k Nearest Neighbors

PhD. Msc. David C. Baldears S.

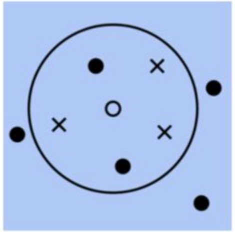
Instance-based learning

Instance-based learning is a machine learning technique that relies on storing and recalling instances or examples of training data.

- Machine learning learns from examples,
  - yet, instance-based learning does not generalize to some function
  - instead uses the examples as truth.

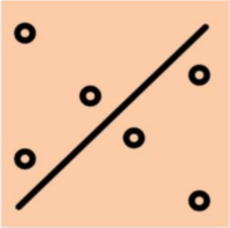
Instance-Based Learning

- Uses entire dataset as model
- Learns with examples
- Needs a similarity measure
- Generalized to new cases based on similarity
- Compared learned examples with new ones



Model-Based Learning

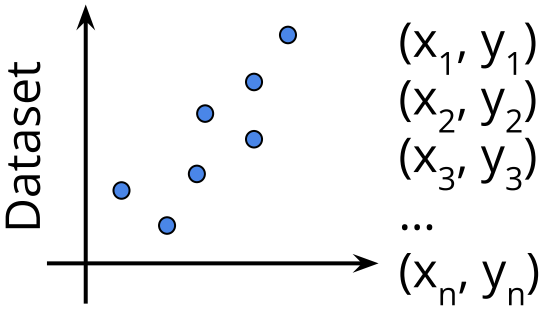
- Uses the training data to create a model that has parameters learned from the training dataset
- The system uses a model
- It learns when the model is trained with the data
- It uses model to make predictions

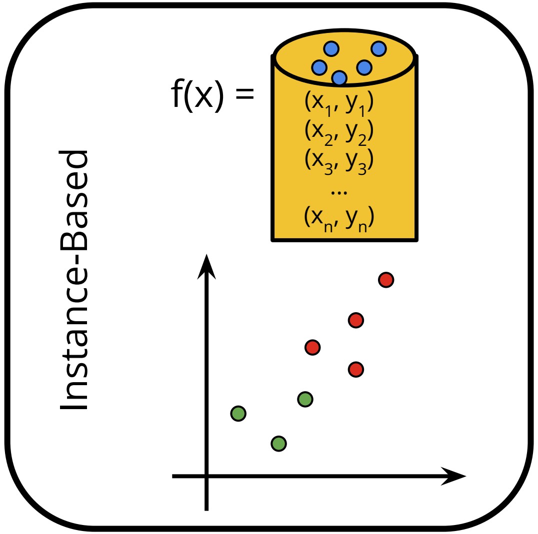


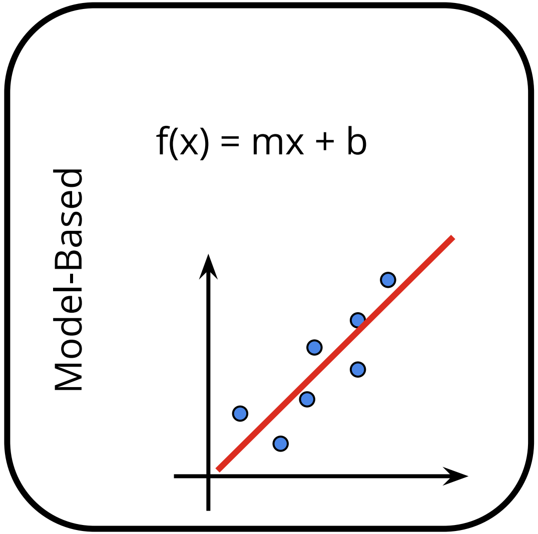
Instance-Based vs Model-Based

- Can't be stored
- Needs to use all data
- Takes times every time to use it on new data
- No need to train and evaluate a model

- The model can be stored/saved
- There is no need to use all data
- It takes less space than all data
- The model is faster making predictions
- Needs to train and evaluate a model







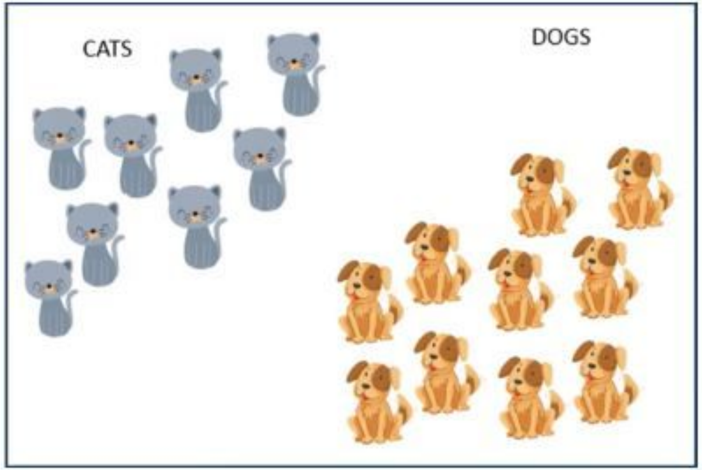
Analogy

Talk about your friends (your neighbors) and I will tell you who you are!!

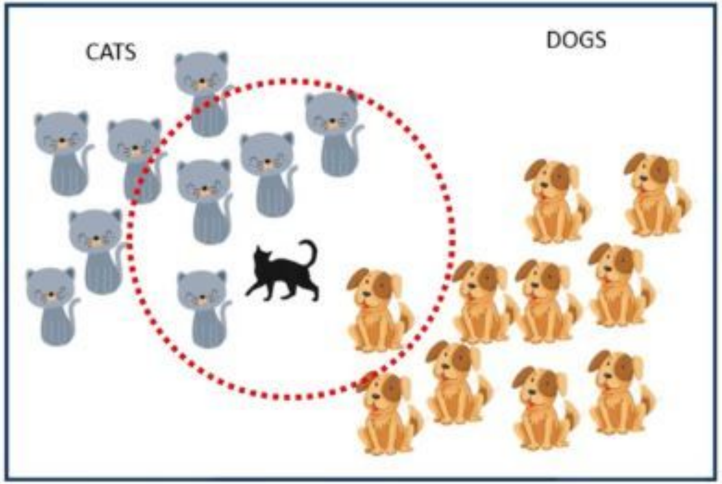
Nearest Neighbor

“Birds of a feather flock together”

Assumes that similar things exist in close proximity



KNN Classifier



Example - T Shirt Size

Calculate the distance

New customer named 'Monica' has height 161 cm and weight 63 kg.

Height (cm)	Weight (kg)	**T Shirt Size**
160	60	**M**
163	60	**M**
163	61	**M**
160	64	**L**
163	64	**L**
165	61	**L**

Example - T Shirt Size

Calculate the distance [161, 63]

$$d = \sqrt{(160 - 161)^2 + (60 - 63)^2}$$

If k = 3 ?

Height (cm)	Weight (kg)	**T Shirt Size**	Distance
160	60	**M**	3.162278
163	60	**M**	3.605551
163	61	**M**	2.828427
160	64	**L**	1.414214
163	64	**L**	2.236068
165	61	**L**	4.472136

Pseudocode

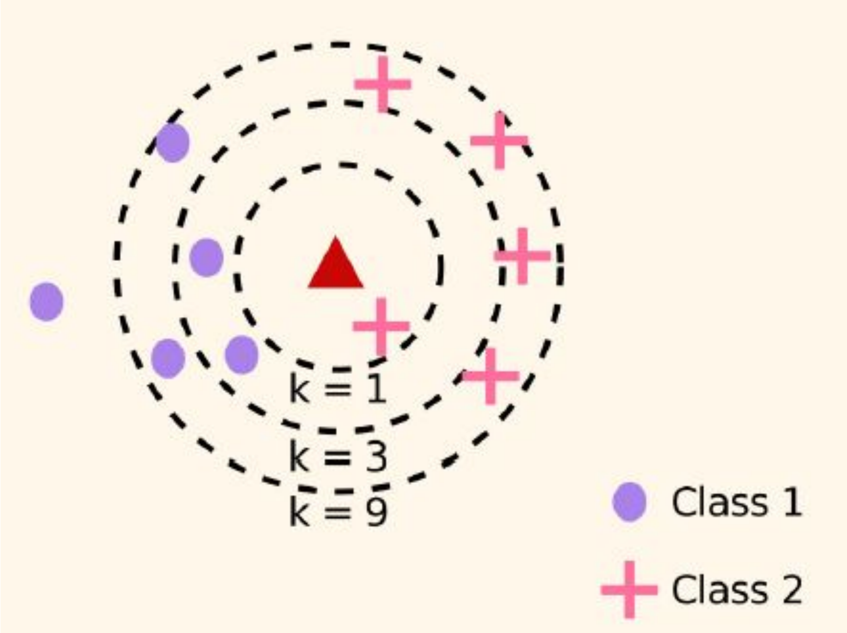
The KNN Algorithm:

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
   3.1. Calculate the distance between the query example and the current example from the data.
   3.2. Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. a) If regression, return the mean of the K labels
7. b) If classification, return the mode of the K labels

kNN Metrics

Nearest Neighbor Classifier

Name	Functions
Euclidean	$d(x_i, u) = \sqrt{\sum_{i=1}^{n} (x_{ij} - u_j)^2}$
Manhattan	$d(x_i, u) = \sum_{i=1}^{n} \lvert x_{ij} - u_j \rvert$
Minkowski	$d(x_i, u) = \left(\sum_{i=1}^{n} (\lvert x_{ij} - u_j \rvert)^q \right)$
Hamming	$d(x_i, u) = \sum_{i=1}^{n} [x_{ij} \neq u_j]$
Mahalanobis	$d(x_i, u) = \sqrt{(x_i - u)' C^{-1} (x_i - u)}$, $C = covariance \quad matrix$



Example Play vs Don't Play

- Determine the value of the attribute Play for Day 8 with kNN algorithm for k = 1 and k = 3

Day	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Rain	Cool	Normal	Weak	??

- The distance between two instance X and Y can be calculated using a variety of distance algorithms; like Euclidean, Manhattan, and so on
- For example for categorical variables - Hamming distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

Example Play vs Don't Play

- Determine the value of the attribute Play for Day 8 with kNN algorithm for k = 1 and k = 3

Day	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Rain	Cool	Normal	Weak	??

Distance matrix for day 8:

Day	1	2	3	4	5	6	7	8
Distance	3	4	3	2	1	2	3	0

Example Play vs Don't Play

- Determine the value of the attribute Play for Day 8 with kNN algorithm for k = 1 and k = 3

Day	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Rain	Cool	Normal	Weak	??

Distance matrix for day 8:

Day	1	2	3	4	5	6	7	8
Distance	3	4	3	2	1	2	3	0

K-Nearest Neighbor for k = 1:
Day 5=> Result: Don't Play

K-Nearest Neighbor for k = 3:
Days 4, 5, 6=> Result: Play
→ The majority vote wins

Main Function

```
from collections import Counter
import math
def knn(data, query, k, distance_fn, choice_fn):
    neighbor_distances_and_indices = []
    # 3. For each example in the data
    for index, example in enumerate(data):
        # 3.1 Calculate the distance between the query example and the current
        # example from the data.
        distance = distance_fn(example[:-1], query)
        # 3.2 Add the distance and the index of the example to an ordered collection
        neighbor_distances_and_indices.append((distance, index))
    # 4. Sort the ordered collection of distances and indices from
    # smallest to largest (in ascending order) by the distances
    sorted_neighbor_distances_and_indices = sorted(neighbor_distances_and_indices)
    # 5. Pick the first K entries from the sorted collection
    k_nearest_distances_and_indices = sorted_neighbor_distances_and_indices[:k]
    # 6. Get the labels of the selected K entries
    k_nearest_labels = [data[i][-1] for distance, i in k_nearest_distances_and_indices]
    # 7. If regression (choice_fn = mean), return the average of the K labels
    # 8. If classification (choice_fn = mode), return the mode of the K labels
    return k_nearest_distances_and_indices , choice_fn(k_nearest_labels)
```

Auxiliary Functions

```
def mean(labels):
	return sum(labels) / len(labels)

def mode(labels):
	return Counter(labels).most_common(1)[0][0]

def euclidean_distance(point1, point2):
	sum_squared_distance = 0
	for i in range(len(point1)):
	sum_squared_distance += math.pow(point1[i] - point2[i], 2)
	return math.sqrt(sum_squared_distance)
```

Classification Data

```
def main():
    '''
    # Classification Data
    #
    # Column 0: age
    # Column 1: likes pineapple
    '''
    clf_data = [ [22, 1], [23, 1], [21, 1], [18, 1], [19, 1], [25, 0], [27, 0], [29, 0], [31, 0], [45, 0], ]
    # Question:
    # Given the data we have, does a 33 year old like pineapples on their pizza?
    clf_query = [33]
    clf_k_nearest_neighbors, clf_prediction = knn(
    clf_data, clf_query, k=3, distance_fn=euclidean_distance, choice_fn=mode
    )

if __name__ == '__main__':
    main()
```

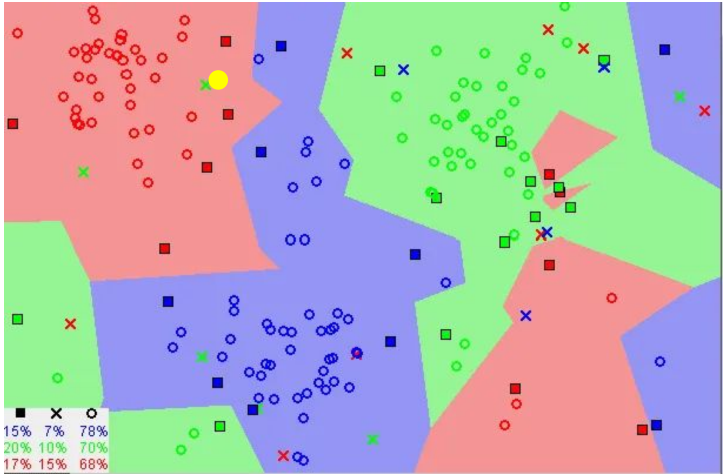
Regression Data

```
def main():
	'''
	# Regression Data
	# Column 0: height (inches)
	# Column 1: weight (pounds)
	'''
	reg_data = [ [65.75, 112.99], [71.52, 136.49], [69.40, 153.03], [68.22, 142.34], [67.79, 144.30], [68.70, 123.30], [69.80, 141.49], [70.01, 
136.46], [67.90, 112.37], [66.49, 127.45], ]

	# Question:
	# Given the data we have, what's the best-guess at someone's weight if they are 60 inches tall?
	reg_query = [60]
	reg_k_nearest_neighbors, reg_prediction = knn(
	reg_data, reg_query, k=3, distance_fn=euclidean_distance, choice_fn=mean
	)

if __name__ == '__main__':
	main()
```

Choosing the right value for K

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable.
   a. Imagine K=1 and a new query point (yellow circle).
   b. The green is the single nearest neighbor.
   c. Yet, the point is most likely red, but because K=1, KNN incorrectly predicts as green.
2. Inversely, if K increases, predictions become more stable (majority voting / averaging)
   a. more likely to make more accurate predictions (up to a certain point).
   b. Eventually, the number of errors increases. At this point the value of K has been pushed too far.



3. For a majority vote among labels, usually make K an odd number to have a tiebreaker.

Advantages vs Disadvantages

Advantages

1. The algorithm is simple and easy to implement.
2. There’s no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm is versatile. It can be used for classification, regression, and search.
4. Learns from lows amounts of training data
5. Easier To Understand The Results

Disadvantages

1. Stores Data In Memory
2. Need to be retrained much more often or always
3. The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Example 2

Movies

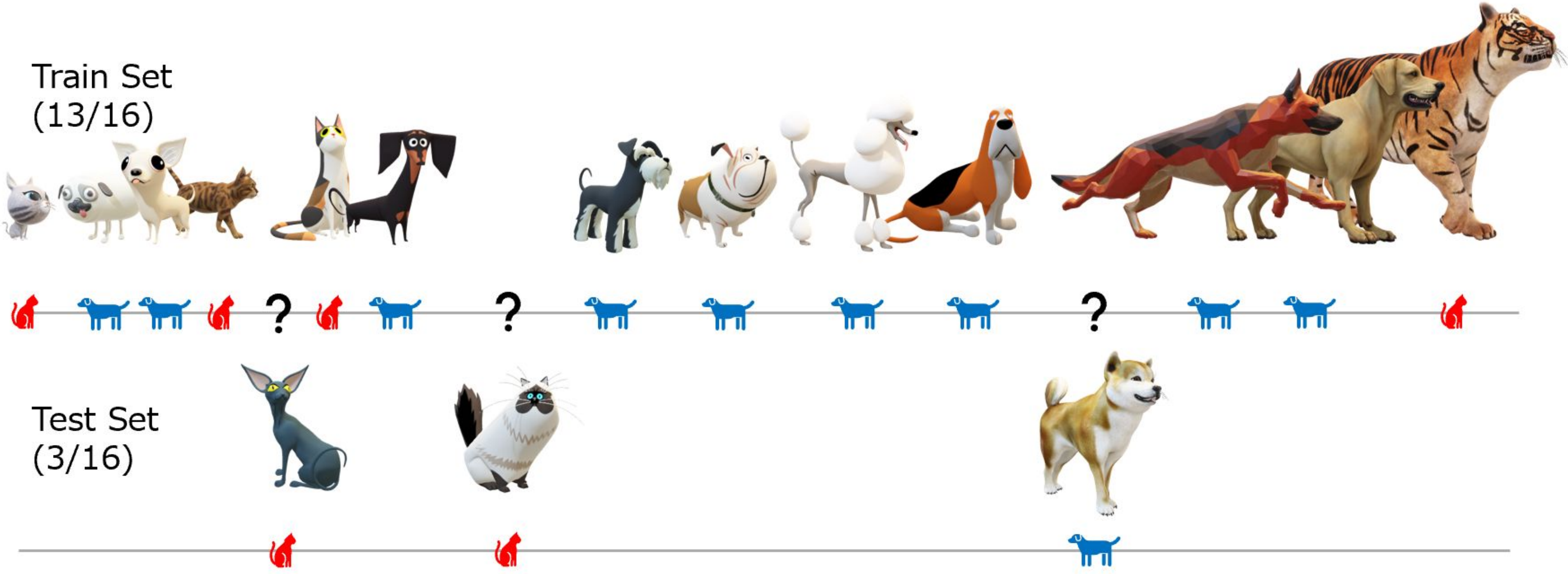
The data contains thirty movies, including data for each movie across seven genres and their IMDB ratings. The labels column has all zeros because we aren’t using this data set for classification or regression.

There are relationships among the movies that will not be accounted for (e.g. actors, directors, and themes)

Movie ID	Movie Name	IMDB Rating	Biography	Drama	Thriller	Comedy	Crime	Mystery	History	Label
58	The Imitation Game	8	1	1	1	0	0	0	0	0
8	Ex Machina	7.7	0	1	0	0	0	1	0	0
46	A Beautiful Mind	8.2	1	1	0	0	0	0	0	0
62	Good Will Hunting	8.3	0	1	0	0	0	0	0	0
97	Forrest Gump	8.8	0	1	0	0	0	0	0	0
98	21	6.8	0	1	0	0	1	0	1	0
31	Gifted	7.6	0	1	0	0	0	0	0	0
3	Travelling Salesman	5.9	0	1	0	0	0	1	0	0
51	Avatar	7.9	0	0	0	0	0	0	0	0
47	The Karate Kid	7.2	0	1	0	0	0	0	0	0
50	A Brilliant Young Mind	7.2	0	1	0	0	0	0	0	0
49	A Time To Kill	7.4	0	1	1	0	1	0	0	0
30	Interstellar	8.6	0	1	0	0	0	0	0	0
94	The Wolf of Wall Street	8.2	1	0	0	1	1	0	0	0
6	Black Panther	7.4	0	0	0	0	0	0	0	0
73	Inception	8.8	0	0	0	0	0	0	0	0
44	The Wind Rises	7.8	1	1	0	0	0	0	0	0
65	Spirited Away	8.6	0	0	0	0	0	0	0	0
48	Finding Forrester	7.3	0	1	0	0	0	0	0	0
27	The Fountain	7.3	0	0	0	0	0	0	0	0
57	The DaVinci Code	6.6	0	0	1	0	0	1	0	0
57	Stand and Deliver	7.3	0	1	0	0	0	0	0	0
14	The Terminator	8	0	0	0	0	0	0	0	0
69	21 Jump Street	7.2	0	0	0	1	1	0	0	0
98	The Avengers	8.1	0	0	0	0	0	0	0	0
17	Thor: Ragnarok	7.9	0	0	0	1	0	0	0	0
12	Spirit: Stallion of the Cimarron	7.1	0	0	0	0	0	0	0	0
1	Hacksaw Ridge	8.2	1	1	0	0	0	0	1	0
86	12 Years a Slave	8.1	1	1	0	0	0	0	1	0
46	Queen of Katwe	7.4	1	1	0	0	0	0	0	0

Example 2

```python
from knn_from_scratch import knn, euclidean_distance
def recommend_movies(movie_query, k_recommendations):
    raw_movies_data = []
    with open('movies_recommendation_data.csv', 'r') as md:
        # Discard the first line (headings)
        next(md)

        # Read the data into memory
        for line in md.readlines():
            data_row = line.strip().split(',')
            raw_movies_data.append(data_row)

    # Prepare the data for use in the knn algorithm by picking
    # the relevant columns and converting the numeric columns
    # to numbers since they were read in as strings
    movies_recommendation_data = []
    for row in raw_movies_data:
        data_row = list(map(float, row[2:]))
        movies_recommendation_data.append(data_row)

    # Use the KNN algorithm to get the 5 movies that are most
    # similar to The Post.
    recommendation_indices, _ = knn(
    movies_recommendation_data, movie_query, k=k_recommendations,
    distance_fn=euclidean_distance, choice_fn=lambda x: None)

    movie_recommendations = []
    for _, index in recommendation_indices:
        movie_recommendations.append(raw_movies_data[index])

    return movie_recommendations

if __name__ == '__main__':
    the_post = [7.2, 1, 1, 0, 0, 0, 0, 1, 0] # feature vector for The Post
    recommended_movies = recommend_movies(movie_query=the_post,
k_recommendations=5)

    # Print recommended movie titles
    for recommendation in recommended_movies:
        print(recommendation[1])
```


Train Set
(13/16)
Test Set
(3/16)