

Statistical Machine Learning

MAT 5314

Hanan Ather

Winter 2022

Contents

1	January 9, 2023	3
1.1	Introduction	3
1.2	Supervised learning	3
1.3	Empirical Risk Minimization	3
1.4	Manifold learning	4
2	January 11, 2023	6
2.1	OLS as ERM	6
2.2	Mathematical formalization of linear regression	6
2.3	Linear Algebra facts	6
2.4	Column Perspective	7
2.5	Spectral decomposition	7
3	January 16, 2023	9
3.1	Projection Prospective of OLS	9
3.2	Occam's razor approach	9
3.3	Regularized Least Square	9
3.4	Recursive Least Squares	9
4	January 18, 2022	11
4.1	Linear Learning Algorithms	11
4.2	Learning Algorithm	11
4.3	Bayes Optimal	11
4.4	Bias-variance tradeoff	12
5	January 23, 2023	13
5.1	Linear Discriminant Analysis	13
6	January 25, 2023	14
6.1	LDA continued	14
6.2	PCA	15
7	January 30, 2023	16
7.1	Loss functions for classification	16
8	February 1, 2023	17
8.1	Support Vector Machines	17
8.2	Non-separable Case	17
8.3	Primal optimization problem	17
9	February 6, 2023	19
9.1	Lagrange Multipliers	19

9.2 Karush-Kuhn-Tucker (KKT) theorem	19
10 February 8, 2023	20
10.1 Learning Theory	20
10.2 Probably Approximately Correct (PAC) Learning	20
11 February 15, 2023	22
12 February 27, 2023	25
13 March 1, 2023	27
14 March 6, 2023	28
14.1 Rademacher Complexity	28
15 March 8, 2023	31
15.1 Concentration Inequalities	31
15.2 Probabilistic inequalities	31
15.3 McDiarmid's Inequality	32
16 March 13, 2023	33
16.1 VC Theory	33
17 March 15, 2023	35
17.1 VC dimension continued	35
17.2 Growth function	36
18 March 20, 2023	38
18.1 Learning theoretic results	38
18.2 Introducing reproducing kernels	38
19 March 22, 2023	41
19.1 Functional Analysis Background	41
19.2 Reproducing kernel Hilbert spaces (RKHS)	41
20 March 27, 2023	43
20.1 Online learning	43
20.1.1 Mathematical Formulation	43
20.1.2 Advantages of Online Learning	43
20.2 Introduction to the perceptron	43
21 March 29, 2023	45
21.1 Perceptron Algorithm	45
21.2 Single-Layer Neural Networks	45
21.3 Multi-Layer Neural Networks	46
21.4 Functions	46
22 April 3, 2023	47
22.1 Optimization	47
22.2 Gradient descent	47
22.3 Batch gradient descent	47
22.4 Mini-batch gradient descent	47
22.5 Stochastic gradient descent	47
22.6 Advanced optimization techniques	48
23 April 5, 2023	49
23.1 Boosting	49
23.2 AdaBost	49

★ These notes were created during my review process to aid my own understanding and not written for the purpose of instruction. I originally wrote them only for myself, and they may contain typos and errors ^a. *No professor has verified or confirmed the accuracy of these notes.* With that said, I've decided to share these notes on the off chance they are helpful to anyone else.

^aAny corrections are greatly appreciated.

§1 January 9, 2023

§1.1 Introduction

Three main paradigms in modern Machine Learning¹:

- Supervised learning
- unsupervised learning
- semi-supervised learning

Supervised learning is historically the oldest machine learning problem and it has a well-developed theory, methods and well-defined canonical formulation. Supervised learning basically amounts to learning some function f via stochastic optimization.

Dataset + Algorithm \rightarrow Predictive Model

§1.2 Supervised learning

Formally in supervised learning we are seeking a mapping $f : X \rightarrow Y$. Where X is the input space and Y is an output space, we denote $x \in X$ as instances, $y \in Y$ as labels. In classification problems Y is a finite set, and in regression $Y \subseteq \mathbb{R}$. We obtain $z_i = (x_i, y_i) \sim p$ on $X \times Y$ labeled data which are generally referred to as samples or examples, made of up independent and identically distributed (i.i.d.) samples from a data generating distribution p .² Our objective is to “learn” f and we accomplish by utilizing the **Empirical Risk Minimization** framework via using some loss function denoted by $V(f, z)$.

We can think of *learning* the mapping $f : X \mapsto Y$ by searching over a set of candidate functions and finding the one that is most consistent with the training examples. More formally, we consider a particular class of function \mathcal{F} and choose a scalar-valued loss function denoted as $V(f, z)$ that measures the disagreement between our prediction $f(x_i)$ for some $f \in \mathcal{F}$ and true label y_i .

Examples of Loss Functions:

- Least squares loss $V(f, z) = (y - f(x))^2$
- Miss-classification (0-1) loss $V(f, z) = 0$ if sign is correct, otherwise 1
- Hinge loss (or logistic loss) $V(f, z) = \log(1 + e^{-yf(x)})$

§1.3 Empirical Risk Minimization

The ERM approach comes from empirical process theory (i.e, control theory). Our objective is to seek $f \in \mathcal{F}$ that minimizes **risk** (i.e., expected loss) over the data generating distribution p :

$$\mathcal{R}(f) = \mathbb{E}_p V(f, z)$$

This optimization problem is intractable because we do not have access to p or all all possible elements of p , and therefore we cannot evaluate the expectation without making some unrealistically

¹Additionally we also have reinforcement learning which is a very important paradigm and can be viewed as a hybrid between supervised and unsupervised learning

²In literature its also common to denote this distribution as D ; i.e., $(x_i, y_i) \sim D$ for all i

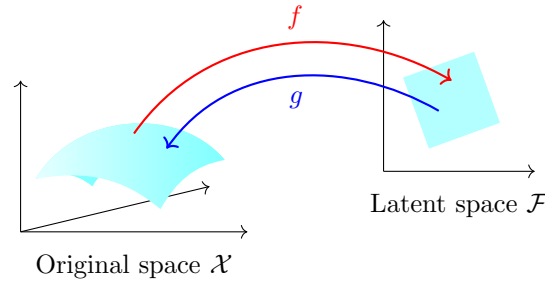
strong assumptions about the structure of D , V or f . However, under i.i.d. assumption we can approximate the risk (expected loss) by averaging loss over the available training data. We use sample average as a stand-in estimator for the true risk, and minimize the **training error** in order to minimize the true risk $\mathcal{R}(f)$:

$$\hat{\mathcal{R}}(f) = \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n V(f, z_i)$$

. In the above equation, we only optimize the loss over the available training examples, and hope this is a good proxy objective over the actual objective.

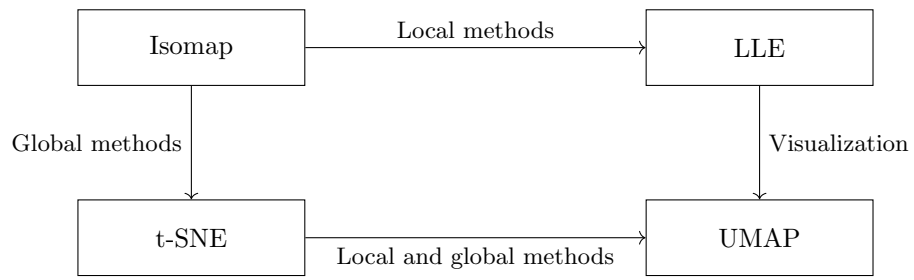
§1.4 Manifold learning

Manifold learning is a type of unsupervised machine learning technique that focuses on discovering the underlying low-dimensional structure or manifold within high-dimensional data. The main goal is to reduce the dimensionality while preserving the intrinsic structure and relationships in the data. This is particularly useful for visualization, data compression, and noise reduction.



A **manifold** is a continuous, smooth, and low-dimensional surface embedded in a high-dimensional space. Real-world data often lies on or near such manifolds, which makes it possible to represent complex data with fewer dimensions. Unlike linear techniques such as PCA, manifold learning algorithms can capture and preserve nonlinear relationships in the data. Manifold learning methods can be classified into two categories based on the type of structure they preserve. *Local methods* focus on preserving neighborhood relationships, while *global methods* try to maintain the overall structure of the data. Some popular manifold learning algorithms include:

- Isomap: It extends the idea of multidimensional scaling (MDS) by using geodesic distances on the manifold. It preserves the global structure of the data.
- Locally Linear Embedding (LLE): LLE focuses on preserving local relationships by reconstructing each data point as a linear combination of its neighbors. It then maps the data to a lower-dimensional space while maintaining these relationships.
- t-Distributed Stochastic Neighbor Embedding (t-SNE): t-SNE is designed for visualizing high-dimensional data. It constructs a probability distribution over pairs of data points in the high-dimensional space and then finds a low-dimensional representation that minimizes the divergence between the two distributions.
- Uniform Manifold Approximation and Projection (UMAP): UMAP is a more recent algorithm that balances the preservation of local and global structures. It constructs a graph representation of the data and optimizes the layout of the low-dimensional representation to match the graph's structure.



§2 January 11, 2023

§2.1 OLS as ERM

Regression is class of statistical/computational methods that seek to estimate relationships between a dependent (target) variable and independent variables (instances/features/inputs). There are many different techniques used for regression: linear regression, support vector machines, neural networks, K-nearest neighbours, random forest. The differences among the methods boils down to two factors: how we choose represent the function/hypothesis $f : X \rightarrow Y$ and what *loss function* we choose to measure the 'goodness' of our model. In linear regression we assume the f function we are seeking belongs to the class of linear functions, which we denoted as: $\mathcal{F} = \{\mathbf{w} \cdot \mathbf{x} | \mathbf{w} \in \mathbb{R}^d\}$ or in an expanded form:

$$f(x) = w_0 + w_1x_1 + \dots + w_dx_d$$

Now given a data set of input variables \mathbf{x} and labels y , how can we reasonably find the weights \mathbf{w} ? One reasonable approach is to find \mathbf{w} such that $\mathbf{w} \cdot \mathbf{x}$ is close to y . There are many different loss functions to find the weight vector $w \in \mathbb{R}^d$. The **least squares or L2 loss** is $V(f, z) = (y - f(x))^2$. In this setting we let $X = \mathbb{R}^d$ and $Y \subset \mathbb{R}$.

§2.2 Mathematical formalization of linear regression

We store the data into two matrices: \mathcal{Y} is an $m \times 1$ column vector containing *labels*, and \mathcal{X} is a $m \times d$ matrix containing *inputs* or *features*. We perform **empirical risk minimization** for the **squared loss** $V(f, z) := (y - f(x))^2$, with $z = (x, y)$. The *optimization* problem is defined as

$$\min_w \sum_{i=1}^m (y_i - w \cdot x_i)^2 \quad \text{or} \quad \min_{\text{argmin } w} \|\mathcal{Y} - \mathcal{X}w\|^2$$

We define an **objective function** on \mathbb{R}^d

$$J(w) = \|\mathcal{Y} - \mathcal{X}w\|^2 \tag{1}$$

$$\begin{aligned} J(w) &= (\mathcal{Y} - \mathcal{X}w)^T (\mathcal{Y} - \mathcal{X}w) \\ &= \mathcal{Y}^T \mathcal{Y} - 2w^T \mathcal{X}^T \mathcal{Y} + w^T \mathcal{X}^T \mathcal{X} w \end{aligned}$$

By differentiation we find that the gradient of the smooth function $J(w)$ is zero iff the matrix equations is satisfied

$$(\mathcal{X}^T \mathcal{X})w = \mathcal{X}^T \mathcal{Y}$$

How do we know there exists a minimizer?

The w which minimizes (1) \Leftrightarrow solves $Aw = b$ where $b = \mathcal{X}^T \mathcal{Y}$ and $A = \mathcal{X}^T \mathcal{X}$. Let $\mathcal{M} = \{w | Aw = b\}$ be the set of minimizers (1) and let $\mathcal{N} = \ker A \subset \mathbb{R}^d$. Since $A = \mathcal{X}^T \mathcal{X}$ is a symmetric matrix by construction, we know from linear algebra that $\mathcal{N} \perp \text{Col } A$. Therefore, $b = \mathcal{X}^T \mathcal{Y} \in \text{Col}(A)$, which implies there exists at least one minimizer to (1) or equivalently, $\mathcal{M} \neq \emptyset$ ³.

§2.3 Linear Algebra facts

Properties of Gram matrix:

- $A = \mathcal{X}^T \mathcal{X}$ is called the **gram matrix** of X , its a matrix of inner products of d columns of X . These columns are called column vectors in \mathbb{R}^m , and we denote them x_1, \dots, x_d .
- A is a *symmetric* matrix (i.e., $A^T = A$). This implies that $\text{null}(A) \perp \text{Col}(A)$ and there exists an orthonormal basis. ($u^T A w = u^T A^T w = (Au)^T w = Au \cdot w$)

³When $A = \mathcal{X}^T \mathcal{X}$ is *invertible* there is a unique solution, otherwise, there are infinitely many solutions

- $A = \sum_i x_i x_i^T$ is the sum of rank 1-matrices.
- A is **semi-definite** ($u^T A u \geq 0$)
- Nullspace of A coincides with null space of X ; $\text{Null}(X) = \text{Null}(A) = \{v \in \mathbb{R}^d : Av = 0\}$.
 $X^T X$ is invertible $\iff X$ is invertible $\iff X$ has a null space of dimension zero \iff
all columns of X are independent
- The *inverse* of non-square matrices is not defined. The **left inverse** is given by $X^\dagger := (X^T X)^{-1} X^T$

When solving the equation $Aw = b$ there are three possibilities:

- (1) There exists a unique solution if A is invertible
- (2) infinitely many solutions if $b \perp \text{null}(A)$, or equivalently $b \in \text{Col}(A)$
- (3) No solutions when $b \not\perp \text{null}(A)$

How do we know there always exists a solution to OLS? Since in the OLS case $b = X^T y$ the condition $b \perp \text{null}(A)$ is always satisfied.⁴, the third case never occurs for us. Indeed, $(X^T \mathcal{Y})^T v = \mathcal{Y}^T \mathcal{X} v$ and this is non-zero when $v \in \mathcal{N}$. This means we always have one solution.

§2.4 Column Perspective

$\mathcal{X}w$ is the linear combination of d vectors $\mathbf{x}_1, \dots, \mathbf{x}_d$ using the weights w_1, \dots, w_d . Our original optimization problem was to find w such that $\mathcal{X}w$ is as close to \mathcal{Y} in squared norm ($\|\mathcal{Y} - \mathcal{X}w\|^2$). In other words, we are looking weights w such that the point $\mathcal{X}w \in S = \text{span}\{\mathbf{x}_1, \dots, \mathbf{x}_d\}$ is closest to \mathcal{Y} . The unique in S that is closest to \mathcal{Y} is the projection of \mathcal{Y} on to the space S . *Is there a unique weighting w of $\mathbf{x}_1, \dots, \mathbf{x}_d$?* Whether or not if there is a unique weighting of w of $\mathbf{x}_1, \dots, \mathbf{x}_d$ depends on if the vectors $\mathbf{x}_1, \dots, \mathbf{x}_d$ are linearly independent. If they are not linearly independent, there are infinitely many different weightings w that will give the same point $p = \mathcal{X}w$, and these w just differ by the elements of \mathcal{N} .

§2.5 Spectral decomposition

We use the spectral decomposition to *understand* solutions to OLS when A is not invertible.

★ Since A is a real-symmetric matrix, there exists an orthonormal basis $\{u_1, \dots, u_d\}$ for \mathbb{R}^d consisting of eigenvectors of A ^a

^aThis is known as the “spectral theorem for symmetric matrices”

- We denote the *basis eigenvectors* as u_1, u_2, \dots for A ; where each eigenvector u_i is associated to λ_i ; (i.e., $Au_i = \lambda_i u_i$) real eigenvalues $\lambda_i \geq 0$, indexed in ascending order
- Since A is positive semi-definite, all $\lambda_i \geq 0$; $u^T A u \geq 0$ for all $u \in \mathbb{R}^d$, so in particular $u_i^T A u_i \geq 0$ but this is λ_i .
- \mathcal{N} is span span of u_i with $\lambda_i = 0$, let $r = \dim(\mathcal{N})$.
- In terms of basis $\{u_i\}$, write $w = \sum \alpha_i u_i$ and $b = \sum \beta_i u_i$.
- Matrix equation $Aw = b$ can be re-written in terms of eigenvalues and eigen vectors

$$\sum_{i>r} \alpha_i \lambda_i u_i = \sum_{i>0} \beta_i u_i$$

- there exists a unique solution if and only if $\beta_1, \dots, \beta_r = 0$ (i.e., $b \perp \mathcal{N}$).

⁴Thus we always have at least one solution to OLS

- And $w = \sum \alpha_i u_i$ if and only if $\alpha_i = \frac{\beta_i}{\lambda_i}, \forall i > r$
- $\alpha_1, \dots, \alpha_r$ are arbitrary due to the fact we that we have a degree of freedom for every $\lambda_1, \dots, \lambda_r = 0$.

Since A is a *real symmetric* matrix there exists an orthonormal basis $\{u_1, \dots, u_d\}$ for \mathbb{R}^d consisting of eigenvectors.⁵

⁵This is known as the spectral theorem for symmetric matrices

§3 January 16, 2023

§3.1 Projection Prospective of OLS

Now we view our matrix X in terms of its columns x_1, \dots, x_d . Multiplying a $\mathbf{X}\mathbf{w} = \sum_{i=1}^d w_i x_i$ is a linear combination of the vectors x_1, \dots, x_d using weights w_1, \dots, w_d . Our objective is to minimize $\|y - \mathbf{X}\mathbf{w}\|^2$. In other words, we are seeking the $w \in \mathbb{R}^d$ such that $\mathbf{X}\mathbf{w}$ is as close as possible to y in L^2 norm. Let $e = y - \mathbf{X}\mathbf{w}$; now \mathbf{w} is a solution if and only if $e^T \mathbf{X}u = 0, \forall u \in \mathbb{R}^d$ ⁶. This amounts to $x^T \underbrace{(y - \mathbf{X}\mathbf{w})}_{=e} = 0$, i.e., $x^T \mathbf{X}\mathbf{w} = X^T y$ as before.

§3.2 Occam's razor approach

If there are infinitely many solutions, one approach is to choose the one with min norm. This is a form of Occam's razor: "Suppose there exist two explanations for an occurrence. In this case the simpler one is usually better". Since our solutions are of the form $w = \alpha_i u_i + \sum_{i>1} (\beta_i \lambda_i) u_i$, the solution with minimum norm is

$$w = \sum_{i>1} \frac{\beta_i}{\lambda_i} u_i,$$

obtained by letting $\alpha_1 = 0$

§3.3 Regularized Least Square

Our ultimate goal in machine learning is to seek functions that *generalize* to all $(x, y) \sim p$. Different functions could achieve the same sample loss, but their generalization outside the training data could vary. In **regularized** least squares (RLS) we modify the objective function with a penalty term to control for "complexity"

$$\min_w \| \mathcal{Y} - \mathcal{X}w \|^2 + \gamma \| w \|^2$$

where $\gamma \geq 0$ is a hyper-parameter that specifies how much the norm of w matters.

★ The regularized objective function has a minimum when

$$(\mathcal{X}^T \mathcal{X} + \gamma I)w = \mathcal{X}^T \mathcal{Y}$$

And if $\gamma > 0$ then $\mathcal{X}^T \mathcal{X} + \gamma I$ is guaranteed to be invertible since all eigenvalues $\lambda_i + \gamma > 0$

§3.4 Recursive Least Squares

Recursive least squares (RecLS) is *online* version of OLS where we update the least square coefficients based on the arrival of new data. Its a well-known *adaptive filtering* algorithm and it has connections to Kalman filter. We assume that we have n data at time n .

$$\mathcal{X}_{(n)} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix}, \mathcal{Y}_{(n)} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (2)$$

⁶the vector e is orthogonal to any vector in the span of column space of matrix X , i.e., $e^T x = 0$

Now we assume that a new observation comes in at time $t = n + 1$ so our new data matrix is

$$\mathcal{X}_{(n+1)} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \\ x_{n+1}^T \end{bmatrix}, \mathcal{Y}_{(n+1)} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{bmatrix} \quad (3)$$

We would now like to compute

$$w_{n+1} = (\mathcal{X}_{(n+1)}^T \mathcal{X}_{(n+1)})^{-1} \mathcal{X}_{(n+1)}^T \mathcal{Y}_{(n+1)}$$

Should we compute this w_{n+1} from scratch? or can we somehow leverage previous computations to compute the new w_{n+1} . To simply we define new notation:

$$C := \mathcal{X}_{(n+1)} = \begin{bmatrix} C_0 \\ C_1 \end{bmatrix} \quad \text{and} \quad b := \mathcal{Y}_{(n+1)} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

Now we let

$$w_{n+1} = \underbrace{(\mathcal{X}_{(n+1)}^T \mathcal{X}_{(n+1)})^{-1}}_{:=P_{n+1}} \underbrace{\mathcal{X}_{(n+1)}^T}_{=C^T} \underbrace{\mathcal{Y}_{(n+1)}}_{=b} = P_{n+1} C^T b \quad (4)$$

Likewise,

$$w_n = \underbrace{(\mathcal{X}_{(n)}^T \mathcal{X}_{(n)})^{-1}}_{:=P_n} \underbrace{\mathcal{X}_{(n)}^T}_{=C_0^T} \underbrace{\mathcal{Y}_{(n)}}_{=b_0^T} = P_n C_0^T b_0 \iff P_n^{-1} w_n = C_0^T b_0^T \quad (5)$$

Now a fact about block matrices is:

$$C^T C = C_0^T C_0 + C_1^T C_1^T \quad \text{and} \quad C^T b = C_0^T b_0^T + C_1^T b_1^T \quad (6)$$

$$\text{Therefore, } w_{n+1} = P_{n+1} C^T b \iff w_{n+1} = P_{n+1} [\underbrace{C_0^T b_0^T}_{=P_n^{-1} w_n} + C_1^T b_1^T],$$

$$\implies w_{n+1} = P_{n+1} [P_n^{-1} w_n + C_1^T b_1^T]$$

$$\text{And since } P_{n+1} = (C^T C)^{-1} \iff P_{n+1}^{-1} = C^T C \iff P_{n+1}^{-1} = \underbrace{C_0^T C_0 + C_1^T C_1^T}_{P_n^{-1}} \iff P_{n+1}^{-1} =$$

$P_n^{-1} + C_1^T C_1^T$, we have

$$P_n^{-1} = P_{n+1}^{-1} - C_1^T C_1^T$$

Combing the facts we get,

$$w_{n+1} = P_{n+1} [(P_{n+1}^{-1} - C_1^T C_1^T) w_n + C_1^T b_1^T] = w_n + P_{n+1} C_1^T (b_1 - C_1 w_n)$$

§4 January 18, 2022

§4.1 Linear Learning Algorithms

In this basic learning setting, we consider $x \in X = \mathbb{R}^d$ and aim to find a predictor for y in a space of linear (or affine) functions. We can differentiate between two scenarios:

1. Regression, where $Y \subset \mathbb{R}$ is an interval.
2. Classification, where $Y = \{-1, +1\}$.

For regression, we saw Ordinary Least Squares (OLS) with squared loss. For classification, we will see Support Vector Machines (SVMs) with hinge loss. We assume that $(x, y) \sim p$, where p is a distribution on $\mathbb{R}^d \times Y$ such that y is linearly related to x . We search for a predictor $f : x \mapsto y$ in the space \mathcal{F} of linear functions or the sign of linear functions, depending on the case.

For case (1), an example is $y = f_0(x) + \epsilon$, where $f_0 \in \mathcal{F}$, and ϵ has zero mean, representing the noise, typically Gaussian, i.e., $N(0, \sigma^2)$. For case (2), the noise probability decreases with the distance of x from a true underlying linear separator. One way this can happen is if $y = \text{sgn}(f_0(x) + \epsilon)$, where the linear separator is the hyperplane $H = \{f_0(x) = 0\}$. Adding noise ϵ flips the sign of y if and only if $|\epsilon| > |f_0(x)|$ with opposing sign.

§4.2 Learning Algorithm

What is learning algorithm?

A **supervised learning algorithm** A is a way of choosing $f \in \mathcal{F}$ based on data $\bar{z} \in Z^n$. We can view the learning algorithm as a map $A : Z^n \rightarrow \mathcal{F}$, where $A : \bar{z} = (z_1, \dots, z_n) \mapsto f_{\bar{z}, A}$. And evaluate learning algorithms, A , by how much they can minimize $\mathcal{R}(f)$.

Definition 4.1 (Bayes Error, Best in class) There are essentially three levels at which we can view our function f at: f_* , $f_{\mathcal{F}}$, $f_{\mathcal{F}, nA} = A(\bar{z})$.

- f_* : the function that minimizes the risk $\mathcal{R}(f) = \mathbb{E}_p V(f, z)$ over the class of all any measurable function over the distribution P . And $\mathcal{R}(f_*)$ is the **Bayes error**.
- $f_{\mathcal{F}}$: the function that minimizes risk $\mathcal{R}(f)$ over some *restricted class* \mathcal{F} of measurable functions for the distribution P . **A only operates within \mathcal{F} .**
- $f_{\mathcal{F}, nA} = A(\bar{z})$: the function produced by the algorithm A , on sample $\bar{z} = (z_1, \dots, z_n)$ from P^n ^a

^ain fact this is an abuse of notation and $f_{\mathcal{F}, \bar{z}, A}$ would be more precise since the algorithm sees a specific sample \bar{z} . However, the notation is emphasizing the fact that the function depends on the algorithm A handling a n -sample from P .

- **generalization error** $= \mathcal{R}(f_{\mathcal{F}, nA}) - \mathcal{R}(f_*)$
- **estimation error** $= \mathcal{R}(f_{\mathcal{F}, nA}) - \mathcal{R}(f_{\mathcal{F}})$
- **approximation error** $= \mathcal{R}(f_{\mathcal{F}}) - \mathcal{R}(f_*)$

$$\text{generalization error} = \text{estimation error} + \text{approximation error}$$

§4.3 Bayes Optimal

Proposition 4.2. Given $V(f, z) = (f(x) - y)^2$, $f_* = \mathbb{E}_p[y|x]$ is the **Bayes optimal**.

Proof. Given the squared error loss function $V(f, z) = (f(x) - y)^2$, we want to find the Bayes optimal function f_* . The Bayes optimal function is the function that minimizes the expected loss with respect to the data-generating distribution $p(x, y)$. It can be written as:

$$\begin{aligned}
f_* &= \arg \min_{f \in F} \mathbb{E}_{p(x,y)} [V(f, z)] \\
&= \arg \min_{f \in F} \mathbb{E}_{p(x,y)} [(f(x) - y)^2]
\end{aligned}$$

Let's condition on $X = x$:

$$\begin{aligned}
\mathbb{E} [(f(x) - y)^2 \mid X = x] &= \mathbb{E} [(f(x) - \mathbb{E}_p[y \mid x] + \mathbb{E}_p[y \mid x] - y)^2 \mid X = x] \\
&= \mathbb{E} [(f(x) - \mathbb{E}_p[y \mid x])^2 \mid X = x] + \mathbb{E} [(\mathbb{E}_p[y \mid x] - y)^2 \mid X = x] \\
&= \mathbb{E} [(f(x) - \mathbb{E}_p[y \mid x])^2 \mid X = x] + [\mathbb{E}_p[y \mid x] - y]^2
\end{aligned}$$

The expression is minimized when $f(x) = \mathbb{E}_p[y \mid x]$. Therefore, the Bayes optimal function is $f_* = \mathbb{E}_p[y \mid x]$. □

§4.4 Bias–variance tradeoff

The bias-variance tradeoff is an important concept in statistical learning theory, describing the balance between model complexity and generalization. It arises when estimating the expected test error for a given learning algorithm. The expected test error can be decomposed into three parts: bias, variance, and irreducible error.

Consider a learning algorithm that estimates a target function $f(x)$ using the hypothesis $h(x)$. Given a loss function $L(f(x), h(x))$, the expected test error can be written as:

$$\mathbb{E} [L(f(x), h(x))]$$

For simplicity, we will consider the squared error loss function:

$$L(f(x), h(x)) = (f(x) - h(x))^2$$

The expected test error can then be decomposed into bias, variance, and irreducible error as follows:

$$\begin{aligned}
\mathbb{E} [(f(x) - h(x))^2] &= \mathbb{E} [(f(x) - \mathbb{E}[h(x)] + \mathbb{E}[h(x)] - h(x))^2] \\
&= \mathbb{E} [(f(x) - \mathbb{E}[h(x)])^2] \\
&\quad + 2\mathbb{E} [(f(x) - \mathbb{E}[h(x)]) (\mathbb{E}[h(x)] - h(x))] \\
&\quad + \mathbb{E} [(\mathbb{E}[h(x)] - h(x))^2] \\
&= (f(x) - \mathbb{E}[h(x)])^2 + 0 + \mathbb{E} [(\mathbb{E}[h(x)] - h(x))^2] \\
&= \text{Bias}^2[h(x)] + \text{Var}[h(x)]
\end{aligned} \tag{7}$$

In the above decomposition, the first term, $\text{Bias}^2[h(x)]$, represents the squared bias, which quantifies the average error introduced by approximating the true function $f(x)$ by the expected hypothesis $\mathbb{E}[h(x)]$. The second term, $\text{Var}[h(x)]$, represents the variance, which quantifies the expected deviation of the hypothesis $h(x)$ from its expected value $\mathbb{E}[h(x)]$.

The bias-variance tradeoff states that as model complexity increases, the bias typically decreases while the variance increases. A model with high complexity might fit the training data very well, but it is also more likely to overfit, resulting in poor generalization to new data. On the other hand, a simpler model might not fit the training data as well, but it is more likely to generalize well to new data. The goal is to find the right balance between bias and variance that minimizes the expected test error.

§5 January 23, 2023

§5.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique used in machine learning and statistics. It aims to find the linear combinations of features that best separate two or more classes. LDA can be used for classification, dimensionality reduction, or feature extraction. It assumes that the data follows a Gaussian distribution and that the covariance matrices of the classes are equal.

Here's a step-by-step explanation of LDA

1. Compute the class means: Calculate the mean vector for each class in the dataset.
2. Compute the overall mean: Calculate the mean vector of the entire dataset, where $\bar{\mu}$ is the overall mean, n is the total number of samples, and \mathbf{y}_i is the i -th sample.
3. Compute the within-class scatter matrix (S_W): This matrix represents the scatter of the samples within each class.
4. Compute the between-class scatter matrix (S_B): This matrix represents the scatter of the class means with respect to the overall mean.
5. Compute the optimal projection (\mathbf{W}): The goal is to find a projection matrix \mathbf{W} that maximizes the ratio of the determinant of the between-class scatter matrix to the determinant of the within-class scatter matrix. The optimal projection matrix \mathbf{W} is formed by selecting the eigenvectors corresponding to the largest eigenvalues of the matrix $\mathbf{S}_W^{-1}\mathbf{S}_B$. The number of eigenvectors selected depends on the desired dimensionality of the reduced space.
6. Project the data: Transform the original data using the projection matrix \mathbf{W} .

Suppose we have 2 classes and d -dimensional samples, $\mathbf{x}_1, \dots, \mathbf{x}_n$ where n_1 samples come from first class and n_2 samples come from second class. Consider a project on a line, where we let the direction of the line be given by a vector \mathbf{v} . The dot product of $\mathbf{v}^t \mathbf{x}_i$ is the distance of projection of \mathbf{x}_i from the origin. Therefore, $\mathbf{v}^t \mathbf{x}_i$ is the projection of \mathbf{x}_i into a one dimensional subspace. Let $\tilde{\mu}_1$ and $\tilde{\mu}_2$ be the means of the projections of classes 1 and 2, and let μ_1 and μ_2 be the means of the classes 1 and 2.

We define **scatter** as

$$s = \sum_{i=1}^n (z_i - \mu_z)^2$$

which measures the spread of the data around the mean just as variance, however, its just on a different scale. Fisher's idea for LDA was to normalize $\tilde{\mu}_1 - \tilde{\mu}_2$ by scatter. Let $y_i = \mathbf{v}^t \mathbf{x}_i$, the scatter for the projected samples of class 1 is

$$\tilde{s}_1^2 = \sum_{y_i \in \text{Class 1}} (y_i - \tilde{\mu}_1)^2$$

and scatter for projected samples of class 2 is

$$\tilde{s}_2^2 = \sum_{y_i \in \text{Class 2}} (y_i - \tilde{\mu}_2)^2$$

Therefore, we want to project on a line \mathbf{v} which maximizes the following function:

$$J(\mathbf{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

§6 January 25, 2023

§6.1 LDA continued

The core idea behind LDA is that we want to project the data such that the points of two classes are “maximally separated”. Recall that, we denote **projected samples** as $y_i = v^t x_i$. And we compute the scatter for project samples of class 1 and 2.

★ Fisher linear discriminant is to project on a line in the direction of v which maximizes

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

If we find v which makes $J(v)$ large, we are guaranteed that the classes are well separated.

Definition 6.1 (LDA Objective Function) The objective function of Fisher Linear Discriminant can be written as:

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{v^t S_B v}{v^T S_W v}$$

Note the invariance of $J(v)$ to re-scaling v so we can instead restate the optimization as:

$$\max v^t S_B v \quad \text{subject to} \quad v^T S_W v = 1$$

1. Calculate the mean of each class, denoted as μ_i for class i , and the overall mean μ .
2. Compute the within-class scatter matrix S_W and the between-class scatter matrix S_B :

$$S_W = \sum_{i=1}^k \sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T,$$

$$S_B = \sum_{i=1}^k N_i (\mu_i - \mu)(\mu_i - \mu)^T,$$

where X_i represents the samples in class i , and N_i is the number of samples in class i .

3. Find the linear transformation matrix W that maximizes the Fisher’s criterion $J(W)$:

$$J(W) = \frac{\det(W^T S_B W)}{\det(W^T S_W W)}.$$

This is achieved by solving the generalized eigenvalue problem:

$$S_W^{-1} S_B w_i = \lambda_i w_i,$$

where w_i and λ_i are the eigenvectors and eigenvalues, respectively.

4. Choose the eigenvectors corresponding to the r largest eigenvalues as the columns of the transformation matrix W . This will reduce the feature space dimension from d to r . Typically, $r < k$.
5. Project the original data samples onto the new feature space using the transformation matrix W . The transformed data can be used for classification or further processing.

LDA has since been extended to handle multiple classes and has found applications in various fields such as pattern recognition, machine learning, and computer vision. It has also served as the basis for other dimensionality reduction techniques, such as Quadratic Discriminant Analysis (QDA), which considers non-linear boundaries between classes.

§6.2 PCA

1. Let \mathbf{X} be the data matrix of size $n \times d$, where n is the number of samples and d is the number of features.

2. Compute the mean vector $\boldsymbol{\mu}$ of the data:

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad (8)$$

3. Center the data matrix \mathbf{X} by subtracting the mean vector $\boldsymbol{\mu}$:

$$\mathbf{X}_{\text{centered}} = \mathbf{X} - \boldsymbol{\mu} \quad (9)$$

4. Calculate the covariance matrix $\boldsymbol{\Sigma}$ of the centered data:

$$\boldsymbol{\Sigma} = \frac{1}{n-1} \mathbf{X}_{\text{centered}}^\top \mathbf{X}_{\text{centered}} \quad (10)$$

5. Compute the eigenvectors \mathbf{v}_i and eigenvalues λ_i of the covariance matrix $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma} \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad (11)$$

6. Sort the eigenvectors and their corresponding eigenvalues in descending order of eigenvalues.

7. Select the first k eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ to form the projection matrix \mathbf{W} :

$$\mathbf{W} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_k] \quad (12)$$

8. Project the centered data matrix $\mathbf{X}_{\text{centered}}$ onto the lower-dimensional space using the projection matrix \mathbf{W} :

$$\mathbf{X}_{\text{PCA}} = \mathbf{X}_{\text{centered}} \mathbf{W} \quad (13)$$

§7 January 30, 2023

§7.1 Loss functions for classification

In machine learning, a loss function, also known as a cost function or objective function, is a mathematical function that measures the discrepancy between the predicted output and the actual output (or target) for a given input. It is used during the training process to evaluate the performance of a model and adjust its parameters to minimize the loss. Loss functions are crucial because they guide the optimization process and determine how well a model learns from the data. Loss functions can be broadly categorized into two types:

- **Regression loss functions:** Used when predicting continuous values.
- **Classification loss functions:** Used when predicting categorical values or class labels.

Thus far we have considered regression, now we study classification. In classification we are studying *discrete value-functions*, and consider the case where $Y = \{-1, +1\}$ ⁷.

- the square loss $V(w, y) = (w - y)^2 = (1 - wy)^2$
- hinge loss $V(w, y) = \max\{1 - wy, 0\} =: |1 - wy|_+$
- logistic loss $V(w, y) = (\ln 2)^{-1} \ln(1 + e^{-wy})$

Looking for $\min_{f \in \mathcal{F}} \sum_{i=1}^n V(f, z_i)$ is an intractable combinatorial problem for 0 – 1 loss⁸. Is this because we would have to consider all possible signs of our data points?

Instead of trying to do ERM on the 0 – 1 loss, We solve this problem by

1. approximating the solutions
2. surrogate $V(f, z_i)$ and minimize this

We rewrite the 0 – 1 loss

$$V_{0:1}(f, z) = \frac{1 - \text{sgn}(yf(x))}{2}$$

The way we have defined t :

- if $t > 0$ this implies that the data is assigned to the correct category.
- on the other hand if $t < 0$ this implies that the data is assigned to the wrong category.
- value to t is called the **margin** because it indicates how far the data point is from the decision boundary.

where $t = yf(x)$ is called the **margin**.

An ideal loss function should have the following properties: (1) *Differentiable*: To allow the use of gradient-based optimization algorithms like gradient descent. (2) *Sensitive to model performance*: It should accurately reflect the performance of the model on the given task. (3) *Scalable*: The loss function should work well with large datasets and high-dimensional spaces.

⁷The choice of a loss function depends on the problem, data distribution, and the desired properties of the model. For instance, in regression tasks, if the data has outliers, using MAE can be more robust than MSE. In classification, cross-entropy loss is generally preferred over other options due to its properties and compatibility with probabilistic outputs.

⁸An intractable combinatorial problem is a problem where the number of possible solutions grows exponentially with the size of the input, making it computationally impractical or impossible to solve optimally using brute-force or exhaustive search methods within a reasonable time frame.

§8 February 1, 2023

§8.1 Support Vector Machines

Support Vector Machines (SVMs) are a supervised machine learning algorithm used for classification and regression tasks. SVMs work by finding the optimal hyperplane that maximizes the margin between different classes in the feature space, providing robust and accurate predictions. They are particularly effective in high-dimensional spaces and can handle linearly separable as well as non-linearly separable data by using kernel functions to transform the input space into higher dimensions, enabling the discovery of complex decision boundaries.

$$\min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{subject to: } y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0$$

We now introduce Lagrange variables $\alpha_i \geq 0, i \in [m]$, associated to the m constraints. The Lagrangian can then be defined for all $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$ and $\alpha \in \mathbb{R}_+^m$, by

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1]$$

The KKT conditions are obtained by setting the gradient of the Lagrangian with respect to the primal variables \mathbf{w} and b to zero and by writing the complementary conditions:

$$\begin{aligned} \nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = 0 & \implies \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \\ \nabla_b \mathcal{L} = - \sum_{i=1}^m \alpha_i y_i = 0 & \implies \sum_{i=1}^m \alpha_i y_i = 0 \\ \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1] = 0 & \implies \alpha = 0 \vee y_i(\mathbf{w} \cdot \mathbf{x} + b) = 1 \end{aligned}$$

From the last condition we can see that if $\alpha \neq 0$, then $y_1(w \cdot x_i + b) = 1$.

§8.2 Non-separable Case

Practically speaking, the training data is not linearly separable, which implies that for any hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$, there exists $\mathbf{x}_i \in S$ such that

$$y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \not\geq 1.$$

Therefore we use a relaxed version of the constraints for each $i \in [m]$, there exists a $\xi_i \geq 0$ such that

$$y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i$$

The variables ξ_i are known as **slack variables**. A slack variable ξ_i measures the distance by which vector \mathbf{x}_i violates the inequality $y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1$.

§8.3 Primal optimization problem

In optimization, the primal problem refers to the original formulation of an optimization problem, while the dual problem is derived from the primal problem using a technique called Lagrangian duality. Both primal and dual problems are interconnected and provide valuable insights into the properties and solutions of each other. In the context of Support Vector Machines (SVMs), solving the dual problem instead of the primal problem has advantages such as the ability to use kernel functions to handle non-linearly separable data and the sparsity of the solution, which

leads to efficient representations and faster predictions.

$$\min_{w,b,\xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i$$

$$\text{subject to: } y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_i, \xi_i \geq 0, i \in [m],$$

The parameter C determines the trade-off between margin-maximization and the minimization of the slack penalty.

- As $C \rightarrow \infty$ any non-zero ξ_i will be forced to 0, therefore, each of the finitely many data points is prevented from being an outlier and attain the hard margin classifier.
- Therefore, the higher the value of C the less slack we are giving the data points.

§9 February 6, 2023

§9.1 Lagrange Multipliers

Lagrange multipliers are a mathematical method used to find the extrema (maxima or minima) of a function subject to constraints. Given a function $f(\mathbf{x})$ that we want to optimize and a constraint function $g(\mathbf{x}) = c$, where $\mathbf{x} \in \mathbb{R}^n$, Lagrange multipliers help us find the points where the gradient of the objective function is parallel to the gradient of the constraint function.

The method is based on introducing a new function, called the Lagrangian, which is defined as follows:

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda(g(\mathbf{x}) - c) \quad (14)$$

Here, λ is called the Lagrange multiplier. To find the extrema of $f(\mathbf{x})$ subject to the constraint $g(\mathbf{x}) = c$, we need to solve the following system of equations:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \nabla f(\mathbf{x}) - \lambda \nabla g(\mathbf{x}) = \mathbf{0} \quad (15)$$

$$g(\mathbf{x}) = c \quad (16)$$

The first equation states that the gradient of the objective function and the gradient of the constraint function are proportional (parallel) at the optimal point. The second equation enforces the constraint. By solving this system of equations, we can find the optimal points \mathbf{x} and the corresponding Lagrange multiplier λ .

§9.2 Karush-Kuhn-Tucker (KKT) theorem

The Karush-Kuhn-Tucker (KKT) theorem is an extension of the method of Lagrange multipliers for optimization problems with inequality constraints. Suppose we have an objective function $f(\mathbf{x})$ that we want to optimize (minimize or maximize) subject to inequality constraints $g_i(\mathbf{x}) \leq 0$ and equality constraints $h_j(\mathbf{x}) = 0$.

To solve this problem, we introduce the KKT conditions, which consist of the following set of equations:

1. Stationarity condition:

$$\nabla f(\mathbf{x}^*) - \sum_{i=1}^m \lambda_i^* \nabla g_i(\mathbf{x}^*) - \sum_{j=1}^p \mu_j^* \nabla h_j(\mathbf{x}^*) = \mathbf{0} \quad (17)$$

2. Primal feasibility:

$$g_i(\mathbf{x}^*) \leq 0, \quad i = 1, \dots, m \quad (18)$$

$$h_j(\mathbf{x}^*) = 0, \quad j = 1, \dots, p \quad (19)$$

3. Dual feasibility:

$$\lambda_i^* \geq 0, \quad i = 1, \dots, m \quad (20)$$

4. Complementary slackness:

$$\lambda_i^* g_i(\mathbf{x}^*) = 0, \quad i = 1, \dots, m \quad (21)$$

Here, \mathbf{x}^* is the optimal solution, and λ_i^* and μ_j^* are the Lagrange multipliers associated with the inequality and equality constraints, respectively.

The KKT theorem states that, under certain conditions, a point \mathbf{x}^* is optimal if and only if there exist Lagrange multipliers λ_i^* and μ_j^* such that the KKT conditions are satisfied.

In summary, the KKT theorem generalizes the method of Lagrange multipliers to handle optimization problems with both equality and inequality constraints.

§10 February 8, 2023

§10.1 Learning Theory

In the field of theoretical machine learning, learning theory serves as a framework for investigating fundamental questions related to machine learning. These may include:

- Inquiries into the effectiveness of different learning algorithms, what kind of algorithms work? Do some work better than others?
- Time complexity: how time does it take to learn an model?
- Sample complexity: The amount of data required to learn the model
- the development of general principles that govern the learning process in different contexts

§10.2 Probably Approximately Correct (PAC) Learning

PAC learning (Probably Approximately Correct learning) is a framework in machine learning that aims to provide theoretical guarantees on the ability of an algorithm to generalize well from a limited amount of training data. The basic idea is that a learning algorithm is considered **PAC-learnable** if it can efficiently learn a **concept** that is “probably” correct (i.e., has a high probability of being accurate) and “approximately” correct (i.e., the learned concept is not too far from the true concept) given a limited number of training examples. The PAC framework provides a mathematical basis for analyzing the sample complexity of learning algorithms, which is the minimum number of training examples required to achieve a certain level of accuracy.

In the binary classification setting, let $Y = \{0, 1\}$ ⁹ be the binary labels. Our goal is to learn a map $f : X \rightarrow Y$ with low risk.

PAC Learning terms:

- **concept** can be represented as a Boolean $c : X \rightarrow \{0, 1\}$, where X is the input space
- Given a set of training examples

$$S = \{(x_1, c(x_1)), \dots, (x_m, c(x_m))\}$$

where x_i is the data point and $c(x_i)$ is the corresponding label

- The goal of PAC learning is to output a **hypothesis**, $h : X \rightarrow \{0, 1\}$ that is “probably” correct with high probability $(1 - \delta)$, and “approximately” correct in the sense that error between h and c is not too large, $\leq \varepsilon$

In PAC learning concept refers to the underlying function or patterns that generates the observed data.

Definition 10.1 (Generalization Error) Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and an underlying distribution \mathcal{D} , the generalization error or risk of h is defined by

$$R(h) = \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq c(x)] = \mathbb{E}_{x \sim \mathcal{D}}[1_{h(x) \neq c(x)}],$$

where 1_ω is the indication function.

Since the distribution \mathcal{D} and the target concept class c are unknown to the learner, the generalization error is not accessible to the learner.

We define the hypothesis class \mathcal{H} to be the set all classifiers considered by the learning algorithm. *Is hypothesis relative to a learning algorithm?* Yes, if we are studying neural networks, then we would consider \mathcal{H} to be the set of all classifiers representable by some neural architecture.

⁹Its important to notice that the PAC learning framework uses the 0 – 1 loss function. Before when we were defining risk, we were considering many different types of loss functions, now we are only considering the 0 – 1 loss

Definition 10.2 (PAC learning) A concept class C is said to be **PAC-learnable** if there exists an algorithm \mathcal{A} and polynomial function $\text{poly}(\cdot, \cdot, \cdot)$ such that for any ε and δ , for all distributions D on \mathcal{X} and for any target concept $c \in C$, the following holds for any sample size $m \geq \text{poly}(1/\varepsilon, 1/\delta, \text{size}(c))^a$:

$$\mathbb{P}_{S \sim \mathcal{D}^m} [R(h_S) \leq \varepsilon] \geq 1 - \delta$$

If \mathcal{A} further runs in $\text{poly}(1/\varepsilon, 1/\delta, \text{size}(c))$, then C is said to be efficiently PAC-learnable. When such an algorithm \mathcal{A} exists, its called a PAC-learning algorithm for C .

^aSometimes $\text{size}(c)$ is omitted from the definition when computational representation of concepts is straight forward or not considered

In otherwords, the concept class C is PAC learnable by the algorithm \mathcal{A} if after seeing a number examples polynomial in $1/\varepsilon$ and $1/\delta$, \mathcal{A} returns an approximately correct (Accuracy $\geq 1 - \varepsilon$) h with high probability (Prob $\geq 1 - \delta$). Its important to note that this is a distribution-free worst case analysis, where D is arbitrary

We are allow the algorithm some exposure to the distribution D through its training, and what we are asking is how well can the algorithm use that exposure to an arbitrary distribution D in order to make a low risk prediction.

§11 February 15, 2023

In class we covered the example of Axis-aligned rectangles algorithm that made no errors on the training set S . We leveraged this property in our proof for the example.

In general we say that a hypothesis h is **consistent** with the sample S if it makes no errors on that sample.

We say an **algorithm is consistent** is for every sample S , it returns a hypothesis h_S which is consistent. For an algorithm \mathcal{A} to be consistent, we need C to be a subset of H , which is known as the **realizable setting**. In this case, the approximation error is zero.

Theorem 11.1 (Learning bounds – finite H , consistent case) — Let H be a finite set of functions mapping from \mathcal{X} to \mathcal{Y} . Let \mathcal{A} be an algorithm that for any target concept $c \in H$ and *i.i.d* sample S returns a consistent hypothesis $h_S : \hat{R}(h_S) = 0$. Then, for any $\varepsilon, \delta > 0$, the inequality $\mathbb{P}_{S \sim \mathcal{D}^m} [R(h_S) \leq \varepsilon] \geq 1 - \delta$ holds if

$$m \geq \frac{1}{\varepsilon} \left(\log |H| + \log \frac{1}{\delta} \right)$$

Proof. Let's denote the true risk $R(h)$ and the empirical risk $\hat{R}(h)$ for a hypothesis $h \in H$. We have:

$$R(h) = \mathbb{P}_{(x,y) \sim \mathcal{D}} [h(x) \neq y]$$

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}[h(x_i) \neq y_i]$$

By using the Union Bound, we can bound the probability of the event that at least one hypothesis in H has a true risk greater than ε :

$$\mathbb{P}[\exists h \in H : R(h) > \varepsilon] \leq \sum_{h \in H} \mathbb{P}[R(h) > \varepsilon]$$

By applying the Chernoff-Hoeffding Bound to each term on the right-hand side, we have:

$$\mathbb{P}[\exists h \in H : R(h) > \varepsilon] \leq \sum_{h \in H} \exp(-2m\varepsilon^2)$$

Using the inequality $m \geq \frac{1}{\varepsilon} (\log |H| + \log \frac{1}{\delta})$, we get:

$$\sum_{h \in H} 2 \exp(-2m\varepsilon^2) \leq \sum_{h \in H} 2 \frac{\delta}{2|H|} = \delta$$

Now, consider the complement of the event that at least one hypothesis in H has the difference between true risk and empirical risk greater than ε . This event implies that for all $h \in H$, we have $|R(h) - \hat{R}(h)| \leq \varepsilon$. Since the algorithm \mathcal{A} returns a consistent hypothesis h_S with $\hat{R}(h_S) = 0$, we have:

$$\mathbb{P}[R(h_S) \leq \varepsilon] \geq 1 - \delta$$

□

Any finite concept class is PAC learnable by consistent algorithm. For axis aligned rectangles we required

$$m \geq \frac{4}{\varepsilon} \log \frac{4}{\delta}$$

but in that case the hypothesis class was infinite.

Corollary 11.2

Fix $\varepsilon > 0$ and S denote i.i.d. sample of size m . Then for any hypothesis $h : X \rightarrow \{0, 1\}$, the following inequalities hold:

$$\begin{aligned}\mathbb{P}_{S \sim \mathcal{D}^m}[\hat{R}(h_S) - R(h_S) \geq \varepsilon] &\leq \exp(-2m\varepsilon^2) \\ \mathbb{P}_{S \sim \mathcal{D}^m}[\hat{R}(h_S) - R(h_S) \leq -\varepsilon] &\leq \exp(-2m\varepsilon^2)\end{aligned}$$

By the union bound, this implies the following two-sided inequality:

$$\mathbb{P}_{S \sim \mathcal{D}^m}[\hat{R}(h_S) - R(h_S) \geq -\varepsilon] \leq 2\exp(-2m\varepsilon^2)$$

Proof. Let S be an i.i.d. sample of size m , and let $Z_i = \mathbb{I}[h(x_i) \neq y_i]$ be an indicator random variable for the i -th sample. Then, we have:

$$\begin{aligned}\hat{R}(h) &= \frac{1}{m} \sum_{i=1}^m Z_i \\ R(h) &= \mathbb{E} \left[\frac{1}{m} \sum_{i=1}^m Z_i \right]\end{aligned}$$

Since Z_i are i.i.d. Bernoulli random variables with mean $R(h)$ and $0 \leq Z_i \leq 1$, we can apply the Chernoff-Hoeffding Bound. The Chernoff-Hoeffding Bound states that for any $\epsilon > 0$, the following inequalities hold:

$$\begin{aligned}\mathbb{P} \left[\frac{1}{m} \sum_{i=1}^m Z_i - R(h) \geq \epsilon \right] &\leq \exp(-2m\epsilon^2) \\ \mathbb{P} \left[\frac{1}{m} \sum_{i=1}^m Z_i - R(h) \leq -\epsilon \right] &\leq \exp(-2m\epsilon^2)\end{aligned}$$

These inequalities are equivalent to:

$$\begin{aligned}\mathbb{P}_{S \sim \mathcal{D}^m}[\hat{R}(h_S) - R(h_S) \geq \epsilon] &\leq \exp(-2m\epsilon^2) \\ \mathbb{P}_{S \sim \mathcal{D}^m}[\hat{R}(h_S) - R(h_S) \leq -\epsilon] &\leq \exp(-2m\epsilon^2)\end{aligned}$$

Applying the union bound, we get the two-sided inequality:

$$\mathbb{P}_{S \sim \mathcal{D}^m}[|\hat{R}(h_S) - R(h_S)| \geq \epsilon] \leq 2\exp(-2m\epsilon^2)$$

□

Corollary 11.3 (Generalization bound- single hypothesis)

Fix hypothesis $h : \mathcal{X} \rightarrow \{0, 1\}$. Then for any $\delta > 0$, the following inequality holds with probability at least:

$$R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

Proof. Let S be an i.i.d. sample of size m , and let $Z_i = \mathbb{I}[h(x_i) \neq y_i]$ be an indicator random variable for the i -th sample. Then, we have:

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^m Z_i$$

$$R(h) = \mathbb{E} \left[\frac{1}{m} \sum_{i=1}^m Z_i \right]$$

Since Z_i are i.i.d. Bernoulli random variables with mean $R(h)$ and $0 \leq Z_i \leq 1$, we can apply the Chernoff-Hoeffding Bound. The Chernoff-Hoeffding Bound states that for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$:

$$\mathbb{P} \left[\frac{1}{m} \sum_{i=1}^m Z_i - R(h) \geq \sqrt{\frac{\log \frac{2}{\delta}}{2m}} \right] \leq \delta$$

This inequality is equivalent to:

$$\mathbb{P}_{S \sim \mathcal{D}^m} [R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log \frac{2}{\delta}}{2m}}] \geq 1 - \delta$$

Thus, the generalization bound for a single hypothesis is:

$$R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

□

§12 February 27, 2023

Next we cover deterministic versus stochastic scenarios, and we also look at estimation versus approximation error. The stochastic scenario is there is not a single y deterministically associated to each x , but rather $\mathbb{P}(y|x)$ for each x value.

We saw the definition of Bayes error earlier in the course, but now we define it with more mathematical formality:

Definition 12.1 (Bayes Error) Given a distribution \mathcal{D} over $\mathcal{X} \times \mathcal{Y}$, the **Bayes error** R^* is defined as the infimum of errors achieved by measurable functions $h : \mathcal{X} \rightarrow \mathcal{Y}$:

$$R^* = \inf_{h \text{ measurable}} R(h)$$

A hypothesis h with $R(h) = R^*$ is called Bayes hypothesis or Bayes classifier.

In practice, hypothesis sets are generally infinite. All the results we previously saw only applied to finite hypothesis classes. Our next objective will be to derive and generalize results for learning guarantees for infinite hypothesis sets. We will see that *Rademacher complexity* will be a useful idea, and helps derive bound via McDiarmid's inequality. Furthermore, two additional notions we will leverage are *VC-dimension* and *growth function*. The idea is to first relate Rademacher complexity to the growth function and then bound the growth function in terms of VC-dimension, since the VC-dimension is often to bound/estimate.

Theorem 12.2 (Learning bounds – finite H , inconsistent case) — Let H be a finite hypothesis set. Then for any $\delta > 0$, the following inequality holds with probability at least:

$$R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}, \quad h \in H$$

This implies that generalization error is proportional to the square root of ratio of bitsize H to sample size m and it drops as $1/\sqrt{m}$

Proof. Let H be a finite hypothesis set, and let S be an i.i.d. sample of size m . For each hypothesis $h \in H$, let $Z_i^{(h)} = \mathbb{I}[h(x_i) \neq y_i]$ be an indicator random variable for the i -th sample. Then, we have:

$$\begin{aligned} \hat{R}(h) &= \frac{1}{m} \sum_{i=1}^m Z_i^{(h)} \\ R(h) &= \mathbb{E} \left[\frac{1}{m} \sum_{i=1}^m Z_i^{(h)} \right] \end{aligned}$$

Since $Z_i^{(h)}$ are i.i.d. Bernoulli random variables with mean $R(h)$ and $0 \leq Z_i^{(h)} \leq 1$, we can apply the Chernoff-Hoeffding Bound. The Chernoff-Hoeffding Bound states that for any $\delta > 0$, the following inequality holds with probability at least $1 - \frac{\delta}{|H|}$ for each hypothesis $h \in H$:

$$\mathbb{P} \left[\frac{1}{m} \sum_{i=1}^m Z_i^{(h)} - R(h) \geq \sqrt{\frac{\log \frac{2|H|}{\delta}}{2m}} \right] \leq \frac{\delta}{|H|}$$

Applying the union bound over all hypotheses $h \in H$, we get:

$$\mathbb{P}_{S \sim \mathcal{D}^m} \left[\exists h \in H : R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}} \right] \geq 1 - \delta$$

Thus, the learning bound for the finite hypothesis set case with inconsistencies is:

$$R(h_S) \leq \hat{R}(h_S) + \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}, \quad h \in H$$

This implies that the generalization error is proportional to the square root of the ratio of the bitsize of H to the sample size m and it drops as $1/\sqrt{m}$. □

This theorem indicates that any finite concept class can also be PAC learned by inconsistent algorithm but with possibly a lower learning rate.

§13 March 1, 2023

§14 March 6, 2023

§14.1 Rademacher Complexity

Definition 14.1 (Empirical Rademacher complexity) Let \mathcal{G} be a family of functions mapping from $Z \rightarrow [a, b]$ and $S = (z_1, \dots, z_m)$ a fixed sample of size m with elements in Z . Then, the empirical **Rademacher complexity** of \mathcal{G} with respect to the sample S is defined as:

$$\hat{\mathcal{R}}_S(\mathcal{G}) = \mathbb{E}_\sigma \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m \sigma_i g(z_i) \right]$$

where $\sigma = (\sigma_1, \dots, \sigma_m)^T$, with σ_i 's independent uniform random variables taking values $\{-1, +1\}$. The random variables σ_i are called **Rademacher variables**.^a

^aNote that we are implicitly assuming that the supremum over the family \mathcal{G} in the definition is measurable.

If we let \mathbf{g}_S denote the vector of values taken by function g over the sample S : $\mathbf{g}_S = (g(z_1), \dots, g(z_m))^T$. Then, the empirical Rademacher complexity can be written as

$$\hat{\mathcal{R}}_S(\mathcal{G}) = \mathbb{E}_\sigma \left[\sup_{g \in \mathcal{G}} \frac{\sigma \cdot \mathbf{g}_S}{m} \right]$$

- the inner product $\sigma \cdot \mathbf{g}_S$ measures the correlation of \mathbf{g}_S with the vector of noise σ
- the supremum $\sup_{g \in \mathcal{G}} \frac{\sigma \cdot \mathbf{g}_S}{m}$ is a measure of how well the function class \mathcal{G} correlates with σ over sample S .

The richer or more complex the family of class of functions \mathcal{G} the better it correlates with random noise, on average. Note that we are implicitly assuming that the supremum over the family \mathcal{G} in the definition is measurable.

Definition 14.2 (Rademacher complexity) Let \mathcal{D} denote the distribution according to which samples are drawn. For any integer $m \geq 1$, the **Rademacher complexity** of \mathcal{G} is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to \mathcal{D} :

$$\mathcal{R}_m(\mathcal{G}) = \mathbb{E}_{S \sim \mathcal{D}^m} [\hat{\mathcal{R}}_S(\mathcal{G})]$$

Now that we have the definitions down, we can derive/prove the generalization bounds based on Rademacher complexity.

Theorem 14.3 — Let \mathcal{G} be a family of functions mapping from $Z \rightarrow [a, b]$. Then for any $\delta > 0$, with probability at least $1 - \delta$ over the draw of an i.i.d. sample S of size m , each of the following holds for all $g \in \mathcal{G}$:

$$\mathbb{E}[g(z)] \leq \frac{1}{m} \sum_{i=1}^m g(z_i) + 2\mathcal{R}_m(\mathcal{G}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \quad (22)$$

$$\mathbb{E}[g(z)] \leq \frac{1}{m} \sum_{i=1}^m g(z_i) + 2\hat{\mathcal{R}}_S(\mathcal{G}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \quad (23)$$

Proof.

$$\begin{aligned}
\mathbb{E}_S[\Phi(S)] &= \mathbb{E}_S \left[\sup_{g \in \mathcal{G}} (\mathbb{E}[g] - \hat{\mathbb{E}}_S(g)) \right] \\
&= \mathbb{E}_S \left[\sup_{g \in \mathcal{G}} \mathbb{E}_{S'} [\hat{\mathbb{E}}_{S'}[g] - \hat{\mathbb{E}}_S(g)] \right] \\
&\leq \mathbb{E}_{S, S'} \left[\sup_{g \in \mathcal{G}} (\hat{\mathbb{E}}_{S'}(g) - \hat{\mathbb{E}}_S(g)) \right] \\
&= \mathbb{E}_{S, S'} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m (g(z'_i) - g(z_i)) \right] \\
&= \mathbb{E}_{S, S', \sigma} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m \sigma_i (g(z'_i) - g(z_i)) \right] \\
&\leq \mathbb{E}_{S', \sigma} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m \sigma_i (g(z'_i)) \right] + \mathbb{E}_{S', \sigma} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m -\sigma_i (g(z'_i)) \right] \\
&= 2 \mathbb{E}_{S', \sigma} \left[\sup_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^m \sigma_i (g(z'_i)) \right] = 2\mathcal{R}_m(\mathcal{G})
\end{aligned}$$

□

Note that above we are using the sub-additivity property of the supremum:

$$\sup(U + V) \leq \sup(A) + \sup(B)$$

Lemma 14.4 — Let \mathcal{H} be a family of functions taking values in $\{+1, -1\}$ and let \mathcal{G} be the family of loss functions associate to \mathcal{H} for the zero-loss:

$$\mathcal{G} = \{(x, y) \mapsto 1_{h(x) \neq y} : h \in \mathcal{H}\}$$

For any sample $S = ((x_1, y_1), \dots, (x_m, y_m))$ in $\mathcal{X} \times \{+1, -1\}$, the empirical Rademacher complexity can be written as ^a:

$$\begin{aligned}
\hat{\mathcal{R}}_S(\mathcal{G}) &= \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i 1_{h(x_i) \neq y_i} \right] \\
&= \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i \frac{1 - y_i h(x_i)}{2} \right] \\
&= \frac{1}{2} \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m -\sigma_i y_i h(x_i) \right] \\
&= \frac{1}{2} \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i h(x_i) \right] = \frac{1}{2} \hat{\mathcal{R}}_{S_x}(\mathcal{H})
\end{aligned}$$

^aWe use the fact that $1_{h(x_i) \neq y_i} = \frac{1 - y_i h(x_i)}{2}$ and the fact that for a fixed $y_i \in \{+1, -1\}$, σ_i and $-y_i \sigma_i$ are distributed the same way

We can leverage the above results of Empirical and Expected Rademacher complexity to derive generalization bounds for simple binary classifiers on the hypothesis set \mathcal{H} :

Theorem 14.5 (Rademacher Complexity bounds for Binary Classifier) — Let \mathcal{H} be a family of functions taking values $\{+1, -1\}$ and let \mathcal{D} be the distribution over the input space \mathcal{X} . Then for any $\delta > 0$, with probability at least $1 - \delta$ over a sample S of size m drawn from \mathcal{D} , the two following inequalities hold:

$$\begin{aligned}\mathcal{R}(h) &\leq \hat{\mathcal{R}}(h) + \mathcal{R}_m(\mathcal{H}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \\ \mathcal{R}(h) &\leq \hat{\mathcal{R}}(h) + \mathcal{R}_S(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}\end{aligned}$$

Proof. These results follow from the previous theorems. \square

Notice that these bounds depend of the data, since the empirical Rademacher complexity $\hat{\mathcal{R}}_S(\mathcal{H})$ is a function of a specific sample drawn S . These bounds are really only useful if we can actually compute $\mathcal{R}_S(\mathcal{H})$. We can do this by using the fact that σ_i and $-\sigma_i$ are distributed the exact same way.

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m -\sigma_i h(x_i) \right] = -\mathbb{E}_{\sigma} \left[\inf_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i h(x_i) \right]$$

Now computing $\inf_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^m \sigma_i h(x_i)$ equivalent to empirical risk minimization, and this can be computational hard for certain hypothesis classes. This is why we need other tools such as growth functions, which are combinatorial measures which are easier to compute.

§15 March 8, 2023

§15.1 Concentration Inequalities

Concentration inequalities are a set of results in probability theory and learning theory that provide bounds on the deviation of a random variable from its expected value or some other reference value. These inequalities are particularly useful in understanding the behavior of random variables in large samples or high-dimensional settings, as they quantify the probability that the random variable deviates from its expected value by a certain amount. Concentration inequalities, such as **Hoeffding's inequality**, **Chernoff bounds**, and **McDiarmid's inequality**, play a crucial role in learning theory, as they help derive generalization bounds and performance guarantees for machine learning algorithms. By bounding the difference between the empirical performance of a model on a finite training set and its expected performance on unseen data, concentration inequalities enable researchers to assess the reliability and robustness of learning algorithms, as well as to guide the development of more efficient and effective methods. Our objective in many learning theory problems is to show that $\mathcal{R}(f)$ is close to $\hat{\mathcal{R}}(f)$, where:

$$\hat{\mathcal{R}} = \frac{1}{m} \sum_{i=1}^m L(f(X_i), Y_i)$$

is the **empirical risk** for a function $f \in \mathcal{F}$. We refer to the empirical minimizing function as

$$f^* = \arg \min_{f \in \mathcal{F}} \hat{\mathcal{R}}(f)$$

As we can note above the empirical risk is an average over the sample S , we can also define

$$\mathcal{R}(f) = \mathbb{E}_{S \sim D}[L(f(X_i), Y_i)]$$

as the **expected risk** or also referred to as the generalization error.

§15.2 Probabilistic inequalities

Two very basic inequalities from probability theory are:

- Markov's inequality: For $X \geq 0$, $\mathbb{P}(X > t) \leq \frac{\mathbb{E}(X)}{t}$
- Chebyshev's inequality: $\mathbb{P}(|X - \mathbb{E}X| \geq t) \leq \frac{\text{Var}(X)}{t^2}$

The advantages of the two above inequalities is they are very general, however, we want learning bounds which give us stronger convergence. To achieve this we have to use the Hoeffding's Inequality.

Lets consider the sum of independent random variables, X_1, \dots, X_n , $S_n = \sum_{i=1}^n X_i$. Then for all $s > 0$, we have

$$\begin{aligned} \mathbb{P}(S_n \mathbb{E}S_n \geq t) &= \mathbb{P}(\exp\{s(S_n - \mathbb{E}S_n)\} \geq e^{st}) \\ &\leq e^{-st} \mathbb{E}e^{s(S_n - \mathbb{E}S_n)} \\ &= e^{-st} \prod_{i=1}^n \mathbb{E}e^{s(X_i - \mathbb{E}X_i)} \end{aligned}$$

The first inequality follows from Markov's inequality and the second equality follows from the independence of X_i . Note that $\mathbb{E}e^{s(X_i - \mathbb{E}X_i)}$ is the Moment generating function of $X_i - \mathbb{E}X_i$

Proposition 15.1 (Hoeffding's identity). For a random variable X with $\mathbb{E}X = 0$ and $a \leq X \leq b$, for $s > 0$, we have

$$\mathbb{E}e^{sX} \leq e^{s^2(b-a)^2/8}$$

Proof. We use the fact that e^{sx} is a convex function of x and that it is uniformly bounded. \square

Hoeffding's inequality is a concentration inequality that allows us to bound the probability that the sum or average of independent, identically distributed (i.i.d.) random variables deviates significantly from their expected sum or average. In other words, it provides an upper bound on the probability that the empirical mean of a set of i.i.d. random variables is far from their true mean. This bound depends on the range of the random variables and the desired level of precision.

Theorem 15.2 (Hoeffding's Inequality) — For a bounded random variables $X_i \in [a_i, b_i]$, where X_1, \dots, X_n are independent, then

$$\mathbb{P}(S_n - \mathbb{E}S_n \geq t) \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

and,

$$\mathbb{P}(\mathbb{E}S_n - S_n \geq t) \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

^a

^aThe tails of the error term are starting to look more Gaussian, since they are decaying exponentially at the rate of t^2

Hoeffding's inequality is particularly useful in the context of machine learning, where we often work with finite samples and want to understand how well the sample statistics (e.g., the empirical error rate) generalize to the entire population. By providing a bound on the deviation of the empirical mean from the true mean, Hoeffding's inequality helps us quantify the uncertainty of our estimates and assess the reliability of our learning algorithms.

§15.3 McDiarmid's Inequality

- **Hoeffding's inequality:** Applies to the sum or average of independent, identically distributed (i.i.d.) random variables that are bounded. It provides an upper bound on the probability that their empirical mean deviates significantly from the true mean. This inequality is particularly useful in machine learning for understanding the generalization performance of algorithms.
- **Chernoff bounds:** A more general form of concentration inequality that applies to the sum of i.i.d. random variables, which can be derived using the moment-generating functions of the random variables. Chernoff bounds can be tighter than Hoeffding's inequality in some cases, especially for rare events or when the random variables have a specific distribution (e.g., Bernoulli or Poisson).
- **McDiarmid's inequality:** Applies to the case where a function of multiple independent random variables has bounded differences, meaning that changing a single random variable can only cause a limited change in the function value. McDiarmid's inequality provides an upper bound on the probability that the function value deviates significantly from its expected value. This inequality is useful for understanding the stability of learning algorithms and randomized algorithms.
- **Markov's inequality:** A basic concentration inequality that provides an upper bound on the probability that a non-negative random variable exceeds a certain multiple of its expected value. This inequality is weaker than the others, but it has the advantage of requiring minimal assumptions and being applicable to a wide range of random variables.
- **Chebyshev's inequality:** A more specific concentration inequality that provides an upper bound on the probability that a random variable with finite variance deviates significantly from its expected value. Chebyshev's inequality is more powerful than Markov's inequality when dealing with random variables with known variance, as it takes this information into account.

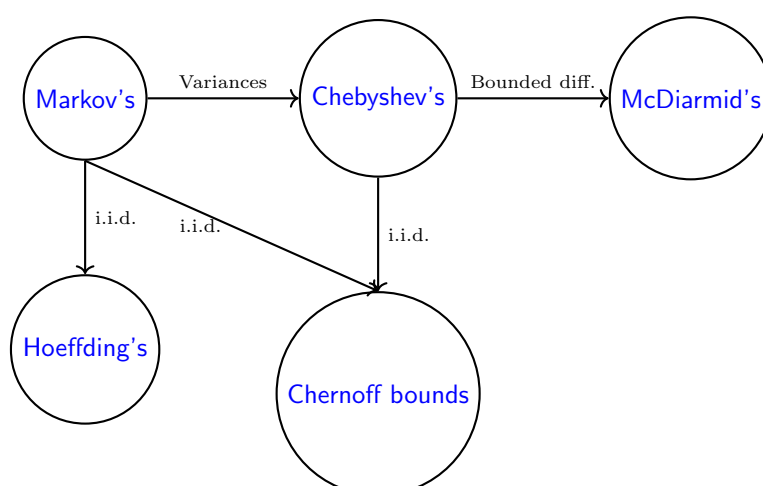


Figure 1: This diagram illustrates Markov's inequality as the most general one, with connections to Chebyshev's inequality (which adds knowledge of variances) and Hoeffding's and Chernoff bounds (which both require i.i.d. random variables). McDiarmid's inequality is connected to Chebyshev's inequality and represents bounded differences instead of variances.

§16 March 13, 2023

§16.1 VC Theory

The motivation behind the **VC-dimension** (Vapnik-Chervonenkis) lies in understanding the generalization ability of learning algorithms, which refers to how well an algorithm performs on unseen data. The VC-dimension helps to quantify the capacity or complexity of a hypothesis class (a set of candidate models) and is crucial in providing bounds on the generalization error, which is the difference between the error on the training data and the error on the entire data distribution. This understanding helps in selecting appropriate models that can balance the trade-off between fitting the training data well and generalizing to new data.

★ **Model complexity and generalization:** When developing a learning algorithm, one of the key challenges is to strike a balance between fitting the training data well and being able to generalize to unseen data. Models that are too simple may not capture the underlying structure of the data, leading to high bias and underfitting. On the other hand, models that are too complex can fit the training data very well, but may not generalize well to new data, leading to high variance and overfitting. The VC dimension provides a way to quantify the complexity of a hypothesis class, which helps in understanding the generalization ability of learning algorithms.

Hypothesis class and shattering: To define the VC-dimension, we need to introduce the notion of a hypothesis class and its ability to shatter a set of data points. A hypothesis class is a collection of candidate models (usually classifiers) that we consider when trying to learn from data. A hypothesis class is said to shatter a set of data points if it can classify all possible label configurations for those data points without any error.

Definition 16.1 (VC-dimension) The **VC-dimension** of a hypothesis set \mathcal{H} is the size of the largest set that can be shattered by \mathcal{H} :

$$VCdim(\mathcal{H}) = \max\{m : \prod_H(m) = 2^m\}.$$

The VC dimension is defined as the largest number of data points that can be shattered by a given hypothesis class. If a hypothesis class can shatter an arbitrarily large number of data points, its VC dimension is said to be infinite. Intuitively, a higher VC dimension indicates a more expressive hypothesis class with greater capacity to fit complex data.

Theorem 16.2 (Representer Theorem) — Let \mathcal{H} be a reproducing kernel Hilbert space (RKHS) with kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $S = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathbb{R}$ be a finite training set, and let $J : \mathcal{H} \rightarrow \mathbb{R}$ be a strictly monotonically increasing function. Consider the following optimization problem:

$$\min_{f \in \mathcal{H}} \left\{ R_S(f) = \sum_{i=1}^n L(f(x_i), y_i) + J(\|f\|_{\mathcal{H}}) \right\},$$

where $L : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function. If $f^* \in \mathcal{H}$ is a minimizer of $R_S(f)$, then there exist coefficients $\alpha_1, \dots, \alpha_n \in \mathbb{R}$ such that

$$f^*(x) = \sum_{i=1}^n \alpha_i K(x_i, x), \quad \forall x \in \mathcal{X}.$$

Proof. Let $f^* \in \mathcal{H}$ be a minimizer of $R_S(f)$. Define the function $g \in \mathcal{H}$ as

$$g(x) = f^*(x) - \sum_{i=1}^n \alpha_i K(x_i, x), \quad \forall x \in \mathcal{X}.$$

Note that $g(x_i) = 0$ for all $i = 1, \dots, n$, as the kernel K has the reproducing property. Now, we define $f_0 \in \mathcal{H}$ as

$$f_0(x) = \sum_{i=1}^n \alpha_i K(x_i, x), \quad \forall x \in \mathcal{X}.$$

We have

$$\|f^*\|_{\mathcal{H}}^2 = \|g + f_0\|_{\mathcal{H}}^2 = \|g\|_{\mathcal{H}}^2 + 2\langle g, f_0 \rangle_{\mathcal{H}} + \|f_0\|_{\mathcal{H}}^2.$$

Since $g(x_i) = 0$ for all $i = 1, \dots, n$, we have

$$\langle g, f_0 \rangle_{\mathcal{H}} = \sum_{i=1}^n g(x_i) \alpha_i = 0.$$

Therefore,

$$\|f^*\|_{\mathcal{H}}^2 = \|g\|_{\mathcal{H}}^2$$

□

§17 March 15, 2023

§17.1 VC dimension continued

Proof. Define $\tilde{X} = \{x_0, x_1, \dots, x_{d-1}\} \subset X$ as a subset shattered by \mathcal{H} . Given any $\epsilon > 0$, we select \mathcal{D} such that it concentrates on \tilde{X} and particularly on the point x_0 .

The probability distribution \mathcal{D} assigns a high likelihood $(1 - 8\epsilon)$ to the point x_0 , with the remaining probability equally distributed among the other points in \tilde{X} .

Assuming that the learning algorithm \mathcal{A} makes no errors on x_0 , we consider the set \tilde{S} as the subset of a sample S containing points from \tilde{X} excluding x_0 . For a given sample S , if the number of elements in \tilde{S} is less than or equal to $\frac{d-1}{2}$, we can establish a lower bound for the expected generalization error over all labelings of \tilde{X} that are consistent with \mathcal{H} .

This expectation is split into two parts: one where the generalization error exceeds ϵ and the other where it does not. The sum of probabilities for the first part is at least 8 times the probability that $R_{\mathcal{D}}(h_S, g)$ is greater than or equal to ϵ .

Finally, we apply a multiplicative Chernoff bound to determine the probability that the sample S has fewer than $\frac{d-1}{2}$ points from \tilde{X} . This leads to a bound on the probability of the generalization error $R_{\mathcal{D}}(h_S, g)$ being at least ϵ , which is strictly greater than δ for $\delta < 0.01$. \square

Proposition 17.1. Suppose α represents a random variable uniformly distributed over the set $\{\alpha_-, \alpha_+\}$, where $\alpha_- = \frac{1}{2} - \epsilon$ and $\alpha_+ = \frac{1}{2} + \epsilon$. Consider a collection of $m \geq 1$ independent and identically distributed random variables X_1, \dots, X_m taking binary values. If these variables are drawn from a distribution \mathcal{D}_α characterized by $\mathbb{P}_{\mathcal{D}_\alpha}[X = 1] = \alpha$, and h maps an m -tuple from X^m to $\{\alpha_-, \alpha_+\}$, then the ensuing relation is valid:

$$\mathbb{E}_\alpha [\mathbb{P}_{S \sim \mathcal{D}_\alpha^m}[h(S) \neq \alpha]] \geq \Phi(2m/\epsilon^2, \epsilon),$$

where $\Phi(m, \epsilon)$ is defined as $\frac{1}{4} \left(1 - \sqrt{1 - \exp\left(-\frac{m\epsilon^2}{1-\epsilon^2}\right)} \right)$ for all sample sizes m and any ϵ .

Proof. One could interpret the lemma in the context of an experiment involving two biased coins with biases α_- and α_+ . To deduce the tossed coin using a decision rule $h(S)$ based on a sample S drawn from either \mathcal{D}_{α_-} or \mathcal{D}_{α_+} , the sample size m is required to be no less than $\Omega(1/\epsilon^2)$. The detailed proof is outlined as an exercise. \square

Proposition 17.2. Let Z be a random variable taking values in $[0, 1]$. Then, for any $\gamma \in [0, 1]$,

$$\mathbb{P}[Z > \gamma] \geq \frac{\mathbb{E}[Z] - \gamma}{1 - \gamma} > \mathbb{E}[Z] - \gamma.$$

Proof. Since the values taken by Z are in $[0, 1]$,

$$\begin{aligned} \mathbb{E}[Z] &= \sum_{z \leq \gamma} z \mathbb{P}[Z = z] + \sum_{z > \gamma} z \mathbb{P}[Z = z] \\ &\leq \sum_{z \leq \gamma} \gamma \mathbb{P}[Z = z] + \sum_{z > \gamma} \mathbb{P}[Z = z] \\ &= \gamma \mathbb{P}[Z \leq \gamma] + \mathbb{P}[Z > \gamma] \\ &= \gamma(1 - \mathbb{P}[Z > \gamma]) + \mathbb{P}[Z > \gamma] \\ &= (1 - \gamma) \mathbb{P}[Z > \gamma] + \gamma, \end{aligned}$$

which concludes the proof. \square

¹⁰Given that Z only adopts values within $[0, 1]$, we can write the expected value of Z as the sum of the probabilities of Z being less than or equal to γ and Z being above γ multiplied by their respective outcomes. This sum can be simplified to γ times the probability of Z being not greater than γ plus the probability of Z being greater than γ . Simplifying further, we obtain that the expectation is equal to γ plus $(1 - \gamma)$ times the probability of Z surpassing γ .

§17.2 Growth function

Definition 17.3 (Growth function) The growth function $\Pi_H : \mathbb{N} \rightarrow \mathbb{N}$ for a hypothesis set H is defined by:

$$\Pi_H(m) = \max_{x_1, \dots, x_m \in X} |\{(h(x_1), \dots, h(x_m)) : h \in H\}|,$$

where m is a natural number, X is a domain set, and H is a set of functions mapping from X to a set of labels (e.g., $\{-1, +1\}$ for binary classification problems).

Unlike Rademacher complexity, this measure does not depend on the distribution; it is purely combinatorial.

To relate it to Rademacher complexity, we use:

Theorem 17.4 (Massart's lemma) — Let $A \subset \mathbb{R}^m$ be a finite set, with $r = \max_{x \in A} \|x\|_2$, then the following holds:

$$\mathbb{E} \left[\sup_{x \in A} \frac{1}{m} \sum_{i=1}^m \sigma_i x_i \right] \leq \frac{r \sqrt{2 \log |A|}}{m},$$

where σ_i are independent uniform random variables taking values in $\{-1, +1\}$ and x_1, \dots, x_m are the components of vector x .

Corollary 17.5

Let G be a family of functions taking values in $\{-1, +1\}$. Then the following holds:

$$\hat{\mathcal{R}}_m(G) \leq \sqrt{\frac{2 \log |G(m)|}{m}},$$

where $\hat{\mathcal{R}}_m(G)$ denotes the empirical Rademacher complexity of G on m samples.

Proof. Use $A = G$ and note that the maximum norm of u is \sqrt{m} . So, we have:

$$\hat{\mathcal{R}}_m(G) = \mathbb{E}_S \left[\mathbb{E}_\sigma \left[\sup_{u \in G} \frac{1}{m} \sum_{i=1}^m \sigma_i u_i \right] \right] \leq \sqrt{\frac{2 \log |G(m)|}{m}}.$$

This follows from Massart's lemma, noting that the growth function $|G(m)|$ bounds the cardinality of A and thus controls the complexity of the hypothesis class G . \square

Corollary 17.6

Let G be a family of functions taking values in $\{-1, +1\}$. Then the following holds:

$$\mathcal{R}_m(G) \leq \sqrt{\frac{2 \log \Pi_G(m)}{m}}.$$

Proof. For a fixed sample $S = (x_1, \dots, x_m)$, we denote by $G|_S$ the set of vectors of function values $(g(x_1), \dots, g(x_m))^T$ where g is in G . Since $g \in G$ takes values in $\{-1, +1\}$, the norm of these vectors is bounded by \sqrt{m} . Applying Massart's lemma:

$$\mathcal{R}_m(G) = \mathbb{E}_S \left[\mathbb{E}_\sigma \left[\sup_{u \in G|_S} \frac{1}{m} \sum_{i=1}^m \sigma_i u_i \right] \right] \leq \mathbb{E}_S \left[\sqrt{\frac{m}{2 \log |G|_S|}} \right].$$

By definition, $|G|_S$ is bounded by the growth function, thus

$$\mathcal{R}_m(G) \leq \mathbb{E}_S \left[\sqrt{\frac{m}{2 \log \Pi_G(m)}} \right] = \sqrt{\frac{2 \log \Pi_G(m)}{m}}.$$

which concludes the proof. \square

Corollary 17.7

Let \mathcal{H} be a family of functions taking values in $\{-1, +1\}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, for any $h \in \mathcal{H}$,

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{2 \log \Pi_{\mathcal{H}}(m)}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Theorem 17.8 (Sauer's Lemma) — Let \mathcal{H} be a hypothesis set with VC-dimension d . Then, for all $m \in \mathbb{N}$, the following inequality holds:

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^d \binom{m}{i}.$$

Proposition 17.9 (Lower Bound for a Realizable Case). Consider a hypothesis space \mathcal{H} with VC-dimension $d > 1$. For any sample size $m \geq 1$ and any learning algorithm \mathcal{A} , it is possible to find a probability distribution \mathcal{D} over the input space X and choose a specific target function g from \mathcal{H} such that the probability of the generalization error $R_{\mathcal{D}}(h_S, g)$ being greater than the fraction $\frac{d-1}{32m}$ is at least 0.01.

§18 March 20, 2023

§18.1 Learning theoretic results

§18.2 Introducing reproducing kernels

Definition 18.1 (Kernels) A function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a kernel over \mathcal{X} . The idea is to define a kernel K such that for any two points $x, x' \in \mathcal{X}$, $K(x, x')$ be equal to an inner product of vectors $\Phi(x)$ and $\Phi(x')$:

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \langle \Phi(x), \Phi(x') \rangle.$$

Theorem 18.2 (Mercer's condition) — Let $\mathcal{X} \subseteq \mathbb{R}^N$ be a compact set and let $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a continuous and symmetric function. Then, K admits a uniformly convergent expansion of the form

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x'),$$

with $a_n > 0$ if and only if for any square integrable function c ($c \in L_2(\mathcal{X})$), the following condition holds:

$$\int_{\mathcal{X}} \int_{\mathcal{X}} c(x) c(x') K(x, x') dx dx' \geq 0.$$

Definition 18.3 (Positive definite symmetric kernels) A kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is said to be positive definite symmetric (PDS) if for any $\{x_1, \dots, x_m\} \subseteq \mathcal{X}$, the matrix $K = [K(x_i, x_j)]_{i,j} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite (SPSD).

The matrix K is SPSPD if it is symmetric and one of the following two equivalent conditions holds:

- the eigenvalues of K are non-negative;
- for any column vector $c = (c_1, \dots, c_m)^\top \in \mathbb{R}^{m \times 1}$,

$$c^\top K c = \sum_{i,j=1}^m c_i c_j K(x_i, x_j) \geq 0.$$

For a sample $S = (x_1, \dots, x_m)$, $K = [K(x_i, x_j)]_{i,j}$ is called the *kernel matrix* or the *Gram matrix* associated to K and the sample S .¹¹

Example 18.4 (Polynomial kernels). For any constant $c > 0$, a polynomial kernel of degree $d \in \mathbb{N}$ is the kernel K defined over \mathbb{R}^N by:

$$\forall x, x' \in \mathbb{R}^N, \quad K(x, x') = (\mathbf{x} \cdot \mathbf{x}' + c)^d.$$

Polynomial kernels map the input space to a higher-dimensional space of dimension $\binom{N+d}{d}$. As an example, for an input space of dimension $N = 2$, a second-degree polynomial ($d = 2$) corresponds

¹¹In the terminology, the kernel matrix associated to a positive definite kernel is positive semidefinite. This is the correct mathematical terminology. However, in the context of machine learning, some authors have chosen to use instead the term *positive definite kernel* to imply a positive definite kernel matrix or used new terms such as *positive semidefinite kernel*.

to the following inner product in dimension 6:

$$\forall x, x' \in \mathbb{R}^2, \quad K(x, x') = (x_1 x'_1 + x_2 x'_2 + c)^2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \\ \sqrt{2}c x_1 \\ \sqrt{2}c x_2 \\ c \end{bmatrix}^\top \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1' x_2' \\ \sqrt{2}c x_1' \\ \sqrt{2}c x_2' \\ c \end{bmatrix}.$$

Thus, the features corresponding to a second-degree polynomial are the original features (x_1 and x_2), as well as products of these features, and the constant feature. More generally, the features associated to a polynomial kernel of degree d are all the monomials of degree at most d based on the original features. The explicit expression of polynomial kernels as inner products, as in the above, proves directly that they are PDS kernels.

Lemma 18.5 (Cauchy-Schwarz inequality for PDS kernels) — Let K be a PDS kernel. Then, for any $x, x' \in \mathcal{X}$,

$$K(x, x')^2 \leq K(x, x)K(x', x').$$

Proof. Consider the matrix $K = \begin{pmatrix} K(x, x) & K(x, x') \\ K(x', x) & K(x', x') \end{pmatrix}$. By definition, if K is PDS, then K is SPDS for all $x, x' \in \mathcal{X}$. In particular, the product of the eigenvalues of K , $\det(K)$, must be non-negative, thus, using $K(x', x) = K(x, x')$, we have

$$\det(K) = K(x, x)K(x', x') - K(x, x')^2 \geq 0,$$

which concludes the proof. \square

Theorem 18.6 (Reproducing kernel Hilbert space (RKHS)) — Let $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a PDS kernel. Then, there exists a Hilbert space \mathcal{H} and a mapping Φ from \mathcal{X} to \mathcal{H} such that:

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \langle \Phi(x), \Phi(x') \rangle.$$

Furthermore, \mathcal{H} has the following property known as the reproducing property:

$$\forall h \in \mathcal{H}, \forall x \in \mathcal{X}, \quad h(x) = \langle h, K(x, \cdot) \rangle.$$

\mathcal{H} is called a reproducing kernel Hilbert space (RKHS) associated to K .

Proof. For any $x \in \mathcal{X}$, define $\Phi(x) : \mathcal{X} \rightarrow \mathbb{R}$ as follows:

$$\forall x' \in \mathcal{X}, \quad \Phi(x)(x') = K(x, x').$$

We define \mathcal{H}_0 as the set of finite linear combinations of such functions $\Phi(x)$:

$$\mathcal{H}_0 = \left\{ \sum_{i \in I} a_i \Phi(x_i) : a_i \in \mathbb{R}, x_i \in \mathcal{X}, |I| < \infty \right\}.$$

Now, we introduce an operation $\langle \cdot, \cdot \rangle$ on $\mathcal{H}_0 \times \mathcal{H}_0$ defined for all $f, g \in \mathcal{H}_0$ with $f = \sum_{i \in I} a_i \Phi(x_i)$ and $g = \sum_{j \in J} b_j \Phi(x'_j)$ by

$$\langle f, g \rangle = \sum_{i \in I, j \in J} a_i b_j K(x_i, x'_j).$$

By definition, $\langle \cdot, \cdot \rangle$ is symmetric. Since K is PDS, we have

$$\langle f, f \rangle = \sum_{i, j \in I} a_i a_j K(x_i, x_j) \geq 0.$$

Thus, $\langle \cdot, \cdot \rangle$ is positive semidefinite bilinear form. \square

To any kernel K , we can associate a *normalized kernel* K' defined by

$$K'(x, x') = \begin{cases} 0 & \text{if } (K(x, x) = 0) \vee (K(x', x') = 0), \\ \frac{K(x, x')}{\sqrt{K(x, x)K(x', x')}} & \text{otherwise.} \end{cases}$$

By definition, for a normalized kernel K' , $K'(x, x) = 1$ for all $x \in \mathcal{X}$ such that $K(x, x) \neq 0$. An example of normalized kernel is the Gaussian kernel with parameter $\sigma > 0$, which is the normalized kernel associated to K : $(x, x') \mapsto \exp\left(-\frac{\|x' - x\|^2}{2\sigma^2}\right)$:

$$\forall x, x' \in \mathbb{R}^N, \quad \frac{K'(x, x')}{\sqrt{K'(x, x)K'(x', x')}} = \frac{\exp\left(-\frac{\|x' - x\|^2}{2\sigma^2}\right)}{\exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)\exp\left(-\frac{\|x'\|^2}{2\sigma^2}\right)} = \exp\left(-\frac{\|x' - x\|^2}{2\sigma^2}\right).$$

Lemma 18.7 (Normalized PDS kernels) — Let K be a PDS kernel. Then, the normalized kernel K' associated to K is PDS.

Proof. Let $\{x_1, \dots, x_m\} \subseteq \mathcal{X}$ and let c be an arbitrary vector in \mathbb{R}^m . We will show that the sum $\sum_{i,j=1}^m c_i c_j K'(x_i, x_j)$ is non-negative. By lemma 6.7, if $K(x_i, x_i) = 0$ then $K(x_i, x_j) = 0$ and thus $K'(x_i, x_j) = 0$ for all $j \in [m]$. Thus, we can assume that $K(x_i, x_i) > 0$ for all $i \in [m]$. Then, the sum can be rewritten as follows:

$$\sum_{i,j=1}^m \frac{c_i c_j K(x_i, x_j)}{\sqrt{K(x_i, x_i)K(x_j, x_j)}} = \sum_{i,j=1}^m c_i c_j \frac{\langle \Phi(x_i), \Phi(x_j) \rangle}{\|\Phi(x_i)\|_{\mathcal{H}} \|\Phi(x_j)\|_{\mathcal{H}}} = \left\| \sum_{i=1}^m \frac{c_i \Phi(x_i)}{\|\Phi(x_i)\|_{\mathcal{H}}} \right\|_{\mathcal{H}}^2 \geq 0,$$

where Φ is a feature mapping associated to K , which exists by theorem 6.8. \square

§19 March 22, 2023

§19.1 Functional Analysis Background

Definition 19.1 A *function space* \mathcal{F} is a space whose elements are functions, e.g., $f : \mathbb{R}^d \rightarrow \mathbb{R}$.

Definition 19.2 An *inner product* is a function $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}$ that satisfies the following properties for every $f, g \in \mathcal{F}$ and $\alpha \in \mathbb{R}$:

1. Symmetric: $\langle f, g \rangle = \langle g, f \rangle$.
2. Linear: $\langle r_1 f_1 + r_2 f_2, g \rangle = r_1 \langle f_1, g \rangle + r_2 \langle f_2, g \rangle$ for any scalars $r_1, r_2 \in \mathbb{R}$.
3. Positive-definite: $\langle f, f \rangle \geq 0$ for all $f \in \mathcal{F}$ and $\langle f, f \rangle = 0$ if and only if $f = 0$.

Definition 19.3 A *norm* is a nonnegative function $\| \cdot \| : \mathcal{F} \rightarrow \mathbb{R}$ such that for all $f, g \in \mathcal{F}$ and $\alpha \in \mathbb{R}$:

1. $\|f\| \geq 0$ and $\|f\| = 0$ if $f = 0$; the zero function.
2. Triangle inequality: $\|f + g\| \leq \|f\| + \|g\|$.
3. Absolutely homogeneous: $\|\alpha f\| = |\alpha| \|f\|$.

A norm can be defined via an inner product as $\|f\| = \sqrt{\langle f, f \rangle}$.

§19.2 Reproducing kernel Hilbert spaces (RKHS)

Definition 19.4 A kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a *positive definite kernel* on \mathcal{X} if for every $N \in \mathbb{N}$ and for all $x_1, \dots, x_N \in \mathcal{X}$, and for any coefficients $a_1, \dots, a_N \in \mathbb{R}$, the sum $\sum_{i,j=1}^N a_i a_j K(x_i, x_j) \geq 0$. This is known as the Mercer condition. If K is also symmetric, that is $\forall x, x' \in \mathcal{X}, K(x, x') = K(x', x)$ then we call K a *reproducing kernel*. Some people call it a PDS kernel.

If K is a reproducing kernel, then the quantity $\sqrt{K(x, x)}$ acts like a norm of x . The Cauchy-Schwarz inequality also holds: $K(x, x') \leq \sqrt{K(x, x)K(x', x')}$. We can normalize a reproducing kernel K by defining $\tilde{K}(x, x') = \frac{K(x, x')}{\sqrt{K(x, x)K(x', x')}}$. Then \tilde{K} is also a reproducing kernel.

Definition 19.5 A Hilbert space H consisting of real-valued functions on \mathcal{X} is a *reproducing kernel Hilbert space (RKHS)* if $\forall x \in \mathcal{X}$, the evaluation functional $\text{Ev}_x : H \rightarrow \mathbb{R}$ defined by $\text{Ev}_x(h) := h(x)$ is continuous. In Hilbert spaces, continuity of a linear functional is equivalent to boundedness, so we can write $\forall x \in \mathcal{X}, \exists C_x > 0$ such that $\forall h \in H, |\text{Ev}_x(h)| \leq C_x \|h\|_H$.

Theorem 19.6 — If H is a RKHS of real-valued functions on \mathcal{X} , then for each $x \in \mathcal{X}$ there exists an element $K_x \in H$ such that $\forall h \in H, \text{Ev}_x(h) = \langle h, K_x \rangle_H$. The map $\Phi : \mathcal{X} \rightarrow H$ that sends $x \mapsto K_x$ is called the *feature map associated to H* .

The normalized kernel $\tilde{K}(x, x')$ is the cosine of the angle between $\Phi(x)$ and $\Phi(x')$ in the Hilbert space H_K .

Theorem 19.7 — Let H be a RKHS of real-valued functions on \mathcal{X} , then the kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ defined by $K(x, x') := \langle K_x, K_{x'} \rangle_H = \langle \Phi(x), \Phi(x') \rangle_H$ is a reproducing kernel, called the *reproducing kernel of H* . We can view K as a manifestation in \mathcal{X} of the inner product on H .

Theorem 19.8 — Let K be a reproducing kernel on \mathcal{X} . Consider linear combinations $\sum_{i=1}^N a_i K_{x_i}$ with $a_i \in \mathbb{R}$ and $x_i \in \mathcal{X}$ and define an inner product on these $\langle K_{x_i}, K_{x_j} \rangle := K(x_i, x_j)$. Then this is a pre-Hilbert space. Its completion is a Hilbert space that happens to be RKHS; we denote it H_K .

Common Kernels:

- Linear kernel: $K(x, x') = \langle x, x' \rangle$
- Gaussian kernel: $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$, where $\sigma > 0$ is the width of the kernel.
- Polynomial kernel: $K(x, x') = (\langle x, x' \rangle + 1)^d$, where $d \in \mathbb{N}$ is the degree of the polynomial.

We illustrate how regularization controls complexity through a simple linear case. Our function space is 1-dimensional lines through the origin with a linear kernel:

$$f(x) = w \cdot x \quad \text{and} \quad K(x, x_i) = \langle x, x_i \rangle$$

giving an RKHS norm of

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \langle K_w, K_w \rangle_{\mathcal{H}} = K(w, w) = \|w\|^2$$

so that our measure of complexity is the slope of the line. We want to separate two classes using lines and see how the magnitude of the slope corresponds to a measure of complexity.

A classification problem can be thought of as "harder" when the distinctions between the two classes are less pronounced. Having a norm that increases with slope is a good choice in this case: by penalizing lines with high slope, we only use complex solutions to problems if doing so is necessary to reduce the training error.

The RKHS norm associated with the linear kernel induces a metric on the space of functions that directly corresponds to the Euclidean norm of the weight vector in the primal space. This norm quantifies the complexity of the function in terms of its deviation from zero, which in the context of linear functions, is directly related to the slope or rate of change.

§20 March 27, 2023

§20.1 Online learning

Online learning is a method within machine learning where the model is trained incrementally by feeding it data samples sequentially, one at a time. This approach is particularly useful when dealing with large datasets that cannot be processed all at once due to memory constraints, or when the data is being collected in a real-time fashion.

20.1.1 Mathematical Formulation

In online learning, we consider a model characterized by parameters θ . When a new data point (x_t, y_t) arrives, the model parameters are updated using a learning rule derived from an objective function $L(\theta; x_t, y_t)$, typically representing the loss of the model's prediction on the new data point.

The update rule is generally of the form:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_{\theta} L(\theta_t; x_t, y_t),$$

where θ_t are the parameters at time step t , η_t is the learning rate, and $\nabla_{\theta} L(\theta_t; x_t, y_t)$ is the gradient of the loss with respect to the parameters.

20.1.2 Advantages of Online Learning

Online learning algorithms offer several advantages:

- **Memory Efficiency:** They do not require loading the entire dataset into memory.
- **Adaptivity:** They can adapt to changes in the underlying data distribution over time.
- **Real-time Processing:** They are suitable for applications that require real-time processing and decision-making.

Some common online learning algorithms include:

- **Online Gradient Descent:** Used for regression and classification tasks.
- **Stochastic Gradient Descent (SGD):** A variant of online gradient descent where the gradient is estimated using a single sample.
- **Perceptron Algorithm:** One of the earliest online learning algorithms used for binary classification.

§20.2 Introduction to the perceptron

The Perceptron algorithm represents a foundational piece in the evolution of machine learning techniques. It operates as an online algorithm for linear classification, developing a decision-making model that is iteratively refined using individual data instances.

This method constructs a weight vector $\mathbf{w}_t \in \mathbb{R}^N$, which delineates a decision boundary. Starting from an initial guess \mathbf{w}_0 , the algorithm proceeds through successive rounds $t \in \{1, 2, \dots, T\}$, making predictions about the classification of new data points \mathbf{x}_t based on the current weight vector. If a discrepancy arises between the predicted and actual labels, the algorithm adapts by adjusting \mathbf{w}_t in the direction proportional to \mathbf{x}_t scaled by the label y_t and a learning rate η , a process grounded in the principles of the gradient descent method.

The adjustment made to the weight vector is informed by the sign of the dot product $y_t \mathbf{w}_t \cdot \mathbf{x}_t$, which signifies whether the point \mathbf{x}_t has been correctly classified. Specifically, the weight vector is modified to enhance the alignment of \mathbf{w}_t with the input vector \mathbf{x}_t corresponding to the correct classification.

The update rule can be formally expressed as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta y_t \mathbf{x}_t \text{ if } y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \leq 0; \text{ otherwise, } \mathbf{w}_{t+1} = \mathbf{w}_t.$$

This iterative refinement is designed to reinforce the correct linear separations imposed by the Perceptron, thereby iteratively reducing the classification error on the training set. The Perceptron's updates cease when it reaches a state where all points are correctly classified, assuming the data is linearly separable.

§21 March 29, 2023

§21.1 Perceptron Algorithm

1. Initialize the weight vector \mathbf{w} and bias b with zeros or small random values.
2. Set the learning rate $\eta > 0$. The learning rate determines the step size during the weight update process.
3. Repeat the following steps until the stopping criterion is met (for example, a predefined number of iterations or a minimum error rate):
 - a) Select a training example (\mathbf{x}_i, y_i) from the dataset, where \mathbf{x}_i is the feature vector and $y_i \in \{-1, +1\}$ is the corresponding label.
 - b) Compute the output of the perceptron for the selected example: $\hat{y}_i = \text{sign}(\mathbf{w}^T \mathbf{x}_i + b)$.
 - c) Update the weight vector and bias if the perceptron made an incorrect prediction:

$$\begin{cases} \mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i & \text{if } \hat{y}_i \neq y_i \\ b \leftarrow b + \eta y_i & \text{if } \hat{y}_i \neq y_i \end{cases}$$

Perceptron Model:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \xrightarrow{\text{Weights}} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \xrightarrow{\text{Dot Product}} \mathbf{w}^T \mathbf{x} \xrightarrow{\text{Add Bias}} \mathbf{w}^T \mathbf{x} + b \xrightarrow{\text{Activation Function}} \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

Pros:

- *Efficiency*: Due to its simplicity, the perceptron algorithm offers computational efficiency, allowing it to manage large datasets relatively quickly.
- *Adaptive learning*: The algorithm supports adaptive learning by processing and updating the model with one training example at a time, making it suitable for scenarios with continuous data arrival that requires model adaptation.
- *Convergence guarantee*: If the dataset is linearly separable, the perceptron algorithm will converge to a solution in a finite number of iterations.

Cons:

- *Linear separability constraint*: The perceptron algorithm can solely learn linear decision boundaries, and for non-linearly separable problems, it will not converge to an accurate solution.
- *Lack of probabilistic interpretation*: The perceptron algorithm does not offer an interpretation of the output or class membership probabilities in a probabilistic sense, unlike some other classifiers (e.g., logistic regression).

§21.2 Single-Layer Neural Networks

A neural network takes an input vector of p variables (X_1, \dots, X_p) build a non-linear function $f(X)$ to predict the response Y . The features, X_1, \dots, X_p , in the terminology of neural networks, make up the input layer. Each of the inputs from the input layer feeds into each of the K hidden units.

§21.3 Multi-Layer Neural Networks

§21.4 Functions

Sigmoid:

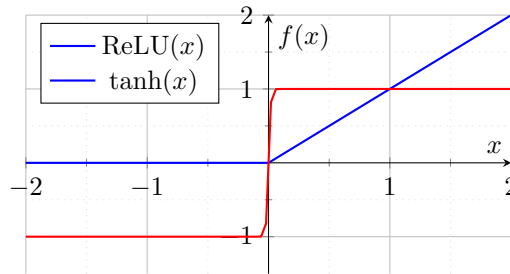
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Hyperbolic Tangent (tanh):

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{if } x \geq 0. \end{cases}$$



Leaky Rectified Linear Unit (Leaky ReLU):

$$\text{Leaky ReLU}(x, \alpha) = \begin{cases} \alpha x, & \text{if } x < 0, \\ x, & \text{if } x \geq 0. \end{cases}$$

Exponential Linear Unit (ELU):

$$\text{ELU}(x, \alpha) = \begin{cases} \alpha(e^x - 1), & \text{if } x < 0, \\ x, & \text{if } x \geq 0. \end{cases}$$

Softmax:

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}, \quad \text{for } i = 1, \dots, K$$

§22 April 3, 2023

§22.1 Optimization

We can reduce the task of learning a model for supervised learning problem to solving an optimization problem we take the form of

$$\theta^* = \arg \min_{\theta} g(\theta),$$

where θ is a parameter and g is combined average loss over all training examples and a regularization penalty.

Derivative free optimization. Derivative-free optimization refers to a class of optimization methods that do not rely on gradient information (i.e., derivatives) to find the optimal solution. These methods are particularly useful when dealing with problems where the objective function is noisy, discontinuous, non-differentiable, or when its derivatives are unavailable or computationally expensive to compute. These methods explore the search space using only function evaluations, without requiring information about the function's gradient or Hessian.

§22.2 Gradient descent

First order methods. By making additional assumptions about g , we can improve the efficiency of optimization. By restricting ourselves to only using class of differentiable functions, we can compute gradients $\nabla_{\theta} g$ via **backpropagation**.

★ The *gradient* is simply a vector of partial derivatives, that gives us the slope of g along every dimension of θ .^a

^aThe gradient allows construct first order approximation Taylor expansion of g , hence “first-order method”

Since we want to minimize g , we can iterative improve θ by adding to it a small amount of negative gradient direction. This is the motivation behind the **gradient descent algorithm**:

1. evaluates the gradient via backpropagation
2. updates the parameters by taking a small step in the direction of negative gradient

§22.3 Batch gradient descent

§22.4 Mini-batch gradient descent

§22.5 Stochastic gradient descent

The datasets we use in practice can be very large, so we can only estimate the gradient using small mini-batch examples (around 100 exmples). This results in perform many approximate updates instead of fewer exact updates.

Algorithm 1 Stochastic Gradient descent

Require: a starting point $\theta \in \text{domain}(g)$

Ensure: a step size $\varepsilon \in \mathbb{R}^*$

Repeat:

- (i) Sample a minibatch of m samples $\{(x_1, y_1), \dots, (x_m, y_m)\}$ from training data
- (ii) Estimate the gradient $\nabla_{\theta} g(\theta) \approx \nabla_{\theta} [\frac{1}{m} \sum_{i=1}^m L(f_{\theta}(x_i), y_i) + R_{\theta}(f_{\theta})]$ with backpropagation.
- (iii) Compute the update direction: $\Delta\theta \leftarrow -\varepsilon \nabla_{\theta} g(\theta)$
- (iv) Perform a parameter update: $\theta \leftarrow \theta + \Delta\theta$

until convergence.

The *learning rate* also called the step size ε , is a crucial parameter in SGD. If the value of ε is too high the algorithm may not converge, if its set too low, it will take too long to converge.

§22.6 Advanced optimization techniques

By modifying the steps in the above algorithm, we can obtain faster convergence. The **momentum** update, is designed to concourse progress along small but consistent directions of the gradient. Let $\mathbf{g} = \nabla_{\theta} g(\theta)$ for the gradient vector, the update $\Delta\theta$ is computed first by updating an intermediate variable $\mathbf{v} := \alpha\mathbf{v} + \mathbf{g}$, it is initialized first at zero. Then we compute the update as $\Delta\theta := -\varepsilon\mathbf{v}$. Why does this make sense? Note that \mathbf{v} contains an exponentially-decaying sum of previous gradient directions. ¹²

1. **Autograd:** Autograd update uses an intermediate variable $\mathbf{r} := \mathbf{r} + \mathbf{g} \otimes \mathbf{g}$ of the squared sum of gradients. (\otimes is elementwise multiplication). The update is modulated as follows:

$$\delta\theta := -\frac{\varepsilon}{\delta + \sqrt{\mathbf{r}}} \otimes \mathbf{g}$$
2. **RMSProp (Root Mean Square Propagation):** RMSProp maintains a moving average of the squared gradients and divides the current gradient by the square root of this moving average, effectively normalizing the gradient before applying the update.
3. **Adam (Adaptive Moment Estimation):** Adam is another adaptive learning rate optimization algorithm that combines the ideas of RMSProp and momentum. Like RMSProp, Adam maintains a moving average of the squared gradients to adjust the learning rate for each parameter. Additionally, Adam also maintains a moving average of the gradients themselves, similar to the momentum method. By combining these two approaches, Adam provides a more stable and efficient optimization process, often leading to faster convergence and improved performance in training neural networks.

¹²This method of update has a close relationship to physics, where the gradient is interpreted as force \mathbf{F} on a particle with position θ .

§23 April 5, 2023

§23.1 Boosting

Definition 23.1 (Weak learning) A concept class \mathcal{C} is said to be weakly PAC-learnable if there exists an algorithm A , $\gamma > 0$, and a polynomial function $\text{poly}(\cdot, \cdot)$ such that for any $\delta > 0$, for all distributions \mathcal{D} on \mathcal{X} and for any target concept $c \in \mathcal{C}$, the following holds.

AdaBoost Algorithm

Given a set $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$:

1. Initialize $D_1(i) = \frac{1}{m}$ for all i .
2. For $t = 1$ to T do:
 - a) Train base classifier h_t in \mathcal{H} with small error $\epsilon_t = \mathbb{P}_{i \sim D_t}[h_t(x_i) \neq y_i]$.
 - b) Choose $\alpha_t = \frac{1}{2} \log \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.
 - c) Compute normalization factor $Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$.
 - d) Update distribution for $i = 1$ to m :

$$D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}.$$

3. Output the final hypothesis:

$$f \leftarrow \sum_{t=1}^T \alpha_t h_t.$$

§23.2 AdaBost