CS167: Machine Learning

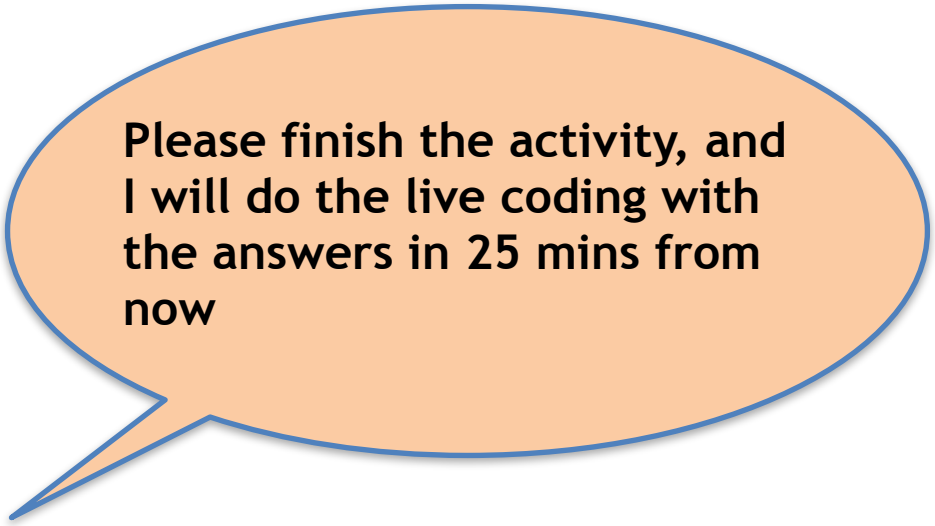
Weighted k-Nearest Neighbor (k-NN)
Graph Plot
Cross-validation (Train/Test Split)

February 23rd, 2026

Drake
UNIVERSITY

Review: kNN implementation using scikit-learn library

- I did the demo on last Monday; however, we didn't get a chance to finish the experiments with the kNN implementation using scikit-learn.



Review: last lecture's topics

- Data normalization — *z-score*

- Data normalization coding

Review: Computing the Z-Score

- For example: On the Titanic:
  - sex mean(0:male, 1:female): 0.35
  - sex standard deviation: 0.48
  - age mean: 29.7
  - age standard deviation: 13

$$Z - score : \frac{x_i - \mu}{\sigma}$$

	sex	age
example 1	1	50
example 2	0	48

	sex	age
example 1	1	50
example 3	1	25

Z-Score for male: $(0 - 0.35)/0.48 \approx -0.73$
Z-Score for female: $(1 - 0.35)/0.48 \approx 1.35$

Z-Score for age 50: $(50 - 29.7)/13 \approx 1.56$
Z-Score for age 48: $(48 - 29.7)/13 \approx 1.41$
Z-Score for age 25: $(25 - 29.7)/13 \approx -0.36$

Review: Computing the Z-Score on Titanic

- Now that we have the data as 1s and 0s, let's calculate the mean and standard deviation

```
s_mean = titanic.sex.mean()
s_std = titanic.sex.std()

#replace column with each entry's z-score
titanic.sex = (titanic.sex - s_mean)/s_std
titanic.head()
```

$$Z - score : \frac{x_i - \mu}{\sigma}$$

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	-0.734928	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	1.359146	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	1.359146	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	1.359146	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	-0.734928	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Today's Agenda

- Weighted k-NN
- Graph Plot
- Cross validation (train/test split)

Quick Review: 3-Nearest Neighbor (3-NN)

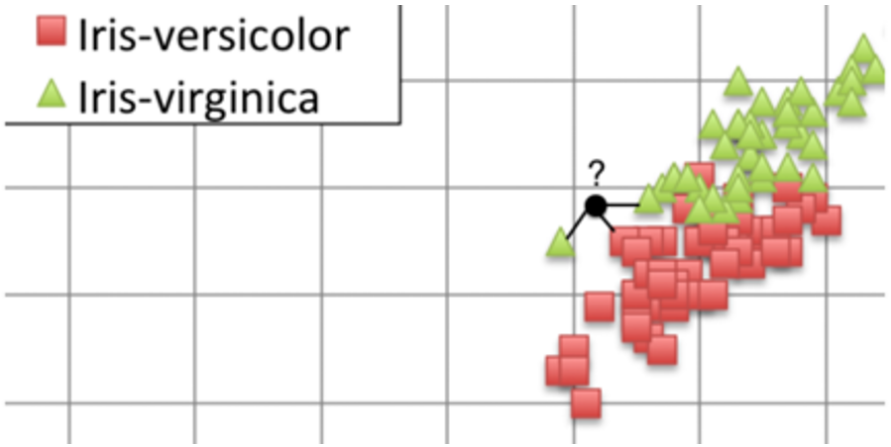
- **3-Nearest-Neighbor Algorithm:** predict the *most commonly appearing* class among the 3 closest training examples
  - In other words, k=3

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

  `Iris-versicolor`

  `Iris-virginica`

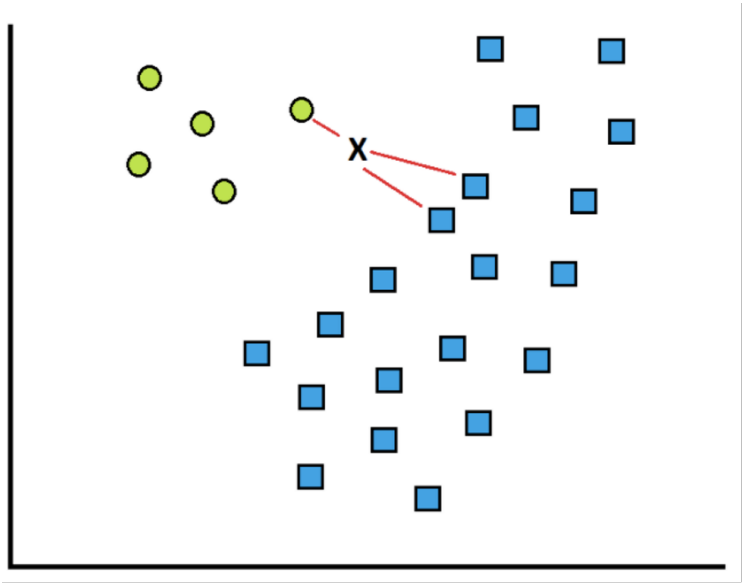
- What class will a **3NN** algorithm predict?



k-Nearest Neighbor (k-NN)

- The way we've learned **k-Nearest-Neighbor (k-NN)** so far, each neighbor gets an equal vote in the decision of what to predict.

- Do we see any problems with this? If so, what?

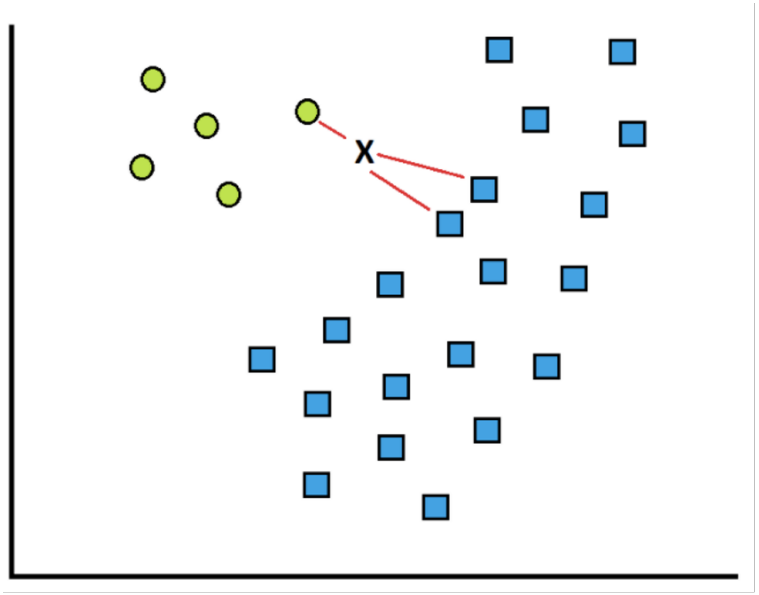


- Should neighbors that are closer to the new instance get a larger share of the vote?

Weighted k-NN Intuition

- In weighted kNN, the nearest k points are given a weight, and the weights are grouped by the target variable. The class with the largest sum of weights will be the class that is predicted

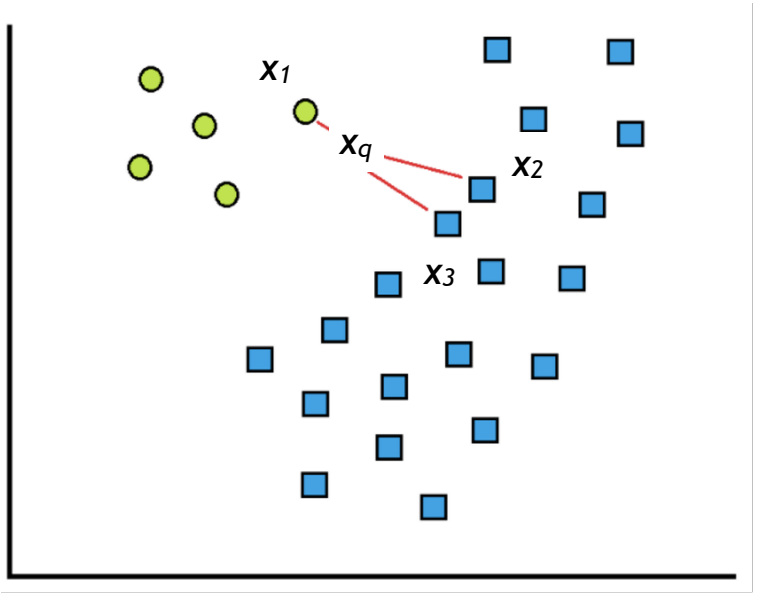
- The intuition is to give more weight to the points that are nearby and less weight to the points that are farther away.
  - distance-weighted voting



Weighted k-NN Intuition

- In **w-kNN**, we want to predict the target variable with the most weight, where the weight is defined by the inverse distance function

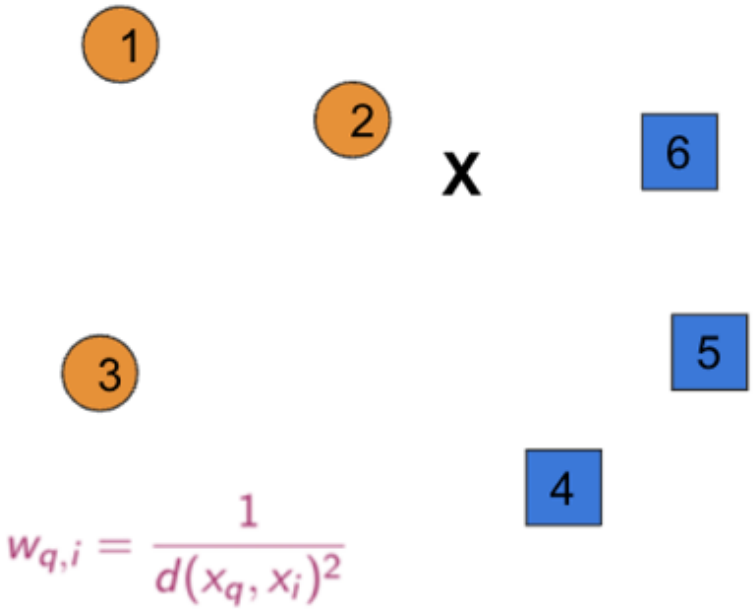
$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$



- In English, you can read that as the **weight** of a training example is equal to 1 divided by the distance between the new instance and the training example squared

Weighted k-NN Example: Step 1

- Start by calculating the distance between the new example ***X***, and each of the other training examples:

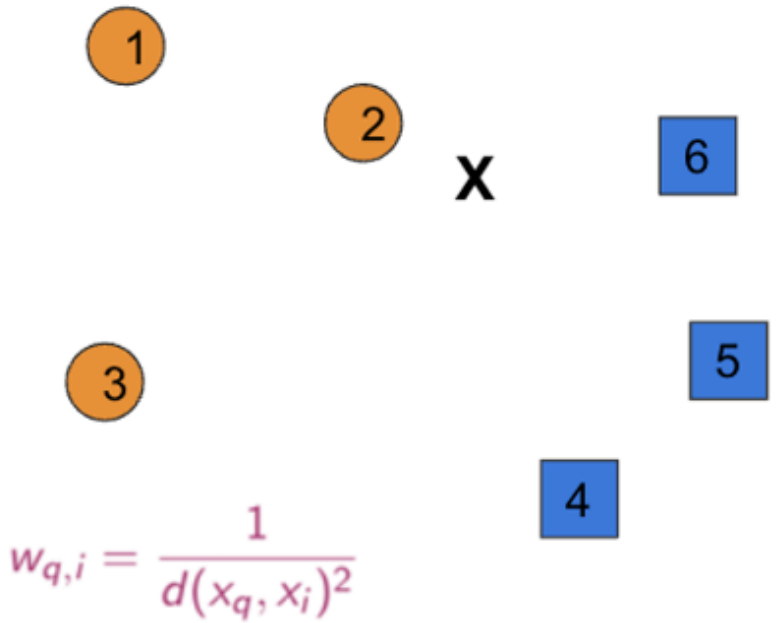


$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

Example #	Distance	Weight
1	5	
2	1	
3	7	
4	5	
5	4	
6	3	

Weighted k-NN Example: Step 2

- Then, **calculate the weight** of each training example using the inverse distance squared.

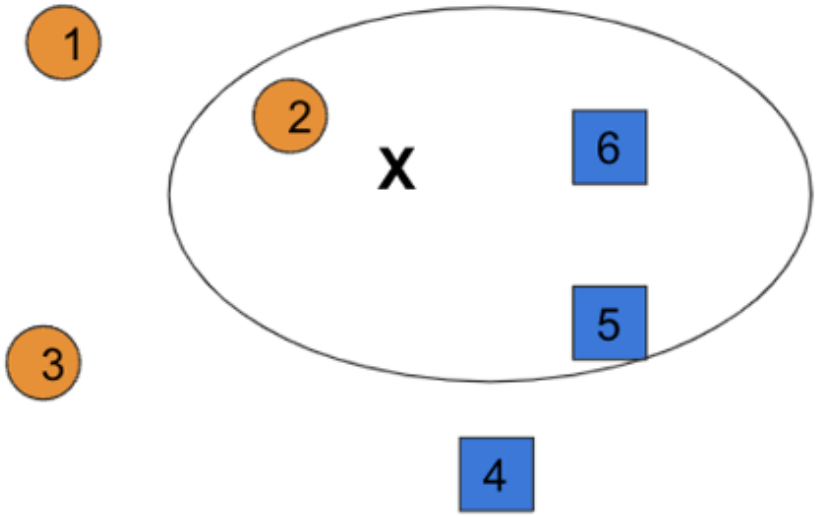


$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

Example #	Distance	Weight
1	5	1/25
2	1	1
3	7	1/49
4	5	1/25
5	4	1/16
6	3	1/9

Weighted k-NN Example: Step 3

- Find the k closest neighbors — let's assume **k=3** for this example:

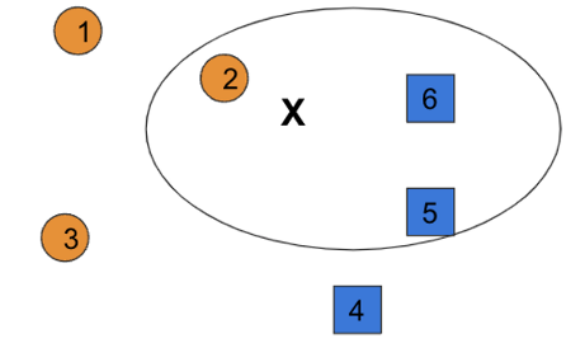


$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

Example #	Distance	Weight
1	5	1/25
2	**1**	**1**
3	7	1/49
4	5	1/25
5	**4**	**1/16**
6	**3**	**1/9**

Weighted k-NN Example: Step 4

- Then, sum the weights for each possible class:
  - Orange: 1
  - Blue: 1/16 + 1/9 = 0.115
- What would a **normal 3NN** predict?
- What would a **Weighted 3NN** predict?



$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

Example #	Distance	Weight
1	5	1/25
2	**1**	**1**
3	7	1/49
4	5	1/25
5	**4**	**1/16**
6	**3**	**1/9**

Code: weighted kNN

- Import the same module `Neighbors` from `sklearn`
  - Instead of `KNeighbors(n_neighbors=5)`, invoke it using an extra parameter `KNeighbors(n_neighbors=5, weights='distance')`

- Use the iris measurements (predictors and target), and try out different k as parameters

Group Exercise

- Group Exercise: talk to your neighbors and work on the group activity after I finish my demo
    - Link to https://github.com/alimoorreza/CS167-sp26-notes/blob/main/Day09_weighted_knn.ipynb

Today's Agenda

- Weighted k-NN
- Graph Plot
- Cross validation (train/test split)

Graph Plot

- Use markers and line styles to differentiate your series:

Markers

character	description
`'.'`	point marker
`','`	pixel marker
`'o'`	circle marker
`'v'`	triangle_down marker
`'^'`	triangle_up marker

Line Styles

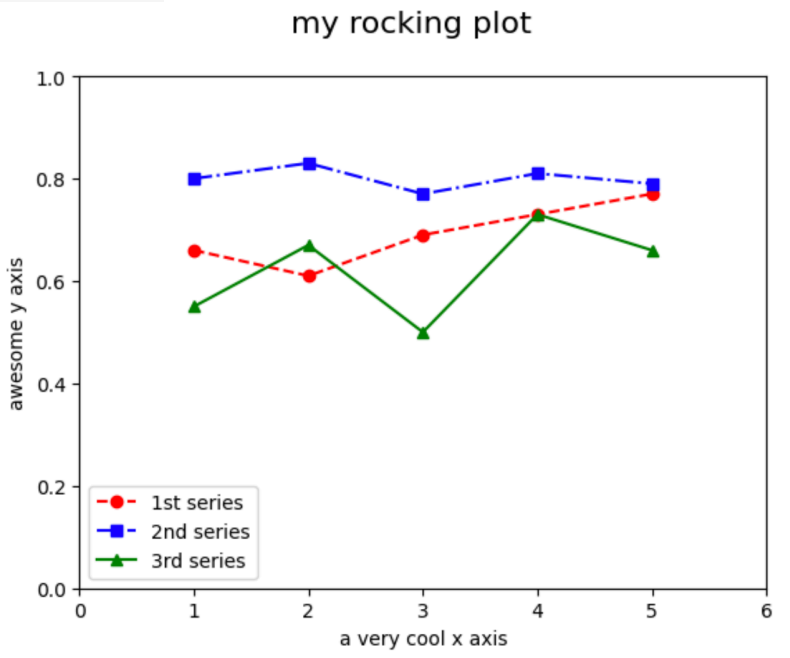
character	description
`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style

```
'b'    # blue markers with default shape
'or'   # red circles
'-g'   # green solid line
'--'   # dashed line with default color
'^k:'  # black triangle_up markers connected by a dotted line
```

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Graph Plot: example#1

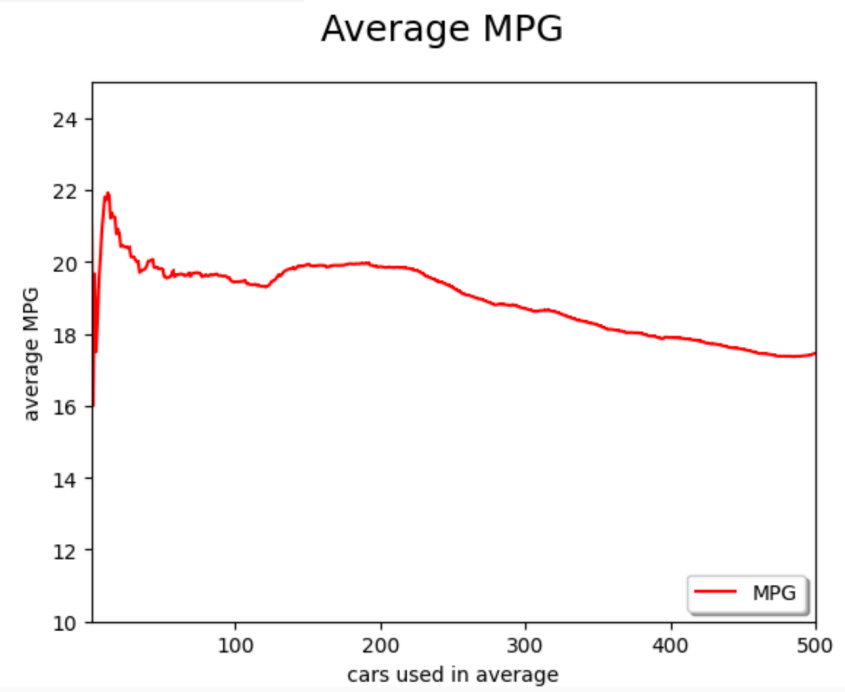
```
import matplotlib.pyplot as plt
%matplotlib inline

#define our data
xvals   = [1,2,3,4,5]
series1 = [0.66,0.61,0.69,0.73,0.77]
series2 = [0.8,0.83,0.77,0.81,0.79]
series3 = [0.55,0.67,0.5,0.73,0.66]

#add titles to axis and graph
plt.suptitle('my rocking plot', fontsize=16)
plt.xlabel('a very cool x axis')
plt.ylabel('awesome y axis')

#plot the data
plt.plot(xvals, series1, 'ro--', label='1st series')
plt.plot(xvals, series2, 'bs-.', label='2nd series')
plt.plot(xvals, series3, 'g^-',  label='3rd series')
plt.legend() #plt.legend(loc='lower right', shadow=True)
plt.axis([0,6,0,1]) #[x_min, x_max, y_min, y_max]
plt.show()
```



https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Graph Plot: example#2

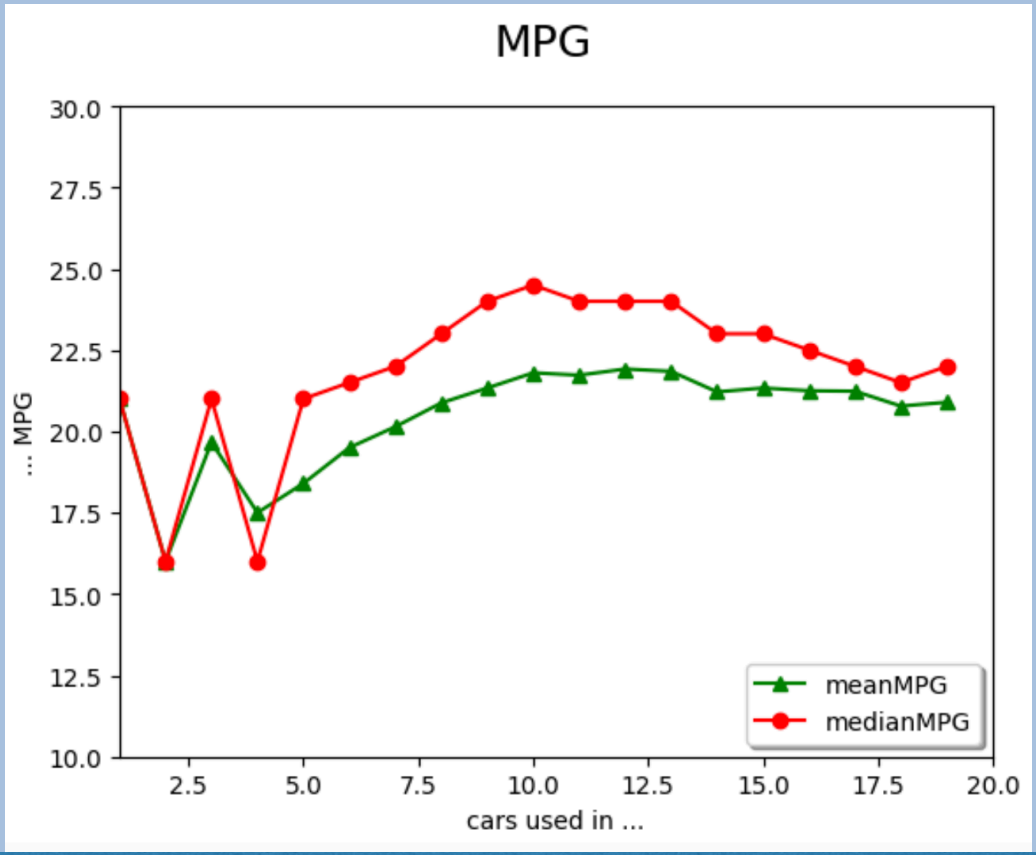
```
gas_vehicles = data[data['fuelType']=='Regular']

# a silly function that returns the average MPG for the first k cars in the df
def getAverageMPG(data, k):
    return data["comb08"].iloc[0:k].mean()

number_of_points = 500

#populate the series list
series = []
for i in range(1, number_of_points):
    val = getAverageMPG(gas_vehicles, i)
    series.append(val)

#plot it!
xvals = range(1, number_of_points)
plt.suptitle('Average MPG', fontsize=18)
plt.xlabel('cars used in average')
plt.ylabel('average MPG')
plt.plot(xvals, series, 'r,-', label='MPG')
plt.legend(loc='lower right', shadow=True)
plt.axis([1, number_of_points, 10,25])
plt.show()
```



https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Take home activity (due on 02/25 by midnight)

- Given the code from the previous slide
  - change the number of points to 20
  - change the line to green triangles
  - also plot the median (red dots)



Today's Agenda

- Weighted k-NN
- Graph Plot
- Cross validation (train/test split)

How do we know if our model is a 'good' model?

- We want to know how good our models are at making predictions... how can we test it? Examples:
  - what k-value should we use in kNN algorithm?
  - what is the effect on accuracy if I normalize the data?
  - should I use a weighted kNN algorithm or a normal kNN?

Evaluation of Machine Learning Algorithms

- We want to know how good our model is at making predictions. How can we test it?

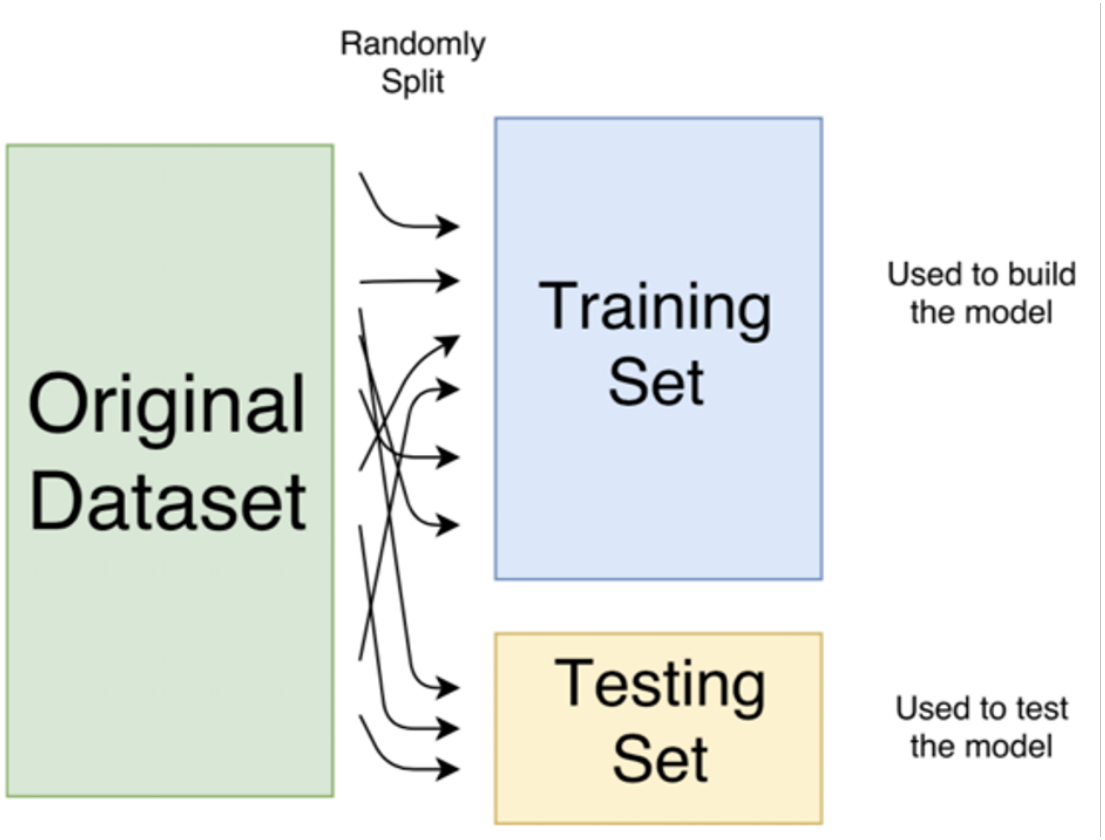
- **Option 1:** Deploy the model in a live setting and see how it does on new examples

- **Option 2:** Run each of our training examples through the model and see how many it gets correct

- **Option 3:** Cross-Validation - set aside some of your training examples to be used for testing
  - don't use testing examples when you train the model, only the rest that were left over. Why?

Cross-Validation

- Don't train the model on the testing data!

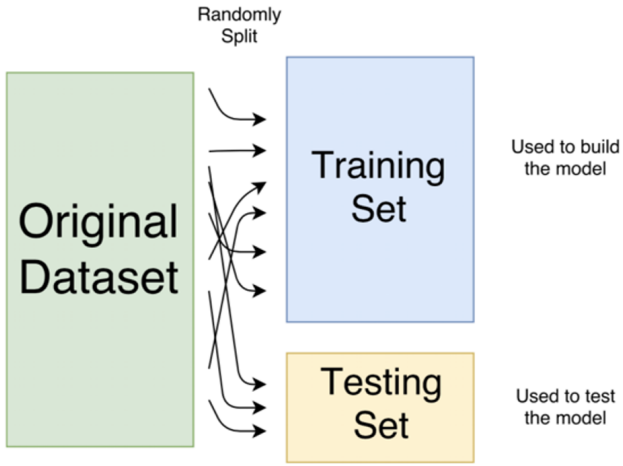


Cross-Validation Code

- A good rule of thumb is that we like to train our model with 80% of the given data examples (training set), and test it on 20% of the given data examples (testing set)

- Splitting datasets into training and testing sets with a Pandas DataFrame:

```
import pandas as pd
import numpy as np


#shuffle the iris "sampling" the full set in random order
shuffled_data = iris.sample(frac=1, random_state=41)

# set up training and testing set
test_data = shuffled_data.iloc[0:20] #test on the first 20 rows of shuffled
train_data = shuffled_data.iloc[20:] #train on the rest
train_data.shape
```



Cross-Validation Metrics

- When doing cross-validation, how do we tell how well our model performed?

- How can we measure it?
  - depends on the task and what we want to know

- What metrics to use for classification and regression?
  - The output variable in **classification** is categorical (or discrete).
  - The output variable in **regression** is numerical (or continuous).