An Optimizer’s View of Statistical Boosting Algorithms
(Colloquium of the Chilean Institute of Operations Research)

Robert Freund (research with Paul Grigas and Rahul Mazumder)

http://web.mit.edu/rfreund/www/talks.html

Massachusetts Institute of Technology

April 2013
The convex optimization problem is:

\[
f^* := \min_x f(x) \quad \text{s.t.} \quad x \in P
\]

\(P \subset \mathbb{R}^n\) is a convex set

\(f(x) : P \to \mathbb{R}\) is a convex function

\(f(x)\) is perhaps differentiable, or not

Let \(f^*\) denote the optimal objective function value

Let us discuss some algorithms for convex optimization . . . .
Four Relevant Algorithms for Convex Optimization

1. Gradient Descent

2. Subgradient Descent

3. Mirror Descent on the $n$-simplex $\{x \in \mathbb{R}^n : x \geq 0 \text{ and } \sum_j x_j = 1\}$

4. Frank-Wolfe Method
Our problem of interest is:

\[ f^* := \min_x f(x) \]
\[ \text{s.t. } x \in \mathbb{R}^n \]

**Gradient Descent method for minimizing** \( f(x) \) **on** \( \mathbb{R}^n \)

Initialize at \( x_1 \in \mathbb{R}^n, k \leftarrow 1 \).

At iteration \( k \):

1. Compute \( \nabla f(x_k) \).
2. Choose step-size \( \alpha_k \).
3. Set \( x_{k+1} \leftarrow x_k - \alpha_k \nabla f(x_k) \).
Here is what we can say about the gradient descent method:

**Computational Guarantees for Gradient Descent**

If the step-size sequence \( \{ \alpha_k \} \) is chosen intelligently, then:

\[
f(x_k) - f^* \leq \frac{L}{k}
\]

where \( L \) is a certain constant related to the function \( f(x) \).
Subgradient Descent

Our problem of interest is:

\[ f^* := \min_x f(x) \quad \text{s.t.} \quad x \in \mathbb{R}^n \]

where \( f(x) \) is not differentiable. Then \( f(x) \) has subgradients.
Subgradient Descent, continued

Our problem of interest is:

\[ f^* := \min_{x} f(x) \quad \text{s.t.} \quad x \in \mathbb{R}^n \]

Subgradient Descent method for minimizing \( f(x) \) on \( \mathbb{R}^n \)

Initialize at \( x_1 \in \mathbb{R}^n, k \leftarrow 1 \).

At iteration \( k \):

1. Compute a subgradient \( g_k \) of \( f(x_k) \).
2. Choose step-size \( \alpha_k \).
3. Set \( x_{k+1} \leftarrow x_k - \alpha_k g_k \).
Here is what we can say about the subgradient descent method:

**Computational Guarantees for Subgradient Descent**

If the step-size sequence \{\alpha_k\} is chosen intelligently, then:

$$f(x_k) - f^* \leq \frac{L}{\sqrt{k}}$$

where $L$ is a certain constant related to the function $f(x)$. 
Our problem of interest is:

\[ f^* := \min_x f(x) \]
\[ \text{s.t. } \sum_j x_j = 1 \]
\[ x \geq 0 \]

The feasible region is the \( n \)-simplex \( \{x \in \mathbb{R}^n : \sum_j x_j = 1, x \geq 0\} \)
Our problem of interest is:

\[
\begin{align*}
  f^* := \min_x & \quad f(x) \\
  \text{s.t.} & \quad \sum_j x_j = 1 \quad \text{and} \quad x \geq 0
\end{align*}
\]

Mirror Descent method for minimizing \( f(x) \) on the \( n \)-simplex

Initialize at \( x_1 = (1/n, 1/n, \ldots, 1/n) \), \( k \leftarrow 1 \).

At iteration \( k \):

1. Compute a subgradient \( g_k \) of \( f(x_k) \).
2. Choose step-size \( \alpha_k \).
3. Set

\[
(x_{k+1})_j \leftarrow \frac{(x_k)_j e^{-\alpha_k(g_k)_j}}{\sum_{\ell} (x_k)_\ell e^{-\alpha_k(g_k)_\ell}}.
\]
Here is what we can say about the mirror descent method:

**Computational Guarantees for Mirror Descent**

If the step-size sequence \( \{\alpha_k\} \) is chosen intelligently, then:

\[
  f(x_k) - f^* \leq \frac{L}{\sqrt{k}}
\]

where \( L \) is a certain constant related to the function \( f(x) \).
Our problem of interest is:

\[
f^* := \min_x f(x) \quad \text{s.t.} \quad x \in P
\]

\(f(x)\) is differentiable

The feasible region \(P\) is convex and it is "easy" to do linear programming on \(P\)
Frank-Wolfe Method, continued

Our problem of interest is:

\[ f^* := \min_{x} f(x) \quad \text{s.t.} \quad x \in P \]

Frank-Wolfe method for minimizing \( f(x) \) on \( P \)

Initialize at \( x_1 \in P, \ k \leftarrow 1 \).

At iteration \( k \):

1. Compute \( \nabla f(x_k) \).
2. Compute \( \tilde{x}_k \leftarrow \arg \min_{x \in P} \{ \nabla f(x_k)^T x \} \).
3. Set \( x_{k+1} \leftarrow x_k + \bar{\alpha}_k (\tilde{x}_k - x_k) \), where \( \bar{\alpha}_k \in [0, 1] \).
Frank-Wolfe, continued

Here is what we can say about the Frank-Wolfe method:

Computational Guarantees for Frank-Wolfe

If the step-size sequence \( \{\bar{\alpha}_k\} \) is chosen intelligently, then:

\[
f(x_k) - f^* \leq \frac{L}{k}
\]

where \( L \) is a certain constant related to the function \( f(x) \).
Two Problems in Statistical Boosting

We consider two problems in statistical boosting:

1. Linear Regression

2. Binary Classification / Supervised Learning
Linear Regression

Consider the linear regression model:

\[ y = X\beta + e \]

- \( n \) observations/records, \( p \) independent variables

- \( X \in \mathbb{R}^{n \times p} \) is the model matrix: \( X_{ij} \) is the \( i^{th} \) observation of independent variable \( j \)

- \( y \in \mathbb{R}^n \) is the vector of observed values of the dependent variable

- \( \beta \in \mathbb{R}^p \) are the regression coefficients, with \( \beta_j \) the coefficient of the \( j^{th} \) independent variable

- \( e \in \mathbb{R}^n \) is the vector of the errors (or noise) in the model
Linear Regression, continued

Linear regression:

\[ y = X\beta + e \]

In the high-dimensional statistical regime, especially when \( p \gg n \gg 0 \), we desire:

- good predictive performance of the model (of course),
- good performance on currently available data (residuals \( r := y - X\beta \) are small),
- an “interpretable model” \( \beta \),
- coefficients are not excessively large (\( \sum_j |\beta_j| \leq \delta \)), and
- a “lean” or “sparse” solution: we do not want to use too many independent variables (\( \beta \) has few non-zero coefficients)
Linear Regression Performance Metrics

Linear regression:

\[ y = X\beta + e \]

The classical metric is the least-squares of the residuals \( r := y - X\beta \)

minimize \( \sum_i (r_i)^2 \) \( ( = \| r \|_2^2 ) \)

Another metric is the maximum correlation of the residuals

minimize \( \max_i |(X^T r)_i| \) \( ( = \| X^T r \|_\infty ) \)
A Boosting Version of Regression

Regression model:

\[ y = X\beta + e \]

- \( n \) observations/records, \( p \) regression models
- \( X \in \mathbb{R}^{n \times p} \) is the model matrix: \( X_{ij} \) is the \( i^{th} \) observation of \( j \)-th regression model
- \( y \in \mathbb{R}^n \) is the vector of observed values of the dependent variable
- \( \beta \in \mathbb{R}^p \) are the boosting coefficients, with \( \beta_j \) the coefficient of the \( j^{th} \) regression model
- \( e \in \mathbb{R}^n \) is the vector of the errors (or noise) in the overall boosting model
Incremental Forward Stagewise Regression ($\text{FS}_\varepsilon$)

Initialize at $r^0 = y$, $\beta^0 = 0$, $k = 0$.

At iteration $k$:

1. Compute $j_k \in \arg \max_{j \in \{1, \ldots, p\}} |(r^k)^T X_j|$

2. Set:

   
   \[
   r^{k+1} \leftarrow r^k - \varepsilon \ \text{sgn}((r^k)^T X_{j_k}) X_{j_k}
   \]

   
   \[
   \beta_{j_k}^{k+1} \leftarrow \beta_{j_k}^k + \varepsilon \ \text{sgn}((r^k)^T X_{j_k})
   \]

   
   \[
   \beta_j^{k+1} \leftarrow \beta_j^k, \ j \neq j_k
   \]
Incrmental Forward Stagewise Regression, continued

What has been known about $F S_\varepsilon$:

- after $k$ iterations, there are at most $k$ non-zero coefficients of $\beta^k$
- after $k$ iterations, $\sum_j |\beta^k_j| \leq k\varepsilon$
- method produces a fairly good model fairly quickly

What has not been known about $F S_\varepsilon$:

- why does it work efficiently in practice?
- will it fail on certain regression problems?
- how might we improve the method?
Incremental Forward Stagewise Regression, continued

Equivalence Theorem

Incremental Forward Stagewise Regression (FS$_\varepsilon$) is equivalent to subgradient optimization applied to the problem

$$\min_{r = y - X\beta} \max_i |(X^T r)_i| \equiv \|X^T r\|_\infty$$

with step-sizes $\alpha_k = \varepsilon$ at every iteration.
Incremental Forward Stagewise Regression, continued

Computational Guarantees for Incremental Forward Stagewise Regression

\( FS_\varepsilon \) has the following guarantee after \( k \) iterations:

The coefficients of \( \beta^\ell \) have at most \( \ell \) non-zeros, for \( \ell = 1, \ldots, k \), and the coefficients are not excessively large. Furthermore,

\[
\min_{\ell \in \{0, \ldots, k\}} \|X^T r^\ell\|_\infty \leq \frac{\|y\|^2}{2\varepsilon(k + 1)} + \frac{\varepsilon\|X\|^2}{2}
\]

If we choose the step-sizes like an optimizer does, then

\[
\min_{\ell \in \{0, \ldots, k\}} \|X^T r^\ell\|_\infty \leq \frac{\|y\|\|X\|}{\sqrt{k + 1}}
\]

and

\[
\sum_j |\beta_j^\ell| \leq \frac{\sqrt{k}\|y\|}{\|X\|}
\]
Frank-Wolfe for Regression

Let us apply Frank-Wolfe to solve the constrained least-squares problem:

\[
f^*_\delta := \min_{\beta} \sum_j (y - X\beta)^2 \quad \text{s.t.} \quad \sum_j |\beta_j| \leq \delta
\]

This is known as the LASSO problem
Frank-Wolfe for Regression, continued

Frank-Wolfe Method for LASSO Regression

Initialize at $\beta_0$ with $\sum_j |\beta_0^j| \leq \delta$.

At iteration $k$:

1. Compute:
   \[
   r^k \leftarrow y - X\beta^k
   \]
   \[
   j_k \leftarrow \arg \max_{j \in \{1, \ldots, p\}} |(r^k)^T X_j|
   \]

2. Set:
   \[
   \beta_{j_k}^{k+1} \leftarrow (1 - \bar{\alpha}_k) \beta_{j_k}^k + \bar{\alpha}_k \delta \text{sgn}((r^k)^T X_{j_k})
   \]
   \[
   \beta_j^{k+1} \leftarrow (1 - \bar{\alpha}_k) \beta_j^k \text{ for } j \neq j_k, \text{ and where } \bar{\alpha}_k \in [0, 1] \]
Suppose we run the Frank-Wolfe method using an intelligent step-size sequence. Then after $k$ iterations, there exists an $\ell \in \{0, \ldots, k\}$ satisfying:

1. $\frac{1}{2} \|y - X\beta^\ell\|^2 - f_\delta^* \leq \frac{17.4\|X\|^2\delta^2}{k}$
2. $\|X^T r^\ell\|_\infty \leq \frac{1}{2\delta} \|y\|^2 + \frac{17.4\|X\|^2\delta}{k}$
3. there are at most $k$ non-zero coefficients of $\beta^\ell$
4. $\sum_j |\beta^\ell_j| \leq \delta$
We are given $m$ points $x^1, \ldots, x^m \in \mathbb{R}^n$ each of which has a label 1 or $-1$

- each point $x^i = (x^i_1, x^i_2, \ldots, x^i_n)$ is an example
- we have $m$ examples
- $x^i_j$ is the value of feature $j$ for example $i$
- the label of point $x^i$ is $y^i$, and $y^i \in \{-1, 1\}$
  - the points $x^i$ with label $y^i = 1$ have property “P”
  - the points $x^i$ with label $y^i = -1$ do not have property “P”

We would like to use these $m$ points to develop a linear rule that can be used to predict whether or not other points $x$ might or might not have property P
Illustration of Supervised Learning

Illustration of the pattern classification problem
Another Illustration of Supervised Learning

Another Illustration of the pattern classification problem
Supervised Learning, continued

We seek a vector $\lambda \in \mathbb{R}^n$ for which:

- $(x^i)^T \lambda > 0$ for all $x^i$ for which $y^i = 1$
- $(x^i)^T \lambda < 0$ for all $x^i$ for which $y^i = -1$

We will then use $\lambda$ to predict whether or not other points $v$ have property P or not, using the rule:

- If $v^T \lambda > 0$, then we declare that $v$ has property P
- If $v^T \lambda < 0$, then we declare that $v$ does not have property P
We seek a vector $\lambda \in \mathbb{R}^n$ for which:

- $(x^i)^T \lambda > 0$ for all $x^i$ for which $y^i = 1$
- $(x^i)^T \lambda < 0$ for all $x^i$ for which $y^i = -1$

Define $a_i = y^i \cdot x^i$

We seek $\lambda$ for which $(a^i)^T \lambda > 0$ for all $i = 1, \ldots, m$

Define

$$A = \begin{bmatrix}
- & - & a^1 & - & - \\
- & - & a^1 & - & - \\
& & & \ddots & \\
- & - & a^m & - & -
\end{bmatrix}$$

We therefore seek $\lambda$ for which:

$$A\lambda > 0$$
Supervised Learning, Goals

We seek $\lambda$ for which:

$$A\lambda > 0$$

where the $i^{th}$ row of $A$ is $a_i := y^i \cdot x^i$

We desire:

- good prospective performance of the model (of course): $v^T\lambda > 0$ when $v$ has property P, $v^T\lambda < 0$ when $v$ does not have property P
- good performance on the training data: $A\lambda > 0$,
- an “interpretable model” $\lambda$
- coefficients are not excessively large: $\sum_j |\lambda_j| \leq \delta$, and
- a “lean” or “sparse” solution: we do not want to use too many features ($\lambda$ has few non-zero coefficients)
Statistical Learning Metrics

We seek $\lambda$ for which:

$$A\lambda > 0$$

where $i^{th}$ row of $A$ is $a_i := y^i \cdot x^i$

One performance metric is the margin, which is the smallest of the $(a^i)^T \lambda$ values:

$$p(\lambda) := \min_{i \in \{1, \ldots, m\}} (A\lambda)_i$$

We seek to maximize $p(\lambda)$ over all $\lambda$ satisfying $\sum_j |\lambda|_j \leq \delta$

Another metric is the exponential loss and is defined to be:

$$L_{\text{exp}}(\lambda) := \ln \left( \frac{1}{m} \sum_{i=1}^{m} \exp \left( - (A\lambda)_i \right) \right)$$

We seek to minimize $L_{\text{exp}}(\lambda)$ over all $\lambda$ satisfying $\sum_j |\lambda|_j \leq \delta$
A Boosting Version of Statistical Learning

We seek $\lambda$ for which:

$$A\lambda > 0$$

where $i^{th}$ row of $A$ is $a_i := y^i \cdot x^i$

- $m$ examples $x^1, \ldots, x^m$,
- $n$ different learning models,
- $A_{ij} = \pm 1$ is prediction of learning model $j$ on the $i^{th}$ example $x^i$
- $\lambda \in \mathbb{R}^n$ are the boosting coefficients, with $\lambda_j$ the coefficient of the $j^{th}$ learning model
The AdaBoost Algorithm

We seek $\lambda$ for which:

$$A\lambda > 0$$

where $i^{th}$ row of $A$ is $a_i := y^i \cdot x^i$

### AdaBoost Algorithm

Initialize at $w^0 = (1/m, \ldots, 1/m)$, $\lambda^0 = 0$, $k = 0$

At iteration $k$:

1. Compute $j_k \in \arg \max_{j=1,\ldots,n} |(w^T A)_j|$, $s = \text{sgn}((w^T A)_{j_k})$

2. Choose $\alpha_k \geq 0$ and set:

   $$\lambda^{k+1} \leftarrow \lambda^k + \alpha_k s e^{j_k}$$

   $$w_{i}^{k+1} \leftarrow w_{i}^{k} e^{-\alpha_k A_{i j_k}}$$ and re-normalize $w^{k+1}$ so that $\sum_j w_{j}^{k+1} = 1$
AdaBoost, continued

What has been known about AdaBoost:

- after $k$ iterations, there are at most $k$ non-zero coefficients of $\lambda^k$
- method produces a fairly good model fairly quickly
- various guarantees on exponential loss criterion depending on very specific step-size assumptions $\{\alpha_k\}$

What has not been known about AdaBoost:

- What, if anything, is AdaBoost optimizing or working on?
- how is it connected, if at all, to optimization methods?
- when is it more or less likely to perform well?
- how might we improve the method?
AdaBoost, continued

**AdaBoost Equivalence Theorem**

AdaBoost is computationally equivalent to Mirror Descent applied to the (dual of the) maximum margin problem:

\[
p^* := \max_{\lambda} \min_i (A\lambda)_i
\]

s.t. \( \sum_j |\lambda_j| \leq 1 \)
AdaBoost, continued

Computational Guarantees for AdaBoost

AdaBoost has the following guarantee after $k$ iterations:

If we re-normalize $\lambda^k$ so that $\sum_j |\lambda_j| = 1$ and if we choose the step-sizes like an optimizer, then

$$p^* - p(\lambda^k) \leq \sqrt{\frac{2 \ln(m)}{k}}$$

Furthermore

- $\lambda^k$ has at most $k$ non-zero coefficients, and
- $\sum_j |\lambda_j^k| = 1$
Frank-Wolfe for Statistical Learning

We seek $\lambda$ for which:

$$A\lambda > 0$$

where $i^{th}$ row of $A$ is $a_i := y_i \cdot x^i$

The exponential loss and is defined to be:

$$L_{\text{exp}}(\lambda) := \ln \left( \frac{1}{m} \sum_{i=1}^{m} \exp \left( -(A\lambda)_i \right) \right)$$

Let us apply Frank-Wolfe to minimize the exponential loss problem:

$$L_{\text{exp}}^* = \min_{\lambda} \quad L_{\text{exp}}(\lambda)$$

s.t. $\sum_j |\lambda_j| \leq \delta$
Frank-Wolfe for Exponential Loss Minimization

Frank-Wolfe method for Minimizing Exponential Loss

Initialize at $\lambda^0$ with $\sum_j |\lambda^0_j| \leq \delta$.

Set $w^0_i = \frac{\exp(-(A\lambda^0)_i)}{\sum_{l=1}^m \exp(-(A\lambda^0)_l)}$ for $i = 1, \ldots, m$. Set $k = 0$.

At iteration $k$:

1. Compute $j_k \in \arg\max_{j=1,\ldots,n} |((w^0)^TA)_j|$, $s = \text{sgn}((w^TA)_{j_k})$

2. Choose $\bar{\alpha} \in [0, 1]$ and set:
   
   $\lambda^{k+1}_{j_k} \leftarrow (1 - \bar{\alpha}_k)\lambda^k_{j_k} + \bar{\alpha}_k s \delta$
   
   $\lambda^{k+1}_j \leftarrow (1 - \bar{\alpha}_k)\lambda^k_j$ for $j \neq j_k$, and where $\bar{\alpha}_k \in [0, 1]$

   $w^{k+1}_j \leftarrow (w^k_j)^{1-\bar{\alpha}_k} \exp(-\bar{\alpha}_k s \delta A_{i,j_k})$, for $i = 1, \ldots, m$ and re-normalize $w^{k+1}$ so that $e^T w^{k+1} = 1$
Suppose that we use Frank-Wolfe method to solve the exponential loss minimization problem, with an intelligent step-size rule. Then after $k$ iterations:

- $L_{\text{exp}}(\lambda^k) - L_{\text{exp}}^* \leq \frac{8\delta^2}{k + 3}$
- $p^* - p(\bar{\lambda}^k) \leq \frac{8\delta}{k + 3} + \frac{\ln(m)}{\delta}$
- $\lambda^k$ has at most $k$ non-zero coefficients, and
- $\sum_j |\lambda_j| \leq \delta$

where $\bar{\lambda}^k$ is the normalization of $\lambda^k$, namely $\bar{\lambda}^k = \frac{\lambda^k}{e^T\lambda^k}$.
Remarks

- Optimization can inform and improve computational statistics
- Statistics can inform and improve optimization theory
- Statistics can inform and improve optimization practice