Overview of PySpark MLlib

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is PySpark MLlib?

- MLlib is a component of Apache Spark for machine learning
- Various tools provided by MLlib include:
  - ML Algorithms: collaborative filtering, classification, and clustering
  - Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines

Why PySpark MLlib?

- Scikit-learn is a popular Python library for data mining and machine learning
- Scikit-learn algorithms only work for small datasets on a single machine
- Spark's MLlib algorithms are designed for parallel processing on a cluster
- Supports languages such as Scala, Java, and R
- Provides a high-level API to build machine learning pipelines

PySpark MLlib Algorithms

- **Classification (Binary and Multiclass) and Regression**: Linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes, linear least squares, Lasso, ridge regression, isotonic regression
- **Collaborative filtering**: Alternating least squares (ALS)
- **Clustering**: K-means, Gaussian mixture, Bisecting K-means and Streaming K-Means

The three C's of machine learning in PySpark MLlib

- Collaborative filtering (recommender engines): Produce recommendations
- Classification: Identifying to which of a set of categories a new observation
- Clustering: Groups data based on similar characteristics

PySpark MLlib imports

- `pyspark.mllib.recommendation`

```
from pyspark.mllib.recommendation import ALS
```

- `pyspark.mllib.classification`

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
```

- `pyspark.mllib.clustering`

```
from pyspark.mllib.clustering import KMeans
```

Let's practice

BIG DATA FUNDAMENTALS WITH PYSPARK

Introduction to Collaborative filtering

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is Collaborative filtering?

- Collaborative filtering is finding users that share common interests
- Collaborative filtering is commonly used for recommender systems
- Collaborative filtering approaches
  - **User-User Collaborative filtering**: Finds users that are similar to the target user
  - **Item-Item Collaborative filtering**: Finds and recommends items that are similar to items with the target user

Rating class in pyspark.mllib.recommendation submodule

- The Rating class is a wrapper around tuple (user, product and rating)
- Useful for parsing the RDD and creating a tuple of user, product and rating

```
from pyspark.mllib.recommendation import Rating
r = Rating(user = 1, product = 2, rating = 5.0)
(r[0], r[1], r[2])
```

```
(1, 2, 5.0)
```

Splitting the data using randomSplit()

- Splitting data into training and testing sets is important for evaluating predictive modeling
- Typically a large portion of data is assigned to training compared to testing data
- PySpark's `randomSplit()` method randomly splits with the provided weights and returns multiple RDDs

```
data = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
training, test=data.randomSplit([0.6, 0.4])
training.collect()
test.collect()
```

```
[1, 2, 5, 6, 9, 10]
[3, 4, 7, 8]
```

Alternating Least Squares (ALS)

- Alternating Least Squares (ALS) algorithm in `spark.mllib` provides collaborative filtering
- `ALS.train(ratings, rank, iterations)`

```
r1 = Rating(1, 1, 1.0)
r2 = Rating(1, 2, 2.0)
r3 = Rating(2, 1, 2.0)
ratings = sc.parallelize([r1, r2, r3])
ratings.collect()
```

```
[Rating(user=1, product=1, rating=1.0),
 Rating(user=1, product=2, rating=2.0),
 Rating(user=2, product=1, rating=2.0)]
```

```
model = ALS.train(ratings, rank=10, iterations=10)
```

predictAll() – Returns RDD of Rating Objects

- The predictAll() method returns a list of predicted ratings for input user and product pair
- The method takes in an RDD without ratings to generate the ratings

```
unrated_RDD = sc.parallelize([(1, 2), (1, 1)])
```

```
predictions = model.predictAll(unrated_RDD)
predictions.collect()
```

```
[Rating(user=1, product=1, rating=1.0000278574351853),
 Rating(user=1, product=2, rating=1.989035570377812 2)]
```

Model evaluation using MSE

- The MSE is the average value of the square of `(actual rating - predicted rating)`

```
rates = ratings.map(lambda x: ((x[0], x[1]), x[2]))
rates.collect()
```

```
[((1, 1), 1.0), ((1, 2), 2.0), ((2, 1), 2.0)]
```

```
preds = predictions.map(lambda x: ((x[0], x[1]), x[2]))
preds.collect()
```

```
[((1, 1), 1.00002785743518 53), ((1, 2), 1.9890355703778122)]
```

```
rates_preds = rates.join(preds)
rates_preds.collect()
```

```
[((1, 2), (2.0, 1.9890355703778122)), ((1, 1), (1.0, 1.00002785743518 53))]
```

```
MSE = rates_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
```

Let's practice!

BIG DATA FUNDAMENTALS WITH PYSPARK

Classification

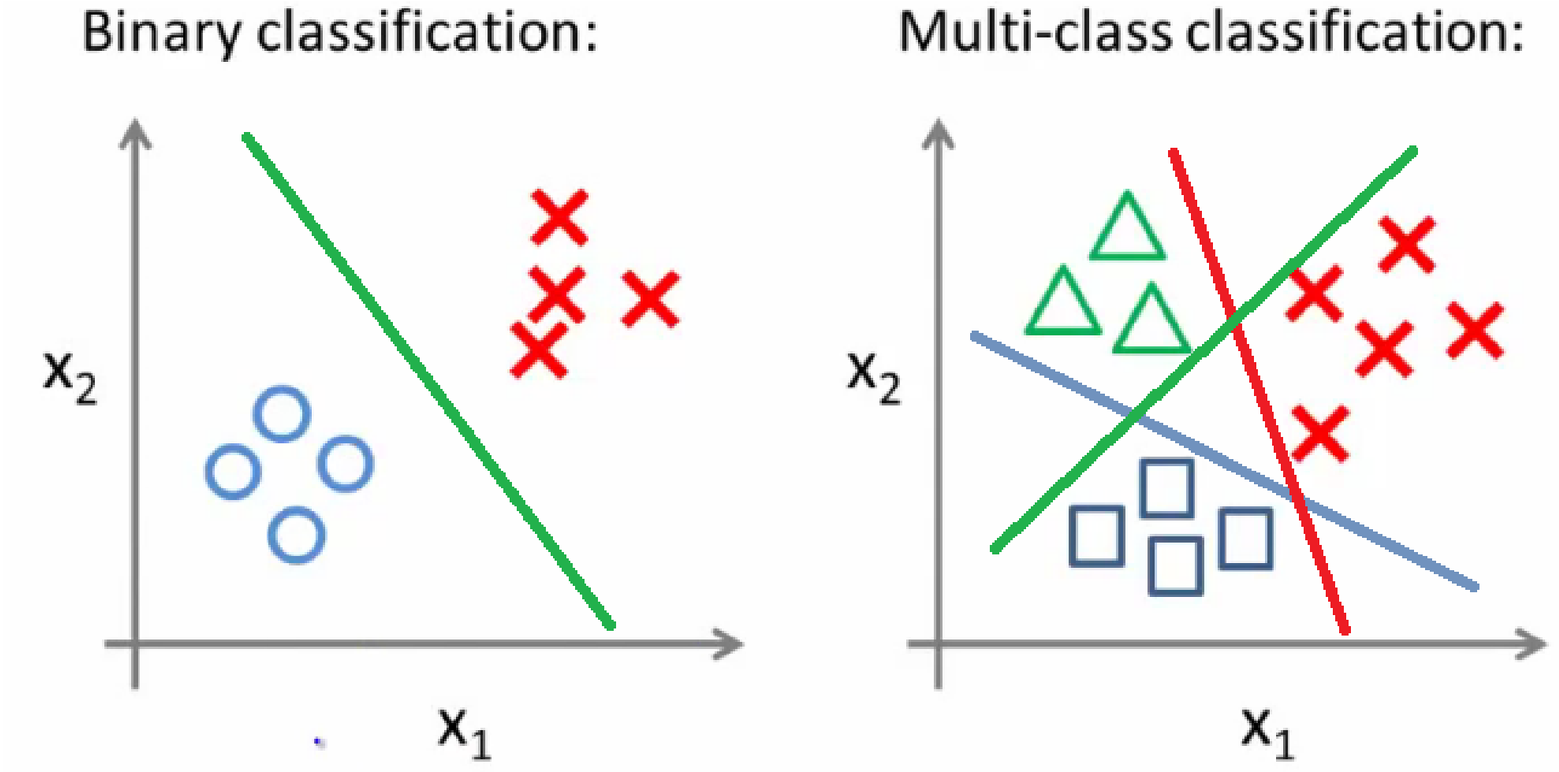
BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

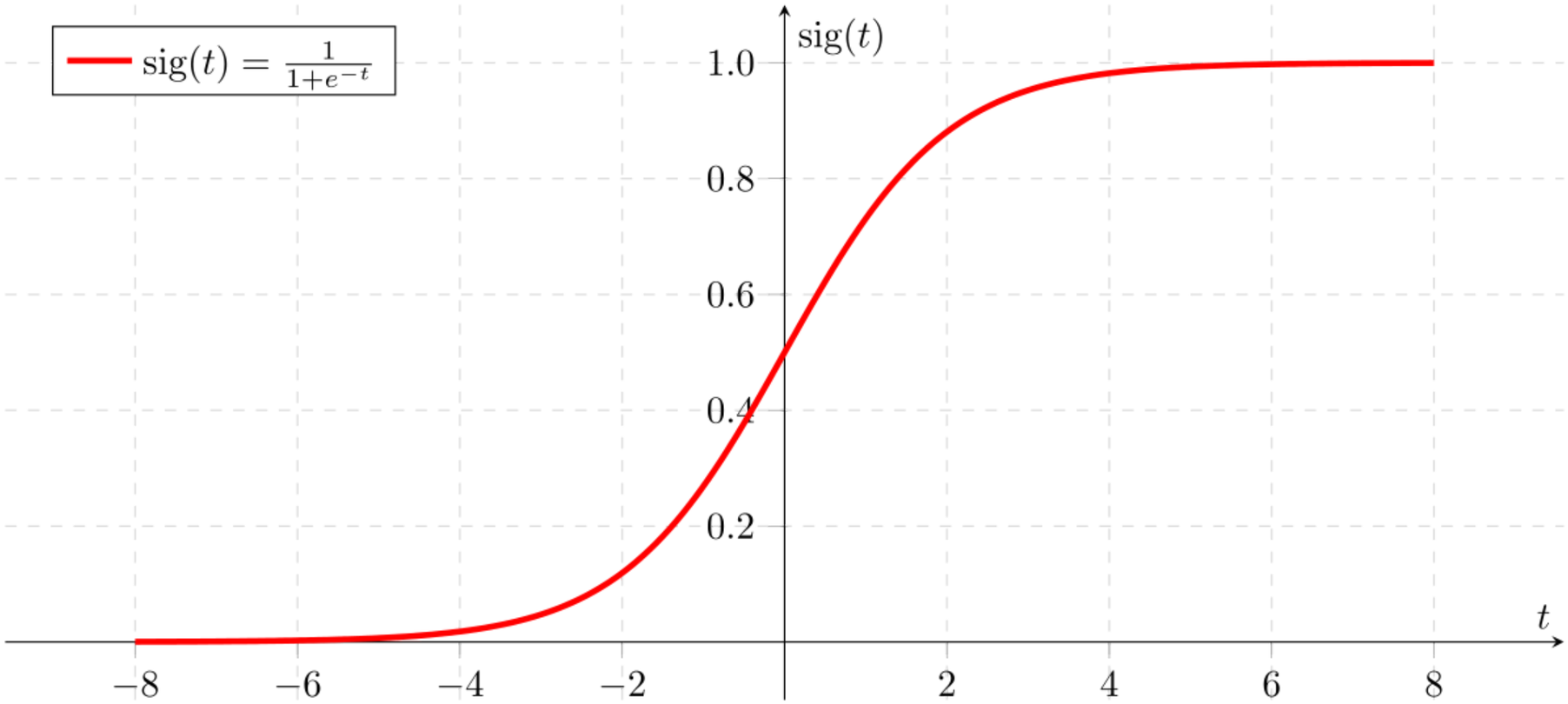
Classification using PySpark MLlib

- Classification is a supervised machine learning algorithm for sorting the input data into different categories



Introduction to Logistic Regression

- Logistic Regression predicts a binary response based on some variables



Working with Vectors

- PySpark MLlib contains specific data types Vectors and LabelledPoint
- Two types of Vectors
  - Dense Vector: store all their entries in an array of floating point numbers
  - Sparse Vector: store only the nonzero values and their indices

```
denseVec = Vectors.dense([1.0, 2.0, 3.0])
```

```
DenseVector([1.0, 2.0, 3.0])
```

```
sparseVec = Vectors.sparse(4, {1: 1.0, 3: 5.5})
```

```
SparseVector(4, {1: 1.0, 3: 5.5})
```

LabelledPoint() in PySpark MLlib

- A LabeledPoint is a wrapper for input features and predicted value
- For binary classification of Logistic Regression, a label is either 0 (negative) or 1 (positive)

```
positive = LabeledPoint(1.0, [1.0, 0.0, 3.0])
negative = LabeledPoint(0.0, [2.0, 1.0, 1.0])
print(positive)
print(negative)
```

```
LabeledPoint(1.0, [1.0,0.0,3.0])
LabeledPoint(0.0, [2.0,1.0,1.0])
```

HashingTF() in PySpark MLlib

- `HashingTF()` algorithm is used to map feature value to indices in the feature vector

```
from pyspark.mllib.feature import HashingTF
sentence = "hello hello world"
words = sentence.split()
tf = HashingTF(10000)
tf.transform(words)
```

```
SparseVector(10000, {3065: 1.0, 6861: 2.0})
```

Logistic Regression using LogisticRegressionWithLBFGS

- Logistic Regression using Pyspark MLlib is achieved using LogisticRegressionWithLBFGS class

```
data = [
        LabeledPoint(0.0, [0.0, 1.0]),
        LabeledPoint(1.0, [1.0, 0.0]),
]
RDD = sc.parallelize(data)
```

```
lrm = LogisticRegressionWithLBFGS.train(RDD)
```

```
lrm.predict([1.0, 0.0])
lrm.predict([0.0, 1.0])
```

```
1
0
```

Final Slide

BIG DATA FUNDAMENTALS WITH PYSPARK

Introduction to Clustering

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

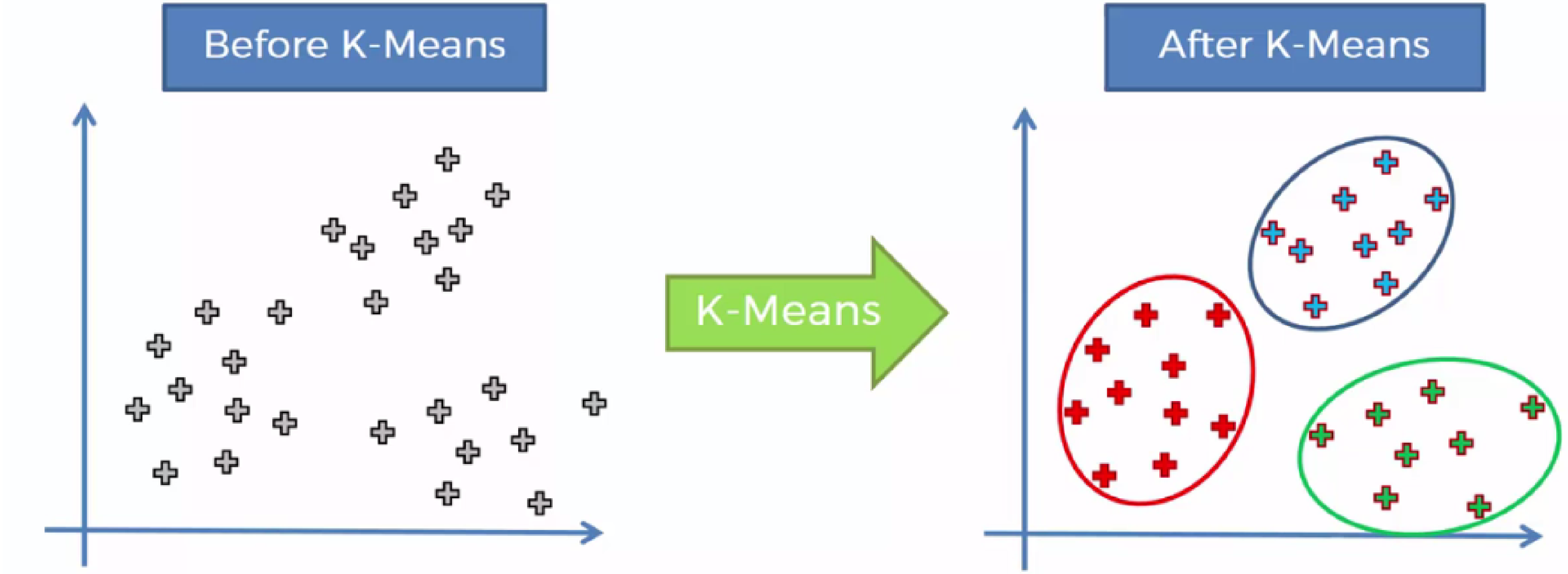
datacamp

What is Clustering?

- Clustering is the unsupervised learning task to organize a collection of data into groups
- PySpark MLlib library currently supports the following clustering models
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Bisecting k-means
  - Streaming k-means

K-means Clustering

- K-means is the most popular clustering method



K-means with Spark MLLib

```
RDD = sc.textFile("WineData.csv"). \
        map(lambda x: x.split(",")).\
        map(lambda x: [float(x[0]), float(x[1])])
RDD.take(5)
```

```
[[14.23, 2.43], [13.2, 2.14], [13.16, 2.67], [14.37, 2.5], [13.24, 2.87]]
```

Train a K-means clustering model

- Training K-means model is done using `KMeans.train()` method

```
from pyspark.mllib.clustering import KMeans
model = KMeans.train(RDD, k = 2, maxIterations = 10)
model.clusterCenters
```

```
[array([12.25573171,  2.28939024]), array([13.636875  ,  2.43239583])]
```

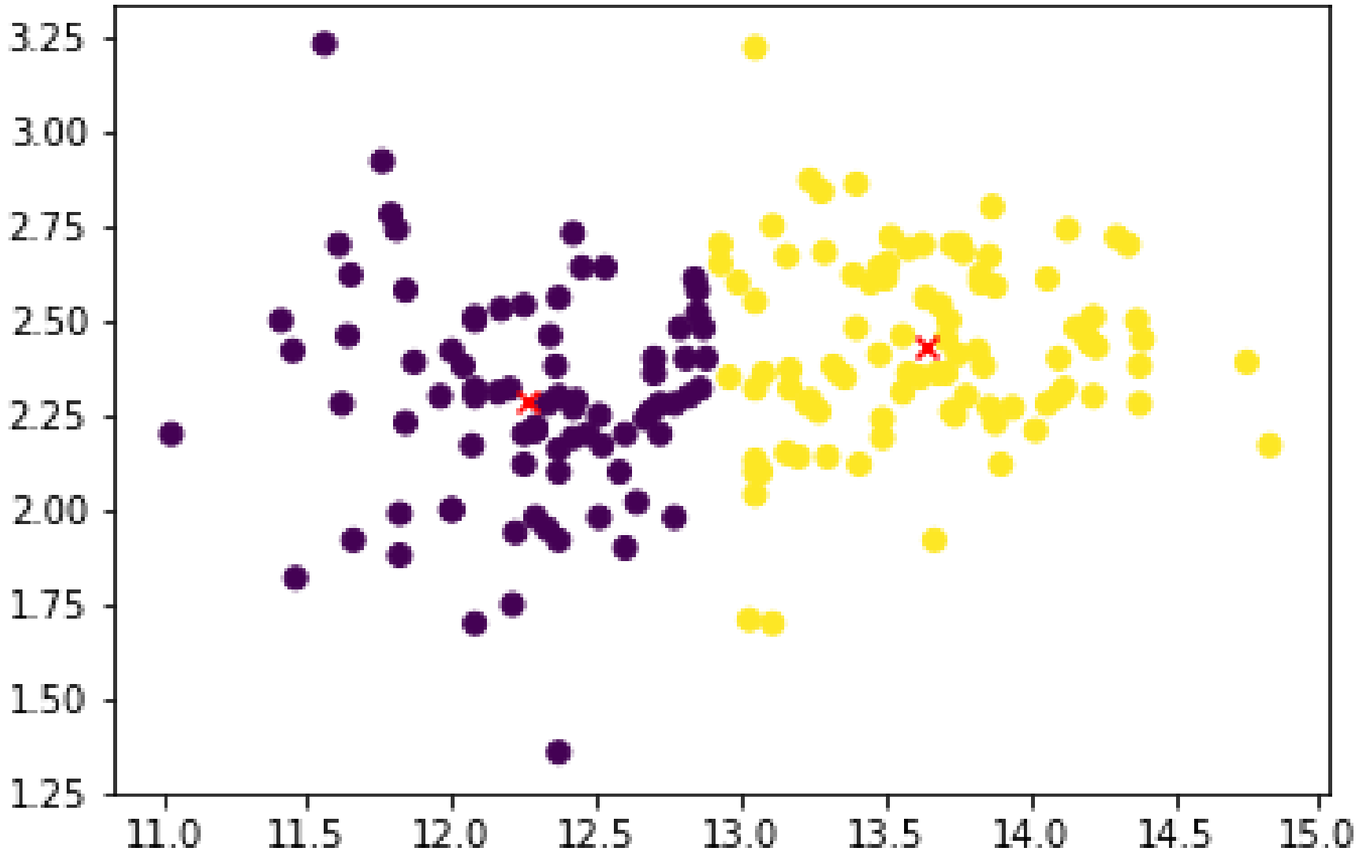
Evaluating the K-means Model

```
from math import sqrt
def error(point):
    center = model.centers[model.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

```
WSSSE = RDD.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))
```

```
Within Set Sum of Squared Error = 77.96236420499056
```

Visualizing K-means clusters



Visualizing clusters

```
wine_data_df = spark.createDataFrame(RDD, schema=["col1", "col2"])
wine_data_df_pandas = wine_data_df.toPandas()
```

```
cluster_centers_pandas = pd.DataFrame(model.clusterCenters, columns=["col1", "col2"])
cluster_centers_pandas.head()
```

```
plt.scatter(wine_data_df_pandas["col1"], wine_data_df_pandas["col2"]);
plt.scatter(cluster_centers_pandas["col1"], cluster_centers_pandas["col2"], color="red", marker="x"
```

Clustering practice

BIG DATA FUNDAMENTALS WITH PYSPARK

Congratulations!

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

Fundamentals of BigData and Apache Spark

- **Chapter 1:** Fundamentals of BigData and introduction to Spark as a distributed computing framework
  - Main components: Spark Core and Spark built-in libraries - Spark SQL, Spark MLlib, Graphx, and Spark Streaming
  - PySpark: Apache Spark's Python API to execute Spark jobs
  - PySpark shell: For developing the interactive applications in python
  - Spark modes: Local and cluster mode

Spark components

- **Chapter 2**: Introduction to RDDs, different features of RDDs, methods of creating RDDs and RDD operations (Transformations and Actions)
- **Chapter 3**: Introduction to Spark SQL, DataFrame abstraction, creating DataFrames, DataFrame operations and visualizing Big Data through DataFrames
- **Chapter 4**: Introduction to Spark MLlib, the three C's of Machine Learning (Collaborative filtering, Classification and Clustering)

Where to go next?

BIG DATA FUNDAMENTALS WITH PYSPARK