CS189
Machine Learning
Notes

Instructor: Jonathan Shewchuk

Kelvin Lee

UC Berkeley

Contents

1

Introduction

1.1 Classification

Goal: Predict which class each new data point belongs to give a set of known classified points.
Classifiers: k-nearest-neighbor classifier, linear classifier.

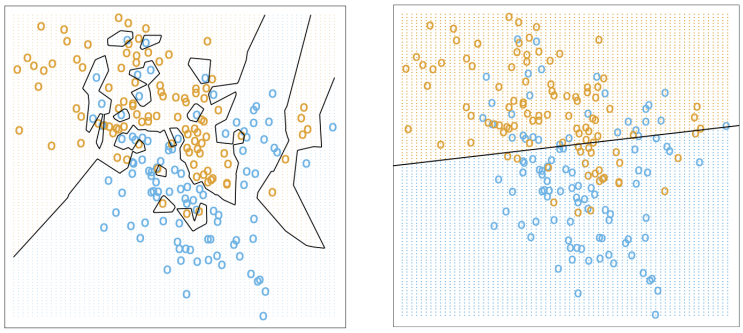

Figure 1.1: *Left:* 1*-nearest-neighbor Classifier. Right: Linear Classifier.*

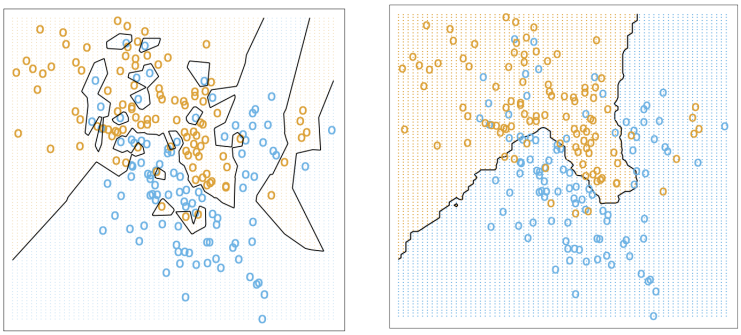

Figure 1.2: *Left:* 1*-nearest-neighbor Classifier. Right:* 15*-nearest-neighbor Classifier.*

Advantages: Left figure correctly classifies all the data points. Right figure has smoother decision boundary, which is more likely to correspond to reality.
Disadvantages: There is an **overfitting** issue with the left figure. The decision boundary is too intricate.

1.1.1 Classifying Digits

We express images as vectors where each entry correspond to each pixel. The linear decision boundary is a **hyperplane**.

1.1.2 Validation

Question. What k should we use for nearest-neighbor classifier and how do we choose between nearest-neighbor classifier or linear classifier?

We figure these out through a process called **validation**. The basic steps are as follows:

- Train a classifier: it learns to distinguish 7 from not 7 for example.
- Test the classifier on **new** images.

There are two types of errors that could happen in these stages: **training set error** and **test set error**.

Definition 1.1.1 (Training set error)**.** Fraction of training images not classified correctly.

Remark. For 1-nearest-neighbor classifier it would be 0 because each point is classified correctly, whereas for linear classifier, it is always positive because we cannot classify all points correctly using a linear decision boundary. Note that zero error in the training set is an indication of overfitting.

Definition 1.1.2 (Test set error)**.** Fraction of misclassified **new** images, not seen during training.

Definition 1.1.3 (Outliers)**.** Points whose labels are atypical.

Definition 1.1.4 (Overfitting)**.** When the test error deteriorates because the classifier becomes too sensitive to outliers or other spurious patterns.

Remark. In machine learning, the goal is to create a classifier that **generalizes** to new examples that are unseen yet. Overfitting is counterproductive to that goal. So we're seeking a compromise: we want decision boundaries that make fine distinctions without being downright superstitious.

Most ML algorithms have a few **hyperparameters** that control over/underfitting, for example the k in k-nearest neighbor algorithm.

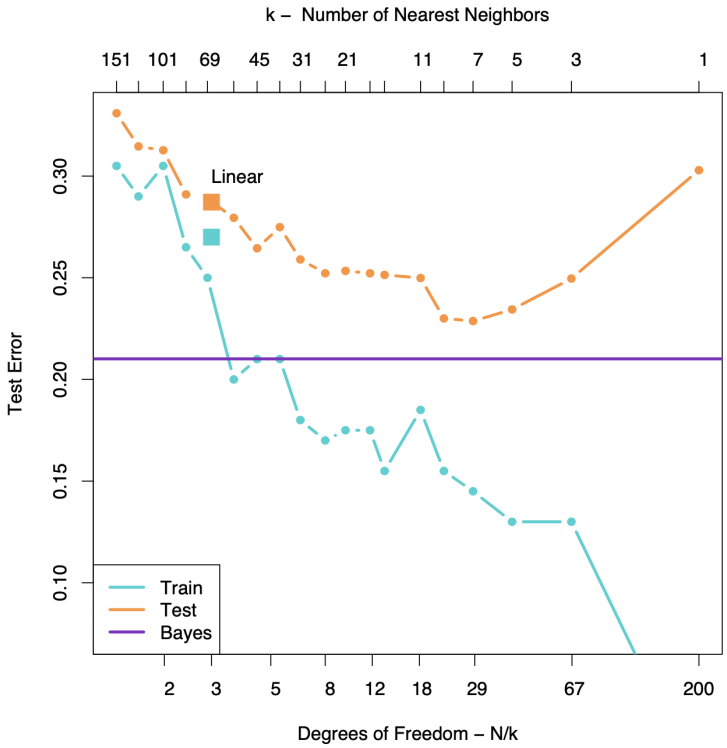

Figure 1.3: The orange curves are test and the blue are training error for k-nearest-neighbor classification. We see that at $k = 7$, which is optimal, is when the test error is the lowest. Underfitting happens when k gets too large and overfitting happens when k is too small. ($k = 1$ has zero training error)

Remark. A low training error does not imply a low testing error. We choose the optimal hyperparameters based on the testing error.

We select hyperparameters by **validation**:

- Hold back a subset of the labeled data, called the **validation set**, which is chosen randomly.
- Train the classifier multiple times with different hyperparameter settings.
- Choose the setting that work best on validation set.

Now we have 3 sets:

- **Training set:** used to learn model weights.
- **Validation set:** used to tune hyperparameters, choose among different models.
- **Test set:** used as **final** evaluation of model. Run **once** at the very end.

2

Linear Classifiers and Perceptron

2.1 Classifiers

Given sample of n observations, each with d features. Some observations belong to class C; some do not. We represent each observation as a point in d-dimensional space, called a **sample point** / **feature vector** / **independent variables**.

Definition 2.1.1 (Decision boundary)**.** The boundary chosen by our classifier to separate items in the class from those not.

Definition 2.1.2 (Overfitting)**.** When sinuous decision boundary fits sample points so well that it doesn't classify future points well.

Some classifiers work by computing a **decision function**:

Definition 2.1.3 (Decision function)**.** Also called **predictor function** or **discriminant function**. It is a function $f(\boldsymbol{x})$ that maps a sample point $\boldsymbol{x}$ to a scalar such that

$$\begin{aligned} f(\boldsymbol{x}) &> 0 && \text{if } \boldsymbol{x} \in \text{class } C; \\ f(\boldsymbol{x}) &\leq 0 && \text{if } \boldsymbol{x} \notin \text{class } C. \end{aligned}$$

For these classifiers, the decision boundary is $\{\boldsymbol{x} \in \mathbb{R}^d \mid f(\boldsymbol{x}) = 0\}$, i.e., a set of points where the decision function is zero. Usually, this set is a $(d-1)$-dimensional surface in $\mathbb{R}^d$.

Definition 2.1.4. $\{\boldsymbol{x} \mid f(\boldsymbol{x}) = 0\}$ is also called an **isosurface** of f for the **isovalue** 0. f has other isosurfaces for other isovalues, e.g., $\{\boldsymbol{x} \mid f(\boldsymbol{x}) = 1\}$.

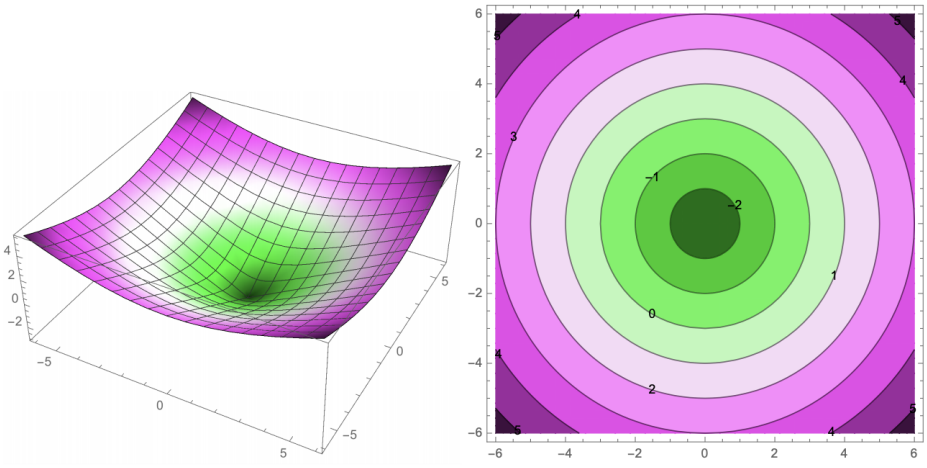

Figure 2.1: 3D plot and isocontour plot of the cone $f(x, y) = \sqrt{x^2 + y^2} - 3$

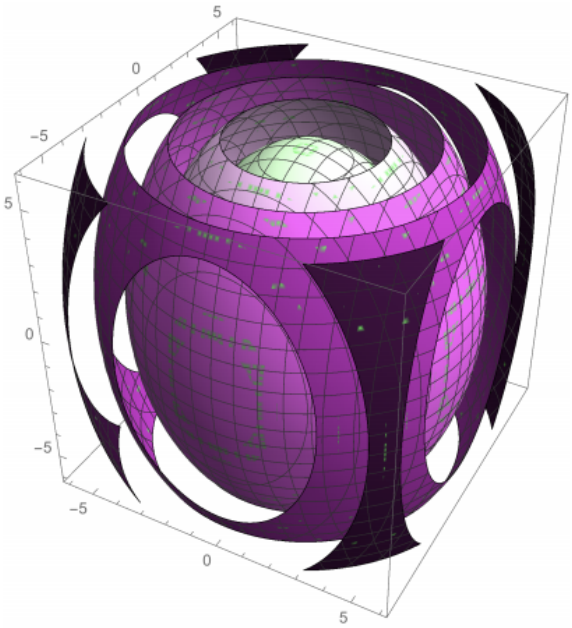

Figure 2.2: One of these spheres could be the decision boundary.

For linear classifier, the decision boundary is a line / plane. Usually a linear decision function is used. Sometimes there is no decision function, which will be covered later.

2.1.1 Math Review

2.1.1.1 Vectors

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix}^\top$$

2.1.1.2 Inner Product (Dot Product)

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{d} x_i y_i$$

2.1.1.3 Euclidean Norm

$$\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}} = \sqrt{\sum_{i=1}^{d} x_i^2}$$

$\|\boldsymbol{x}\|$ is the **Euclidean length** of a vector $\boldsymbol{x}$. The unit vector $\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$ is obtained by normalization.
Dot products can be used to compute angles:

$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \cdot \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|}$$

$\boldsymbol{x} \cdot \boldsymbol{y} > 0$	acute angle
$\boldsymbol{x} \cdot \boldsymbol{y} = 0$	right angle
$\boldsymbol{x} \cdot \boldsymbol{y} < 0$	obtuse angle

2.1.1.4 Hyperplane

Given a linear decision function $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + \alpha$, the decision boundary is

$$\mathcal{H} = \{\boldsymbol{x} \mid \boldsymbol{w} \cdot \boldsymbol{x} + \alpha = 0\},$$

which is a **hyperplane** (a line in 2D, a plane in 3D). Three most important points about hyperplane to keep in mind:

- It has dimension $d - 1$.
- It cuts the d-dimensional space into two halves.
- It's flat and infinite.

Theorem 2.1.5. Let $\boldsymbol{x}, \boldsymbol{y}$ be two points that lie on $\mathcal{H}$. Then $\boldsymbol{w} \cdot (\boldsymbol{y} - \boldsymbol{x}) = 0$.

Proof. Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are both in $\mathcal{H}$, we have

$$\boldsymbol{w} \cdot (\boldsymbol{y} - \boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{y} - \boldsymbol{w} \cdot \boldsymbol{x} = -\alpha - (-\alpha) = 0.$$

□

Recall that if a dot product is equal to 0, it means that the two vectors are **orthogonal** to each other. In this case, we see that $\boldsymbol{w}$ is orthogonal to each vector in $\mathcal{H}$ and we call it the **normal vector** of $\mathcal{H}$.

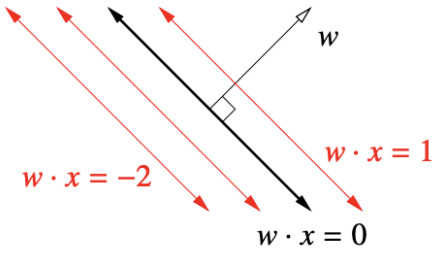

Figure 2.3: Hyperplane

If $\boldsymbol{w}$ is a unit vector, then $\boldsymbol{w} \cdot \boldsymbol{x} + \alpha$ is the **signed distance** from $\boldsymbol{x}$ to $\mathcal{H}$ (the distance between $\boldsymbol{x}$ and the closest point on $\mathcal{H}$). It is positive on $\boldsymbol{w}$'s side of $\mathcal{H}$ and negative otherwise.

The coefficients in $\boldsymbol{w}$, and α are called **weight** (or **parameters** or **regression coefficients**).

Definition 2.1.6 (Linearly separable)**.** The input data is **linearly separable** if there exists a hyperplane that separates all the sample points in class C from all the points not in class C.

2.1.2 A Simple Classifier

2.1.2.1 Centroid method

We compute mean $\boldsymbol{\mu}_C$ of all points in class C and mean $\boldsymbol{\mu}_X$ of all points not in class C and use the decision function

$$f(\boldsymbol{x}) = \underbrace{(\boldsymbol{\mu}_C - \boldsymbol{\mu}_X)}_{\text{normal vector}} \cdot \boldsymbol{x} - (\boldsymbol{\mu}_C - \boldsymbol{\mu}_X) \cdot \underbrace{\frac{(\boldsymbol{\mu}_C + \boldsymbol{\mu}_X)}{2}}_{\text{midpoint between } \boldsymbol{\mu}_C, \boldsymbol{\mu}_X}$$

so the decision boundary is the hyperplane that bisects line segment with endpoints $\boldsymbol{\mu}_C$,$\boldsymbol{\mu}_X$.

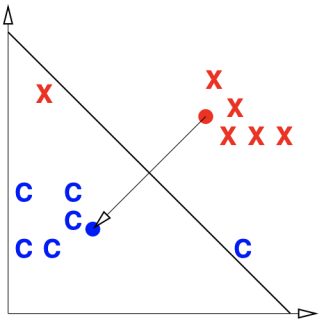

Figure 2.4: Centroid method

We can sometimes improve the classifier by adjusting the scalar term to minimize the number of misclassified points. Then the hyperplane has the same normal vector, but a different position.

2.1.3 Perceptron Algorithm (Frank Rosenblatt, 1957)

Slow but correct for linearly separable points only. It uses a **numerical optimization** algorithm, namely, **gradient descent**.

Consider n sample points $X_1, X_2, \ldots, X_n$. For each sample point, the **label**

$$y_i = \begin{cases} 1 & \text{if } X_i \in \text{class } C, \\ -1 & \text{otherwise.} \end{cases}$$

For simplicity, we only consider decision boundaries that pass through the origin (we'll fix this later). Our goal is to find weight $\boldsymbol{w}$ such that

$$\begin{aligned} \boldsymbol{x}_i \cdot \boldsymbol{w} &\geq 0 \quad \text{if } y_i = 1, \\ \boldsymbol{x}_i \cdot \boldsymbol{w} &< 0 \quad \text{if } y_i = -1. \end{aligned}$$

Equivalently, we have the following **constraint**

$$y_i \boldsymbol{x}_i \cdot \boldsymbol{w} \geq 0.$$

We define a **risk function** $\mathcal{R}$ that's positive if some constraints are violated. Then we use **optimization** to chosse $\boldsymbol{w}$ that minimizes $\mathcal{R}$.

Definition 2.1.7 (Loss function)**.**

$$L(z, y_i) = \begin{cases} 0 & \text{if } y_i z \geq 0, \\ -y_i z & \text{otherwise,} \end{cases}$$

where z is the prediction and y_i is the correct label.

The loss function is zero if z has the same sign as y_i and is positive otherwise. We want to reduce the loss function down to zero or close to zero eventually.

Definition 2.1.8 (Risk function)**.** Also called the **objective function** or **cost function**

$$\begin{aligned} \mathcal{R}(\boldsymbol{w}) &= \sum_{i=1}^{n} L\left(X_i \cdot \boldsymbol{w}, y_i\right) \\ &= \sum_{i \in V} -y_i X_i \cdot \boldsymbol{w} \qquad (V \text{ is the set of indices } i \text{ for which } y_i X_i \cdot \boldsymbol{w} < 0). \end{aligned}$$

If $\boldsymbol{w}$ classifies all $X_1, \ldots, X_n$ correctly, then $\mathcal{R}(\boldsymbol{w}) = 0$. Otherwise, $\mathcal{R}(\boldsymbol{w})$ is positive, and we want to find a better $\boldsymbol{w}$. Then we have an optimization problem: find $\boldsymbol{w}$ that minimizes $\mathcal{R}(\boldsymbol{w})$.

3

Perceptron Learning; Maximum Margin Classifiers

3.1 Perceptron Algorithm

Recall the following:

- **Linear decision function** $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$ (for simplicity, no α).
- **Decision boundary** $\{\boldsymbol{x} \mid f(\boldsymbol{x}) = \boldsymbol{0}\}$, i.e., (a **hyperplane** through the origin).
- **Sample points** $X_1, \ldots, X_n \in \mathbb{R}^d$; **class labels** $y_1, \ldots, y_n = \pm 1$.
- **Goal:** find weights $\boldsymbol{w}$ such that $y_i X_i \cdot \boldsymbol{w} \geq 0$.
- **Revised goal:** find $\boldsymbol{w}$ that minimizes the **risk function**

$$R(\boldsymbol{w}) = \sum_{i \in V} -y_i X_i \cdot \boldsymbol{w},$$

  where V is the set of indicies for which $y_i X_i \cdot \boldsymbol{w} < 0$.

Original problem: find a separating hyperplane in one space called the $\boldsymbol{x}$**-space**.
Transformed problem: find an optimal point in a different space called the $\boldsymbol{w}$**-space**.

Transformation from $\boldsymbol{x}$**-space** to objects in $\boldsymbol{w}$**-space**:

$\boldsymbol{x}$**-space**		$\boldsymbol{w}$**-space**	
hyperplane:	$\{\boldsymbol{z} \mid \boldsymbol{w} \cdot \boldsymbol{z} = 0\}$	point:	$\boldsymbol{w}$
point:	$\boldsymbol{x}$	hyperplane:	$\{\boldsymbol{z} \mid \boldsymbol{x} \cdot \boldsymbol{z} = 0\}$

Point $\boldsymbol{x}$ lies on hyperplane $\{\boldsymbol{z} \mid \boldsymbol{w} \cdot \boldsymbol{z} = 0\} \iff \boldsymbol{w} \cdot \boldsymbol{x} = 0 \iff$ point $\boldsymbol{w}$ lies on hyperplane $\{\boldsymbol{z} \mid \boldsymbol{x} \cdot \boldsymbol{z} = 0\}$.

A hyperplane transforms to its normal vector; a sample point transforms to the hyperplane whose normal vector is the sample point.

Remark. This transformation happens to be **symmetric**: a hyperplane in x-space transforms to a point in w-space the same way that a hyperplane in w-space transforms to a point in x-space. However, this is **not** always true for the decision boundaries learned in this class.

If we want to enforce inequality $x \cdot w \geq 0$, that means:

- In x-space, x should be on the same side of the hyperplane $\{z \mid w \cdot z = 0\}$ as w.
- In w-space, w should be on the same side of the hyperplane $\{z \mid x \cdot z = 0\}$ as x.

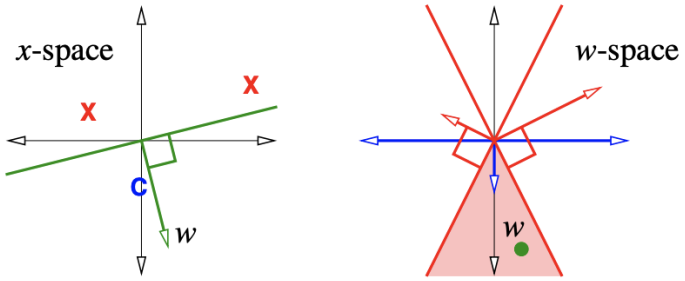

Figure 3.1: *Left.* Green arrow represents the chosen w (must be chosen within the shaded region in the right figure). *Right.* Red arrows are the points that are not in class C. Since they are not in class C, we restrict w to the **opposite** side of that hyperplane from that normal vector, resulting in the shaded region.

We have switched from the problem of finding a hyperplane in x-space to the problem of finding a point in w-space, which will be helpful for us to think about optimization algorithms.

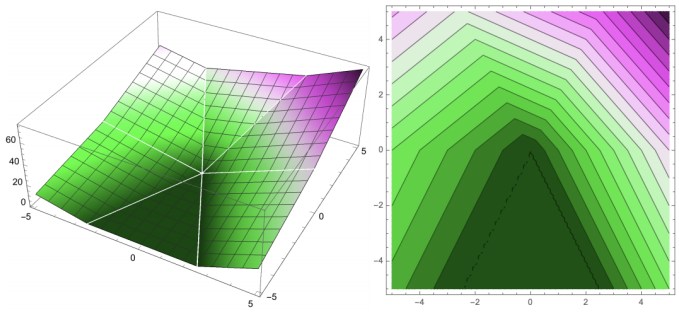

Figure 3.2: Risk function of the three sample points from previous example. R's creases match the w-space above.

We have an optimization problem and now we will use an optimization algorithm: **gradient descent** to solve it.

3.1.1 Gradient Descent

Given a starting point w, find **gradient** of R with respect to w (this is the direction of **steepest ascent**. Take a step in the **opposite** direction.

Recall from vector calculus that:

$$\nabla R(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial R}{\partial w_1} \\ \vdots \\ \frac{\partial R}{\partial w_d} \end{bmatrix} \quad \Longrightarrow \quad \nabla(\boldsymbol{z} \cdot \boldsymbol{w}) = \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix} = \boldsymbol{z}.$$

$$\nabla R(\boldsymbol{w}) = \sum_{i \in V} \nabla - y_i X_i \cdot \boldsymbol{w} = -\sum_{i \in V} y_i X_i.$$

At any point $\boldsymbol{w}$, we walk downhill in direction of **steepest descent**: $-\nabla R(\boldsymbol{w})$.

Algorithm 1 Gradient Descent

$\boldsymbol{w} \leftarrow$ arbitrary nonzero starting point (good choice is any $y_i X_i$ because then we have $y_i^2 \|X_i\|^2 \geq 0$, which means at least one point is correctly classified in the beginning)
while $R(\boldsymbol{w}) > 0$ **do**
  $V \leftarrow$ set of indices i for which $y_i X_i \cdot \boldsymbol{w} < 0$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \epsilon \sum_{i \in V} y_i X_i$
return $\boldsymbol{w}$

$\epsilon > 0$ is the **step size** (or **learning rate**, chosen empirically.

Cons: Slow! Each step takes $O(nd)$ time.

3.1.2 Stochastic Gradient Descent

Idea: each step, pick **one** misclassified X_i and do gradient descent on **loss function** $L(X_i \cdot \boldsymbol{w}, y_i)$.

The algorithm where we apply stochastic gradient descent to the loss function is called the **perceptron algorithm**, in which each step takes $O(d)$ time.

Algorithm 2 Perceptron Algorithm

while some $y_i X_i \cdot \boldsymbol{w} < 0$ **do**
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \epsilon y_i X_i$
return $\boldsymbol{w}$

Remark. Stochastic gradient does not work for every problem that gradient descent works for.

Question. What if separating hyperplane doesn't pass through the origin?

Answer. Add a **fictitious** dimension. For example, if initially $d = 2$, then our decision function becomes

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + \alpha = \begin{bmatrix} w_1 & w_2 & \alpha \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Now we have sample points in $\mathbb{R}^{d+1}$, all lying on hyperplane $\boldsymbol{x}_{d+1} = 1$, and we run the perceptron algorithm in $(\boldsymbol{d+1})$-dimensional space.

3.1.3 Maximum Margin Classifiers

Definition 3.1.1 (Margin)**.** The **margin** of a linear classifier is the distance from the decision boundary to the nearest sample point.

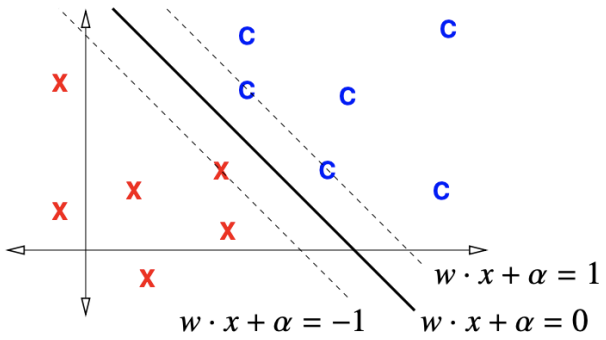

Figure 3.3: Margin is maximized.

We enforce the constraints

$$y_i(\boldsymbol{w} \cdot X_i + \alpha) \geq 1 \qquad \text{for } i \in [1, n],$$

which is a better way to formulate the problem and it is impossible for $\boldsymbol{w}$ to get set to zero.

Recall that if $\|\boldsymbol{w}\| = 1$, signed distance from hyperplane to X_i is $\boldsymbol{w} \cdot X_i + \alpha$. Otherwise, it is $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot X_i + \frac{\alpha}{\|\boldsymbol{w}\|}$. Then the margin would be

$$\min_i \frac{1}{\|\boldsymbol{w}\|} \underbrace{|\boldsymbol{w} \cdot X_i + \alpha|}_{\geq 1} \geq \frac{1}{\|\boldsymbol{w}\|}.$$

There is a slab of width $\frac{2}{\|\boldsymbol{w}\|}$ containing no sample points.

To maximize the margin, we minimize $\|\boldsymbol{w}\|$. Then we have an optimization problem:

$$\begin{aligned} &\text{Find } \boldsymbol{w} \text{ and } \alpha \text{ that minimize} && \|w\|^2 \\ &\text{subject to} && y_i(X_i \cdot w + \alpha) \geq 1 \quad \forall i \in [1, n]. \end{aligned}$$

This is called the **quadratic program** in $d+1$ dimensions and n constraints. It has one **unique** solution if the points are **linearly separable** and no solution otherwise.

Remark. We minimize $\|\boldsymbol{w}\|^2$ instead of $\|\boldsymbol{w}\|$ because $\|\boldsymbol{w}\|$ is not smooth at $\boldsymbol{w} = \boldsymbol{0}$, whereas $\|\boldsymbol{w}\|^2$ is smooth everywhere, which makes optimization easier.

The solution gives us a **maximum margin classifier**, also called a **hard-margin support vector machine**.

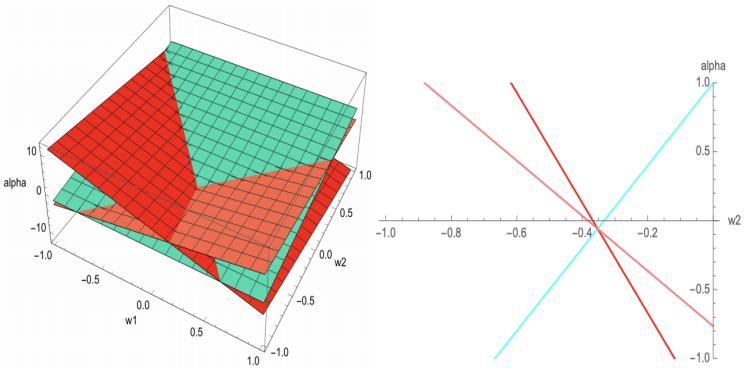

Figure 3.4: Linear constraints in the weight space (w_1, w_2, α). We look for the point nearest the origin that lies above the blue plane (in-class) but below the red and pink planes (out-of-class). Here the optimal point lies at the intersection of the three planes. At right the constraints say that the solution must lie in the leftmost pizza slice, while being as close to the origin as possible, which is the intersection of the three lines.

4

Soft-Margin Support Vector Machines; Features

4.1 Soft-Margin Support Vector Machines (SVMs)

Solve 2 problems:

1. Hard-margin SVMs fail if data is not linearly separable.
2. Hard-margin SVMs are sensitive to outliers.

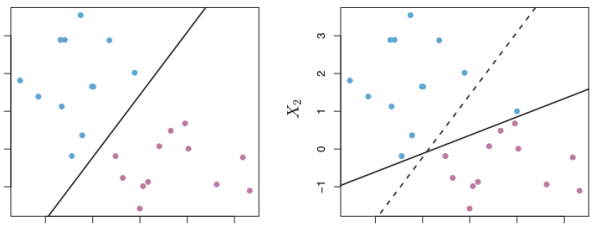

Figure 4.1: Example where one outlier moves the hard-margin SVM decision boundary a lot.

Idea: Allow some points to violate the margin, with **slack variables**. The modified constraint for point i becomes:

$$y_i(X_i \cdot \boldsymbol{w} + \alpha) \geq 1 - \xi_i.$$

Remark. We also impose new constraints that the slack variables are **nonnegative**:

$$\xi_i \geq 0.$$

This ensures that all sample points that *don't* violate the margin are treated the same; they all have $\xi_i = 0$. Point i has nonzero ξ_i if and only if it violates the margin.

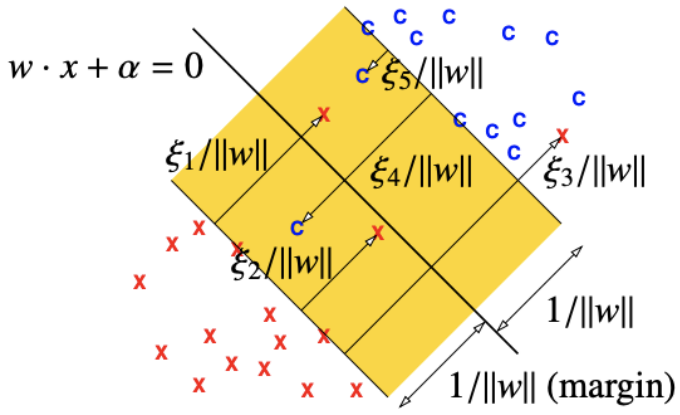

Figure 4.2: A margin where some points violate the margin.

Remark. For soft-margin SVMs, we redefine the word *margin*. Instead of being the distance from the decision boundary to the nearest sample point, it is defined to be $1/\|\boldsymbol{w}\|$.

To prevent abuse of slack, we add a **loss term** to the objective function. Then the optimization problem we have is:

$$\begin{aligned} \text{Find } \boldsymbol{w}, \alpha \text{ and } \xi_i \text{ that minimize} \quad & \|w\|^2 + C \textstyle\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i \left(X_i \cdot w + \alpha\right) \geq 1 - \xi_i \quad \forall i \in [1, n] \\ & \xi_i \geq 0 \qquad\qquad \forall i \in [1, n]. \end{aligned}$$

We have a **quadratic program** in $d + n + 1$ dimensions and $2n$ constraints.

Definition 4.1.1 (Quadratic Program)**.** A problem of optimizing a **quadratic** objective function with **linear** constraints.

$C > 0$ is a scalar **regularization hyperparameter** that trades off:

	small C	**big C**
desire	maximize margin $1/\|\boldsymbol{w}\|$	keep most slack variables zero or small
danger	underfitting (misclassifies much training data)	overfitting (awesome training, awful test)
outliers	less sensitive	very sensitive
boundary	more *flat*	more sinuous

Remark. The last row only applies to **nonlinear** boundaries. A linear decision boundary cannot be *sinuous*.

Recall that we use **validation** to choose C.

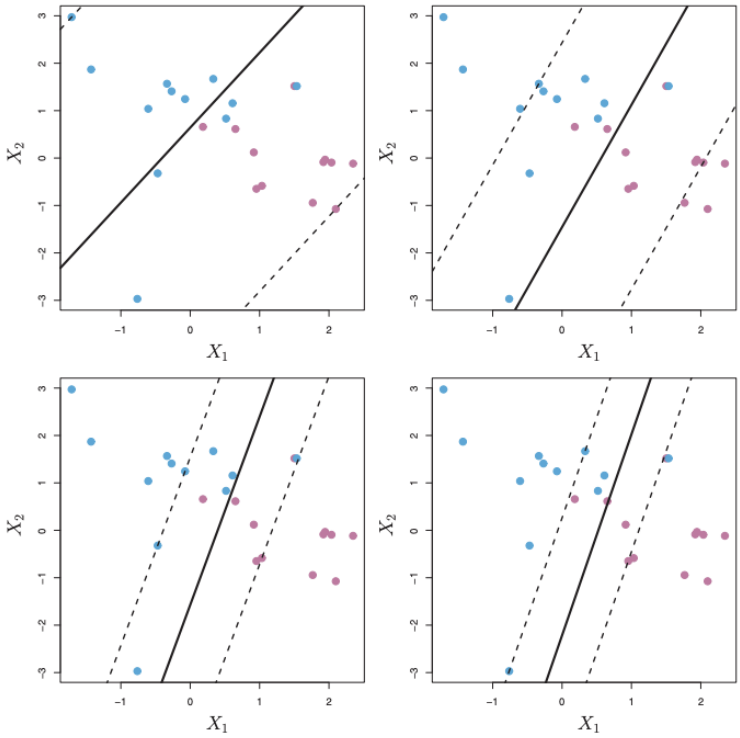

Figure 4.3: Examples of how the slab varies with C.
Upper left. Smallest C; *Lower right.* Largest C.

The **support vectors** are the data points that lie **closest** to the decision boundary, i.e., the points that are within the margin (including those on the margin).

Remark. Here's a great analogy between slack and money: think about slack as the fine you have to pay if a point violates the margin, i.e., the further a point penetrates, the more you have to pay. Our goal is to spend least money while maintaining a widest margin. Hence, if C is small, we are willing to spend lots of money on violations to get a wider margin. If C is big, it means we are cheap and we won't pay much for violations despite a narrower margin. If C is infinite, we have a hard-margin SVM.

4.2 Features

Question. How to do nonlinear decision boundaries?

Answer. Make **nonlinear features** that *lift* points into a higher-dimensional space.

High-dimensional linear classifier $\rightarrow$ low-dimensional nonlinear classifier.

Example 4.2.1 (The parabolic lifting map)**.**

$$\Phi : \mathbb{R}^d \to \mathbb{R}^{d+1}$$

$$\Phi(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x} \\ \|\boldsymbol{x}\|^2 \end{bmatrix} \qquad \text{(lifts } \boldsymbol{x} \text{ onto paraboloid } x_{d+1} = \|\boldsymbol{x}\|^2\text{)}$$

(Note that the first entry $\boldsymbol{x}$ is a vector of dimension d, not a scalar. Hence, the whole vector has length $d+1$ instead of 2) The additional feature gives our linear classifier more power. For example, if we find a linear classifier in Φ-space, it induces sphere classifier in $\boldsymbol{x}$-space:

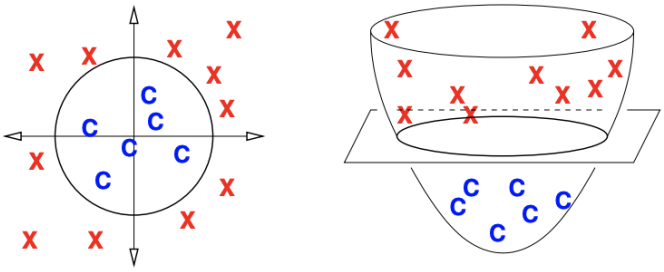

Figure 4.4: The third axis is $x_3 = x_1^2 + x_2^2$. The decision boundary is a circle obtained from the intersection of the paraboloid and the hyperplane.

Theorem 4.2.1. $\Phi(X_1), \ldots, \Phi(X_n)$ are **linearly separable** if and only if $X_1, \ldots, X_n$ are separable by a **hypersphere**. (Possibly an ∞-radius hypersphere = hyperplane.)

Proof. Consider hypersphere in $\mathbb{R}^d$ with center $\boldsymbol{c}$ and radius ρ. Points inside:

$$\|\boldsymbol{x} - \boldsymbol{c}\|^2 < \rho^2$$

$$\|\boldsymbol{x}\|^2 - 2\boldsymbol{c} \cdot \boldsymbol{x} + \|\boldsymbol{c}\|^2 < \rho^2$$

$$\underbrace{\begin{bmatrix} -2\boldsymbol{c}^\top & 1 \end{bmatrix}}_{\text{normal vector in } \mathbb{R}^{d+1}} \underbrace{\begin{bmatrix} \boldsymbol{x} \\ \|\boldsymbol{x}\|^2 \end{bmatrix}}_{\Phi(\boldsymbol{x})} < \rho^2 - \|\boldsymbol{c}\|^2.$$

Hence points inside the sphere $\iff$ same side of hyperplane in Φ-space. □

Example 4.2.1 (Axis-aligned ellipsoid / hyperboloid decision boundaries)**.** In 3D, these have the formula

$$Ax_1^2 + Bx_2^2 + Cx_3^2 + Dx_1 + Ex_2 + Fx_3 + \alpha = 0 \qquad \text{(Capital letters are scalars, not matrices)}$$

$$\Phi : \mathbb{R}^d \to \mathbb{R}^{2d}$$

$$\Phi(\boldsymbol{x}) = \begin{bmatrix} x_1^2 & \dots & x_d^2 & x_1 & \dots & x_d \end{bmatrix}^\top$$

Hyperplane is

$$\underbrace{\begin{bmatrix} A & B & C & D & E & F \end{bmatrix}}_{\boldsymbol{w}} \cdot \Phi(\boldsymbol{x}) + \alpha = 0.$$

Here we have turned d input features into $2d$ features for our linear classifier. If the points are separable by an axis-aligned ellipsoid or hyperboloid, per the formula above, then the points lifted to Φ-space are separable by a hyperplane whose normal vector is $\begin{bmatrix} A & B & C & D & E & F \end{bmatrix}$.

Example 4.2.2 (Ellipsoid / hyperboloid)**.** 3D formula for a general ellipsoid or hyperboloid:

$$Ax_1^2 + Bx_2^2 + Cx_3^2 + Dx_1x_2 + Ex_2x_3 + Fx_3x_1 + Gx_1 + Hx_2 + Ix_3 + \alpha = 0$$

$$\Phi(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^{(d^2+3d)/2}$$

Isosurface defined by this equation is called a **quadric**. In two dimensions, it's also known as a **conic section**. So the decision boundary can be an arbitrary conic section.

Remark. There is a quadratic blowup in the number of features because every *pair* of input features creates a new feature in Φ-space. If the dimension is large, these feature vectors will get huge, and thus a serious computational cost is imposed. Thus, we would want to find good classifiers for data that aren't linearly separable.

Example 4.2.3 (Decision function is degree-p polynomial)**.** A cubic in $\mathbb{R}^2$:

$$\Phi(\boldsymbol{x}) = \begin{bmatrix} x_1^3 & x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2 & x_1x_2 & x_2^2 & x_1 & x_2 \end{bmatrix}^\top$$

$$\Phi(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^{O(d^p)}.$$

Now we are really blowing up the number of features! If we have 100 features per sample point and we decide to use degree-4 decision functions, then each lifted feature vector has a length of roughly 4 million, and it will take forever for the algorithm to run.

There is an extremely clever trick that allows us to work with huge feature vectors very quickly without ever computing them: **kernelization**.

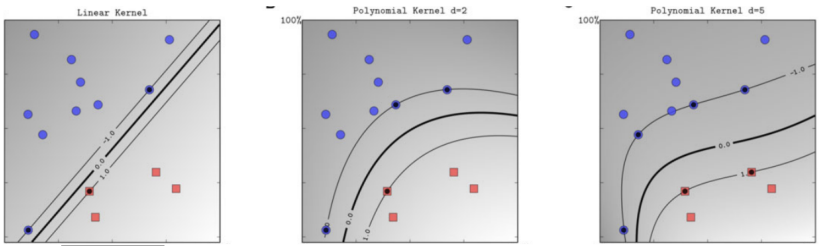

Figure 4.5: Hard-margin SVMs with degree $1, 2, 5$ decision functions. Margin tends to get wider as degree increases.

Increasing the degree accomplishes two things:

1. The data might become **linearly separable** when lifting them to a high enough degree, even if they were not linearly separable.
2. Raising the degree can widen the margin, so we might get a more robust decision boundary that generalizes better to test data.

Remark. If the degree is too high, we will have an **overfitting** problem.

Example 4.2.4 (Edge Detection)**.** An **edge detector** is an algorithm for approximating g grayscale / color gradients in image, for example

- tap filter
- Sobel filter
- oriented Gaussian derivative filter

  Remark. Images are discrete, not continuous fields, so we need to approximate the gradients.

  Collect line orientations in local histograms (each having 12 orientation bins per region); use histograms as features (instead of raw pixels).

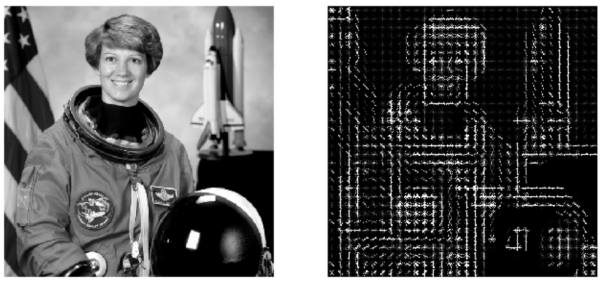

Figure 4.6: *Left.* Input image. *Right.* Histogram of Oriented Gradients.

5

Machine Learning Abstractions and Numerical Optimization

5.1 Machine Learning Abstractions

There are four levels of abstraction that could help think about machine learning:

1. **Application/Data**
    - Are the data **labeled** (**classified**) or not?
    - Yes: **classification** or **regression**?
    - No: **clustering** or **dimensionality reduction**?
2. **Model**
    - Decision functions: linear, polynomial, logistic, neural net, etc.
    - Nearest neighbors, decision trees
    - Features
    - Low vs. high capacity (affects overfitting, underfitting, inference)
3. **Optimization Problem**
    - Variables, objective functions, constraints
    - For example, unconstrained, convex program, least squares, PCA
4. **Optimization Algorithm**
    - Gradient descent, simplex, SVD.

Remark. If one level is changed, then we will probably have to change all the levels underneath it. In addition, not all machine learning methods fit this four-level decomposition.

5.2 Optimization Problems

5.2.1 Uncontrained

Goal: Find $\boldsymbol{w}$ that minimizes (or maximizes) a continuous **objective function** $f(\boldsymbol{w})$, which is **smooth** if its gradient is continuous.

Definition 5.2.1 (Global Minimum)**.** A **global minimum** of f is a value $\boldsymbol{w}$ such that $f(\boldsymbol{w}) \leq f(\boldsymbol{v})$ for every $\boldsymbol{v}$.

Definition 5.2.2 (Local Maximum)**.** A **local minimum** of f is a value $\boldsymbol{w}$ such that $f(\boldsymbol{w}) \leq f(\boldsymbol{v})$ for every $\boldsymbol{v}$ centered around $\boldsymbol{w}$.

Remark. Usually, finding a local minimum is easy but finding a global minimum is hard (or impossible).

However, there's an exception:

Definition 5.2.3 (Convex)**.** A function is **convex** if for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the line segment connecting $(\boldsymbol{x}, f(\boldsymbol{x}))$ to $(\boldsymbol{y}, f(\boldsymbol{y}))$ does not go below f.

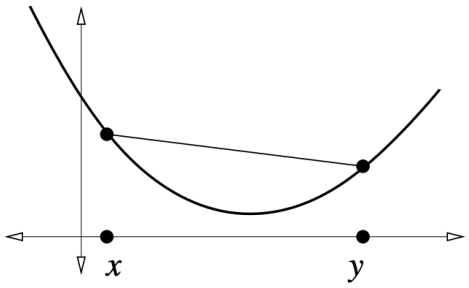

Figure 5.1: An example of a convex function.

A continuous convex function on a closed, convex domain has either

- no minimum (goes to ∞), or
- one local minimum, or
- a connected set of local minima that are all global minima with equal f. (perceptron risk function)

Algorithms for smooth f

1. **Gradient descent:**
   - blind (repeat $\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \nabla f(\boldsymbol{w})$)
   - with line search:
     - secant method
     - Newton–Raphson (may need Hessian matrix of f)

- stochastic (blind) (trains on one point per iteration, or a small batch)

2. **Newton's method** (needs Hessian matrix)
3. **Nonlinear conjugate gradient** (uses the secant or Newton–Raphson line search methods)

Algorithms for non-smooth f

1. **Gradient descent:**
   - blind
   - with direct line search (e.g., golden section search)
2. **BFGS** (**Broyden–Fletcher–Goldfarb–Shanno**)

Remark. These algorithms find a local minimum.

Definition 5.2.4 (Line Search)**.** **Line search** finds a **local minimum** along the search direction by solving an optimization problem in 1D, which is much easier than higher-dimensional one.

Remark. **Neural nets** are **unconstrained** optimization problems with many local minima. They sometimes benefit from line searches or second-order optimization algorithms, but when the input data set is very large, stochastic versions of gradient descent would be an ideal choice.

5.2.2 Constrained Optimization (smooth equality constraints)

Find $\boldsymbol{w}$ that minimizes (or maximizes) $f(\boldsymbol{w})$

subject to $g(\boldsymbol{w}) = 0$ (this is an isosurface and g is a smooth function)

Algorithm: Use **Lagrange multipliers** to transform constrained to unconstrained optimization.

5.2.3 Linear Program

A **linear program** consists of a **linear objective function** and **linear inequality constraints**.

Find $\boldsymbol{w}$ that maximizes (or minimizes) $\boldsymbol{c}\dot{\boldsymbol{w}})$

subject to $A\boldsymbol{w} \leq \boldsymbol{b}$ where A is $n \times d$ matrix, $\boldsymbol{b} \in \mathbb{R}^n$, expressing n **linear constraints**:
$A_i \cdot \boldsymbol{w} \leq b_i \quad \forall i \in [1, n]$.

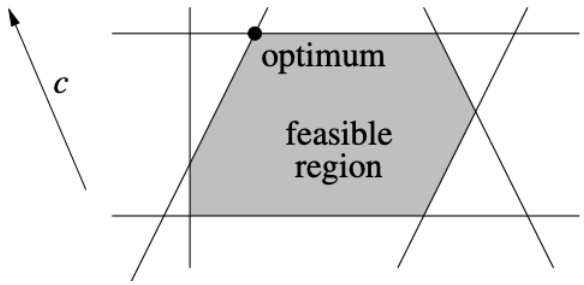

Figure 5.2: The point is the **optimum** since it is furthest in the direction $\boldsymbol{c}$. The two lines that are incident to the vertex where the point lies are **active constraints**.

The set of points $\boldsymbol{w}$ that satisfy all constraints is a convex **polytope** called the **feasible region** (shaded).

Definition 5.2.5 (Convex)**.** A point set $\mathcal{P}$ is **convex** if for every $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{P}$, the **line segment** with endpoints $\boldsymbol{p}, \boldsymbol{q}$ lies entirely in $\mathcal{P}$.

Definition 5.2.6 (Active Constraints)**.** The optimum achieves **equality** for some constraints called the **active constraints** of the optimum. In an SVM, active constraints correspond to the sample points that touch or violate the slab, i.e. the **support vectors**.]

Remark. There can be more than one optimal point. For example, consider the case when the $\boldsymbol{c}$ from above is orthogonal to one of the active constraint. The set of optimal points is always convex.

Example 5.2.1. Every **feasible point** $(\boldsymbol{w}, \alpha)$ gives a linear classifier:

$$\text{Find } \boldsymbol{w}, \text{ that maximizes } 0$$

$$\text{subject to } y_i(\boldsymbol{w} \cdot X_i+) \geq 1 \quad \forall i \in [1, n].$$

Remark. The data are linearly separable if and only if the feasible region is not the **empty set**. This is also true for maximum margin classifier (quadratic program).

Algorithms for linear programming:

- **Simplex** (George Dantzig, 1947)
- **Interior point methods**

Remark. The hard part of linear programming is figuring out which constraints should be the active constraints because there are exponentially many possibilities. A linear program solver can find a linear classifier, but cannot find the maximum margin classifier. We need something more powerful.

5.2.4 Quadratic Program

Recall that a **quadratic program** consists of a quadratic objective function and linear inequality constraints.

$$\text{Find } \boldsymbol{w}, \text{ that minimizes } f(\boldsymbol{w}) = \boldsymbol{w}^\top Q \boldsymbol{w} + \boldsymbol{c}^\top \boldsymbol{w}$$

$$\text{subject to } A\boldsymbol{w} \leq \boldsymbol{b} \quad \text{where } Q \text{ is a symmetric, positive definite matrix.}$$

Definition 5.2.7 (Positive Definite)**.** A matrix is **positive definite** if $\boldsymbol{w}^\top Q \boldsymbol{w} > 0$ fro all $\boldsymbol{w} \neq \boldsymbol{0}$.

There is only one local minimum, which is also the global minimum.

Question. What if Q is not positive definite?

Answer. If Q is indefinite, then f is not convex, the minimum is not always unique, and quadratic programming is NP-hard. If Q is positive semidefinite, meaning $\boldsymbol{w}^\top Q \boldsymbol{w} \geq 0$ for all $\boldsymbol{w}$, then f is convex and quadratic programming but there may be infinitely many solutions.

6

Decision Theory; Generative and Discriminative Models

6.1 Decision Theory (Risk Minimization)

Multiple sample points with different classes could lie at same point. Thus, we need a probabilistic classifier.

Definition 6.1.1 (Loss function)**.** A **loss function** $L(z, y)$ specifies badness if classifier predicts z, true class is Y. One commonly used loss function is the **$0-1$ loss function**, where the loss value is 1 for incorrect predictions (**symmetrical**) and 0 for correct.

Remark. False negative is way worse than false positive, thus should be weighed differenly.

Definition 6.1.2 (Decision Rule)**.** A **decision rule** (**classifier**) is a function $r : \mathbb{R}^d \to \pm 1$ that maps a feature vector $\boldsymbol{x}$ to 1 or -1 (in class / not in class).

Definition 6.1.3 (Risk)**.** The **risk** for r is the expected loss over all values of x, y:

$$\begin{aligned} R(r) &= \mathbb{E}[L(r(X), Y)] \\ &= \sum_x \left(L(r(\boldsymbol{x}), 1)\mathbb{P}(Y = 1 \mid X = \boldsymbol{x}) + L(r(\boldsymbol{x}), -1)\mathbb{P}(Y = -1 \mid X = \boldsymbol{x})\right)\mathbb{P}(X = \boldsymbol{x}) \\ &= \mathbb{P}(Y = 1)\sum_{\boldsymbol{x}} L(r(\boldsymbol{x}), 1)\mathbb{P}(X = \boldsymbol{x} \mid Y = 1) + \mathbb{P}(Y = -1)\sum_{\boldsymbol{x}} L(r(\boldsymbol{x}), -1)\mathbb{P}(X = \boldsymbol{x} \mid Y = -1). \end{aligned}$$

Sometimes we also write

$$R(r) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[L(r(\boldsymbol{x}), y)]$$

Definition 6.1.4 (Bayes Decision Rule)**.** The **Bayes decision rule** (or **Bayes classifier**) is the function r^* that minimizes **functional** (function of a function) $R(r)$. Assuming $L(z, y) = 0$ for $z = y$:

$$r^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } L(-1,1)\mathbb{P}(Y = 1 \mid X = \boldsymbol{x}) > L(1,-1)\mathbb{P}(Y = -1 \mid X = \boldsymbol{x}), \\ -1 & \text{otherwise.} \end{cases}$$

Equivalently, we can also write

$$r^*(\boldsymbol{x}) = \arg\min_j \sum_{k=1}^{K} L(j, k)\mathbb{P}(Y = k \mid X = \boldsymbol{x}).$$

Remark. **When L is symmetric, pick the class with the biggest posterior probability**. If the loss function is asymmetric then weight the posteriors with the losses.

To see this, consider the special case where we have the $0-1$ function. Then

$$r^*(\boldsymbol{x}) = \arg\min_j \sum_{k\neq j} \mathbb{P}(Y=k \mid X=\boldsymbol{x}) = \arg\min_j 1-\mathbb{P}(Y=j \mid X=\boldsymbol{x}) = \arg\max_j \mathbb{P}(Y=j \mid X=\boldsymbol{x}),$$

which is equivalent to selecting the class that **maximizes** the **posterior distribution**.

Definition 6.1.5 (Bayes Risk)**.** The **Bayes risk** (or **optimal risk**), is the risk of the Bayes classifier.

6.2 Continuous Distributions Review

Suppose we have a continuous random variable X with f as its PDF. Then

$$\int_{\infty}^{\infty} f(x)dx = 1.$$

Definition 6.2.1 (Expectation)**.** The **expectation** (or **mean**) of a continuous random variable X is defined as:

$$\mu = \mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x)dx.$$

Similarly, the **expected value** of a function of X, $g(X)$ is defined as

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx.$$

Definition 6.2.2 (Variance)**.** The **variance** of X is defined as

$$\sigma^2 = \text{var}(X) = \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

Definition 6.2.3 (Risk)**.** In the continuous case, the **risk** is simply just

$$\begin{aligned} R(r) &= \mathbb{E}[L(r(X), Y)] \\ &= \mathbb{P}(Y=1)\int L(r(\boldsymbol{x}),1)f(\boldsymbol{x} \mid Y=1)d\boldsymbol{x} + \mathbb{P}(Y=-1)\int L(r(\boldsymbol{x}),-1)f(\boldsymbol{x} \mid Y=-1)d\boldsymbol{x}. \end{aligned}$$

Equivalently, we can write

$$R(r) = \int L(r(\boldsymbol{x}), y)d\mathbb{P}(\boldsymbol{x}, y).$$

For Bayes decision rule, Bayes risk is the area under minimum of functions above. Assuming $L(z,y) = 0$ for $z = y$:

$$R(r^*) = \int \min_{y=\pm 1} L(-y,y)f(\boldsymbol{x} \mid Y=y)P(Y=y)d\boldsymbol{x}.$$

If L is $0-1$ loss, then $R(r) = \mathbb{P}(r(\boldsymbol{x}) \text{ is wrong})$ and the **Bayes optimal decision boundary** is $\{\boldsymbol{x} \mid \underbrace{\mathbb{P}(Y=1 \mid \boldsymbol{x})}_{\text{decision function}} = \underbrace{0.5}_{\text{isovalue}}\}$.

6.2.1 3 Ways To Build Classifiers

1. **Generative models** (e.g., LDA)
    - Assume sample points come from probability distributions, different for each class.
    - Guess form of distributions.
    - For each class C, fit distribution parameters to class C points, giving $\mathbb{P}(X \mid Y = C)$.
    - Bayes' Theorem gives $\mathbb{P}(Y \mid X)$.
    - If 0-1 loss, pick class C that maximizes posterior probability $\mathbb{P}(Y = C \mid \boldsymbol{x})$, which also equivalently maximizes $\mathbb{P}(\boldsymbol{x} \mid Y = C)\mathbb{P}(Y = C)$.
2. **Discriminative models** (e.g., logistic regression)
    - Model $\mathbb{P}(Y \mid X)$ directly
3. Find decision boundary (e.g., SVM)
    - Model $r(\boldsymbol{x})$ directly (no posterior)

Advantage of 1 and 2: $\mathbb{P}(Y \mid X)$ tells us probability our guess is wrong, which is something SVMs don't do.

Advantage of 1: we can diagnose outliers: $\mathbb{P}(X)$ is very small.

Disadvantages of 1: often hard to estimate distributions accurately.

7

Gaussian Discriminant Analysis (QDA and LDA)

7.1 Gaussian Discriminant Analysis (GDA)

Fundamental assumption: each class comes from normal distribution (Gaussian). $X \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2)$:

$$f(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}\|^2}{2\sigma^2}\right).$$

For each class C, suppose we estimate mean $\boldsymbol{\mu_C}$, variance $\boldsymbol{\sigma_C^2}$, and prior $\boldsymbol{\pi_C} = \mathbb{P}(Y = C)$.
Given $\boldsymbol{x}$, Bayes decision rule $r^*(\boldsymbol{x})$ returns class C that maximizes $f(\boldsymbol{x} \mid Y = C)\pi_C$.

$\ln \omega$ is monotonically increasing for $\omega > 0$, so it is equivalent to maximize

$$Q_C(\boldsymbol{x}) = \ln\left((\sqrt{2\pi})^d f_C(\boldsymbol{x})\pi_C\right) = -\frac{\|\boldsymbol{x} - \boldsymbol{\mu_C}\|^2}{2\sigma_C^2} - d\ln\sigma_C + \ln\pi_C.$$

7.1.1 Quadratic Discriminant Analysis (QDA)

Suppose there are 2 classes C, D. Then

$$r^*(\boldsymbol{x}) = \begin{cases} C & \text{if } Q_C(\boldsymbol{x}) > Q_D(\boldsymbol{x}), \\ D & \text{otherwise.} \end{cases}$$

Basically we pick the class with the biggest posterior probability.
Decision function is quadratic in $\boldsymbol{x}$. Bayes decision boundary is $Q_C(\boldsymbol{x}) - Q_D(\boldsymbol{x}) = 0$.

- In 1D, Bayes decision boundary may have 1 or 2 points (roots of a quadratic equation).
- In d-D, it is quadric (conic section for 2D case).

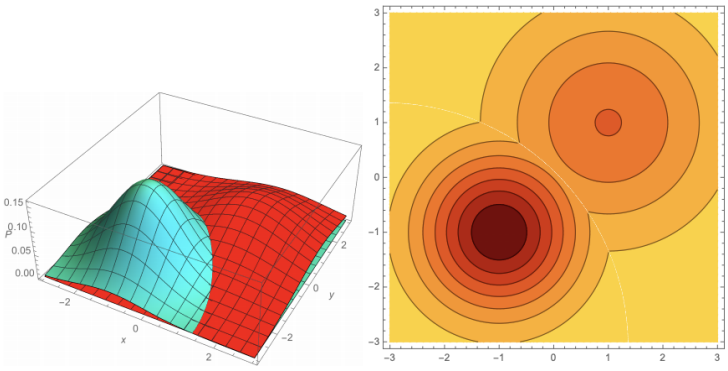

Figure 7.1: Product of gaussian distributions and prior distributions corresponding to two classes. We pick whichever gives the higher probability of a data point. *Note:* isotropic Gaussians has same variance along different directions and thus the isocontors are spheres.

Now we consider the case when there are multiple classes. The following is a picture of the isocontor of multiple classes.

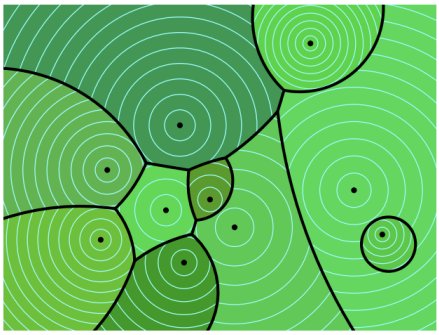

Figure 7.2: Centers of each bell curves represent the means of each Gaussian distribution and the gap between the circles represent the variance (larger gap implies higher variance). The black line represents the decision boundary.

QDA is great because it not only classifies data points for us, but also helps us determine the probability that our classification is correct.

Let's recover the posterior probabilities in the 2-class case using Bayes' rule:

$$\mathbb{P}(Y = C \mid X) = \frac{f(X \mid Y = C)\pi_C}{f(X \mid Y = C)\pi_C + f(X \mid Y = D)\pi_D}.$$

Recall that $e^{Q_C(\boldsymbol{x})} = (\sqrt{2\pi}^d f_C(\boldsymbol{x})\pi_C$ by definition, then we have

$$\begin{aligned}\mathbb{P}(Y = C \mid X = \boldsymbol{x}) &= \frac{e^{Q_C(\boldsymbol{x})}}{e^{Q_C(\boldsymbol{x})} + e^{Q_D(\boldsymbol{x})}} \\ &= \frac{1}{1 + e^{Q_D(\boldsymbol{x}) - Q_C(\boldsymbol{x})}} \\ &= s(Q_C(\boldsymbol{x}) - Q_D(\boldsymbol{x})),\end{aligned}$$

where $s(\gamma) = \frac{1}{1+e^{-\gamma}}$ is called the **logistic function**, or **sigmoid function** and $Q_C(\boldsymbol{x}) - Q_D(\boldsymbol{x})$ is the **decision function**.

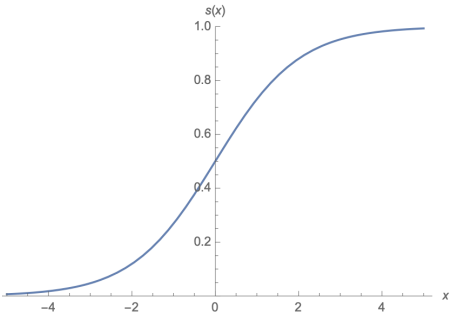

Figure 7.3: Logistic function. We interpret $s(0) = \frac{1}{2}$ as saying that on the decision boundary, there's a 50% chance that the data point belongs to either class.

7.1.2 Linear Discriminant Analysis (LDA)

LDA is a variant of QDA with linear decision boundaries. It is less likely to overfit than QDA.
Fundamental assumption: all the Gaussians have same variance σ.

$$Q_{\mathrm{C}}(\boldsymbol{x}) - Q_{\mathrm{D}}(\boldsymbol{x}) = \underbrace{\frac{(\boldsymbol{\mu_C} - \boldsymbol{\mu_D}) \cdot \boldsymbol{x}}{\sigma^2}}_{\boldsymbol{w}\cdot\boldsymbol{x}} \underbrace{- \frac{\|\boldsymbol{\mu_C}\|^2 - \|\boldsymbol{\mu_D}\|^2}{2\sigma^2} + \ln \pi_{\mathrm{C}} - \ln \pi_{\mathrm{D}}}_{+\alpha}.$$

Now it's a linear classifier because the quadratic terms cancelled out each other. Choose C that maximizes **linear discriminant function**

$$\frac{\boldsymbol{\mu_C} \cdot \boldsymbol{x}}{\sigma^2} - \frac{\|\boldsymbol{\mu_C}\|^2}{2\sigma^2} + \ln \pi_{\mathrm{C}} \quad \text{(works for any number of classes)}$$

In 2-class case: decision boundary is $\boldsymbol{w} \cdot \boldsymbol{x} + \alpha = 0$ and the posterior is $\mathbb{P}(Y = C \mid \boldsymbol{x}) = s(\boldsymbol{w} \cdot \boldsymbol{x} + \alpha)$. (The effect of $w \cdot x + \alpha$ is to scale and translate the logistic fn in x -space. It's a linear transformation.)

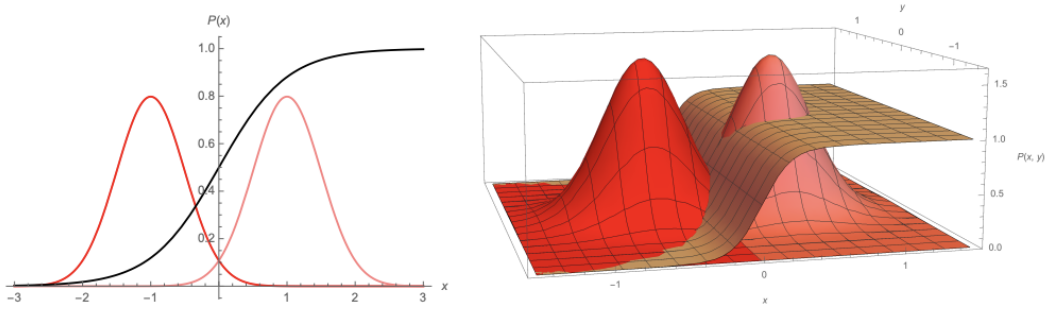

Figure 7.4: Two Gaussians and the logistic function. The logistic function is the right Gaussian divided by the sum of the Gaussians.

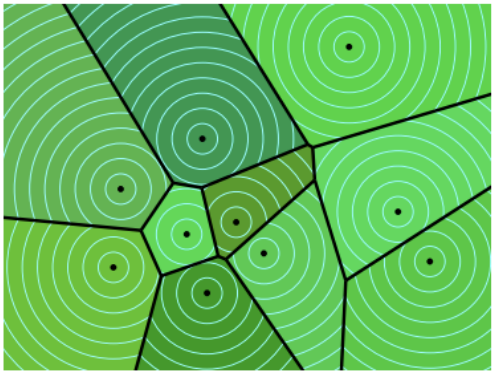

Figure 7.5: Multi-class case decision boundaries for LDA.

7.2 Maximum Likelihood Estimation Of Parameters

To use Gaussian discriminant analysis, we must first fit Gaussians to the sample points and estimate the class prior probabilities. The way we do this is through **Maximum Likelihood Estimation** (**MLE**), which is a method of estimating the parameters of a statistical model by picking the parameters that maximize the likelihood function $\mathcal{L}$.

7.2.1 Likelihood of a Gaussian

Given sample points $X_1, X_2, \ldots, X_n$, our goal is to find best-fit Gaussian. The likelihood of generating these points is

$$\mathcal{L}(\mu, \sigma; \{X_i\}_{i=1}^n) = \prod_{i=1}^n f(X_i).$$

Despite the fact that the probability of a point is zero in continuous case, we still define this as the likelihood. Since we will be dealing with products of exponentials, we can simplify a bit by just considering the **log likelihood** $\ell(\cdot)$, which is just equivalent to $\ln \mathcal{L}(\cdot)$ ($\ln$ is monotonic). Thus, we have

$$\begin{aligned}\ell(\mu, \sigma; \{X_i\}_{i=1}^n) &= \sum_{i=1}^n \ln f(X_i) \\ &= \sum_{i=1}^n \left(-\frac{\|X_i - \mu\|^2}{2\sigma^2} - d \ln \sqrt{2\pi} - d \ln \sigma \right).\end{aligned}$$

Then to find the optimal parameters, we set $\nabla_\mu \ell = 0$ and $\frac{\partial \ell}{\partial \sigma} = 0$:

$$\nabla_\mu \ell = \sum_{i=1}^n \frac{X_i - \mu}{\sigma^2} = 0 \implies \hat{\mu} = \frac{1}{n} \sum_{i=1}^n X_i$$

$$\frac{\partial \ell}{\partial \sigma} = \sum_{i=1}^n \frac{\|X_i - \mu\|^2 - d\sigma^2}{\sigma^3} = 0 \implies \hat{\sigma}^2 = \frac{1}{dn} \sum_{i=1}^n \|X_i - \mu\|^2$$

We substitute $\hat{\mu}$ for μ to compute $\hat{\sigma}^2$ since we don't know μ exactly.

For QDA, we estimate conditional mean $\hat{\mu}$ and conditional variance $\hat{\sigma}^2$ of each class C separately and estimate the priors:

$$\hat{\pi}_C = \frac{n_C}{\sum n}.$$

For LDA, we compute means and priors the same way but we have same variance for all classes:

$$\hat{\sigma}^2 = \frac{1}{dn} \sum_C \sum_{\{i:y_i=C\}} \|X_i - \hat{\mu}_C\|^2 \qquad \text{(pooled within-class variance)}.$$

Remark. Although LDA is computing one covariance for all the data, each sample point contributes with respect to its own class's mean.

8

Eigenvectors and Anisotropic Multivariate Normal Distribution

8.1 Eigenvectors

Definition 8.1.1 (Eigenvectors)**.** Given square matrix A, if $A\boldsymbol{v} = \lambda\boldsymbol{v}$ for some vector $\boldsymbol{v} \neq \mathbf{0}$, scalar λ, then $\boldsymbol{v}$ is an **eigenvector** of A and λ is the **eigenvalue** of A associated with $\boldsymbol{v}$.

Theorem 8.1.2. If $\boldsymbol{v}$ is eigenvector of A with eigenvalue λ, then $\boldsymbol{v}$ is eigenvector of A^k with eigenvalue λ^k.

Proof. Using induction, we can show that $A^2\boldsymbol{v} = A(\lambda\boldsymbol{v}) = \lambda^2\boldsymbol{v}$, etc.. □

Theorem 8.1.3. If A is invertible, then $\boldsymbol{v}$ is eigenvector of A^{-1} with $\frac{1}{\lambda}$.

Proof.

$$A^{-1}\boldsymbol{v} = A^{-1}\left(\frac{1}{\lambda}A\boldsymbol{v}\right) = \frac{1}{\lambda}\boldsymbol{v}.$$

□

Theorem 8.1.4 (Spectral Theorem)**.** Every real, symmetric $n \times n$ matrix has real eigenvalues and n eigenvectors that are mutually orthogonal, i.e.,

$$\boldsymbol{v}_i^\top \boldsymbol{v}_j = 0 \qquad \forall i \neq j.$$

Remark. A matrix can have more than n eigenvector directions. If two eigenvectors share the same eigenvalue, then every linear combination of them is also an eigenvector, hence there are infinitely many eigenvector directions, but they all span the same plane, so we just pick two that are orthogonal to each other as our basis.

8.1.1 Quadratic Forms

Consider the following examples:

$$\|\boldsymbol{z}\|^2 = \boldsymbol{z}^\top \boldsymbol{z}$$
$$\|A^{-1}\boldsymbol{x}\|^2 = \boldsymbol{x}^\top A^{-2}\boldsymbol{x}$$

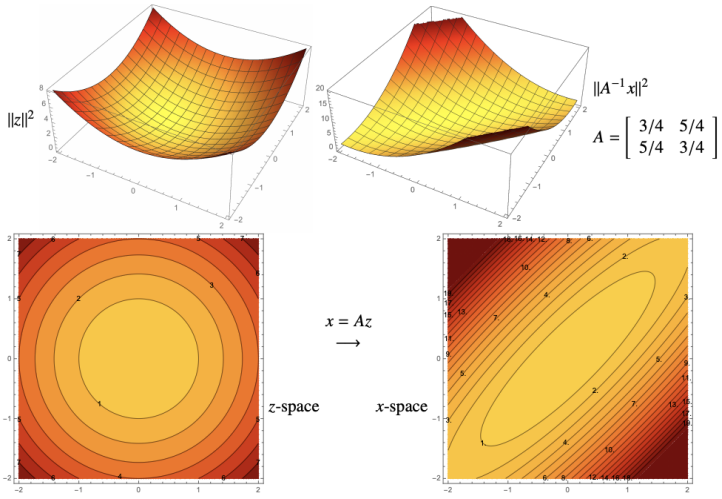

Figure 8.1: *Left.* $\|\boldsymbol{z}\|^2$. *Xight.* $\|A^{-1}\boldsymbol{x}\|^2$. Axes of the ellipsoid are the eigenvectors. The matrix A maps the circles to the ellipses. There're stretching along the direction with eigenvalue 2, and shrinking along the direction with eigenvalue $-\frac{1}{2}$.

$\|A^{-1}\boldsymbol{x}\|^2 = 1$ is an ellipsoid with axes $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ and radii $\lambda_1, \lambda_2, \ldots, \lambda_n$ because if $A^{-1}\boldsymbol{x} = \boldsymbol{v}_i$ is unit length ($\boldsymbol{v}_i$ lies on circle), $\boldsymbol{x} = A\boldsymbol{v}_i$ has length λ_i ($\boldsymbol{x}$ lies on the ellipsoid).

Remark. Special case: if A is diagonal, then the eigenvectors are coordinate axes, and the ellipsoids are axis-aligned.

Definition 8.1.5. A symmetric matrix M is

- **positive definite** if $\boldsymbol{w}^\top M\boldsymbol{w} > 0, \quad \forall \boldsymbol{w} \neq 0$;
- **positive semi-definite** if $\boldsymbol{w}^\top M\boldsymbol{w} \geq 0, \quad \forall \boldsymbol{w}$;
- **indefinite** if M has both positive and negative eigenvalues
- **invertible** if no zero eigenvalue.

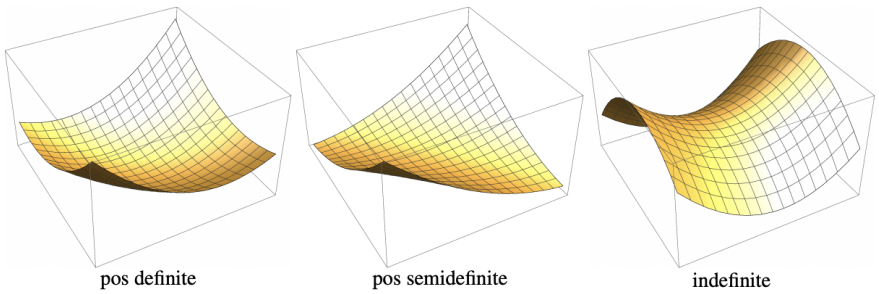

Figure 8.2: Positive/negative eigenvalues correspond to axes where the curvature goes up/down. Zero eigenvalues correspond to axes where the curvature is flat.

Question. What does this tell us about $\boldsymbol{x}^\top A^{-2}\boldsymbol{x}$?

Answer. The eigenvalues of A^{-2} are the inverse squares of the eigenvalues of A, so it cannot have a negative eigenvalue. It also cannot have a zero eigenvalue because then A^{-2} would not exist. Thus, A^{-2} must be positive definite if it exists.

Question. What does this tell us about the isosurfaces of $\boldsymbol{x}^\top M\boldsymbol{x}$ where M is PD?

Answer. The contour plot of M's quadratic form has ellipsoidal isosurfaces whose radii determined by the eigenvalues of $M^{-1/2}$, which are inverse square roots of the eigenvalues of M.

Remark. If M is PSD, then the isosurfaces are cylinders instead of ellipsoids. These cylinders have ellipsoidal cross sections spanning the directions with nonzero eigenvalues, but run in straight lines along the directions with zero eigenvalues.

8.1.2 Building a Quadratic

Let's suppose that we are given some eigenvalues and eigenvectors, we want to construct a matrix that has them as its eigenvalues and eigenvectors.

1. Choose n mutually orthogonal unit vectors from ^{n}: $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$.
2. Let $V = \begin{bmatrix}\boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n\end{bmatrix}$.
3. Then V is an **orthogonal matrix**, i.e. $V^{-1} = V^\top$.
4. Choose some radii λ_i and have

$$\Lambda = \begin{bmatrix}\lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n\end{bmatrix}.$$

Theorem 8.1.6 (Spectral Eigendecomposition)**.**

$$A = V\Lambda V^\top = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top.$$

Remark. $V^\top$ rotates the ellipsoid to be axis-aligned and Λ stretches the ellipsoid, V reverse the rotation done by $V^\top$.

5. A is the desired matrix.

8.2 Anisotropic Gaussians

Now we revisit the multivariate Gaussian distribution, with different variances along different directions.

$$X \sim \mathcal{N}(\mu, \Sigma) \qquad (\Sigma \text{ is the } \textbf{covariance matrix}.)$$

The PDF is given by

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu)^\top \Sigma^{-1}(\boldsymbol{x}-\mu)\right).$$

Definition 8.2.1 (Covariance)**.** Let X, Y be two random variables. Then the **covariance** is defined as

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])^\top] = \mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]^\top.$$

If X is a vector, then the **covariance matrix** is

$$\text{Var}(X) = \begin{bmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_d) \\ \text{Cov}(X_2, X_1) & \text{Var}(X_2) & & \text{Cov}(X_2, X_d) \\ \vdots & & \ddots & \vdots \\ \text{Cov}(X_d, X_1) & \text{Cov}(X_d, X_2) & \dots & \text{Var}(X_d) \end{bmatrix}.$$

For a Gaussian $X \sim \mathcal{N}(\mu, \Sigma)$, we can show that $\text{Var}(X) = \Sigma$.

Property 8.2.2.

$$X_i, X_j \text{ independent} \implies \text{Cov}(X_i, X_j) = 0. \qquad (\text{reverse is not true}).$$

Property 8.2.3.

$$\text{Cov}(X_i, X_j) = 0 \text{ AND multivariate normal distribution} \implies X_i, X_j \text{ independent}$$

Property 8.2.4.

$$\text{All features pairwise independent} \implies \text{Var}(X) \text{ is diagonal.}$$

Property 8.2.5.

$$\text{Var}(X) \text{ is diagonal AND joint normal} \iff \text{axis-aligned Gaussian; squared radii on diagon of } \Sigma = \text{Var}(X)$$
$$\iff f(\boldsymbol{x}) = \prod_{i=1}^{d} f(\boldsymbol{x}_i).$$

When the features are independent, we can write the multivariate Gaussian PDF as a product of univariate Gaussian PDFs. When they aren't, we do a change of coordinates to the eigenvector coordinate system, and write it as a product of univariate Gaussian PDFs in eigenvector coordinates.

Consider the eigendecomposition of the covariance matrix $\Sigma = V\Lambda V^\top$. The eigenvalues of Σ are variances along the eigenvectors. Then this implies that the eigenvalues of $\Sigma^{1/2}$ are the standard deviations, $\sqrt{\Sigma_{ii}} = \sigma_i$.

8.2.1 Maximum Likelihood Estimation for Anisotropic Gaussians

Given sample points $X_1, \ldots, X_n$ and classes $y_1, \ldots, y_n$, we want to find the best-fit Gaussians.

For QDA:

$$\hat{\Sigma}_C = \frac{1}{n_C} \sum_{\{i:y_i=C\}} (X_i - \hat{\mu}_C)(X_i - \hat{\mu}_C)^\top.$$

Choosing C that maximizes $f(X = x \mid Y = \mathrm{C})\pi_\mathrm{C}$ is equivalent to maximizing the quadratic discriminant function

$$Q_\mathrm{C}(x) = \ln\left((\sqrt{2\pi})^d f_\mathrm{C}(x)\pi_\mathrm{C}\right) = -\frac{1}{2}(x - \mu_\mathrm{C})^\top \Sigma_\mathrm{C}^{-1}(x - \mu_\mathrm{C}) - \frac{1}{2}\ln|\Sigma_\mathrm{C}| + \ln \pi_\mathrm{C}.$$

For LDA:

$$\hat{\Sigma} = \frac{1}{n}\sum_C \sum_{\{i:y_i=C\}} (X_i - \hat{\mu}_C)(X_i - \hat{\mu}_C)^\top.$$

$$Q_\mathrm{C}(\boldsymbol{x}) - Q_\mathrm{D}(\boldsymbol{x}) = \underbrace{(\mu_\mathrm{C} - \mu_\mathrm{D})^\top \Sigma^{-1}\boldsymbol{x}}_{w^\top \boldsymbol{x}} \underbrace{- \frac{\mu_\mathrm{C}^\top \Sigma^{-1}\mu_\mathrm{C} - \mu_\mathrm{D}^\top \Sigma^{-1}\mu_\mathrm{D}}{2} + \ln \pi_\mathrm{C} - \ln \pi_\mathrm{D}}_{+\alpha}$$

Choose class C that maximizes the linear discriminant function

$$\mu_\mathrm{C}^\top \Sigma^{-1}\boldsymbol{x} - \frac{1}{2}\mu_\mathrm{C}^\top \Sigma^{-1}\mu_\mathrm{C} + \ln \pi_\mathrm{C}.$$

8.2.2 Some Terms

Let X be a $n \times d$ **design matrix** of sample points. Each row is a sample point.

Definition 8.2.6 (Sample Covariance Matrix)**.** Let X be uniform distribution on sample points. The **sample covariance matrix** is

$$\mathrm{Var}(X) = \frac{1}{n}(X - \hat{X})^\top (X - \hat{X}),$$

where $\hat{X}$ is obtained by subtracting $\mu^\top$ from each row of X.

Definition 8.2.7 (Decorrelation)**.** **Decorrelating** $\hat{X}$ means applying rotation $Z = \hat{X}V$, where $\mathrm{Var}(X) = V\Lambda V^\top$.

9

Regression

Two main types of machine learning problems:

1. **Classification**: given point $\boldsymbol{x}$, predict class.
2. **Regression**: given poing $\boldsymbol{x}$, predict a numerical value.

In regression, we choose form of regression function $h(\boldsymbol{x}; p)$ with parameters p, where h is the **hypothesis**, and choose a cost function to optimize.

9.1 Least-Squares Linear Regression

Problem:

$$\boldsymbol{w}^*, \alpha^* = \arg\min_{\boldsymbol{w},\alpha} \sum_{i=1}^{n} (X_i \cdot \boldsymbol{w} + \alpha - y_i)^2,$$

which is equivalent to the following optimization problem

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \|X\boldsymbol{w} - \boldsymbol{y}\|^2.$$

We optimize by calculus:

$$\begin{aligned} \nabla_{\boldsymbol{w}} \|X\boldsymbol{w}-\boldsymbol{y}\|^2 &= \nabla_{\boldsymbol{w}\boldsymbol{w}^\top X^\top X\boldsymbol{w} - 2\boldsymbol{y}^\top X\boldsymbol{w} + \boldsymbol{y}^\top \boldsymbol{y}} \\ &= 2X^\top X\boldsymbol{w} - 2X^\top \boldsymbol{y} = 0 \\ &\implies X^\top X\boldsymbol{w} = X^\top \boldsymbol{y}. \end{aligned}$$

Here $X^\top X$ is always PSD. We use a linear solver to find

$$\boldsymbol{w}^* = \underbrace{(X^\top X)^{-1} X^\top}_{\text{pseudoinverse } X^\dagger} \boldsymbol{y},$$

so we never actually invert the matrix nor do we compute the pseudoinverse directly.
Advantages: Easy to compute; Unique and stable solution.
Disadvantages: Very sensitive to outliers because errors are squared; fails if $X^\top X$ is singular (multiple solutions).

9.2 Logistic Regression

Problem:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} L(X_i \cdot \boldsymbol{w}, y_i) = -\sum_{i=1}^{n} \Big(y_i \ln s(X_i \cdot \boldsymbol{w}) + (1 - y_i) \ln \left(1 - s(X_i \cdot \boldsymbol{w})\right) \Big).$$

Since the loss function is convex, we can solve by gradient descent. Note that

$$s'(\gamma) = \frac{\partial}{\partial \gamma} \frac{1}{e^{-\gamma}} = s(\gamma)(1 - s(\gamma)).$$

Let $s_i = s(X_i \cdot \boldsymbol{w})$. Then we have

$$\begin{aligned}
\nabla_w \mathcal{J} &= -\sum \left(\frac{y_i}{s_i} \nabla s_i - \frac{1 - y_i}{1 - s_i} \nabla s_i \right) \\
&= -\sum \left(\frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i} \right) s_i \left(1 - s_i\right) X_i \\
&= -\sum \left(y_i - s_i\right) X_i \\
&= -X^\top (y - s(Xw)).
\end{aligned}$$

where $s(Xw) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}$. **Gradient Descent Rule:** $\boldsymbol{w} \longleftarrow \boldsymbol{w} + \epsilon X^\top (\boldsymbol{y} - s(X\boldsymbol{w}))$.

Stochastic Gradient Descent: $\boldsymbol{w} \longleftarrow \boldsymbol{w} + \epsilon(y_i - s(X_i \cdot \boldsymbol{w}))X_i$.

Remark. Logistic regression always separates linearly separable points.

9.3 Least-Squares Polynomial Regression

Replace each X_i with feature vector $\Phi(X_i)$ with all terms of degree $0, \ldots, o$. For example,

$$\Phi(X_i) = \begin{bmatrix} X_{i1}^2 & X_{i1}X_{i2} & X_{i2}^2 & X_{i1} & X_{i2} & 1 \end{bmatrix}^\top,$$

where the 1 is the **fictitious dimension**.

9.4 Weighted Least-Squares Regression

Assign each sample point a weight ω_i and collect ω_i's in $n \times n$ diagonal matrix Ω. Greater ω_i implies that we need to work harder to minimize $|\hat{y}_i - y_i|^2$, where $\hat{y} = Xw$.

Goal: find $\boldsymbol{w}$ that minimizes

$$(X\boldsymbol{w} - \boldsymbol{y})^\top \Omega (X\boldsymbol{w} - \boldsymbol{y}) = \sum_{i=1}^{n} \omega_i (X_i \cdot \boldsymbol{w} - y_i)^2.$$

We can solve this by setting the gradient to zero and reduce to the following normal equations:

$$X^\top \Omega X \boldsymbol{w} = X^\top \Omega \boldsymbol{y}.$$

9.5 Newton's Method

Newton's method is an iterative optimization method for smooth function $\mathcal{J}(\boldsymbol{w})$. It is often much faster than gradient descent.

Idea: we are at some point $\boldsymbol{v}$. We approximate $\mathcal{J}(\boldsymbol{w})$ near $\boldsymbol{v}$ by quadratic function. Jump to its unique critical point and repeat until close to true solution.

Consider the Taylor series about $\boldsymbol{v}$:

$$\nabla\mathcal{J}(\boldsymbol{w}) = \nabla\mathcal{J}(\boldsymbol{v}) + (\nabla^2\mathcal{J}(\boldsymbol{v}))(\boldsymbol{w}-\boldsymbol{v}) + O(\|\boldsymbol{w}-\boldsymbol{v}\|^2),$$

where $\nabla^2\mathcal{J}(\boldsymbol{v})$ is the **Hessian matrix** of $\mathcal{J}$ of $\boldsymbol{v}$. We then find the critical point $\boldsymbol{w}$ by setting $\nabla\mathcal{J}(\boldsymbol{w}) = 0$:

$$\boldsymbol{w} = \boldsymbol{v} - (\nabla^2\mathcal{J}(\boldsymbol{v}))^{-1}\nabla\mathcal{J}(\boldsymbol{v}).$$

Remark. We would not want to compute the inverse of a matrix directly and instead we solve a linear system of equations for efficiency.

The **Newton's method** is as follows:

1. Pick starting point $\boldsymbol{w}$ (usually 0 works well).
2. Repeat until convergence:

$$\begin{aligned}\boldsymbol{e} &\longleftarrow (\nabla^2\mathcal{J}(\boldsymbol{w}))\boldsymbol{e} = -\nabla\mathcal{J}(\boldsymbol{w})\\ \boldsymbol{w} &\longleftarrow \boldsymbol{w} + \boldsymbol{e}.\end{aligned}$$

Remark. Newton's method doesn't know the difference between minima, maxima, saddle points. Starting point must be close enough to desired critical point. The closer $\mathcal{J}$ is to quadratic, the faster Newton's method tends to converge.

Disadvantages: computing the Hessian is expensive when we have high-dimensional weight spaces (not good for neural network). Doesn't work for most nonsmooth functions as well.

9.5.1 Logistic Regression (continued)

Recall that we have

$$\nabla_{\boldsymbol{w}}\mathcal{J} = -\sum_{i=1}^{n}(y_i - s_i)X_i = -X^\top(\boldsymbol{y}-\boldsymbol{s}).$$

Then using this result, we have

$$\nabla^2_{\boldsymbol{w}}\mathcal{J}(\boldsymbol{w}) = \sum_{i=1}^{n} s_i(1-s_i)X_iX_i^\top = X^\top\Omega X, \quad \text{where } \Omega = \begin{bmatrix} s_1(1-s_1) & 0 & \dots & 0 \\ 0 & s_2(1-s_2) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & s_n(1-s_n)\end{bmatrix}.$$

Then the Newton's method for logistic regression is as follows:

1. $\boldsymbol{w} \longleftarrow 0$
2. Repeat until convergence:

$$\boldsymbol{e} \longleftarrow (X^\top \Omega X)\boldsymbol{e} = X^\top(\boldsymbol{y} - \boldsymbol{s})\boldsymbol{w} \longleftarrow \boldsymbol{w} + \boldsymbol{e}.$$

This is an example of iteratively reweighted least squares.

9.6 LDA vs. Logistic Regression

Advantages of LDA:

- Stable for well-separated classes. Logistic regression is unstable.
- Works well for multiclass problems. Logistic regression needs modification (softmax regression).
- More accurate when classes are nearly normal, especially if n is small.

Advantages of Logistic Regression:

- More emphasis on decision boundary, always separates linearly separable points.
- More robust on some non-Gaussian distributions.
- Naturally fits labels between 0 and 1.

9.7 Receiver Operating Characteristics (ROC) Curves (test sets)

- A ROC curve is a way to evaluate our classifier after it is trained.
- Shows the rate of false positives vs. true positives for a range of settings.
- Use posterior probability threshold for GDA or logistic regression as a knob.

9.8 Statistical Justifications For Regression

Recall the four levels of machine learning: the **application**, **the model**, the **optimization problem**, and the **optimization algorithm**. Now take a step up to the second level, the model and see how some models lead to the optimization problems we've seen so far.

Typical model of reality:

- Sample points come from unknown probability distribution: $X_i \sim D$.
- y-values are sum of unknown, non-random function plus random noise:

$$\forall X_i, \quad y_i = g(X_i) + \epsilon_i, \qquad \text{where } \epsilon_i \sim D', \quad \text{and } \mathbb{E}[D'] = 0.$$

Here the function g is unknown, but it is not random. It represents a consistent relationship between X and $\boldsymbol{y}$ that we can estimate. The term ϵ represents measurement errors and other statistical error that could occur in measurement and it is **independent** of X.

Goal of regression: find h that estimates g.
Ideal approach: choose $h(\boldsymbol{x}) = \mathbb{E}_Y[Y \mid X = \boldsymbol{x}] = g(\boldsymbol{x}) + \mathbb{E}[\epsilon] = g(\boldsymbol{x})$.

9.8.1 Least-Squares Regression from Maximum Likelihood

Suppose $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, then $y_i \sim \mathcal{N}(g(X_i), \sigma^2)$. Recall that the log of normal PDF and log likelihood is:

$$\ln f(y_i) = -\frac{(y_i - \mu)^2}{2\sigma^2} - C \qquad (\mu = g(X_i))$$
$$\ell(g; X, \boldsymbol{y}) = \ln\left(\prod_{i=1}^{n} f(y_i)\right) = \ln\left(\sum_{i=1}^{n} f(y_i)\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - g(X_i))^2 - C.$$

Here we see that the maximum likelihood on parameter g is equivalent to estimating g by least-squares regression.

Remark. Recall from previous lecture that least-squares is very sensitive to outliers. It would not be a big deal if the error is truly normal. However, in real world, the distribution for outliers is often not normal. Hence, least-squares isn't a good choice sometimes.

9.8.2 Empirical Risk

Definition 9.8.1 (Risk)**.** The **risk** for hypothesis h is

$$R(h) = \mathbb{E}[L].$$

Discriminative model: X's distribution is unknown. How do we minimize risk?

We approximate the distribution by pretending that the sample points *are* the distribution. This is known as **empirical distribution**, which is the discrete uniform distribution over the sample points.

Definition 9.8.2 (Empirical Risk)**.** The **empirical risk** is the expected loss under such empirical distribution:

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), y_i).$$

9.8.3 Logistic Loss from Maximum Likelihood

Assume the actual probability for point X_i in the class is y_i; predicted probability is $h(X_i)$. Suppose we have β duplicate copies of X_i: $y_i\beta$ are in the class, $(1 - y_i)\beta$ are not. The likelihood is

$$\mathcal{L}(h; X, y) = \prod_{i=1}^{n} h(X_i)^{y_i\beta}(1 - h(X_i))^{(1-y_i)\beta}.$$

The log likelihood is

$$\begin{aligned}\ell(h) &= \ln \mathcal{L}(h) \\ &= \beta \sum_i \left(y_i \ln h(X_i) + (1 - y_i) \ln\left(1 - h(X_i)\right)\right) \\ &= -\beta \sum_i L(h(X_i), y_i).\end{aligned}$$

Hence, the logistic loss functions follows from the principle of maximum likelihood.

9.8.4 The Bias-Variance Decomposition

There are two sources of error in a hypothesis h:

1. **bias**: error due to inability of hypothesis h to fit g perfectly, e.g., fitting quadratic g with a linear h.
2. **variance**: error due to fitting random noise in data, e.g., fit linear g with a linear h, yet $h \neq g$.

Model: $X_i \sim D, \epsilon_i \sim D', y_i = g(X_i) + \epsilon_i$. Fit hypothesis h to $X, \boldsymbol{y}$. h is a random variable since its weights are random.

Consider arbitrary point $\boldsymbol{z} \in \mathbb{R}^d$ (not necessarily a sample point) and $\gamma = g(\boldsymbol{z}) + \epsilon$, where $\epsilon \sim D'$. $\boldsymbol{z}$ is *arbitrary*, whereas γ is *random* $\mathbb{E}[\gamma] = g(\boldsymbol{z})$ and $\text{Var}(\gamma) = \text{Var}(\epsilon)$.

When the loss is equal to squared error, the risk function is

$$\begin{aligned} R(h) &= \mathbb{E}[L(h(\boldsymbol{z}), \gamma)] \\ &= \mathbb{E}[(h(\boldsymbol{z}) - \gamma)^2] \\ &= \mathbb{E}[h(\boldsymbol{z})^2] + \mathbb{E}[\gamma^2] - 2\mathbb{E}[\gamma h(\boldsymbol{z})] \\ &= \text{Var}(h(\boldsymbol{z})) + \mathbb{E}[h(\boldsymbol{z})]^2 + \text{Var}(\gamma) + \mathbb{E}[\gamma]^2 - 2\mathbb{E}[\gamma]\mathbb{E}[h(\boldsymbol{z})] \\ &= (\mathbb{E}[h(\boldsymbol{z})] - \mathbb{E}[\gamma])^2 + \text{Var}(h(\boldsymbol{z})) + \text{Var}(\gamma) \\ &= \underbrace{(\mathbb{E}[h(\boldsymbol{z})] - g(\boldsymbol{z}))^2}_{\text{bias}^2 \text{ of method}} + \underbrace{\text{Var}(h(\boldsymbol{z}))}_{\text{variance of method}} + \underbrace{\text{Var}(\epsilon)}_{\text{irreducible error}} . \end{aligned}$$

This is called the **bias-variance decomposition** of the risk function.
Consequences:

- Underfitting means too much bias.
- Most overfitting are caused by too much variance.
- Training error reflects the bias but not variance; test error reflects both.
- For many distributions, variance $\to 0$ as $n \to \infty$.
- If h can fit g exactly, for many distributions bias $\to 0$ as $n \to \infty$.
- If h cannot fit g well, bias is large at most points.
- Adding a good feature reduces bias; adding a bad feature rarely increases it.
- Adding a feature usually increases variance.
- Irreducible error cannot be reduced.
- Noise in test set affects only $\text{Var}(\epsilon)$; noise in training set affects only bias and $\text{Var}(h)$.
- We cannot precisely measure bias and variance of real-world data, but we can test learning algorithms by choosing g and making synthetic data.

9.9 Ridge Regression

The problem is formulated as

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{J}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \|X\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{w}'\|^2,$$

where $\boldsymbol{w}'$ is $\boldsymbol{w}$ with bias component α replaced by zero and X has fictitious dimensions but we don't penalize α.

We add a **regularization** term, or a penalty term for shrinkage, to encourage small $\|\boldsymbol{w}'\|$ for the following reasons:

1. Guarantees positive definite normal equations and thus there always exists unique solution.

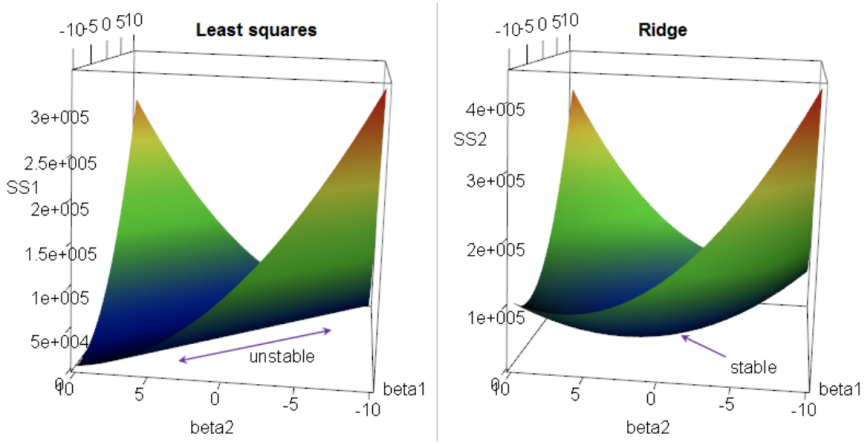

Figure 9.1: Cost function $\mathcal{J}$ with and without regularization. *Left.* Positive semidefinite quadratic form with infinite minima (ill-posed). *Right.* Positive definite quadratic form with one unique minimum (well-posed).

2. Reduces overfitting by reducing variance. (Why?)

Ridge regression works similarly to the regression problems we have done so far. We setting $\nabla\mathcal{J} = 0$ gives normal equations

$$(X^\top X + \lambda I')\boldsymbol{w} = X^\top \boldsymbol{y},$$

where I' is I with bottom right entry set to 0.

Algorithm: Solve for $\boldsymbol{w}$ and return $h(\boldsymbol{z}) = \boldsymbol{w}^\top \boldsymbol{z}$. Increasing λ would give smaller $\|\boldsymbol{w}'\|$.

Assuming data model $y = X\boldsymbol{v} + \boldsymbol{e}$ where $\boldsymbol{v}$ is the relationship we are looking for and $\boldsymbol{e}$ is the noise. Then the variance for ridge regression is

$$\text{Var}(\boldsymbol{z}^\top (X^\top X + \lambda I')^{-1} X^\top).$$

Remark. Note that as $\lambda \to \infty$, variance $\to 0$, but bias increases.

9.9.1 Bayesian Justification for Ridge Regression

Assign a prior probability on $\boldsymbol{w}'$: $\boldsymbol{w}' \sim \mathcal{N}(0, \sigma^2)$. Apply MLE to the posterior probability. Using Bayes' Theorem, we have

$$\begin{aligned} f(\boldsymbol{w} \mid X, \boldsymbol{y}) &= \frac{f(\boldsymbol{y} \mid X, \boldsymbol{w}) \times \text{prior } f(\boldsymbol{w}')}{f(\boldsymbol{y} \mid X)} \\ &= \frac{\mathcal{L}(\boldsymbol{w}) f(\boldsymbol{w}')}{f(\boldsymbol{y} \mid X)}. \end{aligned}$$

Then applying logarithm gives

$$\begin{aligned} \ln \mathcal{L}(\boldsymbol{w}) + \ln f(\boldsymbol{w}') - C &= -C_1 \|X\boldsymbol{w} - \boldsymbol{y}\|^2 - C_2 \|\boldsymbol{w}'\|^2 - C_3 \\ &\implies \min_{\boldsymbol{w}} \|X\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{w}'\|^2. \end{aligned}$$

9.10 Feature Subset Selection

All features increase variance, but not all features reduce bias.

Idea:

- Identify poorly predictive features, ignore them (set weight to zero).
- Less overfitting, smaller test errors.
- The second motivation is inference. Simpler the models convey interpretable wisdom.

This is useful in all classification and regression methods. However, sometimes it's hard because different features can partly encode same information and it is combinatorially hard to choose best feature subset.

Algorithm: Best subset selection. Try all $2^d - 1$ nonempty subset of features (train one classifier per subset). Choose the best classifier by (cross-)validation. This is slow in general.

Heuristic 1: Forward stepwise selection. Start with **null model** (0 features); repeatedly add best feature until validation errors start increasing (due to overfitting) instead of decreasing. At each outer iteration, inner loop tries every feature and chooses the best by validation. Requires training $O(d^2)$ models instead of $O(2^d)$.

Disadvantages: Won't find the best 2-feature model if neither of those features yields the best 1-feature model (that's why it's a heuristic).

Heuristic 2: Backward stepwise selection. Start with all d features; repeatedly remove features whose removal gives best reduction in validation error. Also trains $O(d^2)$ models.

9.11 LASSO

Least Absolute Shrinkage and Selection Operator, or **LASSO** is a regularized regression method similar to ridge regression, but has the advantage that if often naturally sets some of the weights to zero. The problem is formulated as below:

$$\boldsymbol{w}^* = \arg\min \|X\boldsymbol{w} - \boldsymbol{y}\|^2 + \lambda\|\boldsymbol{w}'\|_1, \qquad \text{where } \|\boldsymbol{w}'\|_1 = \sum_{i=1}^{d} |\boldsymbol{w}_i|.$$

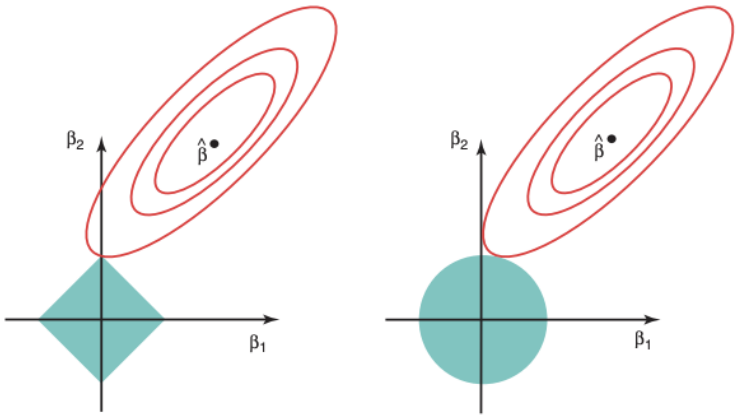

Figure 9.2: *Left.* Isocontours for Lasso (**cross-polytopes**). *Right.* Isocontours for ridge regression.

From the above graph, the solution lies at the intersection of the red isocontour and the blue region. For the Lasso example, weight w_1 gets set to zero because only the tip of the diamond is touched. However, it is also possible that a side of the diamond is touched by the red isosurface.

Remark. In higher dimensions, we could have several weights set to zero. For example, in 3D, if the red isosurface touches a sharp vertex of the cross-polytope, two of the three weights get set to zero. If it touches a sharp edge of the cross-polytope, one weight gets set to zero. If it touches a flat side of the cross-polytope, no weight is zero.