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 describe 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 text books 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 (eg. one feature could be 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

\[ w_1 x_1 + w_2 x_2 + w_0 > 0 \]
\[ w_1 x_1 + w_2 x_2 + w_0 < 0 \]
\[ w_1 x_1 + w_2 x_2 + w_0 = 0 \]
Why linear model?

• Simplest model – fewer parameters to learn (requires less training data to learn reliably)

• Intuitively appealing -- draw a straight line (for 2-d inputs) or a linear hyper-plane (for higher dimensional inputs) to separate positive from negative

• Can be used to learn nonlinear models as well. How?
  – Introducing nonlinear features (e.g., $x_1^2, x_2^2, x_1x_2 \ldots$)
  – Use kernel tricks (we will talk about this later this term)
Learning Goal

• Given a set training examples, each is described by $m$ features $x_1, \ldots, x_m$, and belong to either the positive or negative class, $y \in \{+1, -1\}$

• Let $x = [1, x_1, \ldots, x_m]^T$,

• $w = [w_0, w_1, \ldots, w_m]^T$ defines a decision boundary $w^T x = 0$ that the input space into two parts

The goal is to find a weight vector $w$ such that its decision boundary correctly separate positive training examples from negative training examples
How to learn: the perceptron algorithm

How can we achieve this? Perceptron is one approach.

1. It starts with some (random) vector $\mathbf{w}$ and incrementally updates $\mathbf{w}$ when it makes a mistake.

2. 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 $y \cdot \mathbf{x}^T \mathbf{w}_t < 0$.

3. The perceptron update the weight by: $\mathbf{w}_{t+1} = \mathbf{w}_t + y \cdot \mathbf{x}$
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 $y_i u_i \leq 0$ // If it is misclassified

$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ // update $\mathbf{w}$

Important notes:

- Correcting for a mistake could move the decision boundary so much that previously correct examples are now misclassified.
- As such it must go over the training examples multiple times
- Each time it goes through the whole 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

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

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

• Geometrically
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 \((x_i, y_i), i = 1, ..., N\)
Let \(w \leftarrow (0,0,0,...,0)\)
do
\[
\text{delta} \leftarrow (0,0,0,...,0)
\]  
for \(i = 1\) to \(N\) do
\[
\text{delta} \leftarrow \text{delta} + y_i x_i
\]
end
\[
delta \leftarrow \text{delta} / N
\]
w \leftarrow w + \eta \delta
until \(|\text{delta}| < \varepsilon\)

Essentially perceptron does gradient descent to minimize a certain loss function
\(\text{delta}\) stores the gradient
\(\eta\) is the learning rate of the gradient descent steps
Too large \(\eta\) causes oscillation, too small leads to slow convergence
Common to use large \(\eta\) first, then gradually reduce it
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,

\[
\cosine(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \cdot \|w_t\|}
\]

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

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

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

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

Assume that \( \|x\| \) are bounded by \( D \)

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

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

\[
\frac{t \gamma}{\|w^*\| \cdot \sqrt{tD^2}} < 1 \Rightarrow \sqrt{t} < \frac{D \|w^*\|}{\gamma} \Rightarrow t < D^2 \left( \frac{\gamma^2}{\|w^*\|^2} \right)
\]
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 $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$

for $i = 1,\ldots,N$

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

Store a collection of linear separators $w_0, w_1, ..., w_n$, 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 $N$ separators:

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