JNOTES

CREATIVE NOTES

Logistic Regression is a discriminative approch for classification. In fact unlike generative model that try to model how data is generated ($P(x=x|C=c)$) logistic regression directly models the probability of class membership (nonmovement) given the feature ($P(C=c|X=x)$).

So generative models have the necessity to learn how data was create because for computing the actual probability $P(C|X)$ they use the Bayes rule, so they compute (extime) $P(x,c) = P(x|c)P(c)$ where $P(x|c)$ = How data is generate with each class

The discriminative models instead directly model the posterior distribution $c|x$

Binary Classification

In generative approch with gaussian distribution having the same covariance matrix (tide Covariance) we discovered that the optimal decision boundary is linear and can be written as:

$$\log\frac{P(C=h_1|x)}{P(C=h_0|x)} = \log\frac{f_{x|c}(x|h_1)}{f_{x|c}(x|h_0)} + \log\frac{\pi}{1-\pi} = w^T x + b = \text{log-odds}$$

- $\log\frac{P(C=h_1|x)}{P(C=h_0|x)}$: log posterior probability
- $\log\frac{f_{x|c}(x|h_1)}{f_{x|c}(x|h_0)}$: log likelihood ratio
- $\log\frac{\pi}{1-\pi}$: log prior ratio $\frac{P(C=h_1)}{P(C=h_0)}$

So the log-odds is a linear function of our input feature.

Flipping the Approch with logistic Regression

(generative discriminative)

With logistic regression we assume that **our decision boundary should be linear**

We need to convert this linear score into a probability

if log-odds equal $w^T x + b$

$$\log\frac{P(C=h_1|x)}{P(C=h_0|x)} = w^T x + b \Rightarrow \frac{P(C=h_1|x)}{P(C=h_0|x)} = e^{w^T x + b}$$

if we want to compute $P(C=h_1|x)$ since we know that $P(C=h_0|x) = 1 - P(C=h_1|x)$

$$= P(C=h_1|x) = e^{w^T x + b}\left[1 - P(C=h_1|x)\right]$$

$$= P(C=h_1|x)\left(1 + e^{w^T x + b}\right) = e^{w^T x + b} =$$

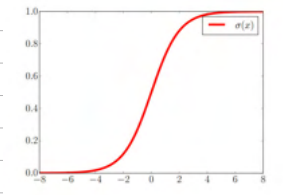
$$\Rightarrow \boxed{P(C=h_1|x) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}} = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)}$$

$\sigma(w x^T + b)$ is called sigmoid function or logistic function

Note is more appropiate to call $P(C=h_1|x)$ as $P(C=h_1|x, w, b)$ because we are assuming the value w, b so the probability depends on them.

Sigmoid Function

The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is the mathematical function that transforms a linear function into a probability, it has several important properties



- it maps any real number to (0,1) [probability]
- Simmetry: $\sigma(-x) = 1 - \sigma(x)$ the function is simmetric around the y ax is
- $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

The expression $p(\text{class}|x,w,b) = \sigma(w^T x + b)$ provides a model to direct calculate the probability that a direct observation x belongs to class he; assuming that the decision boundaries is a linear hyperplane in the feature space.
the decision rule are linear surface orthogonal to w (⊥) with b as the offset from the origin.
So if we Nnew (w,b) then we could compute the predictive distribution for the class labels $p((=he|x,w,b)$.

We have seen an indirect way of computing (w, b) with a generative model

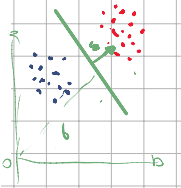
- The tied covariance Gaussian feature model, combined with application class prior, can be cast as a linear decision rule of the form $w^T x + b \lesssim 0$

However, in the following we are interested to find an alternative way to estimate an effective classification rule that does not require an explicit model of the feature vectors distribution

We thus ignore generative models for X and concentrate directly on the form of the class posterior probabilities

Again, we follow a frequentist approach, i.e. compute an estimate for w and b from a set of training samples

(using again liklihood



b can increase or decrease depending on the cost

We start with a labeled dataset:

$D = \{(x_1,c_1), \dots (x_n,c_n)\}$

Where:

- Each x_i is a feature vector (sample)
- Each c_i is a class label

We make a critical assumption: the data point are indipendent and identically distributed (iid) given model parameters. This means:

$[(x_i,c_i) \perp\!\!\!\perp (x_j,c_j)] \mid \theta \quad p(x_i,c_i|\theta) \cdot p(x_j,c_j|\theta) \ \forall i,j$

So Nnowing the outcome of one data point doesn't tell us anything about the outcome of another data point, once we Know the model parameters $\theta = (w,b)$

So the likelihood function represent the probability of observing our entire dataset given the model parameters

$$\mathcal{L}(\theta) = f_{X_1 \dots X_n, C_1 \dots C_n}(x_1, \dots x_n, c_1 \dots c_n \mid \theta) \qquad \text{because of the i.i.d assumption} \Rightarrow$$

$$= \hat{\prod_{i=1}^{n}} f_{X_i, C_i}(x_i, c_i \mid \theta)$$

The main difference from the generative models is in the way we decompose the joint density $f_{x,c|\theta}$

⇓

Generative Decomposition: $P(x,c) = P(x|c)\,P(c)$

Discriminative Decomposition: $P(x,c) = P(c|x)\,P(x)$

So in our likelihood function, each data point contributes:

$$f_{x,c|\theta}(x_i, c_i \mid \theta) = P(C = c_i \mid X = x_i, \theta)\, f(x_i)$$

An Analogy to Understand the Difference
Imagine you're a doctor trying to diagnose a disease:

Generative approach: You study healthy people and people with the disease separately. You learn the typical patterns for each group, understanding what's "normal" and what indicates disease. Then when a new patient arrives, you compare their symptoms to both patterns and see which they match better.

Discriminative approach: You directly study the boundary between healthy and sick patients, focusing only on which symptoms best distinguish between the two groups. You don't need to fully understand what "typical" looks like for each

Where:

- $P(C = c \mid X = x_i, \theta)$ is the conditional probability on the class given the features
- $f(x_i)$ is the marginal density

Note: We will ignore $f(x_i)$ it not depend on θ

For semplicity we consider the logarithm of our likelihood function:

$$\log \mathcal{L}(\theta) = \log \hat{\prod_{i=1}^{n}} f_{x,c|\theta}(x_i, c_i \mid \theta) = \sum_{i=1}^{n} \log P(C = c_i \mid X = x_i, \theta) + \sum_{i=1}^{n} \log f(x_i)$$

Since θ only influence the first sum and we're trying to maximize this function, we can focus on:

$$\ell(\theta) = \sum_{i}^{n} \log P(C = c_i \mid X = x_i, \theta)$$

$$\theta_{ML} = \arg\max_{\theta} \log \mathcal{L}(\theta)$$

The parameter that best fit the distribution

We now need to express the probability of each class label in terms of model parameters (w,b)

[Considering binary problem]

- For positive class $(c=1)$: $y_i = P(c_i = 1 \mid x_i, w, b) = \sigma(w^T x_i + b)$
- For negative class $(c=0)$: $1 - y_i = 1 - \sigma(w^T x_i + b) = P(c_i = 0 \mid x_i, w, b) = \sigma(-w^T x_i - b)$

[σ sigmoid property function]

A key insight is that our classification binary problem follow the Bernoulli distribution pattern. The Bernulli distribution describe a random variable that can take only two values (0 or 1) with some probability p.

$$C|x, w, b \sim Ber(\sigma(w^T x + b)) = Ber(y_i)$$

We are saying that given a specific feature vector x and our model parameters w and b

1. The class label C follows a Bernulli distribution
2. The parameter of the Bernulli distribution is $\sigma(w^T x + b)$

So

$$\begin{cases} P(C=1/x, w, b) = y \\ P(C=0/x, w, b) = 1-y \end{cases}$$

The compact formula

$$P(C=c|x, w, b) = y^c (1-y)^{1-c}$$

Now our log-likelihood function over all data points becomes:

$$\ell(w,b) = \log \prod_{i=1}^{n} P(C_i = c_i | x_i, w, b) = \log \prod_{i=1}^{n} y_i^{c_i} (1-y_i)^{1-c_i} =$$

$$= \sum_{i=1}^{n} [c_i \log y_i + (1-c_i) \log (1-y_i)]$$

Where $y_i = \sigma(w^T x_i + b)$

We've stated that our goal in logistic regression is to find the parameters w^* and b^* that maximize the log-likelihood function:

$$\ell(w,b) = \sum_{i=1}^{n} [c_i \log y_i + (1-c_i) \log (1-y_i)]$$

Since optimization algorithm typically minimize rather than maximize we can convert the minimization problem by simply negating the function:

$$J(w,b) = -\ell(w,b) = \sum_{i=1}^{n} -[c_i \log y_i + (1-c_i) \log (1-y_i)]$$

Now the expression (looking at a single data point):

$$H(c_i, y_i) = -[c_i \log y_i + (1-c_i) \log (1-y_i)]$$ is called the binary cross-entropy

Per TRUE LABEL — X FALSE LABEL (è una somma di 2 cross entropy diverse)

between the true label c_i and the predicted probability y_i

What is cross-Entropy?

The cross-entropy measure the "distance" between two probability distributions. Given two distributions P and Q over the same set of outcomes the cross-entropy is defined as:

$$H(P, Q) = -\underset{P(x)}{E}[\log Q(x)] = -\sum_{x \in S} P(x) \log Q(x)$$

Where:

P is the "true" distribution

Q is our approximation or prediction

The sum is over all possible outcomes x

$E_{P(x)}$ represent the expected value with respect to distribution P (NASPE NO ASW)

In our case for each sample in our dataset we have:

- The actual outcome (0 or 1)
- Our model guess about how likely they were to pass.

Our goal is to make our guesses match reality as close as possible, this is posible by minimizing the cross-entropy.

- P is the "true" distribution: the empirical of class labels, from the point of view of an observer ξ (evaluator) who knows the true (the actual labels)

Given the data point (x_i, c_i) the true distribution puts all the probability mass on the observed class:

$$P(C_i = 1 \mid X_i = x_i, \xi) = \begin{cases} 1 & \text{if } c_i = 1 \\ 0 & \text{if } c_i = 0 \end{cases}$$

$$P(C_i = 0 \mid X_i = x_i, \xi) = \begin{cases} 0 & \text{if } c_i = 1 \\ 1 & \text{if } c_i = 0 \end{cases}$$

$$\Rightarrow P(C_i = 1 \mid X_i = x_i, \xi) = c_i \quad , \quad P(C_i = 0 \mid X = x_i, \xi) = 1 - c_i$$

↓

Bernoulli distribution with parameter c_i

- Q is our model predicted distribution: this is the distribution given by our model parameters wich we call R (Recognizer)

$$Q(C_i = 1 \mid X = x_i, R(w, b)) = y_i = \sigma(w^T x_i + b)$$

$$Q(C_i = 0 \mid X = x_i, R(w, b)) = 1 - y_i = 1 - \sigma(w^T x_i + b)$$

Logistic regression looks for the minimizer of the average cross-entropy between the distribution for the training set labels of an evaluator $\mathcal{E}$ who knows the real label and the distribution for the training set labels as predicted by the model $\mathcal{R}(w, b)$ itself

The cross-entropy is a measure of goodness of the predictions, and the evaluation is performed over the training data itself

[Cross-entropy properties]

① it penalize wrong predictions heavily: if the true label is 1 and our model gives a small probability to that class ($y = 0.1$) the cross-entropy is very large ($-\log(0.1) = 2.3$)

② it rewards correct predictions: with $y = 0.9$ the cross-entropy is $(-\log(0.9) = 0.1)$.

We are finding the parameters that minimize the coding inefficiency when using our model to predict the true label.

So when our predicted distribution Q exactly matches the true distribution P, ($P=Q$) the cross-entropy is minimum

When we minimize cross-entropy, we're trying to make our model's predictions match the empirical distributions of our training dataset as closely as possible, given the constraints of our model

In the case of logistic regression, these constraints are significant - we're limited to linear decision boundaries. The model can't perfectly match any arbitrary data distribution, but it finds the best linear approximation possible

Now, instead of using labels 0 and 1, we can use -1 and 1

The terms z_i still represent class labels, however for samples of class h_1 we have $z_i = 1$, whereas for samples of class h_0 we have $z_i = -1$:

$$z_i = \begin{cases} 1 & \text{if } c_i = 1 \\ -1 & \text{if } c_i = 0 \end{cases}$$

We can rewrite the cross entropy following this conversion

$z_i = 2c_i - 1$

Now let's rewrite our objective function.

$H(c_i, y_i) = -[c_i \log y_i + (1 - c_i) \log(1 - y_i)]$

Where:

c_i is the class label (0 or 1) of the sample

$y_i = \sigma(w^T x + b)$ our predicted probability

$s_i = w^T x + b$ is our linear score (logit)

Following the formula $z_i = 2c_i - 1 \Rightarrow c_i = 1, z_i = 1$

$c_i = 0, z_i = 0$

$$H(c_i, y_i) = \begin{cases} 1 \log(y_i) + (1-1)\log(1-y_i) = -\log y_i = -\log \sigma(s_i) & c_i = 1,\ z_i = 1 \\ 1 \log(1-y_i) = \log(1-\sigma(s_i)) = -\log \sigma(-s_i) & c_i = 0,\ z_i = -1 \end{cases} \Rightarrow$$

x property

Using $z_i = -1$ we can combine this in $\underline{H(c_i, y_i) = -\log \sigma(z_i s_i)}$ =

Now because $\sigma(x) = \frac{1}{1+e^{(-x)}}$

$$\Rightarrow H(c_i, y_i) = \log\left(1 + e^{-z_i s_i}\right) \overset{s_i = w^T x_i + b}{=} \log\left(1 + e^{-z_i (w^T x_i + b)}\right)$$

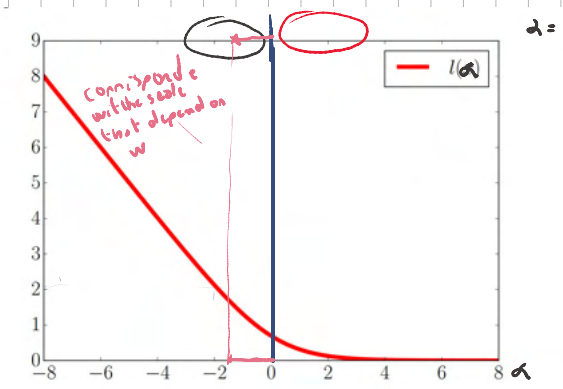
Logistic loss functions

The expression $\log\left(1 + e^{z_i (w^T x_i + b)}\right)$ is called the logistic loss function often denoted as $l(z_i s_i)$ or $l(z_i (w^T x_i + b))$

Our entire objective function then becomes:

$$S(w, b) = \sum_{i=1}^{n} l(z_i (w^T x_i + b)) = \sum_{i=1}^{n} \log\left(1 + e^{-z_i (w^T x_i + b)}\right)$$

Our goal is to find the minimizer of $S(w, b)$



Note:

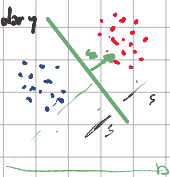
In logistic regression, the linear score $s_i = w^T x_i + b$ has a powerfull geometric interpretation:

① Decision boundary: The equation $w^T x + b = 0$ define an hyperplane in the feature space.

② Orientation: The vector w is orthogonal (perpendicular to this hyperplane)

③ Signed Distance: The value of s_i is proportional to the signed distance from point x_i to the decision boundary.

When we say "signed distance", we mean:
- Positive values ($s_i > 0$) indicate the point is on the positive side (class h_1) [answers the "on which side?"]
- Negative values ($s_i < 0$) indicate the point is on the negative side (class h_0)
- The magnitude $|s_i|$ relates to how far the point is from the boundary



if you remember the score s_i has a direct probabilistic interpretation. it represent the log-ratio of the posterior probabilities:

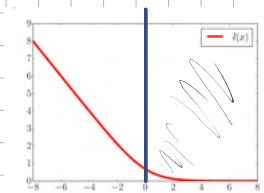
$$\log\left(\frac{P(C_i=1|x_i=x_i)}{P(C_i=0|x_i=x_i)}\right) = w^T x_i + b = s_i$$

When $s_i = 0$ the two class are equally likely
$s_i < 0$ Class h_0 more likely
$s_i > 0$ Class h_1 more likely
The magnitude $|s_i|$ indicate how confident the model is in the predictions

Going back to the logistic loss function $l(z_i s_i)$, this function measures the cost of our predictions for each samples

① When predictions and actual Class Agree $z_i s_i > 0$
This happens in two cases:
- When $z_i = 1$ (actual class is h_1) and $s_i > 0$ (predicted class h_1)
- $z_i = -1$ (actual class is h_0) and $s_i < 0$ (predicted class h_0)

In this case the cost is relatively low. As $|s_i|$ increase (the point moves farther from the boundary to the correct side) the cost decrease exponentially approaching 0 for point far from the boundary

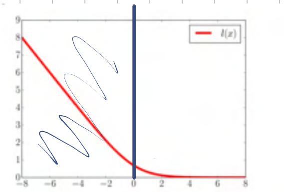


$l(z)$ as value $\to 0$ so $S(w,b) \to 0$ cross-entropy $\to 0$ we are predicting well

② When predictions and Actual Class Disagree: $z_i s_i < 0$
This happen in two cases:
- When $z_i = 1$ (actual class is h_1) and $s_i < 0$ (predicted class is h_0)

- When $z_i = -1$ (actual class is h_0) and $s_i > 0$ (predicted class is h_1)

In this case we pay a much higher cost, As $|s_i|$ increase (the point moves farther from the boundary to the wrong side) we pay a cost that increase (asymptotically) linearly with s_i



l(x) loss value ↗ so S(w,b) ↗ cross-entropy ↗ we are predicting bad

Connection with empirical risk

Logistic regression fits within a broad statistical learning framework (avremo un apprendimento statistico più ampio) called Empirical Risk Minimization (ERM). In this framework:

1. We define a loss function $l(w, x, z_i)$ that measures the cost of our prediction for each sample
2. The empirical risk is the average loss over our traning data: $R(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(w, x_i, z_i)$ (DATA AGGIUNTA $\frac{1}{n}$)
3. We find the model parameters θ that minimize this empirical risk

For logistic regression, the loss function is the logistic loss: $l(w, x_i, z_i) = \log(1 + e^{-z_i(w^T x_i + b)})$

This framework connects logistic regression to other classification algorithms:

- SVM use the hinge loss: $l(w, x_i, z_i) = \max(0, 1 - z_i(w^T x_i + b))$
- LINEAR REGRESSION can be viewed as using squared loss: $l(w, x_i, y_i) = (y_i - w^T x_i - b)^2$

Why "Empirical" Risk?
The term "empirical" emphasizes that we're calculating the risk based only on our observed training samples, not the true underlying data distribution. In statistical learning theory:

The true risk would be the expected loss over the entire data distribution
The empirical risk is an approximation based on our finite training set

Minimizing the empirical risk doesn't guarantee minimizing the true risk (which would give the best generalization), but with enough data and appropriate regularization, the two often align well.

The infinite Solution Problem

Linearly separable data means we can draw a straight line (or hyperplane in higher dimension) that perfectly separates the two classes. Mathematically, there exists some combinations of weights w and bias b such that:

- For all positive examples ($z_i = 1$): $w^T x_i + b > 0$
- For all negative examples ($z_i = -1$): $w^T x_i + b < 0$

This means all traning points lie on the correct side of the decision boundary, making $z_i s_i > 0$ for all training points.

Here is the key insight: if we find such a separating hypeplane defined by w and b, we can create an infinite number of equivalent solutions by simply scalling w and b.
For example, if (w, b) gives perfect separation then $(2w, 2b)$ also give perfect separation

with even greater margins.

As we increase $\|w\|$ (the norm of w), the value of $s_i = w^T x_i + b$ grow larger for all points, pushing them further from the decision boundary. Looking at our logistic loss function $l(x_i, s_i) = \log(1 + e^{-y_i s_i})$ we can see that:

- As $y_i s_i$ becomes increasingly positive, the loss approaches zero
- The total loss function $J(w,b)$ approaches zero as $\|w\| \to \infty$.

This is a problem!! our optimization has no minimum, only a greatest lower bound (infimum) that goes to zero when $w \to \infty$

So the algorithm would keep increasing $\|w\|$ indefinitely, leading to numerical issues.

The Regularization Solution

To solve this problem, we add a regularization term to our objective function that penalize large values of $\|w\|$. The most common form is $L2$ regularization (also called Ridge regularization):

The objective function that we want to minimize becomes:

$$\hat{R}(w,b) = \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \log\left(1 + e^{-y_i(w^T x_i + b)}\right)$$

Or in the averaged form:

$$R(w,b) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + e^{-y_i(w^T x_i + b)}\right)$$

Where:

- λ is the regularization parameter (hyperparameter)
- $\|w\|^2$ is the squared $L2$ norm of w (sum of squared weights)
- The second term is the logistic loss function

Regularization create a trade-off between two competing objectives:

- Minimize the classification error (logistic loss term) ①
- Keep the model simple (regularization term) ②

As $\|w\|$ grow, the first term decreases (better classification), but the second term increases (more complex model). The optimum occurs at a finite value of $\|w\|$, ensuring a well defined solution.

The regularization parameter control the balance between this two objectives:

- Large λ: Strongly favor simpler models (smaller $\|w\|$) potentially at the cost of classification accuracy
- Small λ: Focuses more on minimizing classification error, potentially leading to overfitting
- $\lambda = 0$ Revert to the origin, with the infinite problem.

Importantly, we cannot determinate the optimal λ by minimizing the same objective function, that would give always $\lambda = 0$, defeating the purpose of the validation. Instead we tipically use cross-validation to find the value of λ that gives the best performance on held-out validation set.

Understanding Contour Lines in the Logistic Regression Visualization

Those lines in the plots are contour lines, which are a powerful way to visualize the behavior of the logistic regression model. Let me explain exactly what they represent and why they're so revealing about regularization's effects.

What Are Contour Lines?

In these plots, the contour lines represent locations where the function $w^T x + b$ has the same value. Each line corresponds to a specific value of this function, which you can see labeled on the lines (+15, +10, +5, etc. in the left plot; +3, +2, +1, etc. in the right plot).

Think of these like elevation lines on a topographical map. Just as elevation lines on a hiking map show points of equal height, these contour lines show points where the model's "score" is equal.

The Physical Landscape Analogy

Imagine the decision function $w^T x + b$ creates a 3D landscape above the 2D feature space:

The decision boundary (purple line) is where this landscape has height zero
Areas above zero (positive values) belong to one class
Areas below zero (negative values) belong to the other class

The contour lines are like the elevation lines on this landscape:

Red contour lines show positive values (hills)
Blue contour lines show negative values (valleys)
The purple line is the zero contour (sea level)

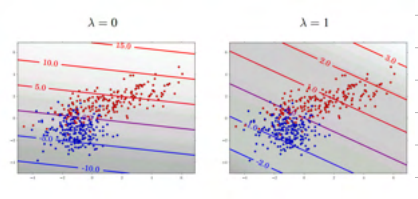
Reading the Spacing Between Lines

The spacing between contour lines tells us about the steepness of this landscape:

Tightly packed lines (left plot, $\lambda = 0$) indicate a steep slope
Widely spaced lines (right plot, $\lambda = 1$) indicate a gentle slope

This spacing directly relates to w:

A large w creates a steep landscape with tightly packed contour lines
A small w creates a gentle landscape with widely spaced contour lines



Why the Values Are Different Between Plots

Notice how the values on the contour lines differ dramatically between the two plots:

Without regularization ($\lambda = 0$):

The contour lines include extreme values: +15, +10, +5, -5, -10
This indicates large coefficient values (large w)
The model is making very confident predictions far from the boundary

With regularization ($\lambda = 1$):

The contour values are more moderate: +3, +2, +1, -1, -2, -3
This indicates smaller coefficient values (smaller w)
The model makes more conservative predictions

The Decision-Making Process

These contour lines also reveal how the model makes decisions:

For a new data point, the model calculates $w^T x + b$
If this value is positive (red region), it predicts one class
If this value is negative (blue region), it predicts the other class
The magnitude tells us about the model's confidence

In the unregularized model, a point just slightly away from the boundary might already have a value of +5 or -5, indicating high confidence. In the regularized model, you'd need to go further from the boundary to reach the same confidence level.

Practical Significance

This visualization demonstrates why regularization improves generalization:

The unregularized model ($\lambda = 0$) is overconfident. It rapidly switches from strongly predicting one class to strongly predicting the other with only small movements in feature space. This makes it sensitive to small perturbations in the data. overfitting problem
The regularized model ($\lambda = 1$) is more cautious. It transitions more gradually between classes, requiring larger movements in feature space to achieve high confidence. This creates a more robust model that better handles variations in data.

The gray shading in the right plot likely represents a region of uncertainty where the model isn't extremely confident in either direction, which is a hallmark of a well-regularized model that knows when to be unsure.

Understanding these contour lines gives you deep insight into how regularization shapes the decision landscape and why this leads to models that generalize better to new data.

Some Consideration

The Regularization Mechanism
When we add L2 regularization to logistic regression, we're adding a penalty term λw^2 to our objective function. This penalizes large weights, encouraging the model to keep all weights small.
But here's the crucial point: this penalty treats all weights equally, regardless of which feature they correspond to. The regularization doesn't know or care about the meaning or scale of your features—it simply pushes all weights toward zero with the same force.
The Problem With Different Feature Scales
Imagine we're predicting house prices using two features:

Feature 1: House size (in square feet), ranging from 1,000 to 5,000
Feature 2: Number of bedrooms, ranging from 1 to 6

Without regularization, the model might learn weights like:

$w_1 = 0.01$ (for house size)
$w_2 = 10{,}000$ (for bedrooms)

These weights could give similar importance to both features:

A 1,000 sq ft increase contributes $0.01 \times 1{,}000 = 10$ to the prediction
An extra bedroom contributes $10{,}000 \times 1 = 10{,}000$ to the prediction

So is usefull to preprocessing data to improve the performance of logistic Regression.

① When we center the data, we subtract the mean from each feature:

$x_i' = x_i - \mu$

This shifts all feature distributions to have a mean of zero. The mean can be calculated in different ways:

- Using the entire training set (most common)
- Using a weighted mean, such as the average of class means (usfull when class are imblanced)

Centring help remove arbitrary offsets in the data. For example, if one feature is temperature in Celsius (perhaps ranging from 20°C to 30°C), centering would transform it to range from roughly -5 to +5, making 0 a meaningful reference point.

② Standardizing Variances

Standardizing goes beyond centering by also scaling each feature to have unit variance:

$$x'_{i,j} = \frac{x_{i,j}}{\sigma_j}$$

Where:

$x_{i,j}$ is the j-th feature of the i-th sample, and σ_j is the standard deviation of the j-th feature across the training set.

This ensure all features contribute equally the distance calculations and regularization penalties.

For istance, if we have income (in thousands, perhaps ranging from 30-200) and age (ranging from 20-60), standardization would put both on comparable scales.

① and ② can be used together = zero normalization

③ Whitening the Covariance matrix (sphering)

Whitening standardize variances and decorrelate features

$x_i' = Ax_i$ where $A = \Sigma^{-\frac{1}{2}}$ the x_i' data will have the I as Covariance matrix

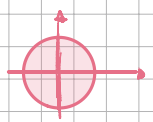
Σ is the covariance matrix of the training data, and $\Sigma^{-\frac{1}{2}}$ is the matrix square root inverse.

This transformatio creates feature that are:

- Uncorrelated with each other (orthogonal)
- All have unit variance

This technique can be particullary helpful when features have strong correlations, wich is common in image data, audio signals, or financial time series. A variation uses the within-class covariance instead of the total covariance, which can better preserve class-discriminative information.

④ L2 Normalization



L2 Normalization scales each samples to have unit length:

$x_i' = x_i / \|x_i\|$

This makes all samples lie on the unit sphere. it's always performed after centering and whitening

L2 normalization is especially useful when the overall magnitude of each features doesn't matter, but rather (piuttosto) the relative proportions between features.

Will we choise one metod or another using cross-validations.

Logistic regression optimizes the probability of observing the training lables. As a result it naturally incorporates the empirical class distribution (prior) from the training data.

This mean if your training set has 90% negative samples and 10% positive samples the model implicitly learn this 9:1 ratio and adjust its predictions accordingly.

However this became problematic when the real world application has a different class distributions than the training data. For examples my model can be deployed in a setting the true probability of positive class is 50% not 10%.

A solution to this can be adjusting the model score to behave like a log-likelihood ratio by subtracting the empirical prior log-odds (of the training dataset)

$$s_{llr} = w^T x + b - \log \frac{n_T}{n_F} = w^T x + b - \log \frac{\pi_{emp}^{tr}}{1 - \pi_{emp}^{tr}}$$

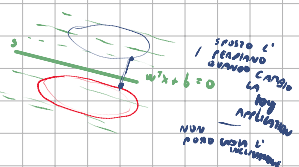
Where:

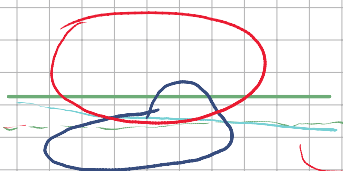
- n_t is the number of positive samples in training
- n_e " " negative samples "
- $\log(n_t/n_e)$ is the log-odds of the empirical prior in the training set.

When we want to use this model for a different application with prior π_T we can make decision by comparing:

$$s_{\pi_T} = -\log\left(\frac{\pi_T}{1-\pi_T}\right)$$

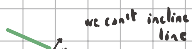
Anyway this is not perfect in a nutshell we are only moving the decision boundary





FOR GAUSSIAN NON È UN PROBLEMA PERCHÉ SONO LINEARI ALLA LINEARITÀ

$$w^T x + b - \log \frac{\pi_{emp}^{tr}}{1 - \pi_{emp}^{tr}} - \log \frac{\pi_T}{1 - \pi_T} \leq 0$$



Prior-Weighted logistic Regression

An even better approach is to directly incorporate the target application's class distribution during training. This is called prior-weighted logistic regression

$$R(w) = \frac{\lambda}{2}\|w\|^2 + \frac{\pi_T}{n_T}\sum_{i|y_i=1} l(z_i, s_i) + \frac{1-\pi_T}{n_F}\sum_{i|y_i=-1} l(z_i, s_i)$$

this effectively reweights the loss distribution from positive and negative examples to match the target application's class distribution. Positive samples are weighted by π_T and negative examples by $(1-\pi_T)$

Even with this approach we can still compute an adjusted log-likelihood ratio score for use with other application if needed

Note: π_T can be estimated

We test the model on MNIST digit pairs (e.g. 0 vs 1, 0 vs 2, ...)

MNIST — Average Pairwise EER for Logistic Regression

DimRed	$\lambda = 0$	$\lambda = 0.00001$	$\lambda = 0.001$	$\lambda = 0.1$	Tied Gau
RAW [768]	1.7%	1.4%	1.2%	2.0%	—
PCA [50]	1.4%	1.4%	1.4%	2.1%	1.7%
PCA [100]	1.3%	1.2%	1.2%	2.0%	1.5%

LogReg obtains better performance than the Gaussian model

Regularization is important, especially when we do not reduce the dimensionality (we have more parameters to estimate, so over-fitting is more sever)

If we regularize too much the model performs poorly again

Multiclass Logistic Regression

[table inspiration]

We begin with insights from the linear Gaussian classifier with uniform priors. For this model, the log posterior probability for any class j takes the form:

$$\log P(C=j|x) = w_j^T x + b_j + h(x) \quad \left[\text{for extended formula see gaussian model}\right]$$

Where

- w_j is the weight vector for class j
- b_j is the bias term for class j
- $h(x)$ is a term that depends only on the input x, not on the class

$h(x)$ is the same for all classes, meaning it cancels out when comparing classes. This gives us the proportionality relationship.

$P(C=j|x) \propto e^{w_j^T x + b_j}$ $e^{h(x)}$

Since we are working with a closed-set classification problem (the sample must belong to one of our K classes) the probabilities must sum to 1:

$$\sum_{j=1}^{K} P(C=j|x) = 1$$

The normalized factor for $P(C=j|x)$ is thus: (across classes) $\sum_{j=1}^{K} e^{w_j^T x + b_j} e^{b(x)} =$

$=$ the posterior probability corresponds to:

$$P(C=k|x) = \frac{e^{w_k^T x + b_k} e^{b(x)}}{\sum_j e^{w_j^T x + b_j} e^{b(x)}} = \frac{e^{w_k^T x + b_k}}{\sum_j e^{w_j^T x + b_j}}$$

Note: This ensure that the sum is $1 \Leftarrow \frac{e^{z_1}+e^{z_2}}{\Sigma e^{z_j}} = \frac{\Sigma e^{z_j}}{\Sigma e^{z_j}}$

The function that perfor this normalization trasforming (last step) raw scores into proper probabilities is called softmax

$$f(s) = \left[\frac{e^{s_k}}{\sum_j e^{s_j}}, \dots, \frac{e^{s_n}}{\sum_j e^{s_j}} \right]$$

The softmax is a natural extantsion of the sigmoid function we used in binary logistic regression. In fact for $k=2$ softmax reduces to sigmoid function when we constrain $w_0 = w_1$, $b_2 = -b_1$ (we have two classes)

Given the model parameters

$$W = [w_1 \ \dots \ w_K], \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_K \end{bmatrix}$$

the logistic regression model allows computing the probability of each class

$$P(C = k|W, b, x) = \frac{e^{w_k^T x + b_k}}{\sum_j e^{w_j^T x + b_j}}$$

If we consider sample x_i, its class posterior distribution is thus a categorical distribution

$\forall i$ of our mini-batch $C_i|W, b, X_i = x_i \sim \mathrm{Cat}(y_i)$ $x_i \to \begin{bmatrix} y_{i1} \\ \vdots \\ y_{iK} \end{bmatrix}$ where

$$y_{ik} = \frac{e^{w_k^T x_i + b_k}}{\sum_j e^{w_j^T x_i + b_j}}$$

41/56

Note:

A categorical distribution is a natural extantion of the Bernoulli distribution to multiple outcome. Instead of just two possible outcome, we now have K possibilities, and we need a specific probabilities for each one.

A categorical distribution is specified by a vector of K probabilities:

$y = [y_1, y_2, \dots y_K]$

Where:

- each y_n represent the probability of outcomes n
- All probabilities are non-negative: $y_n \geq 0$ for all n
- The $\sum_n y_n = 1$

When we say C follow the categorical distribution with parameters y, written as $C \sim Cat(y)$ we mean $P(C=n) = y_n$ for $n = 1, 2, \ldots k$

In logistic regression we compute this probability using the softmax function applied to our linear scores:

$$y_{in} = \frac{e^{w_n^T x_i + b_n}}{\sum_j e^{w_j^T x_i + b_j}}$$

This gives us a vector of probabilities $y_i = [y_{i1}, y_{i2}, \ldots y_{ik}]$ for each sample x_i
When we write:

$$c_i | W, b, X_i = x_i \sim Cat(y_i)$$

We're saying: "Given the input features x_i and our model parameters W and b, the class label c_i follows a categorical distribution with probabilities given by the vector y_i

As for the binary case we can express the log-probability of the training class labels as:

$$\ell(W, b) = \sum_{i=1}^{n} \log P(C_i = c_i | X_i = x_i, W, b)$$

To better represent this we can use the 1-of-K encoding
1 of K encoding is used to to represent the true class.
For example, with a sample with true class c_i (which is a single number between 1 and k) we create a binary vector z_i where:

- $z_{in} = 1$ if $c_i = n$ (n is the true class)
- $z_{in} = 0$ otherwise (for all other class)

$$z_i = [0 \ldots 0, 1, 0 \ldots 0], \quad z_{ik} = \begin{cases} 1 & \text{if } c_i = k \\ 0 & \text{otherwise} \end{cases}$$

So the log-probability become: (for 1 sample)

$$\log P(C_i = c_i | X_i = x_i, W, b) = \log P(C_i = c_i | y_i) = \sum_{n=1}^{k} z_{in} \log y_{in}$$

This sum picks out exactly the log-probability of the true class

The log-likelihood of the entire training set becomes:

$$\ell(W, b) = \sum_{i=1}^{n} \sum_{n=1}^{k} z_{in} \log y_{in}$$

Where

- n is the number of the training samples
- k is the number of classes
- z_{ik} indicates whether samples i belongs to class k
- y_{ik} is the model predicted probability that sample i belong to class k

As for the binary case, the expression

$$H(z_i, y_i) = -\sum_{k=1}^{K} z_{ik} \log y_{ik}$$

represents the (multiclass) cross-entropy between the observed and predicted label distributions for sample x_i

As for the binary case, we estimate W and b as to maximize the likelihood for the training labels

The ML solution is again the solution that minimizes the (average) cross-entropy:

$$\arg\max_{W,b} \ell(W, b) = \arg\max_{W,b} \left[-\sum_{i=1}^{n} H(z_i, y_i) \right] = \arg\min_{W,b} \sum_{i=1}^{n} H(z_i, y_i)$$

$H(z_i, y_i)$ measure the dissimilarity between:

- The true distributions z_i (all the mass to the true class)
- The predicted distributions y_i (softmax probabilities from our model)

When we model multiple classes, an interestin mathematical property emerges: our model has more parameters than it actually needs this is called over-parameterization

In multiclass logistic regression, we have a weight vector w_k, and bias term b_k, for each class k giving us k weight vectors and k bias terms. However not all of these parameters are independent

Why? Because only the difference of the class score matters, not their absolute values if we add the same constant vector to all the weight vectors w_k, the softmax probabilities remain exactly the same

$$P(C=k|x) = \frac{e^{w_k^T x + b_k}}{\sum_j e^{w_j^T x + b_j}}$$

if we can add c we are using more parameters

if we add c (constant) to every w_k:

$$P(C=k|x) = \frac{e^{(w_k+c)^T x + b_k}}{\sum_j e^{(w_j+c)^T x + b_j}} = \frac{e^{w_k^T x + b_k} e^{c^T x}}{e^{c^T x} \sum_j e^{w_j^T x + b_j}}$$

For a binary classification problem ($k=2$) multiclass logistic regressions should reduce to standard binary case

if we subtract w_2 from both w_2 and w_1, we recover the binary lr formula.

$$P(c=2|x) = \frac{e^{w_2^T x + b_2}}{e^{w_1^T x + b_1} + e^{w_2^T x + b_2}}$$

We redefine the parameters $w = w_1 - w_2$ $\quad b = b_1 - b_2$

$$P(c=2|x) = \frac{e^{(w+w_2)^T x + b + b_2}}{e^{(w+w_2)^T x + b + b_2} + e^{w_2^T x + b_2}} = \frac{e^{w_2^T x} e^{b_2} e^{w^T x + b}}{e^{w_2^T x} e^{b_2}\left(1 + e^{w^T x + b}\right)} =$$

$$= \frac{e^{w^T x + b}}{1 + e^{w^T x + b}}$$

Finally, as for the binary class, we can cast the problem as a minimization of a loss function

We rewrite the objective in terms of class labels c as

$$\begin{aligned} J(\boldsymbol{W}, \boldsymbol{b}) &= -\sum_{i=1}^{n}\sum_{k=1}^{K} z_{ik} \log y_{ik} \\ &= -\sum_{i=1}^{n} \log \frac{e^{\boldsymbol{w}_{c_i}^T \boldsymbol{x}_i + b_{c_i}}}{\sum_{c'=1}^{K} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x} + b_{c'}}} \\ &= \sum_{i=1}^{n}\left[\log\left(\sum_{c'=1}^{K} e^{\boldsymbol{w}_{c'}^T \boldsymbol{x} + b_{c'}}\right) - \boldsymbol{w}_{c_i}^T \boldsymbol{x}_i - b_{c_i}\right] \\ &= \sum_{i=1}^{n} l(\boldsymbol{x}_i, c_i, \boldsymbol{W}, \boldsymbol{b}) \end{aligned}$$

l is also called softmax loss

46/56

Where c_i is the true label of sample i, and c_i subscript in w_{c_i} and b_{c_i}, refers to the weights and the biss of the true class.

The interpretation of the softmax loss is:

① The first term $\log\left(\sum_{c'=1}^{K} e^{w_{c'}^T x + b_{c'}}\right)$ is the soft maximum of all the class scores, representing the normalized factor

② The second term $-(w_{c_i}^T x_i + b_{c_i})$ is the negative of the score for the true label

③ Together they encourage the model to increase the score of the correct class while decreasing the scores of incorrect classes

This loss function directly connects to the cross-entropy. it's the score entropy between a one-hot distribution (the true data) and the softmax probabilities (the model predictions)

As for the binary classifications we add regularization to avoid overfitting

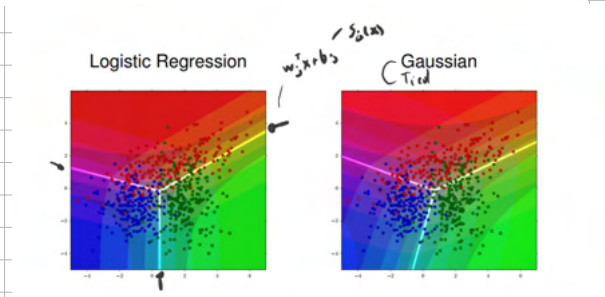
$$R(W,b) = \Omega(W) + \frac{1}{n} J(W,b)$$

A common regularization term is:

$$\Omega(w_1,\dots,w_n) = \frac{1}{2} \sum_i \|w_i\|^2$$

This L2 regularization penalizes large weights, encouraging simpler decision boundaries.

Again, we can replace the average cross-entropy with prior-weighted average cross entropy to account for priors that are different from the empirical training set prior



① Logistic regression create linear decision boundaries without making assumptions about feature distributions

② Gaussian models create quadratic boundaries (for equal covariance) base on distributional assumption

MNIST — Error rates for Logistic Regression

DimRed	$\lambda = 0$	$\lambda = 0.00001$	$\lambda = 0.001$	$\lambda = 0.1$	Tied Gau
RAW [768]	8.0%	7.4%	7.9%	12.9%	—
PCA [50]	8.8%	8.8%	8.9%	13.3%	12.6 %
PCA [100]	7.8%	7.8%	8.2%	12.9%	12.3%
PCA+LDA [9]	10.9%	10.9%	11.0%	12.4%	12.3 %

The multiclass logistic regression performs better than the Gaussian model — indeed, the Gaussian assumption is not very accurate for the features we are considering. LogReg assumes linear separation, but does not assume a specific distribution for the features

Again, regularization is important, especially when the feature space is large

On the MINIS dataset, linear logistic regression outperforms the tied covariance Gaussian Classifier but fall short compared to the non-tied covariance Gaussian Classifier (MVG)

Why does this happen? The answer lies in the decision boundaries
- Linear Logistic regression can only create linear separation boundaries (straight lines in 2D, flat planes in 3D)
- Tied covariance Gaussian models also create linear boundaries
- MVG create quadratic boundaries (curved in 2D)

For complex data not linear boundary can be better.
We can transform our features to enable logistic regression to create non-linear decision boundaries, this is how.

For MVG a non linear Gaussian classifier:

$$\log \frac{P(C=h|x)}{P(C=h|x)} = x^T A x + b^T x + c = s(x, A, b, c) \quad \text{[Quadratic form]}$$

- $x^T A x$ is a quadratic term (like x_0^2 or $x_0 x_1$)
- $b^T x$ is a linear term
- c is a constant

This create curved decision boundaries that can separate more complex patterns.

We use a trick, while the expression is quadratic in the features x, it actually linear in the parameter A and b.

Take a 2D example, the quadratic form $x^T A x$ expand to:

$$\begin{bmatrix} x_0 & x_1 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} \\ a_{11} & a_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = a_{00} x_0^2 + a_{01} x_0 x_1 + a_{10} x_1 x_0 + a_{11} x_1^2 = a_{00} x_0^2 + a_{11} x_1^2 + (a_{01} + a_{10}) x_0 x_1 + a_{00} x_0^2$$

the coefficient $a_0, a_1, \ldots$ are linear $[f(ax+b) = f(a) + f(a_1) + \ldots$
So the function is linear in the parameters but "quadratic in the features"
We can so rewrite the quadratic terms using a matrix inner product:
$s(x, A, b, c) = \langle xx^T, A \rangle + b^T x + c$

The inner product multiplies corrispondin element of the matrices xx^T and A, and sums them all
$\langle A, B \rangle = \sum_i \sum_j A_{ij} B_{ij}$

We can further express $\langle A, xx^T \rangle$ as

$$\langle A, xx^T \rangle = \text{vec}(xx^T)^T \text{vec}(A) \quad \text{[dot product]}$$

vec(n) stack the columns of A into a single column vector

Finally we can combine all our term by defining:

$$\tilde{\phi}(x) = \begin{bmatrix} \text{vec}(xx^T) \\ x \end{bmatrix}$$

$$w = \begin{bmatrix} \text{vec}(A) \\ b \end{bmatrix}$$

the log posterior ratio can be expressed as

$$s(x, w, c) = w^T \tilde{\phi}(x) + c$$

Now instead of feeding our original feature x to logistic regression, we can:
1. Transform our features using the mapping $\tilde{\phi}(x)$ that include all the quadratic product
2. Train the standard logistic regression model on this expanded features.
3. The resulting model will have linear boundaries in the expanded space but quadratic boundaries in the original space

For example, if our original feature were $[x_1, x_2]$ our expanded features would be $[x_1^2, x_1x_2, x_2^2, x_1, x_2]$ capturing all quadratic terms.

The LR model (both binary and multiclass) allows computing linear separation rules for the transformed features $\phi(x)$

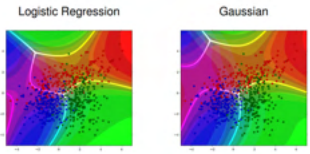
Since expressions $w^T \phi(x) + c$ correspond to quadratic forms in the original feature space, we are actually estimating quadratic separation surfaces in the original space

In general, we can consider a transformation $\phi(x)$ of our feature space such that our classes are (approximately) linearly separable in the expanded feature space

This approach is powerfull but it comes at a computational cost:

For original features of dimension M, a polynomial expansion of degree d result in an expanded feature space of dimension $O(M^d)$ in this case we are using quadratic form so $d=2$ [CAMPO DELLO $\Phi(x)$]

LA DIMENSIONE DEL DATASET RADDOPPIA



MNIST — Average binary pairwise EER for LR with quadratic feature expansion

DimRed	$\lambda = 0$	$\lambda = 1e^{-5}$	$\lambda = 1e^{-3}$	$\lambda = 1e^{-1}$	Gaussian
PCA [50]	1.0%	1.0%	0.9%	1.5%	0.8%

MNIST — Multiclass error rates for LR with quadratic feature expansion

DimRed	$\lambda = 0$	$\lambda = 1e^{-5}$	$\lambda = 1e^{-3}$	$\lambda = 1e^{-1}$	Gaussian
PCA [50]	2.3%	1.9%	1.7%	3.1%	3.6%