CS167: Machine Learning

First ML Algorithm:
k-Nearest Neighbor (k-NN)

Wednesday, February 11th, 2026

Drake
UNIVERSITY

Quick Overview

- So far, we've talked about:
  - Introduction to ML
  - Python review
  - Introduction to Pandas

Today's Agenda

- Topics:
  - Notebook # 1 help
  - k-Nearest Neighbor (k-NN)
  - Distances
  - kNN Implementation using Pandas

Our First Machine Learning Model: kNN

- We are starting with a relatively simple, but foundational, machine learning model, the **k-Nearest Neighbors (kNN)** algorithm.

  - **1-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 1 closest training examples

  - **3-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 3 closest training examples

    …

  - **k-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the k closest training examples

1-Nearest Neighbor (1-NN)

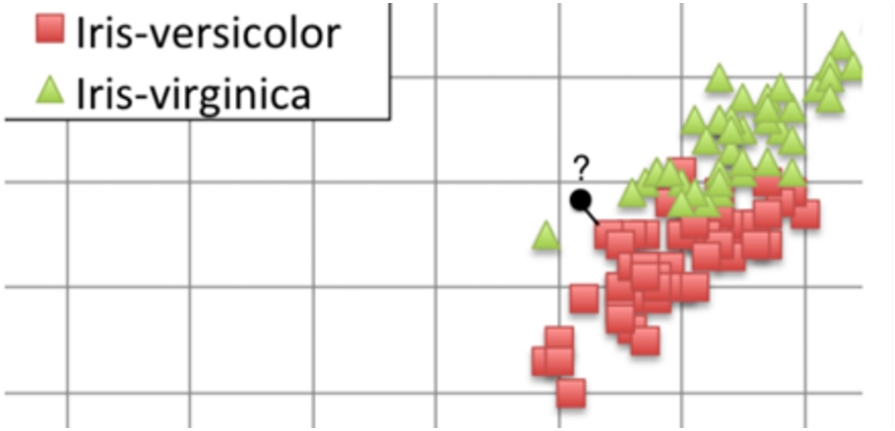
- **1-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 1 closest training examples
  - In other words, k=1

- Let's assume this subset of Iris has only 2 classes (even number):

  `Iris-versicolor`

  `Iris-virginica`

- What class will a **1NN** algorithm predict?



3-Nearest Neighbor (3-NN)

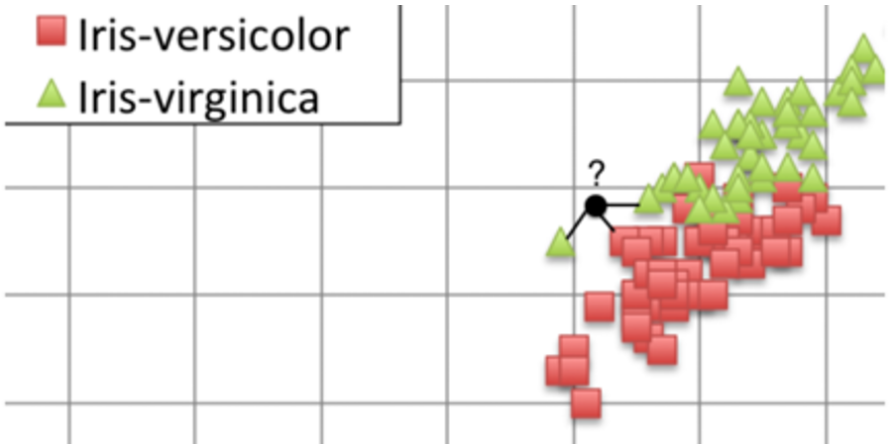
- **3-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 3 closest training examples
  - In other words, k=3

- Let's assume this subset of Iris has only 2 classes (even number):

  `Iris-versicolor`

  `Iris-virginica`

- What class will a **3NN** algorithm predict?



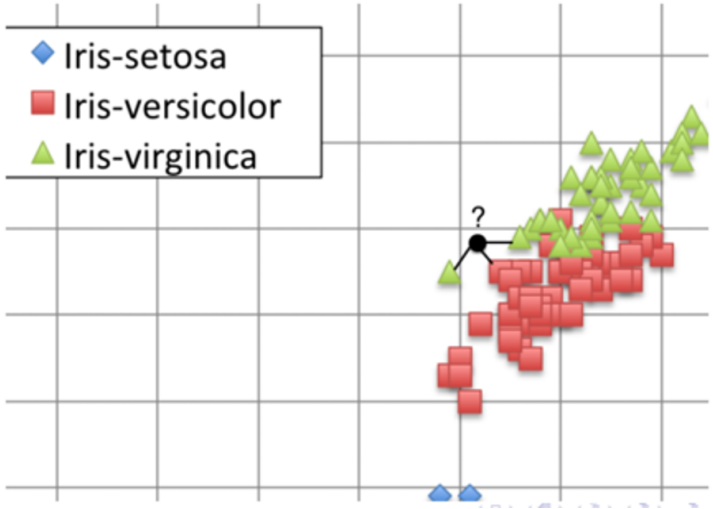
k-Nearest Neighbor (k-NN)

- **k-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the k closest training examples

- The full Iris Dataset has 3 classes (odd number):

```
Iris-versicolor
Iris-virginica
Iris-setosa
```

- What value of k we should use for **k-NN** algorithm to predict correctly?



k-Nearest Neighbor (k-NN)

- **k-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the k closest training examples

- What value of k we should use for **k-NN** algorithm to predict correctly?

  - In general, try out various values of **k=1**, **k=2**, …, **k=100** etc

  - Find out the accuracies on validation set **(NOT ON TRAINING SET)**

  - Pick the k-value for which you get the highest accuracy on validation set **(NOT ON TRAINING SET)**

What do you mean by “closest”?

- **k-Nearest-Neighbor Algorithm**: Predict the *most commonly appearing* class among the **k** closest training examples.

- Alright … but how do we determine which training examples are the ‘closest'?

- **defining 'nearness':** as the machine learning engineer, we get to choose how we define *close*.
  - **What ways can you think of to determine a distance between any two training examples?**

Today's Agenda

- Topics:
  - Notebook # 1 help
  - k-Nearest Neighbor (k-NN)
  - Distances
  - kNN Implementation using Pandas

Distances

- **Euclidean distance:** example in 2D space the distance between two points (x1, y1) and (x2, y2) is:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- **2D Euclidean distance:** example in 2D space the distance between two points (1, 2) and (4, 6):

$$\sqrt{(1-4)^2 + (2-6)^2}$$
$$\sqrt{(-3)^2 + (-4)^2}$$
$$\sqrt{9 + 16}$$
$$\sqrt{25}$$
$$5$$

Distances

- **Euclidean distance in n-Dimensional Space:**

Suppose features of an example: $\langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$

- where n is the number of features.

Then the Euclidean Distance:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Distances

- **Example for Euclidean distance in n-Dimensional Space:**

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Suppose we find a new iris with:

- 7.2 cm sepal length, 2.5 cm sepal width, 5.1 cm petal length, and 1.5 cm petal width

Here are some rows from the training data:

sepal length	sepal width	petal length	petal width	species
4.6	3.2	1.4	0.2	Iris-setosa
6.2	2.8	4.8	1.8	Iris-virginica

Distances

- **Example for Euclidean distance in n-Dimensional Space:**

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Distance from this new iris sample to training data:

- <7.2 cm, 2.5 cm, 5.1 cm, and 1.5 cm>

sepal length	sepal width	petal length	petal width	species
4.6	3.2	1.4	0.2	Iris-setosa
6.2	2.8	4.8	1.8	Iris-virginica

Distance to first row:

$$\sqrt{(4.6 - 7.2)^2 + (3.2 - 2.5)^2 + (1.4 - 5.1)^2 + (0.2 - 1.5)^2}$$

$$\approx 4.76$$

Group Exercise: Distances

- **Example for Euclidean distance in n-Dimensional Space:**

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Distance from this new iris sample to following training data samples:

- <**7.2** cm, **2.5** cm, **5.1** cm, and **1.5** cm>

sepal length	sepal width	petal length	petal width	species
4.6	3.2	1.4	0.2	Iris-setosa
6.2	2.8	4.8	1.8	Iris-virginica
6.2	2.2	4.5	1.5	Iris-versicolor
6.3	2.7	4.9	1.8	Iris-virginica

- **What would the 1-NN version predict? 2-NN? 3-NN? 4-NN?**

Other Distance Functions

- You can use any distance function you want. Other common distances include:

- **Manhattan Distance:** $d(x_i, x_j) = \sum_{r=1}^{n} |a_r(x_i) - a_r(x_j)|$

- **Minkowski distance:** $d(x_i, x_j) = (\sum_{r=1}^{n} |a_r(x_i) - a_r(x_j)|^p)^{1/p}$

- **Or you can make your own:**
  - Online dating use distance scores to predict who you will swipe left/right on
  - Baseball similarity matrix
  - etc.

Today's Agenda

- Topics:
  - Notebook # 1 help
  - k-Nearest Neighbor (k-NN)
  - Distances
  - kNN Implementation using Pandas

k-NN Implementation in Python/Pandas

- Let's build a **5-Nearest-Neighbor** Iris classifier from scratch — using our Pandas/Python skills:

- To implement this 5NN, we need to do 3 things:

1. Calculate the distances from each of the rows to the new instance

3. Sort the data by these distances

5. Select the k closest training examples and use them to predict the most commonly occurring class of the closest neighbors.

Step 1: Calculate the Distances

- Let's start by adding a new column to our `iris` DataFrame that is the distance from each existing row to the new instance with:
  - 5.1 petal length, 7.2 sepal length, 1.5 petal width, and 2.5 sepal width
  - The syntax for adding a new column is as follows:
    - `df['new col name'] = _____`

```
iris_dist     = (new_iris['petal length'] - iris['petal length'])**2 + \
                (new_iris['sepal length'] - iris['sepal length'])**2 + \
                (new_iris['petal width']  - iris['petal width'])**2 + \
                (new_iris['sepal width']  - iris['sepal width'])**2

# insert the distance into a new column in the same DataFrame, 'iris', as follow
iris['distance_column'] = np.sqrt(iris_dist)
# iris['distance_column'] = iris_dist**0.5 # it also calculates the square root
iris.head()
```

Step 2: Sort the data by the Distances

- Let's now sort our data using the built in `sort_values()` function. [documentation]

- We want to find the nearest k neighbors, so sorting them in ascending order (which is the default setting for `sort_values()` will work nicely.

```
iris_sorted = iris.sort_values(by='distance_column')
iris_sorted.head(8) #shortest distances first
```

	sepal length	sepal width	petal length	petal width	species	distance_column
76	6.8	2.8	4.8	1.4	Iris-versicolor	0.591608
52	6.9	3.1	4.9	1.5	Iris-versicolor	0.700000
77	6.7	3.0	5.0	1.7	Iris-versicolor	0.741620
50	7.0	3.2	4.7	1.4	Iris-versicolor	0.836660
129	7.2	3.0	5.8	1.6	Iris-virginica	0.866025
86	6.7	3.1	4.7	1.5	Iris-versicolor	0.877496
58	6.6	2.9	4.6	1.3	Iris-versicolor	0.900000
54	6.5	2.8	4.6	1.5	Iris-versicolor	0.911043

Step 3: Display the most common species among these 5

```
k = 5
print(f'Sorted distances closest {k}')
top_k_samples = iris_sorted.iloc[0:k][ ["species", "distance_column"] ]
top_k_samples.head(k)
```

```
Sorted distances closest 5
```

	species	distance_column
76	Iris-versicolor	0.591608
52	Iris-versicolor	0.700000
77	Iris-versicolor	0.741620
50	Iris-versicolor	0.836660
129	Iris-virginica	0.866025

```
predicted_label = top_k_samples['species'].mode()
print(f'Label of the new sample according to {k}-NN: {predicted_label[0]}')
```

- And Viola! We have successfully implemented our first machine learning model from scratch.

Programming Exercise:

- Rewrite **k-NN** code so that it's a function.

- Pass the iris measurements (specimen), DataFrame, and k as parameters and return the predicted class.

```
def kNN(new_iris, iris, k):
    # write your code in here to make this function work
    predicted_label = -1

    # 1. calculate distances
    # your code here
    # ...


    # 2. sort
    # your code here
    # ...


    # 3. predict
    # your code here
    # ...


    return predicted_label
```

Group Programming Exercise:

- Rewrite **k-NN** code so that it's a function.
- Pass the iris measurements (specimen), DataFrame, and k as parameters and return the predicted class.

```
new_iris = {}
new_iris['petal length'] = 5.1
new_iris['sepal length'] = 7.2
new_iris['petal width'] = 1.5
new_iris['sepal width'] = 2.5

# call the function you just wrote
predicted_label = kNN(new_iris, iris, 55)
print(f'Label of the new sample according to {k}-NN: {predicted_label[0]}')
```

Discussion Question

- What do we do if the features aren't numbers?
  - like Titanic `embark_town`... how can we calculate a distance between `Southampton` and `Queenstown`?

```
[ ] titanic.embark_town.unique()

    array(['Southampton', 'Cherbourg', 'Queenstown', nan]
```

```
pd.get_dummies(titanic.embark_town)
```

	Cherbourg	Queenstown	Southampton
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

891 rows × 3 columns

Discussion Question

- What if our **target variable** is continuous rather than categorical? How would we make a prediction using kNN?
  - Can we do regression with kNN? If so, how?

- Example of Regression problems
  - predict tomorrow's temperature
  - predict the fuel efficiency of a vehicle
  - predict how much someone will like a show on Netflix