Grid Dynamics

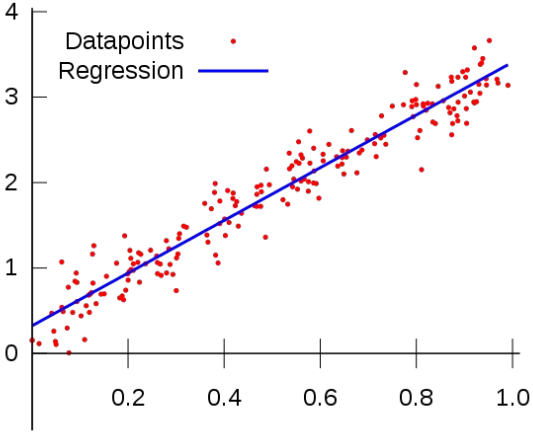
Lecture 2: Regression, Clustering, Collaborative Filtering and Pattern Mining

Lecture 2: Regression, Clustering, Collaborative Filtering and Pattern Mining

- Introduction
- About Spark
- Spark MLlib
- Classification
  - Naive Bayes Classifier
  - Logistic Regression
  - Decision Tree Classifier
  - Random Forest Classifier
  - GBT (Gradient-Boosted Trees) Classifier
  - Multilayer Perceptron Classifier
- Regression
  - Linear Regression
- Clustering
  - K-Means
- Collaborative filtering
  - ALS (alternating least squares)
- Frequent Pattern Mining
  - FPG (Frequent Pattern-Growth)
- Custom Transformer
- Custom Estimator
- Demo
- Summary
- QA

Regression

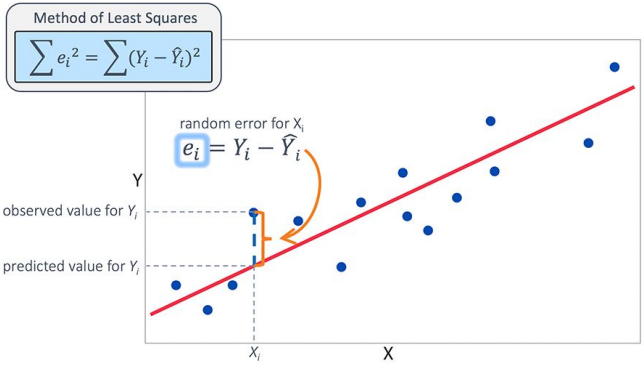
Regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome' or 'response' variable) and one or more independent variables (often called 'predictors', 'covariates', 'explanatory variables' or 'features'). The most common form of regression analysis is linear regression, in which one finds the line (or a more complex linear combination) that most closely fits the data according to a specific mathematical criterion.



Linear Regression

Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Linear regression is an attractive model because the representation is so simple. The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric. The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.



Spark ML LinearRegression

```scala
import org.apache.spark.ml.regression.LinearRegression

// Load training data
val training = spark.read.format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt")

val lr = new LinearRegression()
  .setMaxIter(10)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)

// Fit the model
val lrModel = lr.fit(training)

// Print the coefficients and intercept for linear regression
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// Summarize the model over the training set and print out some metrics
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")
trainingSummary.residuals.show()

println(s"MSE: ${trainingSummary.meanSquaredError}")
println(s"MAE: ${trainingSummary.meanAbsoluteError}")
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")
```

Regression Metrics (MAE, MSE, MRSE, MRSLE)

Mean absolute error (or mean absolute deviation) is another metric which finds the average absolute distance between the predicted and target values

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$$

Regression Metrics (MAE, MSE, MRSE, MRSLE)

MSE is calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives you an absolute number on how much your predicted results deviate from the actual number

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y_i}\right)^2$$

Regression Metrics (MAE, MSE, RMSE, RMSLE)

It is the Root Mean Squared Error of the log-transformed predicted and log-transformed actual values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y_i}\right)^2}$$

Regression Metrics (MAE, MSE, RMSE, RMSLE)

RMSLE adds 1 to both actual and predicted values before taking the natural logarithm to avoid taking the natural log of possible 0 (zero) values.

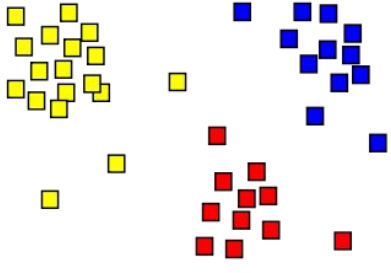
$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\widehat{y_i} + 1)^2}$$

$$chocolate\ bar: \quad y = 100,\ \widehat{y_i} = 150,\ RMSE = 50,\ RMSLE = 0.4$$

$$coffee\ machine: \quad y = 10000,\ \widehat{y_i} = 10050,\ RMSE = 0.0004$$

Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data analysis, and a common technique for statistical data analysis, used in many fields, including pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning.

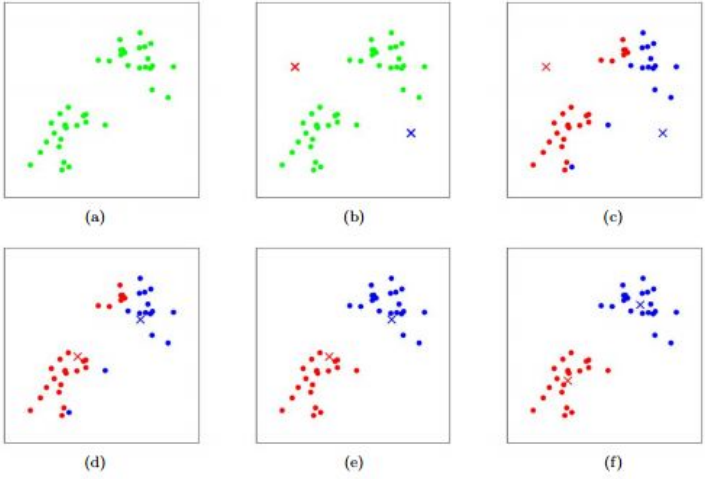


K-Means

The action of the algorithm is such that it seeks to minimize the total quadratic deviation of cluster points from the centers of these clusters:

$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2$$

where k is the number of clusters, S_i are the resulting clusters, i = 1 , 2 , ... , k, and μ_i are the centers of mass of all vectors x from the cluster S_i.

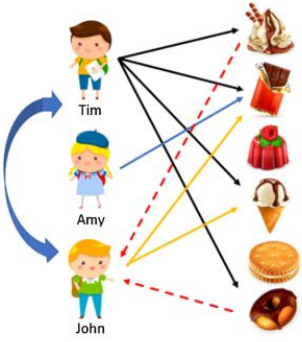


Spark ML KMeans

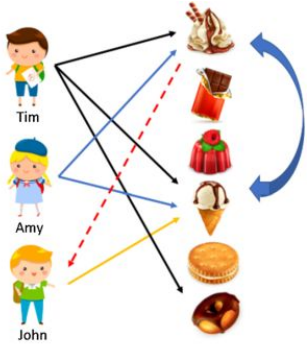
```scala
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator

// Loads data.
val dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

// Trains a k-means model.
val kmeans = new KMeans().setK(2).setSeed(1L)
val model = kmeans.fit(dataset)

// Make predictions
val predictions = model.transform(dataset)

// Evaluate clustering by computing Silhouette score
val evaluator = new ClusteringEvaluator()

val silhouette = evaluator.evaluate(predictions)
println(s"Silhouette with squared euclidean distance = $silhouette")

// Shows the result.
println("Cluster Centers: ")
model.clusterCenters.foreach(println)
```

Collaborative filtering

Collaborative filtering (CF) is a technique used by recommender systems. Collaborative filtering has two senses, a narrow one and a more general one. In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. In the more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.



(a) User-based filtering

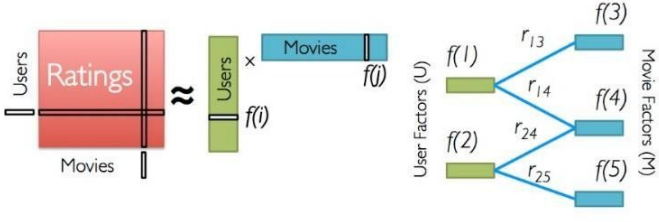


(b) Item-based filtering

ALS (alternating least squares)

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.
Some high-level ideas behind ALS are:

- ALS uses L2 regularization
- ALS minimizes two loss functions alternatively; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix
- ALS runs its gradient descent in parallel across multiple partitions of the underlying training data from a cluster of machines



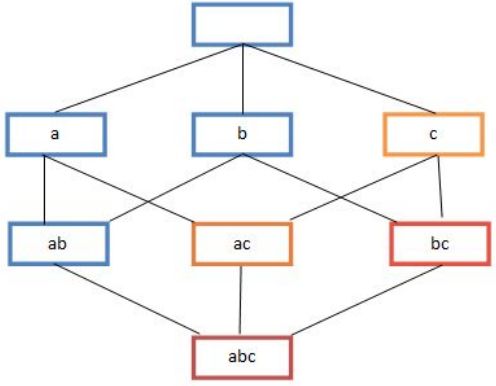
Spark ML ALS

```
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS

case class Rating(userId: Int, movieId: Int, rating: Float,
timestamp: Long)
def parseRating(str: String): Rating = {
  val fields = str.split("::")
  assert(fields.size == 4)
  Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat,
fields(3).toLong)
}

val ratings =
spark.read.textFile("data/mllib/als/sample_movielens_ratings.txt
")
  .map(parseRating)
  .toDF()
val Array(training, test) = ratings.randomSplit(Array(0.8, 0.2))

// Build the recommendation model using ALS on the training data
val als = new ALS()
  .setMaxIter(5)
  .setRegParam(0.01)
  .setUserCol("userId")
  .setItemCol("movieId")
  .setRatingCol("rating")
val model = als.fit(training)
```

```
// Evaluate the model by computing the RMSE on the test data
// Note we set cold start strategy to 'drop' to ensure we
don't get NaN evaluation metrics
model.setColdStartStrategy("drop")
val predictions = model.transform(test)

val evaluator = new RegressionEvaluator()
  .setMetricName("rmse")
  .setLabelCol("rating")
  .setPredictionCol("prediction")
val rmse = evaluator.evaluate(predictions)
println(s"Root-mean-square error = $rmse")

// Generate top 10 movie recommendations for each user
val userRecs = model.recommendForAllUsers(10)
// Generate top 10 user recommendations for each movie
val movieRecs = model.recommendForAllItems(10)

// Generate top 10 movie recommendations for a specified set
of users
val users =
ratings.select(als.getUserCol).distinct().limit(3)
val userSubsetRecs = model.recommendForUserSubset(users,
10)
// Generate top 10 user recommendations for a specified set
of movies
val movies =
ratings.select(als.getItemCol).distinct().limit(3)
  val movieSubSetRecs =
  model.recommendForItemSubset(movies, 10)
```

Frequent Pattern Mining

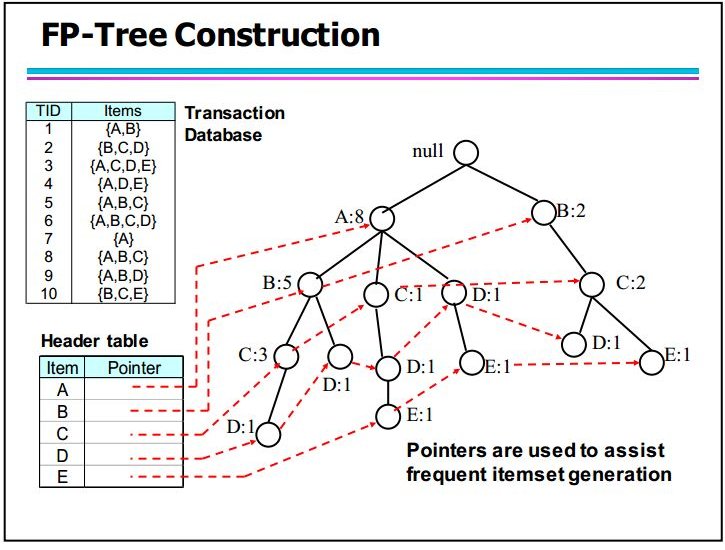
Frequent pattern discovery (or FP discovery, FP mining, or Frequent itemset mining) is part of knowledge discovery in databases, Massive Online Analysis, and data mining; it describes the task of finding the most frequent and relevant patterns in large datasets. The concept was first introduced for mining transaction databases. Frequent patterns are defined as subsets (itemsets, subsequences, or substructures) that appear in a data set with frequency no less than a user-specified or auto-determined threshold.



FPG (Frequent Pattern-Growth)

The algorithm is based on the preprocessing of the transaction database, during which it is transformed into a compact tree structure called the Frequent-Pattern Tree – a tree of popular subject sets (hence the name of the algorithm). In the future, for brevity, we will call this structure an FP-tree. The main advantages of this method include:

- Compressing the transaction database into a compact structure that provides the most efficient and complete extraction of frequent subject sets;
- When building an FP tree, the "divide and conquer" technology is used, which allows you to decompose one complex task into many simpler ones;
- This avoids the costly candidate generation procedure that is typical for the Apriori algorithm.



Spark ML FPGrowth

```
import org.apache.spark.ml.fpm.FPGrowth

val dataset = spark.createDataset(Seq(
  "1 2 5",
  "1 2 3 5",
  "1 2")
).map(t => t.split(" ")).toDF("items")

val fpgrowth = new FPGrowth().setItemsCol("items").setMinSupport(0.5).setMinConfidence(0.6)
val model = fpgrowth.fit(dataset)

// Display frequent itemsets.
model.freqItemsets.show()

// Display generated association rules.
model.associationRules.show()

// transform examines the input items against all the association rules and summarize the
// consequents as prediction
  model.transform(dataset).show()
```

Task 2. Clustering model. The definition of a private university or not.

The task is to train a machine learning clustering model that will predict whether the university private or not private on the presented data.

Conditions:

- Model should be trained only using Spark ML clustering model.
- You need to train the model on a training data set `train.csv`.
- Calculate metrics based on the file `is_private.csv`.