CS167: Machine Learning

Graph Plot
Evaluation Metrics

Thursday, February 22nd, 2024

Announcements

- Quiz #1
  - released
  - will be due next Wednesday 02/28 by 11:59am (noon)

Assignments

Anything that you will turn in will live in this folder.

Attendance
Due date: 5/17/24, 12:00 PM

The score represents the percentage of time the student was physically present in class (max score is 100%]. Attendance was recorded on physical paper, with students adding their signature on the provided sheet.

Quiz #1: Foundations of Machine Learning
Due date: 2/28/24, 11:59 AM

Notebook 0 : Onboarding
Due date: 2/1/24, 11:59 PM

In-class activity#1
Due date: 2/6/24, 12:00 PM

Announcements

- Notebook #3: Cross Validation
    - released
    - due next Thursday 02/29 by 11:59pm
    - to submit, download the `ipynb` file from Colab

Before we get started, let's load in our datasets:

- Make sure you change the path to match your Google Drive.
  - Load the `vehicle.csv` file from your Google Drive

```
[2] #run this cell if you're using Colab:
    from google.colab import drive
    drive.mount('/content/drive')
```

```
#import the data:
#make sure the path on the line below corresponds to the path where you put your
import pandas as pd
import numpy as np
data = pd.read_csv('/content/drive/MyDrive/cs167_fall23/datasets/vehicles.csv')
pd.set_option('display.max_columns', 100)
iris = pd.read_csv('/content/drive/MyDrive/cs167_fall23/datasets/irisData.csv')
```

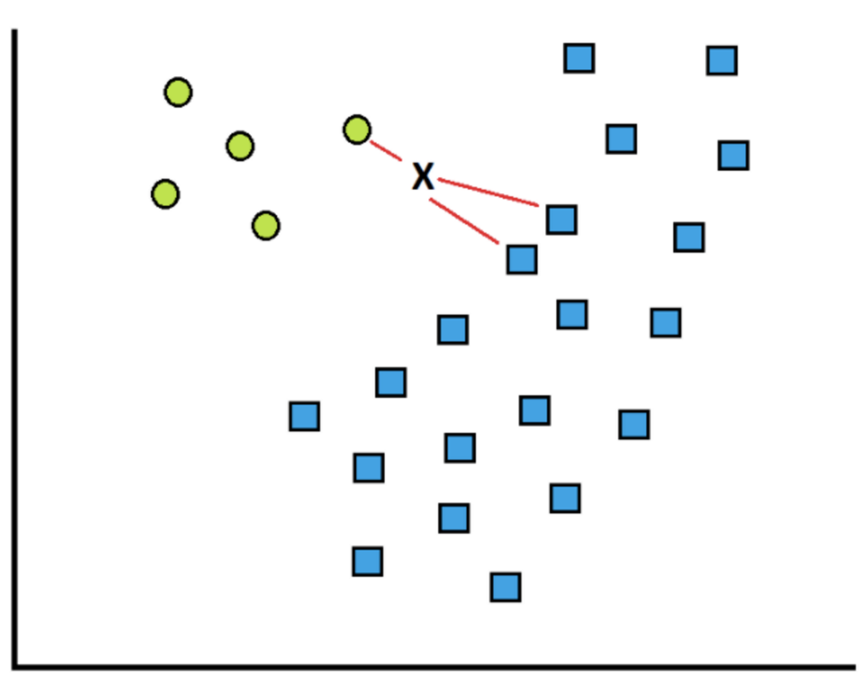
Today's Agenda

- Review: Weighted k-NN
- Graph Plot
- Evaluation Metrics
  - Classification metrics
  - Regression metrics

Quick Review: 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.

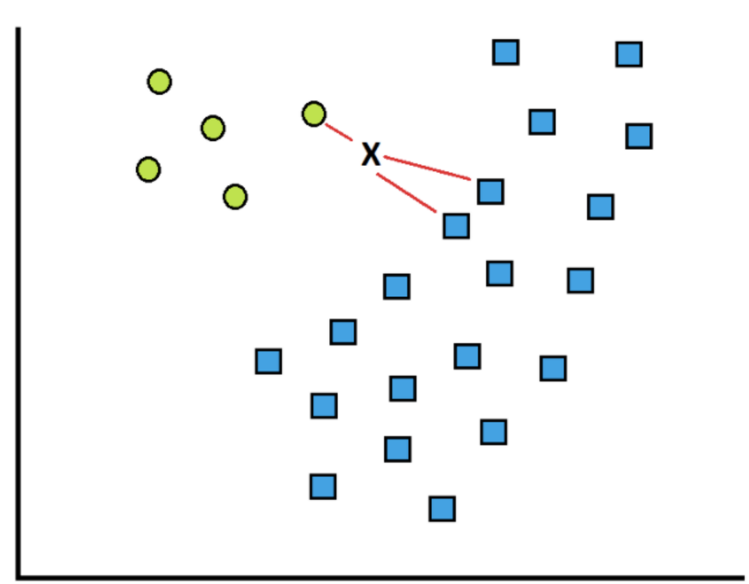
- Do we see any problems with this? If so, what?



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

Quick Review: Weighted k-NN Intuition

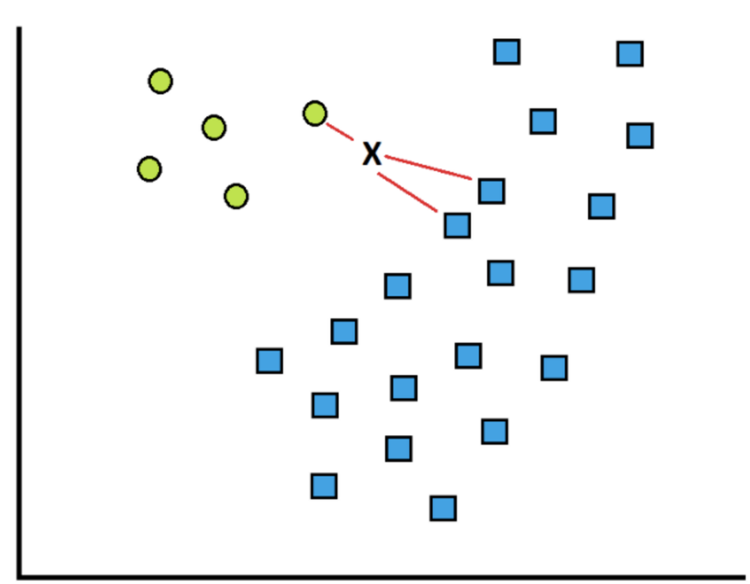
- In weighted kNN, the nearest k points are given a weight, and the weights are grouped by the target variable. The class with the largest sum of weights will be the class that is predicted
- The intuition is to give more weight to the points that are nearby and less weight to the points that are farther away.
  - distance-weighted voting



Quick Review: 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

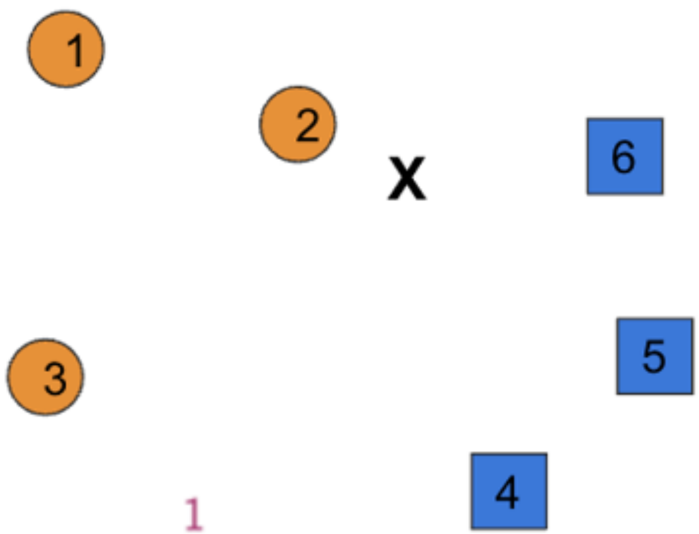
$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$



- 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

Quick Review: Weighted k-NN Example: Step 1

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

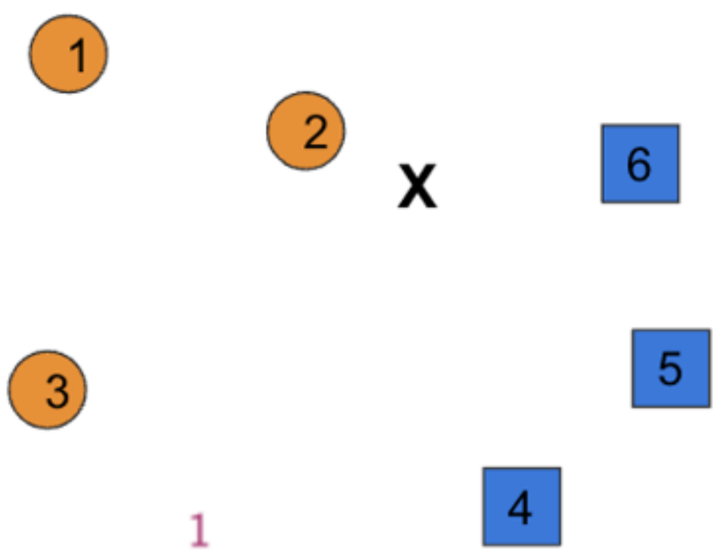


$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

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

Quick Review: Weighted k-NN Example: Step 2

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

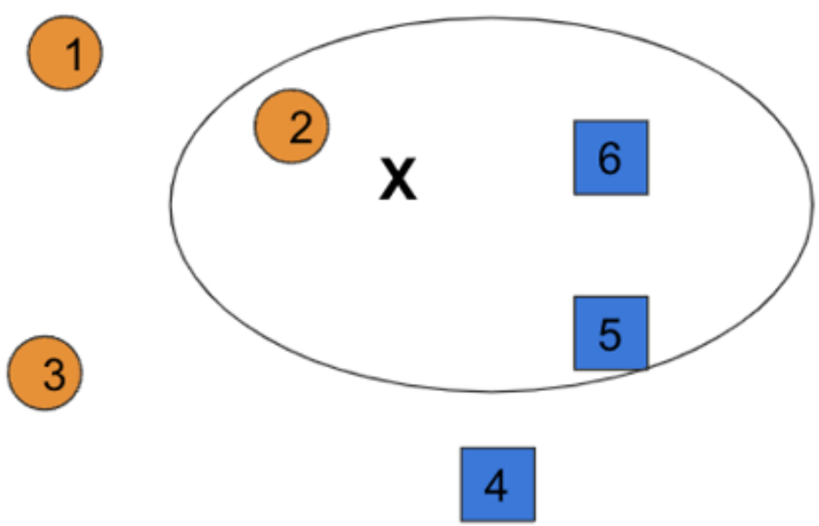


$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

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

Quick Review: Weighted k-NN Example: Step 3

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



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

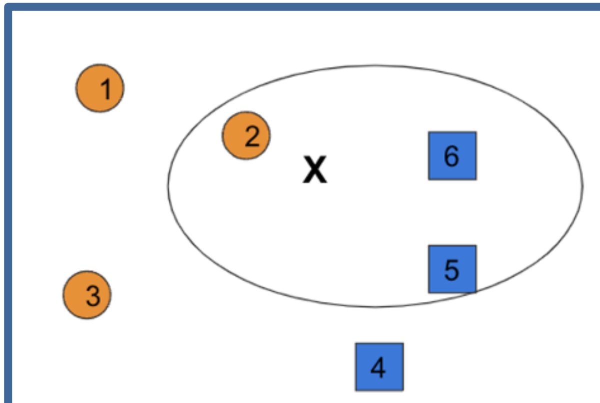
Example #	Distance	Weight
1	5	1/25
2	**1**	**1**
3	7	1/49
4	5	1/25
5	**4**	**1/16**
6	**3**	**1/9**

Quick Review: 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**

Quick Review: Programming Exercise #3

- Write a new function `weighted_knn()`
- Pass the iris measurements (specimen), data frame, and k as parameters and return the predicted class

```
import numpy as np

def weighted_knn(specimen, data, k):

  # step 1: calculate the distances from 'specimen' to all other samples in 'data'
  data['distances'] = np.sqrt( (specimen['petal length'] - data['petal length'])**2 +
                               (specimen['sepal length'] - data['sepal length'])**2 +
                               (specimen['petal width']  - data['petal width'])**2 +
                               (specimen['sepal width']  - data['sepal width'])**2 )

  # step 2: calculate the weights for each sample (remember, weights are 1/d^2)
  # data['weights']    = ... (TBD)

  # step 3: find the k closest neighbors as follows
  # first: sort the data and take the first k samples as neighbors
  sorted_data         = data.sort_values(['distances'])
  print('Nearest k samples in the training data:')
  neighbors           = sorted_data.iloc[0:k]
  # second: use groupby to sum the weights of each species in the closest k
  # TBD


  # third: return the class that has the largest sum of weight.
  # TBD
```

Today's Agenda

- Weighted k-NN
- Graph Plot
- Evaluation Metrics
  - Classification metrics
  - Regression metrics

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

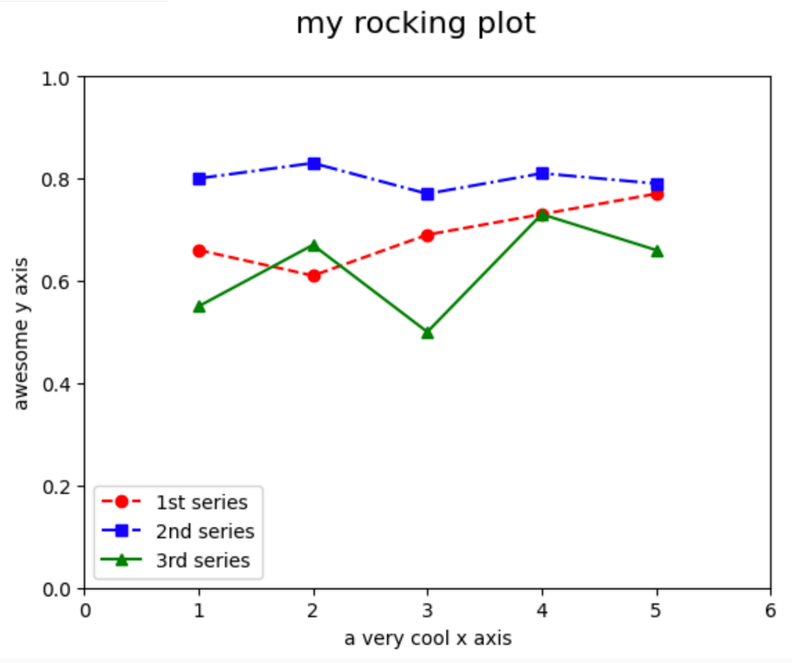
Line Styles

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
```

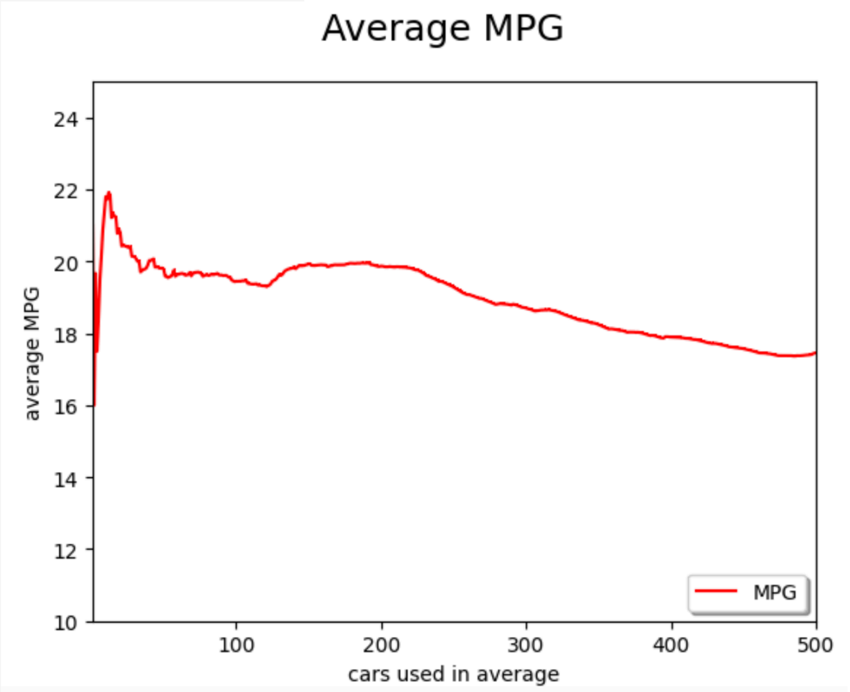
Graph Plot: example#1

```
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()
```



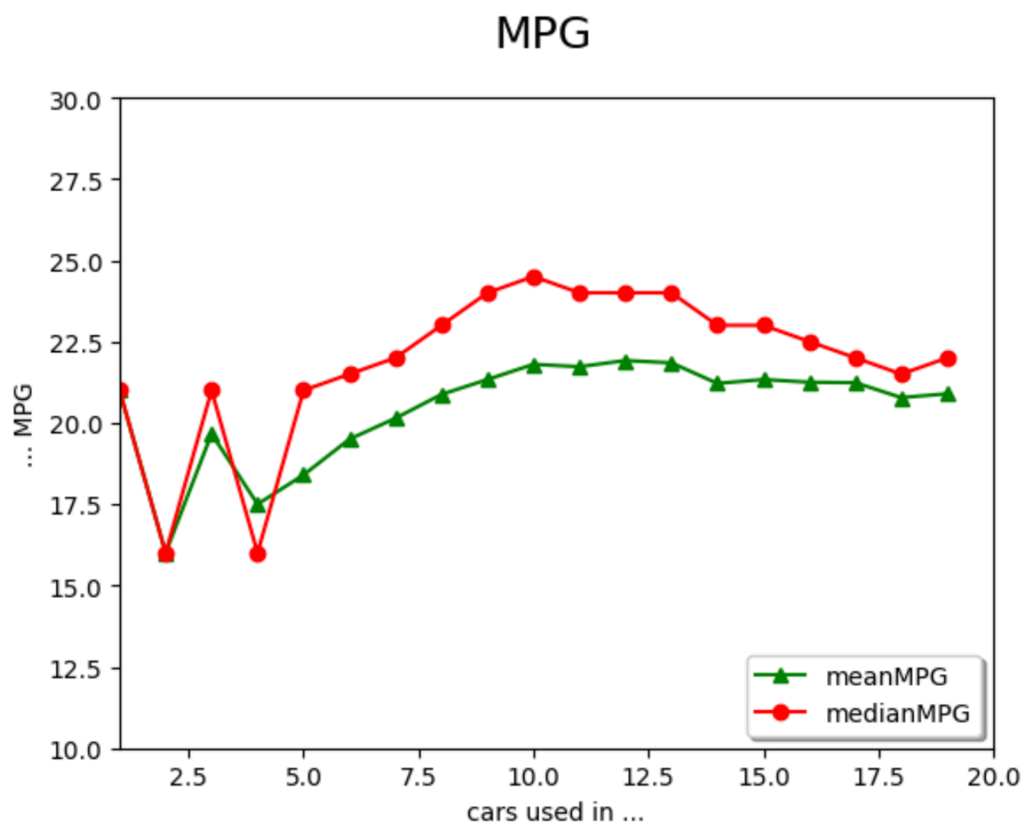
Graph Plot: example#2

```
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()
```



Group Exercise

- 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
- Evaluation Metrics
  - Classification metrics
  - Regression metrics

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?

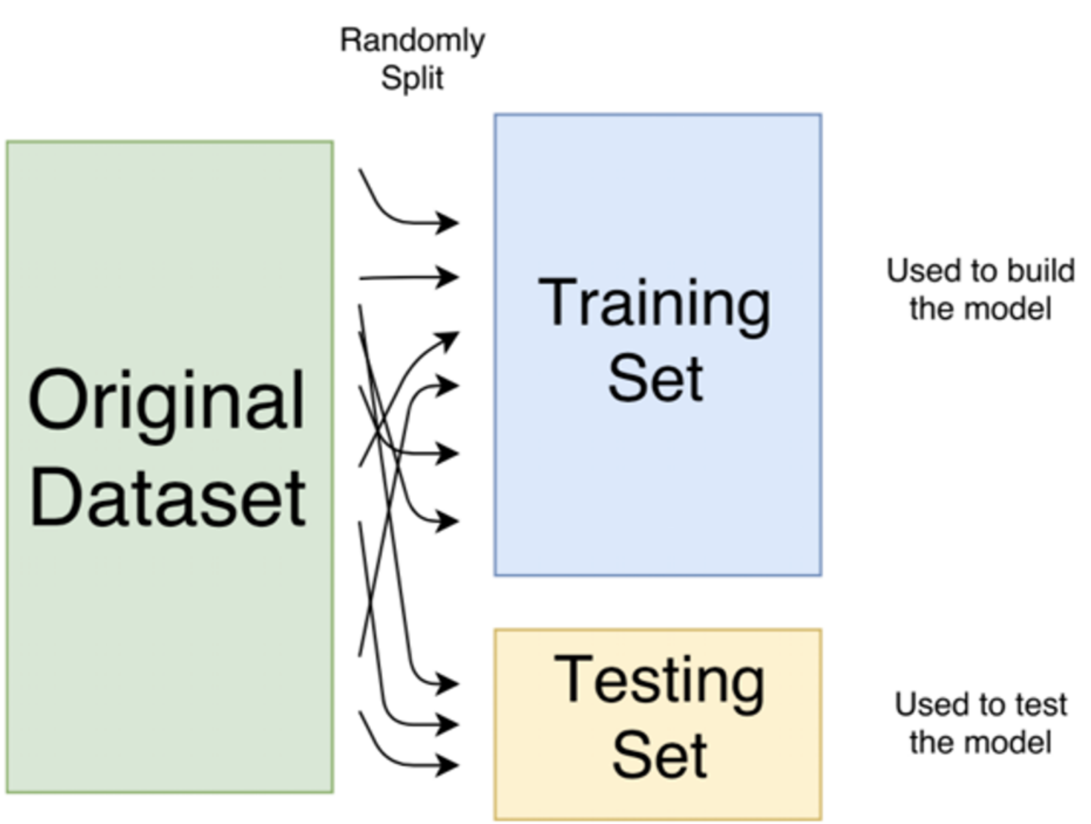
- **Option 1:** Deploy the model in a live setting and see how it does on new examples

- **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!

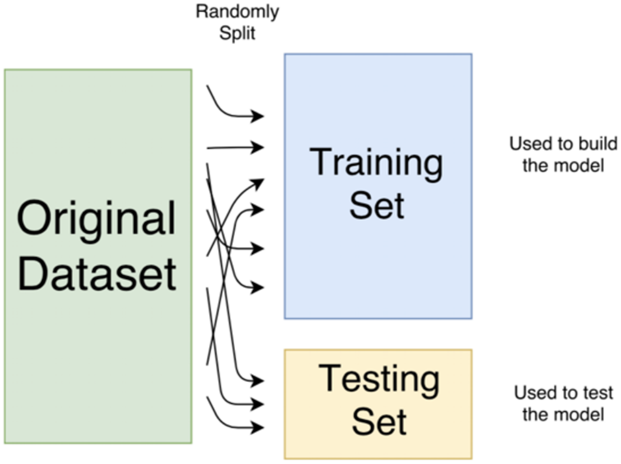


Cross-Validation Code

- 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 **regression** is numerical (or continuous).
  - The output variable in **classification** is categorical (or discrete).

Today's Agenda

- Weighted k-NN
- Graph Plot
- Evaluation Metrics
  - Classification metrics
  - Regression metrics

Classification metrics

- **Accuracy**: The fraction of test examples your model predicted correctly
  - *Example*: 17 out of 20 = 0.85 accuracy

- Issues with accuracy: suppose that a blood test for cancer has 99% accuracy
  - *can we safely assume this is a really good test?*
    - If the dataset is *unbalanced*, accuracy is not a reliable metric for the real performance of a classifier because it will yield misleading results
    - **Example**: Most people don’t have cancer
  - Beware of what your metrics don't tell you

Classification metrics

- **Accuracy**: The fraction of test examples your model predicted correctly
  - *Example*: 17 out of 20 = 0.85 accuracy

- Issues with accuracy: What about false negatives and false positives?
  - **false positives**: a test result which incorrectly indicates that a particular condition or attribute is present
  - **false negative**: a test result which incorrectly indicates that a particular condition or attribute is absent

		Predicted class	
		P	*N*
Actual Class	*P*	5	3
	N	2	90

Classification metrics: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm.

- Each row represents instances in an actual class

- While each column represents the instances in a predicted class
  - It makes it easy to see where your model is confusing the predicted and actual results. For a binary classification problem:

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Classification metrics: `Confusion Matrix`

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP)**	**False negative (FN)**
	Negative (N)	**False positive (FP)**	**True negative (TN)**

- **Confusion matrix:**
    - Each row represents instances in an actual class
    - While each column represents the instances in a predicted class

- To build the confusion matrix let's map the actual classifications and predicted classifications using the following flat table:

Individual Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0
Result												

Exercise: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm

Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0
Result												

- Given the following confusion matrix:
  - how many true positive? 6
  - how many true negatives? 3
  - how many false positive? 1
  - how many false negatives? 2

Exercise: Confusion Matrix

- **Confusion matrix:** A specific table layout that allows the visualization of the performance of an algorithm

Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0
Result	FN	FN	TP	TP	TP	TP	TP	TP	FP	TN	TN	TN

- Given the following confusion matrix:
  - how many true positive? 6
  - how many true negatives? 3
  - how many false positive? 1
  - how many false negatives? 2

Summarize the Results in Confusion Matrix

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

- Given the following confusion matrix:
  - how many true positive?
  - how many true negatives?
  - how many false positive?
  - how many false negatives?
  - what is the accuracy?

		Predicted class	
		P	*N*
Actual Class	*P*	6	2
	N	1	3

Classification metrics: `Confusion Matrix`

- For a multi-class (more than 2) classification problem:
  - the confusion matrix looks like below where each row represents instances in an actual class; while each column represents the instances in a predicted class