Logistic Regression for Classification

Fraida Fund

Contents

In this lecture

- Linear classifiers
- Logistic regression
- Fitting logistic regression

Classification

Suppose we have a series of data points $\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\}$ and there is some (unknown) relationship between $\mathbf{x_i}$ and y_i.

- **Classification**: The output variable y is constrained to be $\in 1, 2, \cdots, K$
- **Binary classification**: The output variable y is constrained to be $\in 0, 1$

Linear classifiers

Binary classification with linear decision boundary

- Plot training data points
- Draw a line (**decision boundary**) separating 0 class and 1 class
- If a new data point is in the **decision region** corresponding to class 0, then $\hat{y} = 0$.
- If it is in the decision region corresponding to class 1, then $\hat{y} = 1$.

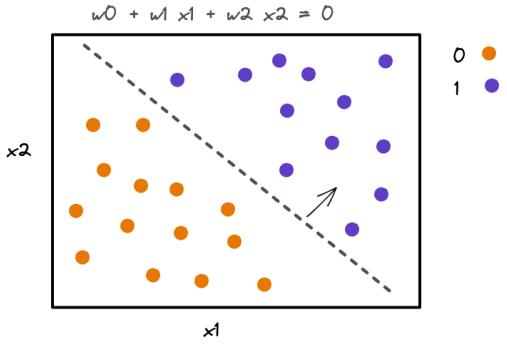

Figure 1: Binary classification problem with linear decision boundary.

Linear classification rule

- Given a **weight vector**: $\mathbf{w} = (w_0, \cdots, w_d)$
- Compute linear combination $z = w_0 + \sum_{j=1}^{d} w_d x_d$
- Predict class:

$$\hat{y} = \begin{cases} 1, z > 0 \\ 0, z \leq 0 \end{cases}$$

Multi-class classification: illustration

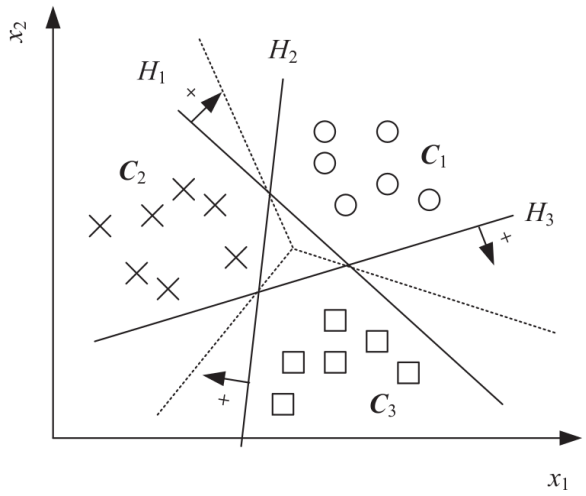

Figure 2: Each hyperplane H_i separates the examples of C_i from the examples of all other classes.

Linear separability

Given training data

$$(\mathbf{x}_i, y_i), i = 1, \cdots, N$$

The problem is **perfectly linearly separable** if there exists a **separating hyperplane** H_i such that all $\mathbf{x} \in C_i$ lie on its positive side, and all $\mathbf{x} \in C_j, j \neq i$ lie on its negative side.

Non-uniqueness of separating hyperplane

When a separating hyperplane exists, it is not unique (there are in fact infinitely many such hyperplanes.)

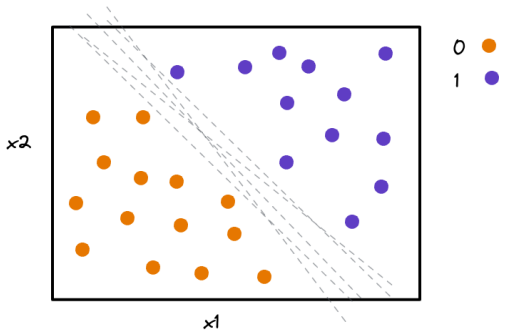

Figure 3: Several separating hyperplanes.

Non-existence of perfectly separating hyperplane

Many datasets *not* linearly separable - some points will be misclassified by *any* possible hyperplane.

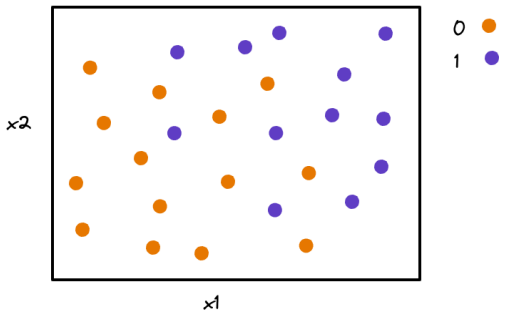

Figure 4: This data is not separable.

Choosing a hyperplane

Which hyperplane to choose?

We will try to find the hyperplane that minimizes loss according to some **loss function**.

Will revisit several times this semester.

Logistic regression

Probabilistic model for binary classification

Instead of looking for a model f so that

$$y_i \approx f(x_i)$$

we will look for an f so that

$$P(y_i = 1|x_i) = f(x_i), P(y_i = 0|x_i) = 1 - f(x_i)$$

We need a function that takes a real value and maps it to range $[0, 1]$. What function should we use?

Logistic/sigmoid function

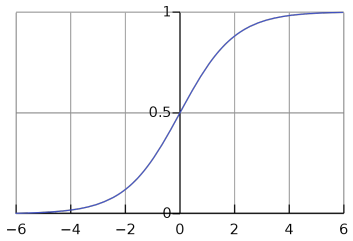

Figure 5: $\sigma(z) = \frac{1}{1+e^{-z}}$ is a classic "S"-shaped function.

Why the sigmoid/logistic function? Among its other nice properties, its derivative is $\sigma(1 - \sigma)$, which makes for an easy update rule for gradient descent.

Also note the intuitive relationship behind this function's output and the distance from the linear separator (the argument that is input to the function).

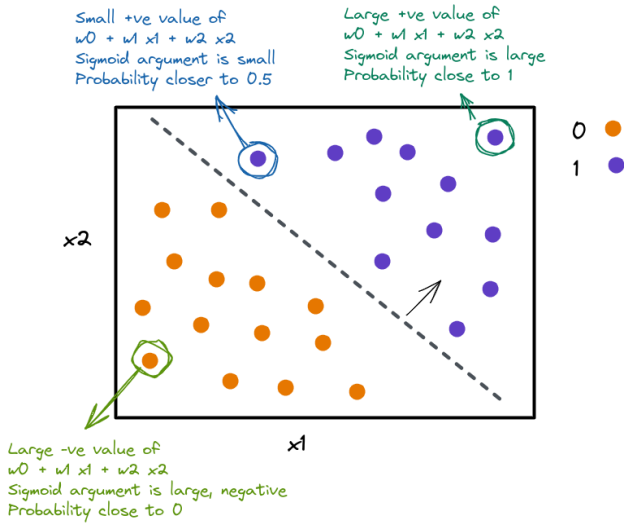

Figure 6: Output is close to 0 or 1 if the argument to the σ has large magnitude (point is far from separating hyperplane, but closer to 0.5 if the argument is small (point is near separating hyperplane).

Logistic function for binary classification

Let $z = w_0 + \sum_{j=1}^{d} w_d x_d$, then

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-z}}, \quad P(y = 0|\mathbf{x}) = \frac{e^{-z}}{1 + e^{-z}}$$

(note: $P(y = 1) + P(y = 0) = 1$)

Logistic function with threshold

Choose a threshold t, then

$$\hat{y} = \begin{cases} 1, & P(y = 1|\mathbf{x}) \geq t \\ 0, & P(y = 1|\mathbf{x}) < t \end{cases}$$

Logistic model as a "soft" classifier

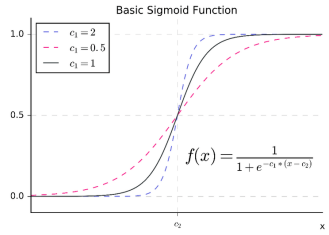

Figure 7: Plot of $P(y = 1|x) = \frac{1}{1+e^{-z}}, z = w_1 x$. As $w_1 \to \infty$ the logistic model becomes a "hard" rule.

Logistic classifier properties (1)

- Class probabilities depend on distance from separating hyperplane
- Points far from separating hyperplane have probability ≈ 0 or ≈ 1
- When $||\mathbf{w}||$ is larger, class probabilities go towards extremes (0,1) more quickly

Logistic classifier properties (2)

- Unlike linear regression, weights do *not* correspond to change in output associated with one-unit change in input.
- Sign of weight *does* tell us about relationship between a given feature and target variable.
- Can add a regularization penalty to cost function to reduce variance, just like with linear regression.

Logistic regression - illustration

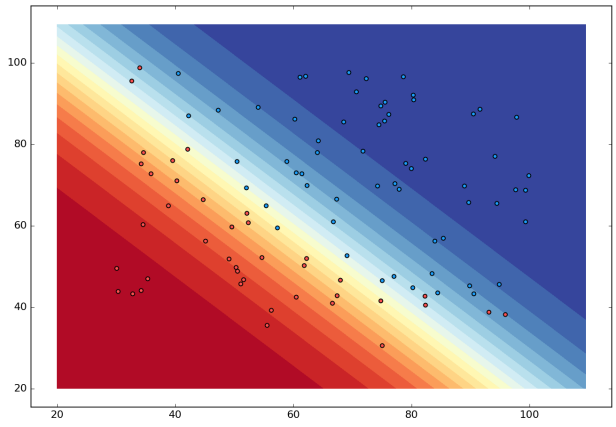

Figure 8: Logistic regression, illustrated with contour plot.

Multi-class logistic regression

Suppose $y \in 1, \ldots, K$. We use:

- $\mathbf{W} \in R^{K \times d}$ (parameter matrix)
- $\mathbf{z} = \mathbf{W}\mathbf{x}$ (K linear functions)

Assume we have stacked a 1s column so that the intercept is rolled into the parameter matrix.

Softmax function

$$g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

- Takes as input a vector of K numbers
- Outputs K probabilities proportional to the exponentials of the input numbers.

Softmax function as a PMF

Acts like a probability mass function:

- $g_k(\mathbf{z}) \in [0, 1]$ for each k
- $\sum_{k=1}^{K} g_k(\mathbf{z}) = 1$
- larger input corresponds to larger "probability"

Softmax function for multi-class logistic regression (1)

Class probabilities are given by

$$P(y = k|\mathbf{x}) = \frac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

Softmax function for multi-class logistic regression (2)

When $z_k \gg z_\ell$ for all $\ell \neq k$:

- $g_k(\mathbf{z}) \approx 1$
- $g_\ell(\mathbf{z}) \approx 0$ for all $\ell \neq k$

Assign highest probability to class k when z_k is largest.

Fitting logistic regression model

We know that to fit weights, we need

- a loss function,
- and a training algorithm to find the weights that minimize the loss function.

Learning logistic model parameters

Weights $\mathbf{W}$ are the unknown **model parameters**:

$$\mathbf{z} = \mathbf{W}\mathbf{x}, \mathbf{W} \in R^{K \times d}$$

$$P(y = k|\mathbf{x}) = g_k(\mathbf{z}) = g_k(\mathbf{W}\mathbf{x})$$

Given training data $(\mathbf{x}_i, y_i), i = 1, \dots, n$, we must learn $\mathbf{W}$.

Maximum likelihood estimation (1)

Let $P(\mathbf{y}|\mathbf{X}, \mathbf{W})$ be the probability of observing class labels $\mathbf{y} = (y_1, \dots, y_n)^T$ given inputs $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$ and weights $\mathbf{W}$.

The **maximum likelihood estimate** is

$$\hat{\mathbf{W}} = \underset{W}{\text{argmax}}\, P(\mathbf{y}|\mathbf{X}, \mathbf{W})$$

It is the estimate of parameters for which these observations are most likely.

Maximum likelihood estimation (2)

Assume outputs y_i are independent of one another,

$$P(\mathbf{y}|\mathbf{X}, \mathbf{W}) = \prod_{i=1}^{n} P(y_i|\mathbf{x_i}, \mathbf{W})$$

We take the log of both sides, because then the product turns into a sum...

Maximum likelihood estimation (3)

Define the **negative log likelihood**:

$$\begin{aligned} L(\mathbf{W}) &= -\ln P(\mathbf{y}|\mathbf{X}, \mathbf{W}) \\ &= -\sum_{i=1}^{n} \ln P(y_i|\mathbf{x_i}, \mathbf{W}) \end{aligned}$$

(the term in the sum is also called cross-entropy)

Note that maximizing the likelihood is the same as minimizing the negative log likelihood.

Maximum likelihood estimation (4)

Now we can re-write max likelihood estimator with a loss function to minimize:

$$\hat{\mathbf{W}} = \underset{W}{\text{argmax}}\, P(\mathbf{y}|\mathbf{X}, \mathbf{W}) = \underset{W}{\text{argmin}}\, L(\mathbf{W})$$

The next step will be to plug in our sigmoid function.

Binary cross-entropy loss (1)

For binary classification with class labels $0, 1$:

$$\begin{aligned}
\ln P(y_i|\mathbf{x_i}, \mathbf{w}) &= y_i \ln P(y_i = 1|\mathbf{x_i}, \mathbf{w}) + (1 - y_i) \ln P(y_i = 0|\mathbf{x_i}, \mathbf{w}) \\
&= y_i \ln \sigma(z_i) + (1 - y_i) \ln(1 - \sigma(z_i)) \\
&= y_i(\ln \sigma(z_i) - \ln \sigma(-z_i)) + \ln \sigma(-z_i) \\
&= y_i \ln \frac{\sigma(z_i)}{\sigma(-z_i)} + \ln \sigma(-z_i) \\
&= y_i \ln \frac{1 + e^{z_i}}{1 + e^{-z_i}} + \ln \sigma(-z_i) \\
&= y_i \ln \frac{e^{z_i}(e^{-z_i} + 1)}{1 + e^{-z_i}} + \ln \sigma(-z_i) \\
&= y_i z_i - \ln(1 + e^{z_i})
\end{aligned} \tag{1}$$

(Note: $\sigma(-z) = 1 - \sigma(z)$)

Binary cross-entropy loss (2)

Binary cross-entropy loss function (negative log likelihood):

$$\sum_{i=1}^{n} \ln(1 + e^{z_i}) - y_i z_i$$

Cross-entropy loss for multi-class classification (1)

Define "one-hot" vector - for a sample from class k, all entries in the vector are 0 except for the kth entry which is 1:

$$r_{ik} = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases}$$

$$i = 1, \ldots, n, \quad k = 1, \ldots, K$$

Cross-entropy loss for multi-class classification (2)

Then,

$$\ln P(y_i|\mathbf{x_i}, \mathbf{W}) = \sum_{k=1}^{K} r_{ik} \ln P(y_i = k|\mathbf{x_i}, \mathbf{W})$$

Cross-entropy loss function is

$$\sum_{i=1}^{n} \left[\ln \left(\sum_k e^{z_{ik}} \right) - \sum_k z_{ik} r_{ik} \right]$$

Minimizing cross-entropy loss

To minimize, we would take the partial derivative:

$$\frac{\partial L(W)}{\partial W_{kj}} = 0$$

for all W_{kj}

But, there is no closed-form expression - can only estimate weights via numerical optimization (e.g. gradient descent)

"Recipe" for logistic regression (binary classifier)

- Choose a **model**:

$$P(y=1|x,w)=\sigma\left(w_0+\sum_{i=1}^{d}w_dx_d\right)$$

$$\widehat{y}=\begin{cases}1, & P(y=1|\mathbf{x})\geq t\\ 0, & P(y=1|\mathbf{x})<t\end{cases}$$

- Get **data** - for supervised learning, we need **labeled** examples: $(x_i,y_i), i=1,2,\cdots,n$
- Choose a **loss function** that will measure how well model fits data: binary cross-entropy

$$\sum_{i=1}^{n}\ln(1+e^{z_i})-y_iz_i$$

- Find model **parameters** that minimize loss: use numerical optimization to find weight vector w
- Use model to **predict** $\widehat{y}$ for new, unlabeled samples.

"Recipe" for logistic regression (multi-class classifier)

- Choose a **model**: find probability of belonging to each class, then choose the class for which the probability is highest.

$$P(y=k|\mathbf{x})=\frac{e^{z_k}}{\sum_{\ell=1}^{K}e^{z_\ell}} \text{ where } \mathbf{z}=\mathbf{W}\mathbf{x}$$

- Get **data** - for supervised learning, we need **labeled** examples: $(x_i,y_i), i=1,2,\cdots,n$
- Choose a **loss function** that will measure how well model fits data: cross-entropy

$$\sum_{i=1}^{n}\left[\ln\left(\sum_k e^{z_{ik}}\right)-\sum_k z_{ik}r_{ik}\right] \text{ where}$$

$$r_{ik}=\begin{cases}1 & y_i=k\\ 0 & y_i\neq k\end{cases}$$

- Find model **parameters** that minimize loss: use numerical optimization to find weight vector w
- Use model to **predict** $\widehat{y}$ for new, unlabeled samples.