Large Scale Machine Learning: SVMs
New Topic: Graph Data!

High dim. data
- Locality sensitive hashing
- Clustering
- Dimensionality reduction

Graph data
- PageRank, SimRank
- Community Detection
- Spam Detection

Infinite data
- Filtering data streams
- Web advertising
- Queries on streams

Machine learning
- SVM
- Decision Trees
- Perceptron, kNN

Apps
- Recommender systems
- Association Rules
- Duplicate document detection
Supervised Learning

- **Example: Spam filtering**

<table>
<thead>
<tr>
<th>viagra</th>
<th>learning</th>
<th>the</th>
<th>dating</th>
<th>nigeria</th>
<th>spam?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{x}_1 = (1, 0, 1, 0, 0)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$y_1 = 1$</td>
</tr>
<tr>
<td>$\vec{x}_2 = (0, 1, 1, 0, 0)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$y_2 = -1$</td>
</tr>
<tr>
<td>$\vec{x}_3 = (0, 0, 0, 0, 1)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$y_3 = 1$</td>
</tr>
</tbody>
</table>

- **Instance space** $x \in X$ ($|X| = n$ data points)
  - Binary or real-valued feature vector $x$ of word occurrences
  - $d$ features (words + other things, $d \sim 100,000$)

- **Class** $y \in Y$
  - $y$: Spam (+1), Ham (-1)

- **Goal:** Estimate a function $f(x)$ so that $y = f(x)$
More generally: 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
    - But Jure, how can we well on data we have never seen before?!?
Idea: Pretend we do not know the data/labels we actually do know

- Build the model $f(x)$ on the training data
- See how well $f(x)$ does on the test data
  - If it does well, then apply it also to $X'$

Refinement: Cross validation

- Splitting into training/validation set is brutal
- Let’s split our data $(X,Y)$ into 10-folds (buckets)
- Take out 1-fold for validation, train on remaining 9
- Repeat this 10 times, report average performance
Binary classification:

\[ f(x) = \begin{cases} +1 & \text{if } w^{(1)} x^{(1)} + w^{(2)} x^{(2)} + \ldots + w^{(d)} x^{(d)} \geq \theta \\ -1 & \text{otherwise} \end{cases} \]

**Input:** Vectors \( x_j \) and labels \( y_j \)
- Vectors \( x_j \) are real valued where \( \|x\|_2 = 1 \)

**Goal:** Find vector \( w = (w^{(1)}, w^{(2)}, \ldots, w^{(d)}) \)
- Each \( w^{(i)} \) is a real number

\[ w \cdot x = 0 \]

**Note:**
- Decision boundary is linear

\[ x \rightarrow \langle x, 1 \rangle \quad \forall x \]
\[ w \rightarrow \langle w, -\theta \rangle \]
(Very) loose motivation: Neuron
Inputs are feature values
Each feature has a weight $w_i$
Activation is the sum:
\[ f(x) = \sum_{i}^{d} w^{(i)} x^{(i)} = w \cdot x \]
If the $f(x)$ is:
- **Positive**: Predict $+1$
- **Negative**: Predict $-1$
**Perceptron**

- **Perceptron:** \( y' = \text{sign}(w \cdot x) \)
- **How to find parameters \( w \)?**
  - Start with \( w_0 = 0 \)
  - Pick training examples \( x_t \) **one by one**
  - Predict class of \( x_t \) using current \( w_t \)
    - \( y' = \text{sign}(w_t \cdot x_t) \)
  - **If \( y' \) is correct** (i.e., \( y_t = y' \))
    - No change: \( w_{t+1} = w_t \)
  - **If \( y' \) is wrong:** Adjust \( w_t \)
    \[
    w_{t+1} = w_t + \eta \cdot y_t \cdot x_t
    \]
    - \( \eta \) is the learning rate parameter
    - \( x_t \) is the t-th training example
    - \( y_t \) is true t-th class label (\(+1, -1\))

Note that the Perceptron is a conservative algorithm: it ignores samples that it classifies correctly.
**Good: Perceptron convergence theorem:**
- If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge

**Bad: Never converges:**
- If the data is not separable weights dance around indefinitely

**Bad: Mediocre generalization:**
- Finds a “barely” separating solution
Updating the Learning Rate

- Perceptron will oscillate and won’t converge
- So, when to stop learning?
  1. Slowly decrease the learning rate $\eta$
     - A classic way is to: $\eta = \frac{c_1}{(t + c_2)}$
     - But, we also need to determine constants $c_1$ and $c_2$
  2. Stop when the training error stops chaining
  3. Have a small test dataset and stop when the test set error stops decreasing
  4. Stop when we reached some maximum number of passes over the data
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 \(w\))?
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.
- **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: Dot product**

$$A \cdot B = \|A\| \cdot \|B\| \cdot \cos \theta$$

![Diagram](image)

$$\|A\| = \sqrt{\sum_{j=1}^{d} (A(j))^2}$$
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**
  - Bigger \( \gamma \) bigger the separation
Distance from a point to a line

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

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

Remember \( x_M^{(1)}w^{(1)} + x_M^{(2)}w^{(2)} = -b \) since \( M \) belongs to line \( L \)
Prediction = 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:
\[\max \min \gamma_i\]

Can rewrite as
\[\max_{w, \gamma} \gamma\]

s.t. \(\forall i, y_i (w \cdot x_i + b) \geq \gamma\)
**Maximize the margin:**

- Good according to intuition, theory (VC dimension) & practice

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

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

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

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

- $x_1 = x_2 + 2\gamma \frac{w}{||w||}$

- We also know:
  - $w \cdot x_1 + b = +1$
  - $w \cdot x_2 + b = -1$

So:

- $w \cdot x_1 + b = +1$
- $w \left(x_2 + 2\gamma \frac{w}{||w||}\right) + b = +1$
- $w \cdot x_2 + b + 2\gamma \frac{w \cdot w}{||w||} = +1$

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

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 $\xi_i$

$$
\min_{w, b, \xi_i \geq 0} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^{n} \xi_i \\
\text{s.t. } \forall i, y_i (w \cdot x_i + b) \geq 1 - \xi_i
$$

- 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

\[ \arg \min_{w,b} \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \max \{ 0, 1 - y_i (w \cdot x_i + b) \} \]

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

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 $w$?

\[
\min_{w,b} \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i \\
\text{s.t. } \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i
\]

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

- Side note:
  - How to minimize convex functions \( g(z) \)?
  - Use gradient descent: \( \min_z g(z) \)
  - Iterate: \( z_{t+1} \leftarrow z_t - \eta \nabla g(z_t) \)
SVM: How to estimate $w$?

- Want to minimize $f(w, b)$:

$$f(w, b) = \frac{1}{2} \sum_{j=1}^{d} (w^{(j)})^2 + C \sum_{i=1}^{n} \max \left\{ 0, 1 - y_i \left( \sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b \right) \right\}$$

Empirical loss $L(x_i, y_i)$

- Compute the gradient $\nabla^{(j)}$ w.r.t. $w^{(j)}$

$$\nabla f^{(j)} = \frac{\partial f(w, b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i, y_i)}{\partial w^{(j)}} = 0 \quad \text{if } y_i (w \cdot x_i + b) \geq 1$$

$$= -y_i x_i^{(j)} \quad \text{else}$$
Gradient descent:

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

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

\( \eta \) … learning rate parameter
\( C \) … regularization parameter
SVM: How to estimate $w$?

- **Stochastic Gradient Descent**
  - Instead of evaluating gradient over all examples, evaluate it for each individual training example.
  
  $\nabla f^{(j)}(x_i) = w^{(j)} + C \cdot \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$

- **Stochastic gradient descent:**

  Iterate until convergence:
  - For $i = 1 \ldots n$
    - For $j = 1 \ldots d$
      - Compute: $\nabla f^{(j)}(x_i)$
      - Update: $w^{(j)} \leftarrow w^{(j)} - \eta \ \nabla f^{(j)}(x_i)$
Support Vector Machines: Example
Example: Text categorization

- **Example by Leon Bottou:**
  - **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
Questions:

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

<table>
<thead>
<tr>
<th></th>
<th>Training time</th>
<th>Value of $f(w,b)$</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 $f(w,b)$
2. SGD-SVM is super fast
3. SGD-SVM test set error is comparable
Optimization “Accuracy”

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

SGD-SVM is super fast
SGD vs. Batch Conjugate Gradient

- **SGD** on full dataset vs. **Conjugate Gradient** on a sample of \( n \) training examples

**Bottom line:** Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) CG update a few times.

Theory says: Gradient descent converges in linear time \( k \). Conjugate gradient converges in \( \sqrt{k} \).

\( k \) ... condition number
Sparse Linear SVM:

- Feature vector \( x_i \) is sparse (contains many zeros)
  - Do not do: \( x_i = [0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,0,...] \)
  - But represent \( x_i \) as a sparse vector \( x_i = [(4,1), (9,5), ...] \)

- Can we do the SGD update more efficiently?

\[
w \leftarrow w - \eta \left( w + C \frac{\partial L(x_i, y_i)}{\partial w} \right)
\]

- Approximated in 2 steps:

\[
w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w} \quad \text{cheap: } x_i \text{ is sparse and so few coordinates } j \text{ of } w \text{ will be updated}
\]

\[
w \leftarrow w(1 - \eta) \quad \text{expensive: } w \text{ is not sparse, all coordinates need to be updated}
\]
Solution 1: \( \mathbf{w} = \mathbf{s} \cdot \mathbf{v} \)

- Represent vector \( \mathbf{w} \) as the product of scalar \( \mathbf{s} \) and vector \( \mathbf{v} \)
- Then the update procedure is:
  - (1) \( \mathbf{v} = \mathbf{v} - \eta \mathbf{C} \frac{\partial L(x_i, y_i)}{\partial \mathbf{w}} \)
  - (2) \( \mathbf{s} = \mathbf{s}(1 - \eta) \)

Solution 2:

- Perform only step (1) for each training example
- Perform step (2) with lower frequency and higher \( \eta \)
Practical Considerations

- **Stopping criteria:**
  - How many iterations of SGD?
    - **Early stopping with cross validation**
      - Create a validation set
      - Monitor cost function on the validation set
      - Stop when loss stops decreasing
    - **Early stopping**
      - Extract two disjoint subsamples $A$ and $B$ of training data
      - Train on $A$, stop by validating on $B$
      - Number of epochs is an estimate of $k$
      - Train for $k$ epochs on the full dataset
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 \( c \)
\[ \arg \max_c w_c x + b_c \]
Idea 2: Learn 3 sets of weights simultaneously!

- For each class $c$, estimate $w_c$, $b_c$
- Want the correct class to have highest margin:
  
  \[ w_{y_i}x_i + b_{y_i} \geq 1 + w_c x_i + b_c \quad \forall c \neq y_i , \forall i \]
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