CS535 BIG DATA

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

For building model	For testing model

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 `BinaryCalssficationMetrics`
  - 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

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

def areaUnderCurve(
    positiveData: RDD[ Rating],
    bAllItemIDs: 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))
```

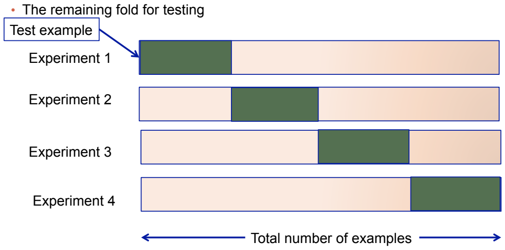

Computing AUC

- continued

```
trainData.cache()
cvData.cache()

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)
```

k-Fold Cross-validation

- Create a k-fold partition of the dataset
  - For each of the k experiments use K-1 folds for training
    - The remaining fold for testing



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

- `MLUtils.kFold()`

```
def predictMostListened(
   sc: SparkContext,
   train: RDD[Rating])(allData:
RDD[( Int, Int)]) = {

   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
  - The number of columns, k
- `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?

- Calculating the similarity between a single product and all related products:

```
For each item in product catalog, I1
  For each customer C who purchased I1
    For each item I2 purchased by customer C
      Record that a customer purchased I1 and I2
  For each item I2
    Compute the similarity between I1 and I2
```


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

Large scale data analysis using 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

$f(x) = w_0 + w_1x_1 + w_2x_2 + \ldots$

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

- 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
    - 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

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

Linear Regression

- Calculating a linear regression
  - Using the least square criterion

	A	B	C	D	E	F	G	H
Math score	85	85	62	67	89	65	80	67
Physics score	89	92	70	95	95	80	75	80

$h_\theta(x) = \theta_0 + \theta_1 x_1$

where, the θs are the **parameters** (Math scores, or slopes)

Linear Regression -- continued

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \ldots$$

- To simplify the notation,

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

Objective function (Cost function)

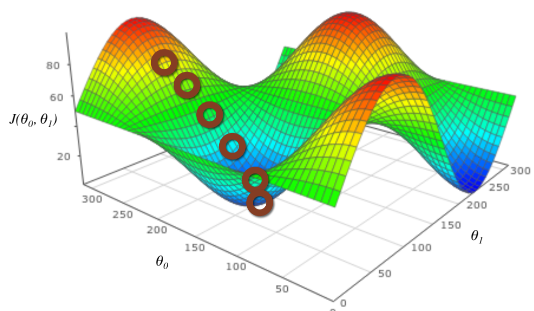
- For a given training set, how do we pick, or learn, the parameter θ?
- Make $h(x)$ close to y
  - Make your prediction close to the real observation
- We define the **objective (cost) function**

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

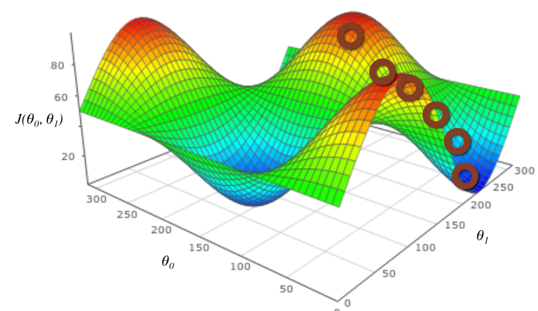
Minimization problem

- 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 θ_0, θ_1
  - Keep changing θ_0, θ_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)

}

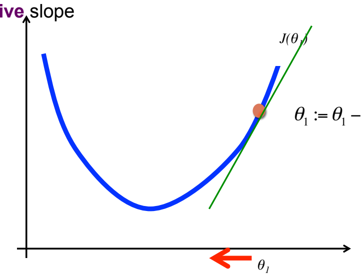
- Simultaneous update
  - Your implementation should perform simultaneous update
    - See slide 41

Simultaneous update

Correct: Simultaneous update	**Incorrect:**
temp0 := $\theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$	temp0 := $\theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$
	θ_0 := temp0
temp1 := $\theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)$	temp1 := $\theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)$
θ_0 := temp0 θ_1 := temp1	θ_1 := temp1

Decreasing θ_1

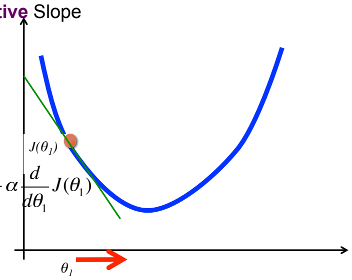
- **Positive** slope



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Increasing θ_1

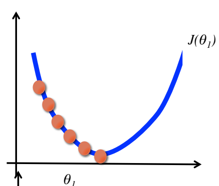
- **Negative** Slope



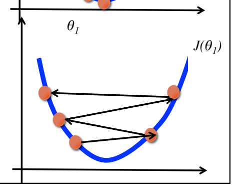
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Learning rates

- If α is too small, gradient descent can be slow

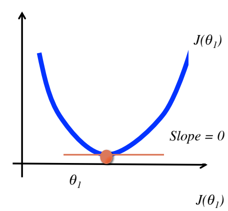


- If α is too large, gradient descent can overshoot the minimum
  - It may fail to converge
  - Or even diverge



Learning rates: starting at the optimal value?

- Current value of θ_1 has converged already
  - No more iteration required



Fixed learning rate α

- 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** α over time

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Using Gradient Descent Algorithm for Linear Regression Model

Gradient descent algorithm

Repeat until convergence {

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

(for j=0 and j=1)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

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

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

$$=\frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x^{(i)}-y^{(i)})^2$$

Case 1, θ_0 $(j=0)$:

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

Case 2, θ_1 $(j=1)$:

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

Gradient descent for Linear Regression

Repeat until convergence {

$$\theta_0 := \theta_0-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})$$

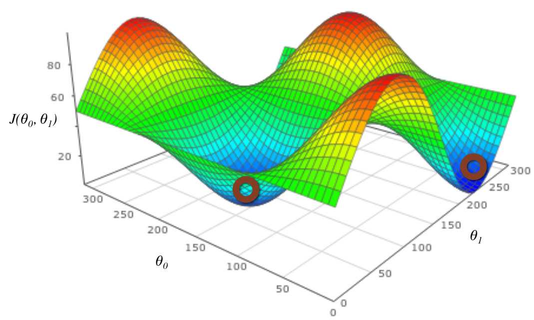
$$\theta_1 := \theta_1-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)}$$

(for j=0 and j=1)

}

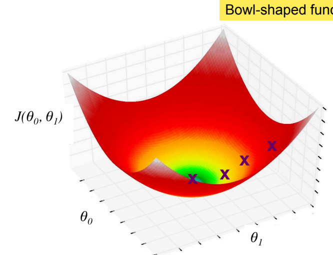
Update θ_0 and θ_1 simultaneously

Multiple local optimal points

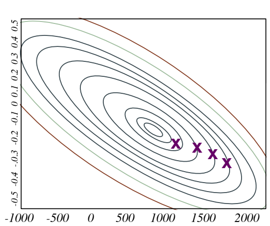


Convex function

Bowl-shaped function

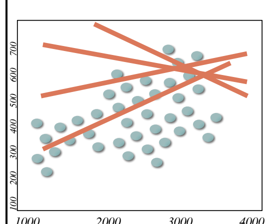


$J(\theta_0,\theta_1)$

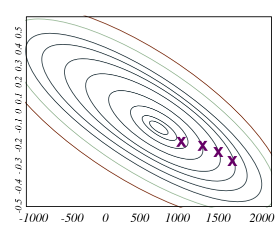


Fitting $h_\theta(x)$

$h_\theta(x)$



$J(\theta_0,\theta_1)$



"Batch" Gradient Descent

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

$$\theta_j := \theta_j + \alpha \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

Running with Spark in parallel

- For the sample size 1,000 (m=1,000)

- Batch gradient descent:

$$\theta_j := \theta_j + \alpha \frac{1}{1{,}000} \sum_{i=1}^{1000} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

- Using 4 machines

continued

- 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

- Step 3. Calculate final results

$$temp1 = \sum_{i=1}^{250} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

$$temp2 = \sum_{i=1}^{500} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

$$temp3 = \sum_{i=1}^{750} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

$$temp4 = \sum_{i=1}^{1000} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$