Regression

After this video you will be able to..

- Define what regression is
- Explain the difference between regression and classification
- Name some applications of regression

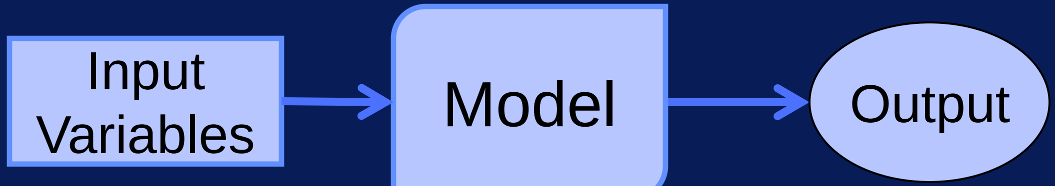
Classification Review

Input Variables → *Model* → Output

- Sunny
- Windy
- Rainy
- Cloudy

Classification:
Given input variables, predict category

Regression



number

Regression:
Given input variables,
predict numeric value

Regression Examples

- **Forecast** high temperature for next day
- **Estimate** average house price for a region
- **Determine** demand for a new product
- **Predict** power usage

Regression is Supervised

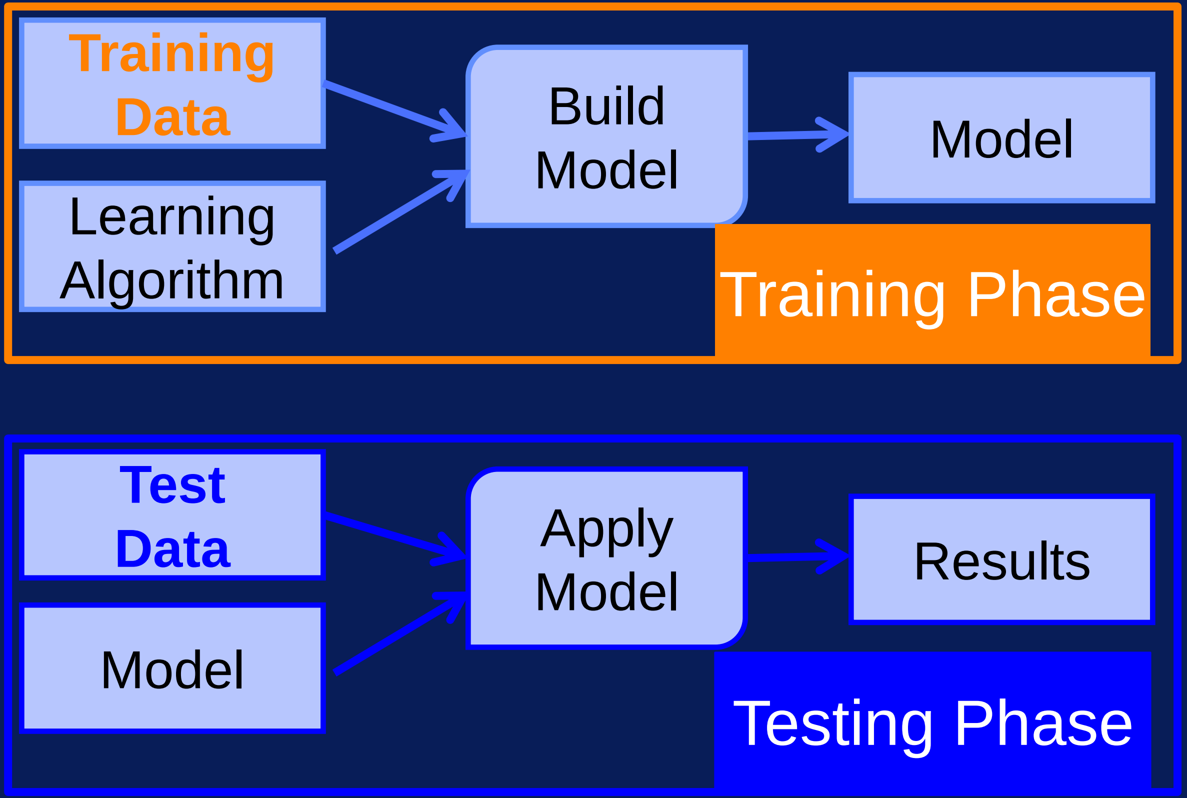
Input Variables

Target Variable

Target is provided

Today's High	Today's Low	Month	Tomorrow's High
79	64	July	81
60	45	October	58
68	49	May	65
57	47	January	54

Training vs. Testing Phases

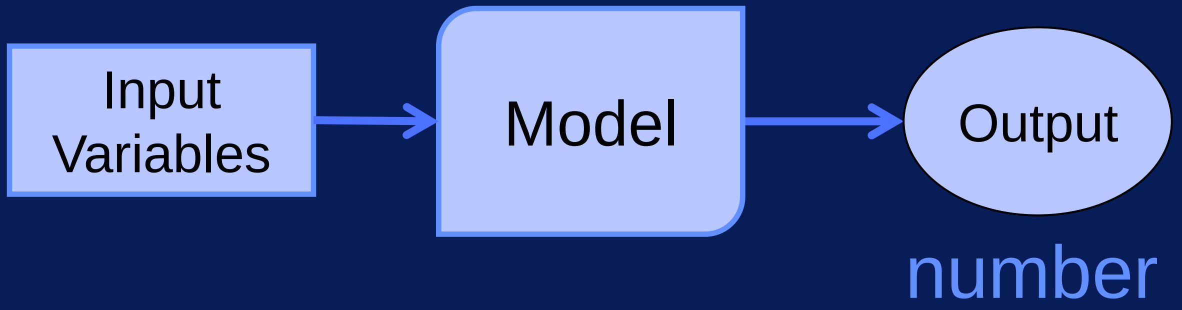


Datasets

Training Data	Validation Data	Test Data
Adjust model parameters	Determine when to stop training (avoid overfitting) Estimate generalization performance	Evaluate performance on new data

Regression Main Points

- Predict number from input variables
- Regression is a supervised task
- Target variable is numerical



Linear Regression

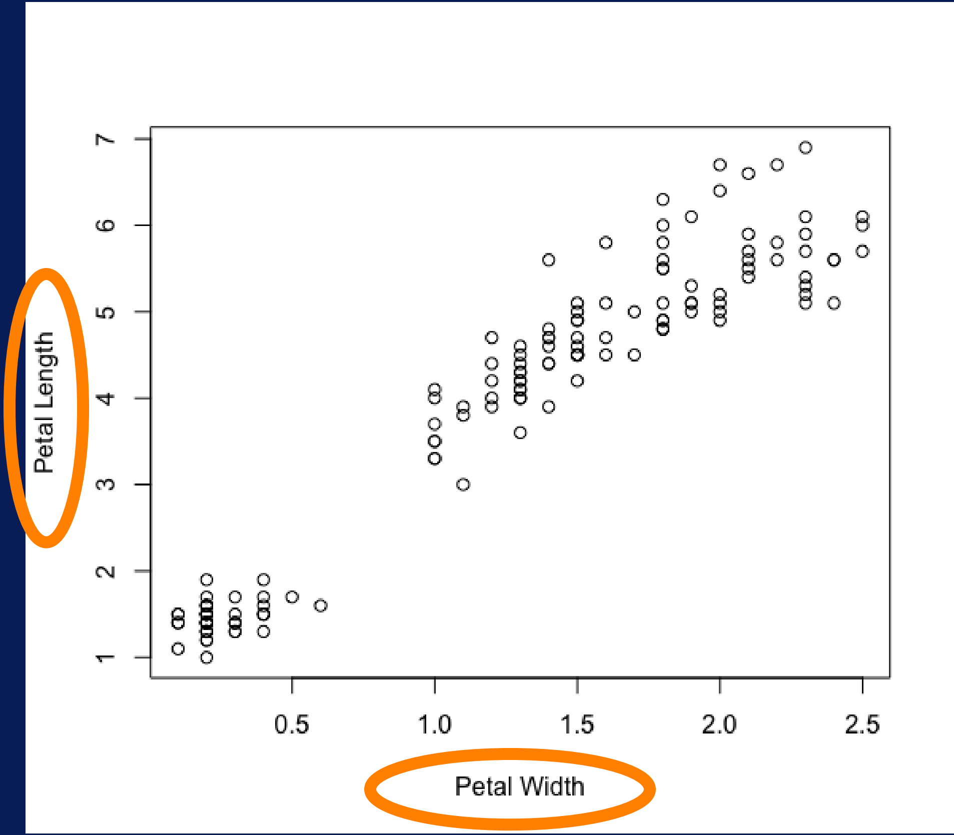
After this video you will be able to..

- Describe how linear regression works
- Discuss how least squares is used in linear regression
- Define simple and multiple linear regression

Linear Regression

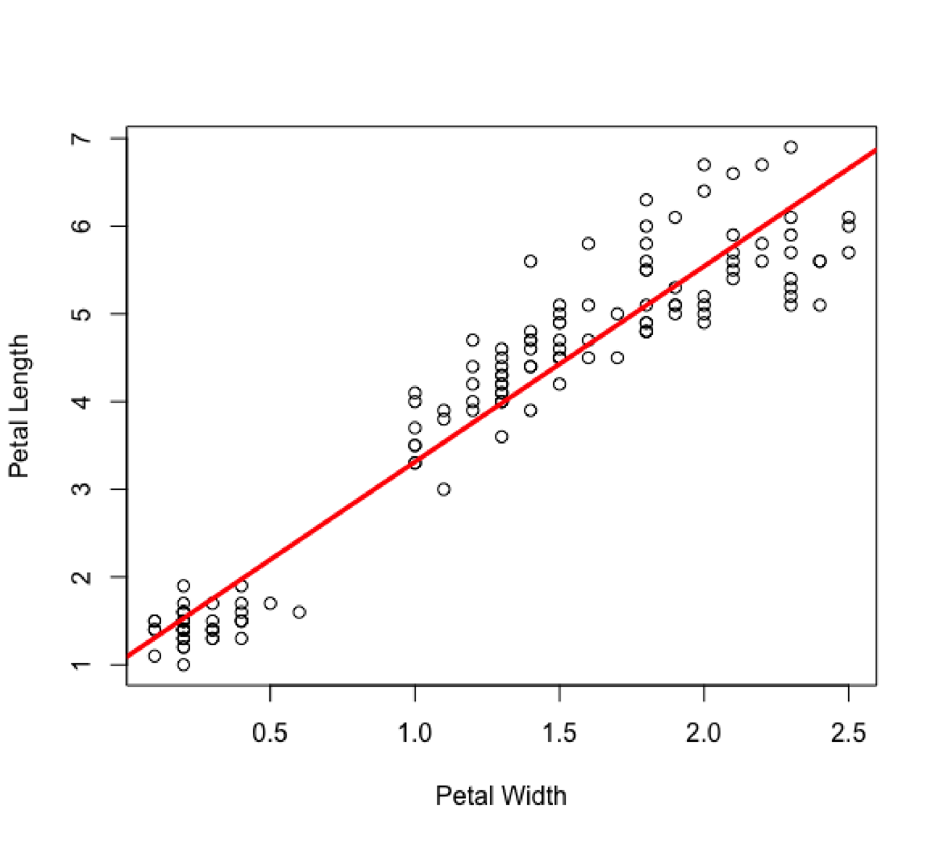
- Captures relationship between numerical output and input variables
- Relationship is modeled as linear

Linear Regression Model



Regression Task:
Given petal width, predict petal length.

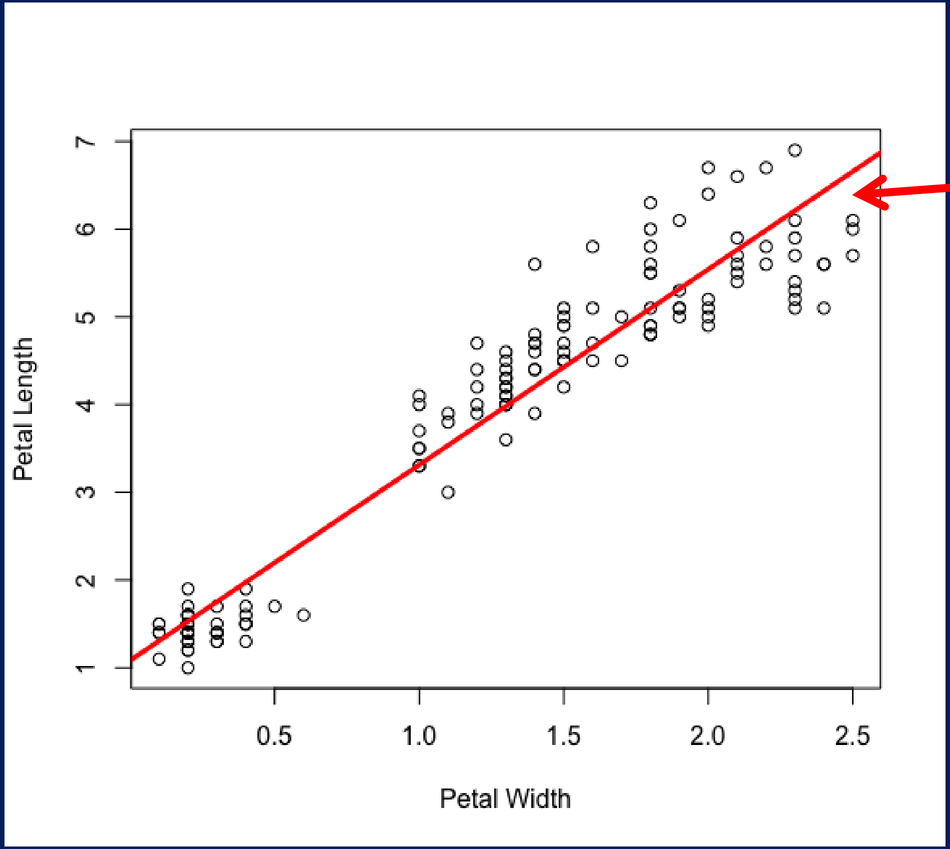
Linear Regression Model



Regression Task:

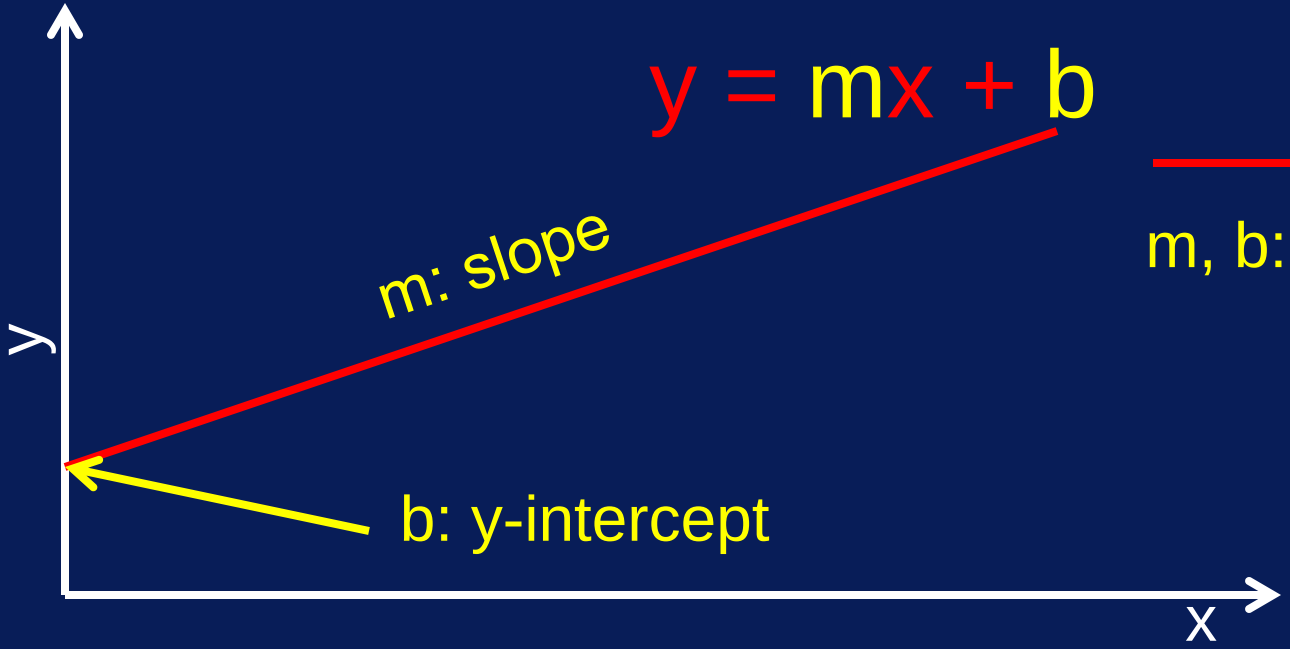
Given petal width, predict petal length.

Linear Regression Model



regression line

Least Squares Algorithm

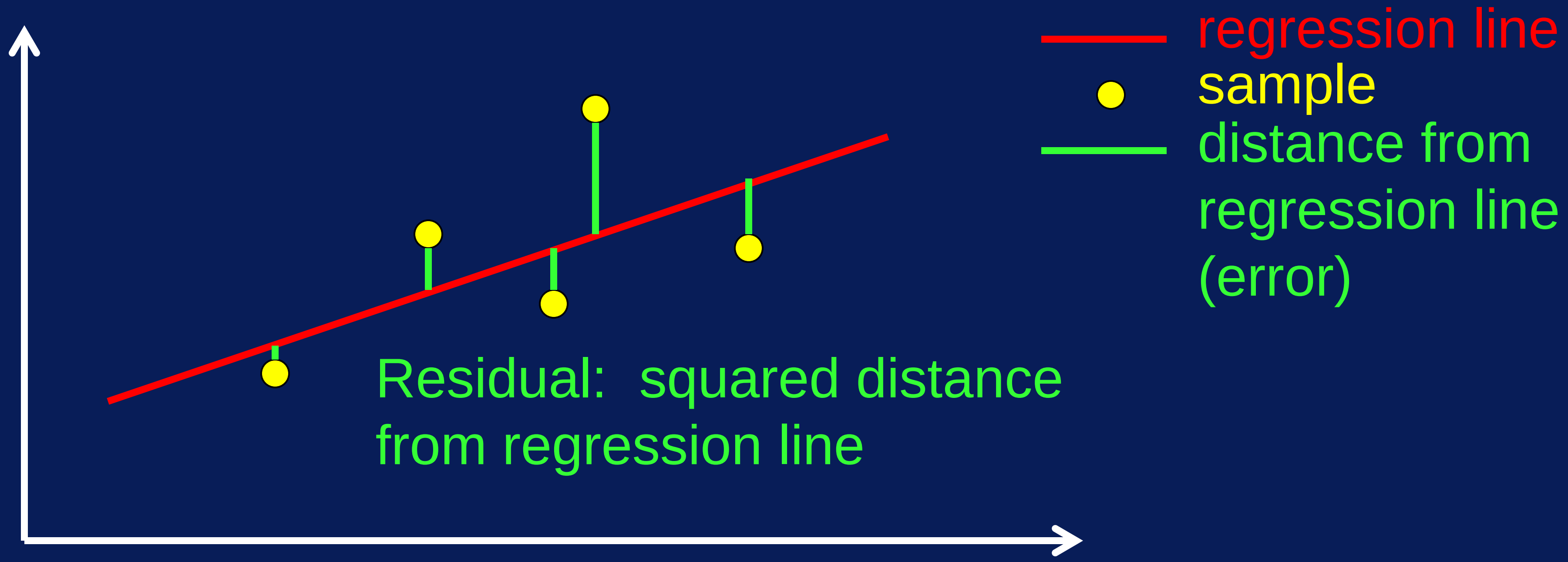


regression line

m, b: model parameters

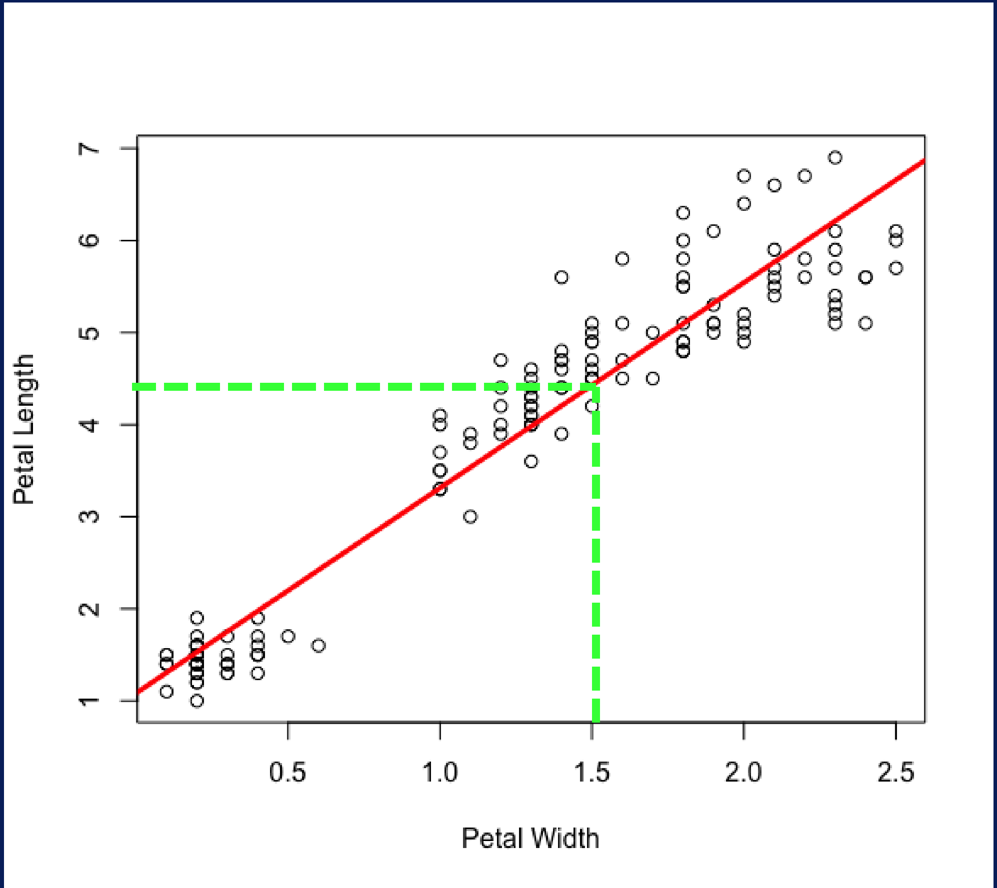
Training linear regression model adjusts model parameters to fit samples

Least Squares Method



Goal: Find regression line that makes sum of residuals as small as possible

Linear Regression Model



Applying model:

Given petal width =1.5, prediction is

petal length = 4.5

Types of Linear Regression

Simple Linear Regression

var1

Input has one variable

Multiple Linear Regression

var1 var2 … varN

Input has >1 variables

Linear Regression Summary

- Captures linear relationship between numerical output and input variables
- Model can be fitted using least squares

Cluster Analysis

After this video you will be able to..

- Articulate the goal of cluster analysis
- Discuss whether cluster analysis is supervised or unsupervised.
- List some ways that cluster results can be applied

Cluster Analysis Overview

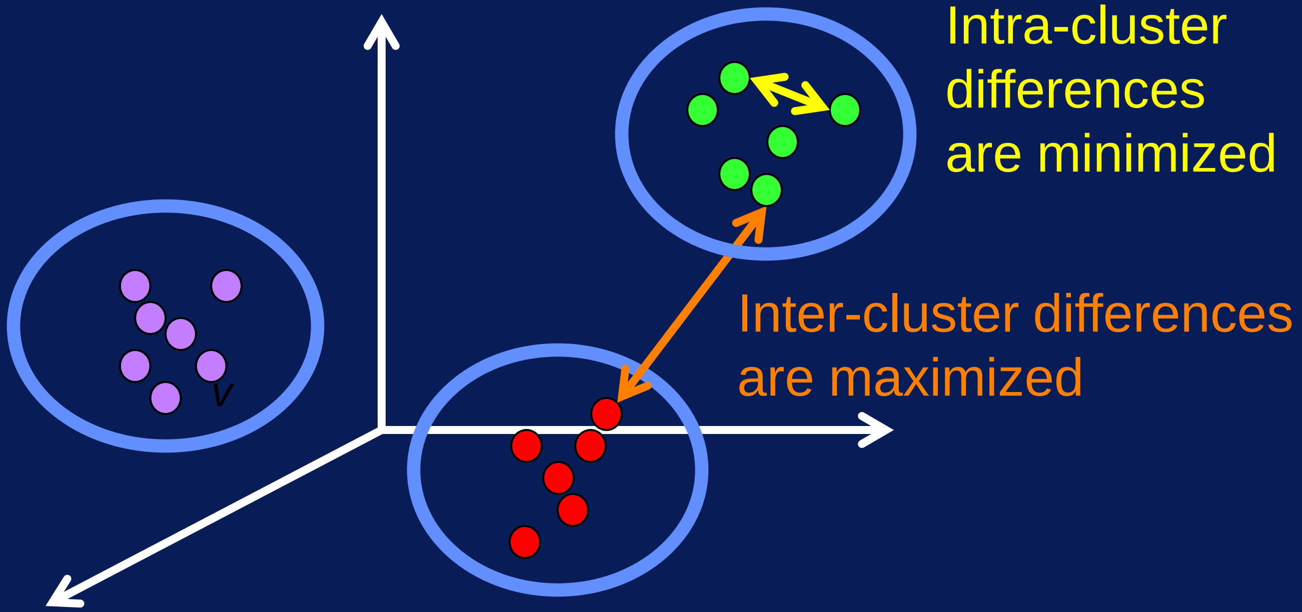
Goal: Organize similar items into groups.

Cluster Analysis Examples

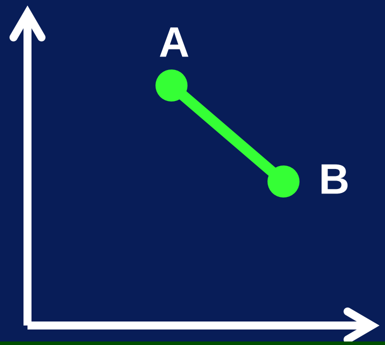
- Segment customer base into groups
- Characterize different weather patterns for a region
- Group news articles into topics
- Discover crime hot spots

Cluster Analysis

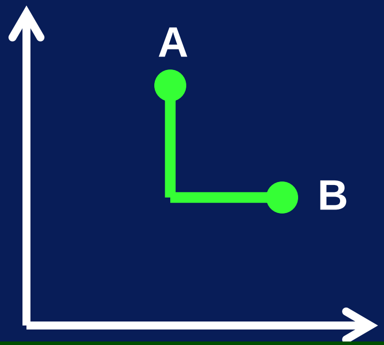
- Divides data into clusters
- Similar items are placed in same cluster



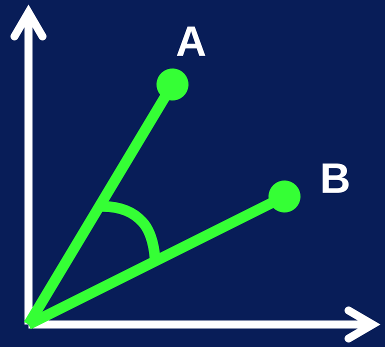
Similarity Measures



Euclidean Distance

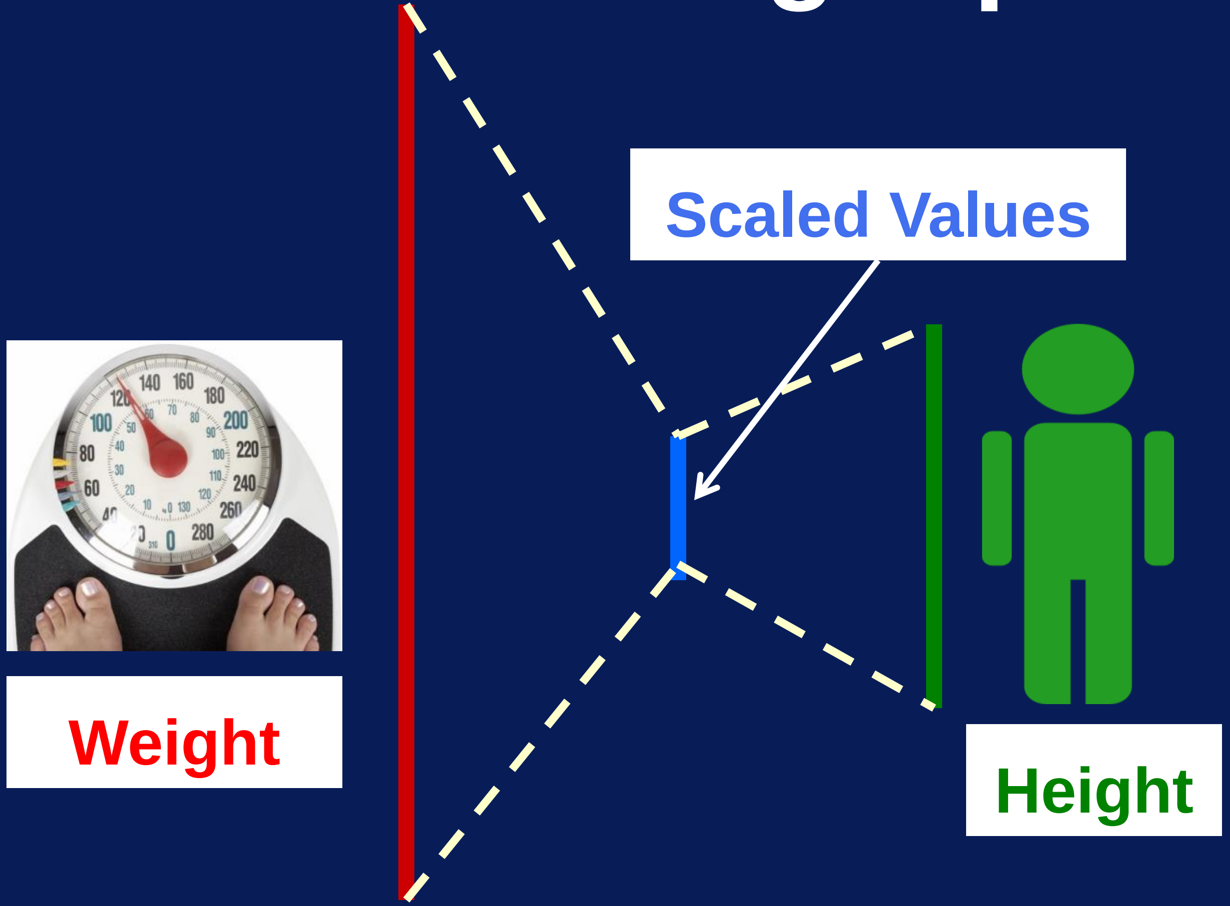


Manhattan Distance



Cosine Similarity

Normalizing Input Variables



Cluster Analysis Notes

Unsupervised

There is no 'correct' clustering

Clusters don't come with labels

Interpretation and analysis required to make sense of clustering results!

Uses of Cluster Results

- **Data segmentation**
  - Analysis of each segment can provide insights

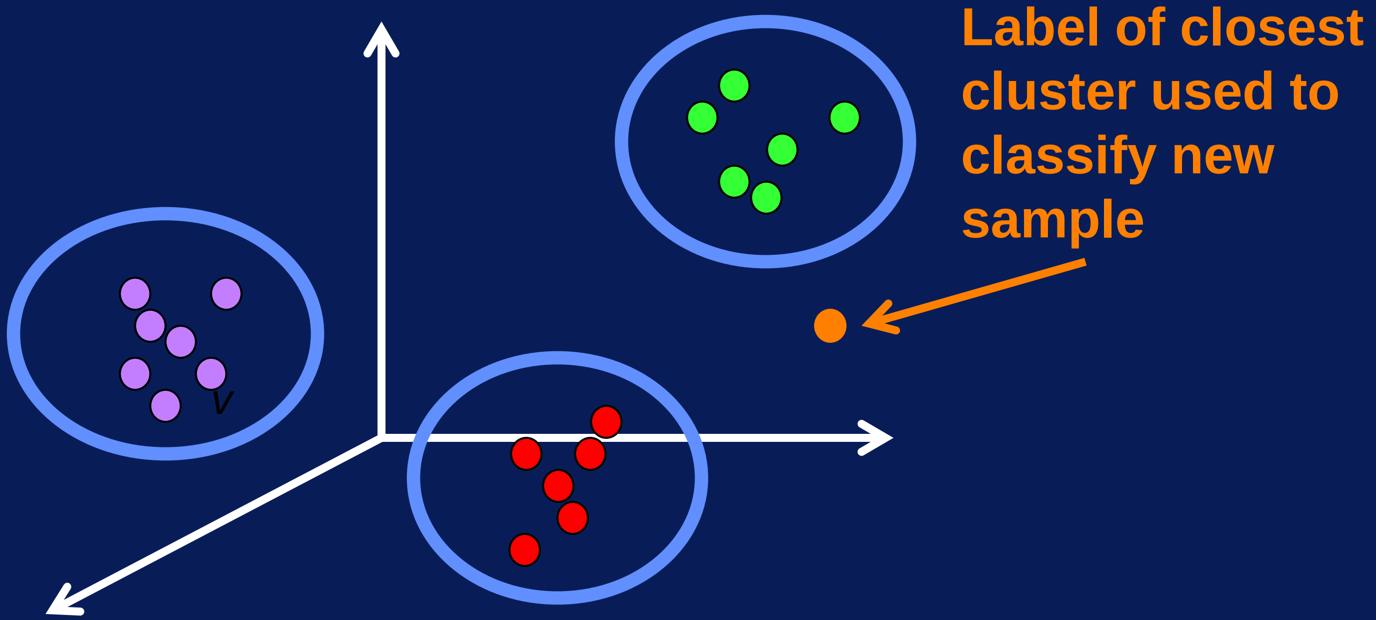
science fiction

non-fiction

children's

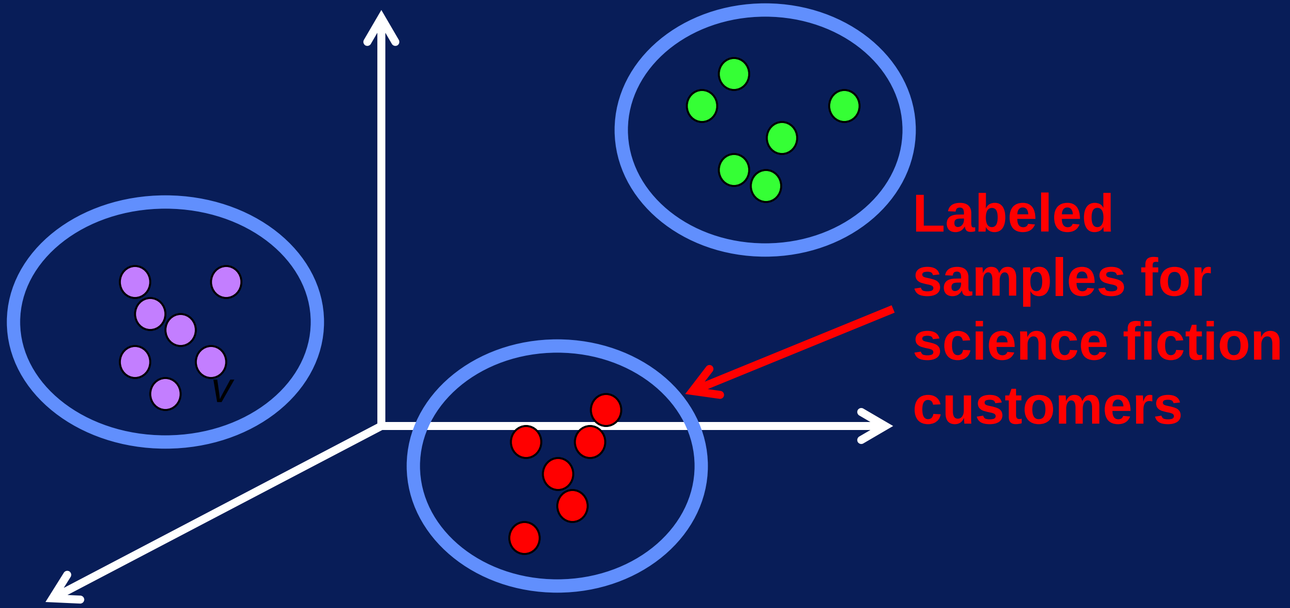
Uses of Cluster Results

- **Categories for classifying new data**
  - New sample assigned to closest cluster



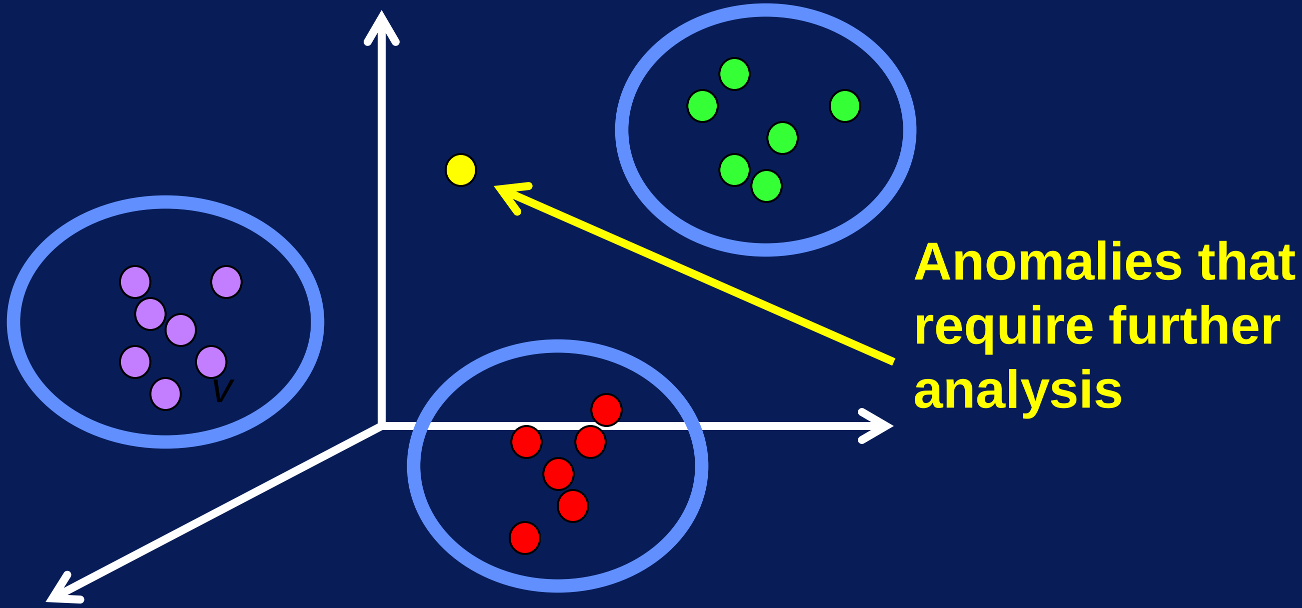
Uses of Cluster Results

- **Labeled data for classification**
  - Cluster samples used as labeled data



Uses of Cluster Results

- **Basis for anomaly detection**
  - Cluster outliers are anomalies



Cluster Analysis Summary

- Organize similar items into groups
- Analyzing clusters often leads to useful insights about data
- Clusters require analysis and interpretation

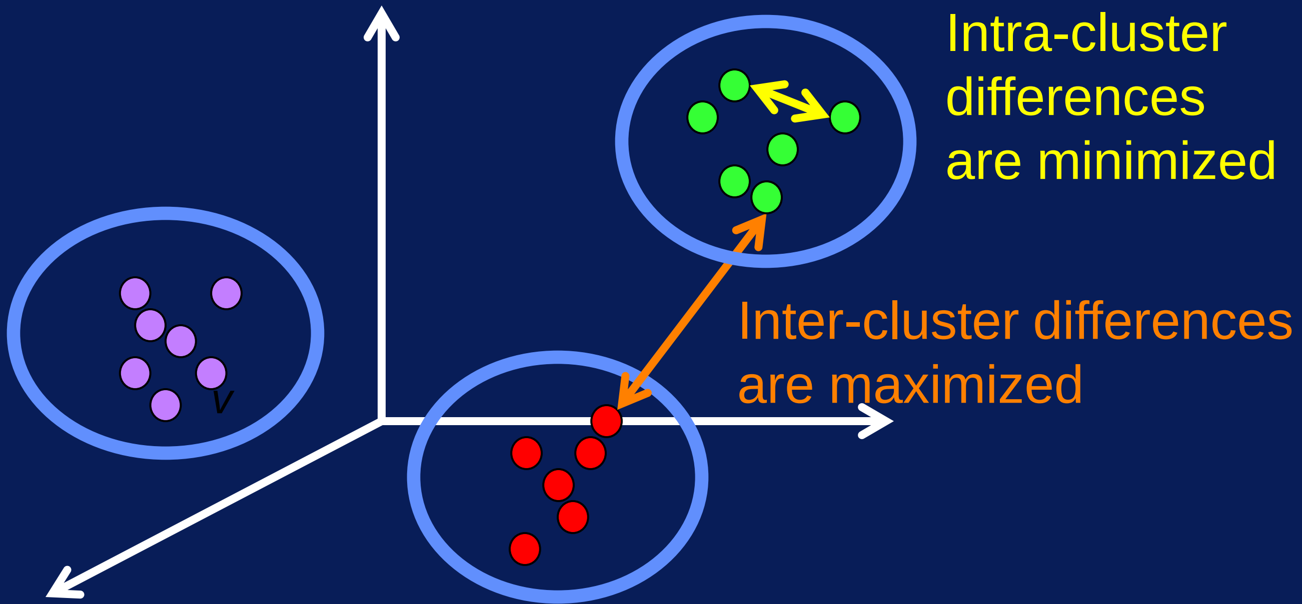
k-Means Clustering

After this video you will be able to..

- Describe the steps in the k-means algorithm
- Explain what the ‘k’ stands for in k-means
- Define what a cluster centroid is

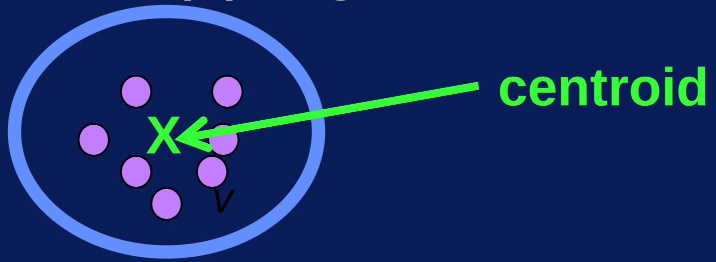
Cluster Analysis

- **Divides data into clusters**
- **Similar items are in same cluster**

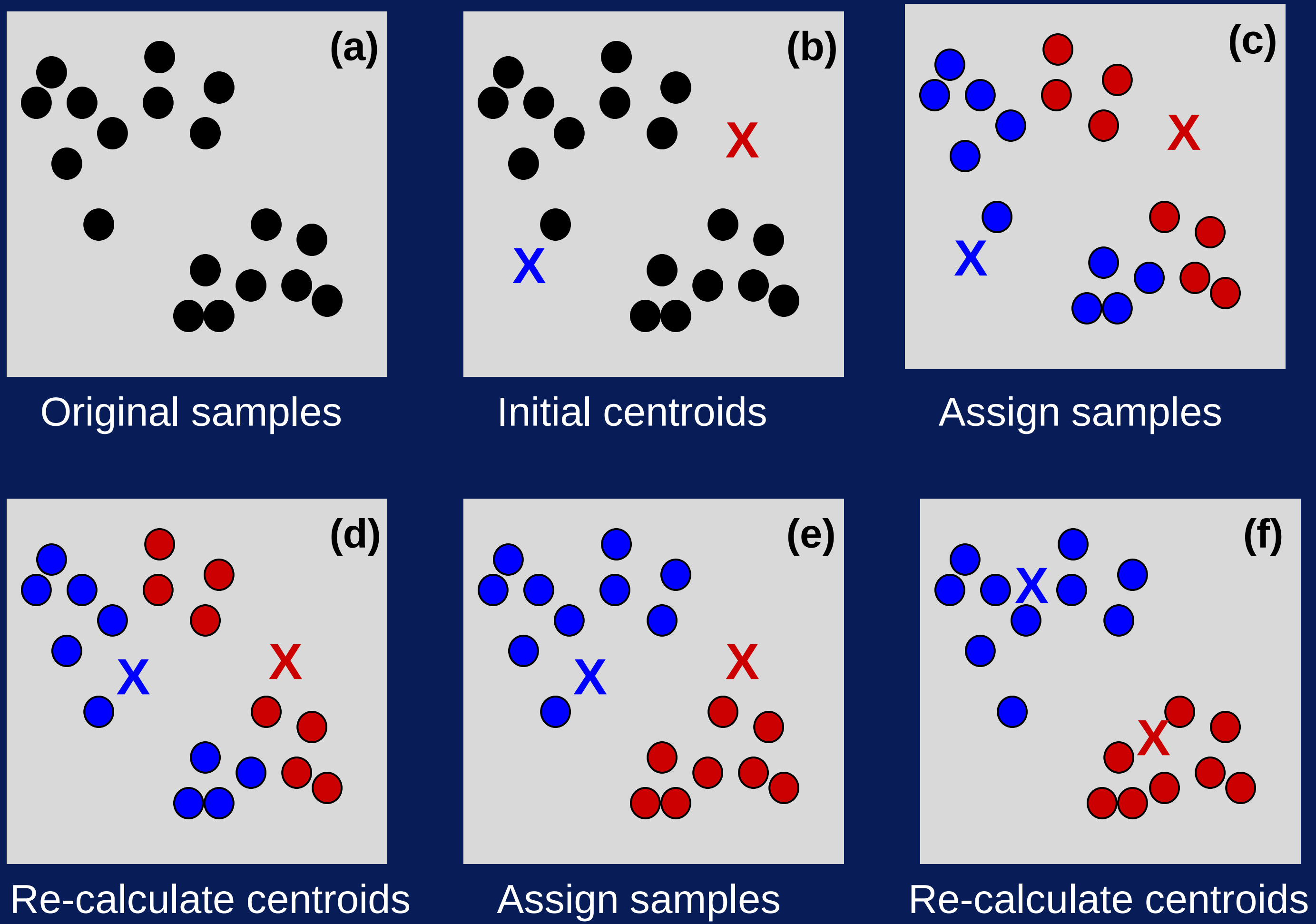


k-Means Algorithm

- Select k initial centroids (cluster centers)
- Repeat
  - Assign each sample to closest centroid
  - Calculate mean of cluster to determine new centroid
- Until some stopping criterion is reached



k-Means

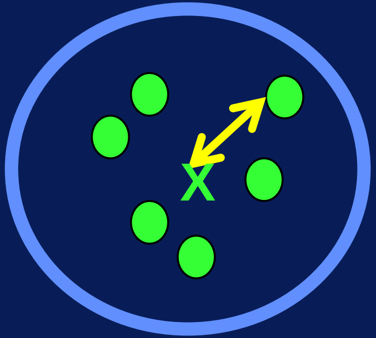

(a)
Original samples
(b)
Initial centroids
(c)
Assign samples
(d)
Re-calculate centroids
(e)
Assign samples
(f)
Re-calculate centroids

Choosing Initial Centroids

Issue:
Final clusters are sensitive to initial centroids

Solution:
Run k-means multiple times with different random initial centroids, and choose best results

Evaluating Cluster Results



error = distance between sample & centroid

squared error = error^2

Sum of squared errors
between all samples & centroid

Sum over all clusters

WSSE

Within-Cluster Sum of Squared Error

Using WSSE

$WSSE_1 < WSSE_2$ → WSSE1 is better *numerically*

Caveats:

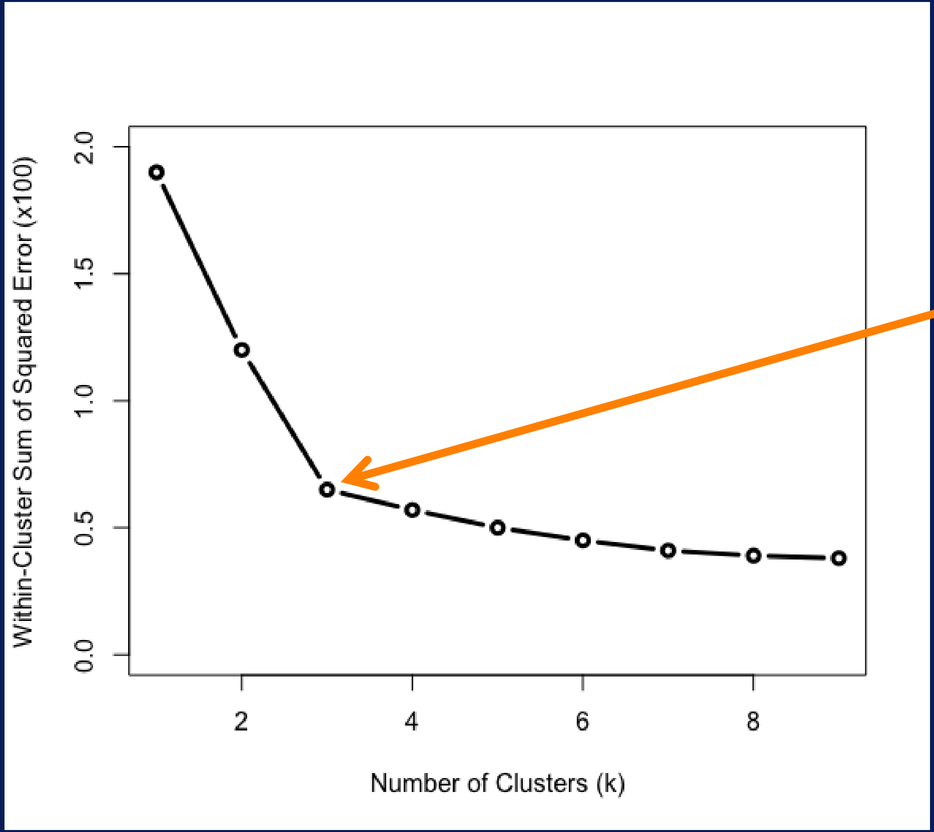
- Does not mean that cluster set 1 is more ‘correct’ than cluster set 2
- Larger values for k will always reduce WSSE

Choosing Value for k

- **Approaches:**
  - Visualization
  - Application-Dependent
  - Data-Driven

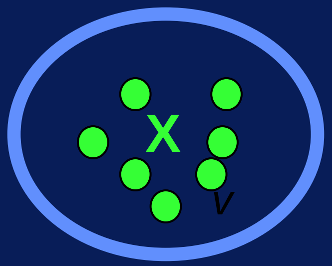
$\boxed{k = ?}$

Elbow Method for Choosing k



“Elbow” suggests value for k should be 3

Stopping Criteria

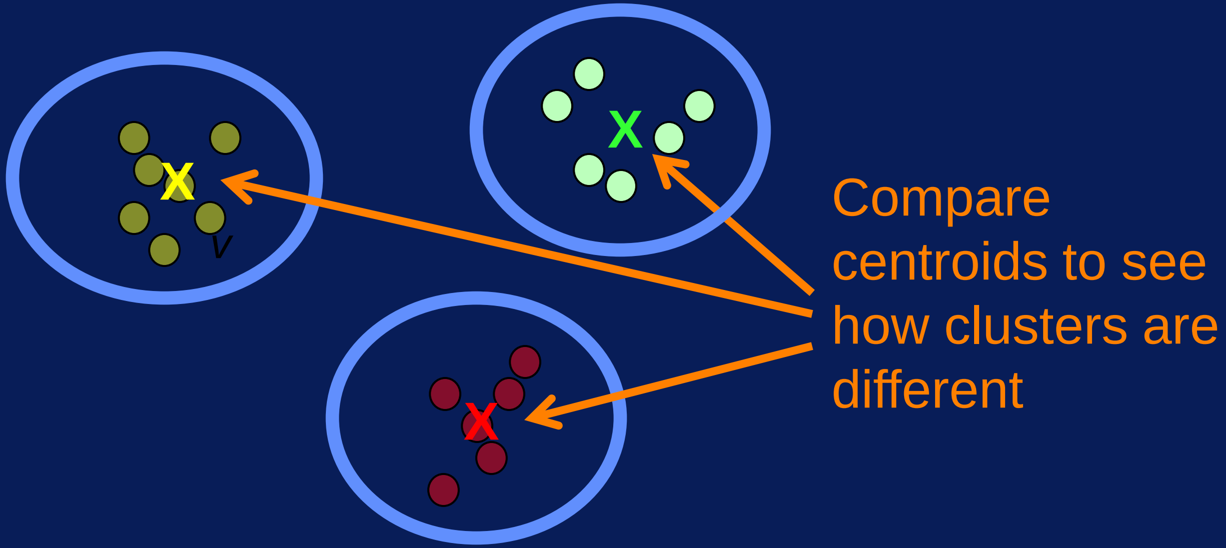


When to stop iterating?

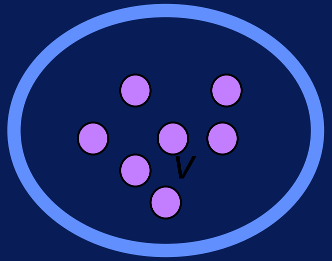
- No changes to centroids
- Number of samples changing clusters is below threshold

Interpreting Results

- **Examine cluster centroids**
  - How are clusters different?



K-Means Summary



- Classic algorithm for cluster analysis
- Simple to understand and implement and is efficient
- Value of k must be specified
- Final clusters are sensitive to initial centroids

Association Analysis

After this video you will be able to..

- Explain what association analysis entails
- List some applications of association analysis
- Define what an item set is

Association Analysis

Goal: Find rules to capture associations between items.

Association Analysis Examples

- Have sales on related items often purchased together
- Recommend items based on purchase/browsing history
- Identify effective treatments to patients

Analysis Association Overview

ID	Items
1	diapers, bread, milk
2	bread, diapers, beer, eggs
3	milk, diapers, beer, butter
4	bread, milk, diapers, beer
5	bread, milk, diapers, butter

Item Sets

{bread, milk} => {diapers}
{milk) => {bread}

Rules

If bread and milk are bought, then diaper is also bought

Association Analysis Steps

1. Create item sets

{bread} {butter} {bread, milk} {milk, beer}

2. Identify frequent item sets

{bread} {bread, milk}

3. Generate rules

{bread, milk} => {diapers}

Association Analysis Notes

Unsupervised

Usefulness of rules is subjective

Need to determine how to use rules

Interpretation and analysis required to make sense of resulting rules!

Association Analysis Summary

- Find rules to capture associations between items
- Rules have intuitive appeal
- Resulting rules require analysis and interpretation using domain knowledge

Association Analysis in Detail

After this video you will be able to..

- Define the terms ‘support’ and ‘confidence’
- Describe the steps in association analysis
- Explain how association rules are formed from item sets

Association Analysis Steps

1. **Create item sets**

   {bread} {butter} {bread, milk} {bread, beer}

2. **Identify frequent item sets**

   {bread} {bread, beer}

3. **Generate rules**

   {bread, milk} => {diapers}

Analysis Association Dataset

ID	Items
1	diapers, bread, milk
2	bread, diapers, beer, eggs
3	milk, diapers, beer, butter
4	bread, milk, diapers, beer
5	bread, milk, diapers, butter

Item Sets

{bread, milk} => {diapers}
{milk) => {bread}

Rules

If bread and milk are bought, then diapers are also bought

Adapted from http://www-users.cs.umn.edu/~kumar/dmbook/index.php

Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

1-Item Sets

Item	Support
bread	4/5
butter	2/5
milk	4/5
beer	3/5
diaper	5/5
eggs	1/5

Support = frequency of item set

‘diaper’ occurs in all transactions

‘eggs’ occurs only once, in transaction 2

Create Item Sets

minimum support = 3/5

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

1-Item Sets

Item	Support
{bread}	4/5
{butter}	2/5
{milk}	4/5
{beer}	3/5
{diaper}	5/5
{eggs}	1/5

Remove these item sets since they have low support.

Create Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,beer}	2/5
{bread,diaper}	4/5
{milk,beer}	2/5
{milk,diaper}	4/5
{beer,diaper}	3/5

minimum support = 3/5

1-item sets:
{bread}, {milk}, {diaper}

‘beer’ and ‘diaper’ occur together 3 times, in transactions 2, 3, & 4

Create Item Sets

minimum support = 3/5

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,beer}	2/5
{bread,diaper}	4/5
{milk,beer}	2/5
{milk,diaper}	4/5
{beer,diaper}	3/5

1-item sets:
{bread}, {milk}, {diaper}

Remove these item sets
since they have low support.

Create Item Sets

minimum support = 3/5

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

Only 3-item set with support > minimum support

1-item sets:
{bread},
{milk},
{diaper}

2-item sets:
{bread,milk},
{bread,diaper},
{milk,diaper},
{beer,diaper}

Frequent Item Sets

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

minimum support = 3/5

1-Item Sets

Item	Support
{bread}	4/5
{milk}	4/5
{diaper}	5/5

2-Item Sets

Item	Support
{bread,milk}	3/5
{bread,diaper}	4/5
{milk,diaper}	4/5
{beer,diaper}	3/5

3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

Rule Terms

Rule

Antecedent $X \rightarrow Y$ Consequent

If X, then Y

Rule Confidence

$$\text{conf}\,(X \rightarrow Y) = \frac{\text{supp}\,(X \cup Y)}{\text{supp}\,(X)}$$

support for X & Y together

support for X

Itemset Support

$$\text{supp}\,(X) = \frac{\#\ \text{transactions with}\ X}{\text{total}\ \#\ \text{transactions}}$$

Rule Generation & Pruning

frequent item sets → association rules

each k-item set → 2^k-2 rules!

frequent item sets → significant rules

Use rule confidence to
constrain rule generation

Keep rule if confidence > minimum confidence

Rule Example

ID	Items
1	diaper, bread, milk
2	bread, diaper, beer, eggs
3	milk, diaper, beer, butter
4	bread, milk, diaper, beer
5	bread, milk, diaper, butter

3-Item Sets

Item	Support
{bread,milk, diaper}	3/5

min confidence = 0.95

$$\text{conf}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$$

Candidate rule: {bread,milk} → {diaper}

$$\text{conf} = \frac{\text{supp (bread,milk,diaper)}}{\text{supp (bread,milk)}} = \frac{3/5}{3/5} = \frac{3}{3} = 1.0$$

Candidate rule: {bread,diaper} → {milk}

$$\text{conf} = \frac{\text{supp (bread,diaper,milk)}}{\text{supp (bread,diaper)}} = \frac{3/5}{4/5} = \frac{3}{4} = 0.75$$

Association Analysis Algorithms

- Use different methods to make efficient:
  - item set creation
  - rule generation efficient
- Algorithms:

  Apriori FP Growth Eclat

Association Analysis Steps

- Item sets created from data
- Frequent item sets identified using support
- Rules generated from frequent item sets and pruned using confidence

Description of Minute Weather Dataset

The minute weather dataset comes from the same source as the daily weather dataset that was used for the hands-on activities in the previous modules. The difference is that the minute weather dataset contains raw sensor measurements captured at one-minute intervals, not processed like the daily weather dataset. The data is in the file *minute_weather.csv*, which is a comma-separated file.

As with the daily weather data, this data comes from a weather station located in San Diego, California. The weather station is equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured.

Each row in minute_weather.csv contains weather data captured for a one-minute interval. Each row, or sample, consists of the following variables:

Variable	**Description**	**Unit of Measure**
rowID	unique number for each row	NA
hpwren_timestamp	timestamp of measure	year-month-day hour:minute:second
air_pressure	air pressure measured at the timestamp	hectopascals
air_temp	air temperature measure at the timestamp	degrees Fahrenheit
avg_wind_direction	wind direction averaged over the minute before the timestamp	degrees, with 0 means coming from the North, and increasing clockwise

avg_wind_speed	wind speed averaged over the minute before the timestamp	meters per second
max_wind_direction	highest wind direction in the minute before the timestamp	degrees, with 0 being North and increasing clockwise
max_wind_speed	highest wind speed in the minute before the timestamp	meters per second
min_wind_direction	smallest wind direction in the minute before the timestamp	degrees, with 0 being North and inceasing clockwise
min_wind_speed	smallest wind speed in the minute before the timestamp	meters per second
rain_accumulation	amount of accumulated rain measured at the timestamp	millimeters
rain_duration	length of time rain has fallen as measured at the timestamp	seconds
relative_humidity	relative humidity measured at the timestamp	percent

Cluster Analysis in Spark

Problem Description

This activity guides you through the process of performing cluster analysis on a dataset using k-means. In this activity, we will perform cluster analysis on the *minute-weather.csv* dataset using the k-means algorithm. Recall that this dataset contains weather measurements such as temperature, relative humidity, etc., from a weather station in San Diego, California, collected at one-minute intervals. The goal of cluster analysis on this data is to identify different weather patterns for this weather station.

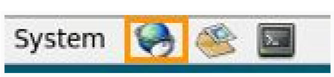
Learning Objectives

By the end of this activity, you will be able to:

1. Scale all features so that each feature is zero-normalized
2. Create an "elbow" plot, the number of clusters vs. within-cluster sum-of-squared errors, to determine a value for k, the number of clusters in k-means
3. Perform cluster analysis on a dataset using k-means
4. Create parallel coordinates plots to analyze cluster centers

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 1. **Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to *localhost:8889/tree/Downloads/big-data-4*:

Open the clustering notebook by clicking on *clustering.ipynb:*

Step 2. **Load minute weather data.** Execute the first cell to load the classes used in this activity:

```
In [1]: from pyspark.sql import SQLContext
        from pyspark.ml.clustering import KMeans
        from pyspark.ml.feature import VectorAssembler
        from pyspark.ml.feature import StandardScaler
        from notebooks import utils
        %matplotlib inline
```

Execute the second cell to load the minute weather data in *minute_weather.csv:*

```
In [2]: sqlContext = SQLContext(sc)
        df = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/minute_weather.csv',
                                  format='com.databricks.spark.csv',
                                  header='true',inferSchema='true')
```

Step 3. **Subset and remove unused data**. Let's count the number of rows in the DataFrame:

```
In [3]: df.count()

Out[3]: 1587257
```

There are over 1.5 million rows in the DataFrame. Clustering this data on your computer in the Cloudera VM can take a long time, so let's only one-tenth of the data. We can subset by calling *filter()* and using the *rowID* column:

```
In [4]: filteredDF = df.filter((df.rowID % 10) == 0)
        filteredDF.count()

Out[4]: 158726
```

Let's compute the summary statistics using *describe():*

```
In [5]: filteredDF.describe().toPandas().transpose()
```

Out[5]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
rowID	158726	793625.0	458203.9375103623	0	1587250
air_pressure	158726	916.8301614102434	3.051716552830638	905.0	929.5
air_temp	158726	61.851589153636304	11.833569210641757	31.64	99.5
avg_wind_direction	158680	162.15610032770354	95.27820101905898	0.0	359.0
avg_wind_speed	158680	2.775214897907747	2.057623969742642	0.0	31.9
max_wind_direction	158680	163.46214393748426	92.45213853838689	0.0	359.0
max_wind_speed	158680	3.400557726241518	2.4188016208098886	0.1	36.0
min_wind_direction	158680	166.77401688933702	97.44110914784567	0.0	359.0
min_wind_speed	158680	2.1346641038568754	1.7421125052424393	0.0	31.6
rain_accumulation	158725	3.178453299732825E-4	0.011235979086039813	0.0	3.12
rain_duration	158725	0.4096267128681682	8.665522693479772	0.0	2960.0
relative_humidity	158726	47.609469778108206	26.214408535062027	0.9	93.0

The weather measurements in this dataset were collected during a drought in San Diego. We can count the how many values of rain accumulation and duration are 0:

```
In [6]: filteredDF.filter(filteredDF.rain_accumulation == 0.0).count()

Out[6]: 157812
```

```
In [7]: filteredDF.filter(filteredDF.rain_duration == 0.0).count()

Out[7]: 157237
```

Since most the values for these columns are 0, let's drop them from the DataFrame to speed up our analyses. We can also drop the *hpwren_timestamp* column since we do not use it.

```
In [8]: workingDF = filteredDF.drop('rain_accumulation').drop('rain_duration').drop('hpwren_timestamp')
```

Let's drop rows with missing values and count how many rows were dropped:

```
In [9]: before = workingDF.count()
        workingDF = workingDF.na.drop()
        after = workingDF.count()
        before - after

Out[9]: 46
```

Step 4. **Scale the data**. Since the features are on different scales (e.g., air pressure values are in the 900's, while relative humidities range from 0 to 100), they need to be scaled. We will scale them so that each feature will have a value of 0 for the mean, and a value of 1 for the standard deviation.

First, we will combine the columns into a single vector column. Let's look at the columns in the DataFrame:

```
In [10]: workingDF.columns

Out[10]: ['rowID',
          'air_pressure',
          'air_temp',
          'avg_wind_direction',
          'avg_wind_speed',
          'max_wind_direction',
          'max_wind_speed',
          'min_wind_direction',
          'min_wind_speed',
          'relative_humidity']
```

We do not want to include *rowID* since it is the row number. The minimum wind measurements have a high correlation to the average wind measurements, so we will not include them either. Let's create an array of the columns we want to combine, and use *VectorAssembler* to create the vector column:

```
In [11]: featuresUsed = ['air_pressure', 'air_temp', 'avg_wind_direction', 'avg_wind_speed', 'max_wind_direction',
                 'max_wind_speed','relative_humidity']
         assembler = VectorAssembler(inputCols=featuresUsed, outputCol="features_unscaled")
         assembled = assembler.transform(workingDF)
```

Next, let's use *StandardScaler* to scale the data:

```
In [12]: scaler = StandardScaler(inputCol="features_unscaled", outputCol="features", withStd=True, withMean=True)
         scalerModel = scaler.fit(assembled)
         scaledData = scalerModel.transform(assembled)
```

The *withMean* argument specifies to center the data with the mean before scaling, and *withStd* specifies to scale the data to the unit standard deviation.

Step 5. **Create elbow plot.** The k-means algorithm requires that the value of k, the number of clusters, to be specified. To determine a good value for *k*, we will use the "elbow" method. This method involves applying k-means, using different values for *k*, and calculating the within-cluster sum-of-squared error (WSSE). Since this means applying k-means multiple times, this process can be very compute-intensive. To speed up the process, we will use only a subset of the dataset. We will take every third sample from the dataset to create this subset:

```
In [13]: scaledData = scaledData.select("features", "rowID")

         elbowset = scaledData.filter((scaledData.rowID % 3) == 0).select("features")
         elbowset.persist()
```

The last line calls the *persist()* method to tell Spark to keep the data in memory (if possible), which will speed up the computations.

Let's compute the k-means clusters for *k* = 2 to 30 to create an elbow plot:

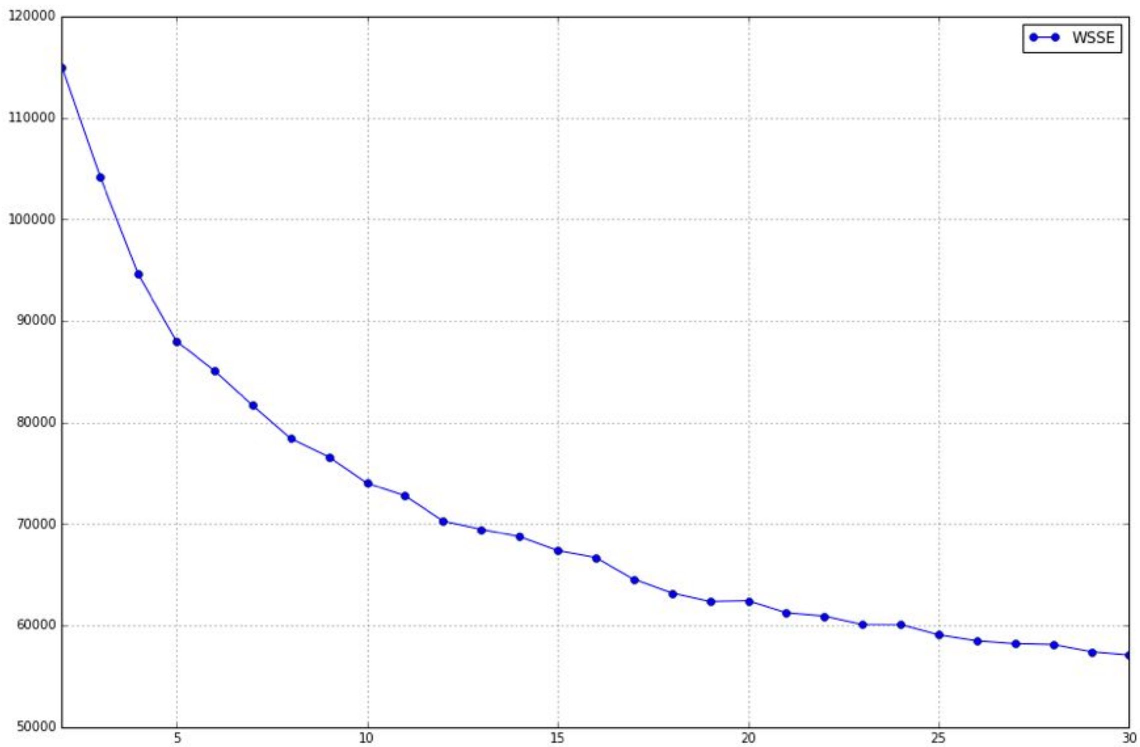
```
In [14]: clusters = range(2,31)
         wsseList = utils.elbow(elbowset, clusters)

         Training for cluster size 2
         ......................WSSE = 114993.13181214454
         Training for cluster size 3
         ......................WSSE = 104181.0978581738
         Training for cluster size 4
         ......................WSSE = 94577.27151288436
         Training for cluster size 5
         ......................WSSE = 87993.46098415818
         Training for cluster size 6
         ......................WSSE = 85084.23922296542
         Training for cluster size 7
         ......................WSSE = 81664.96024487517
         Training for cluster size 8
```

The first line creates an array with the numbers 2 through 30, and the second line calls the *elbow()* function defined in the *utils.py* library to perform clustering. The first argument to *elbow()* is the dataset, and the second is the array of values for *k*. The *elbow()* function returns an array of the WSSE for each number of clusters.

Let's plot the results by calling *elbow_plot()* in the *utils.py* library:

```
In [15]: utils.elbow_plot(wsseList, clusters)
```



The values for *k* are plotted against the WSSE values, and the elbow, or bend in the curve, provides a good estimate for the value for *k*. In this plot, we see that the elbow in the curve is between 10 and 15, so let's choose *k* = 12. We will use this value to set the number of clusters for k-means.

Step 6. **Cluster using selected k**. Let's select the data we want to cluster:

```
In [16]: scaledDataFeat = scaledData.select("features")
         scaledDataFeat.persist()
```

Again, we call the *persist()* method to cache the data in memory for faster access.

We can perform clustering using *KMeans:*

```
In [17]: kmeans = KMeans(k=12, seed=1)
         model = kmeans.fit(scaledDataFeat)
         transformed = model.transform(scaledDataFeat)
```

The first line creates a new *KMeans* instance with 12 clusters and a specific seed value. (As in previous hands-on activities, we use a specific seed value for reproducible results.) The second line fits the data to the model, and the third applies the model to the data.

Once the model is created, we can determine the center measurement of each cluster:

```
In [18]: centers = model.clusterCenters()
         centers

Out[18]: [array([-0.13720796,  0.6061152 ,  0.22970948, -0.62174454,  0.40604553,
                 -0.63465994, -0.42215364]),
          array([ 1.42238994, -0.10953198, -1.10891543, -0.07335197, -0.96904335,
                 -0.05226062, -0.99615617]),
          array([-0.63637648,  0.01435705, -1.1038928 , -0.58676582, -0.96998758,
                 -0.61362174,  0.33603011]),
          array([-0.22385278, -1.06643622,  0.5104215 , -0.24620591,  0.68999967,
                 -0.24399706,  1.26206479]),
          array([ 1.17896517, -0.25134204, -1.15089838,  2.11902126, -1.04950228,
                  2.23439263, -1.12861666]),
          array([-1.14087425, -0.979473  ,  0.42483303,  1.68904662,  0.52550171,
                  1.65795704,  1.03863542]),
          array([ 0.50746307, -1.08840683, -1.20882766, -0.57604727, -1.0367013 ,
                 -0.58206904,  0.97099067]),
          array([ 0.14064028,  0.83834618,  1.89291279, -0.62970435, -1.54598923,
                 -0.55625032, -0.75082891]),
          array([-0.0339489 ,  0.98719067, -1.33032244, -0.57824562, -1.18095582,
                 -0.58893358, -0.81187427]),
          array([-0.22747944,  0.59239924,  0.40531475,  0.6721331 ,  0.51459992,
                  0.61355559, -0.15474261]),
          array([ 0.3334222 , -0.99822761,  1.8584392 , -0.68367089, -1.53246714,
                 -0.59099434,  0.91004892]),
          array([ 0.3051367 ,  0.67973831,  1.36434828, -0.63793718,  1.631528  ,
                 -0.58807924, -0.67531539])]
```

It is difficult to compare the cluster centers by just looking at these numbers. So we will use plots in the next step to visualize them.

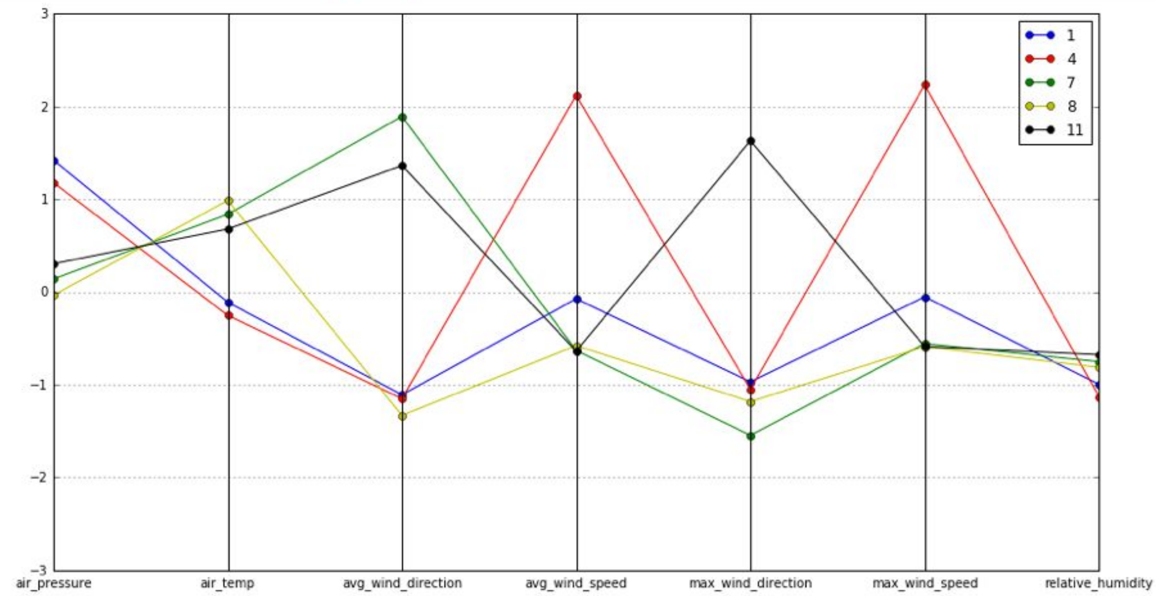
Step 7. **Create parallel plots of clusters and analysis.** A parallel coordinates plot is a great way to visualize multi-dimensional data. Each line plots the centroid of a cluster, and all of the features are plotted together. Recall that the feature values were scaled to have mean = 0 and standard deviation = 1. So the values on the y-axis of these parallel coordinates plots show the number of standard deviations from the mean. For example, +1 means one standard deviation higher than the mean of all samples, and -1 means one standard deviation lower than the mean of all samples.

We'll create the plots with *matplotlib* using a Pandas DataFrame each row contains the cluster center coordinates and cluster label. (Matplotlib can plot Pandas DataFrames, but not Spark DataFrames.) Let's use the *pd_centers()*function in the *utils.py* library to create the Pandas DataFrame:

```
In [19]: P = utils.pd_centers(featuresUsed, centers)
```

Let's show clusters for "Dry Days", i.e., weather samples with low relative humidity:

```
In [20]: utils.parallel_plot(P[P['relative_humidity'] < -0.5], P)
```

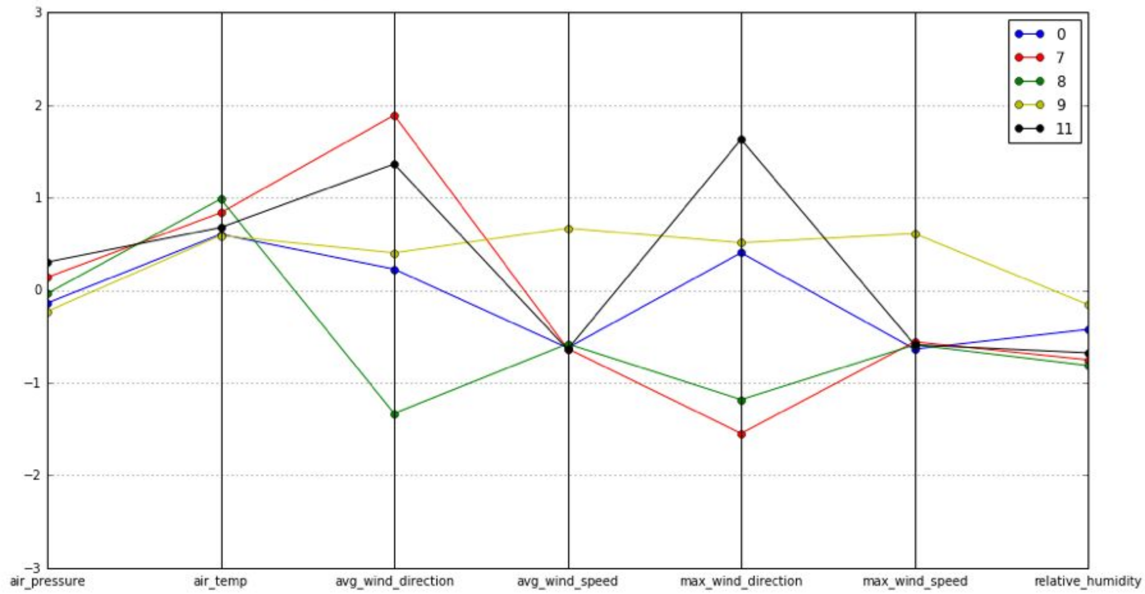


The first argument to *parallel_plot* selects the clusters whose relative humidities are centered less than 0.5 from the mean value. All clusters in this plot have *relative_humidity* < -0.5, but they differ in values for other features, meaning that there are several weather patterns that include low humidity.

Note in particular cluster 4. This cluster has samples with lower-than-average wind direction values. Recall that wind direction values are in degrees, and 0 means wind coming from the North and increasing clockwise. So samples in this cluster have wind coming from the N and NE directions, with very high wind speeds, and low relative humidity. These are characteristic weather patterns for Santa Ana conditions, which greatly increase the dangers of wildfires.

Let's show clusters for "Warm Days", i.e., weather samples with high air temperature:

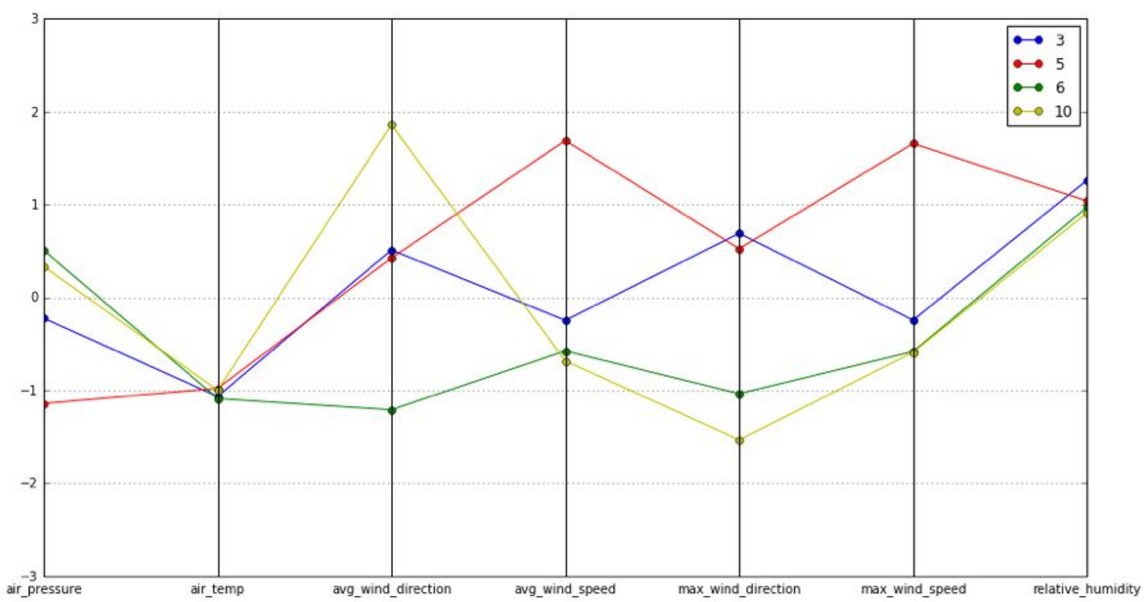
```
In [21]: utils.parallel_plot(P[P['air_temp'] > 0.5], P)
```



All clusters in this plot have *air_temp* > 0.5, but they differ in values for other features.

Let's show clusters for "Cool Days", i.e., weather samples with high relative humidity and low air temperature:

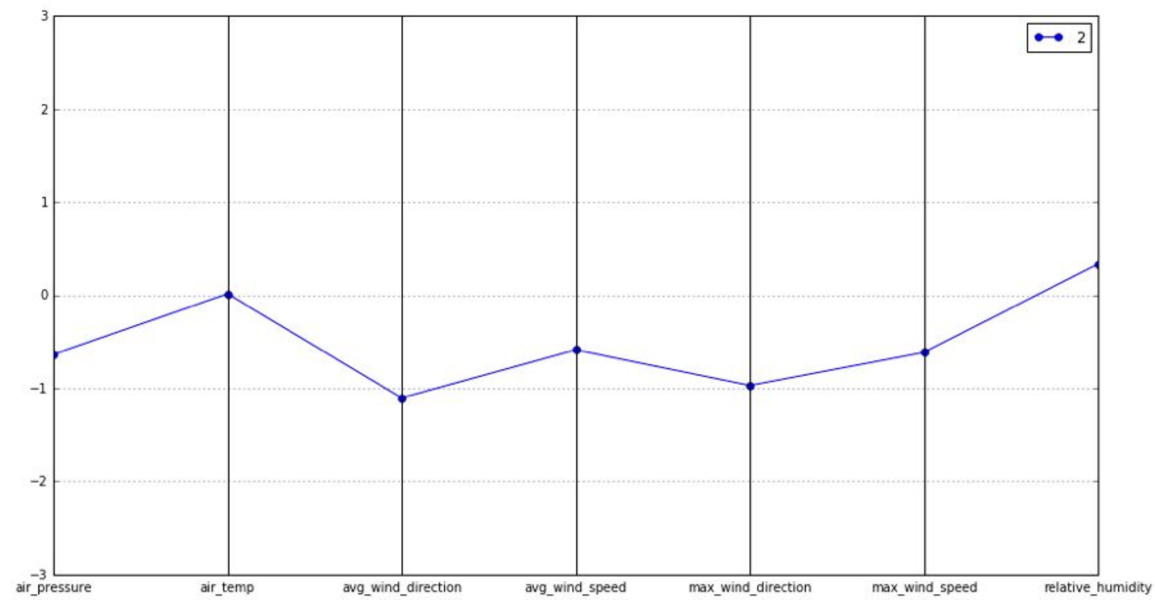
```
In [22]: utils.parallel_plot(P[(P['relative_humidity'] > 0.5) & (P['air_temp'] < 0.5)], P)
```



All clusters in this plot have *relative_humidity* > 0.5 and *air_temp* < 0.5. These clusters represent cool temperature with high humidity and possibly rainy weather patterns. For cluster 5, note that the wind speed values are high, suggesting stormy weather patterns with rain and wind.

So far, we've seen all the clusters except 2 since it did not fall into any of the other categories. Let's plot this cluster:

In [23]:
```
utils.parallel_plot(P.iloc[[2]], P)
```



Cluster 2 captures days with mild weather.