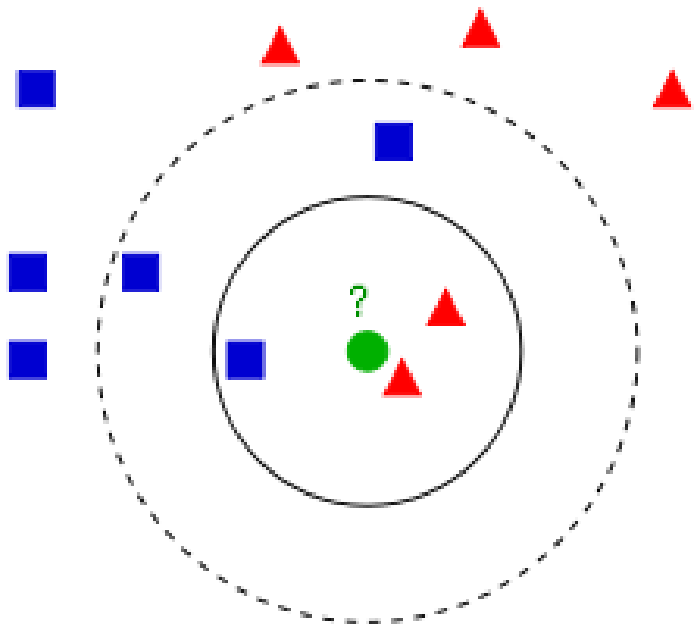
k-nearest neighbors (KNN)



KNN

Why do we need KNN?

What is KNN?

How do we choose the factor ‘K’?

When do we use KNN?

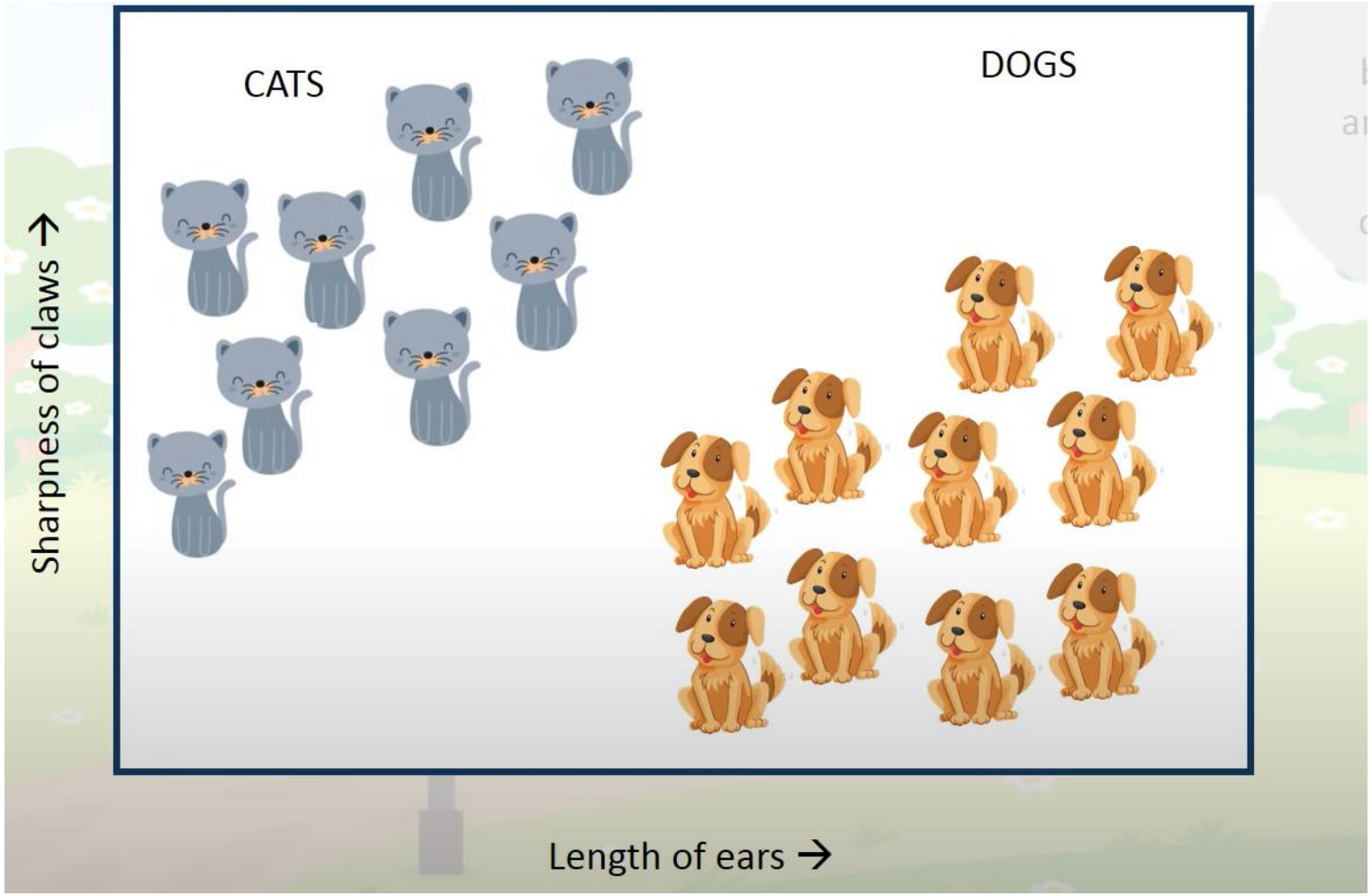
How does KNN Algorithm work?

Use Case: Predict whether a person will have diabetes or not

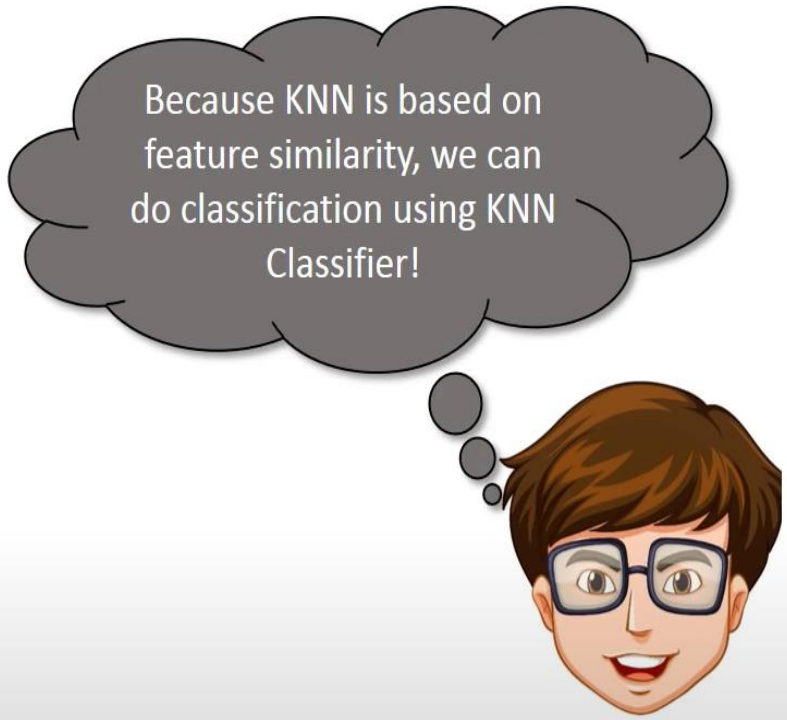
Differentiation Based on Features

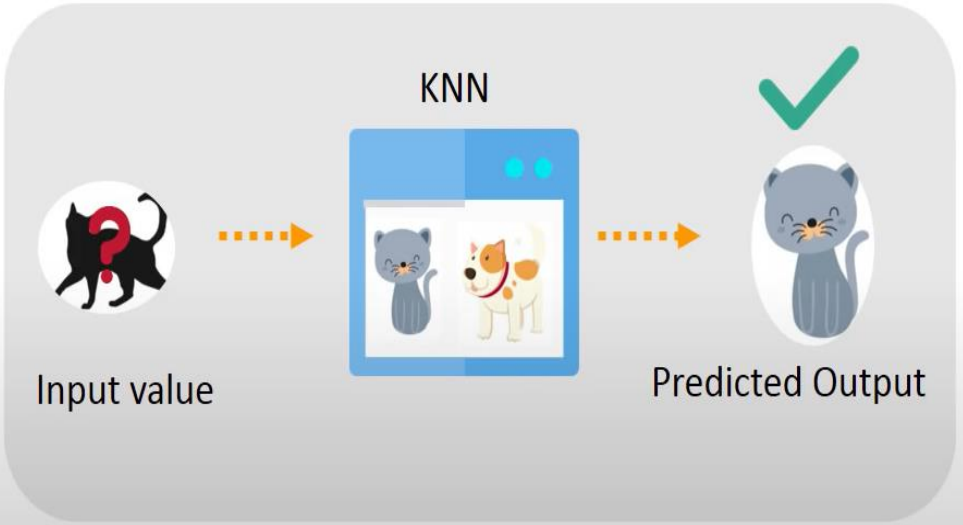
CATS	DOGS
	
Sharp Claws, uses to climb	Dull Claws
Smaller length of ears	Bigger length of ears
Meows and purrs	Barks
Doesn't love to play around	Loves to run around

Differentiation Based on Features



Why KNN?


Because KNN is based on feature similarity, we can do classification using KNN Classifier!


KNN
Input value
Predicted Output

What is KNN?

- KNN uses data with several classes to predict the classification of the new sample point.
- KNN is non-parametric since it doesn't make any assumptions on the data being studied, i.e., the model is distributed from the data.
- **Non-parametric Models** are statistical **models** that do not often conform to a normal distribution, as they rely upon continuous data, rather than discrete values.
- This is often the assumption that the population data are normally distributed. Non-**parametric tests** are "distribution-free" and, as such, can be used for non-Normal variables.
- **Parametric models** are those that make assumptions about the parameters of the population distribution from which the sample is drawn.
- Eg Decision Trees also have different number of leaves and different number of internal nodes, so the space of decision trees is non-parametric

What is KNN?

- KNN doesn’t try to learn a function from a given training data.
- KNN is a lazy algorithm as it doesn’t use the training data points to make any generalization. Which implies:
  - You expect little or no explicit training phase,
  - The training phase is pretty fast,
  - KNN keeps all the training data since they are needed during the testing phase.

DIFFERENT NAMES OF KNN ALGORITHM

- K-Nearest Neighbors
- Memory Based Reasoning
- Example Based Reasoning
- Lazy Learning
- Case-Based Learning
- Instance-Based Learning

Where KNN was born?

- KNN was born out of research done for the armed forces. Fix and Hodge – two officers of USAF School of Aviation Medicine – wrote a technical report in 1951 introducing the KNN algorithm.

Where to use KNN?

- KNN can be used in both regression and classification predictive problems. However, when it comes to industrial problems, it's mostly used in classification since it fairs across all parameters evaluated when determining the usability of a technique
  - **Prediction Power**
  - **Calculation Time**
  - **Ease to Interpret the Output**
- KNN algorithm fairs across all parameters of considerations. But mostly, it is used due to its ease of interpretation and low calculation time.

KNN : Better than more powerful classifiers

- In political science – classifying a political voter to “vote Republican” or “vote Democrat”, or to a “will vote” or “will not vote”.
- Banking system – KNN can be used to predict if a person is fit for loan approval. Or if he or she has similar traits to a defaulter.
- Calculating credit ratings – KNN can help when calculating an individual’s credit score by comparing it with persons with similar traits.
- Other areas that use the KNN algorithm include Video Recognition, Image Recognition, Handwriting Detection, and Speech Recognition.

What is KNN algorithm?

KNN – K Nearest Neighbors, is one of the simplest **Supervised** Machine Learning algorithm mostly used for

Classification

It classifies a data point based on how its neighbors are classified

When to use KNN Algorithm?

We can use KNN when

Data is labeled

Dog

Dataset is small

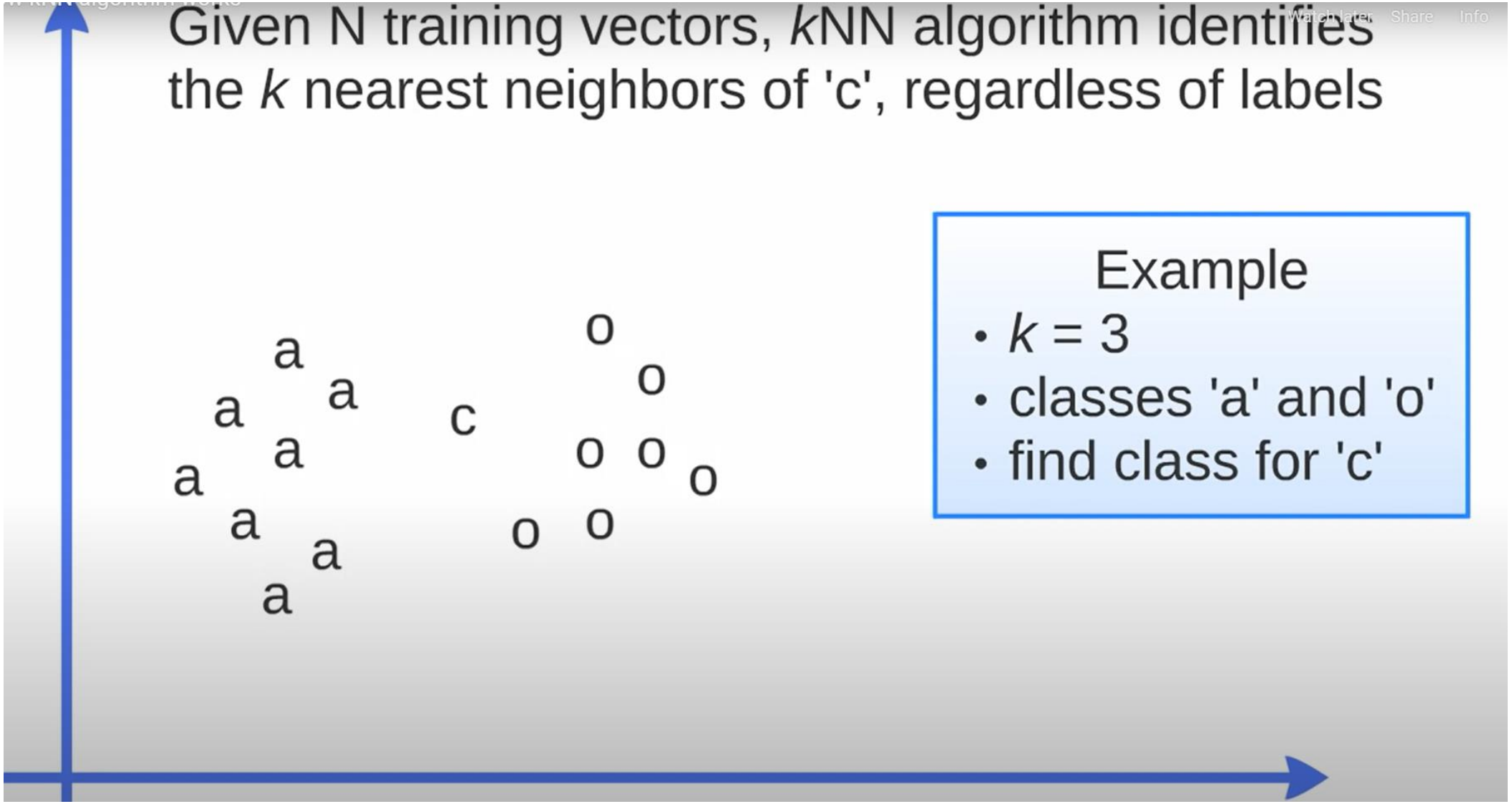
Because KNN is a 'lazy learner' i.e. doesn't learn a discriminative function from the training set

Data is noise free

Weight(x2)	Height(y2)	Class
51	167	Underweight
62	182	one-fourty
69	176	23
64	173	hello kitty
65	172	Normal

Noise

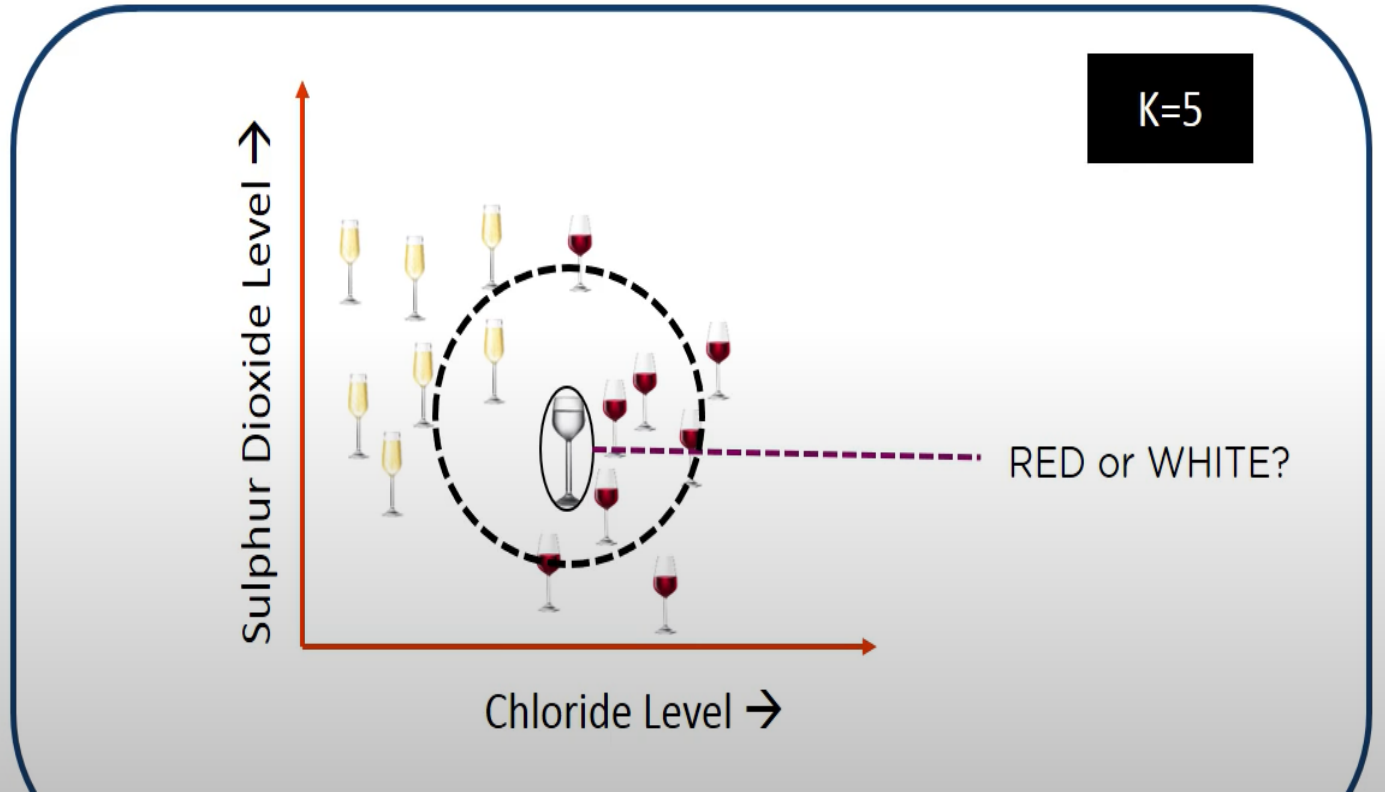
KNN algorithm?



Example : Wines

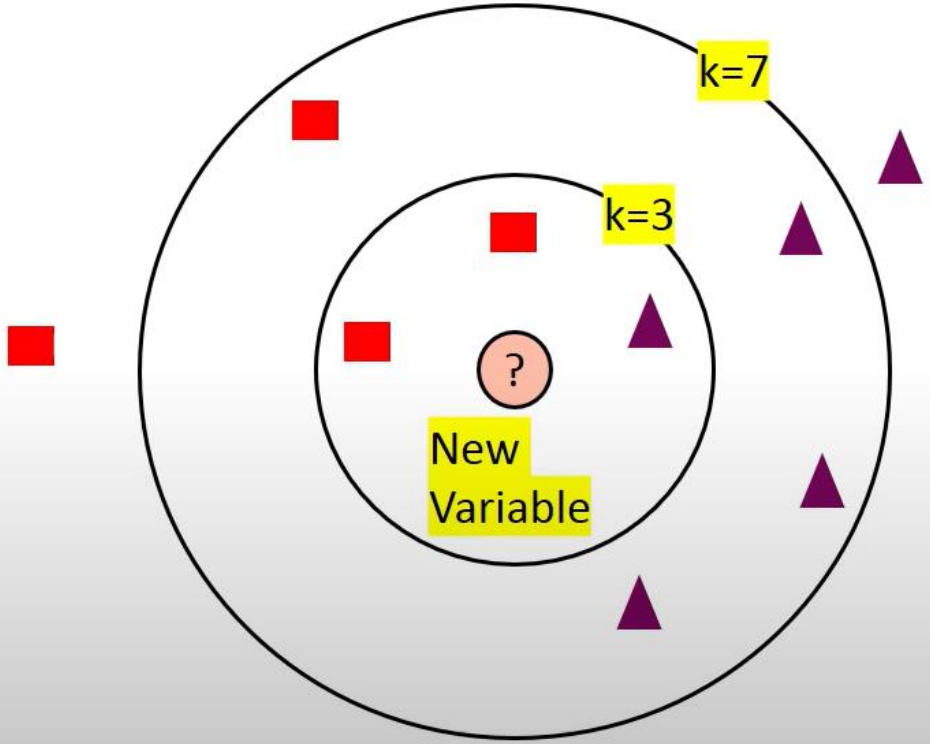
k in **KNN** is a parameter that refers to the number of nearest neighbors to include in the majority voting process

A data point is classified by majority votes from its 5 nearest neighbors



How do we choose 'k'?

KNN Algorithm is based on **feature similarity**: Choosing the right value of *k* is a process called parameter tuning, and is important for better accuracy



The class of unknown data point was ■ at k=3 but changed at k=7, so which k should we choose?

But at k=7, we classify '?' as ▲

How do we choose ‘k’?

- K value can be 1 to infinity, but in most practical cases, k is less than 30.
- To get the right K, you should run the KNN algorithm several times with different values of K and select the one that has the least number of errors.
- The right K must be able to predict data that it hasn’t seen before accurately.
- As your value of K increases, your prediction becomes more stable due to the majority of voters.
- When you start receiving an increasing number of errors, you should know you are pushing your K too far.
- Taking a majority vote among labels needs K to be an odd number to have a tiebreaker. An odd value should be chosen for a 2 class problem.

Working of KNN

Weight(x2)	Height(y2)	Class
51	167	Underweight
62	182	Normal
69	176	Normal
64	173	Normal
65	172	Normal
56	174	Underweight
58	169	Normal
57	173	Normal
55	170	Normal

On the basis of the given data we have to classify the below set as Normal or Underweight using KNN

57 kg	170 cm	?

To find the nearest neighbors, we will calculate Euclidean distance

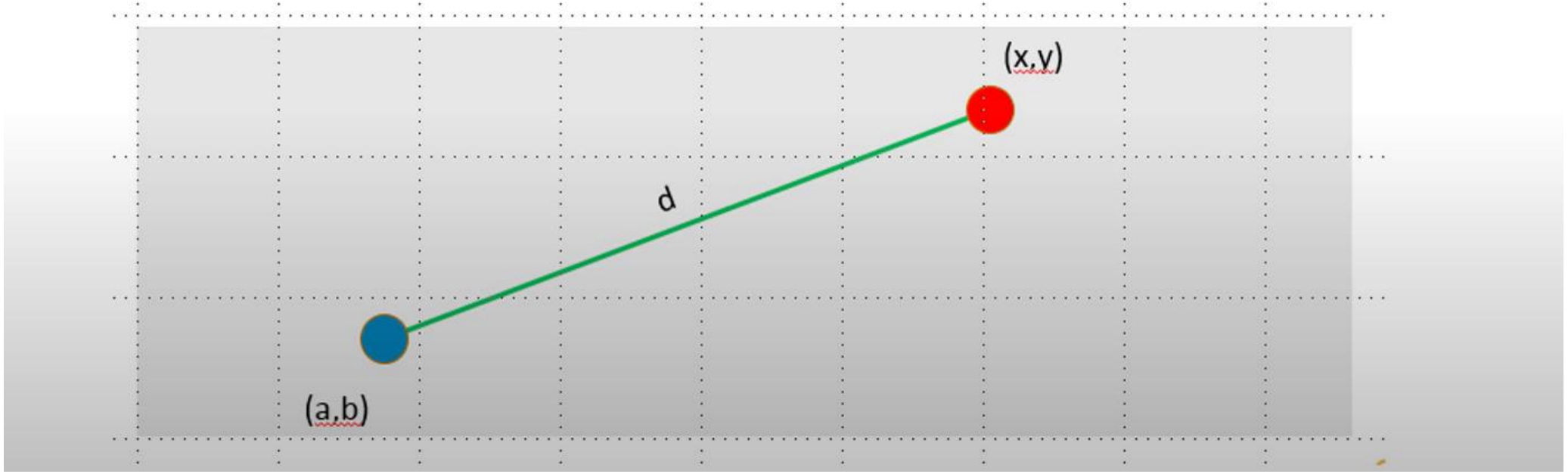
Calculating Distance

- Euclidean distance, Hamming, or Manhattan can be used to calculate the distance between test data and each row of training.
- The Euclidean method, the straight-line distance (also called the Euclidean distance) is the most used when calculating distance.
- It is the most popular and familiar choice.
- The p value in the formula can be manipulated to give us different distances like:
- p = 1, when p is set to 1 we get Manhattan distance
- p = 2, when p is set to 2 we get Euclidean distance

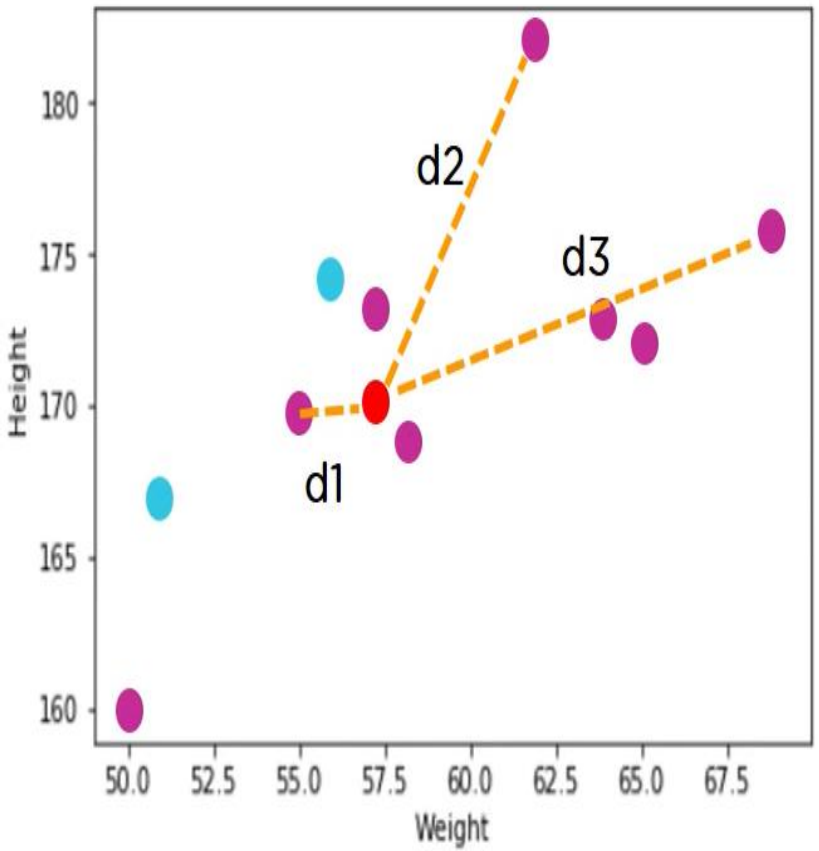
Euclidean distance

According to the **Euclidean distance** formula, the **distance** between two points in the plane with coordinates (x, y) and (a, b) is given by:

$$dist(d) = \sqrt{(x - a)^2 + (y - b)^2}$$



Euclidean distance



dist(**d1**)= √(170-167)² + (57-51)² ~= 6.7

dist(**d2**)= √(170-182)² + (57-62)² ~= 13

dist(**d3**)= √(170-176)² + (57-69)² ~= 13.4

Similarly, we will calculate Euclidean distance of unknown data point from all the points in the dataset

Unknown data point

Euclidean distance

Hence, we have calculated the Euclidean distance of unknown data point from all the points as shown:

Where (x1, y1) = (57, 170) whose class we have to classify

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2

Euclidean distance

Now, lets calculate the nearest neighbor at k=3

Weight(x2)	Height(y2)	Class	Euclidean Distance
51	167	Underweight	6.7
62	182	Normal	13
69	176	Normal	13.4
64	173	Normal	7.6
65	172	Normal	8.2
56	174	Underweight	4.1
58	169	Normal	1.4
57	173	Normal	3
55	170	Normal	2

- **Sort data set in ascending order based on the distance value.**
- **From the sorted array, choose the top K rows.**
- **Based on the most appearing class of these rows, it will assign a class to the test point.**
- **If regression, return the mean of the K labels**

Steps of KNN

- A positive integer k is specified, along with a new sample
- We select the k entries in our database which are closest to the new sample
- We find the most common classification of these entries
- This is the classification we give to the new sample

USE CASE: Predict Diabetes

We have a dataset of 768 people who were or were not diagnosed with diabetes

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1

KNN - Predict whether a person will have diabetes or not

In [ ]:

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

```
dataset = pd.read_csv('../Downloads/diabetes.csv')
```

```
dataset.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Values of columns like 'Glucose', BloodPressure' cannot be accepted as zeroes because it will affect the outcome

We can replace such values with the mean of the respective column:

```
# Replace zeroes
zero_not_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Insulin']
```

```
for column in zero_not_accepted:
    dataset[column] = dataset[column].replace(0, np.NaN)
    mean = int(dataset[column].mean(skipna=True))
    dataset[column] = dataset[column].replace(np.NaN, mean)
```

Feature Scaling

Rule of thumb: Any algorithm that computes distance or assumes normality, **scale your features!**

```
# split dataset
X = dataset.iloc[:, 0:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
```

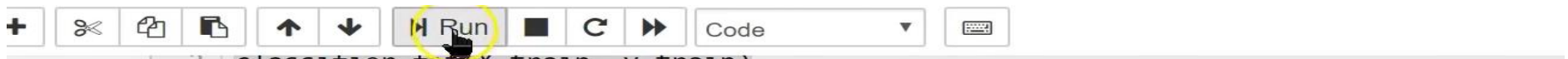
The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and **standard** deviation of 1

```
# Feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
# Define the model: Init K-NN
classifier = KNeighborsClassifier(n_neighbors=11, p=2, metric='euclidean')
```

```
# Fit Model
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
```




```
2 classifier.fit(X_train, y_train)
```

Out[13]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
```

In [14]:
```
1 # Predict the test set results
2 y_pred = classifier.predict(X_test)
3 y_pred
```

Out[14]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      dtype=int64)
```

dtype=int64)

In [16]:
```
# Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

```
[[94 13]
 [15 32]]
```

In [17]:
```
print(f1_score(y_test, y_pred))
```

```
0.6956521739130436
```

In [18]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.8181818181818182
```

In [22]:
```
accuracy_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,df_feat,df['TARGET CLASS'],cv=10)
    accuracy_rate.append(score.mean())
```

In [23]:
```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,df_feat,df['TARGET CLASS'],cv=10)
    error_rate.append(1-score.mean())
```

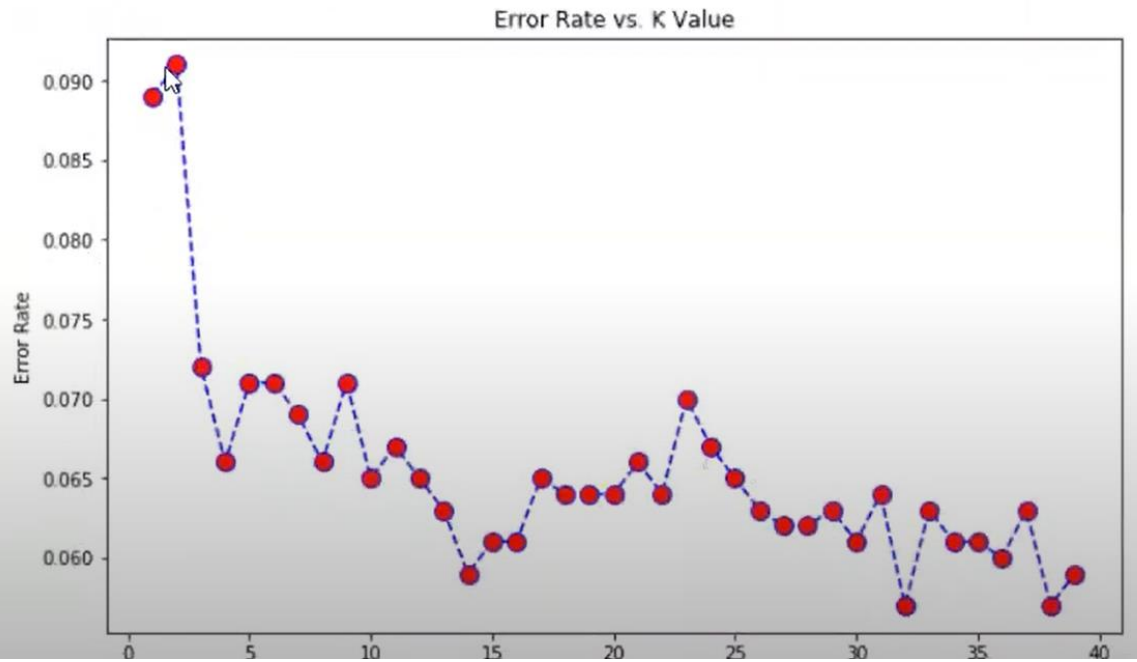
```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [24]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
#plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o',
 #         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

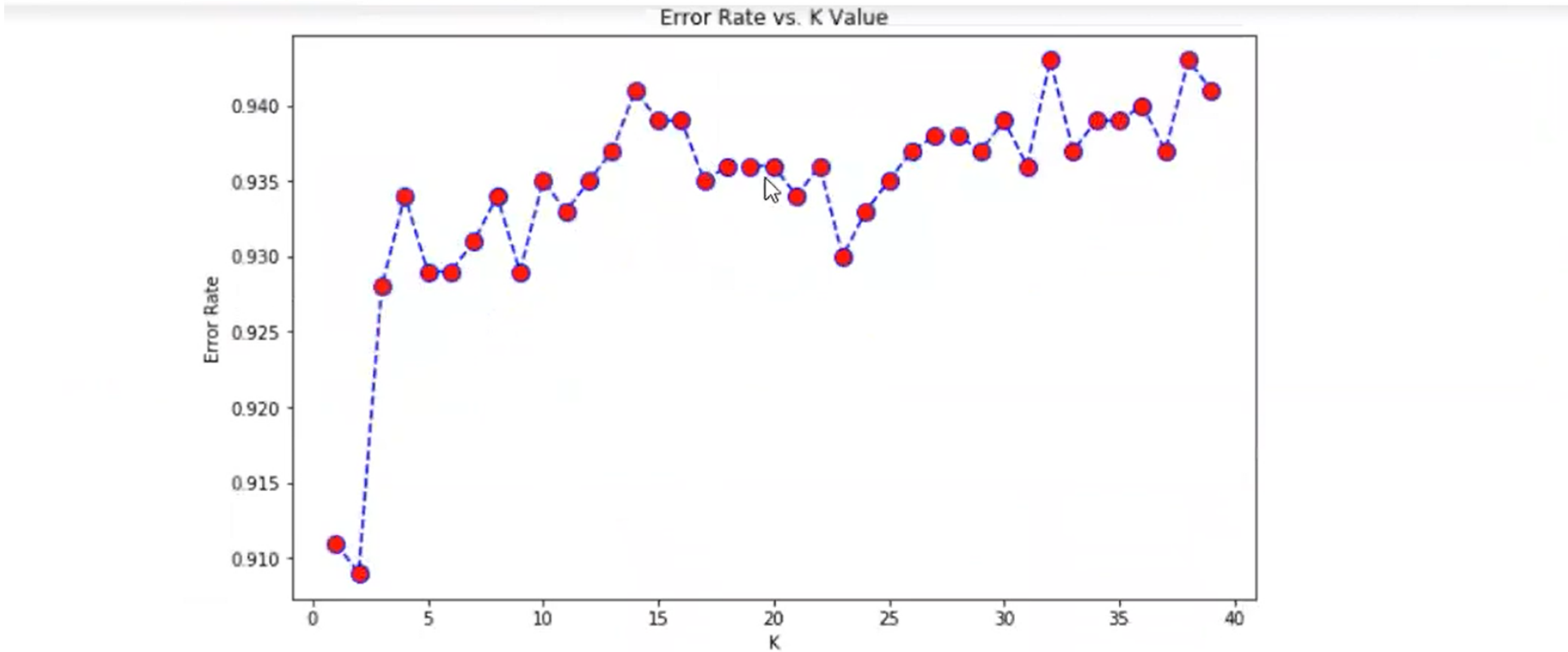
Out[24]: Text(0, 0.5, 'Error Rate')



Here we can see that that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

```
plt.figure(figsize=(10,6))
#plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
  #        markerfacecolor='red', markersize=10)
plt.plot(range(1,40),accuracy_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



KNN Advantages

- Quick calculation time
- Simple algorithm – to interpret
- Versatile – useful for regression and classification
- High accuracy – you do not need to compare with better-supervised learning models
- No assumptions about data – no need to make additional assumptions, tune several parameters, or build a model. This makes it crucial in nonlinear data case.

Disadvantages of KNN

- Accuracy depends on the quality of the data
- With large data, the prediction stage might be slow as it needs to calculate the distance of a given point with all other points
- Sensitive to the scale of the data and irrelevant features
- The testing phase of **K-nearest neighbor** classification **is** slower and costlier in terms of time and **memory**.
- As it **requires large memory** for storing the entire training dataset for prediction.
- **KNN requires** scaling of data because **KNN** uses the Euclidean distance between two data points to find nearest neighbors.
- Given that it stores all of the training, it can be computationally expensive

Companies Using KNN

- Companies like Amazon or Netflix use KNN when recommending books to buy or movies to watch. There was even a $1 million award on Netflix to the team that could come up with the most accurate recommendation algorithm!
- How do these companies make recommendations? Well, these companies gather data on the books you have read or movies you have watched on their website and apply KNN. The companies will input your available customer data and compare that to other customers who have purchased similar books or have watched similar movies.
- The books and movies recommended depending on how the algorithm classifies that data point.

Improving Performance of KNN

- **The steps in rescaling features in KNN are as follows:**
- Load the library.
- Load the dataset.
- Sneak Peak Data.
- Standard Scaling.
- Robust Scaling.
- Min-Max Scaling.
- Tuning Hyperparameters.

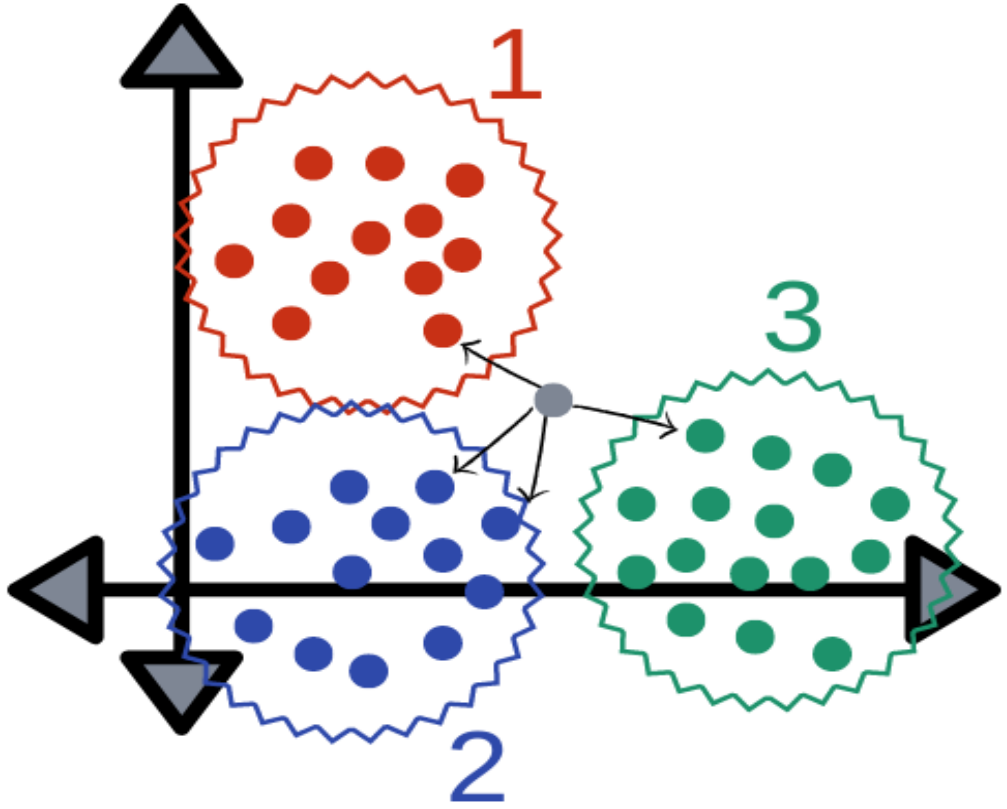
Recommendation System using KNN

- Imagine for a moment. We are navigating the MoviesXb website, a fictional IMDb spin-off, and we encounter *The Post*. We aren't sure we want to watch it, but its genres intrigue us; we are curious about other similar movies. We scroll down to the "More Like This" section to see what recommendations MoviesXb will make, and the algorithmic gears begin to turn.
- The MoviesXb website sends a request to its back-end for the 5 movies that are most similar to *The Post*. The back-end has a recommendation data set exactly like ours. It begins by creating the row representation (better known as a **feature vector**) for *The Post*, then it runs a program similar to the one below to search for the 5 movies that are most similar to *The Post*, and finally sends the results back to the MoviesXb website.

Recommendation System using KNN

- Well companies like Amzon and Netflix will apply KNN on a data set gathered about the movies you've watched or the books you've bought on their website.
- These companies will then input your available customer data and compare that to other customers who have watched similar movies or bought similar books.
- This data point will then be classified as a certain profile based on their past using KNN.
- The movies and books recommended will then depend on how the algorithm classifies that data point.

Recommendation System using KNN



The image above visualizes how KNN works when trying to classify a data point based a given data set. It is compared to its nearest points and classified based on which points it is closest and most similar to. Here you can see the point Xj will be classified as either W1 (red) or W3 (green) based on its distance from each group of points.

Recommendation System using KNN

- There have been various studies conducted on how this algorithm can be improved. These studies aim to make it so you can weigh categories differently in order to make a more accurate classification. The weighting of these categories varies depending on how the distance is calculated.
- In conclusion, this is a fundamental machine learning algorithm that is dependable for many reasons like ease of use and quick calculation time. It is a good algorithm to use when beginning to explore the world of machine learning, but it still has room for improvement and modification.

Conclusion

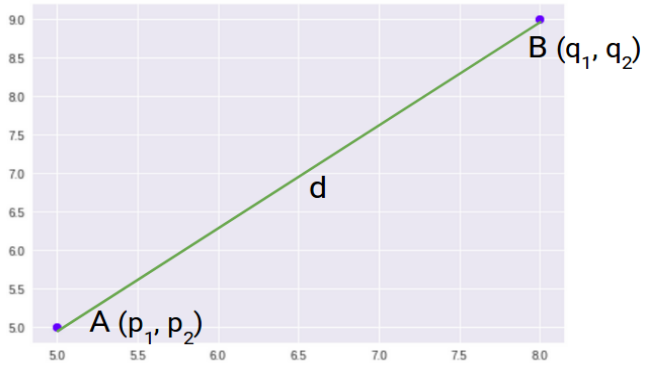
- KNN's main disadvantage of becoming significantly slower as the volume of data increases makes it an impractical choice in environments where predictions need to be made rapidly.
- Moreover, there are faster algorithms that can produce more accurate classification and regression results.
- However, provided you have sufficient computing resources to speedily handle the data you are using to make predictions, KNN can still be useful in solving problems that have solutions that depend on identifying similar objects.
- An example of this is using the KNN algorithm in recommender systems, an application of KNN-search.

Types of Distance Metrics in Machine Learning

- Euclidean Distance
- Manhattan Distance
- Minkowski Distance
- Hamming Distance

1. Euclidean Distance

- Most machine learning algorithms including K-Means use this distance metric to measure the similarity between observations. Let's say we have two points as shown below:



$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

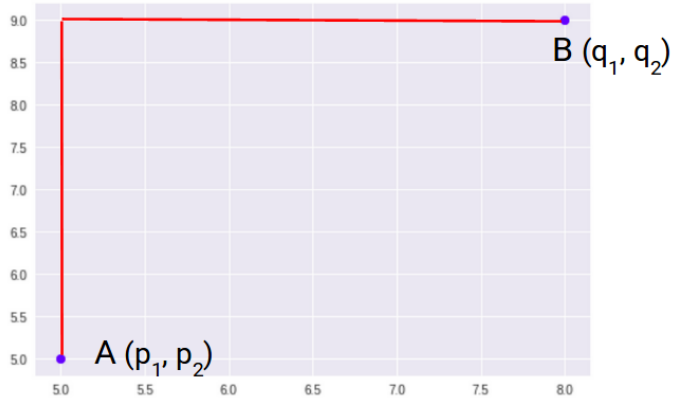
- We use this formula when we are dealing with 2 dimensions. We can generalize this for an n-dimensional space as:

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

- Where,n = number of dimensions and pi, qi = data points

2. Manhattan Distance

- *Manhattan Distance is the sum of absolute differences between points across all the dimensions.*



$$d = |p_1 - q_1| + |p_2 - q_2|$$

- Since the above representation is 2 dimensional, to calculate Manhattan Distance, we will take the sum of absolute distances in both the x and y directions.
- And the generalized formula for an n-dimensional space is given as:

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

3. Minkowski Distance

- *Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.* Here, p represents the order of the norm.

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

4. Hamming Distance

- *Hamming Distance measures the similarity between two strings of the same length. The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.*
- *Eg. Two strings* **“euclidean”** and **“manhattan”**

- seven characters are euclidean and manhattan ters (the last two characters) are similar
- Hence, the Hamming Distance here will be 7. Note that larger the Hamming Distance between two strings, more dissimilar will be those strings (and vice versa).
- **Note : Hamming distance only works when we have strings or arrays of the same length.**

Cosine Similarity

- **Cosine similarity** measures the **similarity** between two vectors of an inner product space.
- It is measured by the **cosine** of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.
- It is often used to measure document **similarity** in text analysis.

- References:

  - Medium
  - Simplilearn
  - Javatpoint.com
  - Wikipedia