CS167: Machine Learning

kNN using **scikit-learn** library

Monday, February 16th, 2026

Drake UNIVERSITY

Review

- Topics:
  - kNN Implementation using Pandas (last lecture)

Review: 3-Nearest Neighbor (3-NN)

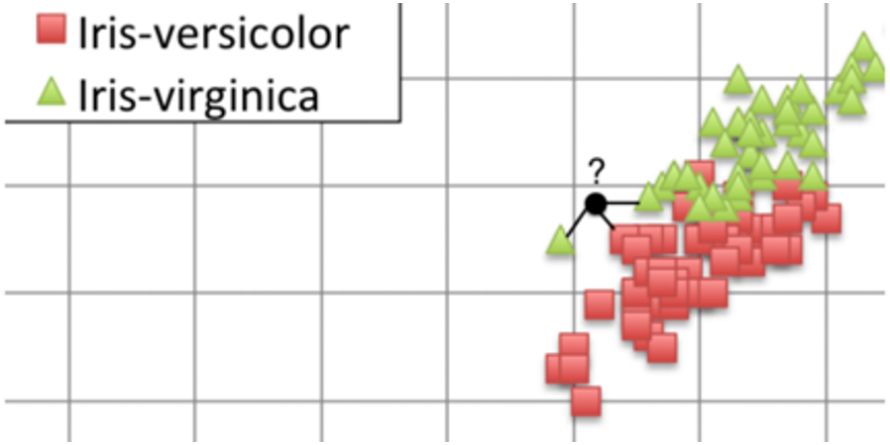
- **3-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 3 closest training examples
  - In other words, k=3

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

  `Iris-versicolor`

  `Iris-virginica`

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



Review: k-Nearest Neighbor (kNN)

- **k-Nearest-Neighbor** predict the most commonly occurring class of the *k nearest neighbors*.

1. Calculate the distance between the new point (e.g. the Iris we would like to make a prediction on), and the existing training examples.

2. Sort the data by the newly calculated distance so that the nearest training examples are first

3. Take the top `k` neighbors:
   - if the problem is a *classification*, **take the mode of the target variable** to find the most commonly appearing class and return that as your prediction
   - if the problem is a *regression*, **take the average of the target variables** for the k closest neighbors and return that as your prediction

Review: 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()
```

Review: 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

Review: 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.

Review: 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.

```
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
```

Review: Group Programming Exercise

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

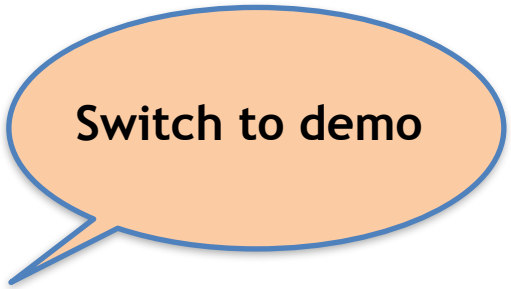
```
new_iris = {}
new_iris['petal length'] = 5.1
new_iris['sepal length'] = 7.2
new_iris['petal width'] = 1.5
new_iris['sepal width'] = 2.5

# 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]}')
```

Today's Agenda

- kNN implementation using scikit-learn library
- Data normalization — *z-score*

kNN implementation using scikit-learn library



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

Today's Agenda

- kNN implementation using scikit-learn library

- Data normalization — *z-score*