PART 1. BATCH COMPUTING MODELS FOR BIG DATA ANALYTICS
5. ADVANCED DATA ANALYTICS WITH APACHE SPARK

Today’s topics
- Advanced Data Analytics with Apache Spark
  - Evaluation methodologies
  - Item-to-item collaborative filtering
  - Linear regression models

What is a “good” recommendation?
- “a popular artist”?
- “artists the user has listened to”?
- “artists the user will listen to”?

Preparing data for evaluation
- To perform a meaningful evaluation, some of the artist play data can be set aside
  - Hidden from the ALS model building process
- The held-out data can be used as a collection of good recommendations for each user
  - Compute the recommender’s score

FAQs
- Term project proposal
  - New deadline: Tomorrow
- PA1 demo
  - No class on Thursday (10/12)
  - There will be an extra class added in the week of the term project final presentations
- PA2 has been posted (11/6)
AUC metric

- Rank 1.0 is perfect, 0.0 is the worst
- Receiver Operating Characteristic (ROC)
- Based on the rank used to decide final recommendations
- Area Under the Curve (AUC) of ROC may be used as the probability that a randomly chosen good recommendation ranks above a randomly chosen bad recommendation.

Computing AUC

- 90% of the data is used for training and the remaining 10% for validation

```
import org.apache.spark.rdd._

def areaUnderCurve(positiveData: RDD[Rating], allItemIDs: Broadcast[Array[Int]], predictFunction: (RDD[(Int, Int)] => RDD[Rating])) = {
  ...
}

val allData = buildRatings(rawUserArtistData, bArtistAlias)
val Array(trainData, cvData) = allData.randomSplit(Array(0.9, 0.1))

val allItemIDs = allData.map(_.product).distinct().collect()
val bAllItemIDs = sc.broadcast(allItemIDs)

val model = ALS.trainImplicit(trainData, 10, 5, 0.01, 1.0)
val auc = areaUnderCurve(cvData, bAllItemIDs, model.predict)
```

\[ E = \frac{1}{K} \sum_{i=1}^{K} E_i \]

- k-fold cross-validation is similar to random subsampling
- The advantage of k-Fold Cross validation
  - All the examples in the dataset are eventually used for both training and testing
  - The true error is estimated as the average error rate
ML Tuning: model selection and hyperparameter tuning

```scala
val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new BinaryClassificationEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(2) // Use 3+ in practice

// If you want to tune your model hyperparameters, run cross-validation, and choose the best set of parameters.
val cvModel = cv.fit(training)

// Full code is available at: https://spark.apache.org/docs/2.1.0/ml-tuning.html
```

Hyperparameter selection

- MatrixFactorizationModel
- ALS.trainImplicit()
- rank = 10
  - The number of latent factors in the model
- iterations = 5
  - The number of iterations that the factorization runs
- lambda = 0.1
  - A standard overfitting parameter
-Higher value guards against overfitting
- Values that are too high will decrease the factorization’s accuracy.
- alpha = 1.0
  - Controls the relative weight of observed versus unobserved user-product interactions in the factorization

5. Advanced Data Analytics with Apache Spark

CASE STUDY: Recommendation Systems

Amazon.com: Item-to-item collaborative filtering

- Amazon.com uses recommendations as a targeted marketing tool
  - Email campaigns
  - Most of their web pages

This material is built based on,

- Greg Linden, Brent Smith, and Jeremy York, "Amazon.com Recommendations, Item-to-Item Collaborative Filtering" IEEE Internet Computing, 2003

- It does NOT match the user to similar customers
  - Item-to-item collaborative filtering
    - Matches each of the user’s purchased and rated items to similar items
    - Combines those similar items into a recommendation list
Determining the most-similar match

- The algorithm builds a similar-items table
  - By finding items that customers tend to purchase together
- How about building a product-to-product matrix by iterating through all item pairs and computing a similarity metric for each pair?
- Many product pairs have no common customer
  - If you already bought a TV today, will you buy another TV again today?

Computing similarity

- Using cosine measure
  - Each vector corresponds to an item rather than a customer
  - M dimensions correspond to customers who have purchased that item

Creating a similar-item table

- Similar-items table is extremely computing intensive
  - Offline computation
    - $O(N^2M)$ in the worst case
    - Where $N$ is the number of items and $M$ is the number of users
    - Average case is closer to $O(NM)$
  - Most customers have very few purchases
  - Sampling customers who purchase best-selling titles reduces runtime even more
  - With little reduction in quality

Scalability

- Amazon.com has around 110 million active customers (244 million total customers) and several million catalog items
- Traditional collaborative filtering does little or no offline computation
- Online computation scales with the number of customers and catalog items.

http://www.fool.com/investing/general/2014/05/24/how-many-customers-does-amazon-have.aspx

Key scalability strategy for amazon recommendations

- Creating the expensive similar-items table offline
- Online component
  - Looking up similar items for the user’s purchases and ratings
  - Scales independently of the catalog size or the total number of customers
  - It is dependent only on how many titles the user has purchased or rated
Recommendation quality

- The algorithm recommends highly correlated similar items
- Recommendation quality is excellent
- Algorithm performs well with limited user data

5. Advanced Data Analytics with Apache Spark
Case study: Linear Models

Spark’s linear models

- Classification
  - Linear Support Vector Machines (SVMs)
  - Logistic regression

- Regression
  - Linear least squares, Lasso, and ridge regression

Implementation

- SVMWithSGD
- LogisticRegressionWithBFGS
- LogisticRegressionWithSGD
- LinearRegressionWithSGD
- RidgeRegressionWithSGD
- LassoWithSGD

Large scale data analysis using Spark
Case study: Linear Models

Linear Regression Model to a Large Dataset

The linear regression model

- The structure is exactly the same as for the linear discriminant function
  \[ f(x) = w_0 + w_1x_1 + w_2x_2 + \ldots \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math score</td>
<td>85</td>
<td>65</td>
<td>95</td>
<td>70</td>
<td>80</td>
<td>67</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>Physics score</td>
<td>89</td>
<td>92</td>
<td>70</td>
<td>95</td>
<td>85</td>
<td>85</td>
<td>50</td>
<td>70</td>
</tr>
</tbody>
</table>

- How big is the error of the fitted model?
  - We would like to minimize this error

- The model that fits the data best
  - The model with the minimum sum of errors on the training data
    - e.g. The sum of means of the squares of the errors
      - Least squares regression
Squared error
- Squared error
  - Strongly penalizes very large errors
  - Drawback
    - Is very sensitive to the data
    - Erroneous or outlying data points can severely skew the resultant linear function
- We should choose the objective function to optimize

Root Mean Squared Error
- Measures the differences between values predicted by model estimator and the values actually observed
  \[ \text{RMSD} = \sqrt{\frac{1}{n} \sum (h_\theta(x) - y)^2} \]

Linear Regression
- Calculating a linear regression
  - Using the least square criterion

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math score</td>
<td>85</td>
<td>85</td>
<td>62</td>
<td>89</td>
<td>65</td>
<td>80</td>
<td>67</td>
</tr>
<tr>
<td>Physics score</td>
<td>89</td>
<td>92</td>
<td>70</td>
<td>95</td>
<td>95</td>
<td>80</td>
<td>73</td>
</tr>
</tbody>
</table>

\[ h_\theta(x) = \theta_0 + \theta_1 x_1 \]

where, the \( \theta \)'s are the parameters (Math scores, or slopes)

Minimization problem
- We have a function \( J(\theta_0, \theta_1) \)
- We want to find \( \min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \)
  - Goal: Find parameters to minimize the cost (output of the objective function)

- Outline of our approach:
  - Start with some \( \theta_0, \theta_1 \)
  - Keep changing \( \theta_0, \theta_1 \) to reduce \( J(\theta_0, \theta_1) \) until we end up at a minimum

Objective function (Cost function)
- For a given training set, how do we pick, or learn, the parameter \( \theta \)?
- Make \( h(x) \) close to \( y \)
  - Make your prediction close to the real observation
- We define the objective (cost) function
  \[ J(\theta) = \frac{1}{2} \sum (h_\theta(x) - y)^2 \]
Concept of Gradient descent algorithm (1/2)

Concept of Gradient descent algorithm (2/2)

Stochastic Gradient descent algorithm

Simultaneous update

Decreasing \( \theta_j \)

Increasing \( \theta_j \)
Learning rates

- If $\alpha$ is too small, gradient descent can be slow
- If $\alpha$ is too large, gradient descent can overshoot the minimum
  - It may fail to converge
  - Or even diverge

Learning rates: starting at the optimal value?

- Current value of $\theta_i$ has converged already
- No more iteration required

Fixed learning rate $\alpha$

- Gradient descent can converge to a local minimum, even with a fixed learning rate
- As we approach a local minimum, gradient descent will automatically take smaller steps
  - No need to decrease $\alpha$ over time

\[
\theta_i := \theta_i - \alpha \frac{d}{d\theta_i} J(\theta_i)
\]