Large Scale Machine Learning: SVMs
New Topic: ML!

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
Given some data:
- “Learn” a function to map from the input to the output
- **Given:**
  - Training examples \( (x_i, y = f(x_i)) \) for some unknown function \( f \)
- **Find:**
  - A good approximation to \( f \)
Many Other ML Paradigms

- **Supervised:**
  - Given “labeled data” \{x, y\}, learn \( f(x) = y \)
- **Unsupervised:**
  - Given only “unlabeled data” \{x\}, learn \( f(x) \)
- **Semi-supervised:**
  - Given some labeled and some unlabeled data
- **Active learning:**
  - Whenever we predict \( f(x) = y \), we then receive true \( y^* \)
- **Transfer learning:**
  - Learn \( f(x) \) so that it works well on new domain \( f(z) \)
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
**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 $L(\hat{y}, y)$ is then measured

**Goal of the learning algorithm:**
Find $f$ that minimizes expected loss $E_P[L]$
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 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)
ML == Optimization

- **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:** \( f_w(x) = w \cdot x + b \)
- **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:

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

  When \( y = 1 \):
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>$\bar{x}_1$ = (1, 0, 1, 0, 0, 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$y_1 = 1$</td>
</tr>
<tr>
<td>$\bar{x}_2$ = (0, 1, 1, 0, 0, 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$y_2 = -1$</td>
</tr>
<tr>
<td>$\bar{x}_3$ = (0, 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)
Spam Detection

- $P(x, y)$: distribution of email messages $x$ and their true labels $y$ (“spam”, “ham”)
- Training sample: a set of email messages that have been labeled by the user
- Learning algorithm: What we study!
- $f$: The classifier output by the learning alg.
- Test point: A new email $x$ (with its true, but hidden, label $y$)
- Loss function $\mathcal{L}(\hat{y}, y)$:

<table>
<thead>
<tr>
<th>predicted label $\hat{y}$</th>
<th>true label $y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam</td>
<td>0</td>
</tr>
<tr>
<td>not spam</td>
<td>1</td>
</tr>
</tbody>
</table>
We will talk about the following methods:

- Support Vector Machines
- Decision trees

Main question:
How to efficiently train (build a model/find model parameters)?
Support Vector Machines
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 \(w, b)\)?
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
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

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

![Diagram of vectors A and B with angle $\theta$ and vector $||A|| \cos \theta$]
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
Let:

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

Distance from a point to a line
\[
M(1) = (x_{M}^{(1)}, x_{M}^{(2)})
\]
\[
d(A, L) = |AH|
\]
\[
= |(A-M) \cdot w|
\]
\[
= |(x_{A}^{(1)} - x_{M}^{(1)}) w^{(1)} + (x_{A}^{(2)} - x_{M}^{(2)}) w^{(2)}|
\]
\[
= |x_{A}^{(1)} w^{(1)} + x_{A}^{(2)} w^{(2)} + b|
\]
\[
= |w \cdot A + b|
\]

**Note we assume** \( \|w\|_2 = 1 \)

\[
\text{Remember } x_{M}^{(1)}w^{(1)} + x_{M}^{(2)}w^{(2)} = -b \\
since M \text{ belongs to line } L
\]
- 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
  $$\max_{w,\gamma} \gamma$$
  s.t. $\forall i, y_i (w \cdot x_i + b) \geq \gamma$
Support Vector Machine

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

\[
\begin{align*}
\max_{w, \gamma} & \quad \gamma \\
\text{s.t.} & \quad \forall i, y_i(w \cdot x_i + b) \geq \gamma
\end{align*}
\]

- \( \gamma \) is margin ... distance from the separating hyperplane

Maximizing the margin
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!

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:

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

\[ \gamma = \frac{||w||}{w \cdot w} = \frac{1}{||w||} \]

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

- **We started with**
  \[
  \max_{w,\gamma} \gamma
  \]
  \[s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma\]
  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:**
  \[
  \min_w \frac{1}{2} \|w\|^2
  \]
  \[s.t. \forall i, y_i (w \cdot x_i + b) \geq 1\]

This is called SVM with “hard” constraints
If data is not separable introduce penalty:

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

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

- 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** $\xi_i$

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

\[\text{s.t.} \forall i, y_i (w \cdot x_i + b) \geq 1 - \xi_i\]

- 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

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

Margin

Regularization parameter

Empirical loss L (how well we fit training data)

- SVM uses “Hinge Loss”:

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

\[\text{s.t. } \forall i, y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i\]

Hinge loss: \(\max\{0, 1-z\}\)

0/1 loss

penalty
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 $J(w,b)$:

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

Empirical loss $L(x_i, y_i)$

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

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

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

$$= -y_i x_i^{(j)} \quad \text{else}$$
SVM: How to estimate \( w \)?

- **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:**
        
        \( w'(j) \leftarrow w(j) - \eta \nabla J(j) \nabla J(j) \leftarrow w' \)

- **Problem:**

  - Computing \( \nabla J(j) \) takes \( O(n) \) time!
    
      - \( n \) ... size of the training dataset

\( \eta \) ... learning rate parameter

\( C \) ... regularization parameter
Stochastic Gradient Descent

Instead of evaluating gradient over all examples evaluate it for each individual training example

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

Stochastic gradient descent:

Iterate until convergence:

• For \( i = 1 \ldots n \)
  • For \( j = 1 \ldots d \)
    • Compute: \( \nabla J^{(j)}(x_i) \)
    • 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
Support Vector Machines: Example
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 $J(w,b)$?
(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 “Accuracy”

For optimizing $J(w, b)$ within reasonable quality
SGD-SVM is super fast
Practical Considerations

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

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

- **Leon suggests:**
  - 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
Practical Considerations

- **Sparse Linear SVM:**
  - Feature vector $x_i$ is sparse (contains many zeros)
    - Do not do: $x_i = [0, 0, 0, 1, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, \ldots]$
    - But represent $x_i$ as a sparse vector $x_i = [(4, 1), (9, 5), \ldots]$
  - 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:
    - \[ w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w} \] cheap: $x_i$ is sparse and so few coordinates $j$ of $w$ will be updated
    - \[ w \leftarrow w(1 - \eta) \] expensive: $w$ is not sparse, all coordinates need to be updated
Practical Considerations

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

- **Solution 2**:  
  - Perform only step \((1)\) for each training example
  - Perform step \((2)\) with lower frequency and higher \(\eta\)

Two step update procedure:

\[(1) \ \mathbf{w} \leftarrow \mathbf{w} - \eta C \frac{\partial L(x_i, y_i)}{\partial \mathbf{w}}\]
\[(2) \ \mathbf{w} \leftarrow \mathbf{w}(1 - \eta)\]
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 (very) small sets of training data A and B
  - Train on A, stop by validating on B
  - Number of training epochs on A 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 $y_i$ to have highest margin:

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

(x, y)
Multiclass SVM

- **Optimization problem:**

\[
\begin{align*}
\min_{w,b} & \quad \frac{1}{2} \sum_c \|w_c\|^2 + C \sum_{i=1}^n \xi_i \\
\text{s.t.} & \quad w_{y_i} \cdot x_i + b_{y_i} \geq w_c \cdot x_i + b_c + 1 - \xi_i, \quad \forall c \neq y_i, \forall i \\
\xi_i & \geq 0, \forall i
\end{align*}
\]

- 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.
Support Vector Machines: Example
Online Learning

- **New setting: Online Learning**
  - Allows for modeling problems where we have a continuous stream of data
  - We want an algorithm to learn from it and slowly adapt to the changes in data

- **Idea: Do slow updates to the model**
  - SGD-SVM makes updates if misclassifying a datapoint
  - **So:** First train the classifier on training data. Then for every example from the stream, if we misclassify, update the model (using a small learning rate)
Example: Shipping Service

- **Protocol:**
  - User comes and tell us origin and destination
  - We offer to ship the package for some money ($10 - $50)
  - Based on the price we offer, sometimes the user uses our service ($y = 1$), sometimes they don't ($y = -1$)

- **Task:** Build an algorithm to optimize what price we offer to the users

- **Features $x$ capture:**
  - Information about user
  - Origin and destination

- **Problem:** Will user accept the price?
Example: Shipping Service

- Model whether user will accept our price:
  \[ y = f(x; w) \]
  - Accept: \( y = 1 \), Not accept: \( y = -1 \)
  - Build this model with say Perceptron or SVM
- The website that runs continuously
- Online learning algorithm would do something like
  - User comes
  - User is represented as an \((x, y)\) pair where
    - \( x \): Feature vector including price we offer, origin, destination
    - \( y \): If they chose to use our service or not
  - The algorithm updates \( w \) using just the \((x, y)\) pair
  - Basically, we update the \( w \) parameters every time we get some new data
We discard this idea of a data “set”
Instead we have a continuous stream of data

Further comments:
- For a major website where you have a massive stream of data then this kind of algorithm is pretty reasonable
- Don’t need to deal with all the training data
- If you had a small number of users you could save their data and then run a normal algorithm on the full dataset
  - Doing multiple passes over the data
Online Algorithms

- An online algorithm can adapt to changing user preferences
- For example, over time users may become more price sensitive
- The algorithm adapts and learns this
- So the system is dynamic