SDSC Summer Institute 2021
Scalable Machine Learning

Mai H. Nguyen & Paul Rodriguez

2021-August-04

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

Scalable Machine Learning Agenda

8:00 - 8:20 -- Machine Learning Overview
8:20 - 9:00 -- R on HPC
9:00 - 9:15 -- Break
9:15 - 10:15 -- Spark
10:15 - 10:45 -- Spark Hands-On

Spark Hands-On

Mai H. Nguyen, Ph.D.

SPARK PROGRAM STRUCTURE

- **Start Spark session**
  - spark = SparkSession.builder.config(conf=conf).getOrCreate()
- **Create distributed dataset**
  - df = spark.read.csv(“data.csv”,header=”True”)
- **Apply transformations**
  - new_df = df.filter(col(“dept”) == “Sales”)
- **Perform actions**
  - df.collect()
- **Stop Spark session**
  - spark.stop()

START SPARK SESSION

Use * to use all available cores, or integer value to specify number of cores to use

```
import pyspark
from pyspark.sql import SparkSession

conf = pyspark.SparkConf().setAll([
         ('spark.master',   'local[*]'),
         ('spark.app.name', 'PySpark Demo')])

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Configuration parameters for Spark session

Get existing Spark session or create new one

LOAD DATA

- **Loading data from local file system**

```
df = spark.read.text("file:///<path>/<file>.txt")

df = spark.read.csv("file:///<path>/<file>.csv",
                    header=True).cache()
```

- **Loading data from HDFS**

```
df = spark.read.text("hdfs:///<path>/<file>.txt") \
                     .cache()

df = spark.read.csv("hdfs:///<path>/<file>.csv",
                     header=True).cache()
```

Indicates whether column headers exist

Cache data

CHAINING

```
text_file = sc.textFile("hdfs://...")

counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://...")
```

Line continuation indicator

Chaining: Making multiple method calls on same object

RDD Wordcount

DROP ROWS WITH NULLS

- **Drop rows with null values**

```
df.dropna()
df.dropna(how='any')
df.dropna(how='all')
```

- **Check number of rows before and after dropping rows**

```
df.count()
```

FILL IN MISSING VALUES

- **Replace null values with empty string**

```
df.na.fill(' ')
```

- **Count number of rows with nulls before and after filling nulls**

```
df.count()
```

PARTITION DATA

- **Partition available data into train and test data sets**

```
train_df,test_df = df.randomSplit(0.8,0.2),seed=<seed>)
```

Percentage of samples for train dataset

Percentage of samples for test dataset

CREATE FEATURE VECTOR COLUMN

- **Create feature vector column**
  - Combines given list of columns into single vector column
  - To feed data to machine learning models

```
from pyspark.ml.feature import VectorAssembler

features  = ['air_temp','relative_humidity']
assembler = VectorAssembler(inputCols=features,
                            outputCol='featureVector')
features_df = assembler.transform(df)
features_df.show()
```

New column appended to features_df

```
air_temp|relative_humidity
62.96   | 63.9
```

```
air_temp|relative_humidity|featureVector
62.96   | 63.9            |[62.96, 63.9]
```

SCALE DATA

- **Scale input data values**
  - Standardize values to have zero mean and unit standard deviation
  - Each feature is scaled separately
  - Create scale transformer using train data, then apply to train/test data

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features_unscaled",
                        outputCol="features_scaled",
                        withStd=True, withMean=True)
scalerModel = scaler.fit(train_df)

train_df = scalerModel.transform(train_df)
test_df  = scalerModel.transform(test_df)
```

BUILD MODEL

- **Build decision tree classifier**
  - Create model
  - Use fit() to train model

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(
          featuresCol='featureVector',
          labelCol='label',
          predictionCol='prediction')

dt_model = dt.fit(<train>)
```

Can specify name of columns for features, labels, and predictions

APPLY MODEL

- **Apply trained model**
  - Use transform()

```
predictions = <model>.transform(<data>)
```

Will have <data> and new ‘prediction’ column appended at the end

Trained model

Input data

EVALUATE CLASSIFICATION MODEL

- **Evaluator for classification model**
  - Calculates F1, precision, recall, accuracy

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

mc_evaluator = MulticlassClassificationEvaluator(
   predictionCol=‘prediction’,
   labelCol=‘label’,
   metricName=‘f1’)

mc_evaluator.evaluate(<predictions>)
```

Column with predictions

Column with labels

Evaluation metric

Contains predictions and labels

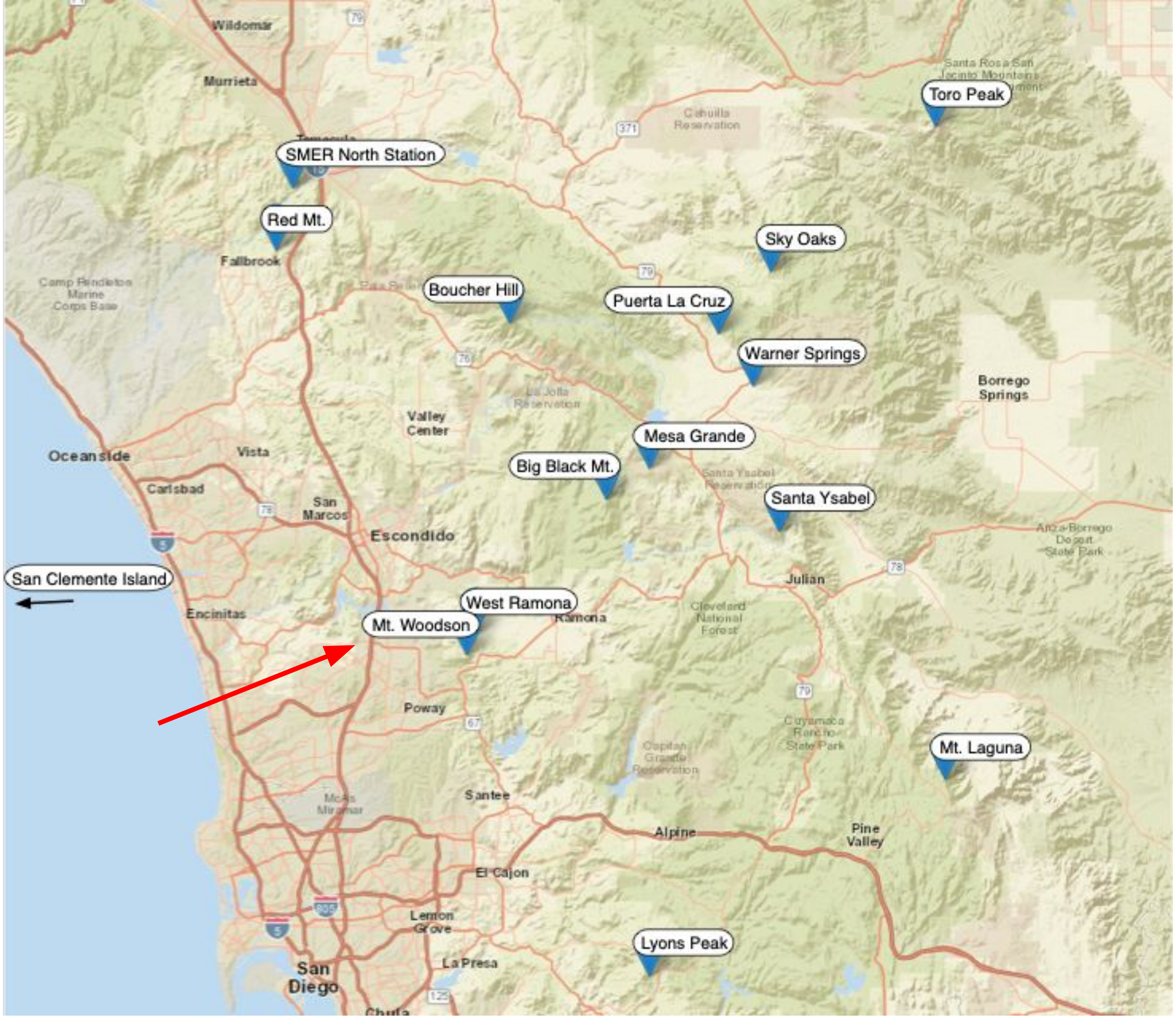
PySpark Cluster Analysis Hands-On

- **Data**
  - Weather station measurements
- **Task**
  - Perform cluster analysis to identify different weather patterns
- **Approach**
  - Spark k-means
- **Notebook**
  - pyspark-clustering.ipynb

Dataset Description

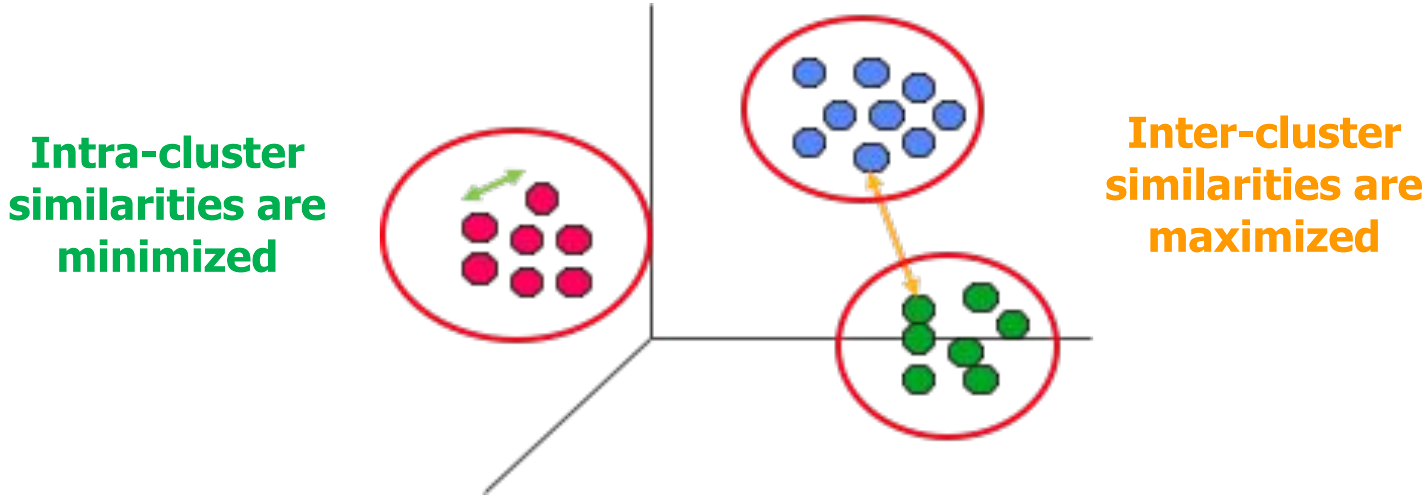
- **Measurements from weather station on Mt. Woodson, San Diego**
- **Air temperature, humidity, wind speed, wind direction, etc.**
- **Three years of data: Sep. 2011 - Sep. 2014**
  - minute_weather.csv: measurement every minute
- **Source**
  - http://hpwren.ucsd.edu

Map of HPWREN Weather Stations



Cluster Analysis

- **Cluster analysis divides data into groups**
  - Grouping is based on some similarity measure.
  - Samples within a cluster are more similar to each other than to samples in other clusters.



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

k-Means Clustering

- **Partitional**
  - Clusters are divided into non-overlapping subsets
- **Centroid-Based**
  - Cluster represented by central vector
- **Simple, classic clustering technique**
  - Data points are grouped into *k* clusters
  - Cluster defined by cluster mean
- **Algorithm**

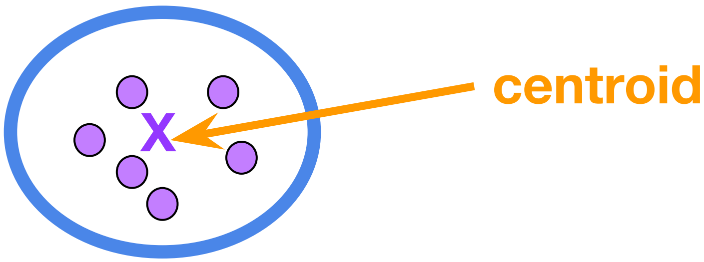
Select *k* initial *centroids* (cluster centers)

Repeat

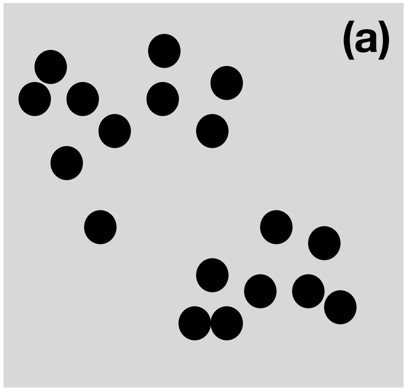
    Assign each sample to closest centroid

    Calculate mean of cluster to determine new centroid

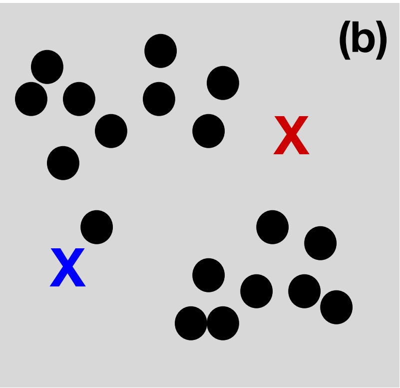
Until some stopping criterion is reached



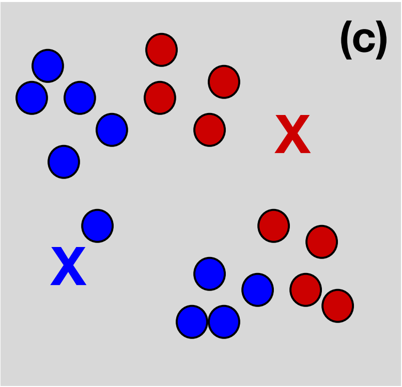
k-Means Clustering Illustration



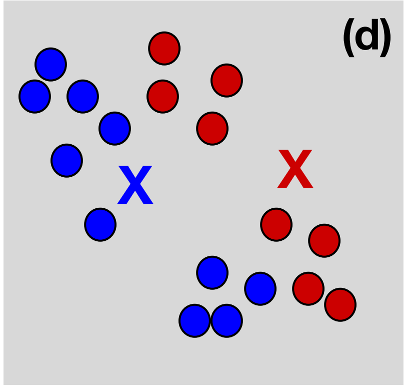
Original samples



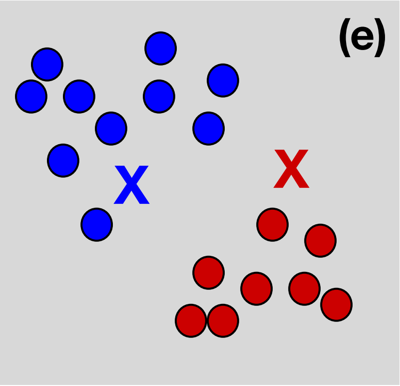
Initial Centroids



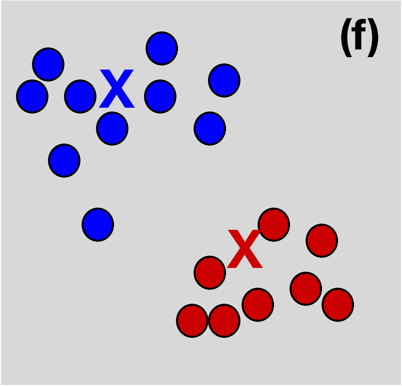
Assign Samples



Re-calculate Centroids



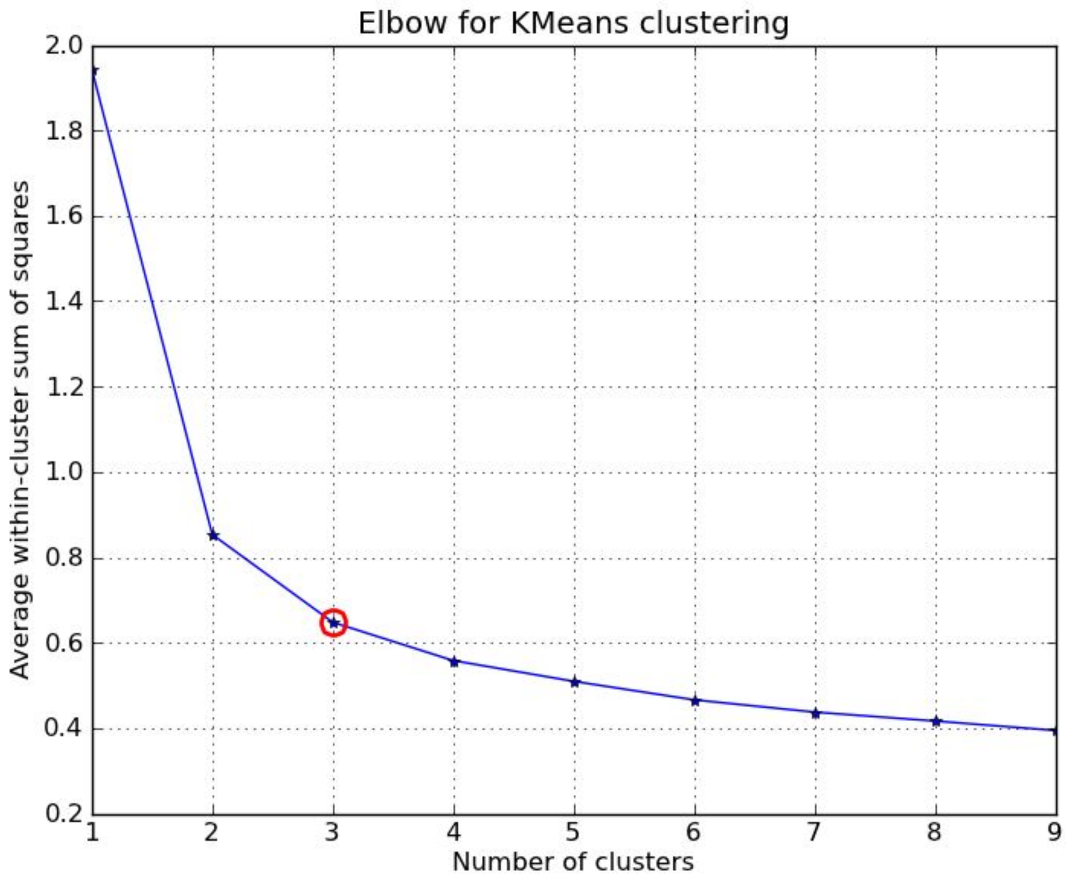
Assign Samples



Re-calculate Centroids

Choosing Number of Clusters (*k*)

- **Elbow method**
  - Plot cluster evaluation metric (e.g., WSSE) vs. different values for k
  - “Elbow” in plot suggests value(s) for *k*



http://stackoverflow.com/questions/6645895/calculating-the-percentage-of-variance-measure-for-k-means

Evaluating Clustering Results

- **Within-Cluster Sum of Squared Error (WSSE)**
- For each sample, error is distance to centroid. Then, **WSSE** is computed as:

$$WSSE = \sum_{i=1}^{K} \sum_{x \in C_i} || x - m_i ||^2$$

x: data sample in cluster $\boldsymbol{C}_i$

$\boldsymbol{m}_i$: cluster centroid (i.e., mean of cluster)

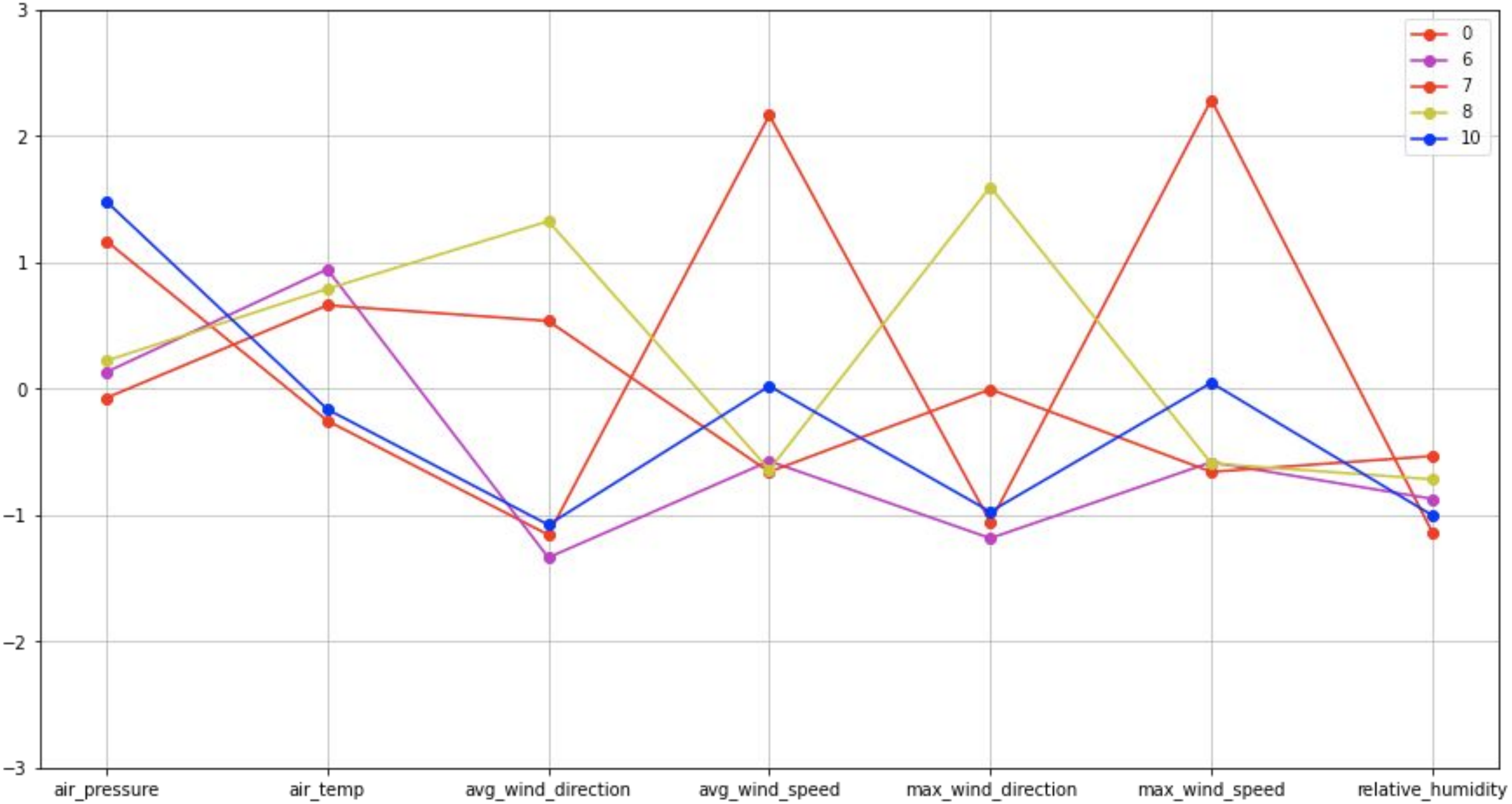
$||x - m_i||^2$: Euclidean distance between $\boldsymbol{m}_i$ and $\boldsymbol{x}$

Clustering Hands-On Overview

- **Setup**
  - Start Spark
  - Load modules
- **Load data**
  - Specify schema
  - Read in data from “minute_weather.csv”
- **Explore data**
  - Look at schema, number of rows, summary statistics
- **Prepare data**
  - Drop nulls
  - Create feature vector
- **Perform k-means cluster analysis**
  - Use elbow plot to determine k
- **Evaluate clusters**
  - Plot cluster profiles
- **Stop Spark session**

Cluster Profile: Parallel Plots

utils.parallel_plot(centersNamed[centersNamed['relative_humidity'] < -0.5], numClusters, colors=colors_used);



Setup

- **Login to Expanse**
  - Open terminal window on local machine
  - ssh login.expanse.sdsc.edu
- **Pull latest from repo**
  - cd <your-SI-repo>
  - git pull
  - URL: https://github.com/sdsc/sdsc-summer-institute-2021

Server Setup for PySpark - Portal

- **Expanse Portal**
  - https://portal.expanse.sdsc.edu
  - Use trainXXX account
  - Interactive Apps -> Jupyter
- **Parameters**
  - Account: crl155
  - Partition: shared
  - Time limit (min): 60
  - Number of cores: 2
  - Memory required per node: 8 GB
  - GPUs: 0
  - Singularity image: /cm/shared/apps/containers/singularity/ciml/2021/pyspark-latest.sif
  - Environment module: singularitypro
  - Reservation: SI2021RES
  - Type: JupyterLab

Server Setup for PySpark - Command Line

- **In terminal window**
  - cd 3.1b_Scalable_Machine_Learning
  - start-spark
    - Alias for:
    - export PATH="/cm/shared/apps/sdsc/galyleo:${PATH}";
    - galyleo launch --account crl155 --reservation SI2021RES --partition shared --cpus-per-task 2 --memory-per-node 8 --time-limit 01:00:00 --env-modules singularitypro --sif /cm/shared/apps/containers/singularity/ciml/2021/pyspark-latest.sif --bind /expanse,/scratch,/cvmfs --quiet"
  - Copy & paste URL in web browser
- **To check queue**
  - squeue -u $USER

PySpark Cluster Analysis Hands-On

- **Code**
  - **pyspark-cluster.ipynb**
    - Notebook for hands-on
    - Replace <<FILL-IN>> with code
  - **pyspark-cluster-w-outputs.ipynb**
    - Has cell outputs
  - **utils.py**
    - Has utility functions

- **Resources**
  - Apache Spark™ - Unified Analytics Engine for Big Data
  - PySpark Documentation — PySpark 3.1.2 documentation
  - Spark SQL and DataFrames - Spark 3.1.2 Documentation
  - Python for Data Science Cheat Sheet (pdf)