

Solutions - Machine Learning Interviews Book  
Version 1.0

Hamna Moieez

April 6, 2023



# Contents

<b>1</b>	<b>Math</b>	<b>7</b>
1.1	Algebra and (little) Calculus: Vectors	7
1.2	Algebra and (little) Calculus: Matrices	10
1.3	Algebra and (little) Calculus: Dimensionality Reduction	14
1.4	Algebra and (little) Calculus: Calculus and Convex Optimization	17
1.5	Probability and Statistics: Probability	17
1.6	Probability and Statistics: Stats	17
<b>2</b>	<b>Computer Science</b>	<b>21</b>
2.1	Algorithms	21
2.2	Complexity and Numerical Analysis	21
2.3	Data Structures	21
<b>3</b>	<b>Machine Learning Workflows</b>	<b>23</b>
3.1	Basics	23
3.2	Sampling and creating training data	31
3.3	Objective functions, metrics, and evaluation	40
<b>4</b>	<b>Machine Learning Algorithms</b>	<b>51</b>
4.1	Classical Machine Learning: Questions	51
4.2	Deep Learning Architectures and Applications: Natural Language Processing	61
4.3	Deep Learning Architectures and Applications: Computer Vision	61
4.4	Deep Learning Architectures and Applications: Reinforcement Learning	67
4.5	Deep Learning Architectures and Applications: Other	67
4.6	Training Neural Networks	67



# Preamble

Before we begin, it is important that a few things are highlighted to the reader.

I prepared this manual (and still am updating) during my interview preparations for machine learning roles. This manual, mainly, contains solutions to problems in Chip Huyen's Introduction to Machine Learning Interviews book. You can read the excellent book by the author here: [BOOK](#). It has helped me refresh a lot of the concepts as I was working towards interview preparations.

## Word of Caution

Please note that I have accessed and used a lot of online resources, several google searches, sometimes ChatGPT as well, stack exchange, medium and other blog posts while writing down the solutions. I have tried to reference answers, as many I could retroactively remember; however, some might not be referenced but it is important to know that it is standing on someone's shoulders.

## A Note on Ordering of Chapters

Since many of the early chapters in the book are readings, the questions start from book's chapter # 05 (Math) in Part II. I did reproduce the questions before writing the solution, but in case the reader wants to cross-reference, the numbering and order should be kept in mind. Simply put, the questions are in order, but this document's chapter # 01 is the book's chapter # 05 and so on.

## Appeal

I tried my best to write these answers as correctly as possible. But, to err is human. Therefore, in case the reader spots a mistake, typo or blatant stupidity on my part, please open a pull request and I will update the document.



# Chapter 1

## Math

### 1.1 Algebra and (little) Calculus: Vectors

#### 1.1.1 Dot Product

1. [E] What's the geometric interpretation of the dot product of two vectors?

The dot product of two vectors is a scalar quantity that represents the projection of one vector onto another. Geometrically, the dot product of two vectors A and B is given by:

$$A \cdot B = |A| * |B| * \cos(\theta)$$

where  $|A|$  and  $|B|$  are the magnitudes of the two vectors, and  $\theta$  is the angle between them.

The dot product can be visualized as follows: Imagine that you have two vectors, A and B, and you want to project vector A onto vector B. The dot product of A and B gives you the length of the projection of A onto B, which is the length of the segment that starts at the tip of vector A and ends at the point where the projection intersects vector B.

If the dot product is positive, it means that the angle between the two vectors is acute, and the projection of A onto B points in the same direction as B. If the dot product is negative, it means that the angle between the two vectors is obtuse, and the projection of A onto B points in the opposite direction as B. If the dot product is zero, it means that the two vectors are perpendicular to each other, and the projection of A onto B is zero.

Thus, the dot product provides us with a way to determine the relationship between two vectors and their angles in a geometric sense.

2. [E] Given a vector u, find a vector v of unit length such that the dot product of u and v is maximum.

u and v are in the same direction with  $\theta = 0$ .

#### 1.1.2 Outer Product

1. [E] Given two vectors  $a = [3, 2, 1]$  and  $b = [-1, 0, 1]$ . Calculate the outer product  $a^T b$ ?

The outer product of two vectors is a way to represent the relationship between two vectors as a matrix. Given two vectors, u and v, their outer product is denoted by  $u \otimes v$  and is defined as:

$$u \otimes v = uv^T$$

where  $uv^T$  represents the matrix obtained by taking the transpose of the column vector u and multiplying it by the row vector v. The resulting matrix is a rectangular matrix with dimensions equal to the length of u by the length of v. For instance for vector u with dimensions  $m \times 1$  and v with dimensions  $n \times 1$   $u^T v$  will have dimensions  $1 \times n$  so  $u^T v$  will be a matrix of dimensions  $m \times n$ .

The elements of the outer product matrix can be computed as:

$$(u \otimes v)_{ij} = u_i * v_j$$

This means that the  $(i, j)^{th}$  element of the outer product matrix is equal to the product of the  $i^{th}$  element of u and the  $j^{th}$  element of v.

As for the question:  $a \otimes b = \begin{bmatrix} -3 & 0 & 3 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

2. [M] Give an example of how the outer product can be useful in ML.

The outer product matrix is useful in various mathematical and scientific applications, including linear algebra, signal processing, and quantum mechanics. It can be used to represent linear transformations, compute the covariance matrix of a set of observations, and represent quantum states and operations.

The outer product can be useful in machine learning for feature engineering and computing the covariance matrix.

In feature engineering, the outer product can be used to create new features from existing features. For example, consider a dataset of images, where each image is represented as a vector of pixel values. The outer product of the pixel vector with itself can be used to create a new feature that captures the correlation between adjacent pixels. This new feature can be used to improve the performance of a machine learning model on image classification tasks.

In computing the covariance matrix, the outer product is used to compute the outer product of a dataset with its transpose. The resulting matrix represents the covariance matrix of the dataset, which captures the relationship between the different features in the dataset. The covariance matrix is useful in machine learning for tasks such as principal component analysis (PCA) and linear discriminant analysis (LDA), which involve finding the most informative features for a given task.

To perform PCA, we first compute the covariance matrix of the data. Then, we find the eigenvectors and eigenvalues of the covariance matrix. We sort the eigenvectors in decreasing order of their corresponding eigenvalues, and select the top  $k$  eigenvectors to define the new coordinate system. We then project the data onto the new coordinate system to obtain the lower-dimensional representation of the data.

### 1.1.3

1. [E] What does it mean for two vectors to be linearly independent?

Two vectors are said to be linearly independent if one of them cannot be expressed as a linear combination of the other. In other words, two vectors  $u$  and  $v$  are linearly independent if there are no constants  $a$  and  $b$ , not both zero, such that:

$$au + bv = 0$$

where  $0$  is the zero vector (i.e., a vector with all its elements equal to zero).

Geometrically, two vectors are linearly independent if they do not lie on the same line in the vector space. If two vectors are linearly dependent, they lie on the same line and one can be expressed as a multiple of the other.

Linear independence is an important concept in linear algebra because it allows us to define a basis for a vector space. A basis is a set of linearly independent vectors that can be used to represent any vector in the vector space as a linear combination of the basis vectors. The number of basis vectors is called the dimension of the vector space.

For example, in a two-dimensional vector space, any two linearly independent vectors form a basis for the space. Similarly, in a three-dimensional vector space, any three linearly independent vectors form a basis for the space. Linear independence is also important in solving systems of linear equations, where it allows us to determine when a system has a unique solution or not.

### 1.1.4

1. [M] Given two sets of vectors  $A = a_1, a_2, a_3, \dots, a_n$  and  $B = b_1, b_2, b_3, \dots, b_m$ . How do you check that they share the same basis?

Two vectors share the same basis if they are both linear combinations of the same set of linearly independent vectors.

In summary, to check if two vectors share the same basis, we can compute the row echelon form of a matrix whose columns are the two vectors and check if it has a row of zeros or not. If it does not have a row of zeros, then the two vectors are linearly independent and share the same basis if and only if the row echelon form has two non-zero rows.

### 1.1.5

1. [M] Given  $n$  vectors, each of  $d$  dimensions. What is the dimension of their span?

The span of a vector (or a set of vectors) is the set of all possible linear combinations of those vectors.

More formally, given a set of vectors  $v_1, v_2, \dots, v_n$  in a vector space  $V$ , the span of these vectors is the set of all linear combinations of these vectors, that is, the set of all vectors that can be expressed in the form:



$$a_1 v_1 + a_2 v_2 + \dots + a_n v_n$$

where  $a_1, a_2, \dots, a_n$  are scalars. The span of these vectors is denoted by  $\text{Span}\{v_1, v_2, \dots, v_n\}$ .

In simpler terms, the span of a vector or a set of vectors is the set of all possible vectors that can be obtained by scaling and adding these vectors in different ways.

Geometrically, the span of a set of vectors is the smallest subspace of the vector space  $V$  that contains all the vectors in the set. The span of a single non-zero vector is a one-dimensional subspace, while the span of two or more vectors can be higher-dimensional depending on their linear independence.

The dimension of the span of  $n$  vectors in  $d$  dimensions can be at most  $d$ , since any vector in the span of these  $n$  vectors can be expressed as a linear combination of them, and each vector in the combination can have at most  $d$  non-zero entries. In other words, the number of linearly independent vectors that can be formed from  $n$  vectors in  $d$  dimensions is at most  $d$ .

If the  $n$  vectors are linearly independent, then the dimension of their span is exactly  $n$ . If they are linearly dependent, then the dimension of their span is less than  $n$  and can be found by computing the rank of the matrix whose columns are the  $n$  vectors. The rank of this matrix is the maximum number of linearly independent columns, which is also the dimension of the column space of the matrix, which is the same as the dimension of the span of the  $n$  vectors.

In summary, the dimension of the span of  $n$  vectors in  $d$  dimensions can be at most  $d$  and is equal to  $n$  if the vectors are linearly independent. Otherwise, it is less than  $n$  and can be found by computing the rank of the matrix whose columns are the  $n$  vectors.

### 1.1.6 Norms and Metrics

1. [E] What's a norm? What is  $L_0, L_1, L_2, L_{\text{norm}}$ ?

A norm is a mathematical function that assigns a positive scalar value to a vector, matrix, or tensor, and satisfies certain properties. In the context of machine learning and data analysis, norms are often used to measure the size or magnitude of a mathematical object.

Some common examples of norms include:

$L_0$  norm: The  $L_0$  norm of a vector is the number of non-zero elements in the vector.

$L_1$  norm: The  $L_1$  norm of a vector is the sum of the absolute values of its elements. Mathematically, for a vector  $x = [x_1, x_2, \dots, x_n]$ , the  $L_1$  norm is defined as:

$$\|x\|_1 = |x_1| + |x_2| + \dots + |x_n|$$

$L_2$  norm: The  $L_2$  norm of a vector is the square root of the sum of the squares of its elements. Mathematically, for a vector  $x = [x_1, x_2, \dots, x_n]$ , the  $L_2$  norm is defined as:

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

$L_p$  norm: The  $L_p$  norm of a vector is the  $p^{\text{th}}$  root of the sum of the  $p^{\text{th}}$  powers of its elements. Mathematically, for a vector  $x = [x_1, x_2, \dots, x_n]$ , the  $L_p$  norm is defined as:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{1/p}$$

Here,  $p$  is a positive real number greater than or equal to 1.

The  $L_2$  norm is the most commonly used norm in machine learning, especially in optimization problems such as regression, classification, and clustering. It is also known as the Euclidean norm and represents the distance of a vector from the origin in the Euclidean space. The  $L_1$  norm is commonly used in sparse modeling, while the  $L_0$  norm is used in the context of compressive sensing. The  $L_p$  norm is a generalized form that can be used to measure the size or magnitude of a vector according to different criteria.

2. [M] How do norm and metric differ? Given a norm, make a metric. Given a metric, can we make a norm?

A norm is a function that assigns a non-negative value to a vector or a matrix, and satisfies certain properties such as non-negativity, homogeneity, and the triangle inequality. A norm is typically used to measure the magnitude or size of a vector or a matrix.

On the other hand, a metric is a function that assigns a non-negative value to a pair of elements in a set, and satisfies certain properties such as non-negativity, symmetry, and the triangle inequality. A metric is typically used to define a notion of distance or similarity between elements in a set.

Given a norm, we can construct a metric as follows:

Let  $X$  be a set of elements and let  $\|\cdot\|$  be a norm defined on  $X$ . Then, the function  $d : X \times X \rightarrow \mathbb{R}$  defined by

$$d(x, y) = \|x - y\|$$

is a metric on  $X$ . Here,  $x$  and  $y$  are elements of  $X$  and  $\|\cdot\|$  denotes the norm defined on  $X$ .

Given a metric, we cannot always construct a norm. A metric must satisfy certain properties, such as symmetry and the triangle inequality, that do not necessarily hold for a norm. In particular, a metric must be defined on a set, while a norm is defined on a vector space.

However, there are cases where a metric can be used to induce a norm on a vector space. For example, given a metric  $d : X \times X \rightarrow \mathbb{R}$  on a set  $X$ , we can define a norm  $\|\cdot\|$  on the vector space  $V$  of real-valued functions on  $X$  by:

$$\|f\| = \sup |f(x)| : x \in X$$

where  $\sup$  denotes the supremum or least upper bound. This norm is called the supremum norm or the  $L^\infty$  norm and it is induced by the metric  $d(x, y) = |x - y|$  on  $X$ . In general, the process of inducing a norm from a metric is called the norming process.

## 1.2 Algebra and (little) Calculus: Matrices

### 1.2.1

1. [E] Why do we say that matrices are linear transformations?

Matrices can be used to represent linear transformations because they satisfy the properties of linearity, which are:

Additivity: If  $T$  is a linear transformation and  $u$  and  $v$  are vectors in the domain of  $T$ , then  $T(u + v) = T(u) + T(v)$ .

Homogeneity: If  $T$  is a linear transformation and  $c$  is a scalar, then  $T(cu) = cT(u)$  for any vector  $u$  in the domain of  $T$ .

These properties can also be expressed in terms of matrices as follows:

Additivity: If  $A$  is a matrix that represents a linear transformation  $T$ , and  $u$  and  $v$  are vectors in the domain of  $T$ , then  $A(u + v) = A(u) + A(v)$ .

Homogeneity: If  $A$  is a matrix that represents a linear transformation  $T$ , and  $c$  is a scalar, then  $A(cu) = cA(u)$  for any vector  $u$  in the domain of  $T$ .

Therefore, matrices can be used to represent linear transformations because they preserve the properties of linearity. When we apply a matrix to a vector, we are essentially applying a linear transformation to that vector. In fact, every linear transformation between finite-dimensional vector spaces is defined by a unique matrix that performs the transformation. That is, if we have two vector spaces in, say  $\mathbb{R}_m$  and  $\mathbb{R}_n$ , and we have a linear transformation  $T$  mapping vectors between them, then there exists a single unique matrix  $A_T$  that performs  $T$ 's mapping. That is, where  $T(x) = A_T x$ . This matrix is called the standard matrix of  $T$ . Moreover, any linear transformation can be represented by a matrix, and any matrix can be interpreted as a linear transformation, that means, there is a one-to-one mapping between linear transformations and matrices. This is why we say that matrices are linear transformations.

### 1.2.2

1. [E] What's the inverse of a matrix? Do all matrices have an inverse? Is the inverse of a matrix always unique?

The inverse of a matrix is a matrix that, when multiplied by the original matrix, results in the identity matrix. In other words, if  $A$  is a square matrix, its inverse  $A^{-1}$  is another square matrix such that  $AA^{-1} = A^{-1}A = I$ , where  $I$  is the identity matrix. The mathematical formula for calculating the inverse of a matrix  $A$  is  $Adj(A)/|A|$

The inverse of a matrix can only be found for square matrices that are invertible, i.e., matrices whose determinant is not zero. If a matrix is not invertible, it is said to be singular.

No, the inverse of a matrix is not always unique. However, if a matrix has an inverse, then it is unique.

Let  $A$  be a square matrix that has an inverse, denoted by  $A^{-1}$ . Suppose that there exists another matrix  $B$  such that  $AB = BA = I$ , where  $I$  is the identity matrix. Then,  $B$  is also an inverse of  $A$ . To see this, consider:

$$AB = I \text{ (by assumption)}$$

$$B(AB) = B \text{ (multiply both sides by B)}$$

$$(BA)B = B \text{ (associative property of matrix multiplication)}$$

$$IB = B \text{ (since A has an inverse, AB = BA = I)}$$

$$B = B$$

So, we have shown that B is also an inverse of A. However, this does not mean that B and  $A^{-1}$  are the same matrix. In fact, B could be a different matrix altogether.

In summary, if a matrix has an inverse, then that inverse is unique. However, not all matrices have inverses, and if a matrix does have an inverse, there may be other matrices that satisfy the definition of an inverse, but they are not necessarily equal to the unique inverse.

### 1.2.3

1. [E] What does the determinant of a matrix represent?

The determinant of a square matrix is a single number that can be related to the area or volume of a region. In particular, the determinant of a matrix reflects how the linear transformation associated with the matrix can scale or reflect objects. In short, the determinant of a matrix is the signed factor by which areas are scaled by this matrix. If the sign is negative the matrix reverses orientation. A good visual explanation can be found [here](#).

### 1.2.4

1. [E] What happens to the determinant of a matrix if we multiply one of its rows by a scalar?

If we multiply one of the rows of a square matrix by a scalar k, the determinant of the resulting matrix is k times the determinant of the original matrix. More generally, if we perform an elementary row operation on a matrix (such as multiplying a row by a scalar, adding a multiple of one row to another, or swapping two rows), the determinant of the resulting matrix is related to the determinant of the original matrix as follows:

- If we multiply one row of the matrix by a scalar k, then the determinant of the resulting matrix is k times the determinant of the original matrix.
- If we add a multiple of one row to another row, the determinant of the resulting matrix is equal to the determinant of the original matrix.
- If we swap two rows of the matrix, the determinant of the resulting matrix is equal to the negative of the determinant of the original matrix.

### 1.2.5

1. [M] A 4x4 matrix has four eigenvalues 3, 3, 2, -1. What can we say about the trace and the determinant of this matrix?

An eigenvalue of a square matrix A is a scalar  $\lambda$  such that there exists a non-zero vector x, called the eigenvector, that satisfies the following equation:

$$Ax = \lambda x$$

In other words, when a square matrix A is multiplied by an eigenvector x, the result is a scaled version of x, where the scaling factor is the eigenvalue  $\lambda$ .

To find the eigenvalues of a matrix, we need to solve the characteristic equation, which is defined as  $\det(A - \lambda I) = 0$ , where  $\det$  is the determinant of the matrix, I is the identity matrix, and  $\lambda$  is the eigenvalue we want to find. The solutions to this equation give us the eigenvalues of the matrix. Once we have the eigenvalues, we can find the corresponding eigenvectors by solving the equation  $(A - \lambda I)x = 0$ , where x is the eigenvector we want to find.

The trace and the determinant of a square matrix are related to its eigenvalues. In particular, the trace of a matrix is equal to the sum of its eigenvalues, and the determinant is equal to the product of its eigenvalues.

For the given matrix, the trace is 7 while the determinant is -18.

### 1.2.6

1. [M] Given the following matrix:

$$\begin{bmatrix} 1 & 4 & -2 \\ -1 & 3 & 2 \\ 3 & 5 & -6 \end{bmatrix}$$

Without explicitly using the equation for calculating determinants, what can we say about this matrix's determinant?

We can see in the matrix that:  $C_3 = -2C_1$ . So while calculating determinant, we can reduce it by canceling proportional columns by performing  $C_3 \rightarrow C_3 + 2C_1$  which in turn makes  $C_3 = 0$  and hence the determinant of this matrix will be zero.

### 1.2.7

1. The covariance matrix  $A^T A$  and the Gram matrix  $AA^T$  are two different matrices that can be obtained from a given matrix A.

The covariance matrix  $A^T A$  is obtained by multiplying the transpose of A by A. Specifically, for an n x m matrix A, the covariance matrix  $A^T A$  is an  $m \times m$  matrix given by:

$$A^T A = (A^T)A$$

The elements of the covariance matrix  $A^T A$  measure the covariance between the columns of A. In other words, the  $(i, j)^{th}$  element of  $A^T A$  measures the covariance between the  $i^{th}$  and  $j^{th}$  columns of A.

On the other hand, the Gram matrix  $AA^T$  is obtained by multiplying A by the transpose of A. Specifically, for an n x m matrix A, the Gram matrix  $AA^T$  is an n x n matrix given by:

$$AA^T = A(A^T)$$

The elements of the Gram matrix  $AA^T$  measure the dot product between the rows of A. In other words, the  $(i, j)^{th}$  element of  $AA^T$  measures the dot product between the  $i^{th}$  and  $j^{th}$  rows of A.

In summary, the covariance matrix  $A^T A$  and the Gram matrix  $AA^T$  are different matrices that capture different types of relationships between the rows and columns of the matrix A. The covariance matrix  $A^T A$  captures the covariance between the columns of A, while the Gram matrix  $AA^T$  captures the dot product between the rows of A.

### 1.2.8

Given  $A \in \mathbb{R}^{n \times m}$  and  $b \in \mathbb{R}^n$

1. [M] Find x such that:  $Ax=b$
2. [E] When does this have a unique solution?
3. [M] Why is it when A has more columns than rows,  $Ax=b$  has multiple solutions?
4. [M] Given a matrix A with no inverse. How would you solve the equation  $Ax=b$ ? What is the pseudo-inverse and how to calculate it?

### 1.2.9

Derivative is the backbone of gradient descent.

1. [E] What does derivative represent?

In mathematics, the derivative represents the rate at which a function changes with respect to its input variable. Geometrically, it represents the slope of the tangent line to the graph of the function at a particular point.

In other words, the derivative of a function  $f(x)$  at a point  $x=a$  is defined as the limit of the ratio of the change in the value of  $f(x)$  over a small change in x, as that change in x approaches zero:

$$f'(a) = \lim_{(\Delta x \rightarrow 0)} \frac{[f(a + \Delta x) - f(a)]}{\Delta x}$$

This limit gives us the instantaneous rate of change of the function at  $x=a$ , and it is represented by the symbol  $f'(a)$  or  $dy/dx|_{x=a}$ , where  $dy/dx$  represents the derivative of y with respect to x.

2. [M] What's the difference between derivative, gradient, and Jacobian?

Derivative, gradient, and Jacobian are related concepts in mathematics, but they have different meanings and uses.

**Derivative:** The derivative of a function of one variable represents the rate of change of the function with respect to that variable at a specific point. For example, the derivative of a function  $f(x)$  at  $x=a$  is denoted by  $f'(a)$  and represents the slope of the tangent line to the graph of the function at the point  $(a, f(a))$ . The derivative can also be extended to functions of multiple variables, in which case it is known as a partial derivative.

**Gradient:** The gradient is a vector that represents the direction and magnitude of the steepest ascent of a function at a specific point. It is defined as the vector of partial derivatives of the function with respect to each variable. For example, if we have a function  $f(x,y,z)$ , the gradient at the point  $(a,b,c)$  is given by the vector  $[\partial f/\partial x(a,b,c), \partial f/\partial y(a,b,c), \partial f/\partial z(a,b,c)]$ . The gradient is often used in optimization problems to find the maximum or minimum value of a function.

**Jacobian:** The Jacobian is a matrix of partial derivatives that describes the local behavior of a function of several variables. It is used to calculate the change in variables of a system of equations or functions. For example, if we have a system of equations  $f(x,y) = 0$  and  $g(x,y) = 0$ , the Jacobian matrix at the point  $(a,b)$  is given by the matrix  $[\partial f/\partial x(a,b), \partial f/\partial y(a,b); \partial g/\partial x(a,b), \partial g/\partial y(a,b)]$ . The Jacobian is used in many areas of mathematics and physics, including differential geometry, control theory, and fluid mechanics.

In summary, while all three concepts deal with the rates of change of a function, derivative focuses on a single variable, gradient provides a vector representation of the steepest ascent, and Jacobian is a matrix that describes the local behavior of a function of several variables.

### 1.2.10

1. [H] Say we have the weights  $w \in \mathbb{R}^{d \times m}$  and a mini-batch  $x$  of  $n$  elements, each element is of the shape  $1 \times d$  so that  $x \in \mathbb{R}^{n \times d}$ . We have the output  $y = f(x; w) = xw$ . What's the dimension of the Jacobian  $\partial y / \partial x$ ?

The output  $y$  is a matrix of size  $n \times m$ , where each element  $y_{i,j}$  is the dot product of the  $i$ th row of  $x$  ( $1 \times d$ ) and the  $j$ th column of  $w$  ( $d \times 1$ ). Therefore, the output  $y$  is a function of the input  $x$  and the weights  $w$ .

The Jacobian matrix of  $y$  with respect to  $x$ , denoted as  $\partial y / \partial x$ , is a matrix of size  $(nm) \times (nd)$ , where each element  $(i,j)$  corresponds to the partial derivative of the  $i$ th element of  $y$  with respect to the  $j$ th element of  $x$ .

In this case, we have:

$$y_{i,j} = \sum_{k=1}^d x_{i,k} w_{k,j}$$

So the partial derivative of  $y_{i,j}$  with respect to  $x_{i,k}$  is:

$$\frac{\partial y_{i,j}}{\partial x_{i,k}} = w_{k,j}$$

Therefore, the Jacobian matrix of  $y$  with respect to  $x$  is:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} w_{1,1} & w_{2,1} & \dots & w_{d,1} & 0 & \dots & 0 \\ w_{1,2} & w_{2,2} & \dots & w_{d,2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ w_{1,m} & w_{2,m} & \dots & w_{d,m} & 0 & \dots & 0 \end{bmatrix}$$

where the first  $d$  columns correspond to the partial derivatives with respect to the input  $x$ , and the remaining  $(n-1) \times m$  columns are zero, since the output for each element of the mini-batch is independent of the others.

Therefore, the dimension of the Jacobian  $\partial y / \partial x$  is  $(nm) \times (nd)$ , or equivalently,  $nm$  rows and  $nd$  columns.

### 1.2.11

1. [H] Given a very large symmetric matrix  $A$  that doesn't fit in memory, say  $A \in \mathbb{R}^{1M \times 1M}$  and a function  $f$  that can quickly compute  $f(x) = Ax$  for  $x \in \mathbb{R}^{1M}$ . Find the unit vector  $x$  so that  $x^T A x$  is minimal.

## 1.3 Algebra and (little) Calculus: Dimensionality Reduction

### 1.3.1

[E] Why do we need dimensionality reduction?

Dimensionality reduction is the process of reducing the number of variables or features in a dataset while still retaining as much of the original information as possible. We need dimensionality reduction for several reasons:

1. Reduce computational complexity: High-dimensional datasets can be computationally expensive and time-consuming to process. Dimensionality reduction helps to reduce the number of features, making the computations faster and more efficient.
2. Avoid overfitting: Overfitting occurs when a model fits too closely to the training data, leading to poor performance on new, unseen data. Dimensionality reduction can help to avoid overfitting by reducing the number of features that the model has to fit to.
3. Visualization: It is difficult to visualize high-dimensional data, and reducing the number of dimensions can make it easier to create visual representations of the data.
4. Interpretation: High-dimensional data can be difficult to interpret, and reducing the number of dimensions can make it easier to understand the relationships between variables.
5. Feature engineering: Feature engineering is the process of selecting and transforming variables in a dataset to improve the performance of a model. Dimensionality reduction can be a useful tool in feature engineering, as it can help to identify the most important features in a dataset.

Overall, dimensionality reduction is a useful tool for improving the efficiency and effectiveness of machine learning models, especially when working with high-dimensional datasets.

### 1.3.2

[E] Eigendecomposition is a common factorization technique used for dimensionality reduction. Is the eigendecomposition of a matrix always unique?

The eigendecomposition of a matrix is not always unique. A matrix may have multiple sets of eigenvectors and eigenvalues that satisfy the equation  $Ax = \lambda x$ , where  $A$  is the matrix,  $\lambda$  is the eigenvalue, and  $x$  is the eigenvector.

However, if a matrix  $A$  is diagonalizable (meaning that it can be written as  $A = PDP^{-1}$ , where  $P$  is a matrix of eigenvectors and  $D$  is a diagonal matrix of eigenvalues), then the eigendecomposition is unique up to the order of the eigenvalues and corresponding eigenvectors.

In some cases, a matrix may not be diagonalizable, and the eigendecomposition may not be unique. For example, if a matrix has repeated eigenvalues, then there may be multiple eigenvectors corresponding to each eigenvalue, and different choices of eigenvectors can result in different eigendecompositions.

In summary, while the eigendecomposition of a matrix is not always unique, it can be unique if the matrix is diagonalizable, but if not, then multiple eigendecompositions may exist.

### 1.3.3

[M] Eigenvalues and eigenvectors are important mathematical concepts that have a wide range of applications in various fields. Here are some applications of eigenvalues and eigenvectors:

1. Principal Component Analysis (PCA): PCA is a popular technique for dimensionality reduction, and it relies on the eigendecomposition of the covariance matrix of a dataset to find the principal components, which are linear combinations of the original variables that capture the most variation in the data.
2. Image processing and computer vision: Eigenvectors are used in techniques such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) for image compression, feature extraction, and pattern recognition.
3. Markov chains: Eigenvalues and eigenvectors are used to study the long-term behavior of Markov chains, which are mathematical models that describe a sequence of events where the probability of each event depends only on the current state.
4. Graph theory: Eigenvectors and eigenvalues are used in graph theory to study the structure and connectivity of networks, where the eigenvectors of the adjacency matrix of a graph can be used to identify important nodes or communities in the network.

Overall, eigenvalues and eigenvectors have a wide range of applications in various fields, including data analysis, image processing, and graph theory.

### 1.3.4

[M] We want to do PCA on a dataset of multiple features in different ranges. For example, one is in the range 0-1 and one is in the range 10 - 1000. Will PCA work on this dataset?

PCA can work on datasets with features in different ranges, but it is often recommended to standardize or normalize the features before performing PCA. This is because PCA is sensitive to the scale of the features, and features with larger ranges or variances can dominate the analysis and skew the results.

In the example you provided, one feature is in the range of 0-1 and the other is in the range of 10-1000. Since the ranges of the two features are quite different, they may have different scales and variances. Therefore, it would be appropriate to normalize or standardize the features before performing PCA.

One common method to normalize or standardize features is to subtract the mean and divide by the standard deviation. This will ensure that each feature has zero mean and unit variance, which can help prevent any one feature from dominating the analysis. Once the features are normalized, PCA can be performed as usual.

In summary, PCA can work on datasets with features in different ranges, but it is often recommended to standardize or normalize the features before performing PCA to ensure that each feature is given equal weight in the analysis.

### 1.3.5

[H] Under what conditions can one apply eigendecomposition? What about SVD?

Eigendecomposition is a factorization technique that can only be applied to certain types of matrices. Specifically, eigendecomposition can be applied to square matrices that are diagonalizable, meaning that they can be transformed into a diagonal matrix by a change of basis.

Here are the conditions under which eigendecomposition can be applied:

The matrix must be square: Eigendecomposition can only be applied to square matrices.

The matrix must be diagonalizable: This means that the matrix can be transformed into a diagonal matrix by a change of basis. A matrix is diagonalizable if it has  $n$  linearly independent eigenvectors, where  $n$  is the dimension of the matrix.

The eigenvectors must be linearly independent: The eigenvectors corresponding to distinct eigenvalues must be linearly independent, meaning that no eigenvector can be expressed as a linear combination of the others.

The matrix must have a complete set of eigenvectors: This means that the matrix has  $n$  linearly independent eigenvectors that span the entire vector space of the matrix.

If these conditions are met, then eigendecomposition can be used to factorize the matrix into the product of the eigenvectors and eigenvalues of the matrix. The eigenvectors represent the directions in which the matrix scales or rotates, while the eigenvalues represent the amount of scaling or rotation in each direction.

It's important to note that not all matrices meet these conditions, and in some cases, other factorization techniques such as Singular Value Decomposition (SVD) may be more appropriate since it is a factorization technique that can be applied to any matrix, regardless of its size, shape, or properties. Additionally, eigendecomposition can be numerically unstable for certain matrices, so numerical methods may be needed to improve the accuracy of the factorization.

1. What is the relationship between SVD and eigendecomposition?

	Singular Value Decomposition (SVD)	Eigendecomposition
Definition	Factorizes a matrix into three components: $U$ , $\Sigma$ , and $V$ , where $U$ and $V$ are unitary matrices and $\Sigma$ is a diagonal matrix containing the singular values of the matrix.	Factorizes a square matrix into the product of its eigenvectors and eigenvalues.
Applicability	Can be applied to any matrix, regardless of its size, shape, or properties.	Can only be applied to diagonalizable square matrices.
Components	$U$ and $V$ are orthogonal matrices that describe the rotation and scaling of the matrix. $\Sigma$ is a diagonal matrix containing the singular values of the matrix.	$Q$ is the matrix of eigenvectors, $\Lambda$ is the diagonal matrix of eigenvalues.
Relationship	The singular values of a matrix $A$ are the square roots of the eigenvalues of $A^T A$ . The columns of $U$ are the eigenvectors of $A A^T$ . The columns of $V$ are the eigenvectors of $A^T A$ .	The SVD of a matrix $A$ can be used to derive its eigendecomposition, and vice versa. The relationship is established by the diagonalization of $A^T A$ or $A A^T$ .

Table 1.1: Comparison between Singular Value Decomposition (SVD) and Eigendecomposition

2. What's the relationship between PCA and SVD?

	Principal Component Analysis (PCA)	Singular Value Decomposition (SVD)
Definition	Linear transformation technique that identifies a new set of orthogonal variables (principal components) that maximize the variance of the original data.	Factorizes a matrix into three components: $U$ , $\Sigma$ , and $V$ , where $U$ and $V$ are unitary matrices and $\Sigma$ is a diagonal matrix containing the singular values of the matrix.
Objective	Reduce the dimensionality of the data by projecting it onto a lower-dimensional space while retaining the maximum amount of variation in the original data.	Extract the underlying structure of the data by identifying the principal components that capture the maximum amount of variation in the data.
Components	Principal components, eigenvalues, eigenvectors.	$U$ and $V$ are orthogonal matrices that describe the rotation and scaling of the matrix. $\Sigma$ is a diagonal matrix containing the singular values of the matrix.
Calculation	PCA is typically computed via eigen-decomposition of the covariance matrix or singular value decomposition of the data matrix after centering and scaling.	SVD can be computed for any matrix, including the covariance matrix of the data.
Relationship	The principal components in PCA are the eigenvectors of the covariance matrix. The eigenvalues of the covariance matrix are the variances of the principal components.	The principal components in PCA are also the columns of $U$ in the SVD of the data matrix. The singular values in the SVD are related to the eigenvalues of the covariance matrix.

Table 1.2: Comparison between Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)

### 1.3.6

[H] How does t-SNE (T-distributed Stochastic Neighbor Embedding) work? Why do we need it?

T-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm used for data visualization and dimensionality reduction. It is particularly useful for visualizing high-dimensional data in a lower-dimensional space.

The basic idea behind t-SNE is to map the high-dimensional data points to a low-dimensional space while preserving the pairwise similarities between them. It does so by modeling the high-dimensional similarities using a Gaussian probability distribution and the low-dimensional similarities using a Student's t-distribution. The algorithm then minimizes the difference between the two distributions using gradient descent.

The algorithm works in the following way:

1. Compute pairwise similarities between data points in the high-dimensional space.
2. Initialize the low-dimensional points randomly.
3. Compute pairwise similarities between the low-dimensional points using a Student's t-distribution.
4. Compute the difference between the high-dimensional and low-dimensional pairwise similarity distributions.
5. Update the positions of the low-dimensional points to minimize this difference using gradient descent.
6. Repeat steps 3-5 until convergence.

One of the main advantages of t-SNE is that it is able to preserve the local structure of the data, meaning that data points that are close together in the high-dimensional space will also be close together in the low-dimensional space. This makes it particularly useful for visualizing clusters or groups of data points.

t-SNE is often used in applications such as image recognition, natural language processing, and genomics research, where high-dimensional data needs to be visualized and analyzed. It is especially useful for exploring and analyzing complex, nonlinear relationships in the data, which can be difficult to capture using other linear dimensionality reduction techniques such as PCA.



## 1.4 Algebra and (little) Calculus: Calculus and Convex Optimization

## 1.5 Probability and Statistics: Probability

## 1.6 Probability and Statistics: Stats

### 1.6.1

[E] Explain frequentist vs. Bayesian statistics.

Frequentist and Bayesian statistics are two different approaches to statistical inference, which is the process of drawing conclusions about a population based on a sample of data.

Frequentist statistics focuses on the long-run behavior of random processes, where probabilities are interpreted as the frequency of events occurring over many repetitions of an experiment. In this framework, statistical inference is based on the properties of the sample data, and the goal is to make statements about the population parameters (such as the mean or the proportion) with a certain level of confidence or probability. Hypothesis testing, confidence intervals, and p-values are common frequentist tools for statistical inference.

On the other hand, Bayesian statistics is a more subjective approach that relies on prior knowledge or beliefs about the population parameters. In this framework, probabilities are interpreted as degrees of belief or uncertainty about a parameter, and statistical inference is based on Bayes' theorem, which updates the prior beliefs based on the observed data. The goal is to estimate the posterior distribution of the parameters, which summarizes the updated beliefs about their values. Bayesian methods can be more flexible than frequentist methods in handling complex models and incorporating prior information, but they can also be more computationally demanding and subjective.

In summary, frequentist statistics is based on the properties of the data, while Bayesian statistics incorporates prior beliefs about the parameters. Both approaches have their strengths and weaknesses and can be useful in different contexts. The choice of approach often depends on the problem at hand and the preferences of the analyst.

### 1.6.2

[E] Given the array [1, 5, 3, 2, 4, 4], find its mean, median, variance, and standard deviation.

Formulae for calculating the following are:

1. Mean: The mean is the sum of all the values in a dataset divided by the number of values (n):

$$\text{mean} = \mu = \frac{x_1 + x_2 + \dots + x_n}{n}$$

2. Median: The median is the middle value in a dataset when the data is arranged in order from smallest to largest. If there is an even number of values, the median is the average of the two middle values.
3. Mode: The mode is the value that occurs most frequently in a dataset. A dataset can have multiple modes if there are multiple values that occur with the same maximum frequency.
4. Variance: The variance measures how spread out the data is from the mean. It is the average of the squared differences between each data point and the mean:

$$\text{variance} = \frac{(x_1 - \text{mean})^2 + (x_2 - \text{mean})^2 + \dots + (x_n - \text{mean})^2}{n}$$

Note:  $x_1, x_2, \dots, x_n$  are the data points in a dataset, n is the number of data points, and mean is the average of the data points.

5. Standard deviation: The standard deviation is the square root of the variance. It measures the typical distance of each data point from the mean:

$$\text{standard deviation} = \sqrt{\text{variance}}$$

As for the array in the question [1, 5, 3, 2, 4, 4];

$$\text{mean} \rightarrow \frac{1 + 5 + 3 + 2 + 4 + 4}{6} \rightarrow 3.15$$

$$\text{median} \rightarrow [1, 2, 3, 4, 4, 5] \rightarrow \frac{3 + 4}{2} \rightarrow 3.5$$

$$\text{variance} \rightarrow \frac{(1 - 3.15)^2 + (5 - 3.15)^2 + (3 - 3.15)^2 + (2 - 3.15)^2 + (4 - 3.15)^2 + (4 - 3.15)^2}{6} \rightarrow 1.8$$

$$\text{standard deviation} \rightarrow \sqrt{1.8} \rightarrow 1.34$$

### 1.6.3

[M] When should we use median instead of mean? When should we use mean instead of median?

Whether to use the median or the mean as a measure of central tendency depends on the nature of the data and the research question.

Use median instead of mean when:

The data contains extreme outliers or skewed distributions that could heavily influence the mean value. The data is ordinal or nominal, and the concept of a mathematical average does not apply. The data is discrete, and the mean is not a possible value. For example, the median might be a better measure of central tendency for salaries in a company, where a few executives earn extremely high salaries that skew the distribution, or for test scores in a class, where the scores might be skewed due to a few students doing much better or worse than the others.

Use mean instead of median when:

The data is normally distributed or approximately symmetric around the central value. The data is continuous or interval-level, and the mean is a meaningful value. The data does not contain extreme outliers that would heavily influence the mean value. For example, the mean might be a better measure of central tendency for heights in a population, where the distribution is approximately normal and there are no extreme outliers, or for weights of packages in a shipment, where the values are continuous and the mean is a meaningful value.

In summary, the choice of whether to use the median or mean as a measure of central tendency depends on the nature of the data, the presence of outliers or skewness, and the research question at hand.

### 1.6.4

[M] What is a moment of function? Explain the meanings of the zeroth to fourth moments.

In statistics, the concept of moments is used to describe various properties of a probability distribution. Specifically, the moment of a function is a numerical measure of the shape, position, and spread of a probability distribution. The moment of a function is usually calculated with respect to a specific point or value, often the mean or the origin.

The zeroth moment of a function is simply the value of the function at a specific point. For example, the zeroth moment of a probability distribution is the probability density or mass function evaluated at a specific point.

The first moment of a function is the mean or the expected value of the function, which is the center of the distribution. It measures the average value of the function with respect to the distribution.

The second moment of a function is the variance, which measures the spread of the distribution around the mean. The square root of the variance is the standard deviation.

The third moment of a function is the skewness, which measures the degree of asymmetry of the distribution around the mean. A positive skewness indicates a longer tail on the right side of the distribution, while a negative skewness indicates a longer tail on the left side.

The fourth moment of a function is the kurtosis, which measures the degree of peakedness or flatness of the distribution. A high kurtosis indicates a sharper peak and heavier tails, while a low kurtosis indicates a flatter peak and lighter tails.

### 1.6.5

[M] Are independence and zero covariance the same? Give a counterexample if not.

The covariance between two random variables  $X$  and  $Y$  is given by the formula:

$$\text{cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$$

where  $E[X]$  and  $E[Y]$  are the expected values of  $X$  and  $Y$ , respectively. This formula measures the degree to which  $X$  and  $Y$  vary together or co-vary. If the covariance is positive, then  $X$  and  $Y$  tend to increase or decrease together. If the covariance is negative, then  $X$  tends to increase when  $Y$  decreases and vice versa. If the covariance is zero, then  $X$  and  $Y$  are uncorrelated.

Another way to express the covariance is:

$$\text{cov}(X, Y) = E[XY] - E[X]E[Y]$$

where  $E[XY]$  is the expected value of the product  $XY$ . This formula shows that the covariance can also be interpreted as the difference between the expected value of the product  $XY$  and the product of the expected values  $E[X]$  and  $E[Y]$ .

$\text{cov}(X, Y) = 0$  does not guarantee  $X$  and  $Y$  are independent. But if they are independent, their covariance must be 0. Take a random variable  $X$  with  $E[X] = 0$  and  $E[X^3] = 0$ , e.g. normal random variable with zero mean. Take  $Y = X^2$ . It is clear that  $X$  and  $Y$  are related, but

$$\text{cov}(X, Y) = E[XY] - E[X].E[Y] = E[X^3] = 0$$

### 1.6.6

[E] Suppose that you take 100 random newborn puppies and determine that the average weight is 1 pound with the population standard deviation of 0.12 pounds. Assuming the weight of newborn puppies follows a normal distribution, calculate the 95% confidence interval for the average weight of all newborn puppies.

To calculate the 95% confidence interval for the average weight of all newborn puppies, we can use the following formula:

$$CI = \bar{x} \pm z_{\alpha/2} * \frac{\sigma}{\sqrt{n}}$$

where  $\bar{x}$  is the sample mean,  $\sigma$  is the population standard deviation,  $n$  is the sample size,  $z_{\alpha/2}$  is the  $z$ -score for the desired level of confidence (95% in this case), and  $\sqrt{n}$  is the square root of the sample size. Substituting the given values, we get:

$$CI = 1 \pm 1.96 * \frac{0.12}{\sqrt{100}}$$

$$CI = 1 \pm 0.02448$$

$$CI = [0.97552, 1.02448]$$

Therefore, we can say with 95% confidence that the true average weight of all newborn puppies is between 0.97552 pounds and 1.02448 pounds.

### 1.6.7

[M] Suppose that we examine 100 newborn puppies and the 95% confidence interval for their average weight is [0.9, 1.1] pounds. Which of the following statements is true?

1. Given a random newborn puppy, its weight has a 95% chance of being between 0.9 and 1.1 pounds.
2. If we examine another 100 newborn puppies, their mean has a 95% chance of being in that interval.
3. We're 95% confident that this interval captured the true mean weight.

The correct statement is: "We're 95% confident that this interval captured the true mean weight."

Explanation:

The 95% confidence interval of [0.9, 1.1] means that we are 95% confident that the true mean weight of all newborn puppies is within this interval. It does not mean that a random newborn puppy has a 95% chance of weighing between 0.9 and 1.1 pounds, nor does it mean that if we examine another 100 newborn puppies, their mean weight will necessarily be within this interval with 95% probability.

Instead, it means that if we were to repeat the same study many times, each time taking a random sample of 100 newborn puppies and constructing a 95% confidence interval, then about 95% of these intervals would contain the true mean weight of all newborn puppies. The statement "We're 95% confident that this interval captured the true mean weight" reflects this idea of confidence in the estimation of the true mean weight based on the given sample.

### 1.6.8

[H] Suppose we have a random variable  $X$  supported on  $[0, 1]$  from which we can draw samples. How can we come up with an unbiased estimate of the median of  $X$ ?

To come up with an unbiased estimate of the median of  $X$ , we can use the following procedure:

1. Draw a sample of size  $n$  from  $X$ .
2. Sort the sample in ascending order, so that  $x_1 \leq x_2 \leq \dots \leq x_n$ .
3. If  $n$  is odd, the median is the middle value of the sorted sample, i.e.  $m = x_{(n+1)/2}$ .
4. If  $n$  is even, the median is the average of the two middle values, i.e.  $m = (x_{n/2} + x_{n/2+1})/2$ .

Repeat steps 1-4 for a large number of independent samples, and take the average of the resulting medians as the unbiased estimate of the true median of  $X$ . This procedure is unbiased because it uses the sample medians to estimate the true median of  $X$ , and the averaging step in step 5 ensures that the estimates are unbiased on average over many samples. Note that this procedure assumes that we can draw independent samples from  $X$  and that  $X$  has a well-defined median.

### 1.6.9

[H] Can correlation be greater than 1? Why or why not? How to interpret a correlation value of 0.3?

No, the correlation between two variables cannot be greater than 1.

The correlation coefficient is a measure of the strength and direction of the linear relationship between two variables, and it always takes on values between -1 and 1. A correlation of 1 indicates a perfect positive linear relationship, meaning that as one variable increases, the other variable increases proportionally. A correlation of -1 indicates a perfect negative linear relationship, meaning that as one variable increases, the other variable decreases proportionally. A correlation of 0 indicates no linear relationship between the two variables.

A correlation value of 0.3 indicates a moderate positive linear relationship between the two variables. This means that as one variable increases, the other variable tends to increase as well, but not necessarily at the same rate. The strength of the relationship is moderate, meaning that the correlation coefficient of 0.3 suggests that there is some degree of association between the two variables, but not a strong or perfect relationship. The interpretation of the correlation coefficient also depends on the context and the nature of the variables being studied.

### 1.6.10

The weight of newborn puppies is roughly symmetric with a mean of 1 pound and a standard deviation of 0.12. Your favorite newborn puppy weighs 1.1 pounds.

1. [E] Calculate your puppy's z-score (standard score).
2. [E] How much does your newborn puppy have to weigh to be in the top 10% in terms of weight?
3. [M] Suppose the weight of newborn puppies followed a skew distribution. Would it still make sense to calculate z-scores?

To calculate the z-score for the favorite newborn puppy weighing 1.1 pounds:

$$z = (x - \mu) / \sigma$$

where  $x$  is the weight of the puppy,  $\mu$  is the mean weight of the population, and  $\sigma$  is the standard deviation of the population.

Substituting the values, we get:

$$z = (1.1 - 1) / 0.12 \approx 0.83$$

Therefore, the z-score for the favorite newborn puppy is approximately 0.83.

To find the weight of the puppy that corresponds to the top 10% in terms of weight, we can use the inverse normal distribution function (also known as the probit function) to find the z-score that corresponds to the top 10

$$z = \text{invNorm}(0.9) \approx 1.28$$

Then, we can solve for  $x$  using the z-score formula:

$$1.28 = (x - 1) / 0.12$$

$$x = 1 + 0.12 * 1.28 \approx 1.15 \text{ pounds}$$

Therefore, a newborn puppy would have to weigh approximately 1.15 pounds to be in the top 10% in terms of weight.

If the weight of newborn puppies followed a skew distribution, it would still make sense to calculate z-scores as a measure of how many standard deviations a given weight is from the mean. However, in this case, the z-score may not fully capture the shape and characteristics of the skewed distribution, and other measures such as the median and interquartile range may be more appropriate for summarizing the distribution.

## Chapter 2

# Computer Science

2.1 Algorithms

2.2 Complexity and Numerical Analysis

2.3 Data Structures



# Chapter 3

## Machine Learning Workflows

### 3.1 Basics

#### 3.1.1

[E] Explain supervised, unsupervised, weakly supervised, semi-supervised, and active learning.

Supervised learning, unsupervised learning, weakly supervised learning, semi-supervised learning, and active learning are all different types of machine learning approaches.

1. **Supervised Learning:** Supervised learning is a type of machine learning where the algorithm is trained on labeled data. In supervised learning, the algorithm is provided with input data and their corresponding output labels, and it learns to map the input to the output. The aim of supervised learning is to make accurate predictions on new, unseen data. Example: Image classification is a common example of supervised learning. In this case, the algorithm is trained on a set of images and their corresponding labels (e.g., dog, cat, etc.), and it learns to identify the objects in new, unseen images.
2. **Unsupervised Learning:** Unsupervised learning is a type of machine learning where the algorithm is trained on unlabeled data. In unsupervised learning, the algorithm is provided with input data without any corresponding output labels, and it learns to identify patterns or structure in the data. Example: Clustering is a common example of unsupervised learning. In this case, the algorithm is provided with a set of data points and it learns to group them into clusters based on their similarity.
3. **Weakly Supervised Learning:** Weakly supervised learning is a type of machine learning where the algorithm is trained on partially labeled data. In weakly supervised learning, the algorithm is provided with input data and some weak supervision signals (e.g., partial labels, noisy labels, or indirect supervision), and it learns to make predictions on new, unseen data. Example: Object detection is a common example of weakly supervised learning. In this case, the algorithm is provided with images and only the location of the object in the image is known (weak supervision signal). The algorithm then learns to detect the object in new, unseen images.
4. **Semi-Supervised Learning:** Semi-supervised learning is a type of machine learning where the algorithm is trained on a combination of labeled and unlabeled data. In semi-supervised learning, the algorithm uses the labeled data to learn a model, and then uses the unlabeled data to improve the model's accuracy. Example: Sentiment analysis is a common example of semi-supervised learning. In this case, the algorithm is trained on a small labeled dataset of positive and negative reviews, and then uses a larger unlabeled dataset to improve its accuracy.
5. **Active Learning:** Active learning is a type of machine learning where the algorithm is able to actively query the user for additional information to improve its accuracy. In active learning, the algorithm selects the most informative data points to query the user, and then uses the user's feedback to improve its accuracy. Example: Text classification is a common example of active learning. In this case, the algorithm selects the most informative documents to query the user for their labels, and then uses the user's feedback to improve its accuracy.

#### 3.1.2

Empirical risk minimization.

1. [E] What's the risk in empirical risk minimization?

Empirical risk minimization (ERM) is a popular approach in machine learning for finding a model that minimizes the average loss on a training set. However, there are several risks associated with ERM that can lead to poor generalization performance on new, unseen data:

- (a) **Overfitting:** ERM can result in overfitting, where the model fits the noise in the training data instead of the underlying patterns. This can result in poor generalization performance on new, unseen data.

- (b) Underfitting: ERM can also result in underfitting, where the model is too simple to capture the underlying patterns in the data. This can also result in poor generalization performance on new, unseen data.
- (c) Bias: ERM can be biased towards certain types of models or assumptions about the data. This can result in poor generalization performance on new, unseen data that does not conform to these assumptions.
- (d) Sampling Bias: ERM assumes that the training data is representative of the underlying distribution of the data. However, if the training data is biased or unrepresentative, the resulting model may also be biased or unrepresentative.
- (e) Label Noise: ERM assumes that the training labels are accurate and correctly reflect the true labels of the data. However, if the labels are noisy or incorrect, the resulting model may also be noisy or incorrect.

Overall, the risk in ERM is that the resulting model may not generalize well to new, unseen data due to overfitting, underfitting, bias, sampling bias, or label noise.

## 2. [E] Why is it empirical?

Empirical in empirical risk minimization (ERM) refers to the fact that the approach involves minimizing the empirical risk or empirical loss function based on the observed data.

In ERM, the goal is to find a model that minimizes the expected loss on new, unseen data. However, since the true underlying data distribution is unknown, it is impossible to directly minimize the expected loss. Instead, ERM minimizes the empirical risk, which is the average loss on the observed training data.

By minimizing the empirical risk, ERM effectively approximates the true expected loss by optimizing the model to perform well on the observed training data. The hope is that by minimizing the empirical risk, the resulting model will also perform well on new, unseen data.

Thus, the term empirical in ERM refers to the fact that the approach is based on empirical observations of the training data, rather than any prior knowledge or assumptions about the true underlying data distribution.

## 3. [E] How do we minimize that risk?

To minimize the risks associated with empirical risk minimization (ERM) and improve the generalization performance of the resulting model, there are several approaches that can be used:

- (a) Regularization: Regularization is a technique that adds a penalty term to the loss function to prevent overfitting. The penalty term encourages the model to be simpler and reduces the impact of noise in the training data. Popular regularization techniques include L1 regularization, L2 regularization, and dropout.
- (b) Cross-Validation: Cross-validation is a technique that involves splitting the training data into several subsets and using each subset in turn as a validation set to assess the performance of the model. This helps to identify overfitting and select the best hyperparameters for the model.
- (c) Data Augmentation: Data augmentation is a technique that involves generating new training data by applying random transformations to the existing data. This can help to reduce overfitting and improve the generalization performance of the model.
- (d) Ensemble Methods: Ensemble methods involve training multiple models and combining their predictions to improve the overall performance. Popular ensemble methods include bagging, boosting, and stacking.
- (e) Transfer Learning: Transfer learning is a technique that involves using a pre-trained model as a starting point for training a new model on a related task. This can help to improve the generalization performance of the new model by leveraging the knowledge learned by the pre-trained model.
- (f) Robust Data Preprocessing: Robust data preprocessing techniques, such as feature scaling, outlier removal, and imputation of missing values, can help to reduce the impact of sampling bias, label noise, and other types of data-related problems.

Overall, minimizing the risk associated with ERM requires careful selection of the model architecture, loss function, and regularization techniques, as well as careful preprocessing and sampling of the data. By taking these steps, it is possible to improve the generalization performance of the resulting model and reduce the risk of overfitting, underfitting, bias, and other types of problems.

### 3.1.3

[E] Occam's razor states that when the simple explanation and complex explanation both work equally well, the simple explanation is usually correct. How do we apply this principle in ML?



Occam's razor is a principle in philosophy and science that states that when there are multiple explanations for a phenomenon, the simplest explanation is usually the correct one. In machine learning (ML), Occam's razor can be applied in the following ways:

**Model Complexity:** When choosing between two models that have similar predictive performance, Occam's razor suggests that the simpler model is more likely to be correct. This is because simpler models are less likely to overfit to the training data and are more likely to generalize well to new, unseen data. Therefore, it is important to choose a model that is complex enough to capture the important patterns in the data, but not so complex that it overfits to the noise in the training data.

**Feature Selection:** In ML, feature selection involves selecting a subset of the available features that are most relevant for predicting the target variable. Occam's razor suggests that it is better to choose a smaller subset of features that are most important for predicting the target variable, rather than using all available features. This is because simpler models are less likely to overfit to noise in the data and are more likely to generalize well to new data.

**Interpretability:** Occam's razor suggests that it is better to choose a model that is simple and easy to interpret, rather than a more complex model that is difficult to understand. This is because simpler models are easier to understand and can provide more insights into the underlying patterns in the data.

**Regularization:** Regularization is a technique in ML that adds a penalty term to the loss function to prevent overfitting. Occam's razor suggests that it is better to choose a simpler model and use regularization to prevent overfitting, rather than using a more complex model that may be prone to overfitting.

Overall, the principle of Occam's razor can be applied in ML by choosing models that are simple, interpretable, and generalize well to new, unseen data. This can help to improve the generalization performance of the resulting model and reduce the risk of overfitting, underfitting, and other types of problems.

### 3.1.4

[E] What are the conditions that allowed deep learning to gain popularity in the last decade?

Deep learning is a subset of machine learning that involves training artificial neural networks with many layers to perform complex tasks such as image recognition, natural language processing, and speech recognition. There are several conditions that have allowed deep learning to gain popularity in the last decade:

1. **Big Data:** Deep learning algorithms require large amounts of data to train effectively. With the explosion of data in recent years, especially with the growth of the internet and social media, there is now a vast amount of data available for training deep learning models.
2. **Advances in Computing Power:** Training deep learning models requires a lot of computing power, especially for large datasets and complex models. The availability of powerful GPUs, cloud computing platforms, and other computing resources has made it easier and faster to train deep learning models.
3. **Availability of Open-Source Libraries:** The availability of open-source libraries such as TensorFlow, Keras, PyTorch, and Theano has made it easier for researchers and developers to implement deep learning models without needing to start from scratch. These libraries provide pre-built neural network architectures, optimization algorithms, and other tools for building and training deep learning models.
4. **Improved Algorithms:** Deep learning algorithms such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks have been developed and improved over the years, leading to better performance on a wide range of tasks.
5. **Success in Real-World Applications:** Deep learning has demonstrated impressive performance on a wide range of real-world applications, such as image recognition, speech recognition, and natural language processing. This has generated a lot of excitement and interest in the field, leading to increased investment and research in deep learning.

Overall, the combination of big data, advances in computing power, availability of open-source libraries, improved algorithms, and success in real-world applications has allowed deep learning to gain popularity in the last decade and become one of the most exciting and rapidly growing fields in artificial intelligence.

### 3.1.5

[M] If we have a wide NN and a deep NN with the same number of parameters, which one is more expressive and why?

In general, a deep neural network (DNN) with the same number of parameters as a wide neural network (WNN) is more expressive because it can represent more complex functions.

This is because deep neural networks can learn hierarchical representations of the input data by processing it through multiple layers of nonlinear transformations. Each layer learns to extract increasingly abstract features from the input data, which allows the DNN to capture more complex relationships between the inputs and the outputs.

On the other hand, a WNN typically has only one or a few layers, and its neurons are all connected to each other. This makes it difficult for the WNN to learn hierarchical representations of the input data, and it may not be able to capture complex relationships between the inputs and outputs as effectively as a DNN.

To illustrate this point, consider the example of image classification. A WNN with the same number of parameters as a DNN might be able to achieve reasonable accuracy on simple image classification tasks, such as distinguishing between cats and dogs. However, when it comes to more complex tasks such as distinguishing between different breeds of dogs or recognizing objects in cluttered scenes, the DNN is likely to outperform the WNN because it can learn more complex and abstract representations of the input data.

Overall, while both wide and deep neural networks have their own strengths and weaknesses, deep neural networks are generally more expressive and better suited for learning complex functions with many parameters.

### 3.1.6

[H] The Universal Approximation Theorem states that a neural network with 1 hidden layer can approximate any continuous function for inputs within a specific range. Then why can't a simple neural network reach an arbitrarily small positive error?

The Universal Approximation Theorem states that a neural network with 1 hidden layer can approximate any continuous function for inputs within a specific range. However, this does not mean that a simple neural network can reach an arbitrarily small positive error.

The reason for this is that the theorem does not provide any guidance on how to train the neural network to approximate the function. In practice, training a neural network involves optimizing the weights and biases of the network to minimize the error between the predicted outputs and the actual outputs. This optimization process is typically done using gradient descent or a related algorithm.

For complex functions, it may be difficult to find the optimal set of weights and biases using a single hidden layer. In such cases, a neural network with more layers or more complex architectures may be required to achieve better performance. Additionally, other techniques such as regularization, early stopping, and data augmentation may be necessary to improve the performance of the neural network.

Furthermore, even if a simple neural network with one hidden layer can approximate a function with low error, there may be practical limitations on the accuracy that can be achieved due to issues such as overfitting, numerical precision, and limited data availability. Therefore, while the Universal Approximation Theorem is a powerful theoretical result, it does not guarantee that a simple neural network can always achieve arbitrarily small positive error in practice.

### 3.1.7

[E] What are saddle points and local minima? Which are thought to cause more problems for training large NNs

In the context of training neural networks, saddle points and local minima are critical concepts that impact the optimization process.

A saddle point is a point in the optimization landscape where the gradient is zero in at least one direction, but not all. At a saddle point, the surface of the loss function curves upwards in some directions and downwards in others, creating a kind of saddle shape. The presence of saddle points can slow down the optimization process because the gradient-based optimization methods may get stuck in a saddle point instead of proceeding to the global minimum.

A local minimum, on the other hand, is a point in the optimization landscape where the loss function is lower than in all the neighboring points, but not necessarily the absolute minimum. Local minima can cause optimization algorithms to converge to sub-optimal solutions instead of reaching the global minimum.

Both saddle points and local minima can cause problems for training large neural networks, but saddle points are thought to be more problematic since they are more common in high-dimensional spaces. When a neural network has many parameters, the optimization landscape becomes more complex, and the probability of encountering saddle points increases. In practice, modern optimization techniques and architectures are designed to mitigate the impact of saddle points and local minima, such as using stochastic gradient descent with momentum or introducing skip connections in deep neural networks.

### 3.1.8

Hyperparameters.

1. [E] What are the differences between parameters and hyperparameters?

Parameters are the internal variables that the model learns from the training data. These parameters are typically adjusted through an optimization process to minimize the difference between the predicted outputs and the actual outputs of the training data. In a neural network, for example, the parameters include the weights and biases of the different layers.

Hyperparameters, on the other hand, are external variables that are not learned by the model. These variables are set before training and control the behavior of the model during training. Examples of hyperparameters include the learning rate, the number of hidden layers, the activation function, and the regularization parameter. The values of hyperparameters can significantly affect the performance of the model, and the optimal values need to be found through experimentation and tuning.

In summary, parameters are learned by the model during training, while hyperparameters are set externally and control the behavior of the training process. The goal of hyperparameter tuning is to find the optimal values that can maximize the performance of the model on unseen data.

2. [E] Why is hyperparameter tuning important?

Hyperparameter tuning is a critical step in the machine learning pipeline that can significantly impact the performance of a model. Here are a few reasons why hyperparameter tuning is important:

- **Impact on model performance:** The values of hyperparameters can significantly impact the performance of a model on unseen data. By finding the optimal values of hyperparameters, we can improve the accuracy and generalization ability of the model.
- **Generalizability:** A model that is optimized for a particular dataset may not generalize well to new data. By tuning hyperparameters, we can make the model more robust to variations in the data and improve its ability to generalize to new data.
- **Time and computational efficiency:** Hyperparameter tuning can help us find the optimal set of hyperparameters that maximize the performance of the model while minimizing the time and computational resources required for training.
- **Avoid overfitting:** Hyperparameter tuning can help us avoid overfitting, which occurs when the model is too complex and fits the training data too well, resulting in poor performance on new data. By adjusting the hyperparameters, we can prevent the model from overfitting and improve its ability to generalize.

In summary, hyperparameter tuning is essential for building accurate and robust machine learning models. It enables us to find the optimal set of hyperparameters that can improve the performance and generalization ability of the model while minimizing the time and computational resources required for training.

3. [M] Explain algorithm for tuning hyperparameters. There are several approaches to hyperparameter tuning, ranging from manual search to automated methods. Here is a basic algorithm for tuning hyperparameters:

- (i) **Define the hyperparameters:** Determine the hyperparameters that need to be tuned for the model. These may include learning rate, number of hidden layers, batch size, activation functions, regularization parameters, and more.
- (ii) **Define the search space:** For each hyperparameter, define a range of possible values to explore. This defines the hyperparameter search space.
- (iii) **Choose a tuning method:** Choose a hyperparameter tuning method based on the available resources and complexity of the model. Some popular methods include Grid Search, Random Search, Bayesian Optimization, and Evolutionary Algorithms.
- (iv) **Train the model:** Train the model using a set of hyperparameters from the search space. Evaluate the performance of the model using a validation set or cross-validation.
- (v) **Evaluate the performance:** Use a performance metric, such as accuracy or mean squared error, to evaluate the performance of the model with the current set of hyperparameters.
- (vi) **Select the next set of hyperparameters:** Based on the performance of the previous set of hyperparameters, select the next set of hyperparameters to try. Repeat steps (iv)-(vi) until the performance of the model stops improving or a maximum number of iterations is reached.
- (vii) **Test the model:** Once the best set of hyperparameters is found, test the model on a held-out test set to estimate its performance on new, unseen data.
- (viii) **Refit the model:** Finally, retrain the model using the best set of hyperparameters on the entire training set to obtain the final model.

This is a basic algorithm, and different tuning methods may have different steps or require additional resources. However, the general process involves defining the hyperparameters, exploring the search space, evaluating the performance, and selecting the best set of hyperparameters.

### 3.1.9

Classification vs. regression.

1. [E] What makes a classification problem different from a regression problem?
2. [E] Can a classification problem be turned into a regression problem and vice versa?

In some cases, it is possible to convert a classification problem into a regression problem or vice versa, depending on the nature of the problem and the goals of the analysis. However, this may require additional data preprocessing or modeling techniques.

	Classification	Regression
<b>Output type</b>	Output type is categorical	Output type is continuous
<b>Task</b>	Predict class of input data	Predict numerical value based on input features
<b>Evaluation metrics</b>	accuracy, precision, recall, F1-score	mean squared error, mean absolute error, R-squared
<b>Algorithms</b>	logistic regression, decision trees, SVMs	linear regression, polynomial regression, decision trees
<b>Training data</b>	labeled with class labels	labeled with continuous numerical values

Table 3.1: Comparison between classification and regression problems

Converting a classification problem to a regression problem involves transforming the categorical output variable into a continuous numerical variable. This can be achieved by assigning numerical values to the different categories. One example of converting a classification problem to a regression problem is to use a probability-based approach, where instead of predicting the class label directly, we predict the probability of the input belonging to each class. This can be useful in cases where there are multiple classes and we want to know the likelihood of an input belonging to each of them.

To illustrate this, let's consider a hypothetical example of a classification problem where we are trying to predict the type of flower based on its petal length and width. The classes are "setosa", "versicolor", and "virginica". We could use a logistic regression algorithm to make predictions and obtain the probability of the input belonging to each class.

However, if we want to convert this classification problem to a regression problem, we could predict the actual length or width of the petal instead of the class label. In this case, we would use a regression algorithm to predict the continuous numerical value of the petal length or width. This could be useful in cases where we are interested in predicting a continuous variable that is related to the categorical label, such as the size of the flower petals.

Converting a regression problem to a classification problem involves discretizing the continuous output variable into a finite number of categories. This can be done by defining thresholds or bins for the numerical values and assigning each value to a category. Once the output variable has been discretized, standard classification algorithms can be used to make predictions.

However, it's important to note that converting a problem from one type to another may result in a loss of information or accuracy, and may not always be appropriate or useful for the analysis. It is generally better to choose the appropriate type of problem based on the nature of the data and the research question at hand.

### 3.1.10

Parametric vs. non-parametric methods.

1. [E] What's the difference between parametric methods and non-parametric methods? Give an example of each method.

Parametric methods and non-parametric methods are two broad categories of statistical and machine learning models. The main difference between the two is the assumptions they make about the underlying data distribution.

Parametric methods assume that the data follows a specific probability distribution, with a fixed number of parameters that can be estimated from the data. These methods have a fixed set of parameters that are learned during training and do not increase with the size of the dataset. Examples of parametric methods include linear regression, logistic regression, and the Naive Bayes algorithm.

Non-parametric methods, on the other hand, do not make any assumptions about the underlying data distribution, and instead seek to estimate the distribution directly from the data. These methods do not have a fixed set of parameters and the number of parameters can increase with the size of the dataset. Examples of non-parametric methods include decision trees, k-nearest neighbors, and support vector machines.

Here is an example of each method:

**Parametric method - Linear Regression:** Linear regression is a parametric method used to predict a continuous output variable based on one or more input variables. It assumes that the relationship between the input and output variables can be modeled by a linear function, where the coefficients of the function are estimated from the data using a least-squares method. Linear regression is a commonly used model in regression problems.

**Non-parametric method - K-Nearest Neighbors:** K-nearest neighbors (KNN) is a non-parametric method used for both classification and regression problems. It works by finding the k nearest neighbors of a given input data point, and then using the average or majority vote of the neighbors' output values to make

a prediction for the new data point. KNN does not make any assumptions about the underlying data distribution and can be useful in cases where the relationship between the input and output variables is complex and cannot be captured by a simple parametric model like linear regression.

2. [H] When should we use one and when should we use the other?

The choice between using a parametric or non-parametric method depends on several factors, including the nature of the data, the research question, and the trade-offs between model simplicity and flexibility.

Parametric methods are typically preferred when the underlying data distribution is known or can be assumed to follow a specific probability distribution. In these cases, parametric models can be more efficient and require fewer data points to estimate the model parameters. They can also provide interpretable results and make it easier to generalize the model to new data. However, if the underlying assumptions of the distribution are not met, the model may not perform well.

Non-parametric methods are typically preferred when the underlying data distribution is unknown or cannot be assumed to follow a specific probability distribution. In these cases, non-parametric models can be more flexible and can capture complex relationships between the input and output variables. They are also less sensitive to outliers and can provide robust results. However, non-parametric models can be computationally expensive and may require a larger amount of data to estimate the model parameters accurately. They may also be less interpretable than parametric models.

In general, the choice between using a parametric or non-parametric method depends on the specific problem and research question. If the underlying data distribution is known or can be assumed to follow a specific probability distribution, a parametric method may be a better choice. If the relationship between the input and output variables is complex and cannot be captured by a simple parametric model, a non-parametric method may be a better choice. Ultimately, the choice between the two methods should be based on the trade-offs between model simplicity, flexibility, and accuracy for the specific problem at hand.

### 3.1.11

- [M] Why does ensembling independently trained models generally improve performance?

Ensembling independently trained models can generally improve performance because it helps to reduce overfitting and improve the generalization ability of the model. There are several reasons why ensembling works:

1. Reducing Variance: Ensemble methods can reduce the variance of the model by averaging the predictions of multiple models. Since each model is trained independently on a different subset of the data or with a different algorithm, it is likely to make different errors. By averaging the predictions of multiple models, the errors can cancel each other out, reducing the overall variance.
2. Capturing More Complex Relationships: Ensemble methods can capture more complex relationships between the input and output variables by combining the strengths of multiple models. Each model may capture different aspects of the data or relationships between the features, which can be combined to improve the accuracy of the ensemble.
3. Reducing Bias: Ensemble methods can also reduce bias by combining the predictions of multiple models. If a single model is biased towards a certain class or set of features, the ensemble can help to correct this bias by taking into account the predictions of multiple models.
4. Robustness: Ensemble methods can also make the model more robust to outliers and noise in the data. Since each model is trained independently, it is less likely to be affected by outliers or noise in the training data.

In general, ensembling independently trained models can improve performance by reducing variance, capturing more complex relationships, reducing bias, and improving the robustness of the model. However, ensembling can also increase the computational complexity of the model and may require additional training data. It is important to weigh the benefits and costs of ensembling for a particular problem to determine if it is an appropriate technique.

### 3.1.12

- [M] Why does L1 regularization tend to lead to sparsity while L2 regularization pushes weights closer to 0?

L1 and L2 regularization are two commonly used techniques to prevent overfitting in machine learning models. They differ in the way they penalize the weights of the model during training.

L1 regularization adds a penalty term to the loss function that is proportional to the absolute value of the weights. This penalty encourages the weights of the model to be sparse, meaning that many of them will be exactly equal to zero. This is because the absolute value of a weight must exceed the regularization penalty for it to be non-zero. Therefore, L1 regularization tends to reduce the number of active features in the model and can be used for feature selection.

L2 regularization adds a penalty term to the loss function that is proportional to the square of the weights. This penalty encourages the weights of the model to be small, but not necessarily zero. This is because the penalty term will always be non-zero, even if the weight itself is small. Therefore, L2 regularization tends to reduce the magnitude of the weights but does not lead to sparsity in the same way as L1 regularization. Overall, L1 regularization tends to lead to sparsity because it encourages many weights to be exactly zero, while L2 regularization tends to push the weights towards smaller values, but does not induce sparsity in the same way as L1 regularization.

### 3.1.13

[E] Why does an ML model's performance degrade in production?

There are several reasons why an ML model's performance can degrade in production compared to during development and testing:

1. **Data Drift:** Machine learning models are trained on a specific dataset and assume that the distribution of the data will remain the same during production. However, in real-world scenarios, the distribution of the data can change over time, which can cause the model's performance to degrade. This is known as data drift. If the model encounters data that is significantly different from the training data, it may not be able to make accurate predictions.
2. **Concept Drift:** Similar to data drift, concept drift refers to changes in the relationship between the input features and the target variable that the model is trying to predict. This can occur due to changes in user behavior, changes in the business environment, or other factors that affect the relationship between the input features and the target variable.
3. **Model Complexity:** Models that are too complex may overfit to the training data and not generalize well to new data in production. In addition, complex models may be computationally expensive and slow to make predictions in real-time, which can limit their use in production.
4. **Insufficient Data:** Machine learning models require sufficient data to learn the underlying patterns in the data. If the model is trained on a small or biased dataset, it may not be able to generalize well to new data in production.
5. **Inadequate Testing:** The model may not have been tested thoroughly in development, which can lead to bugs and errors that are only discovered in production.
6. **Poor Data Quality:** The performance of a machine learning model is only as good as the quality of the data it is trained on. Poor data quality, such as missing values or incorrect labels, can significantly impact the model's performance in production.

In summary, an ML model's performance can degrade in production due to data drift, concept drift, model complexity, insufficient data, inadequate testing, and poor data quality. To mitigate these issues, it is important to monitor the model's performance in production and update the model as needed to ensure that it continues to perform well over time.

### 3.1.14

[M] What problems might we run into when deploying large machine learning models?

Deploying large machine learning models can pose several challenges, including:

1. **Computational Resources:** Large models often require significant computational resources to run, including memory, CPU, and GPU resources. This can be a problem when deploying the model to a production environment, especially if the environment has limited resources or the model needs to run in real-time.
2. **Latency:** Large models can take longer to run, which can increase the latency of the application. This can be a problem if the application needs to make real-time predictions, such as in a chatbot or recommendation system.
3. **Model Size:** Large models can be challenging to deploy, especially if they have many parameters or require significant storage space. This can be a problem if the deployment environment has limited storage capacity.
4. **Versioning:** Keeping track of different versions of large models can be difficult, especially if the models are updated frequently. This can be a problem if there are multiple versions of the model in production, and it is difficult to track which version is being used.
5. **Reproducibility:** It can be challenging to reproduce the results of large machine learning models, especially if they were trained on a large dataset or with a complex architecture. This can be a problem if the results need to be audited or replicated for regulatory purposes.

6. Security: Large machine learning models can pose a security risk, as they may contain sensitive data or be vulnerable to attacks such as adversarial examples or model stealing.

To address these challenges, it is important to carefully consider the deployment environment, optimize the model architecture and size, and develop robust versioning and monitoring strategies. It may also be necessary to use distributed computing or cloud-based services to provide the necessary computational resources.

### 3.1.15

Your model performs really well on the test set but poorly in production.

1. [M] What are your hypotheses about the causes?

There could be several hypotheses as to why a machine learning model performs well on the test set but poorly in production. One possible cause could be overfitting to the test set or the training data. Another cause could be a mismatch between the test and production environments, such as differences in data distribution or input features. It is also possible that the model is not robust enough to handle real-world scenarios or that there are issues with the deployment infrastructure, such as network latency or resource constraints.

2. [H] How do you validate whether your hypotheses are correct?

To validate these hypotheses, one can perform various tests such as:

- Collect more data from the production environment to ensure that the data distribution is similar to the test set
- Analyze the feature distributions and verify that they are consistent across the test and production environments
- Run experiments to determine if the model is overfitting or if it is failing to generalize to new data
- Monitor the model's performance in production and identify any patterns or trends that could indicate issues with the infrastructure or real-world scenarios

3. [M] Imagine your hypotheses about the causes are correct. What would you do to address them

If the hypotheses are correct, there are several steps that can be taken to address them:

- Re-evaluate the model architecture and parameters to ensure that it is not overfitting to the test set or training data
- Re-train the model with additional data or a modified architecture that is better suited for production
- Monitor the model's performance in production and identify any issues with the deployment infrastructure, such as network latency or resource constraints, and optimize accordingly
- Develop a robust testing and deployment process that can catch issues before they are deployed to production.

Ultimately, the key to addressing these issues is to have a thorough understanding of the model, its performance, and the production environment, and to continuously monitor and optimize the system to ensure that it meets the desired performance requirements.

## 3.2 Sampling and creating training data

### 3.2.1

[E] If you have 6 shirts and 4 pairs of pants, how many ways are there to choose 2 shirts and 1 pair of pants?

To choose 2 shirts out of 6, we can use the combination formula, which is:

$$C(6, 2) = \frac{6!}{2! * (6 - 2)!} = 15$$

This means there are 15 ways to choose 2 shirts out of 6.

To choose 1 pair of pants out of 4, we can simply use the multiplication rule, which states that the number of ways to perform two independent tasks is the product of the number of ways to perform each task.

So the number of ways to choose 2 shirts and 1 pair of pants is:

$$15 * 4 = 60$$

Therefore, there are 60 ways to choose 2 shirts and 1 pair of pants out of 6 shirts and 4 pairs of pants.

### 3.2.2

[M] What is the difference between sampling with vs. without replacement? Name an example of when you would use one rather than the other?

Sampling with replacement means that after each selection, the selected item is put back into the population and can be selected again. In contrast, sampling without replacement means that after each selection, the selected item is removed from the population and cannot be selected again.

An example of sampling with replacement is drawing cards from a deck and putting each card back in the deck after it's drawn. An example of sampling without replacement is selecting winners in a raffle, where each person can only win once.

When selecting a sampling method, it's important to consider the nature of the population and the goals of the study. Sampling with replacement can be useful when the population is large relative to the sample size, and the goal is to estimate a population parameter. In this case, each member of the population has an equal chance of being selected on each draw, and the sampling distribution of the statistic of interest will approach a normal distribution as the sample size increases.

Sampling without replacement can be useful when the population is small relative to the sample size, and the goal is to ensure that each member of the population has a fair and equal chance of being selected. In this case, sampling with replacement could lead to repeated selection of the same items, which would bias the results.

In summary, the main difference between sampling with vs. without replacement is whether or not the selected item is put back into the population after each draw. The choice of sampling method depends on the nature of the population and the goals of the study.

### 3.2.3

[M] Explain Markov chain Monte Carlo sampling.

Read [this](#) article for introduction to MCMC and [this](#) for a more intuitive example (specifically, Example: In-class test).

### 3.2.4

[M] If you need to sample from high-dimensional data, which sampling method would you choose?

When sampling from high-dimensional data, it can be challenging to capture the complex interactions and correlations among the variables, and the curse of dimensionality can make the sampling process inefficient. Here are some methods that can be useful in this scenario:

#### Halton sampling

Halton sampling is a method for generating low-discrepancy sequences of points in high-dimensional spaces. It is a type of quasi-random sampling, which is an alternative to random sampling that seeks to minimize the discrepancy between the distribution of the samples and the underlying distribution of the data.

Halton sampling is based on the Halton sequence, which is a one-dimensional sequence of numbers that are generated by successively dividing a number by a prime number. The sequence has good properties in terms of randomness and low discrepancy, and can be extended to higher dimensions by generating multiple Halton sequences with different prime numbers and combining them using a coordinate-wise mapping.

To generate Halton samples in high-dimensional spaces, we start by choosing a set of prime numbers for each dimension. For example, if we are generating samples in three dimensions, we can choose the primes 2, 3, 5. We then generate a Halton sequence for each dimension using the corresponding prime number, and combine them using a coordinate-wise mapping. Specifically, we map the  $i$ -th sample in the  $j$ -th dimension to the  $i$ -th element of the Halton sequence generated using the  $j$ -th prime number.

Halton sampling has several advantages over random sampling, including lower discrepancy, better coverage of the space, and the ability to generate a large number of samples without repeating. However, it may not be suitable for all applications, as the samples may not be representative of the underlying distribution of the data, and may not capture complex interactions and correlations among the variables. The choice of sampling method depends on the specific requirements of the application.

#### LCVT (Lloyd's method for Centroidal Voronoi Tessellations)

LCVT involves dividing the high-dimensional space into Voronoi cells, where each cell contains one sample point that serves as the centroid of the cell. The method iteratively moves the centroid of each cell to the average of the points within the cell, until convergence is reached. The resulting set of sample points forms a centroidal Voronoi tessellation, which can be useful for clustering and generating representative samples.

### 3.2.5

[H] Suppose we have a classification task with many classes. An example is when you have to predict the next word in a sentence – the next word can be one of many, many possible words. If we have to calculate



the probabilities for all classes, it'll be prohibitively expensive. Instead, we can calculate the probabilities for a small set of candidate classes. This method is called candidate sampling. Name and explain some of the candidate sampling algorithms.

Read [this](#) article for an in-depth answer.

### 3.2.6

Suppose you want to build a model to classify whether a Reddit comment violates the website's rule. You have 10 million unlabeled comments from 10K users over the last 24 months and you want to label 100K of them.

1. [M] How would you sample 100K comments to label?

Stratified Sampling: A more sophisticated method is to use stratified sampling, which involves dividing the data into subgroups based on certain characteristics (e.g., user ID or time period) and then sampling a proportionate number of comments from each subgroup. This method can ensure that the labeled data is representative of the overall.

2. [M] Suppose you get back 100K labeled comments from 20 annotators and you want to look at some labels to estimate the quality of the labels. How many labels would you look at? How would you sample them?

We consider the labeled comments from each of these 20 annotators as clusters. Then we randomly sample comments belonging to each of the 20 clusters (cluster sampling) and analyze if the results are coherent among clusters.

Hint: This [article](#) on different sampling methods and their use cases might help.

### 3.2.7

[M] Suppose you work for a news site that historically has translated only 1% of all its articles. Your coworker argues that we should translate more articles into Chinese because translations help with the readership. On average, your translated articles have twice as many views as your non-translated articles. What might be wrong with this argument?

Hint: think about selection bias.

While it is true that translated articles may help increase readership and engagement, there are several issues with this argument:

1. Causality: The argument assumes that translation is the only factor that affects readership, and that increasing translation will necessarily lead to increased readership. However, there may be other factors that affect readership, such as the quality of the content, the relevance to the target audience, or the effectiveness of the marketing and promotion strategies. Without controlling for these factors, it is difficult to establish a causal relationship between translation and readership.
2. Selection bias: The argument assumes that translated articles are representative of the overall content, and that the higher readership of translated articles is not due to some other factor that makes them more popular or more likely to be promoted. However, it is possible that the selection of articles for translation is biased towards topics or styles that are already more popular or appealing to the target audience. This would result in a self-reinforcing cycle where translated articles are more popular simply because they are selected based on popularity.
3. Sample size: The argument is based on a relatively small sample size (1% of all articles) and may not be representative of the overall readership. It is possible that the higher readership of translated articles is due to chance or other factors that are not captured by this small sample.
4. Confounding variables: The argument does not account for other variables that may affect readership, such as the timing of the publication, the competition from other news sources, or the overall trends and preferences of the target audience. These variables may be confounding factors that affect both the decision to translate and the readership, making it difficult to isolate the effect of translation alone.

Therefore, while translating more articles may be a good strategy for increasing readership and engagement, it is important to carefully evaluate the underlying assumptions and potential biases before making a decision. A more rigorous analysis of the data, such as a randomized controlled trial or a regression analysis that controls for confounding variables, may help establish a more robust and reliable relationship between translation and readership.

### 3.2.8

[M] How to determine whether two sets of samples (e.g. train and test splits) come from the same distribution?

Here are some common statistical tests for comparing two distributions:

- Kolmogorov-Smirnov test: This test compares the cumulative distribution functions (CDFs) of the two samples and computes a test statistic that measures the maximum difference between the CDFs. If the test statistic is larger than a critical value, we reject the null hypothesis that the two samples come from the same distribution.
- Anderson-Darling test: This test is similar to the Kolmogorov-Smirnov test but places more weight on differences in the tails of the distribution. It computes a test statistic that takes into account the entire CDF and compares it to a critical value to determine whether to reject the null hypothesis.

Another approach is to use visualization techniques to compare the distributions. This can include creating histograms or kernel density plots of the train and test sets and visually comparing them to identify any differences. Box plots can also be used to visualize the quartiles of the two sets and any outliers.

Another approach is discussed in [this](#) article.

### 3.2.9

[H] How do you know you've collected enough samples to train your ML model?

The most common way to define whether a data set is sufficient is to apply a 10 times rule. This rule means that the amount of input data (i.e., the number of examples) should be ten times more than the number of degrees of freedom a model has. Usually, degrees of freedom mean parameters in your data set.

So, for example, if your algorithm distinguishes images of cats from images of dogs based on 1,000 parameters, you need 10,000 pictures to train the model.

It is important to note here that AI models don't study the data but rather the relationships and patterns behind the data. So it's not only quantity that will influence the results, but also quality.

A good discussion in [this](#) article.

### 3.2.10

[M] How to determine outliers in your data samples? What to do with them?

Answer mostly taken from [here](#).

Outliers are data points that are significantly different from other observations in a sample. They can occur due to various reasons, such as measurement error, data entry errors, or rare events. Outliers can have a significant impact on statistical analyses and can distort results, so it's essential to identify and handle them appropriately. There are several methods to determine outliers in data samples:

1. Visualization: One way to identify outliers is by visualizing the data using a scatter plot or box plot. Outliers are data points that fall far away from the main cluster of data points.
2. Z-score: Another method to identify outliers is by calculating the z-score of each data point. A z-score represents the number of standard deviations that a data point is away from the mean. Typically, a z-score of greater than 3 or less than -3 is considered an outlier.
3. IQR method: The interquartile range (IQR) is another way to identify outliers. The IQR is the difference between the 75<sup>th</sup> percentile and the 25<sup>th</sup> percentile of the data. Data points outside of 1.5 times the IQR above the 75<sup>th</sup> percentile or below the 25<sup>th</sup> percentile are considered outliers.

Once you have identified outliers, there are several ways to handle them:

- Remove them: The simplest way to handle outliers is to remove them from the dataset. However, removing outliers can reduce the sample size and may result in a biased dataset.
- Transform the data: Sometimes, it may be possible to transform the data to reduce the impact of outliers. For example, taking the logarithm of the data may reduce the effect of extreme values.
- Keep them: In some cases, outliers may be important and valuable data points that should not be removed. It's important to carefully consider whether removing outliers is appropriate for the analysis you are performing.

Ultimately, the method of handling outliers depends on the specific dataset and analysis being performed. It's important to carefully consider the impact of outliers on the analysis and choose an appropriate method of handling them.

### 3.2.11

Sample duplication.

1. [M] When should you remove duplicate training samples? When shouldn't you?

Duplicate training samples refer to instances in a dataset that have the exact same feature values as another instance. Removing duplicate training samples can have several advantages, but there are also situations where it may not be necessary or even harmful.

Here are some scenarios where removing duplicate training samples may be appropriate:

- Increased efficiency: If a dataset has many duplicate samples, training a machine learning model on this data can be computationally expensive and time-consuming. Removing duplicates can reduce the amount of data that needs to be processed, which can lead to faster training times and more efficient use of resources.
- Improved model performance: In some cases, having duplicate samples in a dataset can lead to overfitting, where the model learns to memorize the training data rather than learning general patterns that can be applied to new data. Removing duplicates can reduce the likelihood of overfitting and can lead to better model performance on new, unseen data.
- Reduced bias: If duplicates are overrepresented in a dataset, this can introduce bias into the model. Removing duplicates can help to reduce this bias and lead to more accurate predictions.

However, there are also scenarios where it may not be appropriate to remove duplicate training samples:

- Small datasets: In some cases, the dataset may be too small to justify removing any data points, including duplicates. Removing samples in small datasets can reduce the amount of data available for training and may result in a less accurate model.
- Important data: If duplicate samples contain important information that is relevant to the problem being solved, removing them can result in a loss of information and may reduce the accuracy of the model.
- Non-identical duplicates: In some cases, two data points may have the same feature values but represent different instances. For example, in a medical dataset, two patients may have the same age, gender, and diagnosis, but may have different medical histories. Removing these duplicates can lead to a loss of information and can negatively impact the model's performance.

In summary, whether to remove duplicate training samples depends on the specific dataset and problem being solved. It's important to carefully consider the benefits and drawbacks of removing duplicates and to make an informed decision based on the specific situation.

2. [M] What happens if we accidentally duplicate every data point in your train set or in your test set?

If every data point in a train set or a test set is accidentally duplicated, it can lead to several issues in machine learning models.

Firstly, if the duplicated samples are included in the train set, it can lead to overfitting. Overfitting occurs when a machine learning model learns the training data too well, including the noise in the data, and therefore, fails to generalize well to new unseen data. In this case, the duplicated samples will be counted twice while training, and the model will learn to assign more weight to these samples, leading to overfitting.

Secondly, if the duplicated samples are included in the test set, it can lead to an overestimation of the model's performance. Since the duplicated samples are identical to the other samples, the model is likely to make accurate predictions on these samples, leading to an artificially high test accuracy. However, the model's performance on new unseen data is likely to be much worse.

Lastly, duplicating every data point in a train or test set can also lead to an imbalanced dataset. If there are other factors in the data that influence the outcome variable, these factors may be underrepresented in the duplicated samples, leading to an imbalanced distribution of these factors in the dataset.

Therefore, it's important to check the data for duplicate samples before training a machine learning model to avoid these issues. If duplicates are accidentally introduced, it's important to remove them to prevent overfitting and artificially high test accuracy.

### 3.2.12

Missing data

1. [M] In your dataset, two out of 20 variables have more than 30% missing values. What would you do?

If two out of 20 variables in a dataset have more than 30% missing values, it's important to carefully consider how to handle these missing values before proceeding with any analysis or modeling. Here are some potential approaches to consider:

- (a) Drop the variables: If the two variables with missing values are not critical to the analysis or modeling process, one option is to simply drop them from the dataset. However, it's important to consider whether dropping these variables will significantly impact the overall analysis or modeling results.
- (b) Impute the missing values: Another approach is to impute the missing values using a statistical method such as mean imputation, median imputation, or regression imputation. However, it's important to be cautious when using imputation, as it can introduce bias into the data and impact the accuracy of the analysis or modeling results.
- (c) Create a missing value indicator variable: Instead of imputing the missing values, another option is to create a new variable that indicates whether a particular observation has missing values in one of the two variables. This can allow the missingness of the data to be taken into account in the analysis or modeling process.
- (d) Explore the missingness pattern: It's also important to explore the pattern of missing values in the dataset to determine if there is any systematic reason for the missing values. For example, if the missing values are more likely to occur in certain types of observations, this may indicate a problem with the data collection process that needs to be addressed.

Ultimately, the best approach will depend on the specific dataset and the goals of the analysis or modeling process. It's important to carefully consider the potential impact of missing values on the results and to choose an approach that minimizes bias and maximizes accuracy.

2. [M] How might techniques that handle missing data make selection bias worse? How do you handle this bias?

Techniques that handle missing data, such as imputation, can potentially make selection bias worse if the missingness pattern is not random but instead related to the outcome variable or other factors in the data. In such cases, imputing missing values using statistical methods can introduce bias into the data and impact the accuracy of the analysis or modeling results.

For example, if missing values are more likely to occur in observations with a certain value of the outcome variable, imputing the missing values with mean imputation could lead to bias in the estimated mean of the outcome variable. This is because the mean of the outcome variable for observations with missing values may be systematically different from the mean of the outcome variable for observations with complete data.

To handle this bias, it's important to carefully assess the pattern of missing data in the dataset and consider the potential sources of bias that may be introduced by handling the missing data. One approach is to use multiple imputation, which involves creating several imputed datasets with different plausible imputed values for the missing data. This can help to account for the uncertainty in the imputed values and reduce bias in the analysis or modeling results.

Another approach is to use methods that are robust to missing data, such as likelihood-based methods or Bayesian methods, that can account for the missingness pattern in the data and produce unbiased estimates. However, these methods may be computationally intensive and require additional assumptions about the distribution of the data.

In any case, it's important to carefully document the handling of missing data and any potential sources of bias to ensure that the analysis or modeling results are interpreted correctly.

### 3.2.13

- [M] Why is randomization important when designing experiments (experimental design)?

Randomization is an important aspect of experimental design because it helps to reduce bias and increase the reliability and validity of the results. Here are a few reasons why randomization is important:

1. Reducing bias: Randomization helps to ensure that the treatment groups in an experiment are similar in all respects except for the treatment itself. This helps to reduce the risk of bias due to factors such as confounding variables, selection bias, or assignment bias.
2. Increasing reliability: Randomization helps to increase the reliability of the results by reducing the impact of random variation. By randomly assigning subjects to treatment groups, any variation that occurs between the groups is more likely to be due to the treatment itself rather than other factors.
3. Increasing validity: Randomization also helps to increase the validity of the results by ensuring that they are generalizable to the population from which the subjects were drawn. By randomly selecting subjects from the population and randomly assigning them to treatment groups, the results are more likely to be representative of the population as a whole.
4. Enhancing statistical analysis: Randomization also enhances the statistical analysis by allowing for the use of inferential statistics, which can help to determine the significance of the results.

Overall, randomization is an essential component of experimental design because it helps to ensure that the results are unbiased, reliable, and valid.

### 3.2.14

Class imbalance.

1. [E] How would class imbalance affect your model?

Class imbalance can have a significant impact on the performance of a model, particularly in binary classification problems where the number of examples in one class is much smaller than the other. Here are a few ways in which class imbalance can affect a model:

- (a) Biased models: A model that is trained on an imbalanced dataset may become biased towards the majority class, since it sees many more examples from that class than the minority class. This can result in the model being overly optimistic in its predictions for the minority class.
  - (b) Poor performance: When there is class imbalance, accuracy is not always a good metric to evaluate model performance. This is because a model that simply predicts the majority class for all examples may have high accuracy, but will perform poorly on the minority class.
  - (c) Overfitting: Imbalanced datasets can also lead to overfitting, as the model may memorize the majority class and perform poorly on new examples from the minority class.
  - (d) False positives/negatives: Models trained on imbalanced datasets may have a high false positive or false negative rate, particularly for the minority class. This can lead to incorrect predictions and poor performance.
2. [E] Why is it hard for ML models to perform well on data with class imbalance?

Most machine learning algorithms assume data equally distributed. So when we have a class imbalance, the machine learning classifier tends to be more biased towards the majority class, causing bad classification of the minority class.

3. [M] Imagine you want to build a model to detect skin lesions from images. In your training dataset, only 1% of your images shows signs of lesions. After training, your model seems to make a lot more false negatives than false positives. What are some of the techniques you'd use to improve your model? For in-depth review read [this](#) article.

Here are some of the techniques that can be used to improve the model:

- (a) Resampling techniques: A common approach to address class imbalance is to use resampling techniques such as oversampling the minority class (skin lesions) or undersampling the majority class (healthy skin). For example, the SMOTE algorithm is a method of resampling from the minority class while slightly perturbing feature values, thereby creating "new" samples.
- (b) Using a different evaluation metric: In highly imbalanced datasets, accuracy is not an appropriate evaluation metric as it can be misleading. Instead, metrics such as F1-score, area under the receiver operating characteristic curve (AUC-ROC), and precision-recall curve should be used to evaluate the model's performance.
- (c) cost-sensitive training: use a cost-sensitive learning approach where the algorithm is penalized more for making mistakes on the minority class.

### 3.2.15

Training data leakage.

More detailed explanation of data leakage in [this](#) article.

1. [M] Imagine you're working with a binary task where the positive class accounts for only 1% of your data. You decide to oversample the rare class then split your data into train and test splits. Your model performs well on the test split but poorly in production. What might have happened?

One of the major reasons could be:

Data leakage: If the oversampling of the rare class was done incorrectly or improperly, there may be data leakage that could artificially inflate the performance of the model on the test split. For example, if the same images were present in both the training and test splits, the model may have learned to recognize those specific images instead of generalizing to new images.

2. [M] You want to build a model to classify whether a comment is spam or not spam. You have a dataset of a million comments over the period of 7 days. You decide to randomly split all your data into the train and test splits. Your co-worker points out that this can lead to data leakage. How?

Data leakage generally occurs when the training data is overlapped with testing data during the development process of ML models by sharing information between both data sets. In the presented scenario,

note that we are working with time-series data so randomly splitting into test and training data sets will cause data leakage.

Since the comments in the dataset were generated over a period of seven days, there is a possibility that on randomly splitting, some comments in the test set were generated after the training set. In this case, the model may be exposed to patterns in the test set that are not present in the training set, which can lead to overfitting and artificially high performance metrics.

To minimize the data leakage in time-series data, we put a cut-off value on time as it helps avoid getting any information after the time of prediction.

Hint: You might want to clarify what oversampling here means. Oversampling can be as simple as duplicating samples from the rare class.

### 3.2.16

[M] How does data sparsity affect your models?

Hint: Sparse data is different from missing data.

Answer mainly taken from [this](#) article. When there is missing data, it means that many data points are unknown. On the other hand, if the data is sparse, all the data points are known, but most of them have zero value.

Common problems with sparse features include:

1. If the model has many sparse features, it will increase the space and time complexity of models. Linear regression models will fit more coefficients, and tree-based models will have greater depth to account for all features.
2. Model algorithms and diagnostic measures might behave in unknown ways if the features have sparse data.
3. If there are too many features, models fit the noise in the training data. When models overfit, they are unable to generalize to newer data when they are put in production. This negatively impacts the predictive power of models.
4. Some models may underestimate the importance of sparse features and given preference to denser features even though the sparse features may have predictive power. Tree-based models are notorious for behaving like this. For example, random forests overpredict the importance of features that have more categories than those features that have fewer categories.

### 3.2.17

Feature leakage.

For a detailed discussion read [this](#) article.

1. [E] What are some causes of feature leakage?

Leakage in features occurs when something extremely informative about the true label is included as a feature. Feature leakage can occur due to several reasons, including:

- (a) Time-dependent features: If features that are only available in the future or at inference time are used in the model during training, this can lead to feature leakage. For example, if a stock price prediction model uses the future closing price of a stock as a feature, this would result in feature leakage since the future closing price is not available during training.
  - (b) Target leakage: Target leakage occurs when features that are derived from the target variable are used in the model during training. For example, if a credit risk model uses the amount of a loan as a feature, this would result in feature leakage since the amount of the loan is directly related to the target variable.
2. [E] Why does normalization help prevent feature leakage?

Normalization helps prevent feature leakage by ensuring that the scaling of the data is consistent between the training and test sets. If normalization is performed on the entire dataset, including the test set, this can result in feature leakage since information from the test set is used to scale the training data.

By normalizing the training and test sets separately, each set is scaled based only on its own statistics, ensuring that there is no information leakage from the test set. Normalization also helps to prevent the model from being biased towards features with larger values, which can result in poor generalization to new data.

### 3. [M] How do you detect feature leakage?

Detecting feature leakage can be challenging, but there are several techniques that can be used to identify potential instances of feature leakage in a model. Some techniques include:

- (a) Cross-validation: Cross-validation is a technique for evaluating the performance of a model by splitting the data into several folds and training the model on each fold while using the remaining folds for validation. By using cross-validation, it's possible to detect feature leakage since any features that are not available during inference time will result in poor performance on the validation sets.
- (b) Feature importance: Feature importance techniques, such as permutation feature importance or SHAP values, can be used to identify features that are highly predictive for the model. If a feature that is not available during inference time is identified as highly predictive, this may indicate that there is feature leakage in the model.
- (c) Manual review: Manual review of the data and model implementation can also be effective in identifying instances of feature leakage. Reviewing the feature descriptions and data collection process can help to identify features that are not available during inference time. Additionally, reviewing the code and model architecture can help to identify any accidental inclusions of features that are not available during inference time.
- (d) Hypothesis testing: Hypothesis testing can be used to determine whether there is a statistical relationship between a feature and the target variable. If a feature that is not available during inference time is found to be statistically significant, this may indicate that there is feature leakage in the model.

#### 3.2.18

[M] Suppose you want to build a model to classify whether a tweet spreads misinformation. You have 100K labeled tweets over the last 24 months. You decide to randomly shuffle on your data and pick 80% to be the train split, 10% to be the valid split, and 10% to be the test split. What might be the problem with this way of partitioning?

One potential problem with this way of partitioning the data is that it may not take into account the temporal nature of the tweets. Since the data covers the last 24 months, it's possible that there are trends and patterns in the data that are time-dependent. Randomly shuffling the data and partitioning it into train, validation, and test sets may result in data leakage, where information from the future is inadvertently used to predict the past.

For example, if there was a spike in misinformation tweets during a particular time period, randomly partitioning the data may result in some of the validation and test tweets being from that time period, while the training data does not include that time period. This could result in an overly optimistic evaluation of the model's performance on the validation and test sets, as the model may have learned to predict the time period rather than the actual presence of misinformation.

To avoid this problem, it may be better to partition the data based on time, for example, using the first 80% of the tweets for training, the next 10% for validation, and the final 10% for testing. This would ensure that the model is evaluated on data that is representative of the future and has not been seen during training. Alternatively, techniques such as time-series cross-validation or sliding window validation can be used to evaluate the model's performance on temporally-dependent data.

#### 3.2.19

[M] You're building a neural network and you want to use both numerical and textual features. How would you process those different features?

This will be done by preprocessing textual data. It involves converting text into a numerical representation that can be used by a neural network. This can include techniques such as tokenization, stemming, and stopword removal. You may also need to apply encoding techniques such as one-hot encoding or word embeddings to represent words as numerical features that the neural network can understand.

#### 3.2.20

[H] Your model has been performing fairly well using just a subset of features available in your data. Your boss decided that you should use all the features available instead. What might happen to the training error? What might happen to the test error?

Hint: Think about the curse of dimensionality: as we use more dimensions to describe our data, the more sparse space becomes, and the further are data points from each other.

Adding more features to a model can exacerbate the curse of dimensionality. This is because as the number of features (i.e., dimensions) increases, the volume of the feature space grows exponentially, and the density of the data becomes sparser. This can make it harder for the model to identify patterns in the data and can increase

the risk of overfitting. This could affect the training and test error as under:

**Training error:** Adding more features can decrease the training error, as the model now has access to more information to fit the training data. However, it's also possible that the training error will increase, as the model may overfit the training data with the additional features.

**Test error:** Adding more features can increase the test error, as the model may become more complex and overfit the training data. This can lead to poor generalization performance on unseen test data. On the other hand, adding relevant features can decrease the test error by helping the model capture more relevant information about the data.

**Computational complexity:** Adding more features can increase the computational complexity of the model, making it slower to train and evaluate. This can make it harder to experiment with different hyperparameters and model architectures.

## 3.3 Objective functions, metrics, and evaluation

### 3.3.1

Convergence.

1. [E] When we say an algorithm converges, what does convergence mean?

When we say an algorithm converges, it means that the algorithm has reached a point where it has found a solution that is close enough to the optimal solution. In other words, the algorithm has achieved a level of accuracy that is acceptable for the given problem.

In the context of optimization problems, convergence means that the algorithm has found a minimum or maximum of a function. This can be achieved when the algorithm iteratively improves the solution by updating the solution in the direction of the gradient until it reaches a point where the gradient is very small or zero, indicating that the solution is close to the optimal solution.

Convergence is an important property of algorithms, especially in iterative algorithms such as optimization algorithms, where the algorithm repeatedly refines the solution until it converges. If an algorithm fails to converge, it means that it has not found a solution that is accurate enough for the given problem, and further improvements to the algorithm may be needed to achieve convergence.

2. [E] How do we know when a model has converged?

The convergence of a model depends on the type of problem and the algorithm used to train the model. There are different metrics and techniques that can be used to determine whether a model has converged. Here are some common methods:

- (i) **Loss function:** In supervised learning problems, the loss function measures the difference between the predicted output and the actual output. One way to check whether the model has converged is to monitor the loss function during training. If the loss function stops decreasing or reaches a plateau, it indicates that the model has converged.
- (ii) **Gradient:** In optimization problems, the gradient measures the direction and magnitude of the steepest ascent or descent of the function. An algorithm can be said to have converged when the gradient is close to zero or smaller than a predefined threshold.
- (iii) **Validation data:** Another way to check whether the model has converged is to use a validation dataset. The model can be trained on a training dataset and tested on a validation dataset. If the performance on the validation dataset stops improving or reaches a plateau, it indicates that the model has converged.
- (iv) **Early stopping:** Early stopping is a technique that can be used to prevent overfitting and improve the convergence of the model. It involves monitoring the performance on the validation dataset and stopping the training process when the performance stops improving or starts to degrade.

In general, the convergence of a model depends on the complexity of the problem, the size of the dataset, the choice of algorithm, and the hyperparameters used to train the model. It is important to carefully monitor the performance of the model during training and use appropriate techniques to determine whether the model has converged or not.

### 3.3.2

[E] Draw the loss curves for overfitting and underfitting.



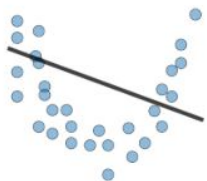
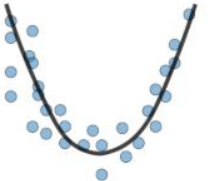
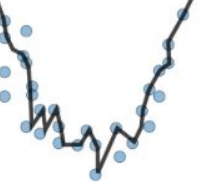
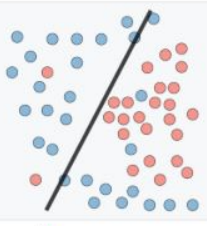
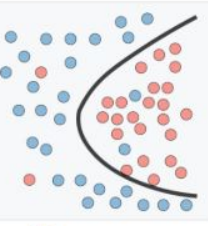
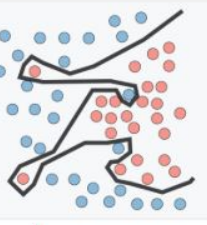

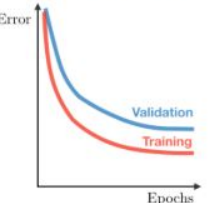

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> <li>• High training error</li> <li>• Training error close to test error</li> <li>• High bias</li> </ul>	<ul style="list-style-type: none"> <li>• Training error slightly lower than test error</li> </ul>	<ul style="list-style-type: none"> <li>• Very low training error</li> <li>• Training error much lower than test error</li> <li>• High variance</li> </ul>
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> <li>• Complexify model</li> <li>• Add more features</li> <li>• Train longer</li> </ul>		<ul style="list-style-type: none"> <li>• Perform regularization</li> <li>• Get more data</li> </ul>

Figure 3.1: Overfitting vs Underfitting

### 3.3.3

#### Bias-variance trade-off

1. [E] What's the bias-variance trade-off?

The bias-variance trade-off is a fundamental concept in machine learning that refers to the trade-off between model complexity and its ability to generalize to new data.

- Bias refers to the error that is introduced by approximating a real-world problem with a simpler model. It represents the difference between the expected predictions of a model and the true values of the target variable. High bias models tend to oversimplify the data and make strong assumptions about the relationship between the input features and the target variable. These models can underfit the data, which means they perform poorly on both the training and test data.
- Variance refers to the error that is introduced by a model that is overly sensitive to the training data. It represents the degree of variability of the model's predictions for different training sets. High variance models tend to overfit the data and capture the noise and random fluctuations in the training data. These models can perform well on the training data but generalize poorly to new, unseen data.

Therefore, the bias-variance trade-off refers to the balance between the simplicity of the model and its ability to capture the underlying patterns in the data. A good model should have low bias and low variance, meaning it is both accurate and robust. However, in practice, achieving this balance can be challenging, and finding the right balance often requires tuning the model's complexity, regularization, or hyperparameters.

2. [M] How's this trade-off related to overfitting and underfitting?

Overfitting occurs when a model fits the training data too closely, capturing the noise and random fluctuations in the training data, but fails to generalize well to new, unseen data. This is due to the model's high variance, meaning it is too complex and too closely fits the training data. As a result, the model's performance on the training data is good, but its performance on the test data is poor.

Underfitting, on the other hand, occurs when the model is too simple and has high bias, resulting in a model that cannot capture the underlying patterns in the data. This leads to poor performance on both the training and test data, as the model's predictions are too simple to capture the complexity of the data.

3. [M] How do you know that your model is high variance, low bias? What would you do in this case?

If a model has high variance and low bias, it means that it is overfitting the training data. You can recognize this by comparing the training error and validation error:

- If the training error is very low but the validation error is high, it indicates that the model is overfitting the training data and not generalizing well to new data.
- If the difference between the training error and validation error is large, it also indicates that the model is overfitting the training data.

To address this problem, we need to reduce the variance of the model while maintaining its bias. Here are some strategies to address high variance in a model:

- Reduce the complexity of the model by using simpler model architectures or reducing the number of features.
- Use regularization techniques such as L1 or L2 regularization to penalize complex models.
- Increase the amount of training data to reduce the noise in the training data.
- Use techniques such as dropout or ensemble learning to reduce overfitting.
- Increase the hyperparameters such as regularization parameter or dropout rate.

4. [M] How do you know that your model is low variance, high bias? What would you do in this case?

If a model has low variance and high bias, it means that it is underfitting the training data. You can recognize this by comparing the training error and validation error:

- If both the training error and validation error are high, it indicates that the model is underfitting the data and cannot capture the underlying patterns.
- If the training error is significantly higher than the validation error, it also indicates that the model is underfitting the data.

To address this problem, we need to increase the complexity of the model to reduce its bias while maintaining its variance. Here are some strategies to address high bias in a model:

- Use a more complex model architecture that is capable of capturing the underlying patterns in the data.
- Add more features to the model that are relevant to the problem.
- Use techniques such as data augmentation or feature engineering to increase the richness of the data.
- Decrease the regularization parameter to reduce the penalty for complex models.
- Use techniques such as dropout or ensemble learning to increase the model's flexibility.

### 3.3.4

Cross-validation.

1. [E] Explain different methods for cross-validation.

Cross-validation is a technique used in machine learning to evaluate the performance of a model. It involves dividing the data into several sets, with each set used for both training and testing the model. Here are some common methods for cross-validation:

- (i) K-fold cross-validation: This method involves dividing the data into K equal-sized subsets (or "folds"). The model is trained on K-1 folds and tested on the remaining fold. This process is repeated K times, with each fold used once as the test set. The results from each fold are then averaged to produce a final estimate of the model's performance.
- (ii) Stratified K-fold cross-validation: This is a variation of K-fold cross-validation that ensures that each fold contains roughly the same proportion of samples from each class. This is particularly useful for datasets with imbalanced classes.
- (iii) Leave-One-Out cross-validation (LOOCV): This method involves training the model on all samples except for one, which is then used for testing. This process is repeated for each sample in the dataset. LOOCV can be computationally expensive, but it provides an unbiased estimate of the model's performance.
- (iv) Shuffle-split cross-validation: This method involves randomly shuffling the data and then splitting it into a specified number of training and testing sets. The model is trained on the training set and tested on the testing set. This process is repeated a specified number of times, with different random splits used for each repetition.

- (v) Time series cross-validation: This method is used for time series data, where the order of the samples is important. It involves training the model on a subset of the data up to a certain point in time and testing it on the remaining data. This process is repeated for a specified number of times, with each iteration using a different point in time to divide the data into training and testing sets.

Overall, cross-validation is an important technique for evaluating the performance of machine learning models and selecting the best model for a given problem. The choice of cross-validation method depends on the nature of the dataset and the specific problem being addressed.

2. [M] Why don't we see more cross-validation in deep learning?

Cross-validation is a powerful technique for evaluating machine learning models, including deep learning models. However, there are several reasons why cross-validation may be less commonly used in deep learning compared to other machine learning techniques:

- Computationally expensive: Deep learning models are typically more computationally expensive to train compared to other machine learning models. Cross-validation involves training and testing the model multiple times, which can be time-consuming and resource-intensive.
- Large datasets: Deep learning models often require large amounts of data to train effectively. When working with large datasets, it may not be practical to use cross-validation due to the large computational requirements.
- Transfer learning: Deep learning models can benefit from transfer learning, where a pre-trained model is used as a starting point for training on a new dataset. In this case, cross-validation may not be necessary as the pre-trained model has already been trained on a large dataset.
- Overfitting and hyperparameter tuning: Deep learning models can be prone to overfitting, and hyperparameter tuning is often required to optimize the model's performance. Cross-validation can help to address these issues, but other techniques such as regularization and grid search may be more effective.

### 3.3.5

Train, valid, test splits.

1. [E] What's wrong with training and testing a model on the same data?

Training and testing a model on the same data is problematic because it can lead to overfitting. Overfitting occurs when a model learns the specific patterns and noise in the training data to the point where it becomes overly complex and cannot generalize well to new, unseen data. This means that the model may perform very well on the training data but poorly on new data, which defeats the purpose of building a machine learning model in the first place.

2. [E] Why do we need a validation set on top of a train set and a test set?

We need a validation set in addition to a train set and a test set for model selection and hyperparameter tuning. The train set is used to train the model, while the test set is used to evaluate the model's performance on new, unseen data. However, we also need a validation set to help us choose the best model and hyperparameters before testing on the test set.

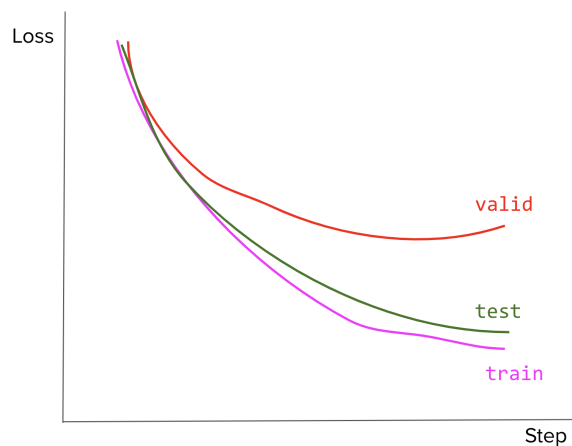
During model training, we adjust the model's parameters to minimize the loss function on the training data. However, this can lead to overfitting, where the model becomes too complex and specialized to the training data, and does not generalize well to new, unseen data. By using a validation set, we can evaluate the model's performance on new, unseen data that it has not been trained on, and use this to guide the model selection and hyperparameter tuning process.

We typically use the validation set to perform tasks such as selecting the best model architecture, adjusting the regularization parameters, and tuning the learning rate. We then evaluate the performance of the selected model on the test set, which provides a more accurate estimate of the model's ability to generalize to new data.

Using a separate validation set helps us to avoid overfitting and select the best model and hyperparameters, which ultimately leads to better performance on new, unseen data.

3. [M] Your model's loss curves on the train, valid, and test sets look like this. What might have been the cause of this? What would you do?

The loss curve shows lower loss for train and test data as compared the validation set loss. This can mean that the training and testing data might have same data points i.e. testing data was not separated from the training data or was a subset of the training data hence resulting in higher performance. Here, the performance on the validation data is true depictor of the model efficacy showing that the model is overfitting the training data since the validation loss is greater than training loss.



### 3.3.6

[E] Your team is building a system to aid doctors in predicting whether a patient has cancer or not from their X-ray scan. Your colleague announces that the problem is solved now that they've built a system that can predict with 99.99% accuracy. How would you respond to that claim?

Accuracy is a commonly used evaluation metric, but it can be misleading in situations where there is a class imbalance. For example, if 95% of the data belongs to one class and only 5% belongs to the other class, a classifier that always predicts the majority class will have a high accuracy of 95%, but it will fail to detect any instances of the minority class, resulting in a low recall score.

Accuracy might not be the correct metric for this kind of problem since it is inherently skewed dataset as very less number of X-ray scans are likely to show cancer i.e. chances of false positives or false negatives are high. For instance, if 9 out of 10 scans show not cancer and 1 shows cancer, and our model any predicts 9 of them correctly, the accuracy of the model is 90% even though it didn't necessarily predict the cancer patient scan correctly. This accuracy figure is misleading because of the underlying data property.

### 3.3.7

F1 score.

1. [E] What's the benefit of F1 over the accuracy?

F1 score is a better evaluation metric than accuracy in situations where there is an imbalanced class distribution or when the cost of false positives and false negatives is different.

F1 score takes into account both precision and recall, and is a better measure of a classifier's overall performance in situations where there is a class imbalance. F1 score combines precision and recall, and it provides a single score that represents the trade-off between them.

2. [M] Can we still use F1 for a problem with more than two classes. How?

Yes, F1 score can be used for problems with more than two classes. The F1 score can be computed for each class individually using a one-vs-all approach, where the F1 score is calculated for each class by treating that class as the positive class and all other classes as the negative class.

There are two commonly used methods to calculate the F1 score for multi-class classification problems: macro-average and micro-average.

- In macro-average, the F1 score is calculated independently for each class and then averaged across all classes, giving equal weight to each class. This method can be useful when the class distribution is balanced or when each class is equally important.
- In micro-average, the F1 score is calculated by considering all predictions and their corresponding true labels across all classes. This method gives more weight to the classes with more instances and can be useful when the class distribution is imbalanced or when some classes are more important than others.

In summary, F1 score can be used for multi-class classification problems by calculating the F1 score for each class individually using a one-vs-all approach and then averaging the scores using either macro-average or micro-average.

### 3.3.8

Given a binary classifier that outputs the following confusion matrix.

	Predicted True	Predicted False
Actual True	30 (TP)	20 (FP)
Actual False	5 (FN)	40 (TN)

Table 3.2: Confusion Matrix

1. [E] Calculate the model's precision, recall, and F1.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

For the question,

precision  $\rightarrow 30/30 + 20 \rightarrow 0.6$ ,

recall  $\rightarrow 30/30 + 5 \rightarrow 0.86$ ,

F1  $\rightarrow 2 * 0.6 * 0.86 / 0.86 + 0.6 \rightarrow 1.032 / 1.46 \rightarrow 0.71$

2. [M] What can we do to improve the model's performance?

### 3.3.9

Consider a classification where 99% of data belongs to class A and 1% of data belongs to class B.

1. [M] If your model predicts A 100% of the time, what would the F1 score be? Hint: The F1 score when A is mapped to 0 and B to 1 is different from the F1 score when A is mapped to 1 and B to 0.

If the model always predicts class A, then it means it will never predict class B. In this case, all the predictions for class B will be false negatives (i.e., the model predicts A when the true label is B), and there will be no false positives (i.e., the model predicts B when the true label is A).

Using the mapping where A is mapped to 0 and B to 1, the confusion matrix for the model would be:

Actual	Predicted as 0	Predicted as 1
Class 0	99% (TN)	0% (FP)
Class 1	1% (FN)	0% (TP)

From this confusion matrix, we can calculate the precision and recall for class B as follows:

$$Precision = \frac{TP}{TP + FP} = \frac{0}{0 + 0} = \text{undefined}$$

$$Recall = \frac{TP}{TP + FN} = \frac{0}{0 + 1\%} = 0$$

Since the precision is undefined (due to the absence of any positive predictions), we cannot calculate the f1 score using the standard formula. However, we can use the equivalent formula for f1 score that involves precision and recall as follows:

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall} = 0$$

Therefore, the F1 score for class B would be 0. Note that this result is not surprising, as the model is always predicting class A, and is not making any positive predictions for class B, resulting in a complete lack of ability to predict class B.

2. [M] If we have a model that predicts A and B at a random (uniformly), what would the expected F1 be?

To calculate the expected F1 score, we first need to understand the precision and recall of the model. Precision is the proportion of true positives (i.e., correctly predicted B) among all predicted positives (both true positives and false positives), while recall is the proportion of true positives among all actual positives (both true positives and false negatives).

Assuming the model predicts A and B at random (uniformly), the precision and recall of class B would be 1% each, as the model has an equal probability of predicting A or B. Therefore, the F1 score, which is the harmonic mean of precision and recall, would be:

$$\begin{aligned} F1 &= 2 * (precision * recall) / (precision + recall) \\ &= 2 * (0.01 * 0.01) / (0.01 + 0.01) \end{aligned}$$

= 0.01

Hence, the expected F1 score of the model would be 0.01, which indicates that the model is not effective in predicting class B. This is because the model is biased towards class A, and the small number of samples in class B is not sufficient to improve the model's performance.

### 3.3.10

[M] For logistic regression, why is log loss recommended over MSE (mean squared error)?

Logistic regression is a classification algorithm that models the probability of an input belonging to a particular class. Therefore, it is more appropriate to use a loss function that is suited for classification problems, such as log loss or cross-entropy loss, instead of a regression loss function like mean squared error (MSE).

Log loss, also known as binary cross-entropy, is a commonly used loss function for logistic regression. It is defined as:

$$-\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

where  $y_i$  is the true label (0 or 1),  $\hat{y}_i$  is the predicted probability of class 1, and  $N$  is the total number of samples. Log loss penalizes incorrect predictions more strongly than correct predictions, which makes it a good choice for imbalanced datasets where one class may have significantly fewer samples than the other.

On the other hand, MSE is a regression loss function that measures the average squared difference between the predicted and true values. While it can be used for binary classification problems by mapping class labels to numerical values (e.g., 0 and 1), it is not ideal because it penalizes errors symmetrically, which may not be appropriate for classification problems where false positives and false negatives may have different costs.

In summary, log loss is recommended over MSE for logistic regression because it is a more appropriate loss function for classification problems, especially when dealing with imbalanced datasets.

### 3.3.11

[M] When should we use RMSE (Root Mean Squared Error) over MAE (Mean Absolute Error) and vice versa?

RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) are both common regression evaluation metrics used to measure the accuracy of a model's predictions. While both metrics measure the average difference between the predicted and true values, they differ in the way they weight errors.

RMSE is calculated as the square root of the average squared difference between the predicted and true values:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

where  $n$  is the number of samples,  $y_i$  is the true value, and  $\hat{y}_i$  is the predicted value.

MAE, on the other hand, is calculated as the average absolute difference between the predicted and true values:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

The choice between RMSE and MAE depends on the specific context and requirements of the problem at hand. Here are some general guidelines:

- RMSE is more sensitive to outliers than MAE. This is because RMSE squares the errors, which amplifies the impact of large errors. Therefore, if outliers are particularly important to your problem, you may prefer to use MAE.
- RMSE is a good choice when you want to penalize large errors more heavily than small errors. This can be useful if the magnitude of errors is particularly important to your problem.
- MAE is more interpretable than RMSE because it represents the average absolute error in the same units as the target variable. This can be useful if you want to explain the accuracy of your model to non-technical stakeholders.

In general, RMSE is a more popular choice than MAE because it is more commonly used in the literature and many machine learning libraries default to using it. In summary, you should use RMSE when you want to penalize large errors more heavily and when the impact of outliers is not a major concern. You should use MAE when you want a more interpretable metric and when the impact of outliers is a concern.

### 3.3.12

[M] Show that the negative log-likelihood and cross-entropy are the same for binary classification tasks.

In binary classification, we are interested in predicting whether a sample belongs to one of two classes (e.g., class 0 or class 1). The output of a binary classification model is usually a single scalar value between 0 and 1, which represents the predicted probability that the sample belongs to class 1.

The negative log-likelihood and cross-entropy are both loss functions that can be used to train a binary classification model. Here's how we can show that they are the same:

Let  $y$  be the true label of a sample (0 or 1) and let  $\hat{y}$  be the predicted probability that the sample belongs to class 1. We can write the probability distribution of  $y$  given  $\hat{y}$  as:

$$P(y|\hat{y}) = \hat{y}^y(1 - \hat{y})^{1-y}$$

The negative log-likelihood is simply the negative logarithm of this distribution:

$$\mathcal{L}_{\text{NLL}} = -\log P(y|\hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

The cross-entropy loss is a generalization of the negative log-likelihood that applies to multiple classes. In the case of binary classification, it is simply the negative logarithm of the predicted probability of the true class:

$$\mathcal{L}_{\text{CE}} = -\log(\hat{y}) \quad \text{if } y = 1$$

$$\mathcal{L}_{\text{CE}} = -\log(1 - \hat{y}) \quad \text{if } y = 0$$

We can see that the cross-entropy loss is equivalent to the negative log-likelihood, because when  $y = 1$ , the cross-entropy loss reduces to  $-\log(\hat{y})$ , which is the same as the second term of the negative log-likelihood. Similarly, when  $y = 0$ , the cross-entropy loss reduces to  $-\log(1 - \hat{y})$ , which is the same as the third term of the negative log-likelihood. Therefore, we can conclude that the negative log-likelihood and cross-entropy are the same for binary classification tasks.

### 3.3.13

[M] For classification tasks with more than two labels (e.g. MNIST with 10 labels), why is cross-entropy a better loss function than MSE?

Cross-entropy is a better loss function than mean squared error (MSE) for classification tasks with more than two labels (i.e., multiclass classification) because it is designed to optimize the probability distribution of the predicted labels, whereas MSE is designed to minimize the difference between the predicted and actual values. In multiclass classification, we want to predict the probability distribution over the different classes for each input sample. Cross-entropy loss measures the dissimilarity between the predicted probability distribution and the true probability distribution of the classes. It does so by calculating the log loss of the predicted probabilities for the true class labels. This ensures that the predicted probabilities of the classes sum up to one and that the probabilities are well calibrated. Cross-entropy loss can also handle cases where the true distribution is sparse, i.e., where there are many classes but only a few of them are relevant for a particular sample.

MSE, on the other hand, measures the average squared difference between the predicted and true values. For multiclass classification, this would require one output neuron per class and would predict a continuous value for each class, which is not suitable for classification tasks.

In summary, cross-entropy loss is preferred over MSE for multiclass classification tasks because it is designed to optimize the probability distribution over the predicted labels, which is the goal of a classification model.

### 3.3.14

[E] Consider a language with an alphabet of 27 characters. What would be the maximal entropy of this language?

The maximal entropy of a language with an alphabet of 27 characters can be calculated using the formula for entropy:

$$H = -\sum P(x) \log_2 P(x)$$

where  $P(x)$  is the probability of observing a character  $x$  in the language.

Since there are 27 characters in the language, we can assume that each character is equally likely to occur, i.e.,  $P(x) = 1/27$  for all characters  $x$ . Substituting this into the entropy formula, we get:

$$H = -\sum (1/27) \log_2(1/27)$$

$$= -(27 * (1/27) * \log_2(1/27))$$

$$= -\log_2(1/27)$$

$\approx 4.7549\text{bits}$

Therefore, the maximal entropy of this language would be approximately 4.7549 bits.

### 3.3.15

[E] A lot of machine learning models aim to approximate probability distributions. Let's say  $P$  is the distribution of the data and  $Q$  is the distribution learned by our model. How do we measure how close  $Q$  is to  $P$ ?

There are various measures to evaluate how close the learned distribution  $Q$  is to the true distribution  $P$  in machine learning. Here are some commonly used ones:

1. Kullback-Leibler (KL) Divergence: KL divergence measures the difference between two probability distributions  $P$  and  $Q$ . It is defined as  $KL(P||Q) = \sum P(x)\log[P(x)/Q(x)]$ . A smaller KL divergence indicates a closer match between the two distributions.
2. Jensen-Shannon (JS) Divergence: JS divergence is a symmetric variation of KL divergence, which measures the similarity between two probability distributions. It is defined as  $JS(P, Q) = 0.5KL(P||(P+Q)/2) + 0.5KL(Q||(P+Q)/2)$ . A smaller JS divergence indicates a closer match between the two distributions.
3. Earth Mover's Distance (EMD): EMD measures the minimum amount of work required to transform one distribution into the other. It is also known as Wasserstein distance. A smaller EMD indicates a closer match between the two distributions.
4. Total Variation (TV) Distance: TV distance measures the absolute difference between the probabilities of the same event in two distributions. It is defined as  $TV(P, Q) = 0.5 \sum |P(x) - Q(x)|$ . A smaller TV distance indicates a closer match between the two distributions.
5. Maximum Mean Discrepancy (MMD): MMD measures the distance between the means of two probability distributions. It is defined as  $MMD(P, Q) = \|\mu_P - \mu_Q\|^2$ , where  $\mu_P$  and  $\mu_Q$  are the means of distributions  $P$  and  $Q$ , respectively. A smaller MMD indicates a closer match between the two distributions.

These measures can be used to compare the learned distribution  $Q$  with the true distribution  $P$  and evaluate the quality of the machine learning model.

### 3.3.16

MPE (Most Probable Explanation) vs. MAP (Maximum A Posteriori)

1. [E] How do MPE and MAP differ?

MPE (Maximum Posterior Estimation) and MAP (Maximum A Posteriori) are both methods used in Bayesian inference to estimate the value of a parameter given some observed data. However, they differ in their approach and the kind of result they produce.

MPE estimates the value of the parameter that maximizes the likelihood of the observed data, without taking into account any prior information about the parameter. In other words, it seeks the parameter that makes the observed data most likely, without considering any additional information that may be available.

On the other hand, MAP estimation seeks the parameter that maximizes the posterior probability of the parameter given the observed data, taking into account both the likelihood of the observed data and any prior information about the parameter. The prior distribution represents the additional information we have about the parameter before observing any data.

2. [H] Give an example of when they would produce different results.

An example of when MPE and MAP would produce different results is in medical diagnosis. Suppose we want to diagnose a rare disease that occurs in 1% of the population. We have a diagnostic test that is 95% accurate, meaning that if a person has the disease, the test will correctly identify them as positive 95% of the time, and if a person does not have the disease, the test will correctly identify them as negative 95% of the time.

Suppose we have a patient who tests positive for the disease. If we use MPE, we would estimate that the patient has the disease, because the likelihood of a positive test result given the disease is 95%, which is higher than the likelihood of a positive test result given no disease (5%). However, if we use MAP, we would consider the prior probability of the patient having the disease, which is only 1%. If we assume a simple uniform prior distribution, then the posterior probability of the patient having the disease given the positive test result is only 16%, which is much lower than the likelihood of a positive test result given the disease.



### 3.3.17

[E] Suppose you want to build a model to predict the price of a stock in the next 8 hours and that the predicted price should never be off more than 10% from the actual price. Which metric would you use?

If the goal is to ensure that the predicted price is not off by more than 10% from the actual price, a suitable metric to use would be the mean absolute percentage error (MAPE). MAPE measures the percentage difference between the predicted and actual values, and is expressed as a percentage.

The formula for MAPE is:

$$\text{MAPE} = (1/n) * \sum \left| \frac{\text{actual} - \text{predicted}}{\text{actual}} \right| * 100\%$$

where n is the number of predictions made.

By setting the maximum allowable error to 10%, we can compute the MAPE of the predictions and ensure that it does not exceed 10%. If the MAPE exceeds 10%, it indicates that the predicted values are off by more than 10% from the actual values, and the model needs to be improved.

It is important to note that MAPE has some limitations, such as being sensitive to small actual values, and producing undefined values when actual values are zero. In such cases, other metrics such as mean absolute error (MAE) or mean squared error (MSE) can be used as alternatives.



# Chapter 4

## Machine Learning Algorithms

### 4.1 Classical Machine Learning: Questions

#### 4.1.1

[E] What are the basic assumptions to be made for linear regression?

Linear regression is a statistical method used to establish a relationship between a dependent variable and one or more independent variables. The basic assumptions for linear regression are:

1. **Linearity:** The relationship between the dependent variable and independent variables should be linear, i.e., a straight line should be able to capture the relationship between them.
2. **Independence:** The observations should be independent of each other, i.e., the value of the dependent variable for one observation should not be influenced by the value of the dependent variable for another observation.
3. **Homoscedasticity:** The variance of the errors (the difference between the predicted and actual values of the dependent variable) should be constant across all levels of the independent variable(s).
4. **Normality:** The residuals (the difference between the predicted and actual values of the dependent variable) should be normally distributed, with a mean of zero.
5. **No Multicollinearity:** The independent variables should not be highly correlated with each other.

If these assumptions are not met, it can affect the accuracy of the linear regression model and its predictions. It is important to check these assumptions before using linear regression and if necessary, consider using other statistical methods.

#### 4.1.2

[E] What happens if we don't apply feature scaling to logistic regression?

Logistic regression is a statistical method used to model the probability of a binary outcome (i.e., 0 or 1) based on one or more independent variables. Feature scaling is the process of standardizing the range of independent variables so that they all have a similar scale, which is often necessary for certain machine learning algorithms to work effectively.

If we don't apply feature scaling to logistic regression, it may lead to some issues:

1. **Slow convergence:** Without feature scaling, logistic regression may take longer to converge to an optimal solution. This is because the algorithm may take longer to reach the minimum value of the cost function due to differences in the scales of the independent variables.
2. **Unbalanced contribution of features:** If some independent variables have a large range of values, they may dominate the cost function and contribute more to the final result compared to other independent variables, which can result in inaccurate predictions.
3. **Incorrect predictions:** If the independent variables are not scaled, it may cause the logistic regression model to give less importance to variables with smaller values and overemphasize variables with larger values, leading to incorrect predictions.

Therefore, it is generally recommended to apply feature scaling to logistic regression to ensure that the algorithm performs optimally and provides accurate predictions.

### 4.1.3

[E] What are the algorithms you'd use when developing the prototype of a fraud detection model?

When developing a prototype for a fraud detection model, several algorithms can be used depending on the type and nature of the data. Here are some commonly used algorithms:

1. Logistic Regression: Logistic regression is a simple and widely used algorithm for binary classification problems like fraud detection. It works well for datasets with a large number of features and is computationally efficient. It is also useful for interpreting the impact of each feature on the target variable.
2. Decision Trees: Decision trees are a tree-like model that partitions the data into smaller subsets based on features. It is an intuitive and interpretable algorithm that can handle both categorical and numerical data. Decision trees can also identify complex interactions between variables.
3. Random Forest: Random Forest is an ensemble learning algorithm that uses multiple decision trees to improve the accuracy of the predictions. It works well for high dimensional data and can handle missing values and outliers. Random Forest is also useful in detecting fraud due to its ability to detect the importance of features.
4. XGBoost: XGBoost is a gradient boosting algorithm that works well for imbalanced datasets. It is also efficient and can handle both numerical and categorical features. XGBoost is particularly useful in fraud detection due to its ability to detect the importance of features and handle missing values.
5. Neural Networks: Neural Networks are deep learning algorithms that can learn complex non-linear relationships in data. They work well with large datasets and can handle both numerical and categorical data. Neural Networks can detect complex fraud patterns that other algorithms may miss.

These are some of the commonly used algorithms for developing a prototype of a fraud detection model. However, the choice of algorithm(s) will depend on the type and nature of the data, as well as the specific requirements and objectives of the project.

### 4.1.4

Feature selection.

1. [E] Why do we use feature selection?

Feature selection is the process of selecting a subset of relevant features (variables or attributes) from a larger set of potential features to use in building a machine learning model. Feature selection is an important step in the machine learning process as it can improve model accuracy, reduce overfitting, and increase interpretability of the model.

There are several reasons why we use feature selection, including:

- (a) Improved model accuracy: By selecting only the most relevant features, we can reduce noise and increase signal in the data, leading to better model accuracy.
- (b) Reduced overfitting: When we use too many features, our model may become too complex and overfit to the training data. Feature selection can help reduce the number of features and thus reduce overfitting.
- (c) Faster training and prediction times: Using fewer features can speed up the training and prediction times of the model, as there are fewer calculations to be made.
- (d) Improved interpretability: By selecting only the most important features, we can gain a better understanding of the factors that are driving the model's predictions, making it more interpretable.

Overall, feature selection is a crucial step in the machine learning process that can help improve model accuracy, reduce overfitting, and increase interpretability.

2. [M] What are some of the algorithms for feature selection? Pros and cons of each.

There are several algorithms for feature selection, each with its own strengths and weaknesses. Here are some of the most common algorithms:

- (a) Filter methods: Filter methods evaluate the relevance of each feature independently of the others and rank them based on statistical measures. Examples of filter methods include correlation-based feature selection, chi-square test, and information gain.
  - Pros: Filter methods are computationally efficient and can be used with any type of model. They are also easy to interpret and can help identify the most relevant features quickly.
  - Cons: Filter methods do not take into account the interactions between features and may not always select the best subset of features.

- (b) Wrapper methods: Wrapper methods select features by testing different subsets of features and evaluating the performance of the model using each subset. Examples of wrapper methods include forward selection, backward elimination, and recursive feature elimination.
  - Pros: Wrapper methods can identify the best subset of features for a particular model and take into account the interactions between features.
  - Cons: Wrapper methods can be computationally expensive and may overfit to the training data.
- (c) Embedded methods: Embedded methods select features as part of the model building process. Examples of embedded methods include Lasso regression, Ridge regression, and decision trees.
  - Pros: Embedded methods can select the most relevant features and improve model accuracy.
  - Cons: Embedded methods can be computationally expensive and may not always select the best subset of features.
- (d) Dimensionality reduction methods: Dimensionality reduction methods reduce the number of features by transforming the original features into a lower dimensional space. Examples of dimensionality reduction methods include principal component analysis (PCA), linear discriminant analysis (LDA), and t-distributed stochastic neighbor embedding (t-SNE).
  - Pros: Dimensionality reduction methods can identify the most important features and reduce computational complexity.
  - Cons: Dimensionality reduction methods may lose some information during the transformation process and may not always improve model accuracy.

Overall, each algorithm has its own advantages and disadvantages, and the choice of algorithm depends on the specific problem and data set.

#### 4.1.5

k-means clustering.

1. [E] How would you choose the value of k?

Choosing the value of k (the number of clusters) in k-means clustering is an important task, as it can affect the quality of the resulting clusters. Here are a few common methods for selecting k:

- (a) Elbow method: Plot the sum of squared distances (SSE) for different values of k and look for the "elbow point" where the curve starts to flatten out. This can be an indication of the point where adding more clusters does not significantly improve the clustering performance. However, this method can sometimes be ambiguous and subjective.
- (b) Silhouette method: Compute the silhouette score for different values of k and choose the k with the highest average silhouette score. The silhouette score measures the quality of a clustering by computing the distance between a data point and its own cluster compared to the distance between that data point and the other clusters. A higher silhouette score indicates better clustering.
- (c) Domain knowledge: If you have prior knowledge or insights about the problem and the data, you can use that information to choose an appropriate value of k. For example, if you know that the data should be partitioned into a specific number of groups based on some domain-specific criteria, you can use that as the value of k.
- (d) Hierarchical clustering: Use hierarchical clustering to create a dendrogram of the data and visually inspect it to determine an appropriate value of k. This method can be useful when the data has a complex or hierarchical structure.
- (e) Trial and error: Try different values of k and evaluate the resulting clusters using some performance metric (such as SSE, silhouette score, or external validation measures). This method can be time-consuming but can give you a better understanding of how different values of k affect the clustering performance.

It's important to note that there is no "best" method for choosing k, and different methods may work better for different datasets and applications. Therefore, it's often a good idea to try multiple methods and compare the results to choose the best value of k.

2. [E] If the labels are known, how would you evaluate the performance of your k-means clustering algorithm?

If the labels are known, you can use several evaluation metrics to measure the performance of your k-means clustering algorithm. Here are a few commonly used metrics:

- (a) Accuracy: You can calculate the accuracy of the k-means clustering algorithm by comparing the predicted labels with the true labels. The accuracy is defined as the number of correct predictions divided by the total number of predictions.



```
# cluster with kmeans
Kmean = skl_cluster.KMeans(n_clusters=2)
Kmean.fit(circles)
clusters = Kmean.predict(circles)

# cluster with spectral clustering
model = skl_cluster.SpectralClustering(n_clusters=2, affinity='
nearest_neighbors', assign_labels='kmeans')
labels = model.fit_predict(circles)

# plot comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
ax1.scatter(circles[:, 0], circles[:, 1], s=15, linewidth=0.1, c=
clusters, cmap='flag')
ax2.scatter(circles[:, 0], circles[:, 1], s=15, linewidth=0, c=
labels, cmap='flag')
plt.show()
```

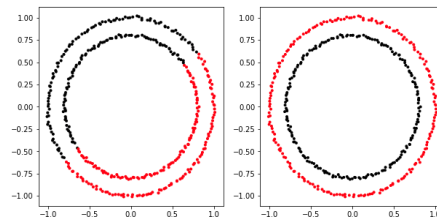


Figure 4.2: Spectral vs k-means for radial data distribution

#### 4.1.6

k-nearest neighbor classification.

answers mainly taken from this [article](#).

1. [E] How would you choose the value of k?

The choice of the value of k in k-nearest neighbors (KNN) classification can have a significant impact on the performance of the algorithm. Here are some general guidelines to help you choose the appropriate value of k:

- (a) Choose an odd value of k to avoid ties when the algorithm has to decide between two classes.
- (b) Choose a small value of k if the dataset is noisy or if the decision boundary between classes is complex, as a smaller value of k will make the decision boundary more flexible.
- (c) Choose a large value of k if the dataset is clean or if the decision boundary between classes is simple, as a larger value of k will make the decision boundary smoother.
- (d) Choose a value of k that is not too small or too large, typically between 3 and 10. This is because a small value of k may lead to overfitting, while a large value of k may lead to underfitting.
- (e) Use cross-validation to evaluate the performance of the KNN algorithm for different values of k and choose the value that gives the best performance on the validation set.
- (f) Consider the computational cost of the algorithm, as a larger value of k will require more computational resources to calculate the distances between instances.

Overall, the choice of k in KNN classification depends on the specific characteristics of the dataset and the problem at hand. It is a hyperparameter that needs to be tuned carefully to achieve the best performance.

2. [E] What happens when you increase or decrease the value of k?

When K is small, we are restraining the region of a given prediction and forcing our classifier to be “blind” to the overall distribution. A small value for K provides the most flexible fit, which will have low bias but high variance, hence more prone to overfitting. Graphically, our decision boundary will be more jagged. On the other hand, a higher K averages more points in each prediction and hence is more resilient to outliers. Larger values of K will have smoother decision boundaries which mean lower variance but increased bias hence more prone to underfitting.

3. [M] How does the value of k impact the bias and variance?

small k → high variance low bias

large k → high bias low variance

4.1.7

k-means and GMM are both powerful clustering algorithms.

(For in-depth understanding, read this [article](#).)

- 1. [M] Compare the two.

	k-means	GMM
Deterministic vs. Probabilistic Approach	uses a deterministic approach and assigns each data point to one unique cluster. This is referred to as a hard clustering method.	uses a probabilistic approach and gives the probability of each data point belonging to any of the clusters. This is referred to as a soft clustering method.
Parameters	only uses two parameters: the number of clusters K and the centroid locations	uses three parameters: the number of clusters K, mean, and cluster covariances
Updating the centroids	uses only the data points assigned to each specific cluster to update the centroid mean	uses all data points within the data set to update the centroid weights, means and covariances
Feature Standardization	the ‘circular’ distance measure around a centroid can make feature standardization necessary if one or more of the dimensions will dominate the calculation	automatically takes the issue into account by its calculation and use of the covariance matrix

Table 4.1: k-means vs GMMs

- 2. [M] When would you choose one over another?  
Hint: Here’s an example of how K-means and GMM algorithms perform on the artificial mouse dataset.

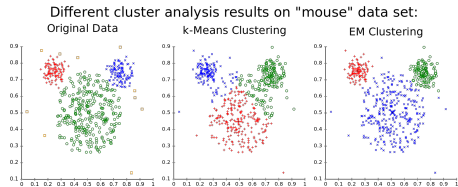


Figure 4.3: Spectral vs k-means for radial data distribution

If the data has a small number of clusters and a simple structure, k-means is a good choice due to its simplicity and efficiency. On the other hand, if the data has a complex structure with a large number of clusters, GMM is a better choice due to its ability to capture the underlying probability distribution of the data. However, GMM is computationally more expensive than k-means and may require more computational resources to run.

4.1.8

Bagging and boosting are two popular ensembling methods. Random forest is a bagging example while XGBoost is a boosting example. (More detail in this [article](#).)

- 1. [M] What are some of the fundamental differences between bagging and boosting algorithms?
- 2. [M] How are they used in deep learning?

Bagging and boosting techniques are not as commonly used in deep learning as they are in other areas of machine learning. However, there are some variations and extensions of these techniques that have been applied to deep learning models.

One common variation of bagging in deep learning is known as dropout. Dropout is a regularization technique that involves randomly dropping out some neurons during training to prevent overfitting. This can be seen as a form of bagging because it trains multiple versions of the network, each with a different set of active neurons. During testing, all neurons are active, and the final prediction is made by combining the predictions of all the networks. This helps to reduce the variance of the model and improve its generalization performance.

In deep learning, boosting techniques are often used to improve the training of neural networks. One popular boosting technique in deep learning is called stochastic gradient boosting, which involves sequentially training a sequence of neural networks on weighted versions of the training data. The weights of



Bagging	Boosting
Involves creating multiple subsets of the training data and training individual models on each subset	Involves training a sequence of weak learners on weighted versions of the training data
The individual models are trained independently on different subsets of the training data	The individual models are trained sequentially, with each model learning from the errors of the previous model
The predictions of the individual models are combined using averaging or voting	The predictions of the individual models are combined using weighted averaging
Primarily used to reduce the variance of the model	Primarily used to reduce bias but can be used to reduce variance of the model as well
Examples include Random Forest and Bagged Decision Trees	Examples include AdaBoost and Gradient Boosting

Table 4.2: Bagging vs Boosting

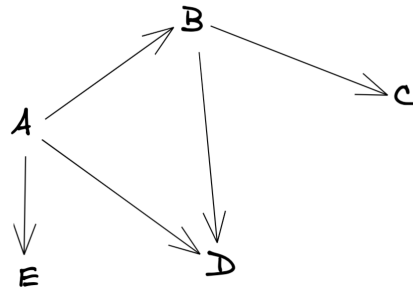


Figure 4.4: Directed graph

the samples are adjusted during each iteration based on their classification error, and the final prediction is made by combining the predictions of all the networks. Stochastic gradient boosting can help to reduce both the bias and variance of the model and improve its overall performance.

#### 4.1.9

Given this directed graph.

1.  $[E]$  Construct its adjacency matrix.

$$\begin{bmatrix}
 & A & B & C & D & E \\
 A & 0 & 1 & 0 & 1 & 1 \\
 B & 0 & 0 & 1 & 1 & 0 \\
 C & 0 & 0 & 0 & 0 & 0 \\
 D & 0 & 0 & 0 & 0 & 0 \\
 E & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}$$

2.  $[E]$  How would this matrix change if the graph is now undirected?

$$\begin{bmatrix}
 & A & B & C & D & E \\
 A & 0 & 1 & 0 & 1 & 1 \\
 B & 1 & 0 & 1 & 1 & 0 \\
 C & 0 & 1 & 0 & 0 & 0 \\
 D & 1 & 1 & 0 & 0 & 0 \\
 E & 1 & 0 & 0 & 0 & 0
 \end{bmatrix}$$

3.  $[M]$  What can you say about the adjacency matrices of two isomorphic graphs?

Two graphs are isomorphic if and only if for some ordering of their vertices their adjacency matrices are equal.

If two graphs are isomorphic, then they have the same structure, but their vertices and edges may be labeled differently. The adjacency matrix of a graph is a matrix that represents the connections between the vertices of the graph. Specifically, the  $(i, j)$ -entry of the adjacency matrix is 1 if there is an edge between vertices  $i$  and  $j$ , and 0 otherwise.

Since isomorphic graphs have the same structure, they will have the same adjacency matrix up to a permutation of their rows and columns. This means that if you relabel the vertices of one graph to match the labeling of the vertices of the other graph, then the resulting adjacency matrices will be identical.

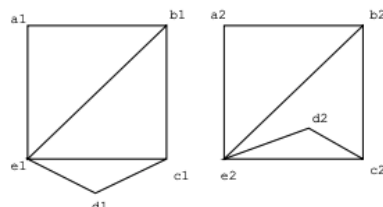


Figure 4.5: Two isomorphic graphs

In other words, given two isomorphic graphs  $G$  and  $H$ , there exists a permutation matrix  $P$  such that the adjacency matrix of  $G$  is equal to  $P$  times the adjacency matrix of  $H$  times the transpose of  $P$ .

Formally, if  $G$  and  $H$  are isomorphic graphs with  $n$  vertices, then there exists a permutation matrix  $P$  such that:

$$A(G) = P * A(H) * P^T$$

where  $A(G)$  and  $A(H)$  are the adjacency matrices of  $G$  and  $H$ , respectively, and  $P^T$  is the transpose of  $P$ .

#### 4.1.10

Imagine we build a user-item collaborative filtering system to recommend to each user items similar to the items they've bought before.

1.  $[M]$  You can build either a user-item matrix or an item-item matrix. What are the pros and cons of each approach?

**User-Item Matrix:** In a user-item matrix, the rows represent users, the columns represent items, and the cells contain the ratings given by users for each item. The main advantage of this approach is that it is easy to understand and interpret. The user-item matrix can be used to identify users who have similar tastes and recommend items that other similar users have enjoyed. However, the user-item matrix suffers from the sparsity problem, meaning that most users only rate a small subset of items, leading to a lot of missing values in the matrix. As a result, it may be difficult to make accurate recommendations for users who have rated only a few items.

**Item-Item Matrix:** In an item-item matrix, the rows represent items, the columns represent users, and the cells contain the ratings given by users for each item. The main advantage of this approach is that it can handle the sparsity problem much better than the user-item matrix. Since the number of items is usually much smaller than the number of users, the item-item matrix is less sparse, making it easier to identify similar items and make recommendations. Additionally, the item-item matrix can be precomputed and stored, making it faster to generate recommendations in real-time. However, the item-item matrix may be more difficult to interpret, since the similarities are based on the behavior of users who have rated the items, rather than the similarities between users.

2.  $[E]$  How would you handle a new user who hasn't made any purchases in the past?

One approach to handle a new user is to use a content-based filtering approach. Content-based filtering recommends items to a user based on the characteristics of the items they have interacted with or shown interest in, rather than the behavior of other users. For example, if the user has provided some demographic information or indicated interests, preferences, or other attributes, we can use that information to suggest items that align with those attributes.

Another approach is to use a hybrid recommender system that combines both collaborative filtering and content-based filtering techniques. In this case, for new users with little or no rating history, we can use the content-based approach to recommend items based on their profile, preferences, or some other data that may be available. Once the user has interacted with a few items, we can switch to collaborative filtering to make recommendations based on the behavior of similar users.

Finally, one common technique to handle a new user is to use a popular items approach. In this case, we can simply recommend the most popular or best-selling items to the new user. This approach may be less personalized but can be effective in providing the user with a starting point and encouraging them to start interacting with the system.

In summary, there are several approaches to handle a new user who hasn't made any purchases in the past. The choice of which approach to use depends on the specific requirements of the recommendation system and the characteristics of the dataset.

#### 4.1.11

[E] Is feature scaling necessary for kernel methods?

Feature scaling is often necessary for kernel methods to perform well. Kernel methods, such as Support Vector Machines (SVMs) and kernelized Ridge Regression, rely on the notion of measuring similarity between pairs of data points using a kernel function. The kernel function is typically computed as a dot product between feature vectors.

If the features are not scaled properly, some features may have a much larger range of values than others, which can lead to unstable or biased results. For example, consider a dataset with two features: age in years and income in thousands of dollars. If age is not scaled down, then the similarity metric computed between two data points will be dominated by the age feature, and the income feature may not have as much of an impact. To address this issue, it is generally recommended to scale the features to have similar ranges of values before using kernel methods. Common techniques for feature scaling include standardization (subtracting the mean and dividing by the standard deviation) or scaling to a fixed range (e.g., between 0 and 1). By doing this, each feature will contribute equally to the similarity measure and the kernel function will be more robust and accurate.

#### 4.1.12

Naive Bayes classifier.

1. [E] How is Naive Bayes classifier naive?

Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. The "naive" in Naive Bayes refers to the simplifying assumption made by the algorithm that all features (or input variables) are independent of each other. In other words, it assumes that the presence of one feature does not affect the presence of any other feature in the dataset.

This assumption, while simplifying the calculations required to estimate the probability of a particular class given a set of input features, is often not true in real-world datasets. There are usually correlations or dependencies between features in a dataset. Despite this simplifying assumption, the Naive Bayes classifier has been shown to work well in practice for a wide range of classification problems.

2. [M] Let's try to construct a Naive Bayes classifier to classify whether a tweet has a positive or negative sentiment. We have four training samples:

Tweet	Label
This makes me so upset	Negative
This puppy makes me happy	Positive
Look at this happy hamster	Positive
No hamsters allowed in my house	Negative

According to your classifier, what's sentiment of the sentence "The hamster is upset with the puppy"?

Vocabulary after removing stop wrds and tokenizing: makes, upset, puppy, happy, look, hamster, allowed, no, house.

Prior probabilities:

$$P(Positive) = 2/4 = 0.5$$

$$P(Negative) = 2/4 = 0.5$$

Conditional probabilities:

$$P(makes|Pos) = 1/2$$

$$P(upset|Pos) = 0$$

$$P(puppy|Pos) = 1/2$$

$$P(happy|Pos) = 1$$

$$P(look|Pos) = 1/2$$

$$P(hamster|Pos) = 1/2$$

$$P(allowed|Pos) = 0$$

$$P(no|Pos) = 0$$

$$P(house|Pos) = 0$$

$$P(makes|Neg) = 1/2$$

$$P(upset|Neg) = 1/2$$

$$P(puppy|Neg) = 0$$

$$P(happy|Neg) = 0$$

$$P(look|Neg) = 0$$

$$P(hamster|Neg) = 1/2$$

$$P(allowed|Neg) = 1/2$$

$$P(no|Neg) = 1/2$$

$$P(house|Neg) = 1/2$$

So, for our sentence: "The hamster is upset with the puppy", we calculate the probability using the formula:

$$P(Positive|words) = P(words|Positive) * P(Positive)$$
$$P(Negative|words) = P(words|Negative) * P(Negative)$$

The words are hamster, upset, puppy.

$$P(Positive|words) = P(hamster|Pos)*P(upset|Pos)*P(puppy|Pos)*P(Positive) \rightarrow 1/2*0*1/2*1/2 = 0$$

$$P(Negative|words) = P(hamster|Neg)*P(upset|Neg)*P(puppy|Neg)*P(Negative) \rightarrow 1/2*1/2*0*1/2 = 0$$

Hence, we can't determine the sentiment of the sentence.

#### 4.1.13

Two popular algorithms for winning Kaggle solutions are Light GBM and XGBoost. They are both gradient boosting algorithms.

1. [E] What is gradient boosting?

Gradient Boosting is a popular machine learning technique used for both regression and classification problems. It is an ensemble method that combines several weak learners (usually decision trees) to create a strong learner. Gradient Boosting works by iteratively adding decision trees to the model, where each subsequent tree attempts to correct the errors of the previous trees.

At a high level, the gradient boosting algorithm works as follows:

- i Initialize the model with a single decision tree.
  - ii Calculate the errors (residuals) between the predicted values and the actual values for the training data.
  - iii Fit a new decision tree to the residuals.
  - iv Update the model by adding the new decision tree, weighted by a learning rate.
  - v Repeat steps 2-4 until a stopping criterion is met (e.g., a maximum number of trees, or until the validation error stops improving).
2. [M] What problems is gradient boosting good for?

Gradient Boosting is particularly good for problems where the data is complex and difficult to model with a single decision tree or linear model. It can often achieve higher accuracy than other machine learning methods, such as Random Forests or Neural Networks, especially when dealing with high-dimensional data with many features. Additionally, Gradient Boosting is known to be robust to overfitting, as long as the learning rate is tuned properly and a sufficient number of trees is used.

Some specific applications where Gradient Boosting has been successful include:

- Predicting housing prices or stock prices
- Predicting customer churn or fraud detection in financial transactions
- Classifying images or detecting objects in images
- Natural Language Processing (NLP) tasks, such as sentiment analysis or text classification.

#### 4.1.14

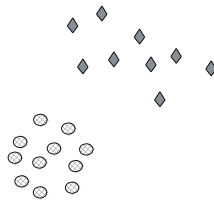
SVM.

1. [E] What's linear separation? Why is it desirable when we use SVM?

Linear separation is a concept used in machine learning and specifically in support vector machines (SVMs). In SVMs, the goal is to find a hyperplane (a line in 2D or a plane in 3D) that separates data into different classes. Linear separation refers to the ability to draw such a hyperplane that can perfectly separate the data into different classes.

Linear separation is desirable in SVM because it ensures that the classifier can make accurate predictions on new, unseen data. If the data is linearly separable, the SVM can confidently predict which class a new data point belongs to based on which side of the hyperplane it falls. If the data is not linearly separable, the SVM may not be able to find a hyperplane that can perfectly separate the data, leading to misclassifications and reduced accuracy.

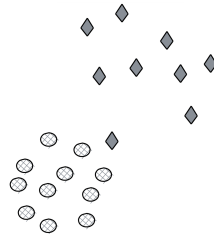
In summary, SVMs aim to find the best hyperplane that can separate the data into different classes, and linear separation is desirable because it leads to accurate predictions on new data.



2. [M] How well would vanilla SVM work on this dataset?

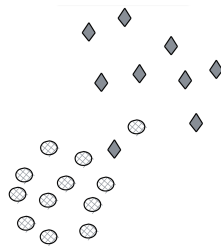
Clean margins, perfect classification

3. [M] How well would vanilla SVM work on this dataset?



One miss-classification for the diamonds class

4. [M] How well would vanilla SVM work on this dataset?



Two miss-classifications, one for diamond, the other for circle.

## 4.2 Deep Learning Architectures and Applications: Natural Language Processing

## 4.3 Deep Learning Architectures and Applications: Computer Vision

### 4.3.1

[M] For neural networks that work with images like VGG-19, InceptionNet, you often see a visualization of what type of features each filter captures. How are these visualizations created?

Hint: check out this Distill post on [Feature Visualization](#).

### 4.3.2

Filter size.

1. [M] How are your model's accuracy and computational efficiency affected when you decrease or increase its filter size?

When increasing the filter size, the model is able to capture larger spatial patterns in the input data. This can lead to improved accuracy by allowing the model to capture more complex features. However, this also increases the number of parameters in the model and the computational cost of the convolution operation, which can slow down training and inference times.

On the other hand, decreasing the filter size can increase the computational efficiency of the model by reducing the number of parameters and the computational cost of the convolution operation. However, this can also lead to a decrease in accuracy because the model may not be able to capture larger spatial patterns in the input data as effectively.

2. [E] How do you choose the ideal filter size?

- 1x1 kernel size is only used for dimensionality reduction that aims to reduce the number of channels. It captures the interaction of input channels in just one pixel of feature map. Therefore, 1x1 was eliminated as the features extracted will be finely grained and local that too with no information from the neighboring pixels.
- 2x2 and 4x4 are generally not preferred because odd-sized filters symmetrically divide the previous layer pixels around the output pixel. And if this symmetry is not present, there will be distortions across the layers which happens when using an even sized kernels, that is, 2x2 and 4x4. So, this is why we don't use 2x2 and 4x4 kernel sizes.

Therefore, 3x3 is the optimal choice to be followed by practitioners until now.

Answer taken from [this](#) article.

### 4.3.3

[M] Convolutional layers are also known as "locally connected." Explain what it means.

Convolutional layers in Convolutional Neural Networks (CNNs) are also known as "locally connected" layers because they operate on small, localized regions of the input data.

Specifically, a convolutional layer applies a set of learnable filters to local regions of the input data, typically with stride greater than 1, which allows for spatial downsampling of the feature maps. This means that each filter is only connected to a small, localized region of the input, rather than being fully connected to all of the input neurons.

This local connectivity allows the model to learn spatially invariant features, which means that the same feature can be detected regardless of its location in the input data. For example, a CNN trained to detect faces should be able to recognize a face regardless of its position or orientation in the input image.

By using locally connected filters, CNNs are able to achieve a high degree of parameter sharing, which can greatly reduce the number of parameters in the model and improve generalization performance. This is because each filter is applied to multiple local regions of the input, allowing it to learn features that are relevant to multiple parts of the input data.

Hence, convolutional layers are technically locally connected layers. To be precise, they are locally connected layers with shared weights. We run the same filter for all the (x,y) positions in the image. In other words, all the pixel positions "share" the same filter weights.

### 4.3.4

[M] When we use CNNs for text data, what would the number of channels be for the first conv layer?

Detailed explanation in [this](#) article.

When using Convolutional Neural Networks (CNNs) for text data, the number of channels for the first convolutional layer will typically be 1, since text data is typically represented as a 1D sequence of tokens.

In a CNN for text classification, the input to the network is typically a sequence of word embeddings or character embeddings, where each word or character is represented by a high-dimensional vector. This sequence of embeddings can be viewed as a 1D signal, where the embeddings correspond to the values along the signal. The first convolutional layer is then applied to this 1D signal, with each filter sliding over a local region of the sequence and producing a single scalar output. The number of channels in the output of the convolutional layer will depend on the number of filters used, but the number of channels in the input to the layer is typically 1.

### 4.3.5

[E] What is the role of zero padding?

Zero padding is a technique used in Convolutional Neural Networks (CNNs) to adjust the spatial dimensions of the input data and/or control the size of the output feature maps produced by convolutional layers.

In a convolutional layer, a filter/kernel slides over the input data to extract features. When the filter slides over the edges of the input data, the resulting output feature map is smaller than the input. This is because the filter does not have enough input data to slide over, resulting in the output feature map being "cropped" at the edges.

To address this issue, zero padding is often used to add extra rows and/or columns of zeros to the edges of the input data. This has the effect of "extending" the input data, so that the filter has more data to slide over at the edges. The amount of padding added can be controlled by adjusting the padding hyperparameter.

Zero padding has several benefits:

1. Maintaining spatial dimensions: By adding zeros to the edges of the input data, zero padding can help maintain the spatial dimensions of the input throughout the convolutional layers, which can be important for preserving the spatial structure of the input data.

2. Controlling the size of output feature maps: By adjusting the amount of zero padding, it is possible to control the size of the output feature maps produced by convolutional layers, which can be useful for controlling the model's complexity and memory requirements.
3. Reducing information loss at edges: Zero padding can help prevent information loss at the edges of the input data, by ensuring that the filter has enough input data to slide over at the edges.

#### 4.3.6

[E] Why do we need upsampling? How to do it?

Upsampling refers to any technique that, well, upsamples your image to a higher resolution.

Upsampling is a technique used in neural networks to increase the spatial resolution of the feature maps produced by convolutional layers. This can be useful in a variety of computer vision tasks, such as image segmentation and generative modeling, where high-resolution feature maps are required.

The need for upsampling arises because convolutional layers applied to input images typically result in down-sampled feature maps. This downsampling reduces the spatial resolution of the feature maps, which can lead to loss of information and accuracy in subsequent layers.

Upsampling can be performed using several different techniques, including:

1. Nearest neighbor interpolation: In this method, each pixel in the input feature map is replicated multiple times to produce a higher-resolution output. The value of each new pixel is set to be the same as the value of the nearest pixel in the input feature map.
2. Bilinear interpolation: In this method, each pixel in the input feature map is replaced by a weighted average of its neighboring pixels. The weights are based on the distances between the pixel being interpolated and its neighbors.
3. Transposed convolution: Also known as "deconvolution", this method involves learning a set of weights that are used to perform an inverse convolution operation. This has the effect of "undoing" the down-sampling that occurred in previous layers and producing a higher-resolution output.

#### 4.3.7

[M] What does a 1x1 convolutional layer do?

A 1x1 convolutional layer is a type of convolutional layer in a Convolutional Neural Network (CNN) that performs a 1x1 filter convolution operation on the input feature maps. This operation is similar to a standard convolution, but with a filter size of 1x1.

A 1x1 convolutional layer can be used for a variety of purposes in a CNN, including:

1. Dimensionality reduction: By reducing the number of feature maps output by a previous convolutional layer, a 1x1 convolutional layer can reduce the computational cost of subsequent layers and improve the model's speed and efficiency.
2. Non-linearity: By applying a non-linear activation function to the output of a 1x1 convolutional layer, the model can introduce non-linearity into the network and improve its expressive power.
3. Channel-wise feature combination: By combining feature maps across channels using 1x1 convolutions, a model can perform cross-channel feature learning and capture complex interactions between different feature maps.
4. Model compression: By using a 1x1 convolutional layer to reduce the number of feature maps in a model, the model's size and complexity can be reduced, making it more memory-efficient and easier to deploy on resource-constrained devices.

#### 4.3.8

Pooling.

1. [E] What happens when you use max-pooling instead of average pooling?

In a convolutional neural network (CNN), max-pooling and average pooling are two common types of pooling layers used for down-sampling the feature maps produced by convolutional layers.

Max-pooling operates by taking the maximum value of a group of pixels in a feature map, while average pooling calculates the average value of the group of pixels.

When max-pooling is used instead of average pooling, the resulting feature maps will emphasize the most salient features within each pooling region. This can be useful in tasks where the precise location of features is less important than their presence or absence, such as in object detection or image classification. By emphasizing the most salient features, max-pooling can improve the model's ability to detect and classify objects.

In contrast, average pooling tends to smooth out the feature maps by taking the average of the group of pixels. This can be useful in tasks where the precise location of features is important, such as in semantic segmentation or object localization. By smoothing out the feature maps, average pooling can help the model produce more precise segmentation masks or bounding boxes.

2. [E] When should we use one instead of the other?

Overall, the choice between max-pooling and average pooling will depend on the specific task and the requirements of the model. Max-pooling is often used in CNNs for image classification and object detection, while average pooling is more commonly used in CNNs for semantic segmentation and object localization.

3. [E] What happens when pooling is removed completely?

In a convolutional neural network (CNN), pooling layers are often used to reduce the spatial dimensionality of the feature maps produced by convolutional layers. The purpose of pooling is to downsample the feature maps and extract the most salient features while reducing the computational cost of the network.

When pooling is removed completely, the resulting network will have higher spatial resolution feature maps, but will also be more computationally expensive. This can be beneficial in tasks where the precise location of features is important, such as in semantic segmentation or object localization, as it allows the model to make more precise predictions about the location of objects in the image.

However, removing pooling can also make the model more prone to overfitting and reduce its ability to generalize to new data. This is because pooling helps to extract the most important features from the feature maps and reduce the impact of noise and minor variations in the input image.

4. [M] What happens if we replace a 2 x 2 max pool layer with a conv layer of stride 2?

If we replace a 2 x 2 max pooling layer with a convolutional layer of stride 2, the resulting down-sampling operation will be similar, but with some important differences.

A 2 x 2 max pooling layer with stride 2 downsamples the input feature map by taking the maximum value of a group of four adjacent pixels and moving the pooling window two pixels at a time in both the horizontal and vertical directions. This operation reduces the spatial dimensionality of the feature map by a factor of two, while retaining the most salient features.

On the other hand, a convolutional layer of stride 2 downsamples the input feature map by applying a convolution operation with a stride of 2, which involves sliding a filter window of size  $k \times k$  over the input feature map and taking a step of size 2 pixels in both the horizontal and vertical directions. This operation reduces the spatial dimensionality of the feature map by a factor of two, but the resulting feature map will be a linear combination of the input features, rather than a selection of the most salient features.

Replacing a max pooling layer with a convolutional layer of stride 2 can be useful in cases where the model needs to learn more complex, non-linear down-sampling operations that are better suited to the task at hand. However, this approach can also increase the computational cost of the network, as the convolution operation involves more parameters and computations than the max pooling operation.

Overall, the decision to replace a max pooling layer with a convolutional layer of stride 2 will depend on the specific task and the trade-offs between computational efficiency, performance, and the complexity of the down-sampling operation required by the model.

#### 4.3.9

[M] When we replace a normal convolutional layer with a depthwise separable convolutional layer, the number of parameters can go down. How does this happen? Give an example to illustrate this.

Answer taken from [this](#) article.

Our Example:

input shape = (32,32,3)

output shape = (32,32,28)

filter size = (3,3,3)

- Normal 2D conv parameters:  
 $(32,32) \times (3,3,3) \times (28) = 774,144$

- Depthwise Separable Conv parameters:

1. Filtering  $\rightarrow$  Split into single channels, so 3x3x1 filter is required in place of 3x3x3, and since there are three channels, so the total number of 3x3x1 filters required is 3, so,  
 $(32 \times 32) \times (3 \times 3 \times 1) \times (3) = 27,648$



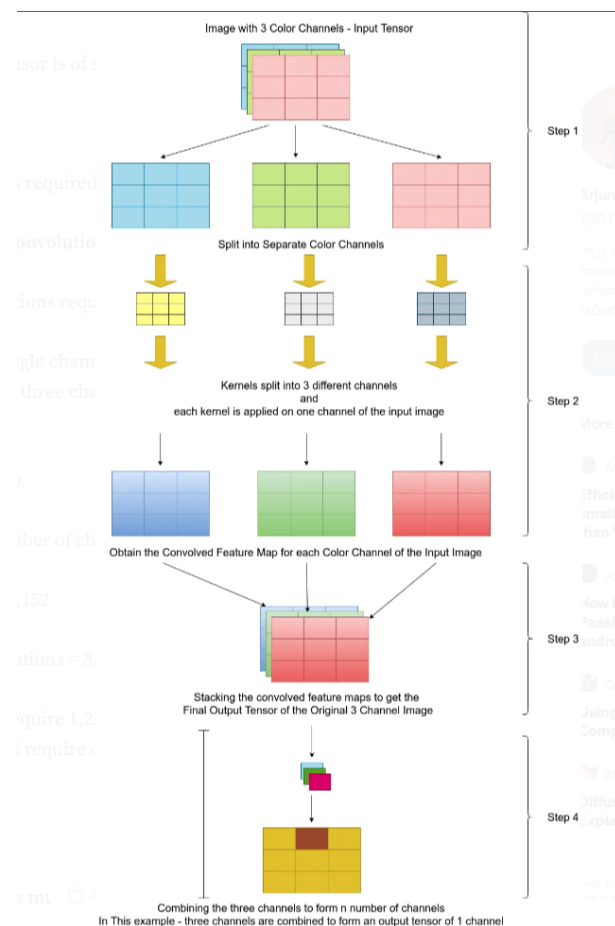


Figure 4.6: Visualization of a depthwise separable convolutional layer

- Combining  $\rightarrow$  Total number of output channels is 28 so,  
 $(32,32) \times (1 \times 1 \times 3) \times (28) = 86,016$

Total parameters = 113,664 which is  $\approx 7\%$  lesser than normal 2D conv.

#### 4.3.10

[M] Can you use a base model trained on ImageNet (image size 256 x 256) for an object classification task on images of size 320 x 360? How?

Yes, it is possible to use a base model trained on ImageNet (which typically uses input images of size 256x256) for an object classification task on images of size 320x360. However, some adjustments may be necessary to ensure optimal performance.

One approach is to resize the input images to the size expected by the base model, in this case 256x256, before passing them through the model. This can be done using various interpolation methods such as bilinear or bicubic interpolation. However, this approach may result in some loss of information due to the downscaling of the images.

Another approach is to fine-tune the pre-trained model on the new dataset using transfer learning. In this approach, the pre-trained model is used as a starting point and then retrained on the new dataset to learn the specific features and patterns relevant to the new task. This can lead to better performance compared to using the pre-trained model as-is.

When fine-tuning the pre-trained model, it is important to adjust the last layer of the model to match the number of classes in the new dataset. Additionally, the learning rate may need to be adjusted to ensure that the model does not forget the previously learned features from the pre-trained model while adapting to the new dataset.

#### 4.3.11

[H] How can a fully-connected layer be converted to a convolutional layer?

For in-depth answer read [this](#) article. Images taken from [this](#) article.

- Convolutions with kernels equal to the input size
- Convolution with 1x1 kernels

### Side Note: It is Possible to Replace Fully Connected Layers by Convolutional Layers

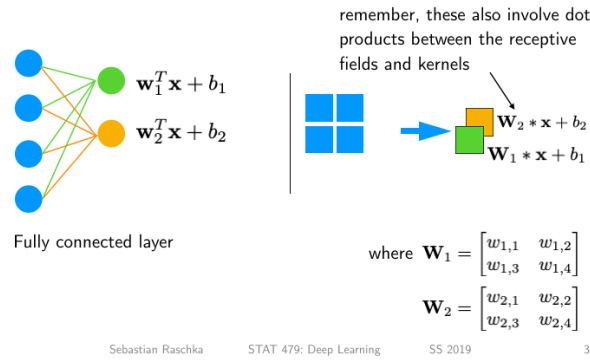


Figure 4.7: Convert fc to conv (Method1) - Convolutions with kernels equal to the input size

### Side Note: It is Possible to Replace Fully Connected Layers by Convolutional Layers

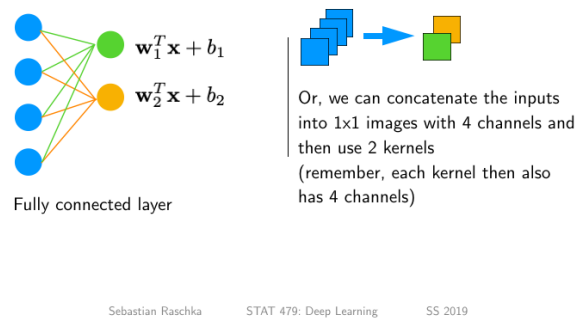


Figure 4.8: Convert fc to conv (Method2) - Convolution with 1x1 kernels

For example: Let us take a convolutional layer which outputs a volume of size  $7 \times 7 \times 512$  and this is followed by a FC layer with 4096 neurons i.e. the output of the FC layers is  $1 \times 4096$  for single image input. This can be equivalently expressed as a Conv layer with  $f=7$ ,  $p=0$ ,  $s=1$  and  $nc=4096$ . You can do the maths. We end up with a volume of shape  $1 \times 1 \times 4096$ .

#### 4.3.12

[H] Pros and cons of FFT-based convolution and Winograd-based convolution.

Hint: Read [Fast Algorithms for Convolutional Neural Networks](#) (Andrew Lavin and Scott Gray, 2015)

FFT-based convolution involves applying the convolution operation in the frequency domain using the Fast Fourier Transform (FFT). This method can be faster than traditional convolution algorithms for large filter sizes, and is highly parallelizable. However, it requires additional memory to store the intermediate results, and the input data needs to be padded to avoid circular convolution artifacts.

Winograd-based convolution involves transforming the convolution operation into a matrix multiplication that can be efficiently computed using the Winograd algorithm. This method can be faster than traditional convolution algorithms for small filter sizes, and has lower memory requirements than FFT-based convolution. However, it requires additional computations to perform the transformation, and may not be as accurate as traditional convolution for some tasks.

Here are the pros and cons of each method:

FFT-based convolution:

- Pros:
  1. Can be faster than traditional convolution for large filter sizes.
  2. Highly parallelizable.
  3. Can handle non-uniform filter sizes.
- Cons:
  1. Requires additional memory to store intermediate results.

2. Input data needs to be padded to avoid circular convolution artifacts.
3. Can be slower than traditional convolution for small filter sizes.

Winograd-based convolution:

- Pros:
  1. Can be faster than traditional convolution for small filter sizes.
  2. Has lower memory requirements than FFT-based convolution.
  3. Can be more efficient for certain types of convolutions, such as separable convolutions.
- Cons:
  1. Requires additional computations to perform the transformation.
  2. May not be as accurate as traditional convolution for some tasks.
  3. Can be slower than traditional convolution for large filter sizes.

## **4.4 Deep Learning Architectures and Applications: Reinforcement Learning**

## **4.5 Deep Learning Architectures and Applications: Other**

## **4.6 Training Neural Networks**