Lecture 3
Linear Classification Models
Perceptron
Classification problem

Let’s look at the problem of spam filtering

Now anyone can learn how to earn $200 - $943 per day or More! If you can type (hunt and peck is ok to start) and fill in forms, you can score big! So don't delay waiting around for the next opportunity...it is knocking now! Start here: http://redbluecruise.com/t/c/381/polohoo/yz37957.html

Do you Have Poetry that you think should be worth $10,000.00 USD, we do!.. Enter our International Open contest and see if you have what it takes. To see details or to enter your own poem, Click link below. http://e-suscriber.com/imys?e=0sAoo4q9s4zYYUoYQ&m=795314&l=0

View my photos!
I invite you to view the following photo album(s): zak-month27

Hey have you seen my new pics yet???? Me and my girlfreind would love it if you would come chat with us for a bit.. Well join us if you interested. Join live web cam chat here: http://e-commcentral.com/imys?e=0sAoo4q9s4zYYUoYQ&m=825314&l=0
Let’s look at the design choices

• Training data?
  – Past emails and whether they are considered spam or not (you can also choose to use non-spam or spam emails only, but that will require different choices later on)

• Target function?
  – Email -> spam or not

• Representation of the function?
  – ?

• Learning algorithm
  – ?

We will focus a lot on these two aspects in this class.
Continue with the design choices

• Representation of the function (email -> spam or not)?
• First of all, how to represent an email?
  – Use bag-of-words to represent an email
  – This will turn an email into a collection of features, e.g., where each feature describes whether a particular word is present in the email (alternatively, the feature could be the count or normalized count of the words)

• This gives us a standard supervised classification problem typically seen in textbooks and papers
  – Training set: a set of examples (instances, objects) with class labels, e.g., positive (spam) and negative (non-spam)
  – Input representation: an example is described by a set of attributes/features (e.g., whether “$” is present, etc.)
  – Goal: Given an unseen email, and its input representation, predict its label

• Next question: what function forms to use?
Linear Classifier

- We will begin with the simplest choice: linear classifiers.
Linear Threshold Units (McCulloch & Pitts 1943)

- Assume each feature $x_j$ and weight $w_j$ is a real number
- Compute $w_0 + w_1 x_1 + \ldots + w_n x_n$ (compactly represented as $x^T w$) and compare its value to a threshold
- Why do we want to use a linear model?
  - Simplest model – fewer parameters to learn (requires less data)
  - Visually intuitive - drawing a straight line to separate positive from negative

$$y = \begin{cases} 
+1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\
-1 & \text{otherwise}
\end{cases}$$
Geometric view

\[ x_1 + x_2 = 1 \]

Referred to as decision boundary

\[ W=(1,1), \text{ points to the positive side} \]
A Canonical Representation

- Given a training example: \([x_1, x_2, \ldots, x_m, y]\)
- transform it to \([1, x_1, x_2, \ldots, x_m, y]\)
- The parameter vector will then be
  \[w = [w_0, w_1, w_2, \ldots, w_m]\]
- Given a training set, we need to learn
  \[g(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = x^T w\]
- Given a input example \(x\), we predict positive if \(g(x, w) > 0\), and predict negative otherwise
- Note that if \(y\) is the true label of \(x\), our prediction is correct if and only if \(y \cdot g(x, w) > 0\)

We call \(w\) the weight vector, or the weight coefficients.
Geometrically, using the canonical representation translates to two things:
1. It will increase the input space dimension by 1, and make the learning of $w_0$ the same as the rest of the coefficients
2. The decision boundary now always passes through the origin.
Geometric view
How to learn: the perceptron algorithm

The goal of learning is to find a weight vector $\mathbf{w}=(w_0, w_1, \ldots, w_m)$ such that its decision boundary correctly separate positive examples from negative examples.

How can we achieve this?

Perceptron is one approach. It starts with some vector $\mathbf{w}$ and incrementally updates $\mathbf{w}$ when it makes a mistake.

Let $\mathbf{w}_t$ be current weight vector, and suppose it makes a mistake on example $\langle \mathbf{x}, y \rangle$, that is to say $y \cdot \mathbf{x}^T \mathbf{w}_t < 0$. The perceptron update rule is: $\mathbf{w}_{t+1} = \mathbf{w}_t + y \cdot \mathbf{x}$
The Perceptron Algorithm

Let $\mathbf{w} \leftarrow (0,0,0,...,0)$  //Start with 0 weights
Repeat  //Receive training examples one by one
    Accept training example $i : (\mathbf{x}_i , y_i)$
    $u_i \leftarrow \mathbf{x}_i^T \mathbf{w}$  //Apply the current weight
    if $y_i \cdot u_i \leq 0$  //If it is misclassified
        $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$  //update $\mathbf{w}$

- Correcting for a mistake could move the decision boundary so much that previously correct examples are now misclassified.
- This algorithm will need to go over the training examples multiple times, each time it goes through the training set it is called an epoch.
- It will terminate if no update is made to $\mathbf{w}$ during one epoch – which means it has converged.
Effect of Perceptron Updating Rule

• Mathematically speaking

\[ y x^T w_{t+1} = y \cdot x^T (w_t + y\cdot x) = yx^T w_t + y^2 ||x||^2 \]

\[ > yx^T w_t \]

The updating rule makes \( yx^T w_t \) more positive, thus can potentially correct the mistake.

• Geometrically

![Diagram showing the effect of the updating rule from step t to step t+1]
Online vs Batch

• We call the above perceptron algorithm an **online algorithm**
• Online algorithms perform learning each time it receives a training example
• In contrast, **batch learning** algorithms collect a batch of training examples and learn from them all at once.
Batch Perceptron Algorithm

Given: training examples \((x^i, y^i), i = 1, \ldots, N\)

Let \(w \leftarrow (0,0,0,\ldots,0)\)

do

\(\delta \leftarrow (0,0,0,\ldots,0)\)

for \(i = 1\) to \(N\) do

\(u^i \leftarrow w \cdot x^i\)

if \(y^i \cdot u^i \leq 0\)

\(\delta \leftarrow \delta + y^i \cdot x^i\)

\(\delta \leftarrow \delta / N\)

\(w \leftarrow w + \eta \delta\)

until \(|\delta| < \varepsilon\)
Good news

• If there is a linear decision boundary that correctly classify all training examples, this algorithm will find it.
• Formally speaking, this is the convergence Property: For linearly separable data (i.e., there exists an linear decision boundary that perfectly separates positive and negative training examples), the perceptron algorithm converges in a finite number of steps.
• Why? If you are mathematically curious, read the following slide, you will find the answer.
• And how many steps? If you are practically curious, read the following slide, answer is in there too.
• The further good news is that you are not required to master this material, they are just provided for the curious ones.
To show convergence, we just need to show that each update moves the weight vector closer to a solution vector by a lower bounded amount.

Let $w^*$ be a solution vector, and $w_t$ be our $w$ at $t$th step,

$$\text{cosine}(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \cdot \|w_t\|}$$

$$w^* \cdot w_t = w^* \cdot (w_{t-1} + y^t x^t) = w^* \cdot w_{t-1} + w^* y^t x^t$$

Assume that $w^*$ classify all examples with a margin $\gamma$, i.e., $w^* y^x > \gamma$ for all examples.

$$w^* \cdot w_t = w^* \cdot w_{t-1} + w^* y^t x^t > w^* \cdot w_{t-1} + \gamma > w^* \cdot w_{t-2} + 2\gamma > \ldots > w^* w_0 + t\gamma = t\gamma$$

$$\|w_t\|^2 = \|w_{t-1} + y^t x^t\|^2 = \|w_{t-1}\|^2 + y^t^2 \|x^t\|^2 + 2w_{t-1} y^t x^t < \|w_{t-1}\|^2 + \|x^t\|^2$$

Assume that $\|x\|$ are bounded by $D$

$$\|w_t\|^2 < \|w_{t-1}\|^2 + \|x^t\|^2 < \|w_{t-1}\|^2 + D^2 < \|w_{t-2}\|^2 + 2D^2 < \ldots < tD^2$$

$$\text{cosine}(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \cdot \|w_t\|} > \frac{t\gamma}{\|w^*\| \cdot \sqrt{tD^2}} = \frac{t\gamma}{\|w^*\| \cdot \sqrt{tD^2}}$$

$$\frac{t\gamma}{\|w^*\| \cdot \sqrt{tD^2}} < 1 \Rightarrow \sqrt{t} < \frac{D\|w^*\|}{\gamma} \Rightarrow t < D^2 \frac{\gamma^2}{\|w^*\|^2}$$
Margin

- The bigger the margin, the easier the classification problem is, the perceptron algorithm will likely find the solution faster!
- Side story: the bigger the margin, the more confident we are about our prediction, which makes it desirable to find the one that gives the maximum margin
- Later in the course this concept will be core to one of the recent exciting developments in the ML field – support vector machine
Bad news

In such cases the algorithm will never stop! How to fix?

One possible solution: look for decision boundary that make as few mistakes as possible – NP-hard (refresh your 325 memory!)
Fixing the Perceptron

Idea one: only go through the data once, or a fixed number of times

Let $\mathbf{w} \leftarrow (0,0,0,...,0)$
for $i = 1,...,N$

    Take training example $i : (\mathbf{x}_i , y_i)$

    $u_i \leftarrow \mathbf{x}_i^T \mathbf{w}$

    if $y_i u_i \leq 0$

    \[ \mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i \]

At least this stops!

Problem: the final $\mathbf{w}$ might not be good
e.g., right before terminating, the algorithm might perform an update on a total outlier...
Voted-Perceptron

Idea two: keep around intermediate hypotheses, and have them “vote” [Freund and Schapire, 1998]

Let $w_0 = (0,0,0,...,0)$
$c_0 = 0$
repeat for a fixed number of steps
Take example $i: (x_i, y_i)$
$u_i \leftarrow x_i^T w_n$
if $y_i u_i <= 0$
    $w_{n+1} \leftarrow w_n + y_i x_i$
    $c_{n+1} = 0$
    $n = n + 1$
else
    $c_n = c_n + 1$

Store a collection of linear separators $w_0, w_1, ..., $ along with their survival time $c_0, c_1, ...$
The c’s can be good measures of reliability of the w’s.

For classification, take a weighted vote among all separators:

$$\text{sgn} \left\{ \sum_{n=0}^{N} c_n \text{sgn}(x^T w) \right\}$$