classification: logistic regression
regression with two classes

• With linear regression, we model the relationship between features and target with a linear equation:

\[ \hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m \]

• Now, suppose we have two classes, i.e., \( y \in \{0, 1\} \). We could use linear regression, but …

  • it will treat the classes as numbers, interpolating between the points
  • it cannot be interpreted as a probability
  • how would we generalize to multiple classes?

• Need a decision threshold, i.e., \( y = 0.5 \)

• In this case, we would never predict the class \( y = 0 \), regardless of what \( x \) is!
Instead of fitting a hyperplane (a line generalized to more than one dimension), use the logistic function

\[ g(v) = \frac{1}{1 + e^{-v}} \]

to translate the output of linear regression to between 0 (as \( v \to -\infty \)) and 1 (as \( v \to \infty \))

- Note that \( 1 - g(v) = \frac{e^{-v}}{1 + e^{-v}} \) (useful for derivations)

- This converts the outputs to probabilities:

\[ f_\beta(x) = g(\beta_0 + \beta^T x) = P(y = 1 \mid x) \]
\[ = \frac{1}{1 + \exp(- (\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m ))} \]

- Now the decision rule
- \( \hat{y}(x) \geq 0.5 \to \hat{y} = 1 \)
- \( \hat{y}(x) < 0.5 \to \hat{y} = 0 \)

has a probabilistic interpretation
interpreting coefficients

• In linear regression, the effect of a coefficient is clear: $\beta_j x_j$ means for every unit change in $x_j$, the model changes by $\beta_j$.

• For logistic regression, we need to find a different interpretation, since the weights no longer have a linear effect.

• Consider the odds, i.e., the ratio $P(y = 1 \mid x)/P(y = 0 \mid x)$:

$$
\frac{P(y = 1 \mid x)}{P(y = 0 \mid x)} = \frac{1}{1 + \exp(- (\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m))} \cdot \frac{1}{\exp(- (\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m))} = \exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m)
$$
interpreting coefficients

• Then we consider the ratio of the odds when \( x_j \) is increased by 1:

\[
\frac{\text{odds}_{x_j+1}}{\text{odds}_{x_j}} = \frac{\exp(\cdots + \beta_j(x_j + 1) + \cdots)}{\exp(\cdots + \beta_j x_j + \cdots)} = e^{\beta_j}
\]

• Thus, a unit change in \( x_{ij} \) corresponds to a factor \( e^{\beta_j} \) change in the odds
  - \( e^{\beta_j} > 1 \): \( x_j \) increases the odds
  - \( e^{\beta_j} < 1 \): \( x_j \) decreases the odds

• Consider

\[
\hat{y} = \frac{1}{1 + \exp( - (3 + 2x_1 + 0.5x_2 - 3x_3))}
\]

• For this model ...
  - \( x_1 \) and \( x_2 \) increase the odds
  - \( x_3 \) decreases the odds
  - \( x_3 \) has the largest factor impact on the odds (assuming the features are normalized!)
training logistic regression

• With linear regression, we can derive a closed-form solution for the parameters in terms of the least-squares equations

• For logistic regression, let’s consider the likelihood of the model over data samples $i = 1, \ldots, n$:

$$L(\beta) = \prod_{i=1}^{n} p(y_i | x_i, \beta) = \prod_{i=1}^{n} (f_\beta(x_i))^{y_i} \cdot (1 - f_\beta(x_i))^{1-y_i}$$

when $y_i = 1$, we want to maximize $f_\beta(x_i)$, and
when $y_i = 0$, we want to maximize $1 - f_\beta(x_i)$

• And then the log likelihood, which is easier to optimize (like we did with GMMs):

$$l(\beta) = \sum_{i=1}^{n} \log \left[ (f_\beta(x_i))^{y_i} \cdot (1 - f_\beta(x_i))^{1-y_i} \right] = \sum_{i=1}^{n} \left[ y_i \log f_\beta(x_i) + (1 - y_i) \log(1 - f_\beta(x_i)) \right]$$

• There is no (known) closed form solution to maximize $l(\beta)$, given the $\log f_\beta(x_i)$ terms
gradient descent (ascent)

- We want to find $\beta$ to maximize $l(\beta)$ but no closed-form exists.

- Consider the **gradient descent (ascent)** algorithm, an iterative procedure for finding a **local minimum (maximum)** of a function by moving away from (towards) the gradient:

  $$
  \beta_{j+1}^{t} = \beta_{j}^{t} - \alpha^{t} \frac{\partial}{\partial \beta_{j}} l(\beta^{t}), \quad \beta_{j+1}^{t} = \beta_{j}^{t} + \alpha^{t} \frac{\partial}{\partial \beta_{j}} l(\beta^{t})
  $$

- Here, $\alpha^{t}$ is the **step size** of the algorithm at time $t$

- Since $l(\beta)$ is a **concave** function, we can guarantee that gradient ascent will eventually converge to the **global maximum**, so long as certain conditions on $\alpha^{t}$ are met

for non-convex functions, no guarantee of convergence to optimum

in general, the step size must be tuned correctly

$\eta$ too small    $\eta$ too large

In-sample Error, $E_{s}$

Weights, $w$
Suppose we have a single parameter $b$ for some model we are trying to train. For this model, we find a log-likelihood function of

$$l(b) = - \left( \frac{b - m}{s} \right)^2$$

where $m$ and $s$ are constants. Derive the iterative procedure for determining the model parameters as a function of the step size $\alpha^t$. Run the procedure for different values of $\alpha^t$ until $t = 10$ and compare the results.
We always want to maximize the log-likelihood, so we use gradient ascent. Letting $b^t$ be the value of $b$ at iteration $t$, our update procedure will be:

$$b^{t+1} = b^t + \alpha^t \frac{d}{db} l(b^t)$$

Evaluating the derivative, this becomes:

$$b^{t+1} = b^t - 2\alpha^t \left( \frac{b^t - m}{s^2} \right)$$

Suppose $\alpha = 0.1$, $m = 5$, $s = 0.7$. If we start at $b^0 = 1.1$ (arbitrary), we get

$$b^1 = 1.1 - 2 \cdot 0.1 \cdot \left( \frac{1.1 - 5}{0.7^2} \right) = 2.692$$
$$b^2 = 2.692 - 2 \cdot 0.1 \cdot \left( \frac{2.692 - 5}{0.7^2} \right) = 3.634$$
$$\ldots$$
$$b^{10} = 4.965 - 2 \cdot 0.1 \cdot \left( \frac{4.965 - 5}{0.7^2} \right) = 4.979$$
solution

Below, we plot the evolution of $b^t$ over $t$ (see the Jupyter notebook), starting with $b^0 = 1.1$ for $\alpha^t = 0.01, 0.05, 0.1, 0.2, 0.4, 0.5$. Again, we set $m = 5$ and $s = 0.7$.

Here, the $y$-axis is actually $-l(b)$, to make the values positive. Maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood.

Tuning $\alpha^t$ is a very important question!
gradient ascent for logistic regression

- Back to logistic regression. Evaluating the partial derivative,

\[
\frac{\partial}{\partial \beta_j} l(\beta) = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{n} \left[ y_i \log f_\beta(x_i) + (1 - y_i) \log(1 - f_\beta(x_i)) \right]
\]

Partial derivative of loss with respect to \( f_\beta(x_i) \)

\[
= \sum_{i=1}^{n} \left( \frac{y_i}{f_\beta(x_i)} - \frac{1 - y_i}{1 - f_\beta(x_i)} \right) \frac{\partial}{\partial \beta_j} f_\beta(x_i)
\]

Partial derivative of logistic function \( g(v) \)

with respect to \( v_i \equiv \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots \)

\[
= \sum_{i=1}^{n} \left( \frac{y_i}{f_\beta(x_i)} - \frac{1 - y_i}{1 - f_\beta(x_i)} \right) f_\beta(x_i)(1 - f_\beta(x_i)) \frac{\partial}{\partial \beta_j} (\cdots + \beta_j x_{ij} + \cdots)
\]

Partial derivative of \( v_i \)

with respect to \( \beta_j \)

\[
\frac{\partial}{\partial v} g(v) = \frac{\partial}{\partial v} (1 + e^{-v})^{-1} = -(1 + e^{-v})^{-2} e^{-v}(-1) = g(v)(1 - g(v))
\]

- Thus, we get the following gradient ascent rule for logistic regression:

\[
\beta_j^{t+1} = \beta_j^t + \alpha^t \sum_{i=1}^{n} (y_i - f_\beta(x_i)) x_{ij}
\]
in python

- from sklearn.linear_model import LogisticRegression


- Most methods (fit, predict, ...) are the same as linear regression

- One difference: Regularization parameter $C$
  - Higher $C$: Less regularization
  - Lower $C$: More regularization

```python
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
metrics.accuracy_score(y_test, y_pred)
```