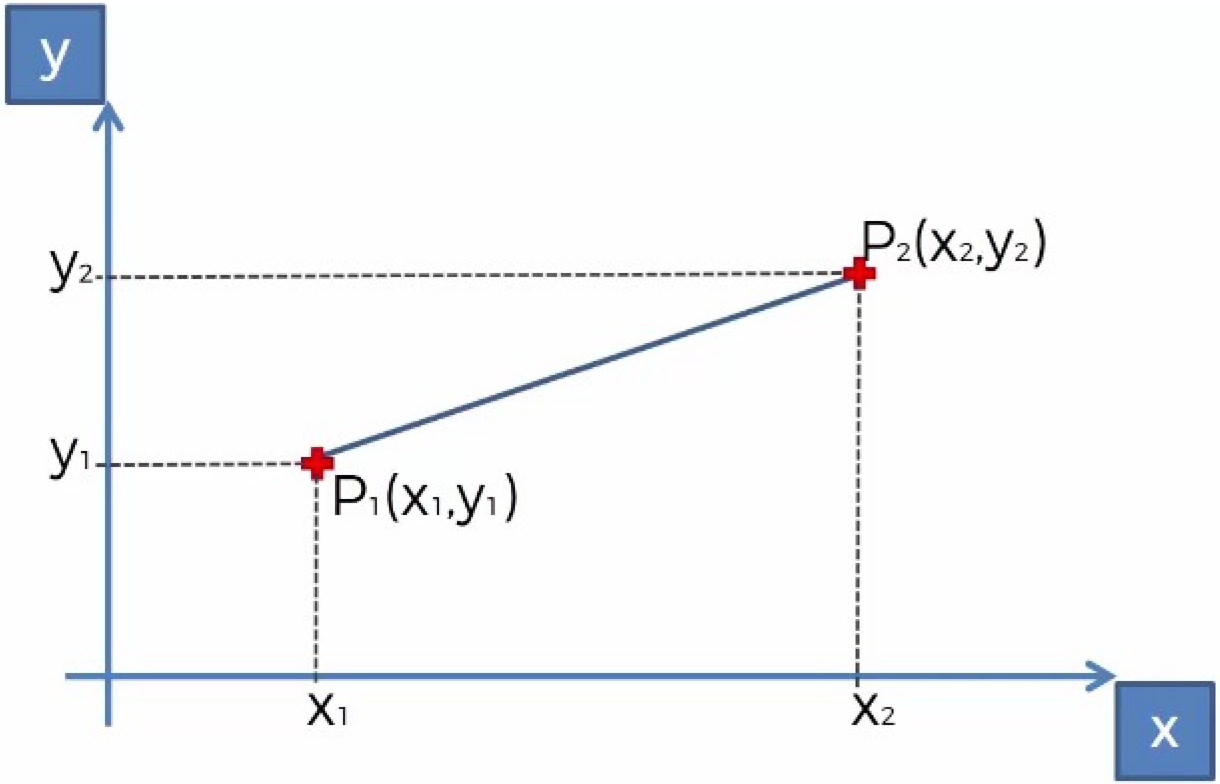
Example:


celery
carrot
lettuce
cucumber
green bean
Vegetables
Fruits
apple
pear
grape
orange
banana
nuts
bacon
shrimp
fish
cheese
Proteins
tomato
how crunchy the food is
how sweet the food tastes

Calculating Distance

- Locating the tomato's nearest neighbors requires a distance function, or a formula that measures the similarity between two instances.
- There are many different ways to calculate distance.
- Traditionally, the kNN algorithm uses Euclidean distance, which is the distance one would measure if you could use a ruler to connect two points, illustrated in the previous figure by the dotted lines connecting the tomato to its neighbors.

Calculating Distance



$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Reference: Super Data Science

Distance

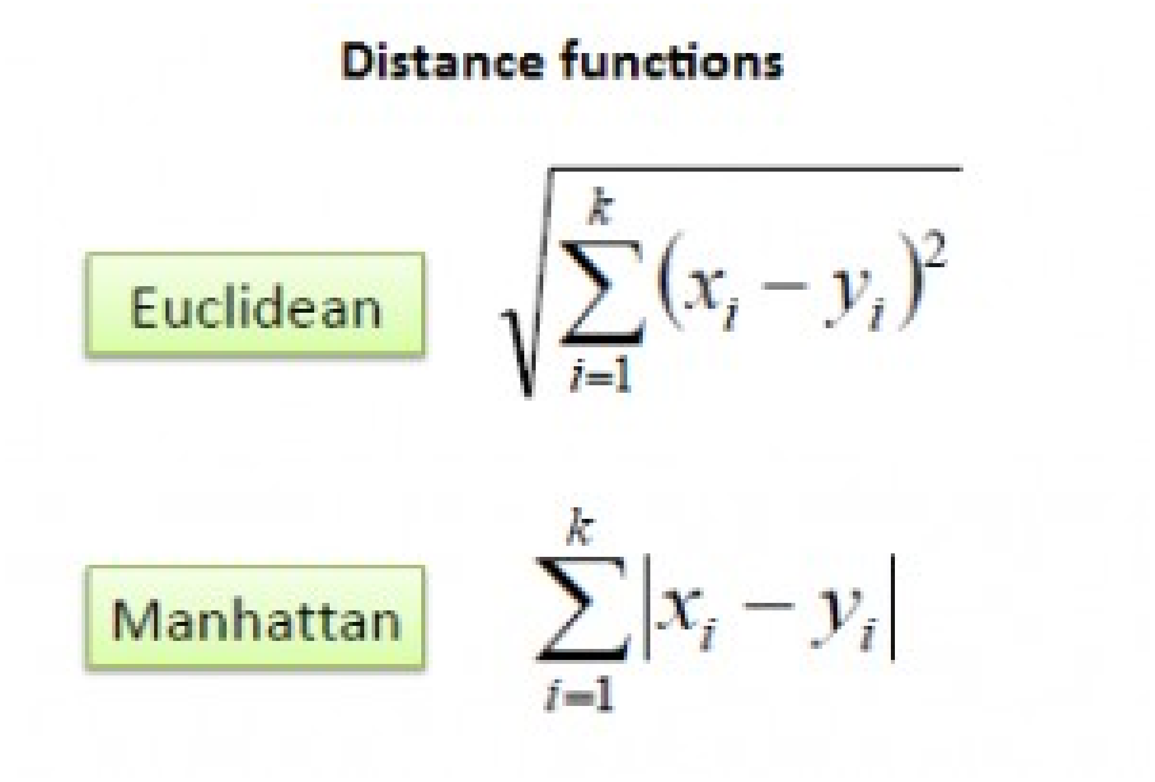
- Euclidean distance is specified by the following formula, where p and q are th examples to be compared, each having n features. The term p1 refers to the value of the first feature of example p, while q1 refers to the value of the first feature of example q:

$$\text{dist}(p,q)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}$$

- The distance formula involves comparing the values of each feature. For example, to calculate the distance between the tomato (sweetness = 6, crunchiness = 4), and the green bean (sweetness = 3, crunchiness = 7), we can use the formula as follows:

$$\text{dist}(\textit{tomato}, \textit{green bean})=\sqrt{(6-3)^2+(4-7)^2}=4.2$$

Distance



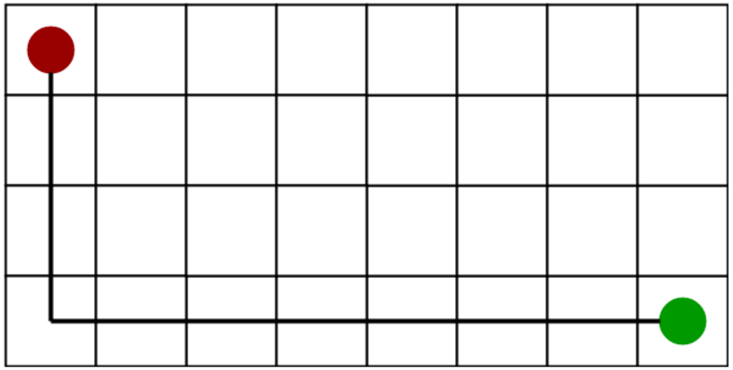
Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

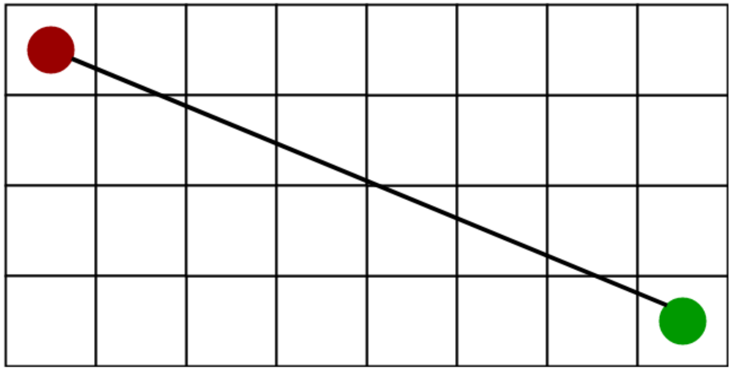
Manhattan $$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Distance

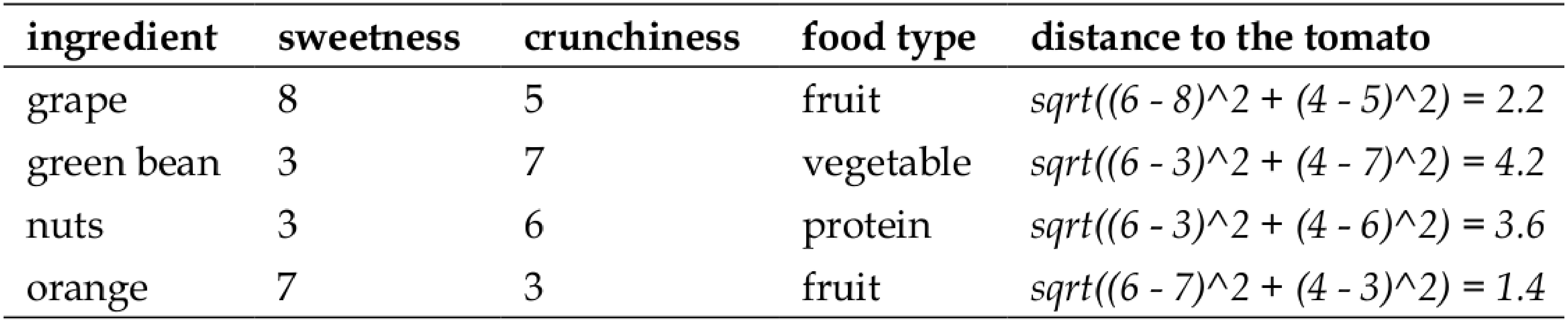
Manhattan Distance



Euclidean Distance



Closest Neighbors

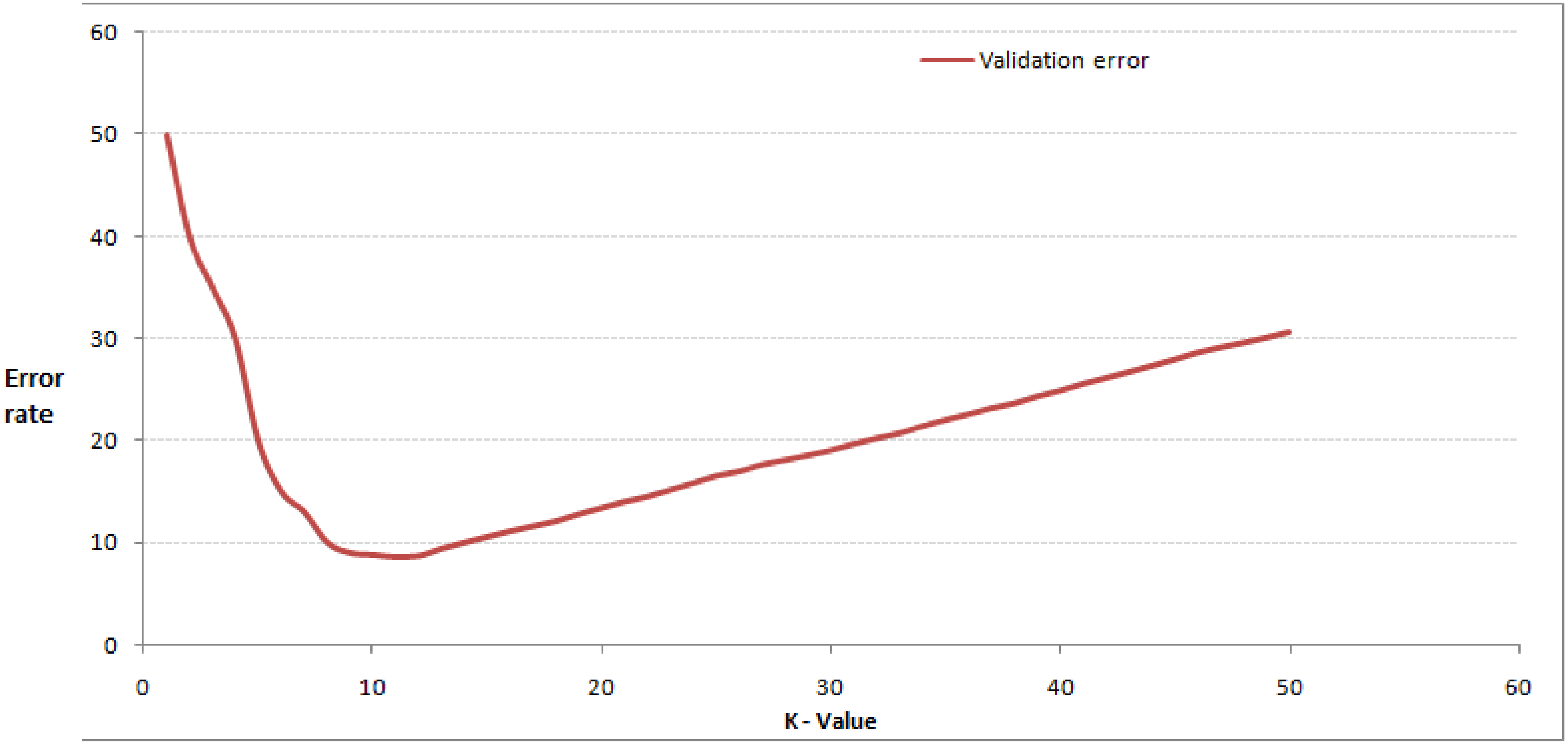


ingredient	sweetness	crunchiness	food type	distance to the tomato
grape	8	5	fruit	*sqrt((6 - 8)^2 + (4 - 5)^2) = 2.2*
green bean	3	7	vegetable	*sqrt((6 - 3)^2 + (4 - 7)^2) = 4.2*
nuts	3	6	protein	*sqrt((6 - 3)^2 + (4 - 6)^2) = 3.6*
orange	7	3	fruit	*sqrt((6 - 7)^2 + (4 - 3)^2) = 1.4*

Choosing appropriate k

- Deciding how many neighbors to use for kNN determines how well the mode will generalize to future data.
- The balance between overfitting and underfitting the training data is a problem known as the bias-variance tradeoff.
- Choosing a large k reduces the impact or variance caused by noisy data, but can bias the learner such that it runs the risk of ignoring small, but important patterns.

Choosing appropriate k



Choosing appropriate k

- In practice, choosing k depends on the difficulty of the concept to be learned and the number of records in the training data.
- Typically, k is set somewhere between 3 and 10. One common practice is to set k equal to the square root of the number of training examples.
- In the classifier, we might set k = 4, because there were 15 example ingredients in the training data and the square root of 15 is 3.87.

Min-Max normalization

- The traditional method of rescaling features for kNN is min-max normalization.
- This process transforms a feature such that all of its values fall in a range between 0 and 1. The formula for normalizing a feature is as follows. Essentially, the formula subtracts the minimum of feature X from each value and divides by the range of X:

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

- Normalized feature values can be interpreted as indicating how far, from 0 percent to 100 percent, the original value fell along the range between the original minimum and maximum.

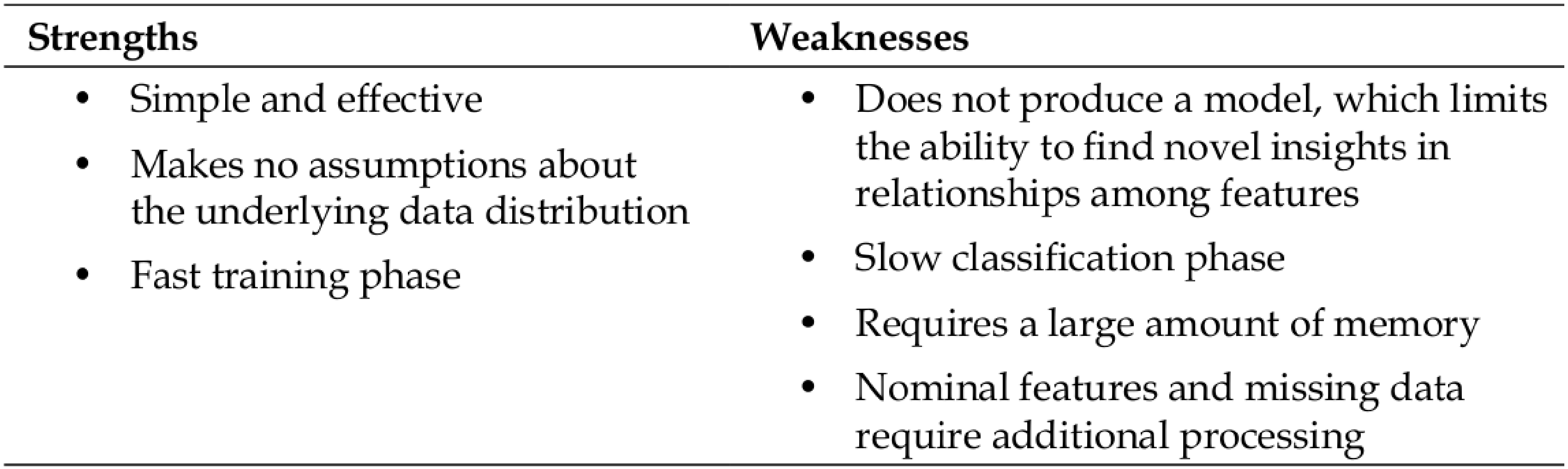
The Lazy Learning

- Using the strict definition of learning, a lazy learner is not really learning anything.
- Instead, it merely stores the training data in it. This allows the training phase to occur very rapidly, with a potential downside being that the process of making predictions tends to be relatively slow.
- Due to the heavy reliance on the training instances, lazy learning is also known as instance-based learning or rote learning.

Few Lazy Learning Algorithms

- K Nearest Neighbors
- Local Regression
- Lazy Naive Bayes

The kNN Algorithm

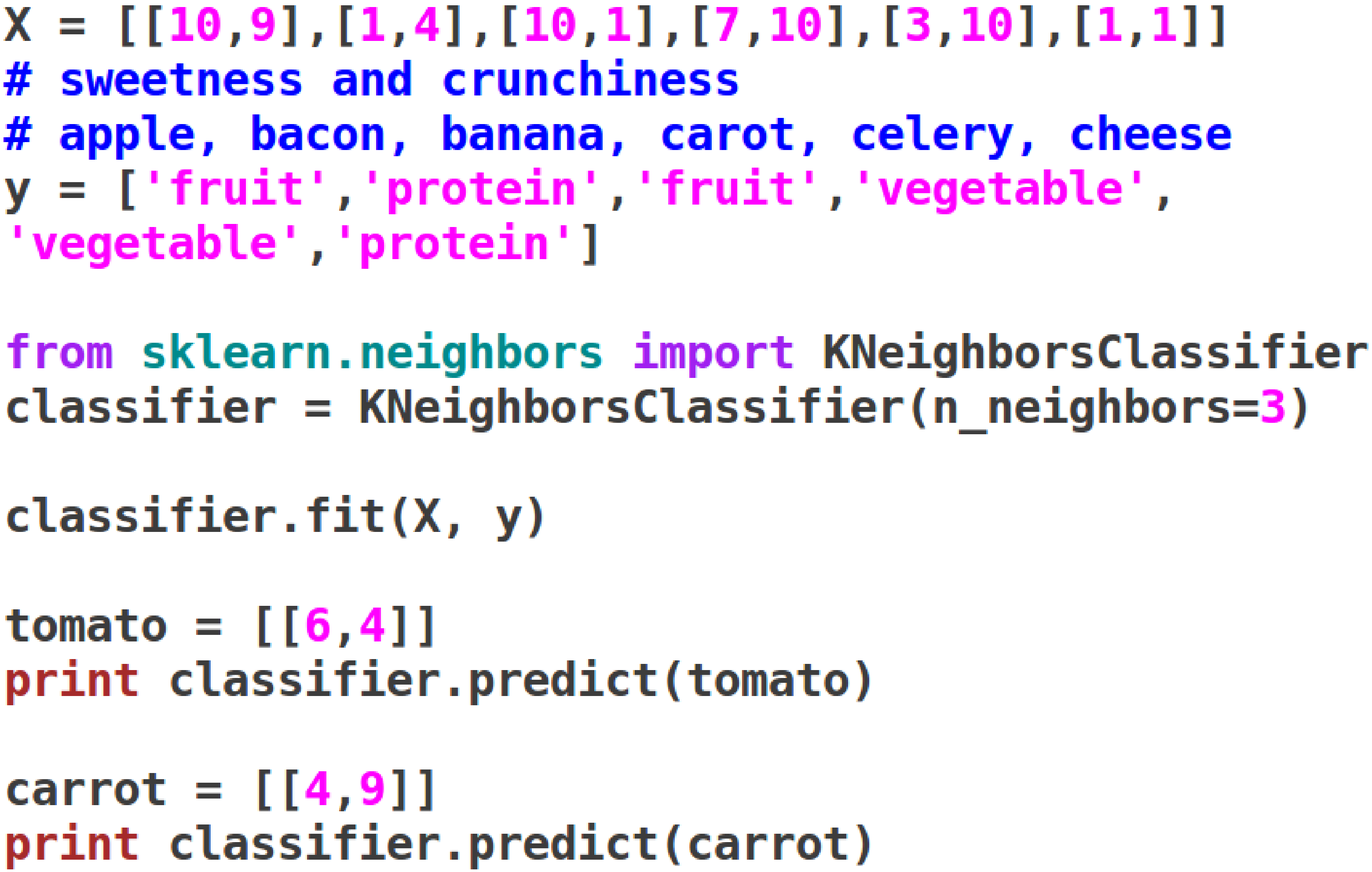


Strengths	Weaknesses
• Simple and effective • Makes no assumptions about the underlying data distribution • Fast training phase	• Does not produce a model, which limits the ability to find novel insights in relationships among features • Slow classification phase • Requires a large amount of memory • Nominal features and missing data require additional processing

Python Packages needed

- pandas
  - Data Analytics
- numpy
  - Numerical Computing
- matplotlib.pyplot
  - Plotting graphs
- sklearn
  - Classification and Regression Classes

Sample Application

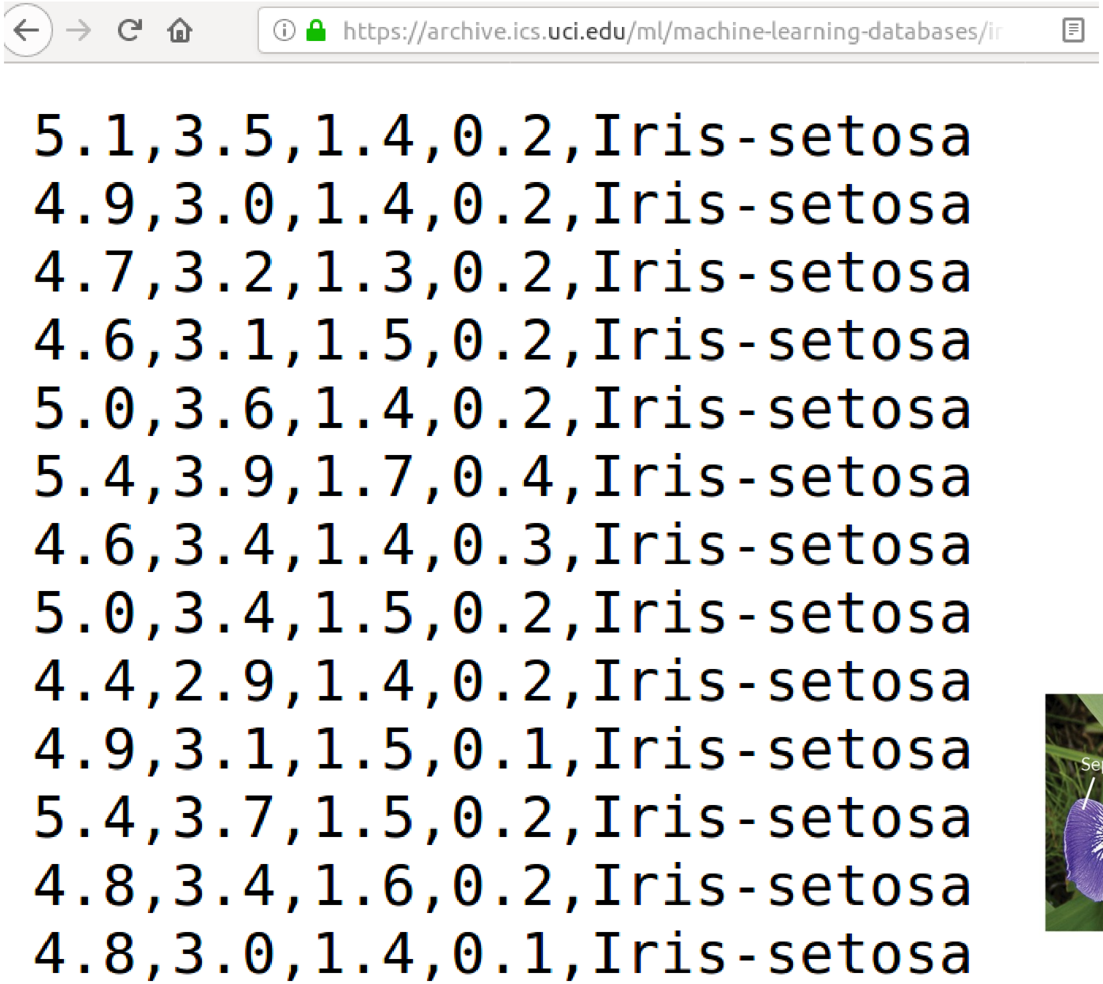


```
X = [[10,9],[1,4],[10,1],[7,10],[3,10],[1,1]]
# sweetness and crunchiness
# apple, bacon, banana, carot, celery, cheese
y = ['fruit','protein','fruit','vegetable',
'vegetable','protein']

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)

classifier.fit(X, y)

tomato = [[6,4]]
print classifier.predict(tomato)

carrot = [[4,9]]
print classifier.predict(carrot)
```

KNN – Classification : Dataset

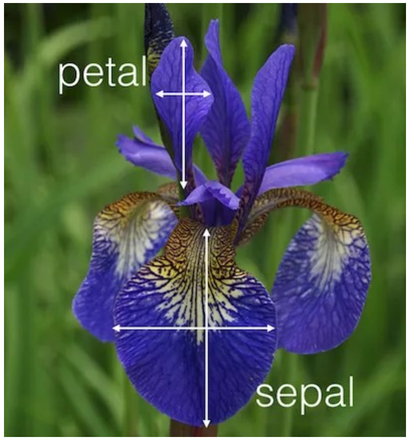
- The best small project to start with on a new tool is the classification of iris flowers (e.g. the iris dataset).
- This is a good project because it is so well understood.
  - Attributes are numeric so you have to figure out how to load and handle data.
  - It is a classification problem, allowing you to practice with perhaps an easier type of supervised learning algorithm.
  - It is a multi-class classification problem (multi-nominal) that may require some specialized handling.
  - It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or A4 page).
  - All of the numeric attributes are in the same units and the same scale, not requiring any special scaling or transforms to get started.

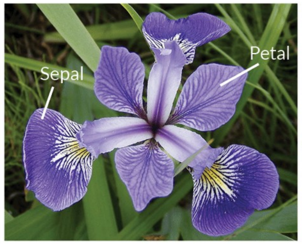
Dataset



https://archive.ics.uci.edu/ml/machine-learning-databases/ir

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
```





Iris Versicolor

Iris Setosa

Iris Virginica

KNN – Classification : Dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# URL to read the dataset
url = "https://archive.ics.uci.edu/ml/
machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length','sepal-width',
'petal-length','petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=names)

print dataset.head()
```

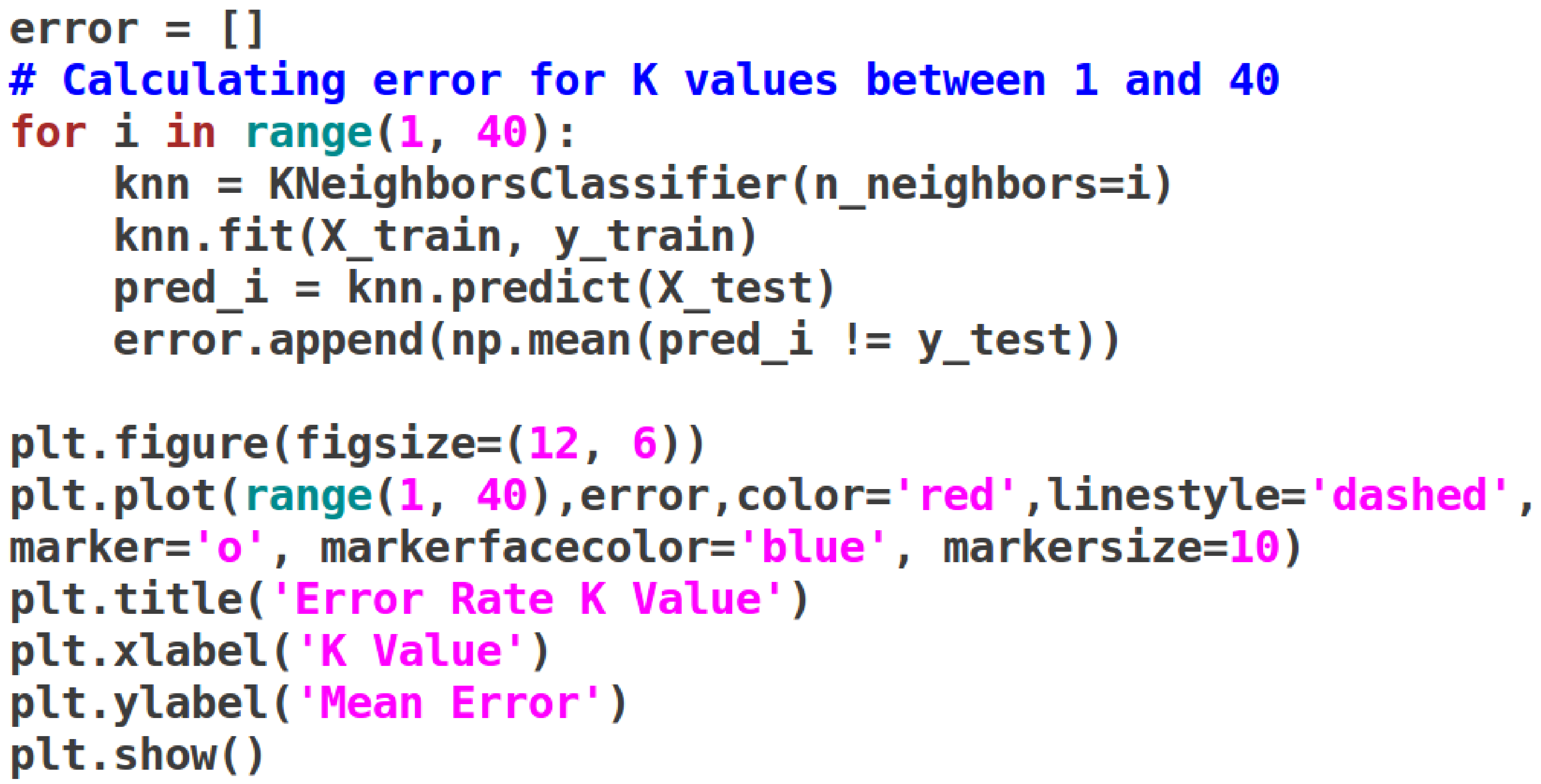
Pre-processing

```python
# Input variables
X = dataset.iloc[:, :-1].values
# Output variable
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.20)

# Feature scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Characterize

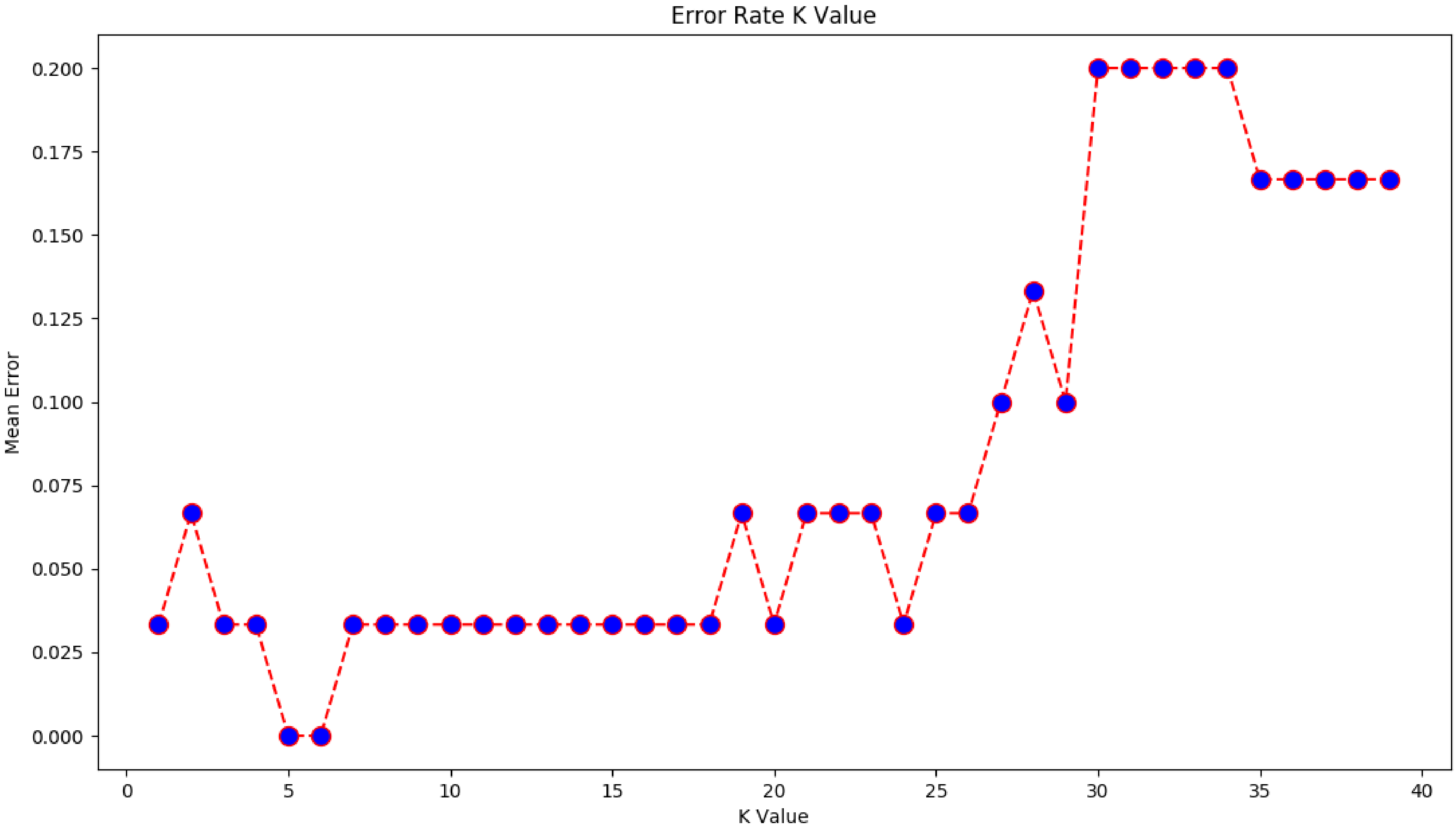
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score
print confusion_matrix(y_test, y_pred)
print classification_report(y_test, y_pred)
print accuracy_score(y_test, y_pred) * 100
```

Finding error with changed k



```
error = []
# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40),error,color='red',linestyle='dashed',
marker='o', markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
```

Error rate


Error Rate K Value
Mean Error
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
25
30
35
40
K Value

Useful resources

- www.mitu.co.in
- www.pythonprogramminglanguage.com
- www.scikit-learn.org
- www.towardsdatascience.com
- www.medium.com
- www.analyticsvidhya.com
- www.kaggle.com
- www.stephacking.com
- www.github.com

Thank you

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com