CS 1810 Spring 2026 Section 3
Classification

1 Classification

In the previous sections, we predicted continuous targets using KNN, kernelized regression and OLS. We now turn to classification, where the target y is discrete, such as identifying bird species from audio recordings or classifying galaxies by morphology. We begin with linear methods.

1.1 Linear Classification

Given an input vector $\boldsymbol{x}$, linear classification assigns it to one of K discrete classes C_k by dividing the input space into regions separated by linear decision boundaries.

In binary linear classification, two classes (labeled -1 and 1) are separated by a single hyperplane. The discriminant function directly assigns each vector to a class:

$$\hat{y} = \text{sign}(h(\boldsymbol{x}; \boldsymbol{w}, w_0)) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + w_0)$$

where $\text{sign}(z) = 1$ if $z \geq 0$ and $\text{sign}(z) = -1$ if $z < 0$. The weight vector $\boldsymbol{w}$ is normal to the decision surface and determines its orientation.

This gives us a classification rule, but we still need a loss function to learn $\boldsymbol{w}$ and w_0 from training data. One natural candidate is mean squared error, but since $y \in \{-1, 1\}$ while $h(\boldsymbol{x})$ is unbounded, the model is penalized for confident correct predictions.

1.2 Perceptron

The perceptron is one approach to learning $\boldsymbol{w}$: a discriminative algorithm for binary classification that finds a linear decision boundary, if one exists.

To define the loss, we use the rectified linear function $\mathrm{ReLU}(z) = \max\{0, z\}$. With $h(\mathbf{x}_i; \mathbf{w}, w_0) = \mathbf{w}^\top \mathbf{x}_i + w_0$ and label y_i, the perceptron loss is:

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{N} \mathrm{ReLU}(-h(\mathbf{x}_i; \mathbf{w}, w_0) y_i) \\ &= -\sum_{i:\, y_i \neq \hat{y}_i} (\mathbf{w}^\top \mathbf{x}_i + w_0) y_i\end{aligned}$$

For any misclassified point, $-h(\mathbf{x}_i; \mathbf{w}, w_0)y_i$ is positive, so minimizing this loss drives the decision boundary toward correct classification.

We minimize the loss using stochastic gradient descent. Correctly classified points have zero gradient; for a misclassified point $(\mathbf{x}_i, y_i)$, the updates are:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i \\ w_0 &\leftarrow w_0 + \eta y_i\end{aligned}$$

where η is the learning rate.

If the training data are linearly separable, the perceptron is guaranteed to converge in finitely many updates.

1.2.1 Concept Question

Why do we choose the ReLU function instead of the $0/1$ function when formulating the loss function?

Solution: The gradient for the $0/1$ function is uninformative. We are either right or wrong. $0/1$ is not differentiable at 0 and its derivative is 0 at all other points.

1.3 Exercise: Boolean Functions and the Perceptron

A perceptron classifies inputs $\mathbf{x} = (x_1, x_2)$ with $x_1, x_2 \in \{0, 1\}$ via the rule $\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + w_0)$.

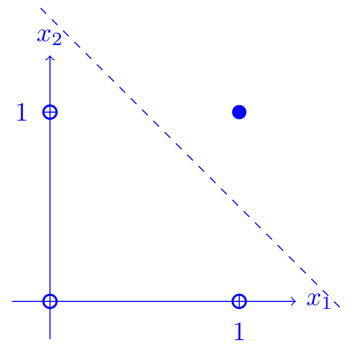
1. Find weights w_1, w_2 and bias w_0 such that the perceptron computes the AND function (output $+1$ if and only if $x_1 = 1$ and $x_2 = 1$).
2. Find weights such that the perceptron computes the OR function (output $+1$ if and only if at least one of x_1, x_2 equals 1).
3. Prove that no choice of weights can compute the XOR function (output $+1$ if and only if exactly one of x_1, x_2 equals 1).
4. What does this tell us about the perceptron algorithm on XOR-labeled data?

Solution:

1. We need $\text{sign}(w_1 x_1 + w_2 x_2 + w_0)$ to output $+1$ only at $(1, 1)$:

$$\begin{aligned} (0,0) \to -1: &\quad w_0 < 0 \\ (0,1) \to -1: &\quad w_2 + w_0 < 0 \\ (1,0) \to -1: &\quad w_1 + w_0 < 0 \\ (1,1) \to +1: &\quad w_1 + w_2 + w_0 \geq 0 \end{aligned}$$

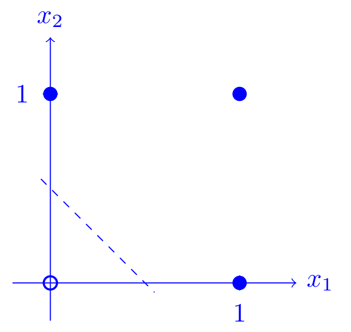
   For example, $w_1 = w_2 = 1, w_0 = -1.5$ satisfies all four constraints. The decision boundary $x_1 + x_2 = 1.5$ separates $(1, 1)$ from the other three points.



2. For OR, we need to output $+1$ whenever at least one input is 1:

$$\begin{aligned}(0,0) &\to -1: \quad & w_0 &< 0\\ (0,1) &\to +1: & w_2 + w_0 &\geq 0\\ (1,0) &\to +1: & w_1 + w_0 &\geq 0\\ (1,1) &\to +1: & w_1 + w_2 + w_0 &\geq 0\end{aligned}$$

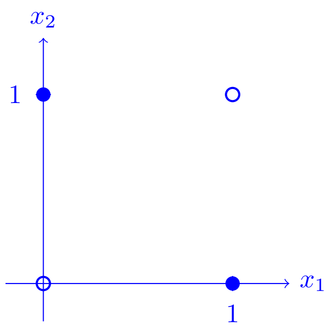
For example, $w_1 = w_2 = 1, w_0 = -0.5$. The decision boundary $x_1 + x_2 = 0.5$ separates $(0,0)$ from the rest.



3. For XOR, we need:

$$\begin{aligned}(0,0) &\to -1: \quad & w_0 &< 0\\ (0,1) &\to +1: & w_2 + w_0 &\geq 0 \implies w_2 \geq -w_0 > 0\\ (1,0) &\to +1: & w_1 + w_0 &\geq 0 \implies w_1 \geq -w_0 > 0\\ (1,1) &\to -1: & w_1 + w_2 + w_0 &< 0\end{aligned}$$

Adding the second and third inequalities gives $w_1 + w_2 \geq -2w_0$. But the fourth requires $w_1 + w_2 < -w_0$. Combining: $-2w_0 \leq w_1 + w_2 < -w_0$. For this to hold, we need $-2w_0 < -w_0$, i.e., $w_0 > 0$. This contradicts $w_0 < 0$, so no perceptron can compute XOR.



4. The XOR function is not linearly separable: no line can separate the $+1$ points from the -1 points. The perceptron is only guaranteed to converge when the data are linearly separable. On XOR data, the perceptron never converges; it cycles indefinitely through the training points.

   This impossibility is a special case of a general result: Minsky and Papert's *Perceptrons* (1969) proved that single-layer perceptrons can only compute linearly separable functions. While multi-layer networks could in principle overcome this limitation, no practical training algorithm was known, and neural network research declined through the 1970s, a period called the AI winter. The field revived with Rumelhart, Hinton and Williams (1986), who showed that backpropagation could train multi-layer networks to learn nonlinear representations of the input. For XOR, a single hidden layer suffices: it maps the four points into a space where they are linearly separable.

2 Probabilistic Classification

The perceptron outputs a class label but does not estimate the probability of each class. A probabilistic classifier instead assigns a probability to each class. There are two ways to do this: a discriminative model specifies $p(y|\mathbf{x})$ directly, while a generative model specifies the class prior and class-conditional distribution, then recovers the posterior via Bayes' rule.

2.1 Discriminative Models

A discriminative model learns $p(y|\mathbf{x})$ without modeling how $\mathbf{x}$ is generated.

2.1.1 Logistic Regression

Logistic regression is a discriminative model for binary classification. The two classes are labeled 0 and 1. Given weights $\mathbf{w}$ and intercept w_0, we model the probability of label y given features $\mathbf{x}$:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + w_0)$$
$$p(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w}^\top \mathbf{x} + w_0)$$

σ denotes the sigmoid function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

The sigmoid takes any value $z \in \mathbb{R}$ and returns an output in $(0, 1)$.

To find the best weights $\mathbf{w}$, we maximize the likelihood of our data. Unlike OLS, there is no closed-form solution. We minimize the negative log-likelihood (NLL): negating turns the maximization into a minimization, and the logarithm converts the product of likelihoods into a sum without changing the optimum.

After fitting, we predict the class for a new point $\mathbf{x}^*$ by computing

$$p(y^* = 1|\mathbf{x}^*; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}^* + w_0)$$
$$p(y^* = 0|\mathbf{x}^*; \mathbf{w}) = 1 - p(y^* = 1|\mathbf{x}^*; \mathbf{w})$$

and assigning whichever class has the higher probability. Although the sigmoid function is nonlinear, the decision boundary of logistic regression is linear: it is the set of points where $\mathbf{w}^\top \mathbf{x} + w_0 = 0$, a hyperplane.

We can extend logistic regression to K classes using the softmax function, which generalizes the sigmoid:

$$\text{softmax}_k(\mathbf{z}) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

Notation. We write $\hat{y}_i = p(y_i = 1|\mathbf{x}_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i + w_0)$ for the predicted probability. We use $\mathcal{L}$ for the loss function, L for the likelihood and ℓ for the log-likelihood.

2.1.2 Exercise: Deriving the Negative Log-Likelihood for Logistic Regression

In this exercise, we derive the loss function for logistic regression from first principles using maximum likelihood estimation (MLE).

Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $y_i \in \{0, 1\}$, and the logistic regression model $p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x})$ (absorbing the intercept into $\mathbf{w}$), let $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$.

1. **Single data point likelihood.** Using the "power trick," show that the likelihood for a single data point $(\mathbf{x}_i, y_i)$ can be written compactly as:

$$p(y_i \mid \mathbf{x}_i; \mathbf{w}) = \hat{y}_i^{y_i}(1 - \hat{y}_i)^{1-y_i}$$

   Verify that this expression is correct for both $y_i = 1$ and $y_i = 0$.

2. **Joint likelihood.** Assuming the data points are independent and identically distributed (i.i.d.), write down the likelihood of the entire dataset $L(\mathbf{w})$.

3. **Log-likelihood.** Take the logarithm of the joint likelihood to obtain the log-likelihood $\ell(\mathbf{w})$. Why is it helpful to work with the log-likelihood instead of the likelihood directly?

4. **Negative log-likelihood (NLL).** Write down the NLL loss $\mathcal{L}_{\text{NLL}}(\mathbf{w}) = -\ell(\mathbf{w})$ and expand it in terms of $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$.

5. **Sigmoid derivative.** Show that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

6. **Gradient of the NLL.** Using this property and the chain rule, compute the gradient $\nabla_{\mathbf{w}} \mathcal{L}_{\text{NLL}}(\mathbf{w})$ and show that it simplifies to:

$$\nabla_{\mathbf{w}} \mathcal{L}_{\text{NLL}}(\mathbf{w}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)\mathbf{x}_i$$

Solution:

1. Suppose we have a single data point with label $y \in \{0, 1\}$. Its likelihood can be expressed as $\hat{y}^y(1 - \hat{y})^{1-y}$, which evaluates to $\hat{y}$ if $y = 1$ and $(1 - \hat{y})$ if $y = 0$. So the likelihood is the probability of the observed class. We can use this trick to construct likelihood expressions:

$$p(y|\mathbf{x}) = p(y = 1|\mathbf{x})^y \cdot p(y = 0|\mathbf{x})^{1-y}$$

   This is the probability mass function of a Bernoulli distribution with parameter $\hat{y}_i$.

2. Since the data points are i.i.d., the joint likelihood is the product of individual likelihoods:

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i; \mathbf{w}) = \prod_{i=1}^{N} \hat{y}_i^{y_i}(1 - \hat{y}_i)^{1-y_i}$$

3. Taking the logarithm:

$$\ell(\mathbf{w}) = \sum_{i=1}^{N} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$

The $\log$ turns products into sums, which are easier to differentiate. Since $\log$ is monotonically increasing, the value of $\mathbf{w}$ that maximizes the log-likelihood also maximizes the original likelihood.

4. The negative log-likelihood loss (also called the "binary cross-entropy loss") is:

$$\mathcal{L}_{\text{NLL}}(\mathbf{w}) = -\ell(\mathbf{w}) = -\sum_{i=1}^{N} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$

Substituting $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$:

$$\mathcal{L}_{\text{NLL}}(\mathbf{w}) = -\sum_{i=1}^{N} \Big[y_i \ln \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \ln\big(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\big) \Big]$$

5. Write $\sigma(z) = (1 + e^{-z})^{-1}$ and apply the chain rule:

$$\begin{aligned} \sigma'(z) &= -(1 + e^{-z})^{-2} \cdot \frac{d}{dz}(1 + e^{-z}) \\ &= -(1 + e^{-z})^{-2} \cdot (-e^{-z}) \\ &= \frac{e^{-z}}{(1 + e^{-z})^2} \end{aligned}$$

Now observe that $1 - \sigma(z) = 1 - \frac{1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}}$. This lets us factor:

$$\sigma'(z) = \underbrace{\frac{1}{1 + e^{-z}}}_{\sigma(z)} \cdot \underbrace{\frac{e^{-z}}{1 + e^{-z}}}_{1-\sigma(z)} = \sigma(z)(1 - \sigma(z))$$

6. We use the chain rule: $\nabla_{\mathbf{w}} \mathcal{L}_{\text{NLL}} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}_{\text{NLL}}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \mathbf{w}}$. We compute each factor separately.

Factor 1: Differentiate the NLL with respect to $\hat{y}_i$. The i-th term of the NLL is $-y_i \ln \hat{y}_i - (1 - y_i) \ln(1 - \hat{y}_i)$, so:

$$\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i}$$

Factor 2: Differentiate $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$ with respect to $\mathbf{w}$. By the chain rule and Part 5:

$$\frac{\partial \hat{y}_i}{\partial \mathbf{w}} = \sigma'(\mathbf{w}^\top \mathbf{x}_i) \cdot \mathbf{x}_i = \hat{y}_i(1 - \hat{y}_i)\mathbf{x}_i$$

Combining: Multiply the two factors and sum over data points:

$$\nabla_{\mathbf{w}} \mathcal{L}_{\text{NLL}} = \sum_{i=1}^{N} \left[-\frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} \right] \hat{y}_i(1 - \hat{y}_i)\mathbf{x}_i$$

Distribute $\hat{y}_i(1 - \hat{y}_i)$ into each term of the bracket:

- First term: $-\dfrac{y_i}{\hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i) = -y_i(1 - \hat{y}_i) \quad$ (the $\hat{y}_i$ cancels)
- Second term: $\dfrac{1 - y_i}{1 - \hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i) = (1 - y_i)\hat{y}_i \quad$ (the $(1 - \hat{y}_i)$ cancels)

Expanding the sum of these two terms: $-y_i + y_i\hat{y}_i + \hat{y}_i - y_i\hat{y}_i = \hat{y}_i - y_i$. So:

$$\nabla_{\mathbf{w}}\mathcal{L}_{\text{NLL}} = \sum_{i=1}^{N}(\hat{y}_i - y_i)\mathbf{x}_i$$

The gradient at each data point is the prediction error $(\hat{y}_i - y_i)$ scaled by the input $\mathbf{x}_i$. To minimize the NLL via gradient descent, we update $\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{i=1}^{N}(\hat{y}_i - y_i)\mathbf{x}_i$.

2.2 Generative Models

A discriminative model directly specifies the posterior $p(y|\mathbf{x})$. A generative model instead specifies the joint distribution $p(\mathbf{x}, y)$, then uses Bayes' Rule to recover the posterior and pick the most likely label y:

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})} \propto p(\boldsymbol{x}|y)p(y)$$

$p(y)$ is the class prior, the probability of each class before observing features. In binary classification, this is a Bernoulli distribution. $p(\mathbf{x}|y)$ is the class-conditional distribution: given a class y, it tells us how likely we are to see features $\mathbf{x}$. We classify by picking the class k that maximizes $p(y = k|\mathbf{x})$.

2.2.1 Exercise: Shapes of Decision Boundaries I

Consider a generative model with $K > 2$ classes and output labels encoded as one-hot vectors $\boldsymbol{y}$ of length K. The class prior is $p(\boldsymbol{y} = C_k) = \pi_k$ for $k \in \{1, \dots, K\}$, and the class-conditional distributions are Gaussian:

$$p(\boldsymbol{x} \,|\, \boldsymbol{y} = C_k) = \mathcal{N}(\boldsymbol{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad k \in \{1, \dots, K\}$$

We classify a new example $\boldsymbol{x}$ by the highest posterior probability $p(\boldsymbol{y} = C_k \,|\, \boldsymbol{x})$. Show that this is equivalent to finding the class k that maximizes:

$$f_k(\boldsymbol{x}) = \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}_k|) - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k).$$

Derive f_k from the posterior $p(\boldsymbol{y} = C_k \,|\, \boldsymbol{x})$. What can we claim about the shape of the decision boundary?

Solution: Starting with the posterior:

$$p(\boldsymbol{y} = C_k \,|\, \boldsymbol{x}) = \frac{p(\boldsymbol{x} \,|\, \boldsymbol{y} = C_k)p(\boldsymbol{y} = C_k)}{\sum_{\ell=1}^{K} p(\boldsymbol{x} \,|\, \boldsymbol{y} = C_\ell)p(\boldsymbol{y} = C_\ell)}$$

The denominator is the same for each class, so we can drop it.

$$\begin{aligned}
&\propto p(\boldsymbol{x} \,|\, \boldsymbol{y} = C_k)p(\boldsymbol{y} = C_k) \\
&= \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)\right)\pi_k
\end{aligned}$$

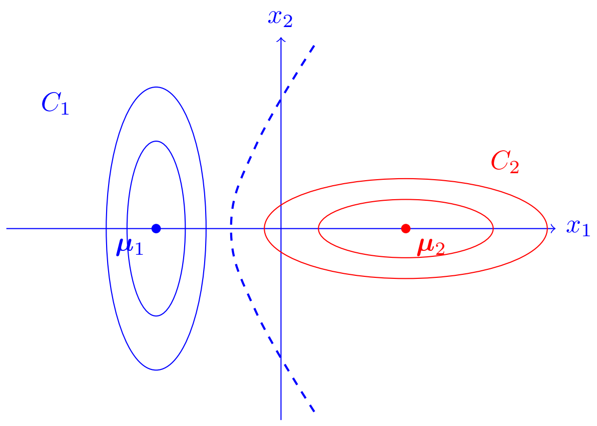
We factor out $(2\pi)^{D/2}$, which is shared across classes.

$$\propto \frac{\pi_k}{|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)\right)$$

Taking the logarithm does not change the maximizing class, giving:

$$f_k(\boldsymbol{x}) = \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}_k|) - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)$$

The decision boundary between classes k and j is the set of points where $f_k(\boldsymbol{x}) = f_j(\boldsymbol{x})$. The term $(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)$ is quadratic in $\boldsymbol{x}$, and when $\boldsymbol{\Sigma}_k \neq \boldsymbol{\Sigma}_j$, the quadratic terms do not cancel. So the decision boundary is quadratic (e.g. an ellipse or hyperbola).



When $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$, the decision boundary is quadratic.

2.2.2 Exercise: Shapes of Decision Boundaries II

Now suppose every class has the same covariance matrix, so $\boldsymbol{\Sigma}_\ell = \boldsymbol{\Sigma}_{\ell'}$ for all classes C_ℓ and $C_{\ell'}$. Simplify this formula further. What can we claim about the shape of the decision boundaries now?

Solution: We start from the result of Part I.

$$\ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}_k|) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)$$

Writing $\boldsymbol{\Sigma}$ for the shared covariance matrix:

$$\begin{aligned} f_k(\boldsymbol{x}) &= \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) \\ &= \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top (\boldsymbol{\Sigma}^{-1}\boldsymbol{x}-\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k) \\ &= \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k) \end{aligned}$$

We combine the cross terms using $\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} = \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$, which holds because $\boldsymbol{\Sigma}^{-1}$ is symmetric and the result is a scalar.

$$\begin{aligned} &= \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} - 2\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k) \\ &= \ln(\pi_k) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k \end{aligned}$$

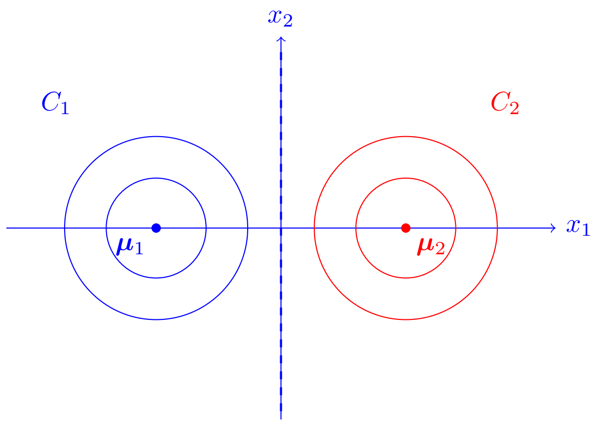
We drop $-\frac{1}{2}\ln(|\boldsymbol{\Sigma}|)$ and $-\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{x}$ since they are class-independent.

$$\propto \ln(\pi_k) + \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$$

Thus, we can use the function

$$\tilde{f}_k(\boldsymbol{x}) = \boldsymbol{x}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln(\pi_k)$$

Since $\tilde{f}_k(\boldsymbol{x})$ is linear in $\boldsymbol{x}$, the decision boundaries are no longer quadratic but linear.



When $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2$, the decision boundary is linear.

3 Evaluating Classifiers

Simple accuracy is misleading when classes are imbalanced: a 99% accuracy rate means little if 99% of examples belong to one class. Accuracy also treats all errors alike, when in practice some are far more costly than others.

3.1 Confusion Matrix

A confusion matrix summarizes classification performance using four counts.

	Predicted Positive	Predicted Negative
Actually Positive	True Positive (TP)	False Negative (FN)
Actually Negative	False Positive (FP)	True Negative (TN)

3.2 TPR and FPR

The true positive rate (TPR) is the fraction of actual positive instances that the model correctly identifies. TPR is also known as sensitivity or recall.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The false positive rate (FPR) is the fraction of actual negative instances that the model incorrectly classifies as positive:

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

The true negative rate (TNR) is the fraction of actual negative instances that the model correctly identifies:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - \text{FPR}$$

The false negative rate (FNR) is the fraction of actual positive instances that the model incorrectly classifies as negative:

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}} = 1 - \text{TPR}$$

3.3 ROC Curve and AUC

The receiver operating characteristic (ROC) curve plots TPR against FPR at different classification thresholds. Lowering the threshold classifies more examples as positive, which increases both the true positive rate and the false positive rate.

At threshold 0, every example is classified positive ($\text{TPR} = \text{FPR} = 1$); at threshold 1, none are ($\text{TPR} = \text{FPR} = 0$). A perfect classifier reaches the top-left corner of the ROC curve, catching every positive ($\text{TPR} = 1$) with no false alarms ($\text{FPR} = 0$).

The area under the ROC curve (AUC) quantifies the tradeoff between TPR and FPR across all thresholds; it ranges from 0 to 1.

Formally, AUC is the integral of the ROC curve over FPR $\in [0, 1]$. A random classifier's ROC curve follows the diagonal from $(0, 0)$ to $(1, 1)$, giving AUC $= 0.5$; a perfect classifier has AUC $= 1$. AUC is useful for comparing classifiers against each other, not only against a random baseline.

3.4 Exercise: Computing ROC Curves and AUC

A bank uses a classifier to flag potentially fraudulent credit card transactions. The classifier outputs a score between 0 and 1, where higher scores indicate higher confidence that the transaction is fraudulent. A transaction is flagged as fraud ("positive") if its score is $\geq t$ for some threshold t. The results for four transactions are:

Transaction	Score	Ground Truth
A	0.9	Fraud
B	0.6	Legitimate
C	0.4	Fraud
D	0.1	Legitimate

1. At threshold $t = 0.5$, write down the confusion matrix and compute the TPR and FPR.
2. Compute the TPR and FPR at each distinct threshold value. Plot the ROC curve.
3. Compute the AUC.
4. The fraud prevention team wants to catch every fraudulent transaction (maximize TPR). The customer experience team wants to ensure no legitimate transaction is ever blocked (minimize FPR). Using the ROC curve, discuss what threshold each team would prefer.

Solution:

1. At $t = 0.5$, we flag transactions A and B (scores ≥ 0.5) and let C and D through (scores < 0.5).

	Flagged	Not Flagged
Fraud	TP = 1 (A)	FN = 1 (C)
Legitimate	FP = 1 (B)	TN = 1 (D)

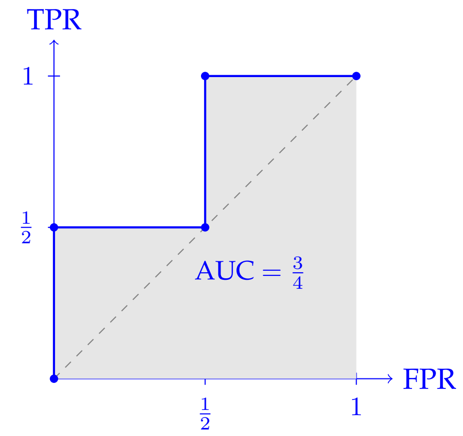
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{1}{1+1} = \frac{1}{2}$$
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = \frac{1}{1+1} = \frac{1}{2}$$

2. There are $P = 2$ fraudulent transactions (A, C) and $N = 2$ legitimate transactions (B, D). At each threshold:

Threshold	Flagged	TPR	FPR
$t > 0.9$	none	0	0
$0.6 < t \leq 0.9$	A	$\frac{1}{2}$	0
$0.4 < t \leq 0.6$	A, B	$\frac{1}{2}$	$\frac{1}{2}$
$0.1 < t \leq 0.4$	A, B, C	1	$\frac{1}{2}$
$t \leq 0.1$	A, B, C, D	1	1

The ROC curve plots (FPR, TPR):

$$(0,0) \to \left(0,\tfrac{1}{2}\right) \to \left(\tfrac{1}{2},\tfrac{1}{2}\right) \to \left(\tfrac{1}{2},1\right) \to (1,1)$$



3. The ROC curve forms a staircase. The area consists of two rectangles: one from FPR $= 0$ to $\frac{1}{2}$ at height $\frac{1}{2}$, and one from FPR $= \frac{1}{2}$ to 1 at height 1.

$$\text{AUC} = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times 1 = \frac{1}{4} + \frac{1}{2} = \frac{3}{4}$$

Since AUC $= 0.75 > 0.5$, this classifier performs better than random guessing.

4. The fraud prevention team (maximize TPR) would choose threshold $0.1 < t \leq 0.4$, achieving TPR $= 1$ (all fraud is caught) at the cost of FPR $= \frac{1}{2}$ (half of legitimate transactions are incorrectly blocked).

   The customer experience team (minimize FPR) would choose threshold $0.6 < t \leq 0.9$, achieving FPR $= 0$ (no legitimate transactions blocked) at the cost of TPR $= \frac{1}{2}$ (half of fraudulent transactions go undetected).

   There is no threshold that achieves both TPR $= 1$ and FPR $= 0$. The ROC curve shows the achievable tradeoffs, and choosing a threshold requires weighing the cost of undetected fraud against the cost of blocking legitimate customers.