Everything you’ve ever wanted to know about linear classifiers (Part 1)
Outline

• Examples of classification models: nearest neighbor, linear
• Empirical loss minimization framework
• Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
The basic *supervised learning* framework

\[ y = f(x) \]

- **Training** (or **learning**): given a *training set* of labeled examples \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), instantiate a predictor \( f \)
- **Testing** (or **inference**): apply \( f \) to a new *test example* \( x \) and output the predicted value \( y = f(x) \)
Nearest neighbor classifier

\[ f(x) = \text{label of the training example nearest to } x \]

- All we need is a distance function for our inputs
- No training required!
Nearest neighbors of images based on raw pixel values

Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

Source: http://cs231n.github.io/classification/
K-nearest neighbor classifier

- For a new point, find the $k$ closest points from training data
- Vote for class label with labels of the $k$ points

$k = 5$
K-nearest neighbor classifier

- For a new point, find the $k$ closest points from training data
- Vote for class label with labels of the $k$ points
- 1-NN vs. $k$-NN

Source: [http://cs231n.github.io/classification/](http://cs231n.github.io/classification/)
Linear classifier

- Find a *linear function* to separate the classes:

\[ f(x) = \text{sgn}(w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \cdots + w^{(D)}x^{(D)} + b) = \text{sgn}(w \cdot x + b) \]
Visualizing linear classifiers

Seismic data classification

- Body wave magnitude
- Surface wave magnitude

- Earthquakes
- Nuclear explosions
Visualizing linear classifiers

Source: http://cs231n.github.io/linear-classify/
Linear classifier: Perceptron view

\[ \text{Input: } x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(D)} \]

\[ \text{Weights: } w^{(1)}, w^{(2)}, w^{(3)}, \ldots, w^{(D)} \]

\[ \text{Output: } \text{sgn}(w \cdot x + b) \]
Loose inspiration: Biological neurons
Perceptrons, linear separability, Boolean functions

- \(x^1 \text{ and } x^2\)
- \(x^1 \text{ or } x^2\)
- \(x^1 \text{ xor } x^2\)

NN vs. linear classifiers: Pros and cons

• **NN pros:**
  + Simple to implement
  + Decision boundaries not necessarily linear
  + Works for any number of classes
  + *Nonparametric* method

• **NN cons:**
  - Need good distance function
  - Slow at test time

• **Linear pros:**
  + Low-dimensional *parametric* representation
  + Very fast at test time

• **Linear cons:**
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?
Outline

• Examples of classification models: nearest neighbor, linear
• Empirical loss minimization framework
Empirical loss minimization

• Let’s formalize the setting for learning of a *parametric model* in a supervised scenario
Empirical loss minimization

• Given: training data \( \{(x_i, y_i), i = 1, \ldots, n\} \)
• Find: predictor \( f \)
• Goal: make good predictions \( \hat{y} = f(x) \) on test data
Empirical loss minimization

- Given: training data \( \{(x_i, y_i), i = 1, \ldots, n\} \)
- Find: predictor \( f \)
- Goal: make good predictions \( \hat{y} = f(x) \) on test data

Source: Y. Liang

What kinds of functions?
Empirical loss minimization

- Given: training data \{(x_i, y_i), i = 1, ..., n\}
- Find: predictor \( f \in \mathcal{H} \)
- Goal: make good predictions \( \hat{y} = f(x) \) on test data

Source: Y. Liang
Empirical loss minimization

- Given: training data \( \{(x_i, y_i), i = 1, \ldots, n\} \)
- Find: predictor \( f \in \mathcal{H} \)
- Goal: make good predictions \( \hat{y} = f(x) \) on test data

Source: Y. Liang
Empirical loss minimization

- Given: training data \( \{(x_i, y_i), i = 1, ..., n\} \) i.i.d. from distribution \( D \)
- Find: predictor \( f \in \mathcal{H} \)
- Goal: make good predictions \( \hat{y} = f(x) \) on test data i.i.d. from distribution \( D \)

Source: Y. Liang
Empirical loss minimization

• Given: training data \( \{(x_i, y_i), i = 1, ..., n\} \) i.i.d. from distribution \( D \)
• Find: predictor \( f \in \mathcal{H} \)
• Goal: make good predictions \( \hat{y} = f(x) \) on test data i.i.d. from distribution \( D \)

Source: Y. Liang

What kind of performance measure?
Empirical loss minimization

• Given: training data \(\{(x_i, y_i), i = 1, ..., n\}\) i.i.d. from distribution \(D\)
• Find: predictor \(f \in \mathcal{H}\)
• S.t. the expected loss is small:
  \[L(f) = \mathbb{E}_{(x, y) \sim D}[l(f, x, y)]\]

Source: Y. Liang
Empirical loss minimization

- Given: training data \{ (x_i, y_i), i = 1, ..., n \} i.i.d. from distribution \( D \)
- Find: predictor \( f \in \mathcal{H} \)
- S.t. the expected loss is small:
  \[ L(f) = \mathbb{E}_{(x,y) \sim D}[l(f, x, y)] \]
- Example losses:

  0 – 1 loss: \( l(f, x, y) = \mathbb{I}[f(x) \neq y] \) and \( L(f) = \text{Pr}[f(x) \neq y] \)

  \( l_2 \) loss: \( l(f, x, y) = [f(x) - y]^2 \) and \( L(f) = \mathbb{E}[ [f(x) - y]^2 ] \)

Source: Y. Liang
Empirical loss minimization

- Given: training data \{(x_i, y_i), i = 1, ..., n\} i.i.d. from distribution \(D\)
- Find: predictor \(f \in \mathcal{H}\)
- S.t. the expected loss is small:

\[
L(f) = \mathbb{E}_{(x,y) \sim D}[l(f, x, y)]
\]

Can’t optimize this directly

Source: Y. Liang
Empirical loss minimization

- Given: training data \( \{(x_i, y_i), i = 1, \ldots, n\} \) i.i.d. from distribution \( D \)
- Find: predictor \( f \in \mathcal{H} \) that minimizes

\[
\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} l(f, x_i, y_i)
\]

Source: Y. Liang
Supervised learning in a nutshell

1. **Collect** *training data* and labels
2. **Specify model:** select *hypothesis class* and *loss function*
3. **Train model:** find the function in the hypothesis class that minimizes the *empirical loss* on the training data
Outline

• Example classification models: nearest neighbor, linear
• Empirical loss minimization
• Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
Training linear classifiers

• Given: i.i.d. training data \( \{(x_i, y_i), i = 1, \ldots, n\} \),
  \( y_i \in \{-1,1\} \)

• Hypothesis class: \( f_w(x) = \text{sgn}(w^T x) \)

• Classification with bias, i.e. \( f_w(x) = \text{sgn}(w^T x + b) \),
  can be reduced to the case w/o bias by letting
  \( \tilde{w} = [w; b] \) and \( \tilde{x} = [x; 1] \)

Source: Y. Liang
Training linear classifiers

- Given: i.i.d. training data \( \{(x_i, y_i), i = 1, ..., n\} \),
  \( y_i \in \{-1, 1\} \)
- Hypothesis class: \( f_w(x) = \text{sgn}(w^T x) \)
- Loss: how about minimizing the number of mistakes on the training data?
  \[
  \hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[\text{sgn}(w^T x_i) \neq y_i]
  \]
- Difficult to optimize directly (NP-hard), so people resort to surrogate loss functions

Source: Y. Liang
Linear regression ("straw man" model)

• Find $f_w(x) = w^T x$ that minimizes $l_2$ loss or mean squared error

$$\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

• Ignores the fact that $y \in \{-1,1\}$ but is easy to optimize

Source: Y. Liang
Linear regression: Optimization

- Let $X$ be a matrix whose $i$th row is $x_i^T$, $Y$ be the vector $(y_1, \ldots, y_n)^T$

$$
\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = \frac{1}{n} \|Xw - Y\|_2^2
$$

Source: Y. Liang
Linear regression: Optimization

- Let $X$ be a matrix whose $i$th row is $x_i^T$, $Y$ be the vector $(y_1, \ldots, y_n)^T$

  $$
  \hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = \frac{1}{n} \|Xw - Y\|_2^2
  $$

- This is a convex function of the weights

Source: Y. Liang
Linear regression: Optimization

• Let $X$ be a matrix whose $i$th row is $x_i^T$, $Y$ be the vector $(y_1, \ldots, y_n)^T$

\[
\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^n (w^T x_i - y_i)^2 = \frac{1}{n} \|Xw - Y\|_2^2
\]

• Find the gradient w.r.t. $w$:

\[
\nabla_w \|Xw - Y\|_2^2
\]

Source: Y. Liang
Linear regression: Optimization

• Let $X$ be a matrix whose $i$th row is $x_i^T$, $Y$ be the vector $(y_1, \ldots, y_n)^T$

\[
\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = \frac{1}{n} \|Xw - Y\|_2^2
\]

• Find the gradient w.r.t. $w$:

\[
\nabla_w \|Xw - Y\|_2^2 = \nabla_w [(Xw - Y)^T (Xw - Y)]
\]

\[
= \nabla_w [w^T X^T Xw - 2w^T X^T Y + Y^T Y]
\]

\[
= 2X^T Xw - 2X^T Y
\]

• Set gradient to zero to get the minimizer:

\[
X^T Xw = X^T Y
\]

\[
w = (X^T X)^{-1} X^T Y
\]

Source: Y. Liang
Linear regression: Optimization

- Linear algebra view
  - If $X$ is invertible, simply solve $Xw = Y$ and get $w = X^{-1}Y$
  - But typically $X$ is a “tall” matrix so you need to find the least squares solution to an over-constrained system

Normal equation: $w = (X^TX)^{-1}X^Ty$

Source: Y. Liang
Linear regression as maximum likelihood estimation

- Interpretation of \( l_2 \) loss: negative log likelihood assuming \( y \) is normally distributed with mean \( f_w(x) = w^T x + b \)
Maximum likelihood estimation

- Given: i.i.d. training data \( \{(x_i, y_i), i = 1, ..., n\} \)
- Let \( \{P_\theta(y|x), \theta \in \Theta\} \) be a family of distributions parameterized by \( \theta \)
- Maximum (conditional) likelihood estimate:

\[
\theta_{ML} = \arg\max_\theta \prod_i P_\theta(y_i|x_i)
\]

\[
= \arg\min_\theta - \sum_i \log P_\theta(y_i|x_i)
\]
Maximum likelihood estimation

\[ \theta_{ML} = \text{argmin}_\theta \sum_i \log P_\theta(y_i|x_i) \]

- Assume \( P_\theta(y|x) = \text{Normal}(y; f_\theta(x), \sigma^2) \)

\[
\log P_\theta(y|x) = \log \left[ \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[ -\frac{(y - f_\theta(x))^2}{2\sigma^2} \right] \right] \\
= -\frac{1}{2\sigma^2} (y - f_\theta(x))^2 - \log \sigma - \frac{1}{2} \log(2\pi) \\
\theta_{ML} = \text{argmin}_\theta \sum_i (y_i - f_\theta(x_i))^2 \]
Linear regression as maximum likelihood estimation

- Interpretation of \( l_2 \) loss: *negative log likelihood* assuming \( y \) is normally distributed with mean \( f_w(x) = w^T x + b \)

- Does this make sense for binary classification?
Problem with linear regression

- In practice, very sensitive to outliers

Figure 4.4  The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.
Problem with linear regression

- In practice, very sensitive to outliers
Next idea

- Instead of a linear function, how about we fit a function representing the confidence of the classifier?
Linear classifiers: Outline

- Example classification models: nearest neighbor, linear
- Empirical loss minimization
- Linear classification models
  1. Linear regression (least squares)
  2. Logistic regression
Logistic regression

- Let’s learn a probabilistic classifier estimating the probability of the input $x$ having a positive label, given by putting a sigmoid function around the linear response $w^T x$:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$
Sigmoid: Properties

\[ P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

- What is the range?
- What is \( \sigma(0) \)?
- What is \( P_w(y = -1|x) \)?
Sigmoid: Properties

\[ P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

- What is the range?
- What is \( \sigma(0) \)?
- What is \( P_w(y = -1|x) \)?

\[
P_w(y = -1|x) = 1 - P_w(y = 1|x) = 1 - \sigma(w^T x)
\]

\[
= \frac{1 + \exp(-w^T x) - 1}{1 + \exp(-w^T x)} = \frac{\exp(-w^T x)}{1 + \exp(-w^T x)} = \frac{1}{\exp(w^T x) + 1}
\]

\[
= \sigma(-w^T x)
\]
Sigmoid: Properties

\[ P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

- Sigmoid is *symmetric*: \( 1 - \sigma(t) = \sigma(-t) \)
Sigmoid: Properties

\[ P_w(\gamma = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

- What happens if we scale \( w \) by a constant?
Sigmoid: Properties

\[ P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \]

• What happens if we scale \( w \) by a constant?
Sigmoid: Interpretation

- We can write out the connection between the *posteriors* $P(y|x)$ and the *class-conditional densities* $P(x|y)$:

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x)}$$

$$= \frac{P(x|y = 1)P(y = 1)}{P(x|y = 1)P(y = 1) + P(x|y = -1)P(y = -1)}$$

$$= \frac{1}{1 + \exp(-a)} = \sigma(a), \quad a = \log \frac{P(y = 1|x)}{P(y = -1|x)}$$
Sigmoid: Interpretation

• Adopting a linear + sigmoid model is equivalent to assuming linear log odds:

\[
\log \frac{P(y = 1|x)}{P(y = -1|x)} = w^T x + b
\]

• This happens when \( P(x|y = 1) \) and \( P(x|y = -1) \) are Gaussians with different means and the same covariance matrices (\( w \) is related to the difference between the means)
Logistic loss

- Given: \( \{(x_i, y_i), i = 1, \ldots, n\}, y_i \in \{-1,1\} \)
- Maximum (conditional) likelihood estimate: find \( w \) that minimizes

\[
\hat{L}(w) = -\frac{1}{n} \sum_{i=1}^{n} \log P_w(y_i | x_i)
\]

\[
l(w, x_i, y_i) = -\log P_w(y_i | x_i)
\]

- If \( y_i = 1 \):
  \[
P_w(y_i | x_i) = \sigma(w^T x_i)
\]
- If \( y_i = -1 \):
  \[
P_w(y_i | x_i) = 1 - \sigma(w^T x_i) = \sigma(-w^T x_i)
\]
- Thus,
  \[
l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)
\]
Logistic loss

\[ l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i) \]
Logistic loss: Optimization

• Given: \( \{(x_i, y_i), i = 1, \ldots, n\} \), \( y_i \in \{-1, 1\} \)

• Find \( w \) that minimizes

\[
\hat{L}(w) = -\frac{1}{n} \sum_{i=1}^{n} \log P_w(y_i|x_i)
\]

• There is no closed-form expression for the minimum and we need to use \textit{gradient descent} to find it
Gradient descent

• Goal: find $w$ to minimize loss $\hat{L}(w)$
• Start with some initial estimate of $w$
• Repeat until convergence:
  • Find $\nabla \hat{L}(w)$, the gradient of the loss w.r.t. $w$
  • Take a small step in the opposite direction: $w \leftarrow w - \eta \nabla \hat{L}(w)$
The gradient vector

\[ \nabla \hat{L}(w) \]

Fastest rate of increase
The gradient vector

\[
\nabla C_L(w)
\]

Fastest rate of decrease

\[
\hat{L}(w)
\]

\[
\hat{L}
\]

\[
\nabla \hat{L}(w)
\]

\[
w
\]

\[
w^1
\]

\[
w^2
\]
Gradient descent

\[ \hat{L}(w - \eta \nabla \hat{L}(w)) \]

\[ \hat{L} \]

\[ w - \eta \nabla \hat{L}(w) \]
Gradient descent

- Goal: find $w$ to minimize loss $\hat{L}(w)$
- Start with some initial estimate of $w$
- Repeat until convergence:
  - Find $\nabla L(w)$, the gradient of the loss w.r.t. $w$
  - Take a small step in the opposite direction: $w \leftarrow w - \eta \nabla \hat{L}(w)$
Gradient descent

- Goal: find $w$ to minimize loss $\hat{L}(w)$
- Start with some initial estimate of $w$
- Repeat until convergence:
  - Find $\nabla \hat{L}(w)$, the gradient of the loss w.r.t. $w$
  - Take a small step in the opposite direction: $w \leftarrow w - \eta \nabla \hat{L}(w)$
  - $\eta$ is the step size or learning rate
Full batch gradient descent

- Since $\hat{L}(w) = \frac{1}{n} \sum_{i=1}^{n} l(w, x_i, y_i)$, we have
  \[
  \nabla \hat{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla l(w, x_i, y_i)
  \]

- For a single parameter update, need to cycle through the entire training set!
Stochastic gradient descent (SGD)

At each iteration, take a single data point \((x_i, y_i)\) and perform a parameter update using \(\nabla l(w, x_i, y_i)\), the gradient of the loss for that point:

\[
w \leftarrow w - \eta \nabla l(w, x_i, y_i)
\]

This is called an online or stochastic update.

In practice, mini-batch SGD is typically used:

- Group data into mini-batches of size \(b\)
- Compute gradient of the loss for the mini-batch \((x_1, y_1), \ldots, (x_b, y_b)\):

\[
\nabla \hat{L} = \frac{1}{b} \sum_{i=1}^{b} \nabla l(w, x_i, y_i)
\]

- Update parameters: \(w \leftarrow w - \eta \nabla \hat{L}\)
SGD for logistic regression

\[ l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i) \]

• Let’s find the gradient:
  \[
  \nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i) \\
  = -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}
  \]

• Derivative of log:
  \[
  [\log(g(a))]' = \frac{g'(a)}{g(a)}
  \]
SGD for logistic regression

\[ l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i) \]

- Let's find the gradient:
  \[
  \nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i) \\
  = - \frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)} \\
  = - \frac{\sigma(y_i w^T x_i) \sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}
  \]

Derivative of sigmoid:

\[
\sigma'(a) = \sigma(a)(1 - \sigma(a)) = \sigma(a)\sigma(-a)
\]
SGD for logistic regression

\[ l(w, x_i, y_i) = - \log \sigma(y_iw^T x_i) \]

- Let's find the gradient:

\[
\nabla l(w, x_i, y_i) = - \nabla_w \log \sigma(y_iw^T x_i) \\
= - \frac{\nabla_w \sigma(y_iw^T x_i)}{\sigma(y_iw^T x_i)} \\
= - \frac{\sigma(y_iw^T x_i) \sigma(-y_iw^T x_i)y_i x_i}{\sigma(y_iw^T x_i)}
\]

- We also used the *chain rule*: 

\[
[g_2(g_1(a))]' = g_2'(g_1(a))g_1'(a)
\]
SGD for logistic regression

\[ l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i) \]

- Let’s find the gradient:
  \[
  \nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i) \\
  = -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)} \\
  = -\frac{\sigma(y_i w^T x_i) \sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)} \\
  = -\sigma(-y_i w^T x_i) y_i x_i
  \]

- SGD update:
  \[ w \leftarrow w + \eta \sigma(-y_i w^T x_i) y_i x_i \]
SGD for logistic regression

- Let’s take a closer look at the SGD update:
  \[
  w \leftarrow w + \eta \sigma(-y_i w^T x_i) y_i x_i
  \]

- What happens if \( x_i \) is *incorrectly*, but confidently, classified?
  - The update rule approaches \( w \leftarrow w + \eta y_i x_i \)

- What happens if \( x_i \) is *correctly*, and confidently, classified?
  - The update approaches zero (but never actually reaches zero)
SGD for logistic regression

- Logistic regression does not converge for linearly separable data!
  - Scaling $w$ by ever larger constants makes the classifier more confident and keeps increasing the likelihood of the data.
SGD for logistic regression

- Logistic regression *does not converge* for linearly separable data!
- Scaling \( w \) by ever larger constants makes the classifier more confident and keeps increasing the likelihood of the data

Source: J. Johnson