Announcements:

- Project Milestone feedback this week
- Thu this week: HW3 due / HW4 released (start early ③)

Large-Scale Machine Learning (1)

CS547 Machine Learning for Big Data Tim Althoff PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

New Topic: ML that scales!



Supervised Learning

Given some data:

 "Learn" a function to map from the input to the output

Given:

<u>Training</u> examples $(x_i, y_i = f(x_i))$ for some unknown function f

Find:

A good approximation to \boldsymbol{f}

Many Other ML Paradigms

Supervised:

• Given "labeled data" $\{x, y\}$, learn f(x) = y

Unsupervised:

• Given only "unlabeled data" $\{x\}$, learn f(x)

Semi-supervised:

Given some labeled $\{x, y\}$ and some unlabeled data $\{x\}$, learn f(x) = y

Active learning:

When we predict f(x) = y, we then receive true y*
Transfer learning:

• Learn f(x) so that it works well on new domain f(z)

Supervised Learning

- Would like to do prediction: estimate a function f(x) so that y = f(x)
- Where y can be:
 - Continuous / Real number: Regression
 - Categorical: Classification
 - Complex object:
 - Ranking of items, Parse tree, etc.

Data is labeled:

- Have many pairs {(x, y)}
 - x ... vector of binary, categorical, real valued features
 - y ... class, or a real number

Supervised Learning

- Task: Given data (X, Y) build a model f() to predict Y' based on X'
- Strategy: Estimate y = f(x) on (X, Y)
 Hope that the same f(x) also works to predict unknown Y'



- The "hope" is called generalization
 - Overfitting: If f(x) predicts well Y but is unable to predict Y'
- We want to build a model that <u>generalizes</u> well to unseen data

Why Large-Scale ML?

Brawn or Brains?

 In 2001, Microsoft researchers ran a test to evaluate 4 of different approaches to ML-based language translation

Findings:

- Size of the dataset used to train the model mattered more than the model itself
- As the dataset grew large, performance difference between the models became small



Banko, M. and Brill, E. (2001), "Scaling to Very Very Large Corpora for Natural Language Disambiguation"

Why Large-Scale ML?

The Unreasonable Effectiveness of Big Data

 In 2017, Google revisited the same type of experiment with the latest Deep Learning models in computer vision

Findings:

- Performance increases logarithmically
 based on volume of training data
- Complexity of modern ML models (i.e., deep neural nets) allows for even further performance gains



Large datasets + large ML models => amazing results!!

"Revisiting Unreasonable Effectiveness of Data in Deep Learning Era": https://arxiv.org/abs/1707.02968

Decision Trees

Decision Tree Learning

- Given one attribute (e.g., lifespan), try to predict the value of new people's lifespans by means of some of the other available attribute
- Input attributes:
 - **d** features/attributes: $x^{(1)}, x^{(2)}, \dots x^{(d)}$
 - Each $x^{(j)}$ has **domain** O_j
 - Categorical: $O_j = \{brown, blue, gray\}$
 - Numerical: $H_j = (0, 10)$
 - Y is output variable with domain O_Y:
 - Categorical: Classification, Numerical: Regression
- Data D:
- *n* examples (*x_i*, *y_i*) where *x_i* is a *d*-dim feature vector, *y_i* ∈ *O_Y* is output variable
 Task:
 - Given an input data vector x predict output label y

Decision Trees

A Decision Tree is a tree-structured plan of a set of attributes to test in order to predict the output



Decision Trees

Decision trees:

- Split the data at each internal node
- Each leaf node makes a prediction
- Lecture today:
 - Binary splits: $X^{(j)} < v$
 - Numerical attributes
 - Regression



How to make predictions?

- Input: Example x_i
- Output: Predicted \hat{y}_i
- "Drop" x_i down the tree until it hits a leaf node
- Predict the value stored in the leaf that x_i hits



Decision Trees: feature space



How to construct a tree?

• Training dataset D^* , $|D^*| = 100$ examples



Tim Althoff, UW CS547: Machine Learning for Big Data, http://www.cs.washington.edu/cse547

How to construct a tree?

- Imagine we are currently at some node *G*
- Let D_G be the data that reaches G
 There is a decision we have to make: Do we continue building the tree?



- If yes, which variable and which value do we use for a split?
 - Continue building the tree recursively
- If not, how do we make a prediction?
 - We need to build a "predictor node"

3 steps in constructing a tree



Require: Node n, Dat<u>a</u> $D \subseteq D^*$

:
$$(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$$
 (1)

2: if StoppingCriteria
$$(D_L)$$
 then

3:
$$n \rightarrow \text{left_prediction} = \text{FindPrediction}(D_L)$$
(3)

4: else

5: BuildSubtree
$$(n \rightarrow \operatorname{left}, D_L)$$

- 6: if StoppingCriteria (D_R) then
- 7: $n \rightarrow \text{right_prediction} = \text{FindPrediction}(D_R)$
- 8: else

9: BuildSubtree $(n \rightarrow \operatorname{right}, D_R)$

Requires at least a single pass over the data!

How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion

- Regression: Purity
 - Find split (X⁽ⁱ⁾, v) that creates D, D_L, D_R: parent, left, right child datasets and maximizes:



 $|\boldsymbol{D}| \cdot \boldsymbol{Var}(\boldsymbol{D}) - (|\boldsymbol{D}_L| \cdot \boldsymbol{Var}(\boldsymbol{D}_L) + |\boldsymbol{D}_R| \cdot \boldsymbol{Var}(\boldsymbol{D}_R))$

• $Var(D) = \frac{1}{n} \sum_{i \in D} (y_i - \overline{y})^2$... variance of y_i in D

How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion

- <u>Classification</u>: Information Gain
 - Measures how much a given attribute X tells us about the class Y
 - IG(Y | X): We must transmit Y over a binary link. How many bits on average would it save us if both ends of the line knew X?



Why Information Gain? Entropy

Entropy: What's the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from *X*'s distribution?

The entropy of X: $H(X) = -\sum_{j=1}^{m} p(X_j) \log p(X_j)$

- "High Entropy": X is from a uniform (flat) distribution
 - A histogram of the frequency distribution of values of X is flat
- "Low Entropy": X is from a varied (peaks/valleys) distrib.
 - A histogram of the frequency distribution of values of X would have many lows and one or two peaks



Why Information Gain? Entropy

- Suppose I want to predict Y and I have input X
 - X = College Major
 - Y = Likes Movie "Casablanca"

Х	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

From	this	data	we	estimate

•
$$P(Y = Yes) = 0.5$$

•
$$P(X = Math \& Y = No) = 0.25$$

•
$$P(X = Math) = 0.5$$

- P(Y = Yes | X = History) = 0Note:
- $H(Y) = -\frac{1}{2}\log_2(\frac{1}{2}) \frac{1}{2}\log_2(\frac{1}{2}) = \mathbf{1}$ • H(X) = 1.5

Why Information Gain? Entropy

- Suppose I want to predict Y and I have input X
 - X = College Major
 - Y = Likes "Casablanca"

Х	Υ
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

Def: Specific Conditional Entropy

H(*Y* | *X* = *v*) = The entropy of *Y* among only those records in which *X* has value *v*

Example:

- H(Y|X = Math) = 1
- H(Y|X = History) = 0
- -H(Y|X=CS) = 0

Why Information Gain?

- Suppose I want to predict Y and I have input X
 - X = College Major
 - Y = Likes "Casablanca"

Х	Υ
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

Def: Conditional Entropy

- H(Y | X) = The average specific conditional entropy of Y
 - = if you choose a record at random what will be the conditional entropy of Y, conditioned on that row's value of X
 - Expected number of bits to transmit Y if both sides will know the value of X

$$= \sum_{j} P(X = v_{j}) H(Y|X = v_{j})$$

Why Information Gain?

Suppose I want to predict Y and I have input X

H(Y | X) = The average specific conditional entropy of Y

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

$$=\sum_{j}P(X=v_{j})H(Y|X=v_{j})$$

Example:

Vj	P(X=v _j)	H(Y X=v _j)
Math	0.5	1
History	0.25	0
CS	0.25	0

So: H(Y|X)=0.5*1+0.25*0+0.25*0 = **0.5**

Why Information Gain?

Suppose I want to predict Y and I have input X

Def: Information Gain

Х	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

 IG(Y|X) = I must transmit Y. How many bits on average would it save me if both ends of the line knew X?
 IG(Y|X) = H(Y) - H(Y | X)

- Example:
 - H(Y) = 1
 - H(Y|X) = 0.5
 - Thus IG(Y|X) = 1 0.5 = 0.5

What is Information Gain used for?

- Suppose you are trying to predict whether someone is going to live past 80 years
- From historical data you might find:
 - IG(LongLife | HairColor) = 0.01
 - IG(LongLife | Smoker) = 0.4
 - IG(LongLife | Gender) = 0.25
 - IG(LongLife | LastDigitOfSSN) = 0.00001
- IG tells us how much information about Y is contained in X
 - So attribute X that has high IG(Y|X) is a good split!

3 steps in constructing a tree



Tim Althoff, UW CS547: Machine Learning for Big Data, http://www.cs.washington.edu/cse547

When to stop?

(2) When to stop?

- Many different heuristic options to avoid overfitting
- Two ideas:
 - (1) When the leaf is "pure"
 - The target variable does not vary too much: Var(y) < ε
 - (2) When # of examples in the leaf is too small
 - For example, $|D| \le 100$
 - (3) Stop at a fixed depth
 - For example, max depth = 4.



How to predict?

(3) How to predict?Many options

Regression:

- Typically: Predict average y_i
 of the examples in the leaf
- Build a linear regression model on the examples in the leaf

Classification:

Predict most common y_i of the examples in the leaf



Building Decision Trees Using MapReduce

Problem: Building a tree

- Given a large dataset with hundreds of attributes
- Build a decision tree!
- General considerations:
 - Tree is small (can keep it memory):
 - Shallow (~10 levels)
 - Dataset too large to keep in memory (Petabytes)
 - Dataset too big to scan over on a single machine
 - MapReduce to the rescue!



Today's Lecture: PLANET

Parallel Learner for Assembling Numerous Ensemble Trees [Panda et al., VLDB '09]

- A sequence of MapReduce jobs that builds a decision tree
- Spark MLlib Decision Trees are based on PLANET

Setting:

- Hundreds of numerical (discrete & continuous, but not categorical) attributes
- Target variable is numerical: Regression
- Splits are binary: X^(j) < v</p>
- Decision tree is small enough for each Mapper to keep it in memory
- Data too large to keep in memory

PLANET Architecture





Tim Althoff, UW C3547 Machine Learning for Big Data, http://www.cs.washington.edu/cse547

PLANET: Building the Tree

- The tree will be built in levels
 - One level at a time:





Steps:

- 1) Master decides candidate splits (n, X^(j), v)
- 2) MapReduce computes quality of those splits
- 3) Master then grows the tree for a level

4) Goto (1)

Decision trees on MapReduce



PLANET Overview



We build the tree level by level

One MapReduce step builds one level of the tree

Mapper

- Considers a number of candidate splits (node, attribute, value) on its subset of the data
- For each split it stores partial statistics
- Partial split-statistics is sent to Reducers

Reducer

- Collects all partial statistics and determines best split
- Master grows the tree for one level

PLANET Overview



- Mapper loads the DT model and info '-- '-- '-- about which attribute splits (split is a tuple <NodelD, Attribute, Value>) to consider
 - Each mapper sees a subset of the data D*
 - Mapper "drops"/classifies each datapoint d using the tree to find the leaf node L where d lands
 - For each leaf node L mapper keeps statistics about
 - (1) the data reaching L
 - (2) the data in left/right subtree under some split S
- Reducer aggregates the statistics (1), (2) and determines the best split for each tree node

PLANET: Components



Master

- Monitors everything (runs multiple MapReduce jobs)
- Three types of MapReduce jobs:
 - (1) MapReduce <u>Initialization</u> (run once first)
 - For each attribute identify values to be considered for splits
 - (2) MapReduce <u>FindBestSplit</u> (run multiple times)
 - MapReduce job to find best split (when there
 - is too much data to fit in memory)
 - (3) MapReduce InMemoryBuild (run once last)
 - Similar to BuildSubTree (but for small data)
 - Grows an entire sub-tree once the data fits in memory

Model file

A file describing the state of the model

PLANET: Components

- (1) Master Node
- (2) MapReduce Initialization (run once first)
- (3) MapReduce <u>FindBestSplit</u> (run multiple times)
- (4) MapReduce InMemoryBuild (run once last)

PLANET: Master



- Master controls the entire process
- Determines the state of the tree and grows it:
 - (1) Decides if nodes should be split
 - (2) If there is little data entering a tree node, Master runs an <u>InMemoryBuild</u> MapReduce job to grow the entire subtree below that node
 - (3) For larger nodes, Master launches MapReduce
 <u>FindBestSplit</u> to evaluate candidates for best split
 - Master also collects results from FindBestSplit and chooses the best split for a node
 - (4) Updates the model

PLANET: Components

- (1) Master Node
- (2) MapReduce Initialization (run once first)
- (3) MapReduce <u>FindBestSplit</u> (run multiple times)
- (4) MapReduce InMemoryBuild (run once last)

Initialization: Attribute metadata

- Initialization job: Identifies all the attribute values which need to be considered for splits
 - Initialization process generates "attribute metadata" to be loaded in memory by other tasks
- Main question: Which splits to even consider?
- A split is defined by a triple: (node n, attribute X^(j), value v)



Initialization: Attribute metadata

Which splits to even consider?

- For small data we can sort the values along a particular feature and consider every possible split
- But data values may not be uniformly populated so many splits may not really make a difference

X(*i*): 1.2, 1.3, 1.4 1.6 2.1, 7.2 8.1 9.8 10.1 10.2 10.3 10.4 11.5 11.7 12.8 12.9

 Idea: Consider a limited number of splits such that splits "move" about the same amount of data

Initialization: Attribute metadata

- Splits for numerical attributes:
 - For attribute X^(j) we would like to consider every possible value v ∈O_i
 - Compute an approx. equi-depth histogram on D*
 - Idea: Select buckets such that counts per bucket are equal



Use boundary points of histogram as splits

Side note: Computing Equi-Depth



Domain values

- Goal: Equal number of elements per bucket
 (*B* buckets total)
- Construct by first sorting and then taking
 B-1 equally-spaced splits

1 2 2 3 4 7 8 9 10 10 10 10 10 11 11 12 12 14 16 16 18 19 20 20 20 20

Faster construction:

Sample & take equally-spaced splits in the sample

Nearly equal buckets

PLANET: Components

- (1) Master Node
- (2) MapReduce Initialization (run once first)
- (3) MapReduce <u>FindBestSplit</u> (run multiple times)
- (4) MapReduce InMemoryBuild (run once last)

FindBestSplit

- Goal: For a particular split node *n* find attribute
 X^(j) and value *v* that maximize Purity:
 - $|D| \cdot Var(D) (|D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R))$
 - D ... training data (x_i, y_i) reaching the node n
 - **D**_L ... training data \mathbf{x}_i , where $\mathbf{x}_i^{(j)} < \mathbf{v}$
 - **D**_R ... training data \mathbf{x}_i , where $\mathbf{x}_i^{(j)} \ge \mathbf{v}$

•
$$Var(D) = \frac{1}{n} \sum_{i \in D} y_i^2 - \left(\frac{1}{n} \sum_{i \in D} y_i\right)^2$$

FindBestSplit

To compute Purity we need

•
$$Var(D) = \frac{1}{n} \sum_{i \in D} y_i^2 - \left(\frac{1}{n} \sum_{i \in D} y_i\right)^2$$

Important observation: Variance can be computed from sufficient statistics:
 N S-Sy O-Sy²

N,
$$S=\Sigma y_i$$
, $Q=\Sigma y_i^2$

- Each Mapper m processes subset of data D_m, and computes N_m, S_m, Q_m for its own D_m
- Reducer combines the statistics and computes global variance and then Purity:

•
$$Var(D) = \frac{1}{\sum_{m} N_{m}} \sum_{m} Q_{m} - \left(\frac{1}{\sum_{m} N_{m}} \sum_{m} S_{m}\right)^{2}$$

FindBestSplit: Map



Mapper:

- Initialized by loading results of Initialization task
 - Current model (to find which node each datapoint x_i ends up)
 - Attribute metadata (all split points for each attribute)
 - Load the set of candidate splits: {(node, attribute, value)}
- For each data record run the Map algorithm:
 - For each tree node store statistics of the data entering the node and at the end emit (to all reducers):

NodelD, { S=Σy, Q=Σy², N=Σ1 } >

- For each split store statistics and at the end emit:
 - <SplitID, { S, Q, N } >
 - SplitID = (node id, attribute X^(j), split value v)

FindBestSplit: Reducer



Reducer:

- (1) Load all the <NodeID, <u>List</u> {S_m, Q_m, N_m}> pairs and aggregate the per node statistics
- (2) For all the <SplitID, List {S_m, Q_m, N_m}> aggregate the statistics

•
$$Var(D) = \frac{1}{\sum_{m} N_m} \sum_{m} Q_m - \left(\frac{1}{\sum_{m} N_m} \sum_{m} S_m\right)^2$$

For each NodelD, output the best split found

Overall system architecture

Master gives the mappers: (1) Tree





Overall system architecture

- Example: Need to split nodes F, G, H, I
 Map and Reduce:
 - FindBestSplit::Map (each mapper)
 - Load the current model M
 - Drop every example x_i down the tree
 - If it hits *F/G/H/I*, update in-memory hash tables:
 - For each node: T_n : (Node) \rightarrow {S, Q, N}
 - For each (Split, Node): $T_{n,j,s}$: (Node, Attribute, SplitValue)→{S, Q, N}
 - Map::Finalize: output the key-value pairs from above hashtables
 - FindBestSplit::Reduce (each reducer)
 - Collect:
 - T1:<Node, List{S, Q, N} > \rightarrow <Node, { Σ S, Σ Q, Σ N} >
 - T2:<(Node, Attr., Val), List{S, Q, N}> \rightarrow <(Node, Attr., Val), { Σ S, Σ Q, Σ N}>
 - Compute Purity for each node using T1, T2
 - Return best split to Master (which then decides on globally best split)



Back to the Master



- Collects outputs from FindBestSplit reducers <Split.NodeID, Attribute, Value, Purity>
- For each node decides the best split
 - If data in D_L/D_R is small enough, later run a MapReduce job
 InMemoryBuild on the node
 - Else run MapReduce
 FindBestSplit job for both nodes



Decision Trees: Conclusion

Decision Trees

Characteristics

Classification & Regression

- Multiple (~10) classes
- Real valued and categorical features
- Few (hundreds) of features
- Usually dense features
- Complicated decision boundaries
 - Early stopping to avoid overfitting!

Example applications

- User profile classification
- Landing page bounce prediction

Decision Trees

Decision trees are the single most popular data mining tool:

- Easy to understand
- Easy to implement
- Easy to use
- Computationally cheap
- It's possible to mitigate overfitting (i.e., with ensemble methods or early stopping)
- They do classification as well as regression!

Learning Ensembles

- Learn multiple trees and combine their predictions
 - Gives better performance in practice

Bagging:

- Learns multiple trees over independent samples of the training data
 - For a dataset **D** on **n** data points: Create dataset **D'** of **n** points but sample from **D** with replacement
 - Dataset D' will include duplicate data points
- Predictions from each tree are averaged to compute the final model prediction

Bagging Decision Trees Instance Random Forest Tree-2 Tree-n Tree-1 Class-B Class-B Class-A Majority-Voting Final-Class

Improvement: Random Forests

- Train a Bagged Decision Tree
- But use a modified tree learning algorithm that selects (at each candidate split) a random subset of the features
 - If we have d features, consider \sqrt{d} random features
- This is called: Feature bagging
 - Benefit: Breaks correlation between trees
 - Otherwise, if one feature is very strong predictor, then every tree will select it, causing trees to be correlated.

Random Forests achieve state-of-the-art results in many classification problems!