Large-Scale Machine Learning (2)

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
Supervised Learning

- Would like to do prediction: estimate a function \( f(x) \) so that \( y = f(x) \)

- Where \( y \) can be:
  - 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: \{+1, -1\}, 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 generalizes well to unseen data
1) Training data is drawn independently at random according to unknown probability distribution $P(x, y)$

2) The learning algorithm analyzes the examples and produces a classifier $f$

Given **new** data $(x, y)$ drawn from $P$, the classifier is given $x$ and predicts $\hat{y} = f(x)$

The loss $\mathcal{L}(\hat{y}, y)$ is then measured

**Goal of the learning algorithm:**

Find $f$ that minimizes expected loss $E_P[\mathcal{L}]$
Why is it hard?

We estimate $f$ on training data but want the $f$ to work well on unseen future (i.e., test) data.
Minimizing the Loss

- **Goal:** Minimize the expected loss
  \[
  \min_f \mathbb{E}_P[\mathcal{L}]
  \]

- But, we don’t have access to \(P\) but only to the training sample \(D\):
  \[
  \min_f \mathbb{E}_D[\mathcal{L}]
  \]

- So, we minimize the average loss on the training data:
  \[
  \min_f J(f) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)
  \]

**Problem:** Just memorizing the training data gives us a perfect model (with zero loss)
Given:
- A set of $N$ training examples
  - $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- A loss function $\mathcal{L}$

Choose the model: $f_w(x) = w \cdot x + b$

Find:
- The weight vector $w$ that minimizes the expected loss on the training data

$$J(f) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(w \cdot x_i + b, y_i)$$
Problem: Step-wise Constant Loss function

Derivative is either 0 or $\infty$
Approximating the Loss

- Approximating the expected loss by a smooth function
  - Replace the original objective function by a surrogate loss function. E.g., hinge loss:

\[ \tilde{J}(w) = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y^{(i)}f(x^{(i)})) \]

When \( y = 1 \):
Support Vector Machines
Want to separate “+” from “-” using a line

Data:
- Training examples:
  - \((x_1, y_1) \ldots (x_n, y_n)\)
- Each example \(i\):
  - \(x_i = (x_i^{(1)}, \ldots, x_i^{(d)})\)
    - \(x_i^{(j)}\) is real valued
  - \(y_i \in \{-1, +1\}\)
- Inner product:
  - \(w \cdot x = \sum_{j=1}^{d} w^{(j)} \cdot x^{(j)}\)

Which is best linear separator (defined by \(w, b\))?
Largest Margin

- Distance from the separating hyperplane corresponds to the "confidence" of prediction
- Example:
  - We are more sure about the class of A and B than of C
Largest Margin

- Margin $\gamma$: Distance of closest example from the decision line/hyperplane

The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.
Why maximizing $\gamma$ a good idea?

- Remember: The Dot product

\[ A \cdot B = ||A|| \cdot ||B|| \cdot \cos \theta \]
Why maximizing $\gamma$ a good idea?

- **Dot product**
  \[ A \cdot B = \|A\| \|B\| \cos \theta \]

- What is $w \cdot x_1$, $w \cdot x_2$?

  - In this case $\gamma_1 \approx \|w\|^2$
  - In this case $\gamma_2 \approx 2\|w\|^2$

- **So, $\gamma$ roughly corresponds to the margin**
  - **Bottom line:** Bigger $\gamma$, bigger the separation
What is the margin?

Let:
- Line $L$: $\mathbf{w} \cdot \mathbf{x} + b = w^{(1)} x^{(1)} + w^{(2)} x^{(2)} + b = 0$
- $\mathbf{w} = (w^{(1)}, w^{(2)})$
- Point $A = (x_A^{(1)}, x_A^{(2)})$
- Point $M$ on a line $= (x_M^{(1)}, x_M^{(2)})$

$$d(A, \ L) = |AH| = |(A-M) \cdot \mathbf{w}| = |(x_A^{(1)} - x_M^{(1)}) w^{(1)} + (x_A^{(2)} - x_M^{(2)}) w^{(2)}| = |x_A^{(1)} w^{(1)} + x_A^{(2)} w^{(2)} + b| = |\mathbf{w} \cdot \mathbf{A} + b|$$

Remember $x_M^{(1)} w^{(1)} + x_M^{(2)} w^{(2)} = -b$ since $M$ belongs to line $L$

Note we assume $\|\mathbf{w}\|_2 = 1$
Prediction = \text{sign}(w \cdot x + b)

“Confidence” = (w \cdot x + b) y

For i-th datapoint:
\[ \gamma_i = (w \cdot x_i + b)y_i \]

Want to solve:
\[ \max_{w,b} \min_i \gamma_i \]

Can rewrite as
\[ \max_{w,\gamma} \gamma \]
\[ \text{s.t.} \forall i, y_i (w \cdot x_i + b) \geq \gamma \]
Maximize the margin:

- Good according to intuition, theory (c.f. “VC dimension”) and practice

\[
\max_{\gamma, w} \gamma \\
\text{s.t.} \forall i, y_i(w \cdot x_i + b) \geq \gamma
\]

- \(\gamma\) is margin ... distance from the separating hyperplane
Support Vector Machines: Deriving the margin
Separating hyperplane is defined by the support vectors

- Points on +/- planes from the solution
- If you knew these points, you could ignore the rest
- Generally, \(d+1\) support vectors (for \(d\) dim. data)
**Canonical Hyperplane: Problem**

- **Problem:**
  - Let \((\mathbf{w} \cdot \mathbf{x} + b)y = \gamma\)
  - then \((2\mathbf{w} \cdot \mathbf{x} + 2b)y = 2\gamma\)
  - Scaling \(\mathbf{w}\) increases margin!

- **Solution:**
  - Work with normalized \(\mathbf{w}\):
    \[
    \gamma = \left( \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x} + b \right)y
    \]
  - Let’s also require **support vectors** \(\mathbf{x}_j\)
    to be on the plane defined by:
    \[
    \mathbf{w} \cdot \mathbf{x}_j + b = \pm 1
    \]
    \[
    ||\mathbf{w}|| = \sqrt{\sum_{j=1}^{d} (w^{(j)})^2}
    \]
Want to maximize margin!

What is the relation between $x_1$ and $x_2$?

$\begin{align*}
x_1 &= x_2 + 2\gamma \frac{w}{||w||} \\
\text{We also know:} & \\
& \quad w \cdot x_1 + b = +1 \\
& \quad w \cdot x_2 + b = -1 \\
\text{So:} & \\
& \quad w \cdot x_1 + b = +1 \\
& \quad w \left(x_2 + 2\gamma \frac{w}{||w||}\right) + b = +1 \\
& \quad w \cdot x_2 + b + 2\gamma \frac{w \cdot w}{||w||} = +1
\end{align*}$

$\Rightarrow \gamma = \frac{||w||}{w \cdot w} = \frac{1}{||w||}$

Note: $w \cdot w = ||w||^2$
Maximizing the Margin

- We started with
  \[ \max_{w, \gamma} \gamma \]
  \[ \text{s.t.} \forall i, y_i (w \cdot x_i + b) \geq \gamma \]
  But \( w \) can be arbitrarily large!

- We normalized and...
  \[ \arg \max \gamma = \arg \max \frac{1}{\|w\|} = \arg \min \|w\| = \arg \min \frac{1}{2} \|w\|^2 \]

- Then:
  \[ \min_{w} \frac{1}{2} \| w \|^2 \]
  \[ \text{s.t.} \forall i, y_i (w \cdot x_i + b) \geq 1 \]

This is called SVM with “hard” constraints
Non-linearly Separable Data

- If data is **not separable** introduce **penalty**:

\[
\min_w \frac{1}{2} \|w\|^2 + C \cdot (\text{# number of mistakes})
\]

\[
s.t. \forall i, y_i (w \cdot x_i + b) \geq 1
\]

- Minimize \( \|w\|^2 \) plus the number of training mistakes
- Set \( C \) using cross validation

- **How to penalize mistakes?**
  - All mistakes are not equally bad!
Support Vector Machines

- **Introduce slack variables** $\xi_i$

$$\min_{w,b,\xi \geq 0} \frac{1}{2} \| w \|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1 - \xi_i$$

- **If point** $x_i$ **is on the wrong side of the margin then get penalty** $\xi_i$

For each data point:
If margin $\geq 1$, don’t care
If margin $< 1$, pay linear penalty
Slack Penalty $C$

$$\min_w \frac{1}{2} \|w\|^2 + C \cdot (\# \text{number of mistakes})$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq 1$$

- **What is the role of slack penalty $C$:**
  - $C=\infty$: Only want to $w, b$ that separate the data
  - $C=0$: Can set $\xi_i$ to anything, then $w=0$ (basically ignores the data)
Support Vector Machines

- **SVM in the “natural” form**
  \[
  \arg \min_{w,b} \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \max\{0, 1 - y_i (w \cdot x_i + b)\}
  \]

  - Margin
  - Empirical loss \(L\) (how well we fit training data)
  - Regularization parameter

- **SVM uses “Hinge Loss”:**
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i
  \]
  \[
  s.t. \forall i, y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i
  \]

  - 0/1 loss
  - 0/1 loss
  - Hinge loss: \(\max\{0, 1-z\}\)
  - \(z = y_i \cdot (x_i \cdot w + b)\)
Support Vector Machines: How to compute the margin?
SVM: How to estimate $w$?

$$\min_{w,b} \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i$$

$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$

- **Want to estimate $w$ and $b$!**
  - **Standard way:** Use a solver!
    - **Solver:** software for finding solutions to “common” optimization problems
  
- **Use a quadratic solver:**
  - Minimize quadratic function
  - Subject to linear constraints

- **Problem:** Solvers are inefficient for big data!
**SVM: How to estimate \( w \)?**

- **Want to minimize** \( J(w,b) \):
  \[
  J(w,b) = \frac{1}{2} \sum_{j=1}^{d} (w^{(j)})^2 + C \sum_{i=1}^{n} \max \left\{ 0, 1 - y_i \left( \sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b \right) \right\}
  \]

- **Compute the gradient** \( \nabla(j) \) **w.r.t.** \( w^{(j)} \)
  \[
  \nabla J^{(j)} = \frac{\partial J(w, b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}
  \]
  \[
  \frac{\partial L(x_i, y_i)}{\partial w^{(j)}} = 0 \quad \text{if} \quad y_i (w \cdot x_i + b) \geq 1
  \]
  \[
  = -y_i x_i^{(j)} \quad \text{else}
  \]

Empirical loss \( L(x_i, y_i) \)
Gradient descent:

Iterate until convergence:
- For \( j = 1 \ldots d \)
  - Evaluate: \( \nabla J^{(j)} = \frac{\partial f(w, b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}} \)
  - Update: \( w'(j) \leftarrow w(j) - \eta \nabla J^{(j)} \)
- \( w \leftarrow w' \)

Problem:
- Computing \( \nabla J^{(j)} \) takes \( O(n) \) time!
  - \( n \) ... size of the training dataset

\( \eta \ldots \text{learning rate parameter} \)
\( C \ldots \text{regularization parameter} \)
SVM: How to estimate \( w \)?

- **Stochastic Gradient Descent**
  - Instead of evaluating gradient over all examples, evaluate it for each **individual** training example

\[
\nabla J^{(j)}(x_i) = w^{(j)} + C \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}
\]

- **Stochastic gradient descent:**

Iterate until convergence:
- For \( i = 1 \ldots n \)
  - For \( j = 1 \ldots d \)
    - Compute: \( \nabla J^{(j)}(x_i) \)
    - Update: \( w^{(j)} \leftarrow w^{(j)} - \eta \nabla J^{(j)}(x_i) \)

We just had:

\[
\nabla J^{(j)} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}
\]

Notice: no summation over \( i \) anymore
Other variations of GD

- **Batch Gradient Descent**
  - Calculates error for each example in the training dataset, but updated model **only after** all examples have been evaluated (i.e., end of training epoch)
  - **PROS**: fewer updates, more stable error gradient
  - **CONS**: usually requires whole dataset in memory, slower than SGD

- **Mini-Batch Gradient Descent**
  - Like BGD, but using smaller batches of training data. Balance between robustness of SGD, and efficiency of BGD.
Support Vector Machines: Example
Example: Text categorization

- **Dataset:**
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One vs. the rest classification
  - \( n = 781,000 \) training examples (documents)
  - 23,000 test examples
  - \( d = 50,000 \) features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words
Example: Text categorization

Questions:

1. Is SGD successful at minimizing $J(w, b)$?
2. How quickly does SGD find the min of $J(w, b)$?
3. What is the error on a test set?

<table>
<thead>
<tr>
<th></th>
<th>Training time</th>
<th>Value of $J(w, b)$</th>
<th>Test error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard SVM</td>
<td>23,642 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
<tr>
<td>“Fast SVM”</td>
<td>66 secs</td>
<td>0.2278</td>
<td>6.03%</td>
</tr>
<tr>
<td>SGD-SVM</td>
<td>1.4 secs</td>
<td>0.2275</td>
<td>6.02%</td>
</tr>
</tbody>
</table>

(1) SGD-SVM is successful at minimizing the value of $J(w, b)$
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable
Optimization quality: $| J(w,b) - J(w^{opt}, b^{opt}) |$

For optimizing $J(w,b)$ within reasonable quality

SGD-SVM is super fast
Practical Considerations

- Need to choose learning rate $\eta$ and $t_0$

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t + t_0} \left( w_t + C \frac{\partial L(x_i, y_i)}{\partial w} \right)$$

- Tricks:
  - Choose $t_0$ so that the expected initial updates are comparable with the expected size of the weights
  - Choose $\eta$:
    - Select a small subsample
    - Try various rates $\eta$ (e.g., 10, 1, 0.1, 0.01, ...)
    - Pick the one that most reduces the cost
    - Use $\eta$ for next 100k iterations on the full dataset
What about multiple classes?

- **Idea 1:**
  - One against all
  - Learn 3 classifiers
    - + vs. \{o, -\}
    - - vs. \{o, +\}
    - o vs. \{+,-\}
  - Obtain:
    \[ w_+ b_+, \ w_- b_-, \ w_o b_o \]
  - **How to classify?**
  - Return class \( c \)
    \[ \text{arg max}_c \ w_c x + b_c \]
Idea 2: Learn 3 sets of weights simultaneously!

- For each class $c$ estimate $w_c$, $b_c$
- Want the correct class $y_i$ to have highest margin:

$$w_{y_i} x_i + b_{y_i} \geq 1 + w_c x_i + b_c \quad \forall c \neq y_i \quad \forall i$$
Optimization problem:

\[
\min_{w,b} \frac{1}{2} \sum_c \|w_c\|^2 + C \sum_{i=1}^{n} \xi_i
\]

\[
w_{y_i} \cdot x_i + b_{y_i} \geq w_c \cdot x_i + b_c + 1 - \xi_i \quad \forall c \neq y_i, \forall i
\]

\[
\xi_i \geq 0, \forall i
\]

To obtain parameters \(w_c, b_c\) (for each class \(c\))
we can use similar techniques as for 2 class SVM

SVM is widely perceived a very powerful
learning algorithm
ML Parallelization
Why Large-Scale ML?

- **The Unreasonable Effectiveness of Data**
  - In 2017, Google revisited a 15-year-old experiment on the effect of data and model size in ML, focusing on 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!!**

Recap

- Last lecture: Decision Trees (and PLANET) as a prime example of **Data Parallelism** in ML

- Today’s lecture: Multiclass SVMs, Neural Networks (especially Deep ones), etc. can leverage both **Data Parallelism and Model Parallelism**
  - State-of-the-art Deep Neural Networks for visual recognition tasks (e.g., ImageNet challenge) can have **more than 100 million parameters**!
M2 and M4 must wait for the 1\textsuperscript{st} stage to complete!
Parallelization overview

- Unsupervised or Supervised Objective
- Minibatch Stochastic Gradient Descent (SGD)
- Model parameters sharded by partition
- 10s, 100s, or 1000s of cores per model
Parameter Server

\[ p' = p + \Delta p \]

Parameter Server: Key/Value store
- Keys index the model parameters (e.g., weights)
- Values are the parameters of the ML model (e.g., a neural network)

Systems challenges:
- High bandwidth
- Synchronization
- Fault tolerance
Parameter Server

Parameter Server \( p' = p + \Delta p \)

Why do parallel updates work?

\[ \Delta p \]

Model
Workers

Data
Shards

Async SGD

- **Key idea**: don’t synchronize, just **overwrite** parameters opportunistically from multiple workers (i.e., servers)
  - Same implementation as SGD, **just without locking**!

- **In theory**, Async SGD converges, but a slower rate than the serial version.
- **In practice**, when gradient updates are sparse (i.e., high dimensional data), **same convergence**!

- Recht et al. “**HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent**”, 2011

RR is a super optimized version of online Gradient Descent
HOGWILD!

1. Initialize \( w \) in shared memory // in parallel, do
2. for \( i = \{1, \ldots, p\} \) do
3. while TRUE do
4. if stopping criterion met then
5. | break
6. end
7. Sample \( j \) from \( 1, \ldots, n \) uniformly at random.
8. Compute \( f_j(w) \) and \( \nabla f_j(w) \) using whatever \( w \) is currently available.
9. Let \( e_j \) denote non-zero indices of \( x_i \)
10. for \( k \in e_j \) do
11. | \( w(k) \leftarrow w(k) - \alpha [\nabla F_j(w)](k) \)
12. end
13. end
14. end

\( \leq P \) is the number of partitions / processors

Scott Gradient Descent (SGD)

Component-wise gradient updates (relies on sparsity)
Asynchronous Distributed SGD


- All ingredients together:
  - Model and Data parallelism
  - Async SGD

- Dawn of modern Deep Learning

From an engineering standpoint, this is much better than a single model with the same number of total machines:

- Synchronization boundaries involve fewer machines
- Better robustness to individual slow machines
- Makes forward progress even during evictions/restarts