Announcement

- No recitations this week

- No lecture next week (Easter holiday)
Course organization

- **Retrieval**
  - Given a query, find “most similar” item in a large data set
  - *Applications*: GoogleGoggles, Shazam, ...

- **Supervised learning** (Classification, Regression)
  - Learn a concept (function mapping queries to labels)
  - *Applications*: Spam filtering, predicting price changes, ...

- **Unsupervised learning** (Clustering, dimension reduction)
  - Identify clusters, “common patterns”; anomaly detection
  - *Applications*: Recommender systems, fraud detection, ...

- **Learning with limited feedback**
  - Learn to optimize a function that’s expensive to evaluate
  - *Applications*: Online advertising, opt. UI, learning rankings, ...
Support Vector Machine

\[
\begin{align*}
\min_{w, b} & \quad w^T w \\
\text{s.t.} & \quad y_i (w^T x_i + b) \geq 1
\end{align*}
\]

- How can we solve this optimization?
- What about local minima?

- This is a convex (quadratic) program
Dealing with massive data sets

- Are we done??
- Complexity of quadratic programming
  - Naïve implementations: \( \mathcal{O}(m^3) \)

- What if the data doesn’t even fit in memory??

- Will see how one can reformulate the SVM optimization problem so that one can solve it on web-scale problems...
Online classification

- Data arrives sequentially
- Need to classify one data point at a time
- Use a different decision rule (lin. separator) each time
- Can’t remember all data points!

X: Classification error
Generally: Online convex programming

- Input:
  - Feasible set \( S \subseteq \mathbb{R}^d \)
  - Starting point \( w_0 \in S \)

- Each round \( t \) do
  - Pick new feasible point \( w_t \in S \)
  - Receive convex function \( f_t : S \rightarrow \mathbb{R} \)
  - Incur loss \( l_t = f_t(w_t) \)

- Regret:

\[
R_T = \sum_{t=1}^{T} l_t \quad \text{min}_{w \in S} \sum_{t=1}^{T} f_t(w)
\]

Solve: \( \min_{x \in S} \sum_{t=1}^{N} f_t(x) \) s.t. \( x \in S \)

E.g., SVM
Online convex programming

- Simple update rule:

\[ w_{t+1} = w_t - \eta_t \nabla f_t(w_t) \]

- How well does this simple algorithm do??
Regret for online convex programming

Theorem [Zinkevich ‘03]
Let \( f_1, \ldots, f_T \) be an arbitrary sequence of convex functions with feasible set \( S \)
Set \( \eta_t = 1/\sqrt{t} \)

Then, the regret of online convex programming is bounded by
\[
R_T \leq \frac{\|S\|^2\sqrt{T}}{2} + \left( \sqrt{T} - \frac{1}{2} \right) \|\nabla f\|^2
\]

Additional loss in accuracy due to online setting
\[
\frac{R_T}{T} = O\left( \frac{1}{\sqrt{T}} \right) = O\left( \frac{1}{\sqrt{T}} \right) \to 0
\]
OCP for SVM formulation

\[
\min_{w,b} \sum_{i=1}^{N} \max \left( 0, 1 - y_i (w^T x_i + b) \right)
\]

s.t. \(|w|_2 \leq \frac{1}{\lambda}
\]
Online convex programming for SVM

\[ w_{t+1} = \text{Proj}_S(w_t - \eta_t \nabla f_t(w_t)) \]

Feasible set: \[ S = \{ w : ||w|| \leq \frac{1}{\lambda} \} \]

Projection:

\[ \text{Proj}_S(w) = \begin{cases} w & \text{if } w \in S \\ \frac{w}{||w||} \cdot \frac{1}{\lambda} & \text{if } w \notin S \end{cases} \]

Gradient:

\[ f_t(w) = \max(0, 1 - y_t(w^T x_t)) \]
Subgradient for SVM

- Hinge loss: $f_t(w) = \max(0, 1 - y_t(w^T x_t + b))$

- Subgradient:

\[
\begin{align*}
\text{If} & \quad 1 - y_t(w^T x_t + b) < 0 \\
\text{If} & \quad 1 - y_t(w^T x_t + b) > 0 \\
\frac{\partial f_t(w)}{} & \quad 0 \\
\text{w} & \quad -y_t x_t
\end{align*}
\]

\[w_{t+1} = \text{Proj}_S \left( w_t - y_t \frac{\partial f_t(w)}{} \right)\]
Example [Bottou]

- Stochastic gradient descent
  - Online convex programming with training samples picked at random

- Data set:
  - Reuters RCV1
  - 780k training examples, 23k test examples
  - 50k dimensions

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMLight</td>
<td>23,642 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
<tr>
<td>SVMPerf</td>
<td>66 secs</td>
<td>0.2278</td>
<td>6.03%</td>
</tr>
<tr>
<td>SGD</td>
<td>1.4 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
</tbody>
</table>
Error

![Graph showing comparison between Testing cost and Training time for SGD and LibLinear methods. The x-axis represents Optimization accuracy (trainingCost-optimalTrainingCost), while the y-axis shows Training time (secs) and Testing cost. The graph illustrates the trade-off between training time and testing cost across different optimization accuracies.](image-url)
Subsampling

Average Test Loss

Time (seconds)

n=10000  n=100000  n=781265

n=30000  n=300000

stochastic
**State of the art: PEGASOS**

**INPUT:** training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$,  
Regularization parameter $\lambda$,  
Number of iterations $T$

**INITIALIZE:** Choose $w_1$ s.t. $\|w_1\| \leq 1/\sqrt{\lambda}$

**FOR** $t = 1, 2, \ldots, T$

Choose $A_t \subseteq S$

$A_t^+ = \{(x, y) \in A_t : y\langle w_t, x \rangle < 1\}$

$\nabla_t = \lambda w_t - \frac{\eta_t}{|A_t|} \sum_{(x, y) \in A_t^+} yx$

$\eta_t = \frac{1}{t\lambda}$

$w'_t = w_t - \eta_t \nabla_t$

$w_{t+1} = \min \left\{ 1, \frac{1/\sqrt{\lambda}}{\|w'_t\|} \right\} w'_t$

**OUTPUT:** $w_{T+1}$
Theorem [Shalev-Shwartz et al. ‘07]:

- Run-time required for Pegasos to find \( \varepsilon \)-accurate solution with probability at least 1-\( \delta \):
  \[
  O^* \left( \frac{d \log \frac{1}{\delta}}{\lambda \varepsilon} \right) = O\left( \frac{d}{\lambda \varepsilon} \right)
  \]

- Depends on
  - number of dimensions \( d \)
  - “difficulty” of problem (\( \lambda \) and \( \varepsilon \))

- Does not depend on #examples \( n \)
Difference between PEGASOS and standard OCP / SGD

- Uses batches of training examples
  - empirically more efficient
- Uses «strongly convex» loss functions
  - improved convergence rate, and better empirical performance
- Only guaranteed to work in the stochastic setting (i.e., can’t handle arbitrary ordering of data)
Dealing with massive data

- Online convex programming lets one train an SVM, processing one data point at a time
  - No need to store data in memory
  - Order doesn’t matter (for general OCP)!

- What about truly massive data?
  - Streaming 1 TB ~4-5 hours

- Can we do parallel processing in data centers?
  - Map reduce for SVM?
Parallel online learning

- Various different approaches [Zinkevich et al ’10]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Latency tolerance</th>
<th>MapReduce</th>
<th>Network IO</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed subgradient [3, 9]</td>
<td>moderate</td>
<td>yes</td>
<td>high</td>
<td>linear</td>
</tr>
<tr>
<td>Distributed convex solver [7]</td>
<td>high</td>
<td>yes</td>
<td>low</td>
<td>unclear</td>
</tr>
<tr>
<td>Multicore stochastic gradient [5]</td>
<td>low</td>
<td>no</td>
<td>n.a.</td>
<td>linear</td>
</tr>
<tr>
<td>Parallel stochastic gradient descent [Zinkevich ‘10]</td>
<td>high</td>
<td>yes</td>
<td>low</td>
<td>linear</td>
</tr>
</tbody>
</table>

- Still active area of research
Parallel stochastic gradient descent
[Zinkevich et al ‘10]

“Data parallel” method for solving

$$\min_w \lambda \|w\|^2 + \frac{1}{T} \sum_{t=1}^{T} f_t(w)$$

- Randomly partition data set to k machines
- Each machine runs SGD independently, produces $w_i$
- After T iterations, compute

$$w = \frac{1}{k} \sum_{i=1}^{k} w_i$$

- How well does this algorithm do?
- Does parallelism help?
Parallel stochastic gradient descent

[Zinkevich et al ‘10]

\[ w = \frac{1}{k} \sum_{i=1}^{k} w_i \]
Parallel stochastic gradient descent

[Zinkevich et al ‘10]

**Theorem:** Suppose each of the $k$ machines runs for

$$T = \Omega \left( \log \frac{k\lambda}{\varepsilon} \right)$$

Then:

$$\mathbb{E}[\text{error}] \leq \mathcal{O} \left( \varepsilon \left( \frac{1}{\sqrt{k\lambda}} + 1 \right) \right)$$

Parallelization helps, but only if

$$k = \mathcal{O} \left( \frac{1}{\lambda} \right)$$

The “more difficult” the learning problem (the smaller $\lambda$), the more parallelization helps!
Performance of parallel online SGD

[Zinkevich et al ‘10]
Summary so far

- **Support Vector Machines**
  - State of the art linear classifier
  - Requires solving convex program

- **Online convex programming**
  - Simple, online algorithm for approximately minimizing additive loss functions
  - Only require (sub-)gradients and reprojection

- **Stochastic gradient descent**
  - Online convex programming in random order

- **Parallelized stochastic gradient descent**
  - Compute gradients independently, then average
  - Amount of effective parallelism depends on “hardness” of problem
More results on supervised learning

- Feature selection
- Dealing with multiple classes
- Linear regression
- Nonlinear classification / regression
In many high-dimensional problems, we may prefer “sparse” solutions:  
\[
\text{sign}(w^T x + b)
\]

where \(w\) contains only few nonzero entries.

Reasons:
- **Interpretability** (would like to “understand” the classifier, identify important variables)
- **Generalization** (simpler models may generalize better)
- **Storage / computation** (don’t need to store / sum data for 0 coefficients...)

27
Feature selection

Suppose we would like to identify top $k$ features

Approach 1
- Try out all sets of at most $k$ variables
- Fit a classifier to each set, ignoring the non-selected variables
- Pick the best set
- Problem?

Approach 2
- Greedily select the features: Add one at a time to maximize improvement in accuracy
- Problem?

Ideally: Solve classification and feature selection in one fell-swoop!
Sparsity enforcing regularizers

Before:
- Support vector machine

\[
\min_{w,b} \lambda \|w\|_2^2 + \sum_i \max(0, 1 - y_i(w^T x_i + b))
\]

- Uses \(\|w\|_2\) to control the weights

Slight modification: replace \(\|w\|_2\) by \(\|w\|_1 = \sum_{i=1}^D |w_i|\)
- L1-SVM

\[
\min_{w,b} \lambda \|w\|_1 + \sum_i \max(0, 1 - y_i(w^T x_i + b))
\]

This alternative penalty encourages coefficients to be exactly 0 \(\Rightarrow\) ignores those features!
Feature selection with L1-SVM

[Zhu et al NIPS ‘03]
Illustration of l1-regularization

\[
\min ||w||_2
\]

s.t. \( y_i: w^T x_i > 1 \)
Data:
- 38 train, 34 test data from a DNA microarray classification experiment (leukemia diagnosis)
- 7129 dimensions

![Table]

<table>
<thead>
<tr>
<th>Method</th>
<th>CV Error</th>
<th>Test Error</th>
<th># of Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-norm SVM UR</td>
<td>2/38</td>
<td>3/34</td>
<td>22</td>
</tr>
<tr>
<td>2-norm SVM RFE</td>
<td>2/38</td>
<td>1/34</td>
<td>31</td>
</tr>
<tr>
<td>1-norm SVM</td>
<td>2/38</td>
<td>2/34</td>
<td>17</td>
</tr>
</tbody>
</table>

[Zhu et al NIPS ‘03]
Online L1-SVM

- Can solve L1-SVM using online convex programming
  \[
  \min_{w,b} \sum_{i} \max(0, 1 - y_i(w^T x_i + b)) \quad \text{s.t. } ||w||_1 \leq \frac{1}{\lambda}
  \]

- Subgradient:
  - calculation stays the same as in SVM!

- Reprojection:
  - Need to solve:
    \[
    \text{Proj}_{S}(w) = \arg\min_{w' \in S} ||w - w'||_2
    \]
More results on supervised learning

- Feature selection
- **Dealing with multiple classes**
- Regression
- Nonlinear methods
Dealing with multiple classes
One-vs-all

\[ y_e \cdot w^T x_i \rightarrow b_e \]

- Solve $c$ SVMs, one for each class
  - Positive examples: all points from class $I$
  - Negative examples: all other points
- Classify using the SVM with largest margin
- Problems?
- Ideally want to optimize all SVMs at the same time
**Multi-class SVM**

\[
\min_{w, b, \xi} \sum_y w^T(y) w(y) + C \sum_i \xi_i \\
\text{s.t. } w^T(y_i) x_i + b(y_i) \geq w^T(y') x_i + b(y') + 1 - \xi_j
\]

- Can be solved using same techniques as single-class SVM
- Multi-class hinge loss:

\[
\ell(W; (x, y)) = \max_{r \in [k] \setminus \{y\}} \frac{1 - (W x)_y + (W x)_r}{\sigma} \max \left( 1 - \frac{\sigma}{2}, 0 \right)
\]
More results on supervised learning

- Feature selection
- Dealing with multiple classes
- Regression
- Nonlinear methods
Regression

- So far, our goal was to predict a discrete label.
- In many problems, we need to predict a real-valued output.

\[ y = f(x; w) + noise \]

- E.g.:
  - Predict grade based on #homeworks solved.
  - Predict flight delay at one airport given delays at other airports.
  - ...
Given \( (x_1, y_1), \ldots, (x_n, y_n) \)

Assume: \( y_i = w^T x_i + noise \)

To optimize \( w \) need to quantify goodness of fit
Square loss

- Want to solve

\[ w^* = \arg \min_w \sum_{i=1}^{n} (y_i - w^T x_i)^2 \]

- Closed form solution:

\[ w^* = (X^T X)^{-1} X^T y \]

- Complexity? \[ O(D^3) \]

- Intractable for large \# of dimensions!

- Will see how we can efficiently compute with OCP!