Linear Classifiers

Adopted from slides by Alexander Ihler
Supervised Learning

- **Given** examples of a function \((X, Y = F(X))\)
- **Find** function \(\hat{Y} = h(X)\) to estimate \(F(X)\)
  - Discrete \(Y\): Classification
- Formulate a loss function and then adjust the parameters to minimize the loss function.
Linear regression

- Contrast with classification
  - Classify: predict discrete-valued target $y$
  - Initially: “classic” binary {-1, +1} classes; generalize later

“Predictor”:
Evaluate line:
\[ r = \theta_0 + \theta_1 x_1 \]
return $r$
Linear Classifier (2 features)

$\text{Classifier } f(x; \theta)$

$\begin{align*}
    r &= \theta_1 x_1 + \theta_2 x_2 + \theta_0 \\
    \text{"linear response"}
\end{align*}$

Threshold Function

$T(r) = \text{output} = \{\text{class decision}\}$

Visualizing for one feature “x”:

$\begin{align*}
    y &= f(x) \\
    T(f) &= \text{or, } \{0, 1\}
\end{align*}$
Perceptrons

- Perceptron = a linear classifier
  - The parameters $\theta$ are sometimes called weights ("w")
    - real-valued constants (can be positive or negative)
    - Input features $x_1 \ldots x_n$;

- A perceptron calculates 2 quantities:
  - 1. A weighted sum of the input features
  - 2. This sum is then thresholded by the $T(.)$ function

- Perceptron: a simple artificial model of human neurons
  - weights = “synapses”
  - threshold = “neuron firing”

(c) Alexander Ihler
Notations

• Inputs:
  - $x_1, x_2, \ldots, x_{n-1}, x_n$ are the values of the $n$ features
  - $x_0 = 1$ (a constant input)
  - $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_n)$: feature vector

• Weights (parameters):
  - $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$,
  - we have $n+1$ weights: one for each feature + one for the constant
  - $\mathbf{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$: parameter vector

• Linear response
  - $\theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \mathbf{\theta} \cdot \mathbf{x}$

• Threshold function
  - $T(r)$

• Linear classifier
  - $f(x; \mathbf{\theta}) = T(\mathbf{\theta} \cdot \mathbf{x})$
Nearest neighbor classifier

![Decision Boundary]

All points where we decide 1

Decision Boundary

All points where we decide 0
Example: Gaussian Bayes for Iris Data

• Fit Gaussian distribution to each class \{0,1,2\}

\[
p(y) = \text{Discrete}\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)
\]

\[
p(x_1, x_2 | y = 0) = \mathcal{N}(x; \mu_0, \Sigma_0)
\]

\[
p(x_1, x_2 | y = 1) = \mathcal{N}(x; \mu_1, \Sigma_1)
\]

\[
p(x_1, x_2 | y = 2) = \mathcal{N}(x; \mu_2, \Sigma_2)
\]
Decision Trees for classification
Perceptron Decision Boundary

• The perceptron is defined by the decision algorithm:

\[
f(x; \theta) = \begin{cases} 
+1 & \text{if } \theta \cdot x^T > 0 \\
-1 & \text{otherwise}
\end{cases}
\]

or \( f(x; \theta) = T(\theta x) \)

• The perceptron represents a hyperplane decision surface in n-dimensional space
  – A point in 1D, a line in 2D, a plane in 3D, etc.

• The equation of the hyperplane is given by

\[
\theta \cdot x^T = 0
\]

This defines the set of points that are on the boundary.
Example, Linear Decision Boundary

\[ \theta = (\theta_0, \theta_1, \theta_2) = (1, .5, -.5) \]
Example, Linear Decision Boundary

\[ \theta = (\theta_0, \theta_1, \theta_2) = (1, .5, -.5) \]

\[ \theta \cdot x' = 0 \]

\[ \Rightarrow .5 \cdot x_1 - .5 \cdot x_2 + 1 \cdot 1 = 0 \]

\[ \Rightarrow -.5 \cdot x_2 = -.5 \cdot x_1 - 1 \]

\[ \Rightarrow x_2 = x_1 + 2 \]
Example, Linear Decision Boundary

\[ \theta \cdot x' = 0 \]

\[ \theta = (\theta_0, \theta_1, \theta_2) \]
\[ = (1, .5, -.5) \]

\[ \theta \cdot x' < 0 \]
\[ \Rightarrow x_1 + 2 < x_2 \]
(this is the equation for decision region -1)

\[ \theta \cdot x' > 0 \]
\[ \Rightarrow x_1 + 2 > x_2 \]
(decision region +1)

From P. Smyth
Separability

- A data set is separable by a learner if
  - There is some instance of that learner that correctly predicts all the data points
- Linearly separable data
  - Can separate the two classes using a straight line in feature space
  - in 2 dimensions the decision boundary is a straight line
Another example
Non-linear decision boundary

(c) Alexander Ihler
Adding features

• Linear classifier can’t learn some functions

1D example:

Not linearly separable

Add quadratic features

$\begin{align*}
  x_2 &= (x_1)^2 \\
  x_1 &
\end{align*}$

Linearly separable in new features...
Adding features

• Linear classifier can’t learn some functions

1D example:

\[ y = T(ax^2 + bx + c) \]

More complex decision boundary: \[ ax^2 + bx + c = 0 \]
Effect of dimensionality

• Data are increasingly separable in high dimension – is this a good thing?
• “Good”
  • Separation is easier in higher dimensions (for fixed # of data m)
  • Increase the number of features, and even a linear classifier will eventually be able to separate all the training examples!
• “Bad”
  • Remember training vs. test error? Remember overfitting?
  • Increasingly complex decision boundaries can eventually get all the training data right, but it doesn’t necessarily bode well for test data…

![Graph showing the relationship between predictive error, complexity, error on training data, and error on test data. The graph illustrates the ideal range, underfitting, and overfitting.](image)
Feature representations

• Features are used in a linear way
• Learner is dependent on representation

• Ex: discrete features
  • Mushroom surface: \{fibrous, grooves, scaly, smooth\}
  • Probably not useful to use \( x = \{1, 2, 3, 4\} \)
  • Better: 1-of-K, \( x = \{[1000], [0100], [0010], [0001]\} \)
  • Introduces more parameters, but a more flexible relationship
Linear Classifiers: Learning
Learning the Classifier Parameters

• Learning from Training Data:
  • training data = labeled feature vectors
  • Find parameter values that predict well (low error)
    • error is estimated on the training data
    • “true” error will be on future test data

• Define a loss function  $J(\theta)$:
  • Classifier error rate (for a given set of weights $\theta$ and labeled data)

• Minimize this loss function (or, maximize accuracy)
  • An optimization or search problem over the vector $(\theta_1, \theta_2, \theta_0)$
Training a linear classifier

• How should we measure error?
  • Natural measure = “fraction we get wrong” (error rate)
    \[
    \text{err}(\theta) = \frac{1}{m} \sum_i 1[y^{(i)} \neq f(x^{(i)}; \theta)]
    \]
    where \(1[y \neq \hat{y}] = \begin{cases} 1 & y \neq \hat{y} \\ 0 & \text{o.w.} \end{cases} \)

• But, hard to train via gradient descent
  • Not continuous
  • As decision boundary moves, errors change abruptly

1D example:

\[
T(f) = \begin{cases} -1 & \text{if } f < 0 \\ +1 & \text{if } f > 0 \end{cases}
\]
Linear regression?

• Simple option: set $\theta$ using linear regression

• In practice, this often doesn’t work so well...
  – Consider adding a distant but “easy” point
  – MSE distorts the solution

(c) Alexander Ihler
Perceptron algorithm

\[
\text{while } \neg \text{ done:} \\
\text{for each data point } j:\ \\
\hat{y}(j) = \text{sign}(\theta \cdot x^{(j)}) \\
\theta \leftarrow \theta + \alpha (y^{(j)} - \hat{y}(j)) x^{(j)}
\]

(predict output for point j)

(“gradient-like” step)
Perceptron algorithm

while ! done:
    for each data point $j$:
        $\hat{y}(j) = \text{sign}(\theta \cdot x(j))$
        $\theta \leftarrow \theta + \alpha (y(j) - \hat{y}(j)) x(j)$
(predict output for point $j$)
("gradient-like" step)

(c) Alexander Ihler
Perceptron algorithm

while not done:
    for each data point $j$:
        $\hat{y}(j) = \text{sign}(\theta \cdot x(j))$  \hspace{1cm} (predict output for point $j$)
        $\theta \leftarrow \theta + \alpha (y(j) - \hat{y}(j))x(j)$  \hspace{1cm} ("gradient-like" step)

(c) Alexander Ihler
Perceptron Convergence

• **Perceptron Convergence Theorem:**
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge

• **Perceptron Cycling Theorem:**
  - If the training data is not linearly separable the Perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop
Surrogate functions

• Another solution: use a “smooth” threshold
  • e.g., approximate the threshold function

• Usually some smooth function of distance
  • Example: logistic “sigmoid”, looks like an “S”

• Now, measure e.g. MSE

\[
J(\theta) = \frac{1}{m} \sum_j \left( \sigma(r(x(j))) - y(j) \right)^2
\]

1D example:

Classification error = 2/9

MSE = (0² + 1² + .2² + .25² + .05² + …)/9
Surrogate loss functions

Class $y = \{-1, 1\}$

$$J(\theta) = \frac{1}{m} \sum_i 1[y^{(i)} \neq \text{sign}(f(x^{(i)}; \theta))]$$

$$= \frac{1}{m} \sum_i 1[y^{(i)} \cdot f(x^{(i)}; \theta) < 0]$$

$$= \frac{1}{m} \sum_i L[y^{(i)} \cdot f(x^{(i)}; \theta)]$$

0 / 1 Loss
Surrogate loss functions

Class \( y = \{0, 1\} \)

\[
J(\theta) = \frac{1}{m} \sum_i 1[y^{(i)} \neq 1[f(x^{(i)}; \theta) > 0]]
\]

\[
= \frac{1}{m} \sum_i (y^{(i)} 1[f(x^{(i)}; \theta) < 0] + (1 - y^{(i)}) 1[f(x^{(i)}; \theta) > 0])
\]

\[
= \frac{1}{m} \sum_i \left( y^{(i)} L(f(x^{(i)}; \theta)) + (1 - y^{(i)}) L(-f(x^{(i)}; \theta)) \right)
\]
Surrogate loss functions

\[ \phi(z) \]

Convex!

Figure 1. A Plot of the 0–1 Loss Function and Surrogates Corresponding to Various Practical Classifiers (--- 0–1; —— exponential; — hinge; —— logistic; ······ truncated quadratic). These functions are
Generic classification formulation

Class $y = \{0, 1\}$

$$J(\theta) = \frac{1}{m} \sum_{i} \left( y^{(i)} \phi \left( f(x^{(i)}; \theta) \right) + (1 - y^{(i)}) \phi \left( -f(x^{(i)}; \theta) \right) \right)$$

Class $y = \{-1, 1\}$

$$J(\theta) = \frac{1}{m} \sum_{i} \phi \left( y^{(i)} f(x^{(i)}; \theta) \right)$$
Generic classification formulation w/ Regularization

Class $y = \{0, 1\}$

$$J(\theta) = \frac{1}{m} \sum_i \left( y^{(i)} \phi \left( f(x^{(i)}; \theta) \right) + (1 - y^{(i)}) \phi \left( -f(x^{(i)}; \theta) \right) \right) + \frac{\lambda}{m} |\theta|_2$$

Class $y = \{-1, 1\}$

$$J(\theta) = \frac{1}{m} \sum_i \phi \left( y^{(i)} f(x^{(i)}; \theta) \right) + \frac{\lambda}{m} |\theta|_2$$
Training the Classifier

• Once we have a smooth measure of quality, we can find the “best” settings for the parameters

• Example: 2D feature space $\Leftrightarrow$ parameter space

(c) Alexander Ihler
Training the Classifier

- Once we have a smooth measure of quality, we can find the “best” settings for the parameters

- Example: 2D feature space $\iff$ parameter space

(c) Alexander Ihler
Minimizing the loss function

- As in linear regression, this is now just optimization

- Methods:
  - Gradient descent
    - Improve loss by small changes in parameters ("small" = learning rate)
Gradient equations

\[
\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_i \left( y^{(i)} \frac{\partial \phi \left( f(x^{(i)}; \theta) \right)}{\partial \theta_j} + (1 - y^{(i)}) \frac{\partial \phi \left( -f(x^{(i)}; \theta) \right)}{\partial \theta_j} \right)
\]

Class $y = \{0, 1\}$

\[
\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_i \frac{\partial \phi \left( y^{(i)} f(x^{(i)}; \theta) \right)}{\partial \theta_j}
\]

Class $y = \{-1, 1\}$
Maximum likelihood learning

\[ p(D)p(W \mid D) = p(D, W) = p(W)p(D \mid W) \]

- **Joint probability**
  - \( p(D)p(W \mid D) \)
  - \( p(D, W) \)
  - \( p(W)p(D \mid W) \)

- **Conditional probability**
  - \( p(W \mid D) \)
  - \( p(D \mid W) \)

- **Joint probability**
  - \( p(W)p(D \mid W) \)

- **Prior probability of weight vector \( W \)**
  - \( p(W) \)

- **Posterior probability of weight vector \( W \) given training data \( D \)**
  - \( p(W \mid D) \)

- **Probability of observed data given \( W \)**
  - \( p(D \mid W) \)

- **Probability of observed data given \( W \)**
  - \( p(D) \)

- **Integrating over \( W \)**
  - \( \int_W p(W)p(D \mid W) \)
Maximize sums of log probs

• We want to maximize the product of the probabilities of the outputs on the training cases
  • Assume the output errors on different training cases, $i$, are independent.

\[
p(D|W) = \prod_{i} p(d^{(i)} |W)
\]

• Because the log function is monotonic, it does not change where the maxima are. So we can maximize sums of log probabilities

\[
\log p (D|W) = \sum_{i} \log p (d^{(i)} |W)
\]

• This is called maximum likelihood learning.

Minimum negative log-likelihood:  
\[
- \log p (D|W) = - \sum_{i} \log p (d^{(i)} |W)
\]
Logistic regression

• Another solution: use a “smooth” threshold
  • e.g., approximate the threshold function

• Usually some smooth function of distance
  • Example: logistic “sigmoid”, looks like an “S”

\[ T(\theta x) \Rightarrow \sigma(\theta x) \]

\[ \sigma(z) = \frac{1}{1 + e^{-z}} \]
Logistic regression

• Interpret \( \sigma(\theta x) \) as a probability that \( y = 1 \), i.e., \( P(Y = 1|x) = \sigma(\theta x) \)

• Use a negative log-likelihood function
  - If \( y = 1 \), loss is \( -\log P[y = 1] = -\log \sigma(\theta x) \)
  - If \( y = 0 \), loss is \( -\log P[y = 0] = -\log(1 - \sigma(\theta x)) \)

• Can write this succinctly:

\[
J(\theta) = -\frac{1}{m} \left( \sum_{i} y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1-y^{(i)}) \log(1-\sigma(\theta \cdot x^{(i)})) \right)
\]

Nonzero only if \( y=1 \)  
Nonzero only if \( y=0 \)
Logistic regression

- Interpret $\sigma(\theta x)$ as a probability that $y = 1$
- Use a negative log-likelihood loss function
  - If $y = 1$, cost is $- \log P[y = 1] = - \log \sigma(\theta x)$
  - If $y = 0$, cost is $- \log P[y = 0] = - \log(1 - \sigma(\theta x))$
- Can write this succinctly:

$$J(\theta) = -\frac{1}{m} \left( \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta \cdot x^{(i)})) \right)$$

1D example:

Classification error $= 2/9$

NLL $= - (\log(0.99) + \log(0.97) + \ldots)/9$
Logistic regression

\[ J(\theta) = -\frac{1}{m} \left( \sum_{i} y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta \cdot x^{(i)})) \right) \]

\[ J(\theta) = \frac{1}{m} \sum_{i} \left( y^{(i)} \phi(f(x^{(i)}; \theta)) + (1 - y^{(i)}) \phi(-f(x^{(i)}; \theta)) \right) \]

\[ \phi(z) = -\log(\sigma(z)) = -\log \frac{1}{1 + e^{-z}} \]

\[ \phi(-z) = -\log(\sigma(-z)) = -\log \frac{1}{1 + e^{z}} = -\log \frac{e^{-z}}{1 + e^{-z}} = -\log \left( 1 - \frac{1}{1 + e^{-z}} \right) = -\log(1 - \sigma(z)) \]
Gradient Equations

\[
\frac{1}{z'} = \frac{1}{z}
\]

\[
(\sigma(z))' = \sigma(z)(1 - \sigma(z))
\]

- Logistic neg-log likelihood loss:

\[
J(\theta) = -\frac{1}{m} \left( \sum_i y^{(i)} \log \sigma(\theta \cdot x^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta \cdot x^{(i)})) \right)
\]

- What’s the derivative with respect to one of the parameters?

\[
\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m} \sum_i \left( y^{(i)} \left(1 - \sigma(\theta x^{(i)}) \right) x_j^{(i)} - (1 - y^{(i)}) \sigma(\theta x^{(i)}) x_j^{(i)} \right)
\]

\[
= \frac{1}{m} \sum_i \left( \sigma(\theta x^{(i)}) - y^{(i)} \right) x_j^{(i)}
\]
(Batch) Gradient descent

Initialize $\theta$
Do {
    $\theta \leftarrow \theta - \alpha \nabla J(\theta)$
} while (stop condition)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left( y^{(i)} \log \sigma(\theta x^{(i)}) + (1 - y^{(i)}) \log \left(1 - \sigma(\theta x^{(i)})\right)\right)$$

$$\frac{\partial J(\theta)}{\theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left( \sigma(\theta x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

(c) Alexander Ihler
Stochastic gradient descent

• Instead of evaluating gradient over all examples evaluate it for each individual training example

\[
J^{(i)}(\theta) = y^{(i)} \log \sigma(\theta x^{(i)}) + (1 - y^{(i)}) \log (1 - \sigma(\theta x^{(i)}))
\]

\[
\frac{\partial J^{(i)}(\theta)}{\theta_j} = (\sigma(\theta x^{(i)}) - y^{(i)}) x_j^{(i)}
\]
Perceptron algorithm

while ¬ done:
   for each data point $j$:
      $\hat{y}(j) = \text{sign}(\theta \cdot x^{(j)})$  \hspace{1cm} \text{(predict output for point j)}
      \[ \theta \leftarrow \theta + \alpha (y^{(j)} - \hat{y}(j)) x^{(j)} \]  \hspace{1cm} \text{("gradient-like" step)}

(c) Alexander Ihler
Stochastic gradient descent

• Update based on each datum at a time
  • Find residual and the gradient of its part of the error & update

Initialize $\theta$

Do {
  for $i = 1 : m$
    $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)
Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

Initialize $\theta$

Do {
  for $i = 1 : m$
    $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)
Stochastic gradient descent

• Update based on each datum at a time
  • Find residual and the gradient of its part of the error & update

Initialize $\theta$
Do {
  for $i = 1 : m$
    $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)
Stochastic gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

Initialize $\theta$

Do {
  for $i = 1 : m$
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
}
while (stop condition)
Stochastic gradient descent

• Benefits
  • Lots of data = many more updates per pass
  • Computationally faster

• Drawbacks
  • No longer strictly “descent”
  • Stopping conditions may be harder to evaluate
    (Can use “running estimates” of J(.), etc.)

• Related: mini-batch updates, etc.

Initialize $\theta$
Do{
  for $i = 1 : m$
  $\theta \leftarrow \theta - \alpha \nabla J^{(i)}(\theta)$
} while (stop condition)
Multi-class linear models

- What about multiple classes? One option:
  - Define one linear response per class
  - Choose class with the largest response

\[ f(x; \theta) = \arg \max_c \theta_c \cdot x^T \]

- Boundary between two classes, c vs. c’?

\[
\begin{cases}
    c & \text{if } \theta_c \cdot x^T > \theta_{c'} x^T \\
    c' & \text{otherwise}
\end{cases}
\]

\[ \iff (\theta_c - \theta_{c'}) x^T > 0 \]

- Linear boundary: \((\theta_c - \theta_{c'}) x^T = 0\)
Multiclass logistic regression

• Define the probability of each class (softmax function):

\[ p(Y = y | X = x) = \frac{\exp(\theta_y \cdot x^T)}{\sum_c \exp(\theta_c \cdot x^T)} \]

• Then, the NLL loss function is:

\[ J(\theta) = -\frac{1}{m} \sum_i \log p(y^{(i)} | x^{(i)}) = -\frac{1}{m} \sum_i \left[ \theta_{y^{(i)}} \cdot x^{(i)} - \log \sum_c \exp(\theta_c \cdot x^{(i)}) \right] \]

(Y binary = logistic regression)

• P: “confidence” of each class
  • Soft decision value
• Decision: predict most probable
  • Linear decision boundary
• Convex loss function

(c) Alexander Ihler
Summary

- Linear classifier $\Leftrightarrow$ perceptron

- Measuring quality of a decision boundary
  - Error rate (0/1 loss)
  - Surrogate functions

- Learning the weights of a linear classifier from data
  - Reduces to an optimization problem
  - Perceptron algorithm
  - Using surrogate functions, we can do gradient descent
  - Gradient equations & update rules (BGD and SGD)
  - Multiclass logistic regression (softmax function)