# (Binary) Linear Classifiers: Representation and Optimization

Vibhav Gogate

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

# Task: Learning Linear Classifiers

- **Given:** Dataset $D$ having $d$ examples defined over a set of features $\mathbf{X} = \{X_0, \ldots, X_n\}$ and desired output variable $Y$ (also called the class variable). Assume that the class variable is binary and $X_0$ is a dummy feature which is always 1.
- **To do:** Find a function $h$ over $\mathbf{X}$ such that: (1) $h$ is the best approximation of $f$ according to a user-defined performance measure, and (2) $h$ is a *linear threshold function* given by

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

  where $w_i \in \mathbb{R}$ is a parameter associated with feature $X_i$.
- **At test time:** Use $h$ to find the class $Y$ given $\mathbf{x}$.

# Performance Measure: Minimize the Mis-classification Error

- Until now, our classifiers (LR and NB) were probability driven. Let us be **error driven**.
- Mis-classification error

$$\text{Error} = \sum_{k=1}^{d} \left( y^{(k)} - h(\mathbf{x}^{(k)}) \right)^2$$

  (the term inside the summation will be 1 iff the predicted value $h(\mathbf{x}^{(k)})$ is not the same as $y^{(k)}$ and 0 otherwise)
- $h$ is not differentiable (discontinuous at 0). It creates problems during optimization.
- Therefore, we need to use a continuous function instead of $h$, which in turn also changes the performance measure.

# Use the Sigmoid/Logistic Function

- What if we use the Sigmoid function instead of the threshold function?

$$h(\mathbf{x}) \approx o(\mathbf{x}) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right) = \frac{\exp\left(\sum_{i=0}^{n} w_i x_i\right)}{1 + \exp\left(\sum_{i=0}^{n} w_i x_i\right)}$$

- Mis-classification error

$$\text{Error} = \sum_{k=1}^{d} \left(y^{(k)} - h(\mathbf{x}^{(k)})\right)^2 \approx \sum_{k=1}^{d} \left(y^{(k)} - o(\mathbf{x}^{(k)})\right)^2$$

# Unifying Theme: Learning = Representation + Evaluation Measure + Optimization

**Error-driven Classifier**

- Approximate $h$ by

$$o(\mathbf{x}) = \frac{\exp\left(\sum_{i=0}^{n} w_i x_i\right)}{1 + \exp\left(\sum_{i=0}^{n} w_i x_i\right)}$$

- Performance Measure: Minimize

$$\sum_{k=1}^{d} \left(y^{(k)} - o(\mathbf{x}^{(k)})\right)^2$$

**Logistic Regression**

- $P(Y = 1|\mathbf{x})$ equals

$$o(\mathbf{x}) = \frac{\exp\left(\sum_{i=0}^{n} w_i x_i\right)}{1 + \exp\left(\sum_{i=0}^{n} w_i x_i\right)}$$

- Performance Measure: Maximize

$$\prod_{k=1}^{d} o(\mathbf{x}^{(k)})^{y^{(k)}} (1 - o(\mathbf{x}^{(k)}))^{1-y^{(k)}}$$

# Optimize the error: Gradient Descent

$$
\begin{aligned}
E &= \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right)^2 \\
\frac{\partial E}{\partial w_i} &= 2 \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) \frac{-\partial o(\mathbf{x}^{(k)})}{\partial w_i} \\
\frac{\partial E}{\partial w_i} &= 2 \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) o(\mathbf{x}^{(k)})(1 - o(\mathbf{x}^{(k)})) \frac{-\partial \sum_{i=0}^{n} w_i x_i^{(k)}}{\partial w_i} \\
\frac{\partial E}{\partial w_i} &= -2 \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) o(\mathbf{x}^{(k)})(1 - o(\mathbf{x}^{(k)})) x_i^{(k)}
\end{aligned}
$$

Step 3: Derivative of sigmoid: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

# Weight update rule for LR versus our error-driven classifier

- Error-driven

$$w_i = w_i - \alpha \frac{\partial E}{\partial w_i}$$

Absorbing the 2 in $\alpha$, we get:

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) o(\mathbf{x}^{(k)})(1 - o(\mathbf{x}^{(k)}))x_i^{(k)}$$

- LR

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) x_i^{(k)}$$

Recall: $\frac{\partial E}{\partial w_i} = -2 \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right) o(\mathbf{x}^{(k)})(1 - o(\mathbf{x}^{(k)}))x_i^{(k)}$

# Other Options: Use tanh

- We can use the *tanh* function. It lies between $-1$ and 1

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- Minor change: we will use $+1$ for positive class as $-1$ for negative class. We will call the new function $s$ to distinguish it from $h$. Formally,

$$s(\mathbf{x}) = +1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \text{ and } -1 \text{ otherwise}$$

- Derivative: $\frac{d}{dx} tanh(x) = 1 - tanh^2(x)$.
- Approximate $s$ by $t$ where $t(\mathbf{x}) = tanh(\sum_{i=0}^{n} w_i x_i)$
- Gradient descent rule:

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left( y^{(k)} - t(\mathbf{x}^{(k)}) \right) (1 - [t(\mathbf{x}^{(k)})]^2) x_i^{(k)}$$

# Other Options: Use a linear approximation

- Approximate $s$ by $l$ where $l(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i$
- Error is approximated using

$$E = \sum_{k=1}^{d} \left( y^{(k)} - s(\mathbf{x}^{(k)}) \right)^2 \approx \sum_{k=1}^{d} \left( y^{(k)} - l(\mathbf{x}^{(k)}) \right)^2$$

- Gradient descent rule:

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left( y^{(k)} - l(\mathbf{x}^{(k)}) \right) x_i^{(k)}$$

# Stochastic Gradient Descent (SGD)

**Batch Algorithm**

Repeat Until Convergence

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} \sum_{k=1}^{d} \left( y^{(k)} - o(\mathbf{x}^{(k)}) \right)^2$$

**Stochastic Algorithm**

Repeat Until Convergence

- For each training example $(\mathbf{x}, y)$

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} (y - o(\mathbf{x}))^2$$

**Stochastic Gradient Descent can approximate batch descent arbitrarily closely if $\alpha$ is made small enough.**

- **Mini-Batch:** Use randomly chosen $t < d$ examples.

# The Perceptron Algorithm

- Turns out that when data is linearly separable, namely there exists a hyper-plane which can separate positive examples from negative examples (in two dimensions, a line), there exists a much simpler algorithm. This algorithm is the classic **Perceptron** algorithm.

**Algorithm:**

- Repeat the following Until Convergence
- For each training example $(\mathbf{x}, y)$ in $D$
  - Update each weight $w_i$ using the following rule:

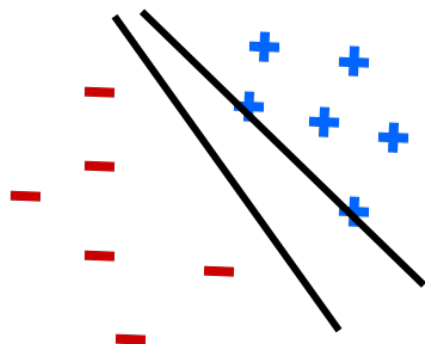
$$w_i = w_i + \alpha(y - s(\mathbf{x}))x_i$$

Recall: $s(\mathbf{x}) = +1$ if $\sum_{i=0}^{n} w_i x_i > 0$ and $-1$ otherwise

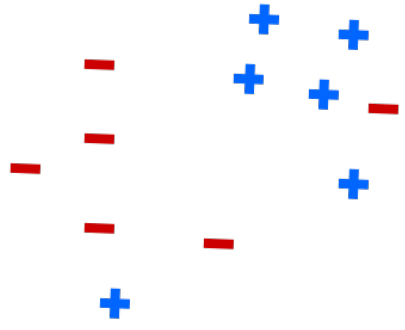
# Properties of the Perceptron Algorithm

- Converges if the data is linearly separable (Proof is a little bit involved.) However, it has mediocre generalization: often finds a "barely" separating solution
- Convergence is not assured if data is not linearly separable. In fact in many cases, it will not converge. In such cases, weights might thrash. Averaging weight vectors over time can help (averaged perceptron)

Separable case



Non-Separable case



# Summary

- A classifier which uses the following rule is a linear classifier

$$\text{If } \sum_i w_i x_i > 0 \text{ then class is positive else negative}$$

- Threshold function is not differentiable and therefore we typically approximate it using a differentiable function
- **Recipe:** Substitute the (differentiable) approximation in a suitable evaluation function and then use the gradient descent (ascent) algorithm to yield a linear classifier.
- Stochastic vs Batch vs Mini-batch gradient descent