PART 1. BATCH COMPUTING MODELS FOR BIG DATA ANALYTICS

2. LARGE SCALE DATA ANALYSIS USING SPARK WITH CASE STUDY

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- PA1 demo is going on
- Term Project
  - 10/7 5:00PM
  - Description available on the course web
  - Presentation schedule
  - Google computing cluster credit is available

Proposal

- Title of your project
- Problem formulation
- Your strategy to solve the problem
- Functions targeted by your software
- Plan for testing
- Evaluation method
- Project timeline (weekly plan)
- Bibliography
- Submission (1,200-1,800 words)

Presentations (10/11, and 10/13)

- Slide 1: Title (with the team info)
- Slide 2: Problem statement
- Slide 3: Your approach
- Slide 4: Your software
- Slide 5: Plan for software testing
- Slide 6: Evaluation Method

Presentation should be no longer than 12 minutes including the Q&A session. (10 minutes: presentation, 2 minutes: Q&A)
- All of the team members should present
- Audience will get 2% of participation score based on their questions and attendance
- Please send me your slides at least 2 hours before your presentation

Objectives

- Recommendation systems
  - Collaborative filtering
  - Latent factor approach
  - Linear Methods

Large scale data analysis using Spark
CASE STUDY: Recommending Music and the Audioscrobbler Dataset
Evaluating the Recommendation Model
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

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
- Spark’s BinaryClassificationMetrics
  - Computes AUC per users and averages the result
  - Generating mean AUC

MAP metric

- Mean average precision
  - Focuses on the top recommendations

Computing AUC

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

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

def areaUnderCurve(
  positiveData: RDD[(Int, Int)],
  predictFunction: (RDD[(Int, Int)]) => RDD[Int]) = {

  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)

  auc
}
```
**k-Fold Cross-validation**

- Create a k-fold partition of the dataset
  - The remaining fold for testing

<table>
<thead>
<tr>
<th>Experiment 1</th>
<th>Experiment 2</th>
<th>Experiment 3</th>
<th>Experiment 4</th>
</tr>
</thead>
</table>

**Total number of examples**

**True error estimate**

- 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

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

**k-Fold Cross-validation with Spark**

```python
def predictMostListened(sc: SparkContext, train: RDD[Rating]):
    val bListenCount = sc.broadcast(
        train.map(r => (r.product, r.rating)).
        reduceByKey(_ + _).
        collectAsMap()!
    )
    allData.map { case (user, product) =>
        Rating(user, product, bListenCount.value.getOrElse(product, 0.0))
    }

    val auc = areaUnderCurve(cvData, bAllItemIDs, predictMostListened(sc, trainData))
```

**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

**Large scale data analysis using Spark**

**CASE STUDY: Recommendation Systems**

**Amazon.com : Item-to-item collaborative filtering**

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

Item-to-item collaborative filtering
- 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(NM) 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.

Recommendation quality

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

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.

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
  \[ f(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots \]

<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>85</td>
<td>62</td>
<td>67</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>75</td>
<td>80</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 or mean of the squares of the errors
  - Least squares regression

Squared error

- Squared error
  - Strongly penalizes very large errors
  - Drawback
    - It 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

\[ \text{RMSD} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2} \]

Linear Regression

- Calculating a linear regression
- Using the least square criterion

<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>85</td>
<td>62</td>
<td>67</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>75</td>
<td>80</td>
</tr>
</tbody>
</table>

\[ h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots \]

- To simplify the notation,

\[ h(x) = \sum_{i=0}^{n} \theta_i x_i \]

where, the \( \theta_i \)s are the parameters (Math scores, or slopes)
Objective function (Cost function)

- For a given training set, how do we pick, or learn, the parameter θ?
- Make $h(x)$ close to $y$
- Make your prediction close to the real observation

- We define the **objective (cost) function**

$$J(\theta) = \frac{1}{2} \sum_{i=0}^{m} (h_\theta(x^i) - y^i)^2$$

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

Concept of Gradient descent algorithm (1/2)

Concept of Gradient descent algorithm (2/2)

Stochastic Gradient descent algorithm

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for j=0 and j=1)

}

- Simultaneous update
  - Your implementation should perform simultaneous update
  - See slide 41

Simultaneous update

<table>
<thead>
<tr>
<th>Correct: Simultaneous update</th>
<th>Incorrect:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{temp0} := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$</td>
<td>$\text{temp0} := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$</td>
</tr>
<tr>
<td>$\text{temp1} := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$</td>
<td>$\text{temp1} := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$</td>
</tr>
<tr>
<td>$\hat{\theta}_0 := \text{temp0}$</td>
<td>$\hat{\theta}_0 := \text{temp0}$</td>
</tr>
<tr>
<td>$\hat{\theta}_1 := \text{temp1}$</td>
<td>$\hat{\theta}_1 := \text{temp1}$</td>
</tr>
<tr>
<td>$\hat{\theta}_{0,1} := \text{temp0}$</td>
<td>$\hat{\theta}_{0,1} := \text{temp1}$</td>
</tr>
</tbody>
</table>
Decreasing $\theta_j$

- Positive slope

$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_j)$

Increasing $\theta_j$

- Negative slope

$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_j)$

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 $\theta_j$ 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

$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_j)$

Using Gradient Descent Algorithm for Linear Regression Model

Gradient descent algorithm

Repeat until convergence

$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta_j)$

(for $j=0$ and $j=1$)

Linear Regression Model

$h_\theta(x) = \theta_0 + \theta_1 x_i$

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$
\[ \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)})^2 \]

\[ = \frac{\partial}{\partial \theta_j} \frac{1}{m} \sum (h_\theta(x^{(i)}) - y^{(i)})^2 \]

Case 1, \( \theta_j = \theta_0 \):

\[ \frac{\partial}{\partial \theta_0} f(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)}) \]

Case 2, \( \theta_j = \theta_1 \):

\[ \frac{\partial}{\partial \theta_1} f(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)}) \]

Gradient descent for Linear Regression

Repeat until convergence {

\[ \theta_0 \leftarrow \theta_0 - \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)}) \]

\[ \theta_1 \leftarrow \theta_1 - \frac{1}{m} \sum h_\theta(x^{(i)}) - y^{(i)}) \] (for \( j=0 \) and \( j=1 \))

Update \( \theta_0 \) and \( \theta_1 \) simultaneously.

Multiple local optimal points

Convex function

Fitting \( h_\theta(x) \)
**“Batch” Gradient Descent**

- Batch
  - Each step of gradient descent uses all of the training example

\[
\theta_j := \theta_j + \alpha \frac{1}{m} \sum_{i=1}^{m} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}
\]

**Running with Spark in parallel**

- For the sample size 1,000 (m=1,000)
  - Batch gradient descent:
    \[
    \theta_j := \theta_j + \alpha \frac{1}{1,000} \sum_{i=1}^{1000} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}
    \]
  - Using 4 machines
  - Step 1. 4 input splits
    - \((x^{(1)}, y^{(1)}), \ldots, (x^{(250)}, y^{(250)})\)
    - \((x^{(251)}, y^{(251)}), \ldots, (x^{(500)}, y^{500})\)
    - \((x^{(501)}, y^{(501)}), \ldots, (x^{(750)}, y^{750})\)
    - \((x^{(751)}, y^{(751)}), \ldots, (x^{(1000)}, y^{1000})\)
  - Step 2. Calculate temp1 - 4
    - \(\text{temp1} = \sum_{i=1}^{250} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}\)
    - \(\text{temp2} = \sum_{i=1}^{250} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}\)
    - \(\text{temp3} = \sum_{i=1}^{250} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}\)
    - \(\text{temp4} = \sum_{i=1}^{250} (y_i^{(i)} - h_\theta(x_i^{(i)}))x_j^{(i)}\)
  - Step 3. Calculate final results