

CS167: Machine Learning

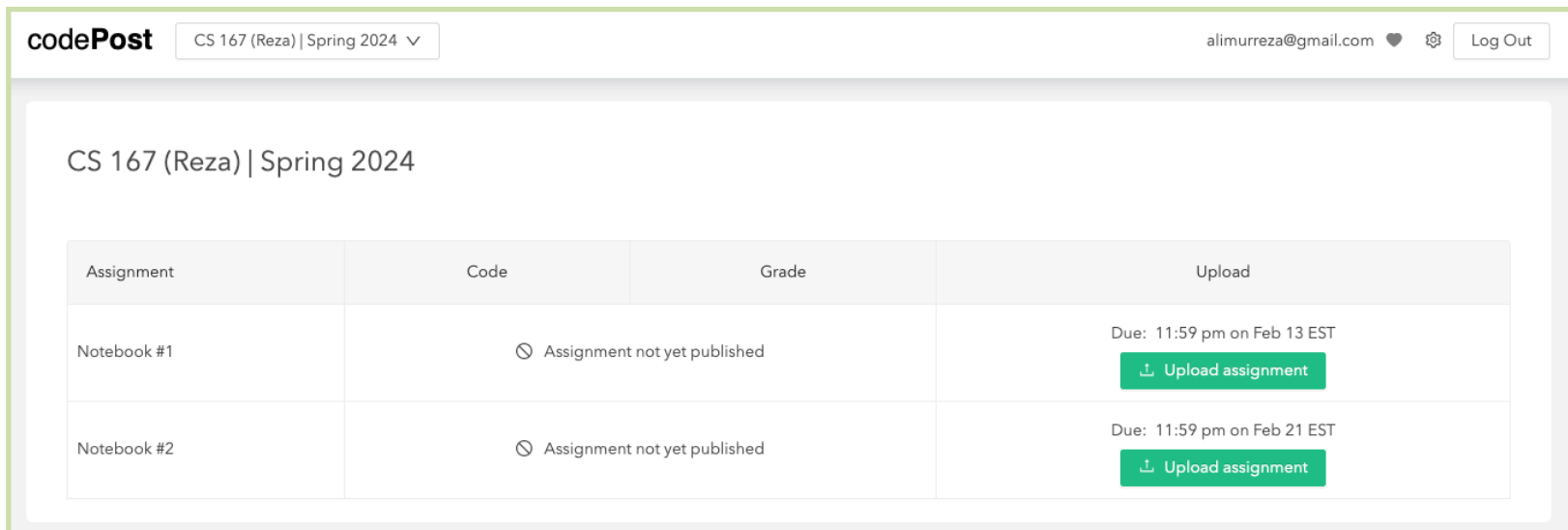
Normalization
Weighted k-Nearest Neighbor (k-NN)

Tuesday, February 20th, 2024



Announcements

- Notebook #2: kNN and Normalization
 - to submit, download the .ipynb file from Colab
 - directly upload to CodePost
 - due tomorrow – Wednesday 02/21 by 11:59pm



codePost CS 167 (Reza) | Spring 2024 alimurreza@gmail.com Log Out

CS 167 (Reza) | Spring 2024

Assignment	Code	Grade	Upload
Notebook #1	⊘ Assignment not yet published		Due: 11:59 pm on Feb 13 EST Upload assignment
Notebook #2	⊘ Assignment not yet published		Due: 11:59 pm on Feb 21 EST Upload assignment

Announcements

- Heads up that Quiz #1
 - will be released today (02/20) at 3:15 pm
 - will be due next Tuesday 02/27 by 11:59pm
 - contains some question which will be covered this week eg, evaluation metric, confusion matrix, cross validation, etc
 - Types of questions:
 - MCQ
 - True/False
 - Fill in the blanks (may or may not require calculations)

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

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

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

```
import pandas as pd  
import numpy as np  
  
path = '/content/drive/MyDrive/cs167_fall23/datasets/penguins_size.csv'  
penguins = pd.read_csv(path)  
penguins.head()  
  
path1 = '/content/drive/MyDrive/cs167_fall23/datasets//irisData.csv'  
iris = pd.read_csv(path1)  
iris.head()  
  
path2 = '/content/drive/MyDrive/cs167_fall23/datasets/titanic.csv'  
titanic = pd.read_csv(path2)  
titanic.head()
```

Quick Review: Missing Data

- Most datasets you will work with will not be in perfect shape
 - you'll need to **clean** the data before you can run any machine learning algorithms on it
- Missing data is a pretty common thing – so much so that there's a special value for missing data:
 - **NaN**, or **not a number**

Quick Review: Missing Data

- The steps of cleaning data normally include:
 - Step 1: Identifying which columns have missing data `df.isna().any()`
 - Step 2: Determining how much data is missing in each column
 - `df.col_missing_data.value_counts(dropna=False)`
 - Step 3: Deciding what to do with the missing data:
 - drop it: `dropna()`
 - remember to either save the returned result
 - `result = df.whatever_column.dropna()`, or use `df.whatever_column.dropna(inplace=True)`
 - fill it: `fillna()`
 - let it be

Quick Review: Identifying Missing Data

```
titanic.loc[0:4]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton

- Now, let's call `isna()`, and see what we get as an output

```
titanic.loc[:4].isna()  
#look at the 'deck' column...
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town
0	False	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	True	False

Quick Review: How much data is missing?

- Let's apply `value_counts()` on the various columns (eg, deck) of Titanic dataset

If you don't do this, it won't show the NaN counts

```
titanic.deck.value_counts(dropna=False)
#688 missing values
```

NaN	688
C	59
B	47
D	33
E	32
A	15
F	13
G	4

Name: deck, dtype: int64

https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html

Quick Review: Fill it using fillna ()

- What do you think we should use to fill in the missing data in the age column?
 - we probably don't want to throw off our statistics...

```
▶ print("before: ", titanic['age'].isna().any())
age_mean = titanic['age'].mean()
titanic['age'].fillna(age_mean, inplace=True)
print("after: ", titanic['age'].isna().any())
titanic.head(7)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.000000	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.000000	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.000000	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.000000	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.000000	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
5	0	3	male	29.699118	0	0	8.4583	Q	Third	man	True	NaN	Queenstown	no	True
6	0	1	male	54.000000	0	0	51.8625	S	First	man	True	E	Southampton	no	True

Quick Review: Missing Data Functions

- `isna()`: returns True for any missing data
- `notna()`: returns True for any data that is **not** NaN
- `any()`: returns true if any of the elements in a Series is True
- `value_counts()`: returns a list of the values in a Series, use `dropna=False` to see NaN values
- `dropna()`: drops rows or columns (specify which axis, 1 or 0) that have missing data. Don't forget to either save the result of the call or add `inplace=True` as a parameter
- `fillna()`: replaces missing data with a given value (generally 0 or the mean)

Review Exercise

- Take care of the missing data in the penguin dataset.

```
[6] # step 1: identify which columns are missing data
```

```
[7] # step 2: determine how much data is missing from each column that is missing data
```

```
[8] # step 3: decide whether to drop, fill, or leave it
```

Today's Agenda

- Topics:
 - Normalization
 - Weighted k-NN

Normalization Motivation

- In datasets that have numeric data, the columns that have the largest magnitude will have a greater 'say' in the decision of what to predict
- In the penguin dataset, `body_mass_g` will have a much bigger say in the prediction than the other options

```
penguins.head()
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0
3	Adelie	Torgersen	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0

Normalization

- Normalizing data:
 - rescale attribute values so they're about the same
 - adjusting values measured on different scales to a common scale

A Simple Normalization:

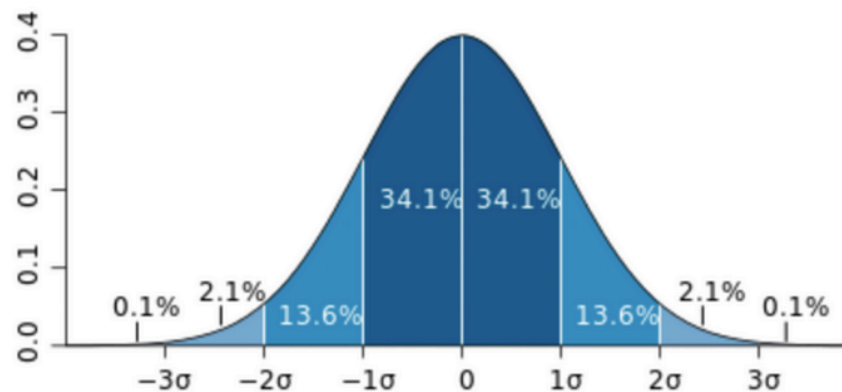
- One simple method of normalizing data is to replace each value with a proportion relative to the max value.
- For example, the oldest person on the [Titanic dataset](#) was 80, so:

age	replaced by
80	$80/80 = 1$
50	$50/80 = 0.625$
48	$48/80 = 0.6$
25	$25/80 = 0.3125$
4	$4/80 = 0.05$

Z-Score: Another Normalization Method

- **Idea:** rather than normalize to proportion of max, normalize based on how many standard deviations they are away from the mean
- **Standard Deviation:** usually represented as σ (sigma), a kind of average distance from the mean value
 - a low standard deviation indicates that the values tend to be close to the mean
 - a high standard deviation indicates that the values are spread out over a wider range

Standard Deviation:



Standard Deviation Calculation

- **Standard Deviation:** usually represented as σ (sigma), a kind of **average distance from the mean value**
 - Find the mean, represented as μ : (pronounced as *mu*)
 - Then, for each number, subtract the mean and square the result
 - Then, find the mean of those squared differences
 - Take the square root of that and we are done

- Let μ be the mean, then standard deviation of x_1, x_2, \dots, x_N is:

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_N - \mu)^2}{N}}$$

Corrected Sample Standard Deviation

- **Bessel's correction** says that you should divide by $N-1$ instead of N when working with a sample (as we usually do in machine learning tasks), and your estimate will be a little less biased.

$$\sigma = \sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_N - \mu)^2}{N-1}}$$

Computing the Z-Score

- After computing the corrected sample standard deviation, to normalize, replace each value x_i with its z-score based on the mean (μ) and standard deviation (σ) of its column.

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

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
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: $(25 - 29.7)/13 \approx -0.36$

Distance Computation Before Normalization

	sex	age
example 1	1	50
example 2	0	48

distance: $\sqrt{(1 - 0)^2 + (50 - 48)^2} \approx 2.24$

	sex	age
example 1	1	50
example 3	1	25

distance: $\sqrt{(1 - 1)^2 + (50 - 25)^2} = 25$

age is overemphasized here in the distance calculation

Distance Computation After Normalization

	sex	age
example 1	1.35	1.56
example 2	-0.73	1.41

$$\begin{aligned} & \text{distance:} \\ & \sqrt{(1.35 - -0.73)^2 + (1.56 - 1.41)^2} \\ & \approx 2.09 \end{aligned}$$

	sex	age
example 1	1.35	1.56
example 3	1.35	-0.36

$$\begin{aligned} & \text{distance:} \\ & \sqrt{(1.35 - 1.35)^2 + (1.56 - -0.36)^2} \\ & = 1.92 \end{aligned}$$

Neither **sex** nor **age** is overemphasized here in the distance calculation

Computing the Z-Score on Titanic

- Called on a DataFrame, will replace values given in `to_replace` with `value`. Let's use this to make the `sex` column of the dataset numeric.

```
▶ titanic['sex'] = titanic['sex'].replace(to_replace='female', value=1)  
titanic['sex'] = titanic['sex'].replace(to_replace='male', value=0)  
titanic.head()
```

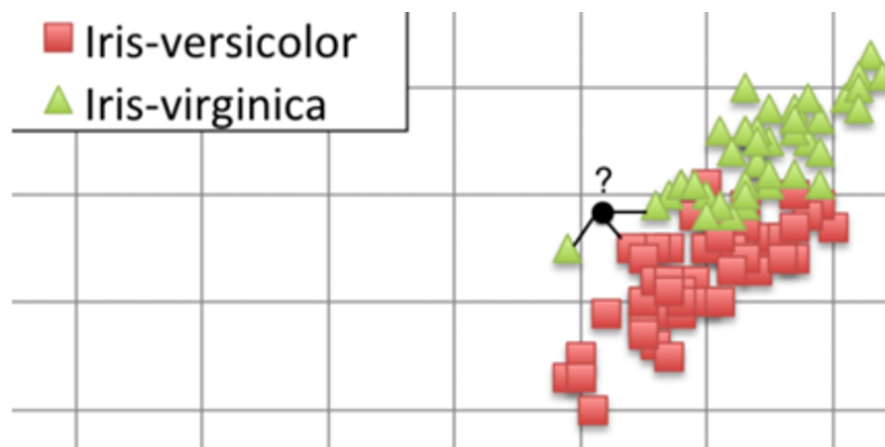
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	0	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	1	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	1	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True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

Programming Exercise #1

- Normalize each of the predictor columns in the iris dataset
 - Note: you need a way to transform the new reading (the specimen) that you will make the prediction on so that the new one and the training data will all be on the same scale. How can you do that?

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?



Code: k-Nearest Neighbor (kNN)

```
[ ] def knn(specimen, data, k):  
    # write your code in here to make this function work  
    # 1. calculate distances  
    data_copy = data.copy() #good practice to make a copy of the data  
    data_copy['distance_to_new'] = np.sqrt(  
        (specimen['petal length'] - data_copy['petal length'])**2  
        +(specimen['sepal length'] - data_copy['sepal length'])**2  
        +(specimen['petal width'] - data_copy['petal width'])**2  
        +(specimen['sepal width'] - data_copy['sepal width'])**2)  
  
    # 2. sort  
    sorted_data = data_copy.sort_values(['distance_to_new'])  
  
    # 3. predict  
    prediction = sorted_data.iloc[0:k]['species'].mode()[0]  
  
    #return prediction  
    return prediction
```

Code: k-Nearest Neighbor (kNN)

```
[ ] #what will you have to do here to make it work?  
  
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  
  
pred = knn(new_iris, iris, 15)  
print(pred)
```

Programming Exercise #1

- Repeat your kNN prediction code with the normalized data.
 - Does the value of k change the predictions?

Programming Exercise #2

- Normalize each of the predictor columns in the iris dataset
 - Write a function called `z_score()` that will take in a list of the names of the columns that you want to normalize, and the DataFrame, and will return a DataFrame where those columns have been z-score normalized.

```
def z_score(columns, data):  
    """  
    takes in a list of columns to normalize using the z-score method  
    Params:  
        columns, a list of columns to normalize  
        data, the dataframe, preferably a copy  
    Return:  
        a copy of the dataframe with the specified columns normalized  
    """  
    normalized_data = data.copy()  
  
    for col in columns:  
        # get the mean and std  
  
        # replace the column with the z-score  
  
    return normalized_data
```

Programming Exercise #2

- The function `z_score()` can be called as follows with the corresponding sample outputs

```
iris_norm = z_score(['sepal length', 'sepal width', 'petal width', 'petal length'], iris)
iris_norm.head()
```

	sepal length	sepal width	petal length	petal width	species
0	-0.897674	1.028611	-1.336794	-1.308593	Iris-setosa
1	-1.139200	-0.124540	-1.336794	-1.308593	Iris-setosa
2	-1.380727	0.336720	-1.393470	-1.308593	Iris-setosa
3	-1.501490	0.106090	-1.280118	-1.308593	Iris-setosa
4	-1.018437	1.259242	-1.336794	-1.308593	Iris-setosa

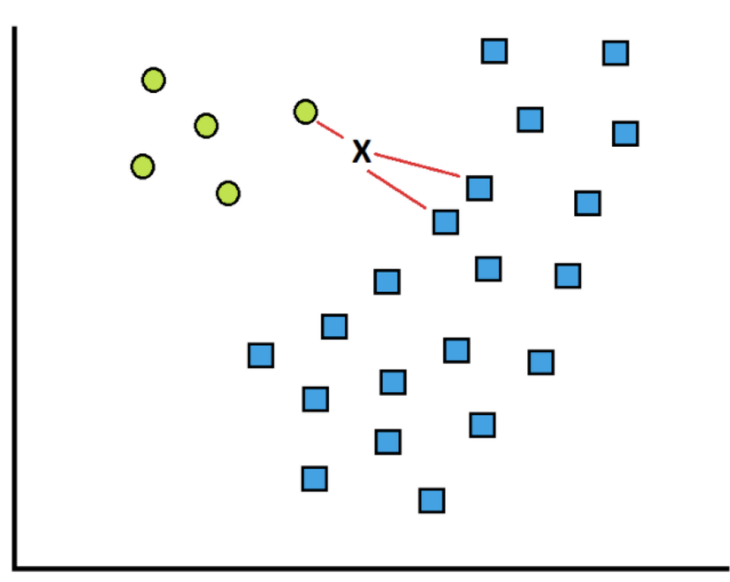


Today's Agenda

- Topics:
 - Normalization
 - Weighted k-NN

k-Nearest Neighbor (k-NN)

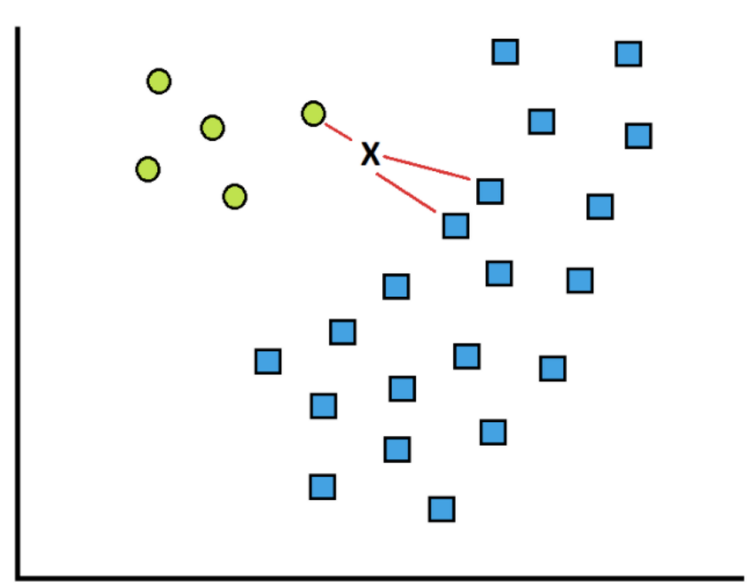
- The way we've learned **k-Nearest-Neighbor (k-NN)** so far, each neighbor gets **an equal vote** in the decision of what to predict.
- Do we see any problems with this? If so, what?



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

Weighted k-NN Intuition

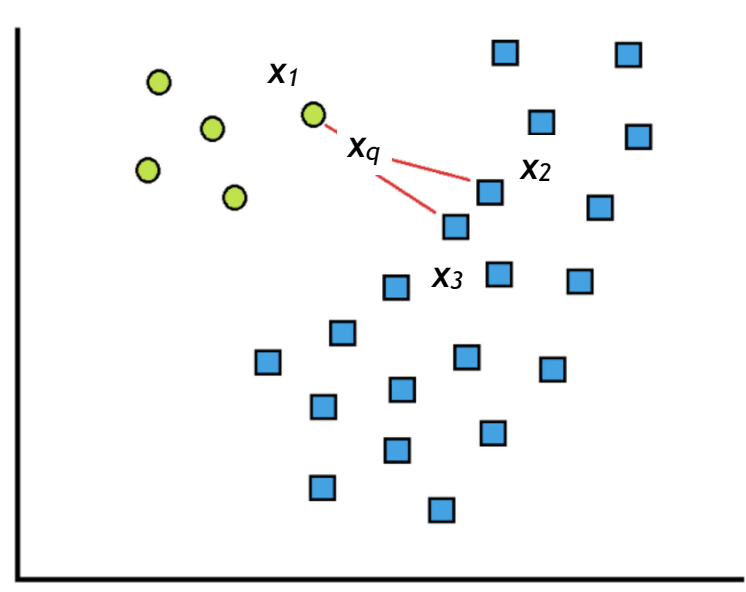
- In weighted kNN, the nearest k points are given a weight, and the weights are grouped by the target variable. The class with the largest sum of weights will be the class that is predicted
- The intuition is to give more weight to the points that are nearby and less weight to the points that are farther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

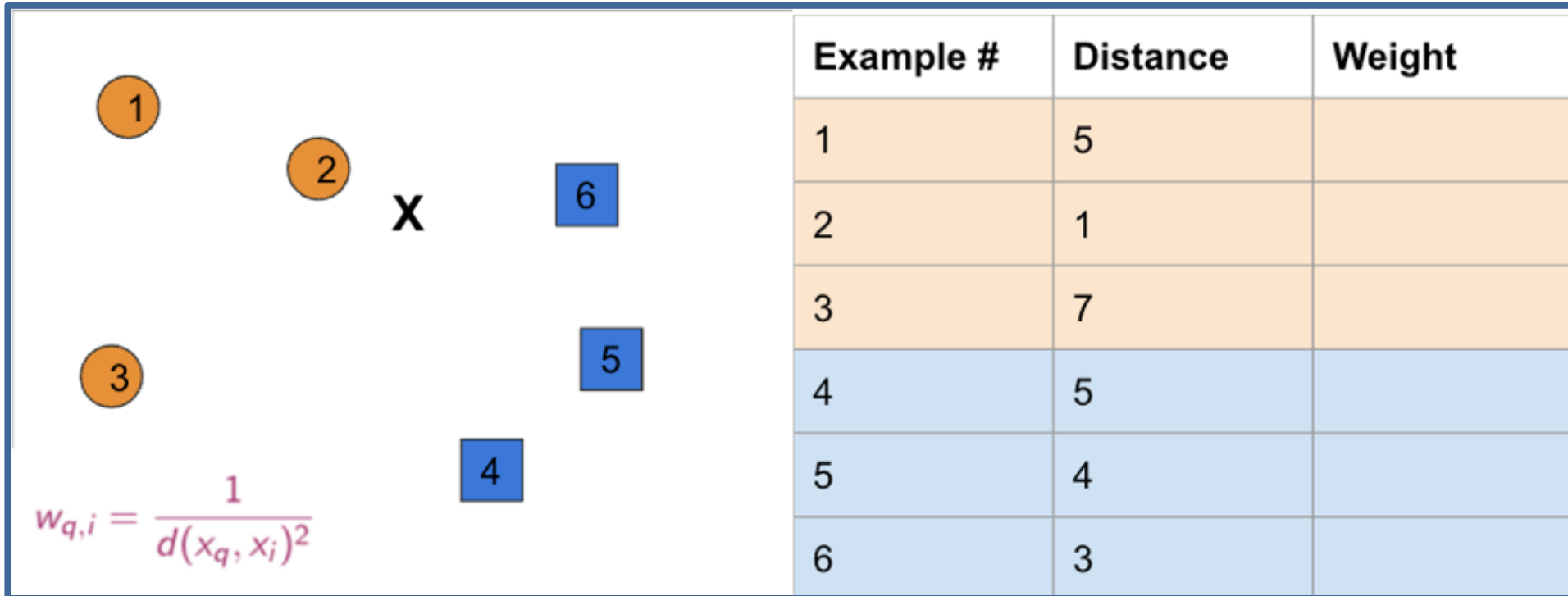
$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$



- 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

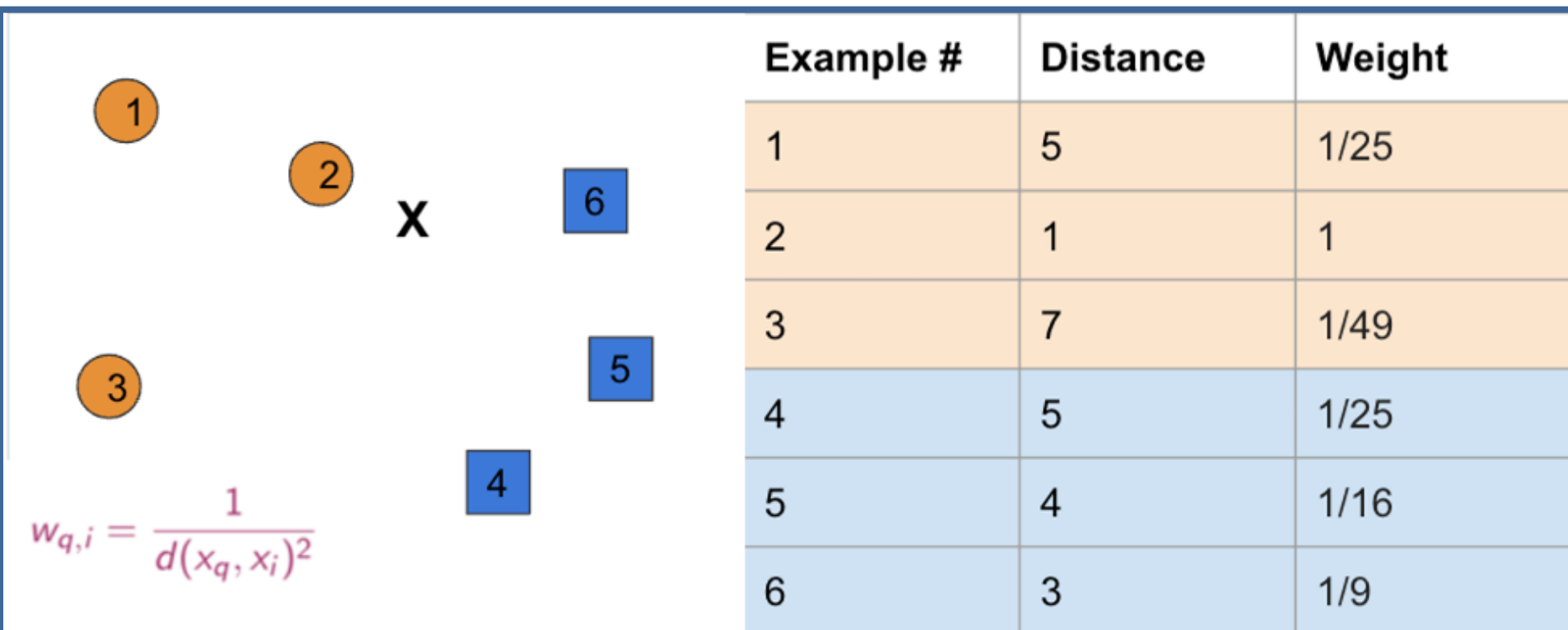
Weighted k-NN Example: Step 1

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



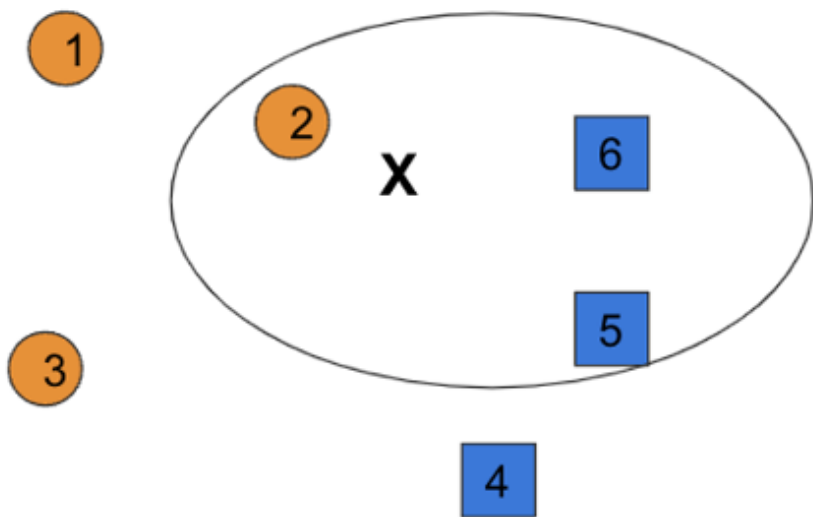
Weighted k-NN Example: Step 2

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



Weighted k-NN Example: Step 3

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

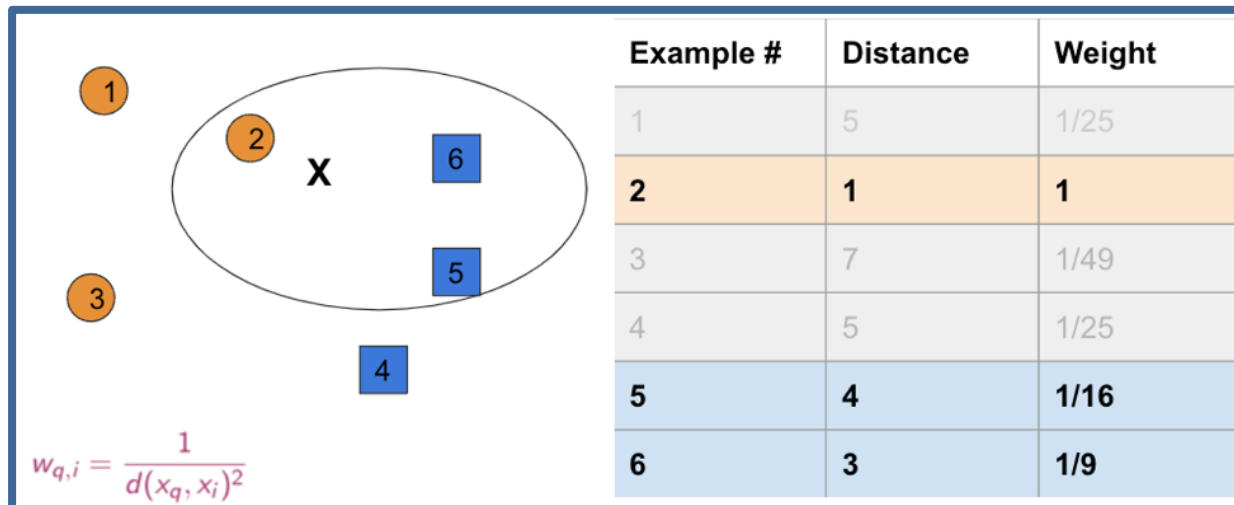


Example #	Distance	Weight
1	5	1/25
2	1	1
3	7	1/49
4	5	1/25
5	4	1/16
6	3	1/9

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

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?



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
```

Recall: Some Handy Functions

- `unique()`, `groupby()`

```
▶ #get the unique values of the Deck column
titanic.deck.unique()

array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F'], dtype=object)
```

```
▶ titanic.groupby(['survived'])['age'].mean()

survived
0    30.626179
1    28.343690
Name: age, dtype: float64
```

```
[19] condition = titanic['survived'] == 0
survivor_0 = titanic[condition]['age']
survivor_0.mean()

30.62617924528302
```

```
[20] condition = titanic['survived'] == 1
survivor_1 = titanic[condition]['age']
survivor_1.mean()

28.343689655172415
```

```
▶ titanic.groupby('survived')['age'].mean()

↳ survived
0    30.626179
1    28.343690
Name: age, dtype: float64
```


Programming Exercise #4

- Normalize each of the predictor columns in the iris dataset, or just use `iris_norm` which we created earlier (ie, Programming Exercise#2)
 - **Note:** you need a way to transform the new reading (the specimen) that you will make the prediction on so that the new one and the training data will all be on the same scale. How can you do that?
 - **Hint:** modify the `z_score()` method to save the `<mean, std>` for each column, then utilize that later

Programming Exercise #4

- Repeat your k-NN prediction code for the normalized data.
 - Does the value of k change the predictions?
 - compare using k=3, and k=5 on each method (normalized and non-normalized), (weighted and unweighted)

Use these tables to keep track of your predictions:

k=3

	not normalized	normalized
unweighted kNN		
weighted kNN		

k=5

	not normalized	normalized
unweighted kNN		
weighted kNN		