Learning for Web Rankings

David Grangier
AT&T Labs Research
grangier@att.com

Jean François (JF) Paiement
AT&T Labs Research
jpaiement@att.com

September 2011
Machine Learning for Web Rankings

- What are Web Rankings?
- What is Machine Learning?
- Classical Learning Algorithms
- Learning Approaches to Web Rankings
Web Rankings

Rankings are everywhere

- Google, Bing **search** results
  text query $\rightarrow$ url ranking
- Yelp, Yellowpages listings
  business category, location $\rightarrow$ business ranking
- Yahoo! related news articles
  current article, user history $\rightarrow$ article ranking
- Netflix, Amazon **recommendations**
  purchasing history, submitted reviews $\rightarrow$ item ranking
Web Rankings

Primary Goal:

Given the query (text query, user profile, location, purchase history, etc.), the ranker orders the corpus items, such that the ordering shows the most relevant first.

(Optional) Secondary Goals: diversity, advertising revenue...
Web Rankings

How?

- a scoring function \( f \) assigns a real value to each \((\text{query, item})\) pair,
- given a query, the items are scored and sorted by decreasing scores.

Example

- query: fast food
- corpus:
  - Wallgreen
  - McDonald’s
  - Century 21
  - Chipotle Mexican
  - Home Depot
Web Rankings

How?

- a **scoring** function $f$ assigns a real value to each \((query, item)\) pair,
- given a query, the items are scored and sorted by **decreasing scores**.

Example

- query: fast food
- scoring:
  - Wallgreen
    \[ f(\text{fast food}, \text{Wallgreen}) = 0.2 \]
  - McDonald’s
    \[ f(\text{fast food}, \text{McDonald’s}) = 4.1 \]
  - Century 21
    \[ f(\text{fast food}, \text{Century 21}) = 0.1 \]
  - Chipotle Mexican
    \[ f(\text{fast food}, \text{Chipotle Mexican}) = 2.1 \]
  - Home Depot
    \[ f(\text{fast food}, \text{Home Depot}) = -0.1 \]
Web Rankings

How?

- a **scoring** function $f$ assigns a real value to each *(query, item)* pair,
- given a query, the items are scored and sorted by **decreasing scores**.

Example

- **query:** fast food
- **sorting:**
  
<table>
<thead>
<tr>
<th>Item</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald’s</td>
<td>4.2</td>
</tr>
<tr>
<td>Chipotle Mexican</td>
<td>2.1</td>
</tr>
<tr>
<td>Wallgreen</td>
<td>0.2</td>
</tr>
<tr>
<td>Century 21</td>
<td>0.1</td>
</tr>
<tr>
<td>Home Depot</td>
<td>-0.1</td>
</tr>
</tbody>
</table>
More formally...

• Each (query, item) pair is represented by a vector of features
  \[ \Phi(\text{query}, \text{item}) \in \mathbb{R}^n. \]

• The scoring function
  \[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]
Features

- Each (query, item) pair is represented by $\Phi(\text{query, item}) \in \mathbb{R}^n$.
- Examples of features from web search:
  - Number of common words between the query and the webpage text
  - Number of common words between the query and the webpage title
  - Number of common words between the query and the webpage URL
  - Webpage length
  - Webpage readability
  - Likelihood to be spam
  - webpage popularity (page rank), etc...
- Feature engineering is a big effort in this industry
Scoring Function

- $f$ is the main factor in the quality of the ranking (other factors: corpus size & accuracy, feature quality, result presentation...)
- 1960s–90s, hand design $f$ optimizing quality on a small query set.
- 2000s–now, **automatic selection** relying on **massive datasets**:
  - editors evaluate massive (query, document) sets for quality,
  - implicit feedback from users (click or non-click on results),
  - traffic statistics from toolbars (e.g. Yahoo, Google, Bing toolbars).
Scoring Function

- \( f \) is the main factor in the quality of the ranking (other factors: corpus size & accuracy, feature quality, result presentation...)
- 1960s–90s, hand design \( f \) optimizing quality on a small query set.
- 2000s–now, **learning from data**:  
  - editors evaluate massive (query, document) sets for quality,  
  - implicit feedback from users (click or non-click on results),  
  - traffic statistics from toolbars (e.g. Yahoo, Google, Bing toolbars).
Scoring Function Learning

Examples of **data sources** for web-search

- **explicit supervision**
  - editors grades the relevance of document \( d \) for query \( q \),
  - editors manually re-order the results for a query,
  - users reports a link as spam, inappropriate...

- **implicit feedback from users**
  - click on a low ranked results (positive feedback)
  - no click on a highly ranked result (negative)
  - click on a link and come back soon to result page (negative)
  - reformulating the query (negative)...

- **other sources**
  - toolbar traces, query misspellings...
Scoring Function Learning

Goal of the learning process

- on a new query, relevant documents should appear above the others.
- this is assessed by evaluating a **quality metric** on a set of **labeled test queries**
A quality metric induces preferences over output configurations e.g. which one is the best ranking?

<table>
<thead>
<tr>
<th></th>
<th>1–relevant</th>
<th>-</th>
<th>2–relevant</th>
<th>-</th>
<th>-</th>
<th>3–relevant</th>
<th>4–relevant</th>
<th>-</th>
<th>5–relevant</th>
<th>-</th>
<th>6–relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Quality metrics for web-search

- Precision at 10
- Discounted Cumulative Gain at 10
- etc...
Quality Metrics

Precision at 10
For each test query, documents are labeled as \{relevant, non-relevant\}

\[
P@10 = \frac{\text{number of relevant doc. in top 10}}{10}
\]
Quality Metrics

Discounted Cumulative Gain at 10
For any test query, doc. labels $\in \{0 \text{ (non-relevant)}, \ldots, 5 \text{ (very relevant)}\}$

$$DCG@10 = \sum_{i=1}^{10} p_i \cdot l(d_i)$$

where $p_i \downarrow$ when $i \uparrow$ (more weight to top positions)

$l(d) \uparrow$ with relevance of $d$ (more to weight to rel. docs)
Web Rankings: Conclusions

- web rankings are ubiquitous
- ranking function quality is core
- lots of data to drive ranking function selection
- **select the ranking function from these training data**

This is a machine learning problem!
BTW What is Machine Learning?
Machine Learning for Web Rankings

- What are Web Rankings?
- **What is Machine Learning?**
- Classical Learning Algorithms
- Learning Approaches to Web Rankings
We are concerned with **predictive modeling**.

given a **labeled training set** of \( n \) (input, output) examples

\[
(x_1, y_1), \ldots, (x_n, y_n)
\]

we want to **predict** the output for any **new** example \( x \),

\[ x \rightarrow y? \]
Examples of learning problems

<table>
<thead>
<tr>
<th></th>
<th>input ((x))</th>
<th>output ((y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>regression</td>
<td>mon.–sat. temperature</td>
<td>sun. temperature</td>
</tr>
<tr>
<td></td>
<td>house features</td>
<td>selling price</td>
</tr>
<tr>
<td>classification</td>
<td>digital image</td>
<td>human face or not</td>
</tr>
<tr>
<td></td>
<td>email features</td>
<td>spam or non-spam</td>
</tr>
<tr>
<td>ranking</td>
<td>query and documents</td>
<td>document ranking</td>
</tr>
<tr>
<td></td>
<td>user’s watching history</td>
<td>movie ranking</td>
</tr>
</tbody>
</table>
Learning Process

Learning Algorithm
- it takes the training set \((x_1, y_1), \ldots, (x_n, y_n)\)
- it selects \(f\) in a family of functions \(F\)

After Learning
- \(f\) is used for prediction, i.e. given a new \(x\), we predict \(f(x)\)
Learning Process

Learning Algorithm

• it takes the training set \((x_1, y_1), \ldots, (x_n, y_n)\)
• it selects \(f\) in a family of functions \(F\)

After Learning

• \(f\) is used for prediction, i.e. given a new \(x\), we predict \(f(x)\)

What objective drives the selection of \(f\)?
Learning Objective:

- **Generalization Performance** We want $f$ to perform well on the unlabeled examples we receive after learning, the **test examples**.

- **Training Performance** This is different from performing well on the points for which the algorithm was given the label, the **training examples**.
How to reach high generalization performance?

**Occam’s Razor** (14th-century logician)

*The simplest explanation is most likely the correct one.*

**Learning Theory**

- when selecting $f$ in $F$, prefer simple functions to complex ones
How to reach high generalization performance?

**Occam’s Razor** (14th-century logician)

*The simplest explanation is most likely the correct one.*

**Learning Theory**

- when selecting $f$ in $F$, prefer simple functions to complex ones
- the complexity of $f$ is called its capacity.
How to reach high generalization performance?

**Occam’s Razor** (14th-century logician)

*The simplest explanation is most likely the correct one.*

**Learning Theory**

- when selecting $f$ in $F$, prefer simple functions to complex ones
- the complexity of $f$ is called its capacity.
- the learning process achieves a trade-off between
  - $f$ is simple, i.e. low capacity.
  - $f$ explains the training data well, i.e. low training error.
How to reach high generalization performance?

Example of the trade-off low capacity, low training error.

- function family: polynomials
- capacity: measured by the polynomial degree,
  e.g. $\text{degree}(2x + 1) = 1$
  $\text{degree}(3x^3 + 5x^2 + x) = 3$
- error: $(f(x) - y)^2$
How to reach high generalization performance?

Data

- Train points
- Test points
How to reach high generalization performance?

Degree 1 Polynomial

training error: 39.1 – test error: 36.4
How to reach high generalization performance?

Degree 2 Polynomial

training error: 5.2 – test error: 6.1
How to reach high generalization performance?

Degree 8 Polynomial

training error: 0.5 – test error: 2,735.9
How to reach high generalization performance?

Learning Curve

<table>
<thead>
<tr>
<th>degree (capacity)</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>300</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>150</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- **train**
- **test**

<table>
<thead>
<tr>
<th>train err.</th>
<th>test err.</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>under-fitting</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>over-fitting</td>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>
How to reach high generalization performance?

![Learning Curve]

- **Underfitting**: High train error, high test error, too low capacity.
- **Overfitting**: Low train error, high test error, too high capacity.

- **Capacity**: Influence of degree (capacity) on error rate.

---

David Grangier, September '11
How to reach high generalization performance?

![Learning Curve](image)

**Validation**

- How to set capacity?
- exclude data from the train set, the validation data (simulate test).
- vary capacity, estimate test err using validation data.
- select model which performs best on the validation set.
What is Machine Learning?

- **Learning**: selecting a predictor $f \in F$ from training data
- **Goal**: low test error
- **How**: training data error/capacity tradeoff
What is a Learning Algorithm?

Given

- family of function $F$
  e.g. linear functions $f_w(x) = w \cdot x = \sum_i w_i x_i$

- measure of capacity, also referred as regularizer
  e.g. weight L1 norm $\text{Reg}(f_w) = \|w\|_1 = \sum_i |w_i|$

- measure of error
  e.g. squared error $(f_w(x) - y)^2$
What is a Learning Algorithm?

Given

- family of function $F$
  - e.g. linear functions $f_w(x) = w \cdot x = \sum_i w_i x_i$
- measure of capacity, also referred as regularizer
  - e.g. weight L1 norm $\text{Reg}(f_w) = \|w\|_1 = \sum_i |w_i|$
- measure of error
  - e.g. squared error $(f_w(x) - y)^2$

Then learning is cast as an optimization problem

- $f^* = \arg\min_f \text{Error}(f, \text{train}) + \lambda \text{Reg}(f)$
  - where $\lambda$ sets the trade-off between train error and capacity
  - e.g. $w^* = \arg\min_w \sum_{(x,y) \in \text{train}} (f_w(x) - y)^2 + \lambda\|w\|_1$
What is a Learning Algorithm?

Given

- family of function $F$
  
  e.g. linear functions $f_w(x) = w \cdot x = \sum_i w_i x_i$
  
- measure of capacity, also referred as regularizer
  
  e.g. weight L1 norm $\text{Reg}(f_w) = \|w\|_1 = \sum_i |w_i|$

- measure of error
  
  e.g. squared error $(f_w(x) - y)^2$

Then learning is cast as an optimization problem

- $f^* = \arg\min_f \text{Error}(f, \text{train}) + \lambda \text{Reg}(f)$
  
  where $\lambda$ sets the trade-off between train error and capacity
  
  e.g. $w^* = \arg\min_w \sum_{(x,y)\in\text{train}} (f_w(x) - y)^2 + \lambda \|w\|_1$

- validation is used to select $\lambda$. 

David Grangier, September '11 26 / 69
What is a Learning Algorithm?

Given

- family of function $F$
  
  e.g. linear functions $f_w(x) = w \cdot x = \sum_i w_i x_i$

- measure of capacity, also referred as regularizer
  
  e.g. weight L1 norm $\text{Reg}(f_w) = \|w\|_1 = \sum_i |w_i|$

- measure of error
  
  e.g. squared error $(f_w(x) - y)^2$

Then learning is cast as an optimization problem

- $f^* = \arg\min_f \text{Error}(f, \text{train}) + \lambda \text{Reg}(f)$
  
  where $\lambda$ sets the trade-off between train error and capacity

  e.g. $w^* = \arg\min_w \sum_{(x,y) \in \text{train}} (f_w(x) - y)^2 + \lambda \|w\|_1$

- specific optimizer often proposed for scalability
Machine Learning for Web Rankings

- What are Web Rankings?
- What is Machine Learning?
- Classical Learning Algorithms
- Learning Approaches to Web Rankings
Classical Learning Algorithms

- **Linear learning algorithms**
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)
- **Non-linear learning algorithms**
  - neural networks
  - decision trees
  - gradient boosted trees
Linear Learning Algorithms

$F$ is the set of linear functions,

$$f_w(x) = w \cdot x \text{ where } x \in \mathbb{R}^d, w \in \mathbb{R}^d$$

- $x$ is a feature vector of dimension $D$ describing an example.
- $w$ is a weight vector defining the function $f_w$.
- $f_w(x)$ is the prediction of $f_w$ for example $x$. 
Linear Learning Algorithms

$F$ is the set of linear functions,

$$f_w(x) = w \cdot x \text{ where } x \in \mathbb{R}^d, w \in \mathbb{R}^d$$

Regularization is (mostly) either

- **L2 regularization**, $|w|^2_2 = \sum_i w_i^2$
  $\rightarrow$ prefer ”flat” weights (all weights are the same).
  $\rightarrow$ the output is not too dependent on a single component of $x$

- **L1 regularization**, $|w|^1_1 = \sum_i |w_i|$
  $\rightarrow$ prefer ”sparse” weights (i.e. most weights are zero)
  $\rightarrow$ only few components in $x$ explains the outputs
Classical Learning Algorithms

- Linear learning algorithms
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)
- Non-linear learning algorithms
  - neural networks
  - decision trees
  - gradient boosted trees
Linear Learning for Regression

Task: predict a real value.
e.g. a temperature, a stock price, an aircraft speed...

The most common error measure is the Mean Squared Error (MSE)

\[
\text{MSE}(f_w, \text{set}) = \frac{1}{|\text{set}|} \sum_{(x,y)\in \text{set}} (f_w(x) - y)^2
\]

Popular linear regression techniques

Ridge regression  MSE with L2 regularization
\[
w^* = \arg\min_w \text{MSE}(f_w, \text{set}) + \lambda \|w\|_2^2
\]

LASSO  MSE with L1 regularization
\[
w^* = \arg\min_w \text{MSE}(f_w, \text{set}) + \lambda \|w\|_1
\]
Classical Learning Algorithms

- Linear learning algorithms
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)
- Non-linear learning algorithms
  - neural networks
  - decision trees
  - gradient boosted trees
Linear Learning for Classification

Task: discriminate between two classes, positive class, $y = +1$; negative class, $y = -1$.

e.g. is a face present in the image? spam or non-spam e-mail? etc.
Linear Learning for Classification

Task: discriminate between two classes, positive class, $y = +1$
negative class, $y = -1$.

e.g. is a face present in the image? spam or non-spam e-mail? etc.

Prediction

$$f_w(x) = w \cdot x > 0 \quad \text{predicts} \quad +1, \text{ positive class}$$
$$< 0 \quad \text{ - } -1, \text{ negative class}$$
Linear Learning for Classification

Task: discriminate between two classes, positive class, \( y = +1 \)

negative class, \( y = -1 \).

e.g. is a face present in the image? spam or non-spam e-mail? etc.

Prediction

\[ f_w(x) = w \cdot x \]

\( > 0 \) predicts \( +1 \), positive class

\( < 0 \) predicts \( -1 \), negative class

Classification error: counts the number of errors, i.e. when \( y f_w(x) < 0 \)

\[ \text{Err}(f_w, \text{set}) = \frac{1}{|\text{set}|} \sum_{(x,y) \in \text{set}} I\{y f_w(x) < 0\} \]

where \( I\{c\} = 1 \) if \( c \) is true, 0 otherwise.
Linear Learning for Classification

Classification error: counts the number of errors, i.e. when \( y f_w(x) < 0 \)

\[
E = \text{Err}(f_w, \text{set}) = \frac{1}{|\text{set}|} \sum_{(x,y) \in \text{set}} I\{y f_w(x) < 0\}
\]

Optimization problem: for learning, we solve

\[
w^* = \arg\min_w \text{Err}(f_w, \text{train}) + \lambda \text{Reg}(f_w)
\]

- optimizers like differentiable, convex functions
- \( \text{Err} \) is not!
Linear Learning for Classification

**Classification error:** counts the number of errors, i.e. when $y \ f_w(x) < 0$

$$E = \text{Err}(f_w, \text{set}) = \frac{1}{|\text{set}|} \sum_{(x,y) \in \text{set}} l\{y \ f_w(x) < 0\}$$

**Optimization problem:** for learning, we solve

$$w^* = \arg\min_w \ \text{Err}(f_w, \text{train}) + \lambda \ \text{Reg}(f_w)$$

- optimizers like differentiable, convex functions
- $\text{Err}$ is not!

→ replace $\text{Err}$ with a differentiable, convex upper bound.
replace \textbf{Err} with a differentiable, convex upper bound.

\[ I\{y \ f_w(x) < 0\} \quad \text{classification error} \]
Linear Learning for Classification

→ replace $\text{Err}$ with a differentiable, convex upper bound.

$$\log(1 + \exp(-y f_w(x))) \quad \text{logistic loss}$$
Linear Learning for Classification

→ replace $\text{Err}$ with a differentiable, convex upper bound.

$$\max(0, 1 - y f_w(x)) \quad \text{hinge loss}$$
Linear Learning for Classification

→ replace $\text{Err}$ with a differentiable, convex upper bound.

$$\exp(-y f_w(x))$$ exponential loss
### Linear Learning for Classification

#### Common algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Loss</th>
<th>Regularizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>Logistic loss</td>
<td>L1 or L2</td>
</tr>
<tr>
<td>(Linear) Support Vector Machines (SVM)</td>
<td>Hinge loss</td>
<td>L2</td>
</tr>
<tr>
<td>Adaboost</td>
<td>Exponential loss</td>
<td>Kind of L1</td>
</tr>
</tbody>
</table>
Classical Learning Algorithms

- **Linear learning algorithms**
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)

- **Non-linear learning algorithms**
  - neural networks
  - decision trees
  - gradient boosted trees
Non-linear Algorithms

Why?
  • some complex input/output relation requires a non-linear model

But...
  • difficult, costly optimization problem for learning
Classical Learning Algorithms

- Linear learning algorithms
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)
- Non-linear learning algorithms
  - neural networks
  - decision trees
  - gradient boosted trees
Artificial Neural Networks

- inspired by biological neural networks (e.g. your brain).
- compose several layers
- each layer performs a linear operation or a non-linear activation.
Non-linear Algorithms: Neural Networks

3 layer neural networks

\[ f_{A,B}(x) = B \ \text{sigmoid}(A x) \]

linear: \[ z_1 = A \ x, \ \text{where} \ x \in \mathbb{R}^d, A \in \mathbb{R}^{d \times h} \] (input layer)
activation: \[ z_2 = \text{sigmoid}(z_1) \] (hidden layer)
linear: \[ z_3 = B \ z_2, \ \text{where} \ B \in \mathbb{R}^h \] (output layer)

Note:
- \( h \) is referred as the number of hidden units.
- Sigmoid applies \( t \rightarrow \frac{1-\exp(-t)}{1+\exp(t)} \) to each component of the vector.
3 layer neural networks are **Universal Approximators**

- any function can be approximated by a neural network
- for any function \( g \) and \( \epsilon > 0 \), there exists a neural network \( f \), s.t.

\[
\forall x, |f(x) - g(x)| < \epsilon
\]
Non-linear Algorithms: Neural Networks

Learning Algorithm

- Learning parameters through gradient descent optimization
  - find $A, B$ which minimize $\text{Loss}(f_{A,B}, \text{train}) + \lambda \text{Regularizer}$
  - works with any differentiable loss.
- NNs can be applied to classification (logistic loss) or regression (MSE)
- regularization by controlling
  - $h$, the number of hidden units
  - L2 regularization on $A, B$...
Classical Learning Algorithms

• Linear learning algorithms
  • regression algorithms (LASSO, Ridge reg.)
  • classification algorithms (logistic reg., linear SVM, adaboost)

• Non-linear learning algorithms
  • neural networks
  • decision trees
  • gradient boosted trees
Decision Trees

- examples travel the tree from the root to the leaf
- each node performs a test on the input $x$
- each leaf corresponds to a prediction

$$f(x) = \begin{cases} 
0.5 & \text{if } x_4 > 0.8 \\
0.8 & \text{if } x_4 \leq 0.8 \\
0.3 & \text{if } x_2 \leq 0.5 \\
0.1 & \text{if } x_2 > 0.5 \\
\end{cases}$$
Decision Trees

Advantages

• low computation cost for prediction
• easy to interpret

Disadvantages

• regression tree only models piecewise constant function
• learning deep trees req. a lot of training examples
• greedy learning yields a sub-optimal test sequence
Decision Trees: Greedy Learning for Regression

Greedy Learning Algorithm

1. starts with a tree containing only the root
2. splitting: for each node with depth $\leq \text{max\_depth}$,
   - find best test & create two leaves from the node
   - find best prediction for these two leaves
3. repeat 2 until no more nodes with depth $\leq \text{max\_depth}$.
Greedy Learning Algorithm

\[ f(x) = 0.5 \]
Greedy Learning Algorithm

\[ f(x) = 0.5 \]

Best split: \( x_2 > .5 \),

Best predictions:  
\[
\begin{align*}
  f(x) &= 0.7 \quad \text{if } x_2 > .5 \\
  f(x) &= 0.2 \quad \text{if } x_2 \leq .5
\end{align*}
\]
Greedy Learning Algorithm

$x_2 > 0.5$

- **true**
  - $f(x) = 0.7$

- **false**
  - $f(x) = 0.2$
Greedy Learning Algorithm

\[ f(x) = 0.7 \]

\[ x_2 > 0.5 \]

\[ f(x) = 0.2 \]

Best split: \( x_4 > 0.8 \),

Best predictions:

\[ f(x) = 0.5 \quad \text{if} \quad x_4 > 0.8 \]
\[ f(x) = 0.8 \quad \text{if} \quad x_4 \leq 0.8 \]
Greedy Learning Algorithm

\[ f(x) = 0.5 \]

\[ f(x) = 0.8 \]

\[ f(x) = 0.2 \]
Greedy Learning Algorithm

Best split: \( x_1 > .2 \),

Best predictions:
\[
\begin{align*}
  f(x) &= 0.3 \quad \text{if} \quad x_1 > .2 \\
  f(x) &= 0.1 \quad \text{if} \quad x_2 \leq .2
\end{align*}
\]
Greedy Learning Algorithm

\[
x_2 > 0.5 \\
x_1 > 0.2 \\
x_4 > 0.8
\]

\[
\text{true} \quad f(x) = 0.5 \\
\text{false} \quad f(x) = 0.8 \\
\text{true} \quad f(x) = 0.3 \\
\text{false} \quad f(x) = 0.1
\]
Learning Decision Trees

- Decision trees can be applied for classification & regression.
- Regularization by controlling
  - depth
  - number of training examples to estimate a split
Decision Trees: Greedy Learning for Regression

Greedy Learning Algorithm

1. starts with a tree containing only the root
2. splitting: for each node with depth < $\text{max\_depth}$,
   - find best test & create two leaves from the node
   - find best prediction for these two leaves
3. repeat 2 until no more nodes with depth < $\text{max\_depth}$.
Decision Trees: Greedy Learning for Regression

Splitting Node $N$ (Step 2)

- Given $S_N$ all training examples reaching $N$,
- we find
  
  $i, t$ defining the test $x_i < t$
  
  $a, b$ the prediction when $x_i < t$ is true or false

  to minimize MSE

  $$
  \sum_{(x, y) \in S_N: x_i \leq t} (a - y)^2 + \sum_{(x, y) \in S_N: x_i > t} (b - y)^2
  $$

- efficient algorithm, complexity $D \times \|S_N\|$
Classical Learning Algorithms

- Linear learning algorithms
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)

- Non-linear learning algorithms
  - neural networks
  - decision trees
  - gradient boosted decision trees
Gradient Boosted Decision Trees

A GBDT is a mixture of decision trees

\[ f(x) = \sum_{t=1}^{T} h_t(x) \text{ where } \forall t, \ h_t \text{ is a tree.} \]

Advantages

- model more complex functions than a single tree.
- learning many shallow trees req. less training data than one deep tree.

Regularization

- like trees, depth and minimum number of training examples in a leaf
- \( T \), number of trees
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

- Goal: find $f$ to minimize

\[
\text{Loss}(f, \text{train}) = \sum_{(x, y) \in \text{train}} \text{Loss}(f(x), y)
\]

- add a new tree at each step:
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$

- current model: $f_t = \sum_{i=1}^{t-1} h_t$
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For \( t = 1, \ldots, T \)

- current model: \( f_t = \sum_{i=1}^{t-1} h_t \)
- for each training example \((x, y)\),
  - predict \( \hat{y} = f_t(x) \) and derive \( g_x = \frac{\partial \text{Loss}}{\partial \hat{y}} (\hat{y}, y) \)

\[ \frac{\partial \text{Loss}}{\partial \hat{y}} \]
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$
- current model: $f_t = \sum_{i=1}^{t-1} h_t$
- for each training example $(x, y)$,
  - predict $\hat{y} = f_t(x)$ and derive $g_x = \frac{\partial \text{Loss}}{\partial \hat{y}}(\hat{y}, y)$
- learn next tree $h_t$
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$

- current model: $f_t = \sum_{i=1}^{t-1} h_t$
- for each training example $(x, y)$,
  - predict $\hat{y} = f_t(x)$ and derive $g_x = \frac{\partial \text{Loss}}{\partial \hat{y}}(\hat{y}, y)$
- learn next tree $h_t$
  - the gradient indicates how to decrease the loss
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$

- current model: $f_t = \sum_{i=1}^{t-1} h_t$
- for each training example $(x, y)$,
  - predict $\hat{y} = f_t(x)$ and derive $g_x = \frac{\partial \text{Loss}}{\partial \hat{y}}(\hat{y}, y)$
- learn next tree $h_t$
  - the gradient indicates how to decrease the loss
  - i.e. $f_t(x) - \alpha g_x$ yields a lower loss (for small $\alpha > 0$)
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$

- current model: $f_t = \sum_{i=1}^{t-1} h_t$
- for each training example $(x, y)$,
  - predict $\hat{y} = f_t(x)$ and derive $g_x = \frac{\partial \text{Loss}}{\partial \hat{y}}(\hat{y}, y)$
- learn next tree $h_t$
  - the gradient indicates how to decrease the loss
  - i.e. $f_t(x) - \alpha g_x$ yields a lower loss (for small $\alpha > 0$)
  - we want a tree s.t. $h_t(x) \simeq -\alpha g_x$
Gradient Boosted Decision Trees

Iterative Learning called Gradient Boosting

For $t = 1, \ldots, T$

- current model: $f_t = \sum_{i=1}^{t-1} h_t$
- for each training example $(x, y)$,
  - predict $\hat{y} = f_t(x)$ and derive $g_x = \frac{\partial \text{Loss}}{\partial \hat{y}}(\hat{y}, y)$
- learn next tree $h_t$
  - the gradient indicates how to decrease the loss
  - i.e. $f_t(x) - \alpha g_x$ yields a lower loss (for small $\alpha > 0$)
  - we want a tree s.t. $h_t(x) \simeq -\alpha g_x$
  - decision tree for regression to minimize $\sum_{(x,y) \in \text{train}} (h_t(x) - (-\alpha g_x))^2$
Classical Learning Algorithms

- Linear learning algorithms
  - regression algorithms (LASSO, Ridge reg.)
  - classification algorithms (logistic reg., linear SVM, adaboost)
- Non-linear learning algorithms
  - neural networks
  - decision trees
  - gradient boosted decision trees
Machine Learning for Web Rankings

- What are Web Rankings?
- What is Machine Learning?
- Classical Learning Algorithms
- Learning Approaches to Web Rankings
Learning the scoring function: \( q, d \to f(\phi(q, d)) \)

- various family of functions
  - e.g. linear, neural networks, trees...
- but the core effort has been about loss functions.
Loss Functions for Web Rankings

A loss function

- measures performance on the training data
- should be **optimizable** during learning:
  - continuous, differentiable, possibly convex in the model parameters.
- common quality metrics like $P@10$ or $DCG@10$ are not
  - depends only on score ordering
  - depends only on whether $f(q, d') < f(q, d)$ for all pairs $d, d'$
- $P@10$, $DCG@10$ are piecewise constant functions
Loss Functions for Web Rankings

3 types of approaches

- regression
- pairwise
- listwise
Loss Functions for Web Rankings: Regression

Training data: query, document pairs labeled with a relevance level.

<table>
<thead>
<tr>
<th>query (q)</th>
<th>document (d)</th>
<th>relevance (r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen to music</td>
<td>pandora radio</td>
<td>+5</td>
</tr>
<tr>
<td>listen to music</td>
<td>san francisco giants</td>
<td>0</td>
</tr>
<tr>
<td>car repairs</td>
<td>car talk</td>
<td>+3</td>
</tr>
<tr>
<td>car repairs</td>
<td>jiffy lube oil change</td>
<td>+2</td>
</tr>
</tbody>
</table>

Regression Learning: predict relevance levels.

\[ f = \arg \min_{f} \sum_{(q, d, r) \in \text{train}} (f(q, d) - r)^2 + \lambda \text{Reg}(f) \]
Loss Functions for Web Rankings: Regression

Drawback

• does not reflect ranking quality closely.
  e.g. 3 documents (a,b,c) with relevance +2, +1, 0

<table>
<thead>
<tr>
<th></th>
<th>score(a)</th>
<th>score(b)</th>
<th>score(c)</th>
<th>ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>(a,b,c)</td>
</tr>
<tr>
<td>case 2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>(c,b,a)</td>
</tr>
</tbody>
</table>

• only relative scores matter for ranking
Loss Functions for Web Rankings: Pairwise Loss

Only relative scores matter for ranking

- given two items $d, d'$ labeled $r, r'$ for query $q$, we want

$$f(q, d) > f(q, d') \text{ if and only if } r > r'$$

Pairwise Loss leverages binary classification
Only relative scores matter for ranking

- given two items \( d, d' \) labeled \( r, r' \) for query \( q \), we want

\[
\begin{align*}
  f(q, d) & > f(q, d') \quad \text{if and only if} \quad r & > r' \\
  f(q, d) - f(q, d') & > 0 \quad \text{and} \quad r - r' & > 0
\end{align*}
\]
Loss Functions for Web Rankings: Pairwise Loss

Only relative scores matter for ranking

- given two items $d, d'$ labeled $r, r'$ for query $q$, we want

\[
\begin{align*}
f(q, d) &> f(q, d') \quad \text{if and only if} \quad r > r' \\
f(q, d) - f(q, d') &> 0 \\
f(q, d) - f(q, d') &> 0
\end{align*}
\]

where $y_{r,r'} = \text{sign}\{r - r'\}$.

Pairwise Loss leverages binary classification

In classification, we want $f(x) > 0$ if and only if $y > 0$
Loss Functions for Web Rankings: Pairwise Loss

Only relative scores matter for ranking

- given two items $d, d'$ labeled $r, r'$ for query $q$, we want

\[
\begin{align*}
    f(q, d) &> f(q, d') \quad \text{if and only if} \quad r > r' \\
    f(q, d) - f(q, d') &> 0 \\
    y_{r, r'} &> 0
\end{align*}
\]

where $y_{r, r'} = \text{sign}\{r - r'\}$.

Pairwise Loss leverages binary classification

- In classification, we want $f(x) > 0$ if and only if $y > 0$
Loss Functions for Web Rankings: Pairwise Loss

- In binary classification,
  \[ \text{Loss}(f(x), y) \]
  where \text{Loss} is
  - logistic loss (logistic regression)
  - hinge loss (SVM)
  - exponential loss (Adaboost)

- For ranking, \text{pairwise loss} works with pair of examples \((d, d')\)
  \[ \text{Loss}(f(q, d) - f(q, d'), y_r, r') \]
  where \text{Loss} is
  - logistic loss (RankNet)
  - hinge loss (Ranking SVM)
  - exponential loss (RankBoost)
Loss Functions for Web Rankings: Pairwise Loss

Example with Hinge Loss (SVM):

- binary classification, we minimize

\[ \sum_{(x,y) \in \text{train}} \max(0, 1 - y f(x)) + \lambda \text{Reg}(f) \]

- pairwise rankings, we minimize

\[ \sum_{(q,d,d',y_{r,r'}) \in \text{train}} \max(0, 1 - y_{r,r'} (f(q, d) - f(q, d'))) + \lambda \text{Reg}(f) \]
Loss Functions for Web Rankings: Pairwise Loss

Drawback

- all pairs are considered equals.
- does not emphasize the top of the ranking.
Ideal Loss for Ranking

- Given the vector $y^q$ with the labels of all documents for query $q$, the vector $z^q$ with the scores of all documents for query $q$,
- $L(z^q, y^q)$ should be such that
  - it is continuous, differentiable
  - its gradient indicates the direction which improves the metric of interest, like $P@10$ or $DCG@10$.
- that is not easy!
Loss Functions for Web Rankings: Listwise Loss

$L(z^q, y^q)$ should be such that

- it is continuous, differentiable
- its gradient indicates the direction which improves the metric of interest, like $P@10$ or $DCG@10$.

Different Approaches

- probabilistic (ListNet, SoftRank),
- gradient-driven (LambdaRank, LambdaMART),
- structured prediction (Structured SVM, Graph Transformer Networks...)

David Grangier, September '11
Loss Functions for Web Rankings: Listwise Loss

$L(z^q, y^q)$ should be such that

- it is continuous, differentiable
- its gradient indicates the direction which improves the metric of interest, like $P@10$ or $DCG@10$.

Probabilistic Approaches

- define $P(\sigma|z^q)$ probability of an ordering given scores
  $L(z^q, y^q) = \sum_\sigma P(\sigma|z^q) \ DCG@10(\sigma, y^q)$
- such that $L(z^q, y^q)$ is differentiable and cheap to compute.
- e.g. ListNet, SoftRank
Machine Learning for Web Rankings

- What are Web Rankings?
- What is Machine Learning?
- Classical Learning Algorithms
- Learning Approaches to Web Rankings

That’s it!
Thank you for your attention.