Logistic regression and linear classifiers

COMS 4771
1. Prediction functions (again)
Learning prediction functions

IID model for *supervised learning*:

\((X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)\) are iid random *pairs* (i.e., *labeled examples*).
Learning prediction functions

**IID model for supervised learning:**

$(X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)$ are iid random *pairs* (i.e., *labeled examples*).

- $X$ takes values in $\mathcal{X}$. E.g., $\mathcal{X} = \mathbb{R}^d$.
- $Y$ takes values in $\mathcal{Y}$. E.g.,
  - *(regression problems)* $\mathcal{Y} = \mathbb{R}$;
  - *(classification problems)* $\mathcal{Y} = \{1, \ldots, K\}$ or $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$.

---

1. We observe $(X_1, Y_1), \ldots, (X_n, Y_n)$, and choose a prediction function (i.e., *predictor*) $\hat{f} : \mathcal{X} \rightarrow \mathcal{Y}$.

2. At prediction time, observe $X$, and form prediction $\hat{f}(X)$.

3. Outcome is $Y$,

   - **Squared loss**: $(\hat{f}(X) - Y)^2$ (regression problems).
   - **Zero-one loss**: $1\{\hat{f}(X) \neq Y\}$ (classification problems).

Note: expected zero-one loss is $E[1\{\hat{f}(X) \neq Y\}] = P(\hat{f}(X) \neq Y)$, which we also call *error rate*. 
Learning prediction functions

**IID model** for *supervised learning*:

\((X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)\) are iid random *pairs* (i.e., *labeled examples*).

- **X** takes values in \(\mathcal{X}\). E.g., \(\mathcal{X} = \mathbb{R}^d\).
- **Y** takes values in \(\mathcal{Y}\). E.g.,
  - *(regression problems)* \(\mathcal{Y} = \mathbb{R}\);
  - *(classification problems)* \(\mathcal{Y} = \{1, \ldots, K\}\) or \(\mathcal{Y} = \{0, 1\}\) or \(\mathcal{Y} = \{-1, +1\}\).

1. We observe \((X_1, Y_1), \ldots, (X_n, Y_n)\), and the choose a *prediction function* (i.e., *predictor*)

\[ \hat{f} : \mathcal{X} \rightarrow \mathcal{Y}, \]

This is called “*learning*” or “*training*”.

Note: expected zero-one loss is 
\[ E \left[ 1 \{ \hat{f}(X) \neq Y \} \right] = P(\hat{f}(X) \neq Y), \]
which we also call *error rate*. 
Learning prediction functions

**IID model for supervised learning:**

$(X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)$ are iid random *pairs* (i.e., *labeled examples*).

- $X$ takes values in $\mathcal{X}$. E.g., $\mathcal{X} = \mathbb{R}^d$.
- $Y$ takes values in $\mathcal{Y}$. E.g.,
  - *(regression problems)* $\mathcal{Y} = \mathbb{R}$;
  - *(classification problems)* $\mathcal{Y} = \{1, \ldots, K\}$ or $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$.

1. We observe $(X_1, Y_1), \ldots, (X_n, Y_n)$, and the choose a *prediction function* (i.e., *predictor*)

   \[ \hat{f}: \mathcal{X} \to \mathcal{Y}, \]

   This is called “learning” or “training”.

2. At prediction time, observe $X$, and form prediction $\hat{f}(X)$. 
Learning prediction functions

**IID model for supervised learning:**

\((X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)\) are iid random pairs (i.e., labeled examples).

- \(X\) takes values in \(\mathcal{X}\). E.g., \(\mathcal{X} = \mathbb{R}^d\).
- \(Y\) takes values in \(\mathcal{Y}\). E.g.,
  - (regression problems) \(\mathcal{Y} = \mathbb{R}\);
  - (classification problems) \(\mathcal{Y} = \{1, \ldots, K\}\) or \(\mathcal{Y} = \{0, 1\}\) or \(\mathcal{Y} = \{-1, +1\}\).

1. We observe \((X_1, Y_1), \ldots, (X_n, Y_n)\), and choose a prediction function (i.e., predictor)

\[ \hat{f} : \mathcal{X} \to \mathcal{Y}, \]

This is called "learning" or "training".

2. At prediction time, observe \(X\), and form prediction \(\hat{f}(X)\).

3. Outcome is \(Y\), and
   - squared loss is \((\hat{f}(X) - Y)^2\) (regression problems).
   - zero-one loss is \(1\{\hat{f}(X) \neq Y\}\) (classification problems).
Learning prediction functions

**IID model** for *supervised learning*:

\((X_1, Y_1), \ldots, (X_n, Y_n), (X, Y)\) are iid random *pairs* (i.e., *labeled examples*).

- \(X\) takes values in \(X\). E.g., \(X = \mathbb{R}^d\).
- \(Y\) takes values in \(Y\). E.g.,
  - *(regression problems)* \(Y = \mathbb{R}\);
  - *(classification problems)* \(Y = \{1, \ldots, K\}\) or \(Y = \{0, 1\}\) or \(Y = \{-1, +1\}\).

1. We observe \((X_1, Y_1), \ldots, (X_n, Y_n)\), and choose a *prediction function* (i.e., *predictor*)

\[
\hat{f} : X \rightarrow Y,
\]

This is called “*learning*” or “*training*”.

2. At prediction time, observe \(X\), and form prediction \(\hat{f}(X)\).

3. Outcome is \(Y\), and

- *squared loss* is \((\hat{f}(X) - Y)^2\) (regression problems).
- *zero-one loss* is \(1\{\hat{f}(X) \neq Y\}\) (classification problems).

**Note**: expected zero-one loss is

\[
\mathbb{E}[1\{\hat{f}(X) \neq Y\}] = \mathbb{P}(\hat{f}(X) \neq Y),
\]

which we also call *error rate*. 
Distributions over labeled examples

\(\mathcal{X}\): space of possible side-information (feature space).

\(\mathcal{Y}\): space of possible outcomes (label space or output space).
Distributions over labeled examples

\( \mathcal{X} \): space of possible side-information (\textit{feature space}).
\( \mathcal{Y} \): space of possible outcomes (\textit{label space} or \textit{output space}).

Distribution \( P \) of random pair \((X, Y)\) taking values in \( \mathcal{X} \times \mathcal{Y} \) can be thought of in two parts:
$\mathcal{X}$: space of possible side-information (feature space).
$\mathcal{Y}$: space of possible outcomes (label space or output space).

Distribution $P$ of random pair $(X, Y)$ taking values in $\mathcal{X} \times \mathcal{Y}$ can be thought of in two parts:

1. **Marginal distribution** $P_X$ of $X$:
   
   $P_X$ is a probability distribution on $\mathcal{X}$. 
Distributions over labeled examples

\( \mathcal{X} \): space of possible side-information (feature space).
\( \mathcal{Y} \): space of possible outcomes (label space or output space).

Distribution \( P \) of random pair \((X, Y)\) taking values in \( \mathcal{X} \times \mathcal{Y} \) can be thought of in two parts:

1. **Marginal distribution** \( P_X \) of \( X \):
   \( P_X \) is a probability distribution on \( \mathcal{X} \).

2. **Conditional distribution** \( P_{Y|X=x} \) of \( Y \) given \( X = x \), for each \( x \in \mathcal{X} \):
   \( P_{Y|X=x} \) is a probability distribution on \( \mathcal{Y} \).
For binary classification, what function $f: \mathcal{X} \rightarrow \{0, 1\}$ has smallest risk (i.e., error rate) $\mathcal{R}(f) := \mathbb{P}(f(X) \neq Y)$?
For binary classification, what function $f : \mathcal{X} \to \{0, 1\}$ has smallest risk (i.e., error rate) $R(f) := \mathbb{P}(f(X) \neq Y)$?

▶ Conditional on $X = x$, the minimizer of conditional risk

\[
\hat{y} \mapsto \mathbb{P}(\hat{y} \neq Y \mid X = x)
\]

is

\[
\hat{y} := \begin{cases} 
1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\
0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2.
\end{cases}
\]
For binary classification, what function $f: \mathcal{X} \to \{0, 1\}$ has smallest risk (i.e., error rate) $R(f) := \mathbb{P}(f(X) \neq Y)$?

- Conditional on $X = x$, the minimizer of conditional risk

$$\hat{y} \mapsto \mathbb{P}(\hat{y} \neq Y \mid X = x)$$

is

$$\hat{y} := \begin{cases} 1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\ 0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2. \end{cases}$$

- Therefore, the function $f^*: \mathcal{X} \to \{0, 1\}$ where

$$f^*(x) = \begin{cases} 1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\ 0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2, \end{cases} \quad x \in \mathcal{X},$$

has the smallest risk.
For binary classification, what function \( f : \mathcal{X} \to \{0, 1\} \) has smallest risk (i.e., error rate) \( \mathcal{R}(f) := \mathbb{P}(f(X) \neq Y) \)?

- Conditional on \( X = x \), the minimizer of conditional risk
  \[
  \hat{y} \mapsto \mathbb{P}(\hat{y} \neq Y \mid X = x)
  \]
  is
  \[
  \hat{y} := \begin{cases} 
  1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\
  0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2.
  \end{cases}
  \]

- Therefore, the function \( f^* : \mathcal{X} \to \{0, 1\} \) where
  \[
  f^*(x) = \begin{cases} 
  1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\
  0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2,
  \end{cases} \quad x \in \mathcal{X},
  \]
  has the smallest risk.

- \( f^* \) is called the Bayes (optimal) classifier.
For binary classification, what function \( f : \mathcal{X} \rightarrow \{0, 1\} \) has smallest risk (i.e., error rate) \( R(f) := \mathbb{P}(f(X) \neq Y) \)?

- Conditional on \( X = x \), the minimizer of conditional risk
  \[
  \hat{y} \mapsto \mathbb{P}(\hat{y} \neq Y \mid X = x)
  \]
  is
  \[
  \hat{y} := \begin{cases} 
  1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\
  0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2.
  \end{cases}
  \]

- Therefore, the function \( f^* : \mathcal{X} \rightarrow \{0, 1\} \) where
  \[
  f^*(x) = \begin{cases} 
  1 & \text{if } \mathbb{P}(Y = 1 \mid X = x) > 1/2, \\
  0 & \text{if } \mathbb{P}(Y = 1 \mid X = x) \leq 1/2,
  \end{cases} \quad x \in \mathcal{X},
  \]
  has the smallest risk.

- \( f^* \) is called the Bayes (optimal) classifier.

For \( \mathcal{Y} = \{1, \ldots, K\} \),
\[
 f^*(x) = \arg \max_{y \in \mathcal{Y}} \mathbb{P}(Y = y \mid X = x), \quad x \in \mathcal{X}.
\]
2. Logistic regression
Logistic regression

Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$. A logistic regression model is a statistical model where the conditional probability function has a particular form:

$$Y \mid X = x \sim \text{Bern}(\eta_\beta(x)), \quad x \in \mathbb{R}^d,$$

with

$$\eta_\beta(x) := \text{logistic}(x^T \beta), \quad x \in \mathbb{R}^d,$$

and

$$\text{logistic}(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}, \quad z \in \mathbb{R}.$$
Logistic regression

Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$. A **logistic regression model** is a statistical model where the conditional probability function has a particular form:

$$Y \mid X = x \sim \text{Bern}(\eta_\beta(x)), \quad x \in \mathbb{R}^d,$$

with

$$\eta_\beta(x) := \text{logistic}(x^T \beta), \quad x \in \mathbb{R}^d,$$

and

$$\text{logistic}(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}, \quad z \in \mathbb{R}.$$
Logistic regression

Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$.

A *logistic regression model* is a statistical model where the conditional probability function has a particular form:

$$ Y \mid \mathbf{X} = \mathbf{x} \sim \text{Bern}(\eta_{\beta}(\mathbf{x})), \quad \mathbf{x} \in \mathbb{R}^d, $$

with

$$ \eta_{\beta}(\mathbf{x}) := \text{logistic}(\mathbf{x}^T \beta), \quad \mathbf{x} \in \mathbb{R}^d, $$

and

$$ \text{logistic}(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}, \quad z \in \mathbb{R}. $$

- Parameters: $\beta = (\beta_1, \ldots, \beta_d) \in \mathbb{R}^d$. 

![Image of logistic function graph](image-url)
Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$.

A **logistic regression model** is a statistical model where the conditional probability function has a particular form:

$$Y \mid X = x \sim \text{Bern}(\eta_\beta(x)), \quad x \in \mathbb{R}^d,$$

with

$$\eta_\beta(x) := \text{logistic}(x^T \beta), \quad x \in \mathbb{R}^d,$$

and

$$\text{logistic}(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}, \quad z \in \mathbb{R}.$$

- **Parameters:** $\beta = (\beta_1, \ldots, \beta_d) \in \mathbb{R}^d$.
- **Conditional distribution of $Y$ given $X$ is Bernoulli; marginal distribution of $X$ not specified.**
MLE for logistic regression

Log-likelihood of $\beta$ in iid logistic regression model, given data $(X_i, Y_i) = (x_i, y_i)$ for $i = 1, \ldots, n$:

$$\ln \prod_{i=1}^{n} \eta_{\beta}(x_i)^{y_i} (1 - \eta_{\beta}(x_i))^{1-y_i} = \sum_{i=1}^{n} y_i \ln \eta_{\beta}(x_i) + (1 - y_i) \ln(1 - \eta_{\beta}(x_i)).$$
MLE for logistic regression

Log-likelihood of $\beta$ in iid logistic regression model, given data $(X_i, Y_i) = (x_i, y_i)$ for $i = 1, \ldots, n$:

$$\ln \prod_{i=1}^{n} \eta_{\beta}(x_i)^{y_i} \left(1 - \eta_{\beta}(x_i)\right)^{1-y_i} = \sum_{i=1}^{n} y_i \ln \eta_{\beta}(x_i) + (1 - y_i) \ln(1 - \eta_{\beta}(x_i)).$$

- No closed-form formula for MLE.
MLE for logistic regression

Log-likelihood of $\beta$ in iid logistic regression model, given data $(X_i, Y_i) = (x_i, y_i)$ for $i = 1, \ldots, n$:

$$\ln \prod_{i=1}^{n} \eta_\beta(x_i)^{y_i} (1 - \eta_\beta(x_i))^{1-y_i} = \sum_{i=1}^{n} y_i \ln \eta_\beta(x_i) + (1 - y_i) \ln(1 - \eta_\beta(x_i)).$$

- No closed-form formula for MLE.
- Nevertheless, there are efficient algorithms that obtain an approximate maximizer of the log-likelihood function.
  E.g., Newton’s method.
Equivalent way to characterize logistic regression model:

The *log-odds function*, given by

\[
\text{log-odds}_\beta(x) = \ln \frac{\eta_\beta(x)}{1 - \eta_\beta(x)} = \ln \left( \frac{e^{x^T \beta}}{1 + e^{x^T \beta}} \right) = x^T \beta,
\]

is a linear function\(^1\), parameterized by \(\beta \in \mathbb{R}^d\).

\(^1\)Some authors allow affine function; we can get this using affine expansion.
Equivalent way to characterize logistic regression model:
The *log-odds function*, given by

\[
\text{log-odds}_\beta(x) = \ln \frac{\eta_\beta(x)}{1 - \eta_\beta(x)} = \ln \left( \frac{e^{x^T \beta}}{1 + e^{x^T \beta}} \right) = x^T \beta,
\]

is a linear function\(^1\), parameterized by \( \beta \in \mathbb{R}^d \).

Bayes optimal classifier \( f_\beta : \mathbb{R}^d \to \{0, 1\} \) in logistic regression model:

\[
f_\beta(x) = \begin{cases} 
0 & \text{if } x^T \beta \leq 0, \\
1 & \text{if } x^T \beta > 0.
\end{cases}
\]

\(^1\)Some authors allow affine function; we can get this using affine expansion.
Log-odds function and classifier

Equivalent way to characterize logistic regression model:
The *log-odds function*, given by

\[
\text{log-odds}_\beta(x) = \ln \frac{\eta_\beta(x)}{1 - \eta_\beta(x)} = \ln \left( \frac{e^{x^T \beta}}{1 + e^{x^T \beta}} \right) = x^T \beta,
\]

is a linear function\(^1\), parameterized by \(\beta \in \mathbb{R}^d\).

Bayes optimal classifier \(f_\beta : \mathbb{R}^d \rightarrow \{0, 1\}\) in logistic regression model:

\[
f_\beta(x) = \begin{cases} 
0 & \text{if } x^T \beta \leq 0, \\
1 & \text{if } x^T \beta > 0.
\end{cases}
\]

Such classifiers are called *linear classifiers*.

---

\(^1\)Some authors allow affine function; we can get this using affine expansion.
3. Origin story for logistic regression
Where does the logistic regression model come from?

The following is one way the logistic regression model comes about (but not the only way).
The following is one way the logistic regression model comes about (but not the only way).

Consider the following generative model for \((X, Y)\) where

\[
Y \sim \text{Bern}(\pi),
\]

\[
X \mid Y = y \sim N(\mu_y, \Sigma).
\]
Where does the logistic regression model come from?

The following is one way the logistic regression model comes about (but not the only way).

Consider the following generative model for \((X, Y)\) where

\[
Y \sim \text{Bern}(\pi),
\]

\[
X \mid Y = y \sim \text{N}(\mu_y, \Sigma).
\]

- Parameters: \(\pi \in [0, 1], \mu_0, \mu_1 \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d} \text{ sym.} & \text{pos. def.}\)
Where does the logistic regression model come from?

The following is one way the logistic regression model comes about (but not the only way).

Consider the following generative model for \((X, Y)\) where

\[
Y \sim \text{Bern}(\pi),
\]

\[
X \mid Y = y \sim N(\mu_y, \Sigma).
\]

- Parameters: \(\pi \in [0, 1], \mu_0, \mu_1 \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d} \text{ sym. \& pos. def.}\)

Figure shows (unconditional) probability density function for \(X\).
Statistical model for conditional distribution

Suppose we are given the following.

- \( p_Y \): probability mass function for \( Y \).
- \( p_{X|Y=y} \): conditional probability density function for \( X \) given \( Y = y \).
Suppose we are given the following.

- \( p_Y \): probability mass function for \( Y \).
- \( p_{X|Y=y} \): conditional probability density function for \( X \) given \( Y = y \).

What is the conditional distribution of \( Y \) given \( X \)?
Statistical model for conditional distribution

Suppose we are given the following.

- $p_Y$: probability mass function for $Y$.
- $p_{X|Y=y}$: conditional probability density function for $X$ given $Y = y$.

What is the conditional distribution of $Y$ given $X$?

By Bayes’ rule: for any $x \in \mathbb{R}^d$,

$$
\mathbb{P}(Y = y \mid X = x) = \frac{p_Y(y) \cdot p_{X|Y=y}(x)}{p_X(x)}
$$

(where $p_X$ is unconditional density for $X$).
Suppose we are given the following.

- $p_Y$: probability mass function for $Y$.
- $p_{X|Y=y}$: conditional probability density function for $X$ given $Y = y$.

What is the conditional distribution of $Y$ given $X$?

By Bayes’ rule: for any $x \in \mathbb{R}^d$,

$$P(Y = y | X = x) = \frac{p_Y(y) \cdot p_{X|Y=y}(x)}{p_X(x)}$$

(where $p_X$ is unconditional density for $X$).

Therefore, log-odds function is

$$\text{log-odds}(x) = \ln \left( \frac{p_{Y}(1)}{p_{Y}(0)} \cdot \frac{p_{X|Y=1}(x)}{p_{X|Y=0}(x)} \right).$$
Log-odds function for our toy model

Log-odds function:

\[
\text{log-odds}(x) = \ln \left( \frac{p_Y(1)}{p_Y(0)} \right) + \ln \left( \frac{p_{X|Y=1}(x)}{p_{X|Y=0}(x)} \right).
\]
Log-odds function for our toy model

Log-odds function:

\[
\text{log-odds}(\mathbf{x}) = \ln \left( \frac{p_Y(1)}{p_Y(0)} \right) + \ln \left( \frac{p_{X|Y=1}(\mathbf{x})}{p_{X|Y=0}(\mathbf{x})} \right).
\]

In our toy model, we have \( Y \sim \text{Bern}(\pi) \) and \( X \mid Y = y \sim N(\mu_y, AA^T) \), so:

\[
\text{log-odds}(\mathbf{x}) = \ln \frac{\pi}{1 - \pi} + \ln \frac{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|^2_2}}{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|^2_2}} \\
= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|^2_2 + \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|^2_2 \\
= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} (\| A^{-1} \mu_1 \|^2_2 - \| A^{-1} \mu_0 \|^2_2) + (\mu_1 - \mu_0)^T (AA^T)^{-1} \mathbf{x}.
\]

constant—doesn’t depend on \( \mathbf{x} \)

linear function of \( \mathbf{x} \).
Log-odds function for our toy model

Log-odds function:

\[
\text{log-odds}(\mathbf{x}) = \ln \left( \frac{p_Y(1)}{p_Y(0)} \right) + \ln \left( \frac{p_X|Y=1(\mathbf{x})}{p_X|Y=0(\mathbf{x})} \right).
\]

In our toy model, we have \( Y \sim \text{Bern}(\pi) \) and \( X \mid Y = y \sim \text{N}(\mu_y, AA^T) \), so:

\[
\text{log-odds}(\mathbf{x}) = \ln \frac{\pi}{1 - \pi} + \ln \frac{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|_2^2}}{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|_2^2}} \\
= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|_2^2 + \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|_2^2 \\
= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} (\| A^{-1} \mu_1 \|_2^2 - \| A^{-1} \mu_0 \|_2^2) + (\mu_1 - \mu_0)^T (AA^T)^{-1} \mathbf{x}.
\]

- This is an affine function of \( \mathbf{x} \).

▶ Hence, the statistical model for \( Y \mid X \) is a logistic regression model (with affine feature expansion).

▶ Important: Logistic regression model forgets about \( p_{X|Y=y} \)!
Log-odds function for our toy model

Log-odds function:

$$\text{log-odds}(\mathbf{x}) = \ln \left( \frac{p_Y(1)}{p_Y(0)} \right) + \ln \left( \frac{p_X|Y=1(\mathbf{x})}{p_X|Y=0(\mathbf{x})} \right).$$

In our toy model, we have $Y \sim \text{Bern}(\pi)$ and $X \mid Y = y \sim \mathcal{N}(\mu_y, \mathbf{A}\mathbf{A}^T)$, so:

$$\text{log-odds}(\mathbf{x}) = \ln \frac{\pi}{1 - \pi} + \ln \frac{e^{-\frac{1}{2} \| \mathbf{A}^{-1}(\mathbf{x} - \mu_1) \|^2}}{e^{-\frac{1}{2} \| \mathbf{A}^{-1}(\mathbf{x} - \mu_0) \|^2}}$$

$$= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} \| \mathbf{A}^{-1}(\mathbf{x} - \mu_1) \|^2 + \frac{1}{2} \| \mathbf{A}^{-1}(\mathbf{x} - \mu_0) \|^2$$

$$= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} (\| \mathbf{A}^{-1}\mu_1 \|^2 - \| \mathbf{A}^{-1}\mu_0 \|^2) + (\mu_1 - \mu_0)^T (\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{x}.$$

- This is an affine function of $\mathbf{x}$.
- Hence, the statistical model for $Y \mid X$ is a logistic regression model (with affine feature expansion).
Log-odds function for our toy model

Log-odds function:

$$\text{log-odds}(\mathbf{x}) = \ln \left( \frac{p_{Y(1)}}{p_{Y(0)}} \right) + \ln \left( \frac{p_{X|Y=1}(\mathbf{x})}{p_{X|Y=0}(\mathbf{x})} \right).$$

In our toy model, we have $Y \sim \text{Bern}(\pi)$ and $X \mid Y = y \sim \mathcal{N}(\mu_y, AA^T)$, so:

$$\text{log-odds}(\mathbf{x}) = \ln \frac{\pi}{1 - \pi} + \ln \frac{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|_2^2}}{e^{-\frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|_2^2}}$$

$$= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_1) \|_2^2 + \frac{1}{2} \| A^{-1}(\mathbf{x} - \mu_0) \|_2^2$$

$$= \ln \frac{\pi}{1 - \pi} - \frac{1}{2} (\| A^{-1} \mu_1 \|_2^2 - \| A^{-1} \mu_0 \|_2^2) + (\mu_1 - \mu_0)^T (AA^T)^{-1} \mathbf{x}.$$

- This is an affine function of $\mathbf{x}$.
- Hence, the statistical model for $Y \mid X$ is a logistic regression model (with affine feature expansion).
- **Important**: Logistic regression model forgets about $p_{X|Y=y}$!
4. Linear classifiers
A linear classifier is specified by a weight vector $\mathbf{w} \in \mathbb{R}^d$:

$$f_{\mathbf{w}}(\mathbf{x}) := \begin{cases} -1 & \text{if } \mathbf{x}^T \mathbf{w} \leq 0, \\ +1 & \text{if } \mathbf{x}^T \mathbf{w} > 0. \end{cases}$$

(Will be notationally simpler to use \{-1, +1\} instead of \{0, 1\}.)
Linear classifiers

A *linear classifier* is specified by a *weight vector* $\mathbf{w} \in \mathbb{R}^d$:

$$f_{\mathbf{w}}(\mathbf{x}) := \begin{cases} -1 & \text{if } \mathbf{x}^T \mathbf{w} \leq 0, \\ +1 & \text{if } \mathbf{x}^T \mathbf{w} > 0. \end{cases}$$

(Will be notationally simpler to use $\{-1, +1\}$ instead of $\{0, 1\}$.)

**Interpretation:** does a linear combination of input features exceed 0?

For $\mathbf{w} = (w_1, \ldots, w_d)$ and $\mathbf{x} = (x_1, \ldots, x_d)$,

$$\mathbf{x}^T \mathbf{w} = \sum_{i=1}^{d} w_i x_i > 0.$$
A hyperplane in $\mathbb{R}^d$ is a linear subspace of dimension $d-1$.

- A $\mathbb{R}^2$-hyperplane is a line.
- A $\mathbb{R}^3$-hyperplane is a plane.
- As a linear subspace, a hyperplane always contains the origin.

A hyperplane $H$ can be specified by a (non-zero) normal vector $w \in \mathbb{R}^d$.

The hyperplane with normal vector $w$ is the set of points orthogonal to $w$:

$$H = \left\{ x \in \mathbb{R}^d : x^T w = 0 \right\}.$$
Classification with a hyperplane

Projection of $x$ onto $\text{span}\{w\}$ (a line) has coordinate $\|x\|_2 \cdot \cos(\theta)$ where $\cos(\theta) = \frac{x^T w}{\|w\|_2 \|x\|_2}$.

(Distance to hyperplane is $\|x\|_2 \cdot |\cos(\theta)|$.)

Decision boundary is hyperplane (oriented by $w$):

$x^T w > 0 \iff \|x\|_2 \cdot \cos(\theta) > 0 \iff x$ on same side of $H$ as $w$.

What should we do if we want hyperplane decision boundary that doesn’t (necessarily) go through origin?
Classification with a hyperplane

Projection of $\mathbf{x}$ onto $\text{span}\{\mathbf{w}\}$ (a line) has coordinate

$$\|\mathbf{x}\|_2 \cdot \cos(\theta)$$

where

$$\cos(\theta) = \frac{\mathbf{x}^\top \mathbf{w}}{\|\mathbf{w}\|_2 \|\mathbf{x}\|_2}.$$ 

(Distance to hyperplane is $\|\mathbf{x}\|_2 \cdot |\cos(\theta)|$.)
Projection of \( x \) onto \( \text{span}\{w\} \) (a line) has coordinate

\[
\|x\|_2 \cdot \cos(\theta)
\]

where

\[
\cos(\theta) = \frac{x^\top w}{\|w\|_2 \|x\|_2}.
\]

(Distance to hyperplane is \( \|x\|_2 \cdot |\cos(\theta)| \).)

*Decision boundary* is hyperplane (oriented by \( w \)):

\[
x^\top w > 0 \iff \|x\|_2 \cdot \cos(\theta) > 0 \iff x \text{ on same side of } H \text{ as } w
\]
Projection of $\mathbf{x}$ onto $\text{span}\{\mathbf{w}\}$ (a line) has coordinate

$$\|\mathbf{x}\|_2 \cdot \cos(\theta)$$

where

$$\cos(\theta) = \frac{\mathbf{x}^\top \mathbf{w}}{\|\mathbf{w}\|_2 \|\mathbf{x}\|_2}.$$  

(Distance to hyperplane is $\|\mathbf{x}\|_2 \cdot |\cos(\theta)|$.)

**Decision boundary** is hyperplane (oriented by $\mathbf{w}$):

$$\mathbf{x}^\top \mathbf{w} > 0 \iff \|\mathbf{x}\|_2 \cdot \cos(\theta) > 0 \iff \mathbf{x} \text{ on same side of } H \text{ as } \mathbf{w}$$

What should we do if we want hyperplane decision boundary that doesn’t (necessarily) go through origin?
Same feature expansions we saw for linear regression models can also be used here to "upgrade" linear classifiers.

- **Elliptical decision boundary**
- **Hyperbolic decision boundary**
Same *feature expansions* we saw for linear regression models can also be used here to “upgrade” linear classifiers.
How can we get features for text?
How can we get features for text?

Classifying words (input = a sequence of characters)

- $x_{\text{starts with anti}} = 1 \{ \text{starts with “anti”} \}$
Text features

How can we get features for text?

Classifying words (input = a sequence of characters)

- $x_{\text{starts with anti}} = 1 \{\text{starts with "anti"} \}$
- $x_{\text{ends with logy}} = 1 \{\text{ends with "logy"} \}$

Classifying documents (input = a sequence of words)

- $x_{\text{contains aardvark}} = 1 \{\text{contains "aardvark"} \}$
- $x_{\text{contains each day}} = 1 \{\text{contains "each day"} \}$

Etc. Typically end up with lots of features!
Text features

How can we get features for text?

Classifying words (input = a sequence of characters)

- $x_{\text{starts\_with\_anti}} = 1\{\text{starts with “anti”}\}$
- $x_{\text{ends\_with\_logy}} = 1\{\text{ends with “logy”}\}$
- ... (same for all four-letter prefixes/suffixes)

Classifying documents (input = a sequence of words)

- $x_{\text{contains\_aardvark}} = 1\{\text{contains “aardvark”}\}$
- ... (same for all words in dictionary)
- $x_{\text{contains\_each\_day}} = 1\{\text{contains “each day”}\}$
- ... (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!

How can we get features for text?

**Classifying words** (input = a sequence of characters)

- \( x_{\text{starts with \text{anti}}} = 1 \{ \text{starts with \text{“anti”}} \} \)
- \( x_{\text{ends with \text{logy}}} = 1 \{ \text{ends with \text{“logy”}} \} \)
- \( x_{\text{length} \leq 3} = 1 \{ \text{length \leq 3} \} \)
- ... (same for all four-letter prefixes/suffixes)

Classifying documents (input = a sequence of words)

- \( x_{\text{contains \text{aardvark}}} = 1 \{ \text{contains \text{“aardvark”}} \} \)
- \( x_{\text{contains each day}} = 1 \{ \text{contains \text{“each day”}} \} \)
- ... (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!
Text features

How can we get features for text?

Classifying words (input = a sequence of characters)

- \( x_{\text{starts\_with\_anti}} = 1 \{ \text{starts with "anti"} \} \)
- \( x_{\text{ends\_with\_logy}} = 1 \{ \text{ends with "logy"} \} \)
- \( \ldots \) (same for all four-letter prefixes/suffixes)
- \( x_{\text{length}\leq3} = 1 \{ \text{length} \leq 3 \} \)
- \( x_{\text{length}\leq4} = 1 \{ \text{length} \leq 4 \} \)

Classifying documents (input = a sequence of words)

- \( x_{\text{contains\_aardvark}} = 1 \{ \text{contains "aardvark"} \} \)
- \( \ldots \) (same for all words in dictionary)
- \( x_{\text{contains\_each\_day}} = 1 \{ \text{contains "each day"} \} \)
- \( \ldots \) (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!
How can we get features for text?

**Classifying words** (input = a sequence of characters)

- $x_{\text{starts_with_anti}} = \mathbb{1}\{\text{starts with "anti"} \}$
- $x_{\text{ends_with_logy}} = \mathbb{1}\{\text{ends with "logy"} \}$
- . . . (same for all four-letter prefixes/suffixes)
- $x_{\text{length} \leq 3} = \mathbb{1}\{\text{length} \leq 3 \}$
- $x_{\text{length} \leq 4} = \mathbb{1}\{\text{length} \leq 4 \}$
- . . . (same for all positive integers up to 20)

**Classifying documents** (input = a sequence of words)

- $x_{\text{contains_aardvark}} = \mathbb{1}\{\text{contains "aardvark"} \}$
- . . . (same for all words in dictionary)
- $x_{\text{contains_each_day}} = \mathbb{1}\{\text{contains "each day"} \}$
- . . . (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!
How can we get features for text?

**Classifying words** (input = a sequence of characters)

- $x_{\text{starts\_with\_anti}} = 1 \{\text{starts with “anti”}\}$
- $x_{\text{ends\_with\_logy}} = 1 \{\text{ends with “logy”}\}$
- ... (same for all four-letter prefixes/suffixes)
- $x_{\text{length} \leq 3} = 1 \{\text{length} \leq 3\}$
- $x_{\text{length} \leq 4} = 1 \{\text{length} \leq 4\}$
- ... (same for all positive integers up to 20)

**Classifying documents** (input = a sequence of words)

- $x_{\text{contains\_aardvark}} = 1 \{\text{contains “aardvark”}\}$
- ... (same for all words in dictionary)
- $x_{\text{contains\_each\_day}} = 1 \{\text{contains “each day”}\}$
- ... (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!
Text features

How can we get features for text?

Classifying words (input = a sequence of characters)

- \( x_{\text{starts\_with\_anti}} = 1 \{ \text{starts with “anti”} \} \)
- \( x_{\text{ends\_with\_logy}} = 1 \{ \text{ends with “logy”} \} \)
- \( \ldots \) (same for all four-letter prefixes/suffixes)
- \( x_{\text{length} \leq 3} = 1 \{ \text{length } \leq 3 \} \)
- \( x_{\text{length} \leq 4} = 1 \{ \text{length } \leq 4 \} \)
- \( \ldots \) (same for all positive integers up to 20)

Classifying documents (input = a sequence of words)

- \( x_{\text{contains\_aardvark}} = 1 \{ \text{contains “aardvark”} \} \)
How can we get features for text?

Classifying words (input = a sequence of characters)
- \( x_{\text{starts\_with\_anti}} = \mathbb{1}\{\text{starts with “anti”}\} \)
- \( x_{\text{ends\_with\_logy}} = \mathbb{1}\{\text{ends with “logy”}\} \)
- \( \ldots \) (same for all four-letter prefixes/suffixes)
- \( x_{\text{length}\leq3} = \mathbb{1}\{\text{length } \leq 3\} \)
- \( x_{\text{length}\leq4} = \mathbb{1}\{\text{length } \leq 4\} \)
- \( \ldots \) (same for all positive integers up to 20)

Classifying documents (input = a sequence of words)
- \( x_{\text{contains\_aardvark}} = \mathbb{1}\{\text{contains “aardvark”}\} \)
- \( \ldots \) (same for all words in dictionary)
How can we get features for text?

**Classifying words** (input = a sequence of characters)

- $x_{\text{starts\_with\_anti}} = 1\{\text{starts with “anti”}\}$
- $x_{\text{ends\_with\_logy}} = 1\{\text{ends with “logy”}\}$
- ... (same for all four-letter prefixes/suffixes)
- $x_{\text{length}\leq3} = 1\{\text{length } \leq 3\}$
- $x_{\text{length}\leq4} = 1\{\text{length } \leq 4\}$
- ... (same for all positive integers up to 20)

**Classifying documents** (input = a sequence of words)

- $x_{\text{contains\_aardvark}} = 1\{\text{contains “aardvark”}\}$
- ... (same for all words in dictionary)
- $x_{\text{contains\_each\_day}} = 1\{\text{contains “each day”}\}$
Text features

How can we get features for text?

### Classifying words (input = a sequence of characters)

- $x_{\text{starts\_with\_anti}} = 1\{\text{starts with “anti”}\}$
- $x_{\text{ends\_with\_logy}} = 1\{\text{ends with “logy”}\}$
- . . . (same for all four-letter prefixes/suffixes)
- $x_{\text{length}\leq3} = 1\{\text{length } \leq 3\}$
- $x_{\text{length}\leq4} = 1\{\text{length } \leq 4\}$
- . . . (same for all positive integers up to 20)

### Classifying documents (input = a sequence of words)

- $x_{\text{contains\_aardvark}} = 1\{\text{contains “aardvark”}\}$
- . . . (same for all words in dictionary)
- $x_{\text{contains\_each\_day}} = 1\{\text{contains “each day”}\}$
- . . . (same for all bi-grams of words in dictionary)

Etc. Typically end up with lots of features!
Sparse representations

If $x$ has few non-zero entries, then better to use *sparse representation*:
Sparse representations

If $x$ has few non-zero entries, then better to use *sparse representation*:

E.g., “see spot run”.

- Sparse representation (e.g., using hash table):

  ```python
  x = { "contains_see":1, "contains_spot":1,
       "contains_run":1, "contains_see_spot":1,
       "contains_spot_run":1 }
  ```
Sparse representations

If $x$ has few non-zero entries, then better to use \textit{sparse representation}:

E.g., “see spot run”.

- Sparse representation (e.g., using hash table):

  $$x = \{ "contains\_see":1, "contains\_spot":1, "contains\_run":1, "contains\_see\_spot":1, "contains\_spot\_run":1 \}$$

- Compare to dense representation (using array):

  $$x = [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, \ldots ]$$

To compute inner product $w^T x$:

$$\text{sum} = 0.$$

for feature in $x$:

$$\text{sum} = \text{sum} + w[\text{feature}] \times x[\text{feature}]$$

Running time $\propto$ number of non-zero entries of $x$.

If $x$’s are sparse, would you expect there to be a good sparse linear classifier?
Sparse representations

If $x$ has few non-zero entries, then better to use *sparse representation*:

E.g., “see spot run”.

- Sparse representation (e.g., using hash table):
  
  $x = \{ \text{"contains\_see":1, \text{"contains\_spot":1,}\
  \text{"contains\_run":1, \text{"contains\_see\_spot":1,}\
  \text{"contains\_spot\_run":1 } } \}$

- Compare to dense representation (using array):
  
  $x = [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, \ldots ]$

To compute inner product $w^T x$:

```
    sum = 0.
    for feature in x:
        sum = sum + w[feature] * x[feature]
```

Running time $\propto$ number of non-zero entries of $x$. 

Sparse representations

If $x$ has few non-zero entries, then better to use *sparse representation*:

E.g., “see spot run”.

▶ Sparse representation (e.g., using hash table):

$$x = \{ "contains\_see" : 1, "contains\_spot" : 1, "contains\_run" : 1, "contains\_see\_spot" : 1, "contains\_spot\_run" : 1 \}$$

▶ Compare to dense representation (using array):

$$x = [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, \ldots ]$$

To compute inner product $w^T x$:

$$\text{sum} = 0.$$  
$$\text{for feature in x:}$$  
$$\quad \text{sum} = \text{sum} + w[\text{feature}] \times x[\text{feature}]$$

Running time $\propto$ number of non-zero entries of $x$.

If $x$’s are sparse, would you expect there to be a good sparse linear classifier?
5. Learning linear classifiers
Learning linear classifiers

Even if Bayes optimal classifier is not linear (in our chosen feature space), we can hope that it has a good linear approximation.
Learning linear classifiers

Even if Bayes optimal classifier is not linear (in our chosen feature space), we can hope that it has a good linear approximation.

**Goal**: learn $\hat{w} \in \mathbb{R}^d$ using iid sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ such that

\[
\text{Excess risk}(f_{\hat{w}}) := \mathcal{R}(f_{\hat{w}}) - \min_{w \in \mathbb{R}^d} \mathcal{R}(f_w)
\]

is as small as possible, where the **loss function is zero-one loss**.
Learning linear classifiers

Even if Bayes optimal classifier is not linear (in our chosen feature space), we can hope that it has a good linear approximation.

**Goal**: learn $\hat{w} \in \mathbb{R}^d$ using iid sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ such that

$$\text{Excess risk}(f_{\hat{w}}) := \mathbb{R}(f_{\hat{w}}) - \min_{w \in \mathbb{R}^d} \mathbb{R}(f_w)$$

is as small as possible, where the **loss function is zero-one loss**.

**Empirical risk minimization (ERM)**: find $f_w$ with minimum **empirical risk**

$$\hat{\mathbb{R}}(f_w) := \frac{1}{n} \sum_{i=1}^{n} 1\{f_w(X_i) \neq Y_i\}.$$
Learning linear classifiers

Even if Bayes optimal classifier is not linear (in our chosen feature space), we can hope that it has a good linear approximation.

**Goal**: learn $\hat{w} \in \mathbb{R}^d$ using iid sample $(X_1, Y_1), \ldots, (X_n, Y_n)$ such that

$$\text{Excess risk}(f_{\hat{w}}) := \mathcal{R}(f_{\hat{w}}) - \min_{w \in \mathbb{R}^d} \mathcal{R}(f_w)$$

is as small as possible, where the **loss function is zero-one loss**.

*Empirical risk minimization (ERM)*: find $f_w$ with minimum *empirical risk*

$$\hat{\mathcal{R}}(f_w) := \frac{1}{n} \sum_{i=1}^{n} 1 \{f_w(X_i) \neq Y_i\}.$$ 

**Theorem**: ERM solution $f_{\hat{w}}$ satisfies

$$\mathbb{E} \mathcal{R}(f_{\hat{w}}) \to \min_{w \in \mathbb{R}^d} \mathcal{R}(f_w)$$

as $n \to \infty$ at rate $\sqrt{\frac{d}{n}}$ (and sometimes even faster!).
Unfortunately, cannot compute (zero-one loss) ERM in general.
Unfortunately, cannot compute (zero-one loss) ERM in general.

- The following problem is NP-hard (i.e., “computationally intractable”):
  
  **input**  \( n \) labeled examples from \( \mathbb{R}^d \times \{0, 1\} \) with promise that there is a linear classifier with empirical risk 0.01.
  
  **output** a linear classifier with empirical risk \( \leq 0.49 \).

(Zero-one loss is very different from squared loss!)
Unfortunately, cannot compute (zero-one loss) ERM in general.

- The following problem is NP-hard (i.e., “computationally intractable”):
  
  **input** $n$ labeled examples from $\mathbb{R}^d \times \{0, 1\}$ with promise that there is a linear classifier with empirical risk 0.01.
  
  **output** a linear classifier with empirical risk $\leq 0.49$.

  (Zero-one loss is very different from squared loss!)

**Potential saving grace:**

- Real-world problems we need to solve do not look like the encodings of difficult 3-SAT instances.
6. Linearly separable data and Perceptron
Suppose there is a linear classifier that perfectly classifies the training examples
\[ S := ((x_1, y_1), \ldots, (x_n, y_n)), \]
i.e., for some \( w_* \in \mathbb{R}^d \),
\[ f_{w_*}(x) = y, \quad \text{for all} \ (x, y) \in S. \]

In this case, we say the training data is \emph{linearly separable}. 
Problem: given training examples $S$ from $\mathbb{R}^d \times \{-1, 1\}$, determine whether or not there exists $w \in \mathbb{R}^d$ such that

$$f_w(x) = y, \text{ for all } (x, y) \in S;$$

(and find such a vector if one exists).
Finding a linear separator

**Problem**: given training examples $S$ from $\mathbb{R}^d \times \{-1, 1\}$, determine whether or not there exists $w \in \mathbb{R}^d$ such that

$$f_w(x) = y, \quad \text{for all } (x, y) \in S;$$

(and find such a vector if one exists).

- $d$ variables: $w \in \mathbb{R}^d$
- $|S|$ inequalities: for $(x, y) \in S$,

  - if $y = -1$: $x^T w \leq 0$,
  - if $y = +1$: $x^T w > 0$.

Can be solved in polynomial time using algorithms for **linear programming**.
Finding a linear separator

**Problem:** given training examples $S$ from $\mathbb{R}^d \times \{-1, 1\}$, determine whether or not there exists $w \in \mathbb{R}^d$ such that

$$f_w(x) = y, \quad \text{for all } (x, y) \in S;$$

(and find such a vector if one exists).

- $d$ variables: $w \in \mathbb{R}^d$
- $|S|$ inequalities: for $(x, y) \in S$,

  - if $y = -1$: $x^Tw \leq 0$,
  - if $y = +1$: $x^Tw > 0$.

Can be solved in polynomial time using algorithms for **linear programming**.

If one exists, and the inequalities can be satisfied with some non-negligible “wiggle room”, then there is a very **simple** algorithm that finds a solution.
Perceptron (Rosenblatt, 1958)

**Perceptron**

**input** Labeled examples $S$ from $\mathbb{R}^d \times \{-1, +1\}$.

1. initialize $\hat{w}_1 := 0$.
2. for $t = 1, 2, \ldots, \text{do}$
3. if there is an example in $S$ misclassified by $f_{\hat{w}_t}$ then
4. Let $(x_t, y_t)$ be any such misclassified example from $S$.
5. Update: $\hat{w}_{t+1} := \hat{w}_t + y_t x_t$.
6. else
7. return $\hat{w}_t$.
8. end if
9. end for

Note 1: An example $(x, y)$ is misclassified by $f_w$ if $y x^T \hat{w} \leq 0$.

Note 2: If Perceptron terminates, then $f_{\hat{w}_t}$ perfectly classifies the data!
Perceptron (Rosenblatt, 1958)

**Perceptron**

**input** Labeled examples $S$ from $\mathbb{R}^d \times \{-1, +1\}$.

1. **initialize** $\hat{w}_1 := 0$.
2. **for** $t = 1, 2, \ldots, \text{do}$
3.  **if** there is an example in $S$ misclassified by $f_{\hat{w}_t}$ **then**
4.     Let $(x_t, y_t)$ be any such misclassified example from $S$.
5.     **Update**: $\hat{w}_{t+1} := \hat{w}_t + y_t x_t$.
6.  **else**
7.     **return** $\hat{w}_t$.
8. **end if**
9. **end for**

**Note 1**: An example $(x, y)$ is misclassified by $f_w$ if $yx^T w \leq 0$. 
Perceptron (Rosenblatt, 1958)

**Perceptron**

**input** Labeled examples $S$ from $\mathbb{R}^d \times \{-1, +1\}$.

1. **initialize** $\hat{w}_1 := 0$.
2. **for** $t = 1, 2, \ldots, \text{do}$
3. **if** there is an example in $S$ misclassified by $f_{\hat{w}_t}$ **then**
4. Let $(x_t, y_t)$ be any such misclassified example from $S$.
5. **Update**: $\hat{w}_{t+1} := \hat{w}_t + y_t x_t$.
6. **else**
7. **return** $\hat{w}_t$.
8. **end if**
9. **end for**

**Note 1:** An example $(x, y)$ is misclassified by $f_w$ if $yx^T w \leq 0$.

**Note 2:** If Perceptron terminates, then $f_{\hat{w}_t}$ perfectly classifies the data!
Scenario 1

Current vector $\hat{\mathbf{w}}_t$ comparable to $\mathbf{x}_t$ in length.
Scenario 1

Updated vector $\hat{w}_{t+1}$ now correctly classifies $(x_t, y_t)$. 
Scenario 2

\[ \hat{w}_t \]

\[ y_t = 1 \]

\[ x_t^T \hat{w}_t \leq 0 \]

Current vector \( \hat{w}_t \) much longer than \( x_t \).
Perceptron demo

Scenario 2

\[ x_t \begin{array}{c} \hat{w}_t \leq 0 \\ \hat{w}_{t+1} \end{array} \]

\[ y_t = 1 \]

Updated vector \( \hat{w}_{t+1} \) does not correctly classify \((x_t, y_t)\).

Not obvious that Perceptron will eventually terminate!
Scenario 2

Updated vector $\hat{\mathbf{w}}_{t+1}$ does not correctly classify $(\mathbf{x}_t, y_t)$.

Not obvious that Perceptron will eventually terminate!
When is there a lot of “wiggle room”?

Suppose $w \in \mathbb{R}^d$ satisfies

$$\min_{(x,y) \in S} yx^T w > 0.$$
When is there a lot of “wiggle room”?

Suppose $w \in \mathbb{R}^d$ satisfies

$$\min_{(x,y) \in S} yx^T w > 0.$$ 

Then so does, e.g., $w/100$. 

"Theorem": Perceptron converges quickly when there is a short $w$ with

$$\min_{(x,y) \in S} yx^T w = 1.$$ 

This distance is called the margin.
When is there a lot of “wiggle room”?

Suppose $w \in \mathbb{R}^d$ satisfies

$$\min_{(x,y) \in S} yx^T w > 0.$$ 

Then so does, e.g., $w/100$. Let’s fix a particular scaling of $w$. 

"Theorem": Perceptron converges quickly when there is a short $w$ with $\min_{(x,y) \in S} yx^T w = 1$. 

Therefore, if $w$ is a short vector satisfying $\min_{(x,y) \in S} yx^T w = 1$, then it corresponds to a linear separator with large margin on examples in $S$. 

(Rescale $w$ so that $\tilde{y}\tilde{x}^T w = 1$.) Now distance from $\tilde{y}\tilde{x}$ to $H$ is $\|w\|_2$. This distance is called the margin.
When is there a lot of “wiggle room”?

Suppose \( \mathbf{w} \in \mathbb{R}^d \) satisfies

\[
\min_{(x,y) \in S} yx^\top \mathbf{w} > 0.
\]

Then so does, e.g., \( \mathbf{w}/100 \). Let’s fix a particular scaling of \( \mathbf{w} \).

Let \( (\tilde{x}, \tilde{y}) \) be any example in \( S \) that achieves the minimum.
When is there a lot of “wiggle room”?

Suppose \( w \in \mathbb{R}^d \) satisfies
\[
\min_{(x,y) \in S} yx^T w > 0.
\]

Then so does, e.g., \( w/100 \). Let’s fix a particular scaling of \( w \).
Let \((\tilde{x}, \tilde{y})\) be any example in \( S \) that achieves the minimum.

Let’s rescale \( w \) so that \( \tilde{y}\tilde{x}^T w = 1 \). (Now scaling is fixed.)

Now distance from \( \tilde{y}\tilde{x}^T \) to \( H \) is
\[
1 \parallel w \parallel_2.
\]
This distance is called the margin.

Therefore, if \( w \) is a short vector satisfying \( \min_{(x,y) \in S} yx^T w = 1 \),
then it corresponds to a linear separator with large margin on examples in \( S \).
When is there a lot of “wiggle room”?

Suppose \( \mathbf{w} \in \mathbb{R}^d \) satisfies

\[
\min_{(x,y) \in S} yx^T \mathbf{w} > 0.
\]

Then so does, e.g., \( \mathbf{w}/100 \). Let’s fix a particular scaling of \( \mathbf{w} \).

Let \( (\tilde{x}, \tilde{y}) \) be any example in \( S \) that achieves the minimum.

- Rescale \( \mathbf{w} \) so that \( \tilde{y}\tilde{x}^T \mathbf{w} = 1 \).
  (Now scaling is fixed.)
When is there a lot of “wiggle room”?

Suppose \( w \in \mathbb{R}^d \) satisfies

\[
\min_{(x, y) \in S} y x^T w > 0.
\]

Then so does, e.g., \( w/100 \). Let’s fix a particular scaling of \( w \).

Let \( (\tilde{x}, \tilde{y}) \) be any example in \( S \) that achieves the minimum.

- Rescale \( w \) so that \( \tilde{y} \tilde{x}^T w = 1 \).
  (Now scaling is fixed.)
- Now distance from \( \tilde{y} \tilde{x} \) to \( H \) is \( \frac{1}{\|w\|_2} \).
  This distance is called the margin.
When is there a lot of “wiggle room”?

Suppose $w \in \mathbb{R}^d$ satisfies

$$\min_{(x, y) \in S} yx^T w > 0.$$ 

Then so does, e.g., $w/100$. Let’s fix a particular scaling of $w$.

Let $(\tilde{x}, \tilde{y})$ be any example in $S$ that achieves the minimum.

- Rescale $w$ so that $\tilde{y}\tilde{x}^T w = 1$.
  (Now scaling is fixed.)
- Now distance from $\tilde{y}\tilde{x}$ to $H$ is $\frac{1}{\|w\|_2}$.
  This distance is called the margin.

Therefore, if $w$ is a short vector satisfying

$$\min_{(x, y) \in S} yx^T w = 1,$$

then it corresponds to a linear separator with large margin on examples in $S$. 
When is there a lot of “wiggle room”?

Suppose \( w \in \mathbb{R}^d \) satisfies
\[
\min_{(x,y) \in S} yx^T w > 0.
\]
Then so does, e.g., \( w/100 \). Let’s fix a particular scaling of \( w \).
Let \((\tilde{x}, \tilde{y})\) be any example in \( S \) that achieves the minimum.

- Rescale \( w \) so that \( \tilde{y}\tilde{x}^T w = 1 \).
  (Now scaling is fixed.)
- Now distance from \( \tilde{y}\tilde{x} \) to \( H \) is \( \frac{1}{\|w\|_2} \).
  This distance is called the margin.

Therefore, if \( w \) is a short vector satisfying
\[
\min_{(x,y) \in S} yx^T w = 1,
\]
then it corresponds to a linear separator with large margin on examples in \( S \).

“Theorem”: Perceptron converges quickly when there short \( w \) with \( \min_{(x,y) \in S} yx^T w = 1 \).
Theorem. Assume there exists \( \mathbf{w}_\star \in \mathbb{R}^d \) such that

\[
\min_{(x,y) \in S} yx^T \mathbf{w}_\star = 1.
\]

Let \( L := \max_{(x,y) \in S} \|x\|_2 \). Then Perceptron halts after \( \|\mathbf{w}_\star\|^2_2 L^2 \) iterations.
Theorem. Assume there exists $w_\star \in \mathbb{R}^d$ such that
\[ \min_{(x,y) \in S} y x^T w_\star = 1. \]

Let $L := \max_{(x,y) \in S} \|x\|_2$. Then Perceptron halts after $\|w_\star\|_2^2 L^2$ iterations.

Proof. Suppose Perceptron does not exit the for-loop in iteration $t$. 

Theorem. Assume there exists \( \mathbf{w}_* \in \mathbb{R}^d \) such that
\[
\min_{(x,y) \in S} yx^T \mathbf{w}_* = 1.
\]
Let \( L := \max_{(x,y) \in S} \|x\|_2 \). Then Perceptron halts after \( \|\mathbf{w}_*\|^2_2 L^2 \) iterations.

Proof: Suppose Perceptron does not exit the for-loop in iteration \( t \).

Then there is a labeled example in \( S' \) — which we call \( (x_t, y_t) \) — such that:
\[
\begin{align*}
\text{\( y_t \mathbf{w}_*^T x_t \geq 1 \) (by definition of \( \mathbf{w}_* \))} \\
\text{\( y_t \hat{\mathbf{w}}_t^T x_t \leq 0 \) (since \( f_{\hat{\mathbf{w}}_t} \) misclassifies the example)} \\
\text{\( \hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t + y_t x_t \) (since an update is made)}
\end{align*}
\]
Theorem. Assume there exists $\mathbf{w}_* \in \mathbb{R}^d$ such that

$$\min_{(x,y) \in S} yx^T \mathbf{w}_* = 1.$$ 

Let $L := \max_{(x,y) \in S} \|x\|_2$. Then Perceptron halts after $\|\mathbf{w}_*\|_2^2 L^2$ iterations.

**Proof.** Suppose Perceptron does not exit the for-loop in iteration $t$. Then there is a labeled example in $S$ — which we call $(\mathbf{x}_t, y_t)$ — such that:

- $y_t \mathbf{w}_*^T \mathbf{x}_t \geq 1$ (by definition of $\mathbf{w}_*$)
- $y_t \hat{\mathbf{w}}_t^T \mathbf{x}_t \leq 0$ (since $f_{\hat{\mathbf{w}}_t}$ misclassifies the example)
- $\hat{\mathbf{w}}_{t+1} = \hat{\mathbf{w}}_t + y_t \mathbf{x}_t$ (since an update is made)

Use this information to (inductively) lower-bound

$$\cos \left( \text{angle between } \mathbf{w}_* \text{ and } \hat{\mathbf{w}}_{t+1} \right) = \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2}.$$
Proof of Perceptron convergence theorem

**Goal:** lower-bound \( \frac{w^*_T \hat{w}_{t+1}}{\|w_*\|_2 \|\hat{w}_{t+1}\|_2} \).

\[
\text{Numerator:} \quad w^*_T \hat{w}_{t+1} = w^*_T (\hat{w}_{t+1} + y_t x_t) = w^*_T \hat{w}_{t+1} + y_t w^*_T x_t \geq w^*_T \hat{w}_{t+1} + 1.
\]

Therefore, by induction (with \( \hat{w}_{1} = 0 \)), \( w^*_T \hat{w}_{t+1} \geq t \).

\[
\text{Denominator:} \quad \|\hat{w}_{t+1}\|_2^2 = \|\hat{w}_{t+1} + y_t x_t\|_2^2 = \|\hat{w}_{t+1}\|_2^2 + 2y_t \hat{w}_T x_t + \|x_t\|_2^2 \leq \|\hat{w}_{t+1}\|_2^2 + L^2.
\]

Therefore, by induction (with \( \hat{w}_{1} = 0 \)), \( \|\hat{w}_{t+1}\|_2^2 \leq L^2 \cdot t \).

\[
\cos (\text{angle between } w_* \text{ and } \hat{w}_{t+1}) = \frac{w^*_T \hat{w}_{t+1}}{\|w_*\|_2 \|\hat{w}_{t+1}\|_2} \geq \frac{t}{\|w_*\|_2 L \sqrt{t}}.
\]

Since cosine is at most one, we conclude \( t \leq \frac{\|w_*\|_2^2}{L^2} \).
Proof of Perceptron convergence theorem

**Goal**: lower-bound \[
\frac{\mathbf{w}_* \hat{\mathbf{w}}_{t+1}}{\| \mathbf{w}_* \|_2 \| \hat{\mathbf{w}}_{t+1} \|_2}.
\]

- **Numerator**:
  \[
  \mathbf{w}_* \hat{\mathbf{w}}_{t+1} = \mathbf{w}_* (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t)
  \]
Proof of Perceptron convergence theorem

**Goal:** lower-bound \( \frac{\mathbf{w}_*\hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \).

- **Numerator:**

  \[ \mathbf{w}_*\hat{\mathbf{w}}_{t+1} = \mathbf{w}_*(\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*\hat{\mathbf{w}}_t + y_t \mathbf{w}_*\mathbf{x}_t \]
Proof of Perceptron convergence theorem

**Goal:** lower-bound \( \frac{w^*_\mathbf{w}_{t+1}}{\|w_*\|_2 \|\mathbf{w}_{t+1}\|_2} \).

- **Numerator:**

  \[
  w^*_\mathbf{w}_{t+1} = w^*_t (\mathbf{w}_t + y_t \mathbf{x}_t) = w^*_t \mathbf{w}_t + y_t w^*_x \mathbf{x}_t \geq w^*_t \mathbf{w}_t + 1.
  \]
Proof of Perceptron convergence theorem

**Goal**: lower-bound \( \frac{\| w^T \hat{w}_{t+1} \|}{\| w_* \|_2 \| \hat{w}_{t+1} \|_2} \).

- **Numerator**:
  \[
  w_*^T \hat{w}_{t+1} = w_*^T (\hat{w}_t + y_t x_t) = w_*^T \hat{w}_t + y_t w_*^T x_t \geq w_*^T \hat{w}_t + 1. 
  \]

  Therefore, by induction (with \( \hat{w}_1 = 0 \)), \( w_*^T \hat{w}_{t+1} \geq t \).
Proof of Perceptron convergence theorem

**Goal**: lower-bound \( \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \).

- **Numerator**:
  \[
  \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} = \mathbf{w}_*^T (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*^T \hat{\mathbf{w}}_t + y_t \mathbf{w}_*^T \mathbf{x}_t \geq \mathbf{w}_*^T \hat{\mathbf{w}}_t + 1.
  \]
  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = 0 \)), \( \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} \geq t \).

- **Denominator**:
  \[
  \|\hat{\mathbf{w}}_{t+1}\|_2^2 = \|\hat{\mathbf{w}}_t + y_t \mathbf{x}_t\|_2^2
  \]
  Since cosine is at most one, we conclude \( t \leq \|\mathbf{w}_*\|_2^2 \cdot \mathbf{L}^2 \).
Proof of Perceptron convergence theorem

**Goal:** lower-bound \( \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \).

- **Numerator:**
  \[
  \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} = \mathbf{w}_*^T (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*^T \hat{\mathbf{w}}_t + y_t \mathbf{w}_*^T \mathbf{x}_t \geq \mathbf{w}_*^T \hat{\mathbf{w}}_t + 1.
  \]
  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = 0 \)), \( \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} \geq t \).

- **Denominator:**
  \[
  \|\hat{\mathbf{w}}_{t+1}\|_2^2 = \|\hat{\mathbf{w}}_t + y_t \mathbf{x}_t\|_2^2 = \|\hat{\mathbf{w}}_t\|_2^2 + 2y_t \hat{\mathbf{w}}_t^T \mathbf{x}_t + \|\mathbf{x}_t\|_2^2
  \]
  Since cosine is at most one, we conclude \( t \leq \|\mathbf{w}_*\|_2^2 \cdot \frac{L^2}{\sqrt{t}} \).
Proof of Perceptron convergence theorem

**Goal:** lower-bound $\frac{w_\star^T \hat{w}_{t+1}}{\|w_\star\|_2 \|\hat{w}_{t+1}\|_2}$.

- **Numerator:**
  
  $w_\star^T \hat{w}_{t+1} = w_\star^T (\hat{w}_t + y_t x_t) = w_\star^T \hat{w}_t + y_t w_\star^T x_t \geq w_\star^T \hat{w}_t + 1$.

  Therefore, by induction (with $\hat{w}_1 = 0$), $w_\star^T \hat{w}_{t+1} \geq t$.

- **Denominator:**
  
  $\|\hat{w}_{t+1}\|_2^2 = \|\hat{w}_t + y_t x_t\|_2^2 = \|\hat{w}_t\|_2^2 + 2 y_t \hat{w}_t^T x_t + \|x_t\|_2^2 \leq \|\hat{w}_t\|_2^2 + L^2$.
Proof of Perceptron convergence theorem

**Goal**: lower-bound \( \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \).

- **Numerator**: 
  \[
  \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} = \mathbf{w}_*^T (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*^T \hat{\mathbf{w}}_t + y_t \mathbf{w}_*^T \mathbf{x}_t \geq \mathbf{w}_*^T \hat{\mathbf{w}}_t + 1.
  \]
  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = 0 \)), \( \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} \geq t \).

- **Denominator**: 
  \[
  \|\hat{\mathbf{w}}_{t+1}\|_2^2 = \|\hat{\mathbf{w}}_t + y_t \mathbf{x}_t\|_2^2 = \|\hat{\mathbf{w}}_t\|_2^2 + 2y_t \hat{\mathbf{w}}_t^T \mathbf{x}_t + \|\mathbf{x}_t\|_2^2 \leq \|\hat{\mathbf{w}}_t\|_2^2 + L^2.
  \]
  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = 0 \)), \( \|\hat{\mathbf{w}}_{t+1}\|_2^2 \leq L^2 \cdot t \).
Proof of Perceptron convergence theorem

**Goal:** lower-bound \( \frac{w^*_T \hat{w}_{t+1}}{\|w_*\|_2 \|\hat{w}_{t+1}\|_2} \).

- **Numerator:**
  \[
  w^*_T\hat{w}_{t+1} = w^*_T(\hat{w}_t + y_t x_t) = w^*_T \hat{w}_t + y_t w^*_x x_t \geq w^*_T \hat{w}_t + 1.
  \]

  Therefore, by induction (with \( \hat{w}_1 = 0 \)), \( w^*_T \hat{w}_{t+1} \geq t \).

- **Denominator:**
  \[
  \|\hat{w}_{t+1}\|_2^2 = \|\hat{w}_t + y_t x_t\|_2^2 = \|\hat{w}_t\|_2^2 + 2y_t \hat{w}_T x_t + \|x_t\|_2^2 \leq \|\hat{w}_t\|_2^2 + L^2.
  \]

  Therefore, by induction (with \( \hat{w}_1 = 0 \)), \( \|\hat{w}_{t+1}\|_2^2 \leq L^2 \cdot t \).

\[
\cos(\text{angle between } w_* \text{ and } \hat{w}_{t+1}) = \frac{w^*_T \hat{w}_{t+1}}{\|w_*\|_2 \|\hat{w}_{t+1}\|_2}
\]
Proof of Perceptron convergence theorem

**Goal**: lower-bound \( \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \).

- **Numerator**:
  \[
  \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} = \mathbf{w}_*^T (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*^T \hat{\mathbf{w}}_t + y_t \mathbf{w}_*^T \mathbf{x}_t \geq \mathbf{w}_*^T \hat{\mathbf{w}}_t + 1.
  \]

  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = \mathbf{0} \)), \( \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} \geq t \).

- **Denominator**:
  \[
  \|\hat{\mathbf{w}}_{t+1}\|_2^2 = \|\hat{\mathbf{w}}_t + y_t \mathbf{x}_t\|_2^2 = \|\hat{\mathbf{w}}_t\|_2^2 + 2y_t \hat{\mathbf{w}}_t^T \mathbf{x}_t + \|\mathbf{x}_t\|_2^2 \leq \|\hat{\mathbf{w}}_t\|_2^2 + L^2.
  \]

  Therefore, by induction (with \( \hat{\mathbf{w}}_1 = \mathbf{0} \)), \( \|\hat{\mathbf{w}}_{t+1}\|_2^2 \leq L^2 \cdot t \).

\[
\cos \left( \text{angle between } \mathbf{w}_* \text{ and } \hat{\mathbf{w}}_{t+1} \right) = \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \geq \frac{t}{\|\mathbf{w}_*\|_2 L \sqrt{t}}.
\]
Proof of Perceptron convergence theorem

**Goal:** lower-bound $\frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2}$.

- **Numerator:**
  \[ \mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} = \mathbf{w}_*^T (\hat{\mathbf{w}}_t + y_t \mathbf{x}_t) = \mathbf{w}_*^T \hat{\mathbf{w}}_t + y_t \mathbf{w}_*^T \mathbf{x}_t \geq \mathbf{w}_*^T \hat{\mathbf{w}}_t + 1. \]
  Therefore, by induction (with $\hat{\mathbf{w}}_1 = \mathbf{0}$), $\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1} \geq t$.

- **Denominator:**
  \[ \|\hat{\mathbf{w}}_{t+1}\|_2^2 = \|\hat{\mathbf{w}}_t + y_t \mathbf{x}_t\|_2^2 = \|\hat{\mathbf{w}}_t\|_2^2 + 2y_t \hat{\mathbf{w}}_t^T \mathbf{x}_t + \|\mathbf{x}_t\|_2^2 \leq \|\hat{\mathbf{w}}_t\|_2^2 + L^2. \]
  Therefore, by induction (with $\hat{\mathbf{w}}_1 = \mathbf{0}$), $\|\hat{\mathbf{w}}_{t+1}\|_2^2 \leq L^2 \cdot t$.

\[
\cos \left( \text{angle between } \mathbf{w}_* \text{ and } \hat{\mathbf{w}}_{t+1} \right) = \frac{\mathbf{w}_*^T \hat{\mathbf{w}}_{t+1}}{\|\mathbf{w}_*\|_2 \|\hat{\mathbf{w}}_{t+1}\|_2} \geq \frac{t}{\|\mathbf{w}_*\|_2 \sqrt{t}}.
\]

Since cosine is at most one, we conclude
\[ t \leq \|\mathbf{w}_*\|_2^2 \cdot L^2. \]
We can obtain the linear separator with the largest margin by finding the *shortest* vector $w$ such that

$$\min_{(x,y) \in S} yx^T w = 1.$$
We can obtain the linear separator with the largest margin by finding the *shortest* vector $\mathbf{w}$ such that

$$
\min_{(x,y) \in S} yx^T \mathbf{w} = 1.
$$

This can be expressed as a mathematical optimization problem that is the basis of *support vector machines*.

(More on this later.)
7. Online Perceptron
Online Perceptron

If data is not linearly separable, Perceptron runs forever!
If data is not linearly separable, Perceptron runs forever!

**Alternative:** consider each example once, then stop.

---

**Online Perceptron**

**input** Labeled examples \((x_i, y_i)_{i=1}^n\) from \(\mathbb{R}^d \times \{-1, +1\}\).

1. **initialize** \(\hat{w}_1 := 0\).
2. **for** \(t = 1, 2, \ldots, n\) **do**
3.   **if** \(y_t x_t^T \hat{w}_t \leq 0\) **then**
4.     \(\hat{w}_{t+1} := \hat{w}_t + y_t x_t\).
5.   **else**
6.     \(\hat{w}_{t+1} := \hat{w}_t\)
7. **end if**
8. **end for**
9. **return** \(\hat{w}_{n+1}\).
Online Perceptron

If data is not linearly separable, Perceptron runs forever!

Alternative: consider each example once, then stop.

**Online Perceptron**

**input** Labeled examples \( ((x_i, y_i))_{i=1}^{n} \) from \( \mathbb{R}^d \times \{-1, +1\} \).

1. *initialize* \( \hat{w}_1 := 0 \).
2. *for* \( t = 1, 2, \ldots, n \) *do*
3.  *if* \( y_t x_t^T \hat{w}_t \leq 0 \) *then*
4.   \( \hat{w}_{t+1} := \hat{w}_t + y_t x_t \).
5.  *else*
6.   \( \hat{w}_{t+1} := \hat{w}_t \)
7. *end if*
8. *end for*
9. *return* \( \hat{w}_{n+1} \).

Final classifier \( f_{\hat{w}_{n+1}} \) is not necessarily a linear separator (even if one exists!).
Online learning

*Online learning algorithms:*

Go through examples $$(x_1, y_1), (x_2, y_2), \ldots$$ one-by-one.
Online learning

*Online learning algorithms:*

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.

*Theorem*: If there is a short \(w^*\) with

\[
\min_t y_t x^T_t w^* \geq 1,
\]

then Online Perceptron makes a small number of mistakes.

*Theorem*: If there is a short \(w^*\) that makes a small number of "hinge loss" mistakes, then Online Perceptron makes a small number of mistakes. (See reading assignment—we’ll discuss “hinge loss” in an upcoming lecture.)
**Online learning**

*Online learning algorithms:*

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.
- Upon seeing \(x_t\), learner makes a prediction: \(\hat{y}_t := \hat{f}_t(x_t)\).
Online learning algorithms:

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.
- Upon seeing \(x_t\), learner makes a prediction: \(\hat{y}_t := \hat{f}_t(x_t)\).
- If \(\hat{y}_t \neq y_t\), then the prediction was a “mistake”.

Theorem: If there is a short \(w^\star\) with \(\min_t y_t x_t^T w^\star \geq 1\), then Online Perceptron makes a small number of mistakes.

Theorem: If there is a short \(w^\star\) that makes a small number of “hinge loss mistakes”, then Online Perceptron makes a small number of mistakes. (See reading assignment—we’ll discuss “hinge loss” in an upcoming lecture.)
Online learning

**Online learning algorithms:**

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.
- Upon seeing \(x_t\), learner makes a prediction: \(\hat{y}_t := \hat{f}_t(x_t)\).
- If \(\hat{y}_t \neq y_t\), then the prediction was a “mistake”.
- See correct label \(y_t\); update \(\hat{f}_t\) to get \(\hat{f}_{t+1}\).

Typically, update is very computationally cheap to compute.
Online learning

**Online learning algorithms:**

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.
- Upon seeing \(x_t\), learner makes a prediction: \(\hat{y}_t := \hat{f}_t(x_t)\).
- If \(\hat{y}_t \neq y_t\), then the prediction was a “mistake”.
- See correct label \(y_t\); update \(\hat{f}_t\) to get \(\hat{f}_{t+1}\).

Typically, update is very computationally cheap to compute.

“**Theorem**”: If there is a short \(w_\star\) with \(\min_t y_t x_t^\top w_\star \geq 1\), then Online Perceptron makes a small number of mistakes.
Online learning

**Online learning algorithms:**

Go through examples \((x_1, y_1), (x_2, y_2), \ldots\) one-by-one.

- Before seeing \((x_t, y_t)\), learner has a “current” classifier \(\hat{f}_t\) in hand.
- Upon seeing \(x_t\), learner makes a prediction: \(\hat{y}_t := \hat{f}_t(x_t)\).
- If \(\hat{y}_t \neq y_t\), then the prediction was a “mistake”.
- See correct label \(y_t\); update \(f_t\) to get \(\hat{f}_{t+1}\).

Typically, update is **very computationally cheap** to compute.

“**Theorem**”: If there is a short \(w_*\) with \(\min_t y_t x_t^T w_* \geq 1\), then Online Perceptron makes a small number of mistakes.

“**Theorem**”: If there is a short \(w_*\) that makes a small number of “hinge loss mistakes”, then Online Perceptron makes a small number of mistakes.

(See reading assignment—we’ll discuss “hinge loss” in an upcoming lecture.)
What good is a small mistake bound?

Sequence of classifiers $\hat{f}_1, \hat{f}_2, \ldots$ is accurate (on average) in predicting labels of iid sequence $(X_1, Y_1), (X_2, Y_2), \ldots$

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
<th>$X_6$</th>
<th>$X_7$</th>
<th>$X_8$</th>
<th>$X_9$</th>
<th>$X_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{f}_1$</td>
<td>$\hat{f}_2$</td>
<td>$\hat{f}_3$</td>
<td>$\hat{f}_4$</td>
<td>$\hat{f}_5$</td>
<td>$\hat{f}_6$</td>
<td>$\hat{f}_7$</td>
<td>$\hat{f}_8$</td>
<td>$\hat{f}_9$</td>
<td>$\hat{f}_{10}$</td>
</tr>
<tr>
<td>$\hat{y}_1$</td>
<td>$\hat{y}_2$</td>
<td>$\hat{y}_3$</td>
<td>$\hat{y}_4$</td>
<td>$\hat{y}_5$</td>
<td>$\hat{y}_6$</td>
<td>$\hat{y}_7$</td>
<td>$\hat{y}_8$</td>
<td>$\hat{y}_9$</td>
<td>$\hat{y}_{10}$</td>
</tr>
<tr>
<td>$Y_1$</td>
<td>$Y_2$</td>
<td>$Y_3$</td>
<td>$Y_4$</td>
<td>$Y_5$</td>
<td>$Y_6$</td>
<td>$Y_7$</td>
<td>$Y_8$</td>
<td>$Y_9$</td>
<td>$Y_{10}$</td>
</tr>
</tbody>
</table>
What good is a small mistake bound?

- **Sequence of classifiers** $\hat{f}_1, \hat{f}_2, \ldots$ is accurate (on average) in predicting labels of iid sequence $(X_1, Y_1), (X_2, Y_2), \ldots$

- Combine classifiers $\hat{f}_1, \hat{f}_2, \ldots$ into a **single, accurate classifier** $\hat{f}$.
What good is a small mistake bound?

- **Sequence of classifiers** $\hat{f}_1, \hat{f}_2, \ldots$ is accurate (on average) in predicting labels of iid sequence $(X_1, Y_1), (X_2, Y_2), \ldots$

- **Combine classifiers** $\hat{f}_1, \hat{f}_2, \ldots$ into a **single, accurate classifier** $\hat{f}$.

This is achieved via *online-to-batch conversion*. 
Online-to-batch conversion

- Run online learning algorithm on sequence of examples
  \[(x_1, y_1), \ldots, (x_n, y_n)\]
  (in random order) to produce sequence of binary classifiers
  \[\hat{f}_1, \ldots, \hat{f}_{n+1},\]
  \[ (\hat{f}_i : \mathcal{X} \rightarrow \{\pm 1\}). \]
Online-to-batch conversion

- Run online learning algorithm on sequence of examples

\[ (x_1, y_1), \ldots, (x_n, y_n) \]

(in random order) to produce sequence of binary classifiers

\[ \hat{f}_1, \ldots, \hat{f}_{n+1}, \]

(\( \hat{f}_i : \mathcal{X} \to \{\pm 1\} \)).

- Final classifier: majority vote over the \( \hat{f}_i \)'s

\[
\hat{f}(x) := \begin{cases} 
-1 & \text{if } \sum_{i=1}^{n+1} \hat{f}_i(x) \leq 0, \\
+1 & \text{if } \sum_{i=1}^{n+1} \hat{f}_i(x) > 0.
\end{cases}
\]
Online-to-batch conversion

- Run online learning algorithm on sequence of examples
  \[(x_1, y_1), \ldots, (x_n, y_n)\]
  (in random order) to produce sequence of binary classifiers
  \[\hat{f}_1, \ldots, \hat{f}_{n+1},\]
  \[\hat{f}_i : \mathcal{X} \to \{\pm 1\}\].

- Final classifier: majority vote over the \[\hat{f}_i\]’s
  \[\hat{f}(x) := \begin{cases} -1 & \text{if } \sum_{i=1}^{n+1} \hat{f}_i(x) \leq 0, \\ +1 & \text{if } \sum_{i=1}^{n+1} \hat{f}_i(x) > 0. \end{cases} \]

There are many variants of online-to-batch conversions that make sense!
Practical suggestions for online-to-batch variant

Run online learning algorithm to make a few (e.g., two) passes over training examples, each time in a different random order.

Pass 1:
- $(x_2, y_2)$,
- $(x_9, y_9)$,
- $(x_6, y_6)$,
- $(x_1, y_1)$,
- $(x_3, y_3)$,
- $(x_4, y_4)$,
- $(x_{10}, y_{10})$,
- $(x_8, y_8)$,
- $(x_7, y_7)$,
- $(x_5, y_5)$

(Total of 2 rounds)

For linear classifiers, instead of using majority vote to combine, just average weight vectors.

\[ \hat{w} = \frac{1}{2n+1} \sum_{i=1}^{n} \hat{w}_i. \]

Don't use weight vectors from first pass through data.

\[ \hat{w} = \frac{1}{n+1} \sum_{i=n+1}^{2n+1} \hat{w}_i. \]

(Weight vectors at start are rubbish anyway.)
Practical suggestions for online-to-batch variant

▶ Run online learning algorithm to make a few (e.g., two) passes over training examples, each time in a different random order.

Pass 1:

$(x_2, y_2), (x_9, y_9), (x_6, y_6), (x_1, y_1), (x_3, y_3), (x_4, y_4), (x_{10}, y_{10}), (x_8, y_8), (x_7, y_7), (x_5, y_5)$

Pass 2:

$(x_4, y_4), (x_5, y_5), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_{10}, y_{10}), (x_7, y_7), (x_9, y_9), (x_6, y_6), (x_8, y_8)$

(Total of $2n$ rounds)
Practical suggestions for online-to-batch variant

- Run online learning algorithm to make a few (e.g., two) passes over training examples, each time in a different random order.

  Pass 1:
  \[(x_2, y_2), (x_9, y_9), (x_6, y_6), (x_1, y_1), (x_3, y_3), (x_4, y_4), (x_{10}, y_{10}), (x_8, y_8), (x_7, y_7), (x_5, y_5)\]

  Pass 2:
  \[(x_4, y_4), (x_5, y_5), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_{10}, y_{10}), (x_7, y_7), (x_9, y_9), (x_6, y_6), (x_8, y_8)\]

  (Total of \(2n\) rounds)

- For linear classifiers, instead of using majority vote to combine, just average weight vectors.

  \[
  \hat{\mathbf{w}} := \frac{1}{2n + 1} \sum_{i=1}^{2n+1} \hat{\mathbf{w}}_i.
  \]
Practical suggestions for online-to-batch variant

- Run online learning algorithm to make a few (e.g., two) passes over training examples, each time in a different random order.

  **Pass 1:**
  \((x_2, y_2), (x_9, y_9), (x_6, y_6), (x_1, y_1), (x_3, y_3), (x_4, y_4), (x_{10}, y_{10}), (x_8, y_8), (x_7, y_7), (x_5, y_5)\)

  **Pass 2:**
  \((x_4, y_4), (x_5, y_5), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_{10}, y_{10}), (x_7, y_7), (x_9, y_9), (x_6, y_6), (x_8, y_8)\)

  (Total of \(2n\) rounds)

- For linear classifiers, instead of using majority vote to combine, just average weight vectors.

  \[
  \hat{w} := \frac{1}{2n + 1} \sum_{i=1}^{2n+1} \hat{w}_i.
  \]

- Don’t use weight vectors from first pass through data.

  \[
  \hat{w} := \frac{1}{n + 1} \sum_{i=n+1}^{2n+1} \hat{w}_i.
  \]

  (Weight vectors at start are rubbish anyway.)
Example: restaurant review classification

Data:

- 1M reviews of Pittsburgh restaurants (from Yelp).
- $\mathcal{Y} = \{\text{at least 4 stars, below 4 stars}\}$
  
  (66.2% have at least 4 (out of 5) stars.)

- Reviews represented as bags-of-words:
  
  e.g., $x_{\text{great}} = \# \text{ times "great" appears in review.}$
  
  ($d \approx 200000$ unique words.)
Example: restaurant review classification

Data:

▶ 1M reviews of Pittsburgh restaurants (from Yelp).
▶ \( \mathcal{Y} = \{ \text{at least 4 stars, below 4 stars} \} \)
  (66.2\% have at least 4 (out of 5) stars.)
▶ Reviews represented as bags-of-words:
  e.g., \( x_{\text{great}} = \# \text{times “great” appears in review.} \)
  (\( d \approx 200000 \) unique words.)

Results

▶ Online Perceptron + affine expansion:
  ▶ Training risk: 23.3\%.
  ▶ Test risk (on 320123 held-out test examples): 23.4\%.
Example: restaurant review classification

Data:

- 1M reviews of Pittsburgh restaurants (from Yelp).
- \( \mathcal{Y} = \{ \text{at least 4 stars, below 4 stars} \} \)
  
  (66.2% have at least 4 (out of 5) stars.)
- Reviews represented as bags-of-words:
  
  e.g., \( x_{\text{great}} = \# \text{ times } \text{“great” appears in review.} \)
  
  \( (d \approx 200000 \text{ unique words.}) \)

Results

- Online Perceptron + affine expansion:
  
  - Training risk: 23.3%.
  - Test risk (on 320123 held-out test examples): 23.4%.

- Online Perceptron + affine expansion + online-to-batch variant:
  
  - Training risk: 10.1%.
  - Test risk (on 320123 held-out test examples): 10.5%.
  
  Running time: \( \approx 1 \text{ minute on single Intel Xeon 2.30GHz CPU.} \)
Other approaches for learning linear (or affine) classifiers

- Naïve Bayes generative model MLE
- Gaussian generative model MLE (sometimes)
- Methods for linear regression (+ thresholding)
Other approaches for learning linear (or affine) classifiers

- Naïve Bayes generative model MLE
- Gaussian generative model MLE (sometimes)
- Methods for linear regression (+ thresholding)
- Logistic regression, linear programming, Perceptron, Online Perceptron
Other approaches for learning linear (or affine) classifiers

- Naïve Bayes generative model MLE
- Gaussian generative model MLE (sometimes)
- Methods for linear regression (+ thresholding)
- Logistic regression, linear programming, Perceptron, Online Perceptron
- Support vector machines, etc. (Next lecture.)
Key takeaways

1. Logistic regression model.
2. Structure/geometry of linear classifiers.
4. Linear separability; two approaches to find a linear separator.
5. Online Perceptron; online-to-batch conversion.