LOGISTIC REGRESSION
**Binary classification**

### Linear classifiers

Given data \((x_i, y_i)\), where \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, +1\}\), a linear classifier takes a vector \(w \in \mathbb{R}^d\) and scalar \(w_0 \in \mathbb{R}\) and predicts

\[
y_i = f(x_i) = \text{sign}(x_i^T w + w_0).
\]

We discussed two methods last time:

- Least squares: Sensitive to outliers
- Perceptron: Convergence issues, assumes linear separability

Can we combine the separating hyperplane idea with probability to fix this?
Bayes linear classification

Linear discriminant analysis (a Bayes classifier)

We already saw an example of where a probabilistic interpretation gives a linear classification rule (with the definition $y \in \{0, 1\}$).

For the model $y \sim \text{Bern}(\pi)$ and $x \mid y \sim N(\mu_y, \Sigma)$, declare $y = 1$ given $x$ if

$$\ln \frac{p(x \mid y = 1)p(y = 1)}{p(x \mid y = 0)p(y = 0)} > 0.$$ 

In this case, the log odds is equal to

$$\ln \frac{p(x \mid y = 1)p(y = 1)}{p(x \mid y = 0)p(y = 0)} = \ln \frac{\pi_1}{\pi_0} - \frac{1}{2} (\mu_1 + \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0)$$

a constant $w_0$

$$+ x^T \Sigma^{-1}(\mu_1 - \mu_0)$$

a vector $w$
Log odds and Bayes classification

Original formulation

Recall that originally we wanted to declare $y = 1$ given $x$ if

$$\ln \frac{p(y = 1|x)}{p(y = 0|x)} > 0$$

- We didn’t have a way to define $p(y|x)$, so we used Bayes rule:
  $$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$
  and let the $p(x)$ cancel each other.

- We could then define $p(y)$ as Bernoulli (coin flip distribution) and $p(x|y)$ however we wanted (e.g., Gaussian with shared covariance).

Defining $p(y|x)$ directly is hard. Can we instead directly define the log odds?
Log odds and hyperplanes

Classifying \( x \) based on the log odds

\[
L = \ln \left( \frac{p(y = 1|x)}{p(y = 0|x)} \right),
\]

we notice that

1. \( L \gg 0 \): more confident \( y = +1 \),
2. \( L \ll 0 \): more confident \( y = -1 \),
3. \( L = 0 \): can go either way

The linear function \( x^T w + w_0 \) captures these three objectives:

- The distance of \( x \) to a hyperplane \( H \) defined by \((w, w_0)\) is \( \left| \frac{x^T w - w_0}{\|w\|_2} \right| \).
- The sign of the function captures which side \( x \) is on.
- As \( x \) moves away/towards \( H \), we become more/less confident.
Logistic link function

We can directly plug in the hyperplane representation for the log odds:

\[
\ln \frac{p(y = 1|x)}{p(y = 0|x)} = x^T w + w_0
\]

**Question:** What is different from the previous Bayes classifier?

**Answer:** There was a strict formula for calculating \( w \) and \( w_0 \) based on the prior model. Now, we put *no prior restrictions* on these values.

Setting \( p(y = 0|x) = 1 - p(y = 1|x) \), we solve for \( p(y = 1|x) \) to find that

\[
p(y = 1|x) = \frac{\exp\{x^T w + w_0\}}{1 + \exp\{x^T w + w_0\}} = \sigma(x^T w + w_0).
\]

- This is called the *sigmoid function*.
- We have chosen \( x^T w + w_0 \) as the *link function* for the log odds.
Red line: Sigmoid function $\sigma(x^Tw + w_0)$, which maps $x$ to $p(y = 1|x)$.

The function $\sigma(\cdot)$ vs the $(x^Tw + w_0)$-axis captures our desire to be more confident as we move away from the separating hyperplane.

(Blue dashed line: On a later slide.)
As with regression, absorb the offset: \( w \leftarrow \begin{bmatrix} w_0 \\ w \end{bmatrix} \) and \( x \leftarrow \begin{bmatrix} 1 \\ x \end{bmatrix} \).

**Definition**

Let \((x_1, y_1), \ldots, (x_n, y_n)\) be a set of binary labeled data. *Logistic regression* models the probability distribution of \( y \) conditioned on \( x \) as

\[
y_i | x_i \overset{iid}{\sim} \text{Bern}\{\sigma(x_i^T w)\}, \quad \sigma(x_i; w) = \frac{e^{x_i^T w}}{1 + e^{x_i^T w}}.
\]

**Discriminative vs Generative classifiers**

- This is referred to as a *discriminative* classifier because \( x \) is not directly modeled with a class-conditional probability distribution.

- Bayes classifiers are known as *generative* because \( x \) is modeled.

Discriminative: \( p(y|x) \) Generative: \( p(x|y)p(y) \).
**Logistic regression likelihood**

**Data likelihood**

Define $\sigma_i(w) = \sigma(x_i^Tw)$. The joint likelihood of $y_1, \ldots, y_n$ is

$$p(y_1, \ldots, y_n|x_1, \ldots, x_n, w) = \prod_{i=1}^{n} p(y_i|x_i, w)$$

$$= \prod_{i=1}^{n} \sigma_i(w)^{1(y_i=+1)} \{1 - \sigma_i(w)\}^{1(y_i=-1)}$$

- This is a Bernoulli process where $x$ modifies each probability of success.
- Predicting new data is the same:
  - If $x^Tw > 0$, then $\sigma(x^Tw) > 1/2$ and predict $y = +1$, and vice versa.
  - We now get a confidence in our prediction via the probability $\sigma(x^Tw)$. 
**Logistic regression and maximum likelihood**

**Maximum likelihood**

**Notation trick:** Use $\sigma_i(y_i \cdot w) = \sigma_i(w)^{1(y_i=+1)} \{1 - \sigma_i(w)\}^{1(y_i=-1)}$ and find

$$\hat{w}_{\text{ML}} = \arg \max_w \sum_{i=1}^n \ln \sigma_i(y_i \cdot w) = \arg \max_w \mathcal{L}$$

As with the Perceptron, we can’t directly set $\nabla_w \mathcal{L} = 0$, so we need an iterative algorithm. At step $t$, we can update

$$w^{(t+1)} = w^{(t)} + \eta \nabla_w \mathcal{L}, \quad \nabla_w \mathcal{L} = \sum_{i=1}^n \{1 - \sigma_i(y_i \cdot w)\} y_i x_i.$$  

We will see that this results in an algorithm very similar to the Perceptron.
Logistic regression algorithm (steepest ascent)

**Input:** Training data \((x_1, y_i), \ldots, (x_n, y_n)\) and step size \(\eta \in (0, 1]\)

1. Set \(w^{(1)} = \vec{0}\)

2. For step \(t = 1, 2, \ldots\) do
   - Update \(w^{(t+1)} = w^{(t)} + \eta \sum_{i=1}^{n} \{1 - \sigma_i(y_i \cdot w)\}y_ix_i\)

**Perceptron:** Search for misclassified \((x_i, y_i)\), update \(w^{(t+1)} = w^{(t)} + \eta y_ix_i\).

**Logistic regression:** Though it’s usually not done, we *could* pick a \((x_i, y_i)\) and update \(w^{(t+1)} = w^{(t)} + \eta \{1 - \sigma_i(y_i \cdot w)\}y_ix_i\).

- Recall that \(\sigma_i(y_i \cdot w)\) automatically picks out the probability for the right event.
- Therefore \(1 - \sigma_i(y_i \cdot w)\) is the probability the classifier gets it wrong.
- Perceptron is “all-or-nothing,” so \(1 - \sigma_i(y_i \cdot w) \rightarrow \{0, 1\}\).
- Logistic regression has a probability “fudge-factor.”
LOGISTIC REGRESSION ALGORITHM (NEWTON)

**Input:** Training data \((x_1, y_i), \ldots, (x_n, y_n)\) and step size \(\eta \in (0, 1]\)

1. Set \(w^{(1)} = \vec{0}\)

2. For step \(t = 1, 2, \ldots\) do
   - Update \(w^{(t+1)} = w^{(t)} + \eta(-\nabla^2_w \mathcal{L})^{-1} \nabla_w \mathcal{L}\)

In practice, people often use “Newton’s method.” The matrix of second derivatives \(\nabla^2_w \mathcal{L}\) is called the “Hessian” and equals

\[
\nabla^2_w \mathcal{L} = -\sum_{i=1}^{n} \sigma_i(w) \{1 - \sigma_i(w)\} x_i x_i^T.
\]

Compare with *steepest ascent* which only uses the first derivative.

We won’t discuss in detail today, but Newton’s method helps step in better directions and can speed up convergence. It is widely used.
**Problem:** If a hyperplane can separate all training data, then $\|\hat{w}_{ML}\|_2 \to \infty$. This drives $\sigma_i(y_i \cdot w) \to 1$ for each $(x_i, y_i)$.

Even for nearly separable data it might get a few very wrong in order to be more confident about the rest. This is a case of “over-fitting.”

**A solution:** Regularize $w$ with $\lambda w^T w$:

$$\hat{w}_{MAP} = \arg \max_w \sum_{i=1}^{n} \ln \sigma_i(y_i \cdot w) - \lambda w^T w$$

We’ve seen how this corresponds to a Gaussian prior distribution on $w$.

How about the posterior $p(w|X, Y)$?
Laplace approximation
Bayesian logistic regression

Posterior calculation

Define the prior distribution on $w$ to be $w \sim N(0, \lambda^{-1}I)$. The posterior is

$$p(w|X, Y) = \frac{p(w) \prod_{i=1}^{n} \sigma_i(y_i \cdot w)}{\int p(w) \prod_{i=1}^{n} \sigma_i(y_i \cdot w) \, dw}$$

This is not a “standard” distribution and we can’t calculate the denominator. A major reason is that the prior is not conjugate to the likelihood.

A conjugate prior results in a closed-form posterior distribution in the same family as the prior. We saw one with Bayesian linear regression.

Therefore we can’t actually say what $p(w|X, Y)$ is. Can we approximate it?
Measuring closeness between distributions

Can we just pick a distribution for $p(w|X, Y)$ and set its parameters to approximate the true posterior?

We can, but we need a measure of what it means for two distributions to be “close.” We use the Kullback-Leibler (KL) divergence.

Let $q(w)$ approximate $p(w|X, Y)$. Their KL-divergence is

$$KL(q||p) = \int q(w) \ln \frac{q(w)}{p(w|X, Y)} dw .$$

The KL-divergence can be thought of as a distance between $q$ and $p$ because

- $KL(q||p) \geq 0$, the more different $q$ and $p$ are, the larger KL
- $KL(q||p) = 0$ when $q = p$, i.e., the same exact distribution
LAPLACE APPROXIMATION

Picking $q(w)$
Let’s approximate $p(w|X, Y)$ with the distribution $q(w) = N(w|\mu, \Sigma)$. Minimizing their KL-divergence entails calculating

$$\text{KL}(q||p) = \int q(w) \ln q(w) dw - \int q(w) \left( \sum_{i=1}^{n} \ln \sigma_i (y_i \cdot w) - \frac{\lambda}{2} w^T w \right) + \text{const.}$$

= negative of entropy of $q$
= $\ln p(y|X,w)p(w) = f(w)$

Taylor expansions
We still have a problem with the second integral. We use a 2nd order Taylor expansion to approximate “$f(w)$.”

$$f(w) \approx f(z) + (w - z)^T \nabla f(z) + \frac{1}{2} (w - z)^T (\nabla^2 f(z)) (w - z)$$

This is a general technique called the Laplace approximation.
1. Define \( q(w) = N(\hat{w}_{MAP}, \Sigma) \), so we only need to learn \( \Sigma \).

2. Taylor expand \( f(w) \) about the point \( \hat{w}_{MAP} \) and plug into the KL equation:

\[
KL(q||p) \approx \int q(w) \ln q(w) dw + \text{const.} \\
= -\frac{1}{2} \ln |\Sigma| - \frac{1}{2} \ln 2\pi e \\
- \int q(w)f(\hat{w}_{MAP}) dw \\
\quad = f(\hat{w}_{MAP}) \\
- \int q(w)(w - \hat{w}_{MAP})^T \nabla f(\hat{w}_{MAP}) dw \\
\quad = 0 \\
- \int \frac{1}{2} q(w)(w - \hat{w}_{MAP})^T (\nabla^2 f(\hat{w}_{MAP}))(w - \hat{w}_{MAP}) dw \\
\quad = \frac{1}{2} \text{trace} \{ \nabla^2 f(\hat{w}_{MAP}) \mathbb{E}_q[(w - \hat{w}_{MAP})(w - \hat{w}_{MAP})^T] \} \\
\quad = \Sigma
\]
Summary: General derivation

- We have the following approximation to the KL-divergence:

\[
\tilde{KL}(q||p) = -\frac{1}{2} \ln |\Sigma| - \frac{1}{2} \text{trace}\{\nabla^2 f(\hat{w}_{\text{MAP}}) \Sigma\} + \text{const.}
\]

- Setting the derivative to zero, we find:

\[
\nabla_{\Sigma} \tilde{KL}(q||p) = -\frac{1}{2} \Sigma^{-1} - \frac{1}{2} \nabla^2 f(\hat{w}_{\text{MAP}}) = 0
\]

\[
\Sigma = -[\nabla^2 f(\hat{w}_{\text{MAP}})]^{-1}
\]

- Therefore, given data \( \mathcal{D} \) the Laplace approximation of \( p(w|\mathcal{D}) \) is

\[
N(\hat{w}_{\text{MAP}}, -[\nabla^2 f(\hat{w}_{\text{MAP}})]^{-1})
\]

where \( f(w) = \ln p(\mathcal{D}|w)p(w) \).
Laplace approximation for logistic regression

Given labeled data \((x_1, y_1), \ldots, (x_n, y_n)\) and the likelihood and prior model

\[
y_i \sim \text{Bern}\{\sigma_i(w)\}, \quad w \sim N(0, \lambda^{-1}I), \quad \sigma_i(w) = \sigma(x_i^T w)
\]

to approximate the posterior of \(w\):

1. Find: \(\hat{w}_{\text{MAP}} = \arg \max_w \sum_{i=1}^{n} \ln \sigma_i(y_i \cdot w) - \frac{\lambda}{2} w^T w\)

2. Set: \(M = -\lambda I - \sum_{i=1}^{n} \sigma_i(y_i \cdot \hat{w}_{\text{MAP}}) \{1 - \sigma_i(y_i \cdot \hat{w}_{\text{MAP}})\} x_i x_i^T\)

3. Approximate: \(p(w|X, Y) = N\left(\hat{w}_{\text{MAP}}, -M^{-1}\right)\).
PROBIT CLASSIFICATION
GENERALIZATIONS

Cumulative distribution functions
The sigmoid $\sigma(x^Tw)$ works because:

- $\sigma(x^Tw) \to 1$ as $x^Tw \to +\infty$
- $\sigma(x^Tw) \to 0$ as $x^Tw \to -\infty$
- $\sigma(x^Tw)$ transitions smoothly in between

i.e., $\sigma(x^Tw)$ can be interpreted as a *cumulative distribution function* (CDF). This suggests that we can pick other CDF’s instead.

Probit classification replaces $\sigma(x^Tw) = \frac{e^{x^Tw}}{1+e^{x^Tw}}$ with $\Phi\left(\frac{x^Tw}{\xi}\right)$, the CDF of a univariate Gaussian with mean 0 and variance 1 evaluated at $\frac{x^Tw}{\xi}$.

For the right $\xi$, the probit CDF and logistic CDF are very close (see figure).
**Probit classification**

**Definition**

Let \((x_1, y_1), \ldots, (x_n, y_n)\) be a set of binary labeled data. *Probit regression* models the probability distribution of \(y\) conditioned on \(x\) as

\[
y_i | x_i \overset{iid}{\sim} \text{Bern}\{\Phi\left(\frac{x_i^T w}{\xi}\right)\}, \quad \Phi\left(\frac{x_i^T w}{\xi}\right) = \int_{-\infty}^{x_i^T w} \frac{1}{(2\pi\xi)^{1/2}} e^{-\frac{1}{2\xi^2}z^2} dz.
\]

**Latent variables**

Directly finding \(\hat{w}_{\text{MAP}}\) is difficult. Probit classifiers introduce a *latent variable* (a very general concept) that results in an *equivalent* model:

\[
y_i | z_i = \text{sign}(z_i), \quad z_i | x_i, w \sim N(x_i^T w, \xi^2).
\]

- Learning this model requires a technique called the “EM algorithm.”
- For now, we can notice that classification has become regression again.
- However, this time we’re learning \(z_i\) instead of being given it.