Linear classification models: Perceptron
Classification problem

- Given input $x$, the goal is to predict $y$, which is a categorical variable
  - $y$ is called the class label
  - $x$ is the feature vector

- Example:
  - $x$: monthly income and bank saving amount;
    $y$: risky or not risky
  - $x$: review text for a product
    $y$: sentiment positive, negative or neutral
Linear Classifier

• We will be 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)
Binary classification: General Setup

• Given a set of training examples \((x_1, y_1), \ldots, (x_n, y_n)\), where each \(x_i \in \mathbb{R}^d\), \(y_i \in \{-1, 1\}\)

• Learn a linear function

\[
g(x, w) = w_0 + w_1 x_1 + \cdots + w_d x_d
\]

Given an example \(x = [x_1, \ldots, x_d]^T\):

– predict \(y(x) = 1\) if \(g(x, w) \geq 0\)
– predict \(y(x) = 0\) otherwise

• Compactly the classifier can be represented as:

\[
y(x) = \text{sign}(w_0 + w_1 x_1 + \cdots + w_d x_d) = \text{sign}(w^T x)
\]

where \(w = [w_0, w_1, \ldots, w_d]^T\), and \(x = [1, x_1, \ldots, x_d]^T\)

• Goal: find a good \(w\) that minimizes some loss function \(J(w)\)
0/1 Loss

\[
J_{0/1}(w) = \frac{1}{n} \sum_{m=1}^{n} L(\text{sign}(w^T x_m), y_m)
\]

where \( L(y', y) = 0 \) when \( y' = y \), otherwise \( L(y', y) = 1 \).

Issue: does not produce useful gradient since the surface of \( J_{0/1} \) is piece-wise flat.
Perceptron Loss

\[ J_p(w) = \frac{1}{n} \sum_{m=1}^{n} \max(0, -y_m w^T x_m) \]

- If prediction is correct, \( y_m w^T x_m > 0 \), \( \max(0, -y_m w^T x_m) = 0 \)
- If incorrect, \( y_m w^T x_m \leq 0 \), \( \max(0, -y_m w^T x_m) = -y_m w^T x_m > 0 \)
  - A linear function of input features
- \( J_p \) is piecewise linear
  - Has a nice gradient leading to the solution region

0/1 loss

Perceptron criterion
**Stochastic Gradient Descent**

- The objective function consists of a sum over data points---**Stochastic Gradient Descent** updates the parameter after observing each example.

\[
J(w) = \frac{1}{n} \sum_{m=1}^{n} \max(0, -y_m w^T x_m)
\]

\[
J_m(w) = \max(0, -y_m w^T x_m)
\]

\[
\nabla J_m = \begin{cases} 
0 & \text{if } y_m w \cdot x_m > 0 \\
-y_m x_m & \text{otherwise}
\end{cases}
\]

**Update Rule**

After observing \((x_m, y_m)\), if it is a mistake \(w \leftarrow w + y_m x_m\)
Online Perceptron
(Stochastic gradient descent)

Let \( \mathbf{w} \leftarrow (0,0,0,...,0) \)

Repeat until convergence

for every training example \( m = 1,...,n \):

\[
\begin{align*}
\mathbf{u}_m &\leftarrow \mathbf{w}^T \mathbf{x}_m \\
\text{if } y_m \cdot \mathbf{u}_m &\leq 0 \quad \mathbf{w} &\leftarrow \mathbf{w} + y_m \mathbf{x}_m
\end{align*}
\]
Red points belong to the positive class, blue points belong to the negative class.
Convergence Theorem
(Block, 1962, Novikoff, 1962)

Given training example sequence \((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\).

If \(\forall i, \|x_i\| \leq D\), and \(\exists u, \|u\| = 1\) and \(y_i u^T x_i \geq \gamma > 0\) for all \(i\),
then the number of mistakes that the perceptron algorithm makes is at most \((D / \gamma)^2\).

Note that \(\|\cdot\|\) is the Euclidean norm of a vector.
Let \( \mathbf{u} \) be a solution vector, we know then \( \alpha \mathbf{u} \) is also a solution.

Let \( \mathbf{x}_k \) be the kth mistake, we have \( \mathbf{w}(k+1) = \mathbf{w}(k) + y_k \mathbf{x}_k \)

\[
\|\mathbf{w}(k+1) - \alpha \mathbf{u}\|^2 \\
= \|\mathbf{w}(k) + y_k \mathbf{x}_k - \alpha \mathbf{u}\|^2 = \|(\mathbf{w}(k) - \alpha \mathbf{u}) + y_k \mathbf{x}_k\|^2 \\
= \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2 y_k [\mathbf{x}_k \cdot (\mathbf{w}(k) - \alpha \mathbf{u})] + (y_k)^2 \|\mathbf{x}_k\|^2 \\
= \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2 y_k \mathbf{x}_k \cdot \mathbf{w}(k) - 2 y_k \alpha \mathbf{u} \cdot \mathbf{x}_k + \|\mathbf{x}_k\|^2 \\
\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 + 2 y_k \mathbf{x}_k \cdot \mathbf{w}(k) - 2 y_k \alpha \mathbf{u} \cdot \mathbf{x}_k + D^2 \quad \text{, because } \|\mathbf{x}_k\| \leq D \\
\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - 2 \alpha y_k \mathbf{u} \cdot \mathbf{x}_k + D^2 \quad \text{, because } y_k \mathbf{x}_k \cdot \mathbf{w}(k) \leq 0 \\
\leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - 2 \alpha \gamma + D^2 \quad \text{, because } y_k \mathbf{u} \cdot \mathbf{x}_k \geq \gamma
\]

Because \( \alpha \) is an arbitrary scaling factor, we can set \( \alpha = \frac{D^2}{\gamma} \)

\[
\|\mathbf{w}(k+1) - \alpha \mathbf{u}\|^2 \leq \|\mathbf{w}(k) - \alpha \mathbf{u}\|^2 - D^2
\]
Proof (cont.)

By induction on $k$

$$||w(k+1) - \alpha u||^2 \leq ||w(1) - \alpha u||^2 - kD^2 = \alpha^2 ||u||^2 - kD^2 = \alpha^2 - kD^2$$

$\iff \alpha^2 - kD^2 \geq 0$

$\iff k \leq \frac{\alpha^2}{D^2} \quad (\alpha = \frac{D^2}{\gamma})$

$\iff k \leq (D / \gamma)^2$
Margin

• $\gamma$ is referred to as the **margin**
  – Min distance from data points to the decision boundary
  – Bigger margin -> easier the classification problem
  – Bigger margin -> more confidence in our prediction

• This concept will be utilized in later methods: support vector machines
Batch Perceptron Algorithm

Given: training examples \((x_m, y_m), m = 1, \ldots, n\)

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

repeat{
    \(
    delta \leftarrow (0,0,0,\ldots,0)
    \)
    for \(m = 1\) to \(n\) {
        \(u_m \leftarrow w^T x_m\)
        if \(y_m u_m \leq 0\): \(delta \leftarrow delta - y_m x_m\)
    }
    \(delta \leftarrow delta / n\)
    \(w \leftarrow w - \lambda \, delta\)
} until \(|delta| < \varepsilon\)
Online VS. Batch Perceptron

- Batch learning learns from a batch of examples collectively
- Online learning learns from one example at a time
- Both learning mechanisms are useful in practice
- Online Perceptron is sensitive to the order training examples are received
- In batch training, the corrections are accumulated and applied at once
- In online training, each correction is applied immediately once a mistake is encountered, which will change the decision boundary, thus different mistakes maybe encountered for online and batch training
- Online training performs stochastic gradient descent, an approximation to the real gradient descent used by the batch training
Not linearly separable case

- In such cases the algorithm will never converge! How to fix?
- Look for decision boundary that make as few mistakes as possible – NP-hard!
Fixing the Perceptron

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

Let $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$

Repeat for $T$ times

    for each training example $i$:

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

    if $y_i u_i \leq 0$ \hspace{1cm} $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$

• At least this stops
• Problem: the final $\mathbf{w}$ might not be good e.g. the last update could be on a total outlier
Voted Perceptron

• Keep intermediate hypotheses and have them vote [Freund and Schapire 1998]

Let \( \mathbf{w} \leftarrow (0,0,0,\ldots,0) \)
\( c_0 = 0, \ n = 0 \)
Repeat for T times
  for each training example \( i \):
    \( u_i \leftarrow \mathbf{w}^T x_i \)
    if \( y_i u_i \leq 0 \)
      \( \mathbf{w}_{n+1} \leftarrow \mathbf{w}_n + y_i x_i \)
    \( n = n + 1 \)
    \( c_n = 0 \)
  else \( c_n = c_n + 1 \)

The output will be a collection of linear separators \( \mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_M \) along with their survival time \( c_0, c_1, \ldots, c_M \)

The \( c \)'s can be viewed as measures of the reliability of the \( \mathbf{w} \)'s

For classification, take a weighted vote among all separators:

\[
\hat{y} = \text{sign}\{\sum_{n=0}^{N} c_n \text{sign}(\mathbf{w}_n^T \mathbf{x})\}
\]
Average Perceptron

• Voted perceptron requires storing all intermittent weights
  – Large memory consumption
  – Slow prediction time

• Average perceptron
  \[
  \hat{y} = \text{sign} \left\{ \left( \sum_{n=0}^{N} c_n w_n^T \right) x \right\}
  \]
  – Take the weighted average of all the intermittent weights
  – Can be implemented by maintaining an running average, no need to store all weights
  – Fast prediction time
Final Discussion

• Perceptron learns $\hat{y} = f(x)$ directly – a discriminative method

• Gradient descent to optimize the perceptron loss
  – Online version performs stochastic gradient descent

• Guaranteed to converge in finite steps if linearly separable
  – The upper bound on the number of corrections needed is inversely proportional to the margin of the optimal decision boundary

• If not linearly separable, voted or average perceptrons can be used

• Hyper-parameter: the number of epochs (T)
  – Very large T could still lead to overfitting