AIML - CS 337

Lecture 1: Probability Theory

8 August 2022

Lecturer: Abir De *Scribe: Group 1 and Group 2*

In the first lecture of the course, we began with a review of the basics of probability theory and some discussions on "what is ML?". We discussed how ML is used to recognize patterns in data, without explicitly being told which patterns there are.

We recalled the characteristics of Supervised Learning and Unsupervised Learning, which differ in whether the training data available is labelled or not. We then began with the review of probability theory.

1 Probability

1.1 Review

- **Sample Space (S)** : In probability theory, the sample space of an experiment or random trial is defined as the set of all possible outcomes of that experiment.

$$P(S) = 1,\ P(\emptyset) = 0$$

- **Probability Distribution (p)** : Probability Distribution is a Mathematical function which outputs the probabilities of occurrence of different possible outcomes of an experiment.

$$p : S \rightarrow [0, 1]$$
$$s.t. \sum_{x \in S} p(x) = 1$$

- **Random Variable (X)**: A random variable can be defined as a numerical description of the outcome of a statistical random experiment. It is a mapping from the set of outcomes in the sample space to numbers.

- **Event(E)** : Events can be defined as the set of outcomes of an experiment. An event can be just one outcome or it can be a combination of more than one outcome from an experiment. It can be defined as a subset of the experiment's sample space.

$$E \subseteq S$$

  For example, in the dice roll experiment, the sample space is S = {1,2,3,4,5,6} ; one possible outcome is "1"; while a possible event is "The dice rolled an odd number", E = {1,3,5}.
  We often describe the events using conditions and these events can be combined using logical operations like **or**,**and**.

- **Probability of an Event** : It tells us the likelihood of happening of the event. Mathematically it is the total weight assigned to all the elements in the event by a given probability distribution.

$$P(E) = \sum_{x \in S} p(x)$$
$$P(\overline{E}) = 1 - P(E),\ \overline{E} = S \setminus E$$
$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

  - **Union bound :** This is an inequality used extensively in probability theory. It states that the probability of union of some events is less than or equal to the sum of probabilities of each individual event.

$$P(E_1 \cup E_2) \leq P(E_1) + P(E_2)$$

  - If E_1, E_2, . . ., E_n are pairwise disjoint events, then

$$P(\bigcup_{i=1}^{n} E_i) = \sum_{i=1}^{n} P(E_i)$$

  - **Conditional Probability** It is the probability of an event happening given that another event has already occurred. For events E_1 and E_2 (s.t $P(E_2)>0$)

$$P(E_1|E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

  - **Bayes' Rule** This Rule can be derived from the definition of conditional probability. It is named after Thomas Bayes(1701-1761). For events A and B (s.t P(A),P(B)>0)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

1.2 Applications of probability

1.2.1 Probability in Medical Testing

A lab test has a probability 0.95 of detecting a disease when applied to a person suffering from said disease, and a probability 0.10 of giving a false positive when applied to a non-sufferer. If 0.5% of the population are sufferers, what is the probability of a test being positive?

Solution: Let

Pos be the event that result of the test is Positive
Neg be the event that result of the test is Negative

S be the event that person is a sufferer
NS be the event that person is not a sufferer.

Given that

$$P(Pos|S) = 0.95$$
$$P(Pos|NS) = 0.10$$
$$P(S) = 0.005$$

we need to find probability of a test being positive i.e $P(Pos)$

$$P(Pos) = P(Pos \cap S) + P(Pos \cap NS)$$
$$P(Pos) = P(Pos|S) * P(S) + P(Pos|NS) * P(NS)$$
$$P(Pos) = 0.95 * 0.005 + 0.10 * 0.995$$
$$P(Pos) = 0.10425$$

0.10425 is the probability of a test being positive

1.2.2 Probability in NLP

Part Of Speech (POS) tagging is a popular Natural Language Processing (NLP) Problem.
Here Input is a set of N words and the output is POS tag for each word.

Assuming each word is independently drawn from a fixed vocabulary, find the probability that a sentence of length m contains a 'noun' given that it contains a 'verb'.

Solution: Let p_k be the probability that a word is of POS type 'k' and let A_k be the event that sentence contains POS type 'k'.

we need to find $P(A_{noun}|A_{verb})$. we already know that

$$P(A_{noun}|A_{verb}) = \frac{P(A_{noun} \cap A_{verb})}{P(A_{verb})}$$
$$P(A_{noun} \cap A_{verb}) = 1 - P(\overline{A_{noun} \cap A_{verb}})$$
$$P(A_{noun} \cap A_{verb}) = 1 - P(\overline{A_{noun}} \cup \overline{A_{verb}})$$
$$P(A_{noun} \cap A_{verb}) = 1 - (P(\overline{A_{noun}}) + P(\overline{A_{verb}}) - P(\overline{A_{noun}} \cap \overline{A_{verb}}))$$
$$P(A_{noun} \cap A_{verb}) = 1 - P(\overline{A_{noun}}) - P(\overline{A_{verb}}) + P(\overline{A_{noun}} \cap \overline{A_{verb}})$$

we now represent $P(A_k)$ in terms of p_k and length of the sentence i.e m by subtracting the probability from 1 that no POS type 'k' word is present in the sentence.

$$P(\overline{A_k}) = (1 - p_k)^m$$
$$P(A_k) = 1 - (1 - p_k)^m$$

$P(\overline{A_{noun}} \cap \overline{A_{verb}})$ is nothing but the probability of both A_{noun} and A_{verb} events not happening

$$P(\overline{A_{noun}} \cap \overline{A_{verb}}) = (1 - (p_{noun} + p_{verb}))^m$$

Now we use this to calculate the required answer

$$P(A_{noun} \cap A_{verb}) = 1 - (1 - p_{noun})^m - (1 - p_{verb})^m + (1 - (p_{noun} + p_{verb}))^m$$
$$P(A_{verb}) = 1 - (1 - p_{verb})^m$$

Final Answer:

$$P(A_{noun}|A_{verb}) = \frac{1 - (1 - p_{noun})^m - (1 - p_{verb})^m + (1 - (p_{noun} + p_{verb}))^m}{1 - (1 - p_{verb})^m}$$

2 Marginals and Independence

- Joint Distribution: It can be simply defined as the probability of two events (corresponding to two random variables) happening together. It can be extended to > 2 RVs.

$$p_{XY}(x, y) = P(X = x \ \ and \ \ Y = y)$$

- Marginal Distribution: Probability distribution of the variables contained in the subset of a collection of random variables.

$$Pr(X = x) = \sum_{y \in S_2} Pr((X, Y) = (x, y)), x \in S_1$$

- Independent Random Variable

$$X \amalg Y \Leftrightarrow P(X = x, Y = y) = P(X = x)P(Y = y) \quad \forall x, y$$

  In terms of conditional probability: $P(X|Y) = P(X)$

- Conditionally Independent Random Variable
  X and Y are conditionally independent given Z :

$$p(x, y|z) = p(x|z)p(y|z) \quad \forall x, y, z \quad (p(z) > 0)$$

In machine learning, if given a pair of dependent random variables, we often try to find a random variable Z, so that they become independent conditional to Z.

3 Expectation

For a random variable X on $\mathbb{R}$, with a probability mass function P, the expectation is defined as $\mathbf{E}[X] = \sum_x P(X = x)$. Expectation has the following properties:

- **Linearity** : $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$
- **Linearity** : $\mathbf{E}[cX] = c\mathbf{E}[X]$
- **Expectation as the minimizer of squared error** : We try to find z such that $\mathbf{E}[(X - z)^2]$ is minimized. Equating the derivative of this with respect to z to 0, we get $z = \mathbf{E}[X]$
- **Expecation of product of independent random variables** : $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$. This can be shown as:

$$\mathbf{E}[XY] = \sum_{x,y} P(X = x, Y = y)xy = \sum_x P(X = x)x \sum_y P(Y = y)y = \mathbf{E}[X]\mathbf{E}[Y]$$

Refer to the Appendix for proof of these properties.

4 Variance

Variance of a random variable is a measure of the deviation of the random variable from it's mean. The **variance** of a random variable X with E[X] = μ is defined as:

$$Var[X] = E[(X - \mu)^2] = E[X^2] - \mu^2 \tag{1}$$

Properties:

- $Var[X + \beta] = Var[X]$
- $Var[\alpha X] = \alpha^2 Var[X]$
- If $X_1, \ldots, X_n$ are pairwise independent, then $Var[\sum_i X_i] = \sum_i Var[X_i]$
- If $X_1, \ldots, X_n$ are pairwise independent, each with variance σ^2, then $Var[\sum_i X_i/n] = \frac{\sigma^2}{n}$

Refer to the Appendix for proof of these properties.

4.1 Chebyshev's Inequality

If X is a random variable with mean μ and variance σ^2, then $\forall \alpha > 0$

$$Pr[|X - \mu| \geq \alpha] \leq \frac{\sigma^2}{\alpha^2} \tag{2}$$

4.2 Covariance

It is a measure of **joint variability** of two random variables. For random variables X and Y, **covariance** is defined as:

$$Cov[X, Y] = E[(X - E(X))(Y - E(Y))] = E[XY] - E[X]E[Y] \tag{3}$$

Properties:

- $Cov[X, X] = Var[X]$
- $Cov[X + Z, Y] = Cov[X, Y] + Cov[Z, Y]$
- $Cov[\sum_i X_i, Y] = \sum_i Cov[X_i, Y]$
- $Cov[X, Y] = 0$, if X and Y are independent

5 Sum of random variables

Consider two independent random variables X, Y. Then, their sum $Z = X + Y$ is a random variable with a probabilities mass function as the convolution of the probability mass functions of X and Y.

$$P(Z = z) = \sum_x P(X = x)P(Y = z - x)$$

6 Some Important Distributions

Notation: $X \sim D$ if the random variable X takes its values according to some distribution $D : S \to [0, 1]$.

Bernoulli Random Variable
Takes values from $S = 0, 1$. A single **parameter** $q \in [0, 1]$ fully specifies a Bernoulli random variable, where $Pr[X = 1] = q$ (and $Pr[X = 0] = 1 - q$).
It is denoted by $Bern(q)$.

If $X \sim Bern(q)$ then

- $E[X] = (1 - q) \times 0 + q \times 1 = q$
- $Var[X] = q - q^2 = q(1 - q)$

Binomial Random Variable
A **binomial random variable** models how often a particular outcome occurs in a fixed number of trials of an experiment whose outcome can be modeled as a Bernoulli random variable.
It is denoted by $B(n, q)$.

Formally, if $Y = \sum_{i=1}^{n} X_i$, where $X_i \sim Bern(q)$ are i.i.d., then $Y \sim B(n, q)$. If $Y \sim B(n, q)$ then

- $P(Y = k) = \binom{n}{k} q^k (1-q)^{n-k}$
- $E[Y] = nq$
- $Var[Y] = nq(1-q)$

7 Continuous distributions

A continous random variable can be defined as a random variable which can take an infinite number of values across a range.

For a continuous random variables X, on a domain S, we define

- *Probability Distribution Function*(PDF) as a function $f : S \to \mathbb{R}_0^+$ with the probability of X taking a value inside $D \subseteq S$ being $\int_D f(x)dx$.
- *Cumulative Distribution Function*(CDF) as a function $F : \mathbb{R} \to [0, 1]$

$$\begin{aligned} F(x) &= P(X \leq x) \\ &= \int_{-\infty}^{x} f(x)dx \end{aligned}$$

Note The key difference between continous and discreet distributions lies in the the concepts of mass and density.

When the sample space S of random variable is:

- Discrete: The distribution assigns weights/probabilities to individual points
- Continuous: The distribution assigns density to all points such that the integral over the whole range is 1

7.1 Joint distribution

For two random variables X, Y on $\mathbb{R}$, the joint distribution is a function $f : \mathbb{R}^2 \to \mathbb{R}_0^+$, defined such that for all $D \subseteq \mathbb{R}^2$, we have

$$P((X, Y) \in D) = \int_D f((x, y))dxdy$$

Marginals For a joint distribution f, the marginals f_x and f_y are the probabilities of the variables X and Y taking some particular value. Hence, $f_x(t) = \int_{-\infty}^{\infty} f(t, y)dy$ and $f_y(t) = \int_{-\infty}^{\infty} f(x, t)dt$. This integral is often difficult to compute in practice, so we make use of the following trick:

$$\begin{aligned} f_x(t) &= \int_{-\infty}^{\infty} P(X = t|Y = y) f_y(y) dy \\ &= \mathbf{E}_{y \sim f_y}[P(X = t|Y = y)] \end{aligned}$$

This can be evaluated easily using the law of large numbers if it is easy to sample from the distribution f_y

Independence The variables X and Y are said to be independent if their joint distribution can be decomposed into the product of their PDFs i.e. $f(X, Y) = f_x(X)f_y(Y)$.

Conditionals The conditional density $f_x(x|Y = y)$, is the probability density of X, given that Y takes the value y. It is equal to $\frac{f(x,y)}{f_y(y)}$

8 Recovering the PDF from data

Consider a PDF $f : \mathbb{R} \to \mathbb{R}_0^+$ for a random variable X, which is unknown to us. We have a generator which allows us to generate i.i.d. samples from this PDF countably many times. We have the task of estimating what the PDF is, given the i.i.d. samples $\{x_1, x_2, x_3, \cdots x_n\}$.

We will first find the moments of the distribution using the Law of Large Numbers.

$$\mathbf{E}_f[X^k] = \lim_{n \to \infty} \frac{x_1^k + x_2^k + \cdots + x_n^k}{n}$$

Now, we find the moment generating function $M(\omega)$ of the distribution.

$$\begin{aligned} M(\omega) &= \int_{-\infty}^{\infty} e^{\iota\omega x} f(x) dx \\ &= \int_{-\infty}^{\infty} \left(1 + \iota\omega x - \frac{\omega^2 x^2}{2!} - \iota\frac{\omega^3 x^3}{3!} \cdots\right) f(x) dx \\ &= 1 + \iota\omega\mathbf{E}[X] - \frac{\omega^2\mathbf{E}[X^2]}{2!} - \iota\frac{\omega^3\mathbf{E}[X^3]}{3!} \cdots \end{aligned}$$

Once we have evaluated the moment generating function, we use the Inverse Fourier Transform to recover the PDF.

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\iota\omega x} M(\omega) d\omega$$

9 Appendix

9.1 Properties of Expectation

1. $E[X+Y] = E[X] + E[Y]$
   Proof. Let S is the sample space

$$\begin{aligned} E[X+Y] &= \Sigma_{s\in S}(X(s)+Y(s))P(s) \\ &= \Sigma(s \in S)X(s)P(s) + \Sigma_{s\in S}Y(s)P(s) \\ &= E[X] + E[Y] \end{aligned}$$

□

2. $E[(X-c)^2] \geq E[(X-\mu)^2]$ where, $\mu = E[X]$ for any constant c and random variable X
   Proof. By linearity of expectation, we have,

$$\begin{aligned} E[(X-a)^2] &= E[X^2 + a^2 - 2aX] \\ &= E[X^2] + a^2 - 2aE[X] \\ &= (E[X]-a)^2 + E[X^2] - E[X]^2 \end{aligned}$$

   As we will see later that, $Var(X) = E[X^2] - E[X]^2 = E[(X-\mu)^2]$, where $\mu = E[X]$ therefore,

$$E[(X-a)^2] \geq E[(X-\mu)^2]$$

□

3. $E[cX] = cE[X]$ for any constant c and random variable X.
   Proof.

$$\begin{aligned} E[cX] &= \Sigma_{s\in S} cX(s)P(s) \\ &= c\Sigma_{s\in S}X(s)P(s) \\ &= cE[X] \end{aligned}$$

□

4. $E[XY] = E[X]E[Y]$, If X and Y are independent
   Proof.

$$\begin{aligned} E[XY] &= \Sigma_{x,y} xyP(X=x)P(Y=y) \\ &= \Sigma_{x,y}[xP(X=x)][yP(Y=y)] \\ &= \Sigma_x xP(X=x)\ \Sigma_y yP(Y=y) \\ &= E[X]E[Y] \end{aligned}$$

□

9.2 Properties of Variance

1. $Var[X + \beta] = Var[X]$

   Proof.

$$\begin{aligned} Var[X+\beta] &= E[(X+\beta)^2] - E[(X+\beta)]^2 \\ &= E[X^2 + \beta^2 + 2X\beta] - (E[x] + \beta)^2 \\ &= E[X^2] + 2\beta E[X] + \beta^2 - E[X]^2 - \beta^2 - 2\beta E[X] \\ &= E[X^2] - E[X]^2 \\ &= Var[X] \end{aligned}$$

   □

2. $Var[\alpha X] = \alpha^2 E[X]$

   Proof.

$$\begin{aligned} Var[\alpha X] &= E[(\alpha X)^2] - E[(\alpha X)]^2 \\ &= E[\alpha^2 X^2] - (\alpha E[x])^2 \\ &= \alpha^2 E[X^2] - \alpha^2 E[X]^2 \\ &= \alpha^2 (E[X^2] - E[X]^2) \\ &= \alpha^2 Var[X] \end{aligned}$$

   □

3. $Var[X + Y] = Var[X] + Var[Y] + 2(E[XY] - E[X]E[Y])$

   Proof.

$$\begin{aligned} Var[X+Y] &= E[(X+Y)^2] - E[(X+Y)]^2 \\ &= E[X^2 + Y^2 + 2XY] - (E[X] + E[Y])^2 \\ &= E[X^2] + E[Y^2] + 2E[XY] - E[X]^2 - E[Y]^2 - 2E[X]E[Y] \\ &= (E[X^2] - E[X]^2) + (E[Y^2] - E[Y]^2) + 2(E[XY] - E[X]E[Y]) \\ &= Var(X) + Var(Y) + 2(E[XY] - E[X]E[Y]) \end{aligned}$$

   If X and Y are independent, then

$$Var[X + Y] = Var[X] + Var[Y]$$

   □

4. If $\{X_i\}_{i=1}^n$ are pairwise independent of each other
   $Var[X_1 + X_2 + \cdots + X_n] = \sum_i Var[X_i]$

9.3 Properties of Covariance

1. $Cov[X, X] = Var[X]$

   Proof.

   $$\begin{aligned} Cov[X, X] &= E[(X - E[X])(X - E[X])] \\ &= E[(X - E[X])^2] \\ &= Var(X) \end{aligned}$$

   Hence, proved. □

2. $Cov[X + Z, Y] = Cov[X, Y] + Cov[Z, Y]$

   Proof.

   $$\begin{aligned} Cov[X + Z, Y] &= E[(X + Z)Y] - E[(X + Z)]E[Y] \\ &= E[XY + ZY] - E[X + Z]E[Y] \\ &= E[XY] + E[ZY] - E[X]E[Y] - E[Z]E[Y] \\ &= (E[XY] - E[X]E[Y]) + E[ZY] - E[Z]E[Y]) \\ &= Cov[X, Y] + Cov[Z, Y] \end{aligned}$$

   □

   $Cov[\Sigma_i X_i, Y] = \Sigma_i Cov[X_i, Y]$ is a trivial extension of this property

AIML - CS 337

Lecture 2: Overview of Linear Algebra for ML

August 8^{th}, 2022

Lecturer: Abir De *Scribe: Group 3, Group 4*

Linear Algebra is one of the key fundamental concepts required in Machine Learning. Uses of Linear Algebra include modelling a system as a linear transformation of input data, helping solve a system of linear equations, data compression using Principal Component Analysis, etc.

1 Vectors and their Properties

Vectors are the basic representation unit of a data-point , an ordered tuple of numbers. We start with few basic notations, properties, and functions that can be applied to vectors.

1.1 Dot product

Definition 1.1. The dot product of two equi-dimensional vectors **u , v** $\in \mathbb{R}^n$ is given by

$$\begin{aligned}\mathbf{u} \cdot \mathbf{v} &= \sum u_i v_i \\ &= \mathbf{u^T v}\end{aligned}$$

where x_i denotes the i^{th} component of vector x, and x^T represent the transpose of a vector. In the above definition of dot product, it has been assumed that the vectors are represented as columns.

1.2 Independence

A set of vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$ is said to be linearly independent if one cannot be represented by any linear combination of other vectors. More formally,

Definition 1.2. a set of vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$ is said to be linearly independent iff for scalars c_i's we have,

$$\sum_{i=1}^{n} c_i \mathbf{v_i} = 0 \implies c_i = 0 \qquad \forall i \in [1, n]$$

The above definition can also be seen that the solution of the equation

$$\mathbf{Ac} = \mathbf{0}$$

is only the trivial solution $c = 0$, where is the matrix $\mathbf{A} = [\mathbf{v_1}\ \mathbf{v_2}\ \ldots\ \mathbf{v_n}]$

1.3 Vector space

A set of vectors **V** is said to qualify as a vector space if it is closed under the operations of addition and scalar multiplication i.e. :

Definition 1.3. A set of vectors $\mathbf{V}$, is said to be a vector space , if for any two vectors $\mathbf{u}, \mathbf{v} \in \mathbf{V}$ we have

$$a * \mathbf{u} \in \mathbf{V}, \qquad a \in \mathbb{R} \tag{1}$$

$$a * \mathbf{u} + b * \mathbf{v} \in \mathbf{V}, \qquad a, b \in \mathbb{R} \tag{2}$$

Definition 1.4. If a subset $\mathcal{V}_S$ of any vector space $\mathcal{V}$ is itself a vector subspace then we say that it is a subspace of $\mathcal{V}$.

An example of a vector space is $\mathbb{R}^n \ \forall n > 0$. A subset of independent vectors of a vector space

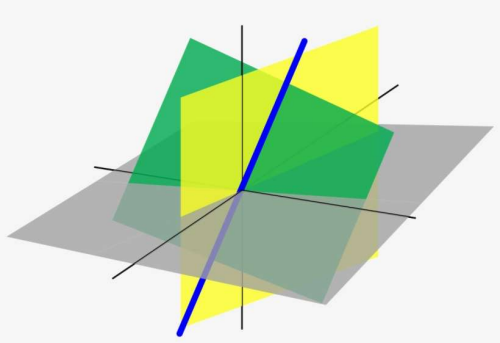


Figure 1: Examples of linear subspaces of $\mathbb{R}^3$

of highest cardinality is called a **basis** for the vector space. Every vector in the vector space can be represented as a linear combination of some or all vectors in a basis of the vector subspace. Note that the basis need not be unique for the vector space.

2 Matrices

A matrix is a rectangular array or a table of numbers, symbols , or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object. Matrices form a fundamental part of linear algebra which have geometric meaning associated to them while dealing with data in Machine Learning.
An N $\times$ M matrix has N rows and M columns. Following is an example of a 2x2 matrix A.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Any point in $\mathbb{R}^n$ can be seen as an n dimensional vector, wherein multiplying a matrix with a vector then has the geometric significance of **applying a linear operation** on the vector.

2.1 Matrix Multiplication

Definition 2.1. If V is a matrix of size n $\times$ m and W is an m $m \times p$ matrix, then the matrix product U = VW is a size of $n \times p$ and can be computed as follows:

$$V = \begin{bmatrix} v_{1,1} & \dots & v_{1,m} \\ \vdots & \vdots & \vdots \\ v_{n,1} & \dots & v_{n,m} \end{bmatrix}, W = \begin{bmatrix} w_{1,1} & \dots & w_{1,p} \\ \vdots & \vdots & \vdots \\ w_{m,1} & \dots & w_{m,p} \end{bmatrix}, U = \begin{bmatrix} u_{1,1} & \dots & u_{1,p} \\ \vdots & \vdots & \vdots \\ u_{n,1} & \dots & u_{n,p} \end{bmatrix}$$

where $u_{i,j} = \sum_{k=1}^{m} v_{i,k} w_{k,j}$

2.2 Determinants

Determinant is a scalar value that is a function of the entries of a square matrix. In the case of a 2x2 matrix the determinant can be defined as

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Higher order determinants recursively follow.

2.3 Linear Systems

A linear system or a system of linear equation is a collection of one or more linear equations involving the same variables. A general system of linear equation with n unknows and coefficients can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

We can represent a system of linear equations using a coefficient matrix A, a vector of variables $\mathbf{x}$ and constants $\mathbf{b}$ as : $\mathbf{Ax} = \mathbf{b}$. The problem then reduces to finding an x for a given A and b.

3 Solving System of Equations

A system of linear equations in n variables can have the following solutions :

- Exactly one solution
- No solution
- An infinite number of solutions

3.1 Matrix Inversion

Given a linear system of equation represented by Ax = b, we can find a solution x = $A^{-1}b$, where A^{-1} is called the inverse of the matrix.

- If **A** is invertible we can find unique solution to our systems of equation.
- **Gauss-Jordan :** We perform elimination step on the augmented matrix [A I] to give the augmented matrix [I A^{-1}]. Gauss-Jordan elimination gives a Reduced Row Echelon Form. An **augmented matrix** $A|b$ is basically

$$\left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & b_m \end{array}\right]$$

3.2 Gaussian Elimination

Given a system of linear equations denoted by

$$\mathbf{A}x = \mathbf{b}$$

we can solve the equations using Gauss Jordan Elimination. We note the elementary row operations are given as swapping two rows in the matrix , multiplying rows by a constant factor , and adding two rows. After performing Gaussian Elimination on the coefficient matrix we get a Row Echelon Form.

3.3 LU decomposition and application

We can also solve for the system of linear equations $Ax = b$, by first converting **A** into a product of two matrices, such that one is completely a lower triangular matrix L and other is an upper triangular matrix U,

$$A = LU$$

Matrix L and U , can be found by the help of Gaussian Elimination applied on the matrix A. Once we have L and U , we can then solve for x by first solving the equation ,

$$Ly = b$$

for y , and then the equation

$$Ux = y$$

Algorithm 1: Gaussian Elimination

Data: $\mathbf{A}, \mathbf{b}$
Result: $\mathbf{X} = [X_1 X_2 \dots X_n]$ such that $\mathbf{AX} = \mathbf{b}$
$A\_b \leftarrow A|b$;
/* Indexing assumed to start from 1 */
$n \leftarrow A.shape[1]$;
/* Forward Elimination */
for *i in 1 to n-1* **do**
  /* Identify pivot */
  if *$A\_b[i][i]$ is 0* **then**
    /* Note: There is a variation of the algorithm where you move on to the next column if encountering zero */
    $X \leftarrow$ "No solution exists";
  else
    for *j in i+1 to n* **do**
      for *k in 1 to n+1* **do**
        $A\_b[j][k] \leftarrow A\_b[j][k] - A\_b[j][i]/A\_b[i][i] * A\_b[i][k]$;
      end
    end
  end
end
/* Back Substitution */
$X_n \leftarrow A\_b[n][n+1]/A\_b[n][n]$;
for *i in n-1 to 1* **do**
  $X_i \leftarrow A\_b[i][n+1]$;
  for *j in i+1 to n* **do**
    $X_i \leftarrow X_i - A\_b[i][j] * X_j$;
  end
  $X_i \leftarrow X_i/A\_b[i][i]$;
end

4 Column space, Null space and Invertibility

4.1 Column space :

Column space of a matrix A, or $C(A)$, is the space consisting of all possible linear combinations of the columns of A. For any real vector x, Ax will lie in the column space of A.

- If there exists a solution for the equation $b = Ax$ for a real matrix A and real vectors b and x, then b lies in the $C(A)$ and vice versa.

4.2 Null space :

Null space of a matrix A, or $N(A)$, is the space spanned by all the solutions x, of $Ax = 0$.

The solutions form a vector space because :

- If $Ax = 0$ then $A(cx) = c(Ax) = 0$
- If $Ax = 0$ and $Ay = 0$ then $A(x + y) = 0$

Having a vector x other than the null vector, such that $Ax = 0$, implies that the columns of matrix A are dependent.

4.3 Rank of a matrix :

Rank of a matrix A is defined as the size of the maximal set of independent columns of the matrix A, and those columns form a basis for $C(A)$.

- If A^{-1} exists, the only solution to $Ax = b$ is $x = A^{-1}b$
- In addition to the first statement, A is singular iff there are solutions other than $x = 0$ to $Ax = 0$, or in other words, iff it has a non-singular null-space $N(A)$.

A matrix A is called a full column rank matrix if all the columns in A are independent.

4.4 Invertibility :

A square matrix A is invertible if there exists a square matrix B such that $AB = BA = I_n$, and B is known as inverse of A, also denoted by A^{-1}.

- A square matrix is invertible iff it is a full column rank matrix.

4.5 Computing the Inverse - Gauss Jordan Elimination :

The Gauss-Jordan elimination method addresses the problem of solving several linear systems $Ax_i = b_i (1 \leq i \leq N)$ at once, such that each linear system has the same coefficient matrix A but a different right hand side b_i.

We have seen how elimination matrices are used to convert a coefficient matrix A into some upper triangular matrix U,

$$U = E_{32}(E_{31}(E_{21}A)) = (E_{32}E_{31}E_{21})A$$

Now, further apply elimination steps until U gets transformed into identity matrix:

$$I = E_{13}(E_{12}(E_{23}(E_{32}(E_{31}(E_{21}A))))) = (E_{13}E_{12}E_{23}E_{32}E_{31}E_{21})A$$

By definition, $X = (E_{13}E_{12}E_{23}E_{32}E_{31}E_{21})$ must be A^{-1}

Note that the above method works only if A is invertible.

So, Gauss-Jordan is basically elimination steps over the augmented matrix $[AI]$ (representing the equation $AX = I$) to give the augmented matrix $[IA^{-1}]$ (representing the equation $IX = A^{-1}$)

4.6 Dealing with Rectangular matrices :

What if A is not a square matrix but rather a rectangular matrix of size mxn, such that $m \neq n$. Does there exist a notion of A^{-1}? The answer depends on the rank of A.

- If A is full row rank and $n > m$, then AA^T is a full rank mxm matrix $\iff (AA^T)^{-1}$ exists with $A^T(AA^{T-1})$ leading to I and is therefore called the right inverse of A. When the right inverse of A is multiplied on its left, we get the projection matrix $A^T(AA^T)^{-1}A$, which projects matrices onto the row space of A.
- If A is full column rank and $m > n$, then A^TA is full rank nxn matrix $\iff (A^TA)^{-1}$ exists with $(A^TA)^{-1}A^T$ leading to I and is therefore called the left inverse of A. When the left inverse of A is multiplied on its right, we get the projection matrix $A(A^TA)^{-1}A^T$, which projects matrices onto the column space of A.

If A is a full column rank matrix (that is, its columns are independent), A^TA is invertible.

We will show that the null space of A^TA is 0, which implies that the square matrix A^TA is full column (as as well as row) rank is invertible. That is if $A^TAx = 0$, then $x = 0$. Note that if $A^TAx = 0$, then $x^TA^TAx = ||Ax|| = 0$ which implies that $Ax = 0$. Since the columns of A are linearly independent, its null space is 0 and therefore, $x = 0$.

AIML - CS 337

Lecture 3: Linear Algebra Review, Classification Task

11 August 2022

Lecturer: Abir De *Scribe: Advait, Ganesh, Suman, Vaibhav*

This lecture starts with a review of Linear Algebra and some of its uses in Machine Learning. Later on, the general Machine Learning problem of Classification is discussed.

1 Linear Algebra (continued)

Apart from the general usage of Linear Algebra in Machine Learning as a tool to represent features (as vectors/matrices) and weights (as matrices), there exist many tasks requiring deeper insights.

1.1 Time Dependence in Networks

Consider a classification task where we get sequences as input. In a normal classification task, features are of the form **(*feature*, *label*)** while for a sequence, there is a time component and each interval has a feature vector - $x_t, x_{t+1}, \ldots$ with possible dependence across features. One possible relationship is a linear dependence of the features at time $t+1$ on the features at time t.

Suppose we have a network where edges represent influence of nodes over each other. This influence travels over time and the information/data at one node affects its neighbors and other nodes in the future. Social networks (on Facebook/Instagram) can be represented in this form and the information in the nodes can possibly be sentiment of people towards politicians or extent of spread of misinformation.

Consider a graph similar to the one shown to the right. Suppose that nodes i and j share an influence over each other with strength w_{ij} (not necessarily same as w_{ji}). For instance, the influence of 1 over 6 is twice that of 2 over 6 in the example graph (which is undirected, hence symmetric). These influences can be represented in a matrix $\mathbf{W}$ of size $N \times N$ where N is the number of nodes in the graph. The elements on the diagonal w_{ii} will represent remembrance of the previous state at some node.

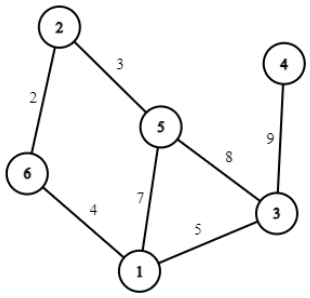


Figure 1: Example Graph

Suppose the state at some time t of the i^{th} node is $\mathbf{x_t^i}$. These state vectors can be stacked row-wise in a matrix $\mathbf{X_t}$ which evolves with time. Assume that the state changes as following -

$$\mathbf{x_{t+1}^i} = \sum_{j=1}^{N} w_{ij} \mathbf{x_t^j}$$

The evolution of states can then be represented as - $\mathbf{X_{t+1}} = \mathbf{W X_t}$

1.2 Stochastic Matrices

Under some special conditions, the state denoted by $\mathbf{X_t}$ will converge. Such is the case of stochastic matrices.

Definition 1.1. Matrix $\mathbf{W}$ is **stochastic** if all entries are non-negative and sum of all entries in a column equals 1. $\mathbf{W}$ is further **positive** if all entries are positive numbers.

Lemma 1.2. *Properties of eigenvalues of a stochastic matrix* $\mathbf{W}$ *are given below -*

1. 1 *is an eigenvalue of* $\mathbf{W}$.
2. *If* λ *is an eigenvalue (real/complex) of* $\mathbf{W}$, $|\lambda| \leq 1$.

Some more important results related to stochastic matrices -

Definition 1.3. For a stochastic matrix $\mathbf{W}$, a *steady state* is an eigenvector with eigenvalue 1 whose entries are positive and add to 1.

Theorem 1.4 (Perron-Frobenius Theorem)**.** *For a positive stochastic matrix* $\mathbf{W}$*, there exists a unique steady state vector* v *which spans the* 1*-eigenspace. Also, for any vector* v_0 *with entries adding to* s*, iterates* $v_1 = Av_0, v_2 = Av_1, \ldots, v_{t+1} = Av_t, \ldots$ *tend to* sv *as* t *gets large.*

More discussion can be found at [1].

1.3 Average Consensus

In this case, the weight matrix is such that all neighbors get equal weights and non-neighbors don't have any contribution. Suppose the graph is G and its edge set is E.

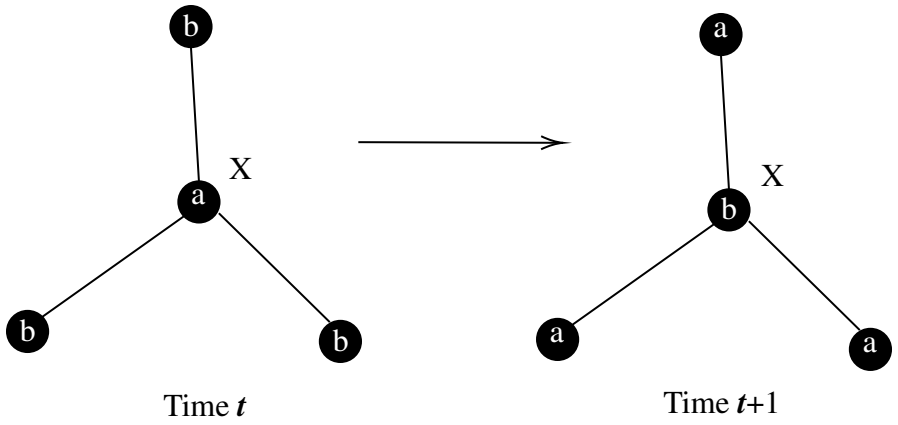
$$\mathbf{x_{t+1}^i} = \frac{\sum_{j \in N(i)} \mathbf{x_t^j}}{|N(i)|}$$

Upon converting the equation to matrix form, we get the weight matrix $\mathbf{W}$ such that $w_{ij} = \frac{1}{d_i}$ (degree of vertex i) if $(i, j) \in E(G)$, else $w_{ij} = 0$. Here, each sum rows to 1 and this is a right stochastic matrix.

In this case, $\mathbf{1}$ is an eigenvector of $\mathbf{W}$ with eigenvalue 1 because this multiplication just amounts to adding up every row which we already have as 1.

1.4 Non-convergence of state

We can create instances of graphs where convergence is not reached. Consider the below graph with a central node $\mathbf{X}$ connected to some nodes (a star type network) . Suppose $\mathbf{X}$ has a state of $\mathbf{a}$ while all its neighbors have a value of $\mathbf{b}$. If the node state relation is a simple average over neighbors (average consensus), these values will keep oscillating - $\mathbf{a}$ on $\mathbf{X}$, $\mathbf{b}$ on neighbors to $\mathbf{a}$ on neighbors, $\mathbf{b}$ on $\mathbf{X}$. Convergence will never be reached in this case.



2 Introduction to the Classification Task

Machine Learning deals with multiple tasks which can be broadly classified into Classification, Regression, Clustering etc. Of these, classification is a very important task & is specified below -

Task 2.1. *Given K items, as features of the form $X_1, X_2, \ldots, X_K$, classify them into one of N possible labels $1, 2, \ldots, N$.*

Solution Approach - To solve the above task, we will need a function $\mathbf{H}$ such that $\mathbf{H}(X_i) = Y_i$ where Y_i is the expected label for the item with features X_i. The labels are defined by users, i.e., we don't know the definition or the meaning of a label. Suppose that no information about the relation between labels and features is provided to the function. A simple permutation of labels on the user side will lead to mistakes in the labels resulting from the function $\mathbf{H}$ even though the input didn't change. Hence, some information about the relation between labels and features is required. The easiest way to get this information is through examples, which is the first step in the solution method.

1. Ask the user for information about labels in the form of examples i.e. $\{X_i, y_i\}_K$, formally called the **Training Set**.

The user will provide us some already classified data (examples) and once we have the example pairs, the next step is to learn from them to be able to predict the labels for **unseen examples**.

2. Create an algorithm **ALG** which *learns* from provided examples and is able to do *inference* i.e. predict labels for arbitrary input.

Once the algorithm **ALG** is ready, we need to verify if it indeed works well for unseen inputs. For this, we would need some more examples as pairs to test results.

3. Verify **ALG** on examples not included in the Training Set (checking the correctness of **ALG**). This set of examples is formally called the **Validation Set**. We don't use this set to edit the algorithm, it just helps accept/discard an algorithm.

Asking the user for more examples at this stage can be cumbersome because label collection can be expensive with respect to time and money. Hence, all examples are asked for at the start of the process and the full dataset is divided into the 2 portions - **Training Set** & **Validation Set**. Our goal during training is *generalizability* - the algorithm should be as suitable for the Validation set as for the Training set.

Note - We blindly trust the data provided to us by the user i.e., we assume that the data is correct and user doesn't do any mistakes.

References

[1] Dan Margalit and Joseph Rabinoff. *Interactive Linear Algebra*. 2019.

AIML - CS 337

Lecture 4: Introduction to Loss Functions

18/08/22

Lecturer: Abir De *Scribe:*

In this lecture we develop methods that evaluate how well machine learning models learn our data set. This is accomplished by means of *loss* functions. We see what entails in the design of such loss functions.

1 Motivation for Loss Functions

We study about loss function in a simple setting of classification which is defined as follows:

1.1 Review of Classification Task

The general classification task can be described as follows :

Classification Task : Given a dataset $\{(x_i, y_i)\}$ where $x_i \in \mathbb{R}^d$ and $y \in \mathcal{Y}$, where d is dimension of the input features and $\mathcal{Y}$ is the target variables, our goal is to devise an algorithm that selects a hypothetsis $h \in H : \mathbb{R}^d \to \mathcal{Y}$, So that given some unseen test case x_j and the corresponding label y_j, $h(x_j) = y_j$. For example, in `MNIST` image classification task, each $x_i \in \mathbb{R}^{28\times 28}$ and $\mathcal{Y} = \{0, 1, \cdots 9\}$ are the class labels.

However, in order to devise a good algorithm that selects the best hypothesis $h^* \in H$, the developer needs to know a more concrete metric which is used by the user to evaluate the algorithm on the test set. This metric which mathematically quantifies how well the algorithm models the data in any dataset is called a **Loss Function**.

1.2 Loss Minimization Task

Now that we are aware of the metric which is used by the user to evaluate the algorithm H we try to model the classification task as a loss minimization task instead. The general classification task can thus be modelled by the following loss minimization task :

$$h^* = \arg\min_{h\in H} \sum_{j=1}^{M} \mathbb{1}(h(x_j) \neq y_j)$$

Here x_j is an image in the test dataset, M is the number of images in the test dataset and $\mathbb{1}(\cdot)$ is the indicator function which is equal to 0 if its argument is false and equals 1 if the argument holds

true. If we assume that (a) we could enumerate all the possible functions $H : \mathbb{R}^d \rightarrow Y$ and (b) the function that minimizes the value over training set also minimizes it over the test set we could obtain h^* as follows :

$$h^* = \arg\min_{h \in H} \sum_{i=1}^{D} \mathbb{1}(h(x_i) \neq y_i)$$

where x_i are the images in the training dataset, y_i are the corresponding labels and D is the number of images in the training dataset. However, both our assumptions are clearly unreasonable. Most importantly it is not possible to enumerate all possible functions H particularly when H is an infinite set which is often the case. Therefore, we look for relaxations in the loss function in order to make the loss minimization task solvable. Some suitable relaxations for the loss function have been discussed in the next section.

2 Relaxation of Loss Function

In this section, we will try to look at the art of designing loss functions. We will look at, how we can arrive at a mathematically appealing loss function that models the classification problem in steps.
In all the subsections below we will consider the classification problem where $\forall x_i \in X$, there exists a label $y_i \in \{-1, 1\}$, over the training set.

2.1 Constant Hypothesis

While developing the loss function, one student suggested to use constant hypothesis. In other words, $H(x_i) = c$, where c is a uniformly generated random number in the interval $[-1, 1]$.
The loss function optimization problem can then be written as,

$$c^* = \arg\min_{c} \sum_{i=1}^{M} \mathbb{1}(c \neq y_i)$$

We do know that the point probability of a continuous distribution is 0 whence $P(c = 1)$ and $P(c = -1)$ are both zero. Hence, the loss function value is always M in this case. We cannot do better if we stick to this model with uniform distribution.
What if we choose the constant c among the values $\{-1, 1\}$. Let us denote n_+ to be the number of points in the training data set having labels as +1, and similarly denote n_- to be the number of points in the training data set having labels as -1. The optimization problem is same as above, but constrained to the fact that $c \in \{-1, 1\}$. It can be seen easily that if we take, $c = \max(n_+, n_-)$, the loss say L is $\min(n_+, n_-)$. This is the minimum that we can get. This method is Majority Mode, since we took the hypothesis to be the mode in the training set data.

2.2 Linear Hypothesis with Indicator Cost

Lets see if we can do better by increasing the complexity of our hypothesis class H. We suppose, $h(x_i) = w^T x_i + b$, where w is the vector of parameters of the same dimensions as x_i and b is the bias parameter. Hence, we have the following task in hand,

$$\{w^*, b^*\} = \arg\min_{w,b} \sum_{i=1}^{M} \mathbb{I}(w^T x_i + b \neq y_i)$$

For brevity, let L_1 denote the minimum loss achieved by considering a constant hypothesis and L_2 denote the minimum loss achieved by considering a linear hypothesis, then it is guaranteed that $L_2 \leq L_1$. The reason is obvious as we are trying to search a larger space to fit the data. Hence, our Linear model is reducible to constant hypothesis model by taking $(w, b) = (\mathbf{0}, c)$.
We can establish a generalized statement here. As model complexity increases, performance on the data used to build the model (training data) improves. However, performance on an independent set (validation data) may improve up to a point, then starts to get worse. This is called **overfitting**. Nevertheless, we have definitely done better as compared to constant hypothesis.

2.3 Linear Hypothesis with Absolute Difference Cost

Another Loss function was suggested which takes the cost as the absolute value of the difference between hypothesis and the label value (Assuming that the labels are mapped to some subset of integers). In this case the optimization problem becomes,

$$\{w^*, b^*\} = \arg\min_{w,b} \sum_{i=1}^{M} |w^T x_i + b - y_i|$$

But even this is not a good choice. $w^T x_i + b$ takes values in $\mathbb{R}$ but $y_i \in \{-1, 1\}$. A good loss function should act on predictions that are on the same scale as that of the ground truth targets which we allude to next.

2.4 Linear Hypothesis with Sign and Indicator Cost

Instead of looking at the value of $w^T x_i + b$, what if we look at it's sign. Hence, if $w^T x_i + b > 0$, then the estimated label should be 1 and vice-versa (boundary condition can be included within any one of them). This gives us the following optimization problem,

$$\{w^*, b^*\} = \arg\min_{w,b} \sum_{i=1}^{M} \mathbb{I}(\text{sgn}(w^T x_i + b) \neq y_i)$$

where $\text{sgn}(.)$ denotes the signum function.

2.5 Linear Hypothesis with Sigmoid Mapping

While the above loss function is good, it is not conducive to design efficient search algorithms that find h^* and is difficult to optimize. Nonetheless, the above loss function can still be used to test the performance of our trained models.

To make optimization convenient, we map $w^T x_i + b$ that takes values in $\mathbb{R}$ to the interval $[-1, 1]$. To do this, we can use the sigmoid activation function,

$$f(x_i) = \frac{1}{1 + e^{-(w^T x_i + b)}}$$

There's one downside to this, the sigmoid function doesn't map to $[0, 1]$, instead it does to $(0, 1)$. Hence, using the below optimization problem wouldn't make sense,

$$\{w^*, b^*\} = \underset{w,b}{arg\ min} \sum_{i=1}^{M} \mathbb{I}(f(x_i) \neq \frac{y_i + 1}{2})$$

One may be tempted to use the absolute difference squared cost, $|f(x_i) - \frac{y_i+1}{2}|^2$, which makes sense. The issue which we may incur due to this is a non-convex loss function of parameters. It may be difficult to converge to global minima in such cases.

There's a probabilistic approach to loss functions as well, which will be discussed in the upcoming lectures, but we will just state the result here,

$$\{w^*, b^*\} = \underset{w,b}{\arg\min} \sum_{i=1}^{M} \left[-\left(\frac{y_i + 1}{2}\right) \log(f(x_i)) - \left(1 - \frac{y_i + 1}{2}\right) \log(1 - f(x_i)) \right]$$

We can also define the loss in the following manner. We will incur a loss if $f(x_i) > 0.5$ and $y_i = -1$ OR $f(x_i) \leq 0.5$ and $y_i = 1$. We can hence write the following optimization problem,

$$\{w^*, b^*\} = \underset{w,b}{\arg\min} \sum_{i=1}^{M} \max\left(0, \left(\frac{1}{2} - f(x_i)\right) y_i\right)$$

This kind of loss is inspired from the ReLU function which is defined as

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{1}$$

AIML - CS 337

Lecture 5: Classification and Ranking Problem

22nd August 2022

Lecturer: Abir De *Scribe: Students and TA Team*

In this lecture, we begin by trying to make our classification model more robust to noise, after which we move on to modelling ranking problems

1 Introduction

Given data in the form of (x_i, y_i) (i.e. measurement features and label pairs) where $x_i \in \mathbb{R}^d$ and $y \in \{-1, 1\}$ for all $i \in \{1, 2, \ldots, D\}$ we want to construct an estimator $H : \mathbb{R}^d \to \{-1, 1\}$ that makes predictions as $y_i^{\text{pred}} = H(x_i)$. The way to go is to look at a particular class of functions for H and obtain the best parameters in that family of estimators. But before that, choosing that family is also important. We discuss the case when H is linear in x_i (in a sense).

Practically, we would divide the entire dataset into a training set and a validation set and it is to be noted that the following discussion is done only with respect to this training subset of the data. But for simplicity we still denote the set as the same $\{1, 2, \ldots, D\}$ since it doesn't change the discussion.

2 Linear estimators

Say first we want to find an optimal H from the family of constant functions $H(x_i) = c$. The c that is most optimal is the one that minimizes the particular loss function:

$$L = \sum_{i=1}^{D} \mathbb{I}\left(H(x_i) \neq y_i\right) \qquad \text{where } \begin{cases} \mathbb{I}(\text{true})=1 \\ \mathbb{I}(\text{false})=0 \end{cases} \tag{1}$$

c_{optimal} is ± 1 according to which is bigger: the number of $+1$ labels (η_+) or the number of -1 labels (η_-) so that it matches the most values from y_i. Here L_{min} (minimum loss) will be $\min(\eta_+, \eta_-)$.

Coming to linear estimators, suppose instead that we want $H(x_i) = \text{sgn}(\omega^\top x_i + b)$ (assuming, for now, that we don't encounter $\text{sgn}(0)$. sgn is signum function). This function is representative of assigning a score for each x_i (by a linear operation $\omega^\top x_i + b$) and classifying x_i according as if this score is positive or negative. We can try to minimise the same loss as before:

$$L = \sum_{i=1}^{D} \mathbb{I}\left(H(x_i) \neq y_i\right) \tag{2}$$

but the objective function to minimise here is not continuous. We can try to come up with a different, but continuous, loss function that is representative of the same needs.

Say true label (i.e. y_i) is $+1$. Then a desirable x_i will have $\omega^\top x_i + b > 0$ (positive score). Instead of assigning a fixed loss of 1 for the undesirable case of $\omega^\top x_i + b < 0$ (negative score), which is making the loss function discontinuous, a continuous loss metric can be $l(x_i) = \max(0, -(\omega^\top x_i + b))$. Similarly if label was -1 we would desire a negative score and a continuous loss for that case may be as $l(x_i) = \max(0, \omega^\top x_i + b)$. All in all our new loss function to minimise is:

$$L = \sum_{i=1}^{D} \max\left(0, -y_i(\omega^\top x_i + b)\right) \tag{3}$$

From now we will shorten $\max(0, \cdot)$ to $(\cdot)_+$. This function is sometimes referred to as 'ReLU' (Rectified Linear Unit).

Note that the loss term is still zero when, for example, $\text{signum}(\omega^\top x_i + b) = y_i$ which is in line with $\mathbb{I}(H(x_i) \neq y_i)$. With alternate loss metrics to minimise, we are still rewarding the same functions but are just penalising them differently. These 'alternate' loss functions are called surrogate loss functions when they are used instead of $\mathbb{I}(H(x_i) \neq y_i)$

3 Accounting for noise in measurements

So far, the classification method for our predictions say that:

$$y_i^{\text{pred}} = \begin{cases} +1 & \omega^\top x_i + b > 0 \\ -1 & \omega^\top x_i + b < 0 \end{cases} \tag{4}$$

But this way our predicted values can change from 1 to -1 if a small amount of noise in x_i causes $\omega^\top x_i + b$ to go from positive to negative. A common way to address this apparent discontinuity in our classification is to instead do this:

$$y_i^{\text{pred}} = \begin{cases} +1 & \omega^\top x_i + b > 1 \\ -1 & \omega^\top x_i + b < -1 \end{cases} \tag{5}$$

This is 'making sure' that the score $\omega^\top x_i + b$ is 'positive enough' before saying $y_i^{\text{pred}} = +1$, for example. We could have replaced $1, -1$ with some $a, -a$ as well but it is an equivalent problem since we are finding optimal ω, b and so will just divide by a. Again we are assuming, for now, that $\omega^\top x_i + b$ does not lie between 1 and -1. If it is, it's only an issue for the actual classification since our continuous loss functions do not 'break' for any x_i this way.

The condition assumed earlier, that $\omega^\top x_i + b$ does not lie between 1 and -1 can occur only when the convex hull determined by all the points with $y = 1$ and the convex hull determined by the points with $y = -1$ do not intersect (A convex hull of a set of points is defined as the smallest area convex polygon that encloses all the points). In this case, an ML model is not really needed since a deterministic construction of the convex hull would give well enough results. It is when there are ambiguous cases that ML truly shines.

Anyway, we will now accordingly need to modify our Loss function from before. We use the same reasoning to replace the constant loss of 1 in the undesirable case to a continuous extension of the desirable case. Say the score (z) is $z = \omega^\top x_i + b$. If $y_i = +1$ then loss is modelled as $(0, 1 - z)_+$ and if $y_i = -1$ then loss is $(0, 1 + z)_+$. Here we want to count some loss even if z and y_i have the same sign but $|z| < 1$ (correct prediction, but not by enough margin).

We will therefore write:

$$L = \sum_{i=1}^{D} \big(1 - y_i(\omega^\top x_i + b)\big)_+ \tag{6}$$

Where $(\cdot)_+ = \max(0, x)$. This type of loss (of the form $(1 - yz)_+$ where z is the score and y is the label) is called a Hinge loss function and is the basis for Support vector machines (a.k.a SVM).

4 Likelihood Estimator for Classification Problems

Sigmoid function is very commonly used in ML solution formulations. It is defined as :-

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

We have already seen ways to define the loss metric for classification problems. Here, we define a estimator of the probability of getting the given estimates from our ML model, that forms the basis of the Maximum Likelihood Estimator (MLE).
In order to use MLE, we have to make two important assumptions, which are typically referred to together as the i.i.d. assumption. These assumptions state that:

1. Data must be independently distributed.
2. Data must be identically distributed.

Our initial loss function was defined as :-

$$L = \sum_{i \in D} \mathbb{I}(\mathbf{H}(x_i) \neq y_i)$$

Since we can scale the loss function by any factor, let us divide it by $|D|$. So, now we have :-

$$\begin{aligned} L &= \frac{1}{|D|} \sum_{i \in D} \mathbb{I}(\mathbf{H}(x_i) \neq y_i) \\ &= \mathbb{E}\left[\mathbb{I}(\mathbf{H}(x) \neq y)\right] \\ &= Pr(\mathbf{H}(x) \neq y) \end{aligned}$$

The last step follows because we are taking expectation of the indicator function.

The value associated with a given data sample is given by the same linear transformation as before ($\vec{w}^T\vec{x} + b$). To get the probability, we apply the Sigmoid function to this value to get:-

$$\begin{aligned} Pr(y = 1|x) &= \frac{1}{1 + e^{-(\vec{w}^T\vec{x}+b)}} \\ Pr(y = -1|x) &= \frac{1}{1 + e^{(\vec{w}^T\vec{x}+b)}} \\ Pr(y|x) &= \frac{1}{1 + e^{-y(\vec{w}^T\vec{x}+b)}} \end{aligned}$$

Since, the data samples can be considered to be independent of each other, it makes intuitive sense to compute the product of the probabilities so obtained.

$$Pr(Y|X) = \prod_{i \in D} Pr(y_i|x_i)$$

This probability tells us the likelihood of getting the measurements we have using the linear transformation tuple $(\vec{w}, b)$. The higher this probability, the better. Equivalently; the lower the negative log likelihood, the better.

$$\begin{aligned} -\ln Pr(Y|X) &= \sum_{i \in D} -\ln Pr(y_i|x_i) \quad (7) \\ &= \sum_{i \in D} \ln\left(1 + e^{-y_i(\vec{w}^T\vec{x_i}+b)}\right) \quad (8) \end{aligned}$$

This may be taken as yet another loss metric to be minimised. In fact, the function $\ln(1 + e^z)$ happens to be similar to $(z)_+$ in shape (here $z = -y_i(\vec{w}^T\vec{x_i} + b)$) with the added benefit of being differentiable at 0. This supports the notion that the earlier formulation of loss function using $(z)_+$ is probably right.

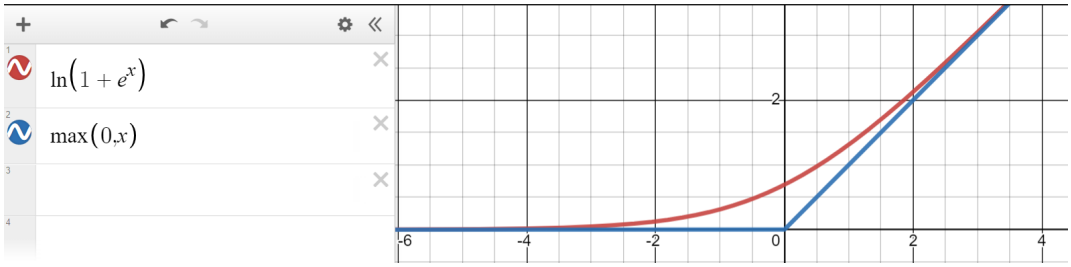


Figure 1: Comparing $(x)_+$ and $\ln(1+e^x)$

5 Example Problem for finding Loss Function

Imagine a website (Amazon, if you will) where products have 3 attributes :-

1. price (p)
2. quality (q)
3. brand (b)

We have to classify the products as 1 (products customer would be willing to buy) and -1 (products customer would not be willing to buy). We have the following condition to determine the value of y :-

$$y = \begin{cases} 1 & \vec{w_p}^T \vec{P} + \vec{w_q}^T \vec{Q} > 1 \\ 1 & \vec{w_p}^T \vec{P} + \vec{w_q}^T \vec{Q} < -1 \text{ and } \vec{w_b}^T \vec{B} \geq 0 \\ -1 & \vec{w_p}^T \vec{P} + \vec{w_q}^T \vec{Q} < -1 \text{ and } \vec{w_b}^T \vec{B} < 0 \end{cases}$$

We take the loss function as :-

$$loss = \sum_{i \in D} max(0, \vec{w_b}\vec{B_i} \cdot min(\vec{w_p}^T \vec{P_i} + \vec{w_q}^T \vec{Q_i}, 0)y - max(\vec{w_p}^T \vec{P_i} + \vec{w_q}^T \vec{Q_i}, 0)y)$$

Here, if $\vec{w_p}^T \vec{P_i} + \vec{w_q}^T \vec{Q_i} > 0$, then $loss = 0$ if $y = 1$, and positive if $y = -1$. If $\vec{w_p}^T \vec{P_i} + \vec{w_q}^T \vec{Q_i} < 0$, then if $\vec{w_b}^T \vec{B} > 0$, $loss = 0$ if $y = 1$ and positive if $y = -1$. Finally if $\vec{w_p}^T \vec{P_i} + \vec{w_q}^T \vec{Q_i} < 0$, and if $\vec{w_b}^T \vec{B} < 0$, $loss = 0$ if $y = -1$ and positive if $y = 1$. This agrees with the conditions given to us in the beginning and hence, the function is an appropriate loss function.

6 Ranking Problem

Up till now, we have been dealing with classification problems where we had to put labels on a specific data element. Now, we make our problem a bit more complex.

This problem is inspired by how search engines rank pages based on queries given by the user.

We have been given a set of pages $\{P_i\}$. Our dataset consists of tuples of queries and orderings of the form $\{Q_i, O_i\}$ where O_i is an ordering of the pages in P. The aim is to find the page ranking for queries Q_i^{new}, that are previously unseen.

The first step to solving this problem is to design an appropriate loss function.

We define the similarity between a query q and a page c as a feature vector $\vec{x_{qc}}$. If we use the linear weight $\vec{w}$ and bias b, then we can associate with each page (for a specific query) the value given by $\vec{w}^T\vec{x_{qc}} + b$.

Since we want a ranking here, we want a loss function such that :-

1. If $\vec{w}^T\vec{x_{qc_1}} + b > \vec{w}^T\vec{x_{qc_2}} + b$ and $r_{c_1} < r_{c_2}$ (i.e. c_1 comes before c_2), then we want the loss to be 0 or close to 0.
2. If $\vec{w}^T\vec{x_{qc_1}} + b > \vec{w}^T\vec{x_{qc_2}} + b$ and $r_{c_1} > r_{c_2}$ (i.e. c_1 comes after c_2), we want the loss to be positive, preferably a monotonic function of the difference between the ranks and difference between the values.

Here, we find an appropriate function :-

$$\sum_{c_1,c_2 \in P} max(1 + (r_{c_1} - r_{c_2})(\vec{w}^T(\vec{x_{qc_1}} - \vec{x_{qc_2}})), 0)$$

In case 1, the first argument of the max function becomes negative, and so it gives a 0 loss. In case 2, the first argument of the max becomes positive and the greater the difference in the ranks and values, the greater is the value of the loss.

We have added 1 to the first argument because we need to separate the values by a sizeable margin to reduce ambiguity and the effect of noise on the model. This is similar to the logic used before in the previous model.

AIML - CS 337

Lecture 6: Loss Function and Models in Regression

25 August 2022

Lecturer: Abir De *Scribe: Group 11 and Group 12*

In this lecture, we lay the foundations of regression, as well as initiate a discussion on the various loss functions used to judge the quality of a model, and how to choose appropriate loss functions based on the noise present and other circumstances.

1 Regression : Housing Prices Scenario

We begin with the problem of determining the price of a house based on the various *features* of the house, such as it's area, location etc. Assuming that all the features of the house can be quantized, we represent the *feature vector* of the house by the column vector $\boldsymbol{x} = [\text{Area} \;\; \text{Location} \;\; \ldots]^T \in \mathbb{R}^{n\times 1}$, where we assume we have n features. The price of the house is represented by the scalar y, and thus our **regression problem** asks for a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ which *closely simulates* the prices $\{y_i\}_{i\in D}$ given the feature vectors $\{\boldsymbol{x_i}\}_{i\in D}$.

If we constrain the space of functions $f : \mathbb{R}^n \mapsto \mathbb{R}$ such that f is linear in the feature variables, the problem is then termed as **linear regression**. f is of the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$$

where $w \in \mathbb{R}^n, b \in \mathbb{R}$ are *parameters* of f.

Note that deciding which features are useful in determining the price of the house and how to quantize them is another problem in itself, called the *feature selection problem*. We ignore this problem for now and dive into what is meant by "closely simulates" by exploring a couple of loss functions.

Devising a Loss Metric

To say that a function f *closely simulates* given data, we must have some form of error/loss metric that we try to minimize, such that the predictions from f are close to the true values.

Let $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ and **D** be dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^m, \boldsymbol{x}_i \in \mathbb{R}^{n\times 1}, y_i \in \mathbb{R}$. The error/loss function can be denoted as $F(f, D)$. For the current problem of regression, we assume $\{y_i\}$ to be continuous valued. Few candidates for the function could be :

- $\sum_i (y_i - f(\boldsymbol{x}_i))$

  Note: There is a glaring problem with this function in that we can minimize this infinitely towards $-\infty$ without actual predicting values near y_i

- $\sum_i |y_i - f(\boldsymbol{x}_i)|$
- $\sum_i (y_i - f(\boldsymbol{x}_i))^2$
- $\sum_i e^{(y_i - f(\boldsymbol{x}_i))^2}$
- $\sum_i |y_i - f(\boldsymbol{x}_i)|^3$ *This is rarely used*

In general, decision of the loss function to choose depends on

- Distribution of errors
- Demands of the problem (For instance, extent of penalization for outliers)

We now explore some commonly used loss functions in detail -

Mean Squared Loss

$$F(f, D) := \sum_{\{\boldsymbol{x}_i, y_i\} \in D} (y_i - f(\boldsymbol{x_i}))^2$$

One may have seen this loss in many places and for good reason: **Minimizing this loss is equivalent to maximizing the likelihood** $P(\{y_i\}_{i \in D} | \{\boldsymbol{x_i}\}_{i \in D})$ **under the assumption there is a Gaussian noise between** $f(\boldsymbol{x_i})$ **and** y_i.
Let us prove this:
Assuming that y indeed varies over $\boldsymbol{x}$ according to the distribution f, but with an iid Gaussian (with mean 0 and variance σ^2) noise ε, one notes that $P(y|\boldsymbol{x})$ is equivalent to asking how likely is it that the noise in y was sampled from a Gaussian distribution, and thus

$$P(\{y_i\}_{i \in D} | \{\boldsymbol{x_i}\}_{i \in D}) = \prod_{i \in D} P(y_i | \boldsymbol{x_i}) \propto \prod_{i \in D} e^{-\frac{(y_i - f(\boldsymbol{x_i}))^2}{2\sigma^2}} = \exp\left(\frac{-1}{2\sigma^2} \sum_{i \in D} (y_i - f(\boldsymbol{x_i}))^2\right)$$

where the first equality follows from an independence assumption. Then one can clearly see that minimizing $F(f, D)$ maximizes the likelihood that the prices indeed are drawn from the distribution we have modelled. An extra merit of above loss is that the gradient of the function becomes zero when the loss becomes zero.
As for the fact that we assumed our noise to be Gaussian, this is a standard assumption throughout machine learning: In unknown scenarios where the nature of the noise is unclear, assuming Gaussian noise generally works well in practical cases. The belief in the Gaussian nature of the noise is so deep-rooted, that one of the common techniques in machine learning to simulate noise is as follows

$$\varepsilon_{\boldsymbol{x}} \sim \mathcal{N}(0, \xi(\boldsymbol{x})) = \mathcal{N}(0, \sigma_{\boldsymbol{x}}^2)$$

where $\xi : \mathbb{R}^n \mapsto \mathbb{R}$ represents a neural network which takes as input $\boldsymbol{x}$ and returns the variance of the noise (assumed to be a mean 0 Gaussian RV) between our predictions and the ground reality.
However, the pre-eminence of Gaussian noise models (and consequently squared-error functions) does by no means preclude the possibility of other types of noises, and thus we shall explore another common error function, one based on the ℓ_1-norm.

The ℓ_1-norm loss

The ℓ_1-norm loss is formally defined as

$$F(f, D) := \sum_{i \in D} |y_i - f(\boldsymbol{x_i})|$$

One of the most apparent scenarios which comes to mind is that if we know that the noise in our model follows a **Laplacian distribution**, ie:- $\propto \exp\left(\frac{-|x-\mu|}{b}\right)$. Then similar to the proof given above, a ℓ_1-norm loss would maximize the likelihood of our model simulating the ground reality.

However, it isn't the case that the ℓ_1-norm loss arises only in such esoteric cases: Indeed, imagine a scenario where our loss should encourage sparsity in the final model; such scenarios commonly occur in the fields of image compression, compressed sensing, sparse graphs (in the context of social media relations), etc. Note that the mean squared error forces low errors on every point in our domain D, since any error anywhere is amplified by the square function and increases the overall loss. However, as mentioned above, in applications where sparsity is more of a priority, large errors in some parts of our domain can be forgiven as long as the error on most of the domain is very low, ie:- if one takes the difference of the ground reality and our predictions, then the difference **forms a sparse vector/matrix, ie:- a vector most of whose entries are 0**.

Given this background, one might wonder why don't we minimize the ℓ_0-norm instead, where the ℓ_0-norm of a vector is defined to be the *number of non-zero elements of the vector*. Indeed, a ℓ_0-loss exactly satisfies the sparsity requirement. Unfortunately, minimizing the ℓ_0-norm is a **NP-hard** problem in most scenarios, and thus is an intractable loss function. Interestingly, the ℓ_1-norm, under *reasonable regularity conditions*, yields almost the same optimal solution as the ℓ_0-norm would have yielded. This is a fact widely employed in the field of compressed sensing where ℓ_1-norms are routinely deployed as proxies for the ℓ_0-norms to induce sparsity in the model.

How to choose the correct loss function? The distribution of errors is not generally given and neither can be guessed directly, We can test out a loss function by training over a *training dataset* and testing the model performance over a *validation dataset* disjoint from the training dataset. We can then compare performance metrics to decide the best loss function.

The decision of choosing loss function depends on the user requirement too. For cases when we want a larger penalty for larger errors i.e. prevent blatant outliers, **Mean Squared Loss** function is better. Whereas, when we do not care about outliers as much, then $\boldsymbol{\ell_1}$ **Norm** is better.

2 Conclusion

In short, what noise model is to be deployed depends on the context and the requirements: If very little is known about the noise, and the requirements are generic, then the mean squared error

usually works well. However, if the requirements are such that sparsity is required, ie:- almost perfect accuracy on most predictions at the cost of some big misses for a few, then the ℓ_1-norm loss is preferred.

Analytic Derivation of Linear Regression under MSE loss

Consider the linear regression function $f(\boldsymbol{x_i}) = \boldsymbol{w^T x_i} + b$, where $\boldsymbol{w}, \boldsymbol{x_i} \in \mathbb{R}^n$ and $b \in \mathbb{R}$.
Under a MSE loss, if we're to find the best model, then the problem can be stated mathematically as

$$(\boldsymbol{w}_*, b_*) := \arg\min \sum_{i \in D} (y_i - \boldsymbol{w^T x_i} - b)^2$$

It's not hard to show that if we assume that the $\boldsymbol{x_i}$'s are sampled from some mean zero distribution, then $\mathbb{E}[y] = b$, and thus one can set $b_* = \mathbb{E}[y]$.
Thus henceforth we'll assume $b = 0$. Thus

$$\boldsymbol{w}_* := \arg \frac{d}{d\boldsymbol{w}} \left(\sum_{i \in D} (y_i - \boldsymbol{w^T x_i})^2 \right) = 0$$

Side Note:
We can vectorize this summation by stacking the y_is and $\boldsymbol{x_i}$s, and obtain analytical solution using matrix calculus.
Suppose we have n features and d data points so that $\mathbf{X} \in \mathbb{R}^{n \times d}, \mathbf{y} \in \mathbb{R}^{1 \times d}$, then we'll have the solution as $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$. To get to this solution, we make the assumption that $\mathbf{X}\mathbf{X}^\top$ is invertible. We can't proceed further without that assumption.

Next, let's focus on the case with just one data point. Here, $\boldsymbol{w}_* := \arg \frac{d}{d\boldsymbol{w}} \left((y - \boldsymbol{w^T x})^2 \right) = 0$

Simplifying the equation $\frac{d((y - \boldsymbol{w^T x})^2)}{d\boldsymbol{w}} = 0$ yields $\boldsymbol{x x^T w}_* = y\boldsymbol{x}$. However, without further assumptions, this is analytically intractable.

To circumvent this issue, we employ a neat trick: We minimize $(y - \boldsymbol{w^T x})^2 + \lambda \|\boldsymbol{w}\|_2^2$ instead, and hope that we'll get the right answer as $\lambda \to 0$. An interesting observation about this formulation is that if we assume the probability distribution of $\boldsymbol{w}$ to be **Gaussian** and maximize the posterior,

$$\max_{\boldsymbol{w_*}} P(y|\boldsymbol{x})P(\boldsymbol{w}) \propto \max_{\boldsymbol{w_*}} \prod_{i\in D} \exp\left(-\frac{(y_i - \boldsymbol{w^T x_i})^2}{2\sigma^2}\right) e^{-\frac{||\boldsymbol{w_*}||}{2\sigma_*^2}}$$
$$= \max_{\boldsymbol{w_*}} \exp\left(\frac{-1}{2\sigma^2}\sum_{i\in D}(y_i - \boldsymbol{w^T x_i})^2 \; - \; \frac{1}{2\sigma_*^2}\|\boldsymbol{w}\|_2^2\right)$$

We see that maximizing $P(y|\boldsymbol{x})P(\boldsymbol{w})$ likelihood is same as minimizing $\sum_{i\in D}(y_i - \boldsymbol{w^T x_i})^2 + \lambda\|\boldsymbol{w}\|_2^2$ with $\lambda = \frac{1}{2\sigma_*^2}$.

This is, in short, the **Bayesian interpretation** of *regularised* L2 regression.

The expression one gets on simplifying $\frac{d((y-\boldsymbol{w^T x})^2+\lambda\|\boldsymbol{w}\|_2^2)}{d\boldsymbol{w}} = 0$ is $(\lambda\boldsymbol{I} + \boldsymbol{xx^T})\boldsymbol{w}_* = y\boldsymbol{x}$.One can then use the **Sherman-Morrison formula** to get that

$$\boldsymbol{w}_* = \frac{y}{\lambda}\left(\boldsymbol{I} - \frac{\boldsymbol{xx^T}}{\boldsymbol{\lambda} + \|\boldsymbol{x}\|_\mathbf{2}^\mathbf{2}}\right)\boldsymbol{x}$$

Unfortunately, this expression doesn't converge as $\lambda \to 0$. However, as λ becomes small, $\boldsymbol{w}_*$ becomes approximately parallel to the vector $\boldsymbol{v} := \left(\boldsymbol{I} - \frac{\boldsymbol{xx^T}}{\|\boldsymbol{x}\|_\mathbf{2}^\mathbf{2}}\right)\boldsymbol{x}$.

AIML - CS 337

Lecture 7: Linear Regression- Regularization and Stability

August 29^{th}, 2022

Lecturer: Abir De *Scribe: Group 13, Group 14*

In the previous lecture, we briefly discussed the regression problem and explained the different types of loss functions we can use to train a regression model. This lecture will be a continuation of the linear regression problem and we will introduce a very important technique to improve loss functions called regularization. In the later part of this lecture, we will learn about a new feature of models called "stability".

1 Choosing between loss functions: MSE vs Absolute Error

In the previous class, we also discussed the comparison between MSE loss function and the absolute error function. We had discussed that minimizing the MSE loss is akin to obtaining the maximum likelihood estimate for the target function f, assuming that the noise in the y was sampled from a Gaussian distribution.

Similarly, minimizing the absolute error loss is the same as obtaining the MLE for the target function given that the noise was sampled from a Laplacian Distribution.

The user prefers the absolute error function when the user wants to maximize the number of correct predictions, but does not care much about the error when a prediction is wrong. On the other hand, the MSE loss function is preferred when the difference between the actual value and the predicted value is also of significance.

2 Regression

Continuing where we left in the last class, we have the following loss function which needs to be minimized to get optimal parameters for the regression problem:

$$L(\boldsymbol{w}, b) = \sum_{i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x_i} - b)^2$$

$$\text{s.t.} \quad y_i \in \mathbb{R}, \quad \boldsymbol{x_i} \in \mathbb{R}^d \quad i.e. \quad \boldsymbol{x_i} = \begin{bmatrix} x_i^1 & x_i^2 & \cdots & x_i^d \end{bmatrix}^T$$

Here, dataset $D = \{\boldsymbol{x_i}, y_i\}$ is the set of all points over which the analysis is done, $\boldsymbol{x_i}$ is the input vector for i^{th} sample and d is the number of features in each vector $\boldsymbol{x_i}$. y_i is the **label** associated with each **sample** $\boldsymbol{x_i}$ and **weight** $\boldsymbol{w} \in \mathbb{R}^d$ is the vector of weights assigned to each individual feature.

Observe that we can avoid explicitly mentioning b in the loss function, since it is a scalar and a modification of $\boldsymbol{x_i}$ can incorporate it. Thus our minimization problem reduces to:

$$\begin{aligned}
\min_{\boldsymbol{w}} L'(\boldsymbol{w}) = & \quad \min_{\boldsymbol{w}} \quad \sum_{i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2 \\
= & \quad \min_{\boldsymbol{w}} \quad \sum_{i \in D} (y_i^2 - 2\boldsymbol{w}^T \boldsymbol{x_i} y_i + (\boldsymbol{w}^T \boldsymbol{x_i})^2) \\
= & \quad \min_{\boldsymbol{w}} \quad (\vec{\boldsymbol{y}} - \boldsymbol{X}\boldsymbol{w})^T (\vec{\boldsymbol{y}} - \boldsymbol{X}\boldsymbol{w}) \\
= & \quad \min_{\boldsymbol{w}} \quad (\vec{\boldsymbol{y}}^T \vec{\boldsymbol{y}} - 2\vec{\boldsymbol{y}}^T \boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{w})
\end{aligned}$$

$$\text{where} \quad \vec{\boldsymbol{y}} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} \text{ and } \quad n = |D|$$

On differentiating w.r.t $\boldsymbol{w}$ and equating the *gradient* to 0, we get:

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L'(\boldsymbol{w}) = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \vec{\boldsymbol{y}}) &= 0 \\
\boldsymbol{X}^T \vec{\boldsymbol{y}} &= \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{w} \\
\boldsymbol{w} &= (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \vec{\boldsymbol{y}} \\
\boldsymbol{w} &= (\boldsymbol{X}^T \boldsymbol{X})^{-1} \sum_{i \in D} y_i \boldsymbol{x_i}
\end{aligned}$$

Thus, we have an expression for the optimal $\boldsymbol{w}$ ($\boldsymbol{w}^*$). Observe: we need $\boldsymbol{X}^T\boldsymbol{X}$ to be invertible for this model to work. In the following subsection, we will discuss this in detail.

2.1 Invertibility of $\boldsymbol{X}^T\boldsymbol{X}$

Note that, $\boldsymbol{X}^T\boldsymbol{X}$ will be invertible if and only if the determinant of the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is non-zero, $i.e.$, the matrix is non singular.

We know that each $\boldsymbol{x_i}$ is an i.i.d (identical, independent draw) from a continuous distribution. Thus for $n > d$ where $\boldsymbol{X}$ is of dimensions $n \times d$, we can claim that $|\boldsymbol{X}^T\boldsymbol{X}|$ has a continuous distribution. Hence, the probability that determinant of $\boldsymbol{X}^T\boldsymbol{X}$ will take exactly 0 value is 0, $i.e.$

$$P(|\boldsymbol{X}^T\boldsymbol{X}| = 0) = 0$$

Intuitively, we can also see that the matrix X is highly likely to be full rank. A dxN matrix (where $N \geq d$) is full rank if atleast d columns out of the N columns are linearly independent.

At the same time, note that the probability of $|\boldsymbol{X}^T\boldsymbol{X}|$ lying within some ϵ near 0 is not zero $i.e.$ $P(|\boldsymbol{X}^T\boldsymbol{X}| < \epsilon) \neq 0$

In mathematics, a condition number is a number representative of the change of an output proportionate to a change in the input of a function. For example, if a small change in the input results in a small change in the output, the function produces a small condition number and is said to be well-conditioned. Alternatively, if a small change in the input results in a large change in the output, the function produces a large condition number and is defined as ill-conditioned. The condition number is mathematically given as the ratio of the maximum and minimum eigenvalues.

Poorly conditioned matrix A is a matrix with a high condition number. A^{-1} amplifies input errors. Small errors in x can change the output of $A^{-1}x$ rapidly.

Definition 2.1. Condition number of a matrix A is the ratio of its minimum eigenvalue to maximum eigenvalue

$$Cond(\boldsymbol{A}) = \frac{min(eigen(\boldsymbol{A}))}{max(eigen(\boldsymbol{A}))}$$

Condition number of a matrix **A** is a good indicator of the invertibility of the matrix. A high condition number results in a well conditioned matrix which increases the chances of **A** being invertible, whereas, a low condition number results in an ill conditioned matrix, $i.e.$ reduced chances of invertiblilty.

We want to ensure that the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is well-conditioned. One possible approach for this is replacing $\boldsymbol{X}^T\boldsymbol{X}$ with $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$, $i.e.$

$$\boldsymbol{w}* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\vec{\boldsymbol{y}}$$

Note here that such a transformation changes the condition number as follows:

$$\begin{aligned}
&Cond(\boldsymbol{X}^T\boldsymbol{X}) \rightarrow Cond(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}) \\
&\frac{min(eigen(\boldsymbol{X}^T\boldsymbol{X}))}{max(eigen(\boldsymbol{X}^T\boldsymbol{X}))} \rightarrow \frac{min(eigen(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}))}{max(eigen(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}))} \\
&\frac{a}{b} \rightarrow \frac{a+\lambda}{b+\lambda} \quad \text{(assuming} \quad Cond(\boldsymbol{X}^T\boldsymbol{X}) = \frac{a}{b})
\end{aligned}$$

Thus, the $\lambda\boldsymbol{I}$ term adds a lower bound to the value of condition number improving the chances of invertibility of desired matrix.

Also, we find that this type of $\boldsymbol{w}*$ is the solution to the following loss minimization problem:

$$\min_{\boldsymbol{w}} \quad \sum_{i \in D}(y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2 + \lambda\|\boldsymbol{w}\|^2$$

More about this in the next section.

3 Regularization

Theoretically, **regularization** is a popular technique used to reduce errors by fitting the function appropriately on the given training set and to avoid overfitting

The most common regularization techniques are:

1. L1 regularization
2. L2 regularization
3. Dropout regularization

Here we will discuss L2 regularization.

We saw an example of L2 regularization at the end of the last section trying to ensure that the matrix whose inverse we are calculating is well conditioned. Following was the loss minimization expression for it:

$$\min_{\boldsymbol{w}} \quad \sum_{i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2 + \lambda \|\boldsymbol{w}\|^2$$

Here, $\lambda \|\boldsymbol{w}\|^2$ is called the **regularizer** term. In the following sections, we discuss some **advantages** of regularization.

3.1 Ensures Well-Conditioned Matrix

As we saw in **Section 1.1**, if we add the L2 regularizer term to our Loss function then the solution for the optimal $\boldsymbol{w}^*$ will be:

$$\boldsymbol{w}* = (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \vec{\boldsymbol{y}}$$

Instead of:

$$\boldsymbol{w}* = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \vec{\boldsymbol{y}}$$

The regularizer adds a lower bound to the condition number of the desired matrix matrix, thus increasing the chances of invertibility of $\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I}$ compared to $\boldsymbol{X}^T \boldsymbol{X}$.

3.2 Prevents Overfitting

Overfitting occurs when the model is constrained to the training set and not able to perform well on the test set, here the gap between the training error and testing error is large.

Example:
Let us look at a regularization problem with the following loss function:

$$\min_{\boldsymbol{w}} \quad \sum_{i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2$$

Suppose the number of features in each data point $\boldsymbol{x_i}$ in the dataset increases significantly say from d_1 to d_2 such that $d_2 > d_1$. In response to this, the dimensions of our learnable parameter $\boldsymbol{w}$ also change from $d_1 \times 1$ to $d_2 \times 1$. Thus, the overall model will become more **complicated**.
Also, more features will help bring down the training loss and will increase the training accuracy, but since the model has been trained on the same training set with more features, it loses **generalization**. The **test accuracy** will become significantly **low**er due to this overfitting of model on training data.

3.2.1 How to avoid overfitting?

The solution is again regularization. On adding regularizer term to our original L2-loss, we obtain the new loss function as follows:

$$\min_{\boldsymbol{w}} \quad \sum_{i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2 + \lambda \|\boldsymbol{w}\|^2$$

If the number of features of $\boldsymbol{x_i}$ increases then the dimension of $\boldsymbol{w}$ will also increase, and hence the dimension of $\boldsymbol{w}*$ will increase. Due to this increase in dimension the value of the regularizer term $\lambda \|\boldsymbol{w}\|^2$ rises, increasing the overall loss value.

So, the training loss can be greater with a higher number of features compared to one obtained with less number of features $i.e.$ with the regularized loss function, a model with higher number of features is not necessarily a better model for the training set. This helps prevent overfitting of the model.

3.2.2 Alternate technique to avoid overfitting

If regularization is not allowed, how to decide the optimal number of features to train the model?

We can use the method discussed here to find the optimal subset of features to be used for a training set with a large number of extractable information/features. Let the total number of features be d, the algorithm follows below:

- Separate a portion of the training data as **validation set**.
- Iterate over all subsets $\boldsymbol{s}$ of features with cardinality $|\boldsymbol{s}| = d' \leq d$

- Compare across all these subsets by finding their loss on the validation set. Choose the set of features that gives minimum loss on the validation set.

The above steps can be summarized by the following loss function:

$$\min_{\boldsymbol{w_s}, \boldsymbol{s}} \quad \sum_{i \in D} (y_i - \boldsymbol{w}_s^T \boldsymbol{x_{i,s}})^2$$

Here, $|\boldsymbol{s}| \leq d$ and $\boldsymbol{x_{i,s}}$ refers to the subset of $\boldsymbol{x_i}$ which has features corresponding to the set $\boldsymbol{s}$.

But this optimization problem is **NP Hard** due to the exponential search space involving iteration over all subsets.

4 Stability

A model is called stable if on addition of a new data point to training dataset, the learnable parameter $\boldsymbol{w}$ does not change very much.

Mathematically speaking, let D be our initial training dataset and $D \cup k$ be the new training dataset with the addition of a new data point k. Then we define the following quantity:

$$l = \|\boldsymbol{w}^*_{(D)} - \boldsymbol{w}^*_{(D \cup k)}\|$$

Stability is ensured when this l is small. This means that on addition/modification/removal of a data point, the learnable parameter $\boldsymbol{w}$ did not $change$ very much.

Mathematically, we call a model stable if l follows this condition:

$$l = \|\boldsymbol{w}^*_{(D)} - \boldsymbol{w}^*_{(D \cup k)}\| < \epsilon, \quad \text{where} \quad \epsilon \sim O(\frac{1}{n}) = O(\frac{1}{|D|})$$

where $|D| = n$ is the size of the complete dataset.

We can prove that if the model follows the stability condition stated above then,

$$\text{test-case error} \leq \alpha\epsilon + \beta$$

where, α and β are constants that are not in our control but ϵ is decided by model predictions.

The constant β is decided by the quality of dataset provided. For example if, $\boldsymbol{x_i}$ is sampled from a normal distribution $\mathcal{N}(0, 1)$ and corresponding y_i are sampled from uniform distribution $U(0, 1)$, then the value of constant β will be higher since dataset is randomly created.

4.1 Advantages of Stability

4.1.1 Smaller dataset needed for training

When a model is trained on a small dataset D but is stable, then on addition of a new data point **k** the optimal learnable parameter $\boldsymbol{w}*$ does not change a lot $i.e.$, the previously learned $w^*(D)$ is similar to $w^*(D \cup k)$ even on addition of a new unseen point.

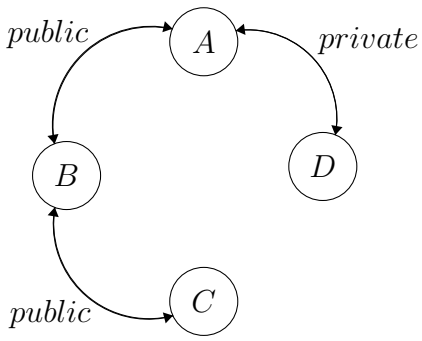
Thus, our model behaves as if it has been trained on the larger set and will also work well on unseen test samples.

4.1.2 Prevents leak of information to adversary

This is one of the information security concerns of the model. Suppose on addition of a data point k, the model's learnable parameter $\boldsymbol{w}^*$ changes a lot. This can reveal important characteristics of the model to the adversary which can be used to predict label y_i of the model on input x_i with some confidence and thus an adversary can exploit the model.

To understand this point better, let us present a small example:

Suppose A, B, C, D are 4 people on Facebook. As shown in the diagram below, Person A has a private Facebook connection with D which is not visible to others. All other connections are publicly visible.



Now suppose that the Facebook algorithm recommends Person D's profile to Person C, then C can estimate with certain confidence that either of B or A has a private connection with D. If Person C get to know that Person B has no private connection then Person C can say with certainty that Person A has a private connection with Person D.

Thus, the recommendation algorithm of Facebook's model could have been designed better to ensure data privacy in such a case.

4.2 Individual and Overall Loss

Definition 4.1. Individual loss: $l(\boldsymbol{w}^T\boldsymbol{x_i}, y_i) = (y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2 + \lambda\|\boldsymbol{w}\|^2$

Definition 4.2. Overall loss: $\sum_{i\in D} l(\boldsymbol{w}^T\boldsymbol{x_i}, y_i)$

Notice the subtle difference between the regularized l2-loss introduced in section 2 and the overall loss function described here. In the overall loss definition, $\lambda\|w\|^2$ will be repeated $|D|$ times where $|D|$ is the size of the dataset while in the regularized l2-loss, it is included exactly once.

The relation between λ (for overall loss function of D) and λ' (for overall loss function of $D \cup k$):

$$\lambda' = \lambda\frac{|D \cup k|}{|D|}$$

4.3 Bounding the *change* in Overall Loss

We will attempt to obtain upper and lower bounds for the quantity $\|L(D\cup k, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)})\|$. Notice that this expression seems to model the $change$ in overall loss on addition of a new datapoint k to D, $i.e.$:

$$L(D, \boldsymbol{w}^*_{(D)}) = \sum_{i\in D} l(\boldsymbol{w}^*_{(D)}\boldsymbol{x_i}, y_i) \tag{1}$$

To upper bound this quantity we will use Lipchitz Continuity or the 'Lipchitzness' of overall loss function.

A function is Lipschitz continuous if there exists a constant L such that:

$$\|f_x - f_y\| \leqslant L\|x - y\| \quad \forall x, y \tag{2}$$

Thus, we get:

$$\begin{aligned}
\|L(D \cup k, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)})\| &= \|L(D, \boldsymbol{w}^*_{(D\cup k)}) + L(k, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)})\| \\
&= \|L(D, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)}) + L(k, \boldsymbol{w}^*_{(D\cup k)})\| \\
&\leq \|L(D, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)})\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\|
\end{aligned}$$

Which implies:

$$\begin{aligned}
&\|L(D, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)}) + L(k, \boldsymbol{w}^*_{(D\cup k)})\| \\
&\quad\leq \|\sum_{i\in D} l(\boldsymbol{w}^*_{(D\cup k)}\boldsymbol{x_i}, y_i) - \sum_{i\in D} l(\boldsymbol{w}^*_{(D)}\boldsymbol{x_i}, y_i)\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\| \quad \text{from (1)} \\
&\qquad\leq \sum_{i\in D} \|l(\boldsymbol{w}^*_{(D\cup k)}\boldsymbol{x_i}, y_i) - l(\boldsymbol{w}^*_{(D)}\boldsymbol{x_i}, y_i)\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\| \\
&\qquad\leq \sum_{i\in D} \alpha\|\boldsymbol{w}^*_{(D\cup k)} - \boldsymbol{w}^*_{(D)}\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\| \quad \text{from (2)} \\
&\qquad= \alpha|D|\|\boldsymbol{w}^*_{(D\cup k)} - \boldsymbol{w}^*_{(D)}\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\|
\end{aligned}$$

Hence we have finally:

$$\|L(D \cup k, \boldsymbol{w}^*_{(D\cup k)}) - L(D, \boldsymbol{w}^*_{(D)})\| \leq \alpha|D|\|\boldsymbol{w}^*_{(D\cup k)} - \boldsymbol{w}^*_{(D)}\| + \|L(k, \boldsymbol{w}^*_{(D\cup k)})\|$$

In the above proof we have used **Triangle inequality** repeatedly in the initial steps and finally used Lipschitz continuity condition on the loss function with **Lipchitz constant $\boldsymbol{\alpha}$**. We can use this condition on our loss function since it is differentiable and differentiability implies Lipchitz continuity.

The next lecture will carry forward this discussion of Lipchitzness for proving an upper bound and will introduce a lower bound for the $change$ in overall loss using convexity of the loss function.

AIML - CS 337

Lecture 8: Classifiers and Loss Functions, Tutorial 1 Solutions

September 1, 2022

Lecturer: Abir De *Scribe: Group 15 and Group 16*

1 Classifiers

Suppose we are given two datasets - $\mathcal{D}_{\text{Train}} = \{\mathcal{X} \times \mathcal{Y}\}^M$ and $\mathcal{D}_{\text{Test}} = \{\mathcal{X} \times \mathcal{Y}\}^N$. Here $\mathcal{X} \subset \mathbb{R}^d$ is a set consisting of d-dimensional input features, while $\mathcal{Y} \subset \mathbb{R}$ consists of the corresponding labels.

The true classifier h^* satisfies $h^*(x_j) = y_j$ for all $(x_j, y_j) \in \mathcal{D}_{\text{Test}}$. On the other hand, we develop a classifier $\widehat{h}$ using only the training dataset, satisfying $\widehat{h}(x_i) = y_i$ for all $(x_i, y_i) \in \mathcal{D}_{\text{Train}}$. Our aim is to find the best classifier $\widehat{h}$ which can mimic the true classifier h^*.

2 Types of Classifiers

2.1 Complex $\widehat{h}$

The most obvious classifier is simply a dictionary for $\mathcal{D}_{\text{Train}}$, i.e. a **Table Look-up** function. Here, we remember the label of each $x_i \in \mathcal{X}_{\text{Train}}$ in our classifier $\widehat{h}$. However, even though this can represent any function, this classifier will behave terribly on unseen data from the test set, on which we have no prior information about the label. Another disadvantage is that, as the data complexity of the dataset itself increases, storage and maintenance of this type of classifier can become cumbersome. Hence, we discard this classifier.

2.2 Modest $\widehat{h}$

2.2.1 Voronoi classifier

Another class of non-parametric classifiers consists of those using nearest neighbors. For example, the **Voronoi classifier** involves making a convex polyhedron (called the Voronoi Cell) for each point in the training dataset and then finding the polyhedron in which a new data point lies.

2.2.2 Exhaustive k-Nearest Neighbor classifier

Another example is an **Exhaustive k-Nearest Neighbor classifier**. Here, k is a hyperparameter and not a learnable parameter. The idea is, for each new test data point, find the k nearest neighbors from the training data, and give the majority class of these k points. Note that this does not require any processing of the training data, and so the training time is negligible (probably the only $\mathcal{O}(0)$ training time classifier we would see).

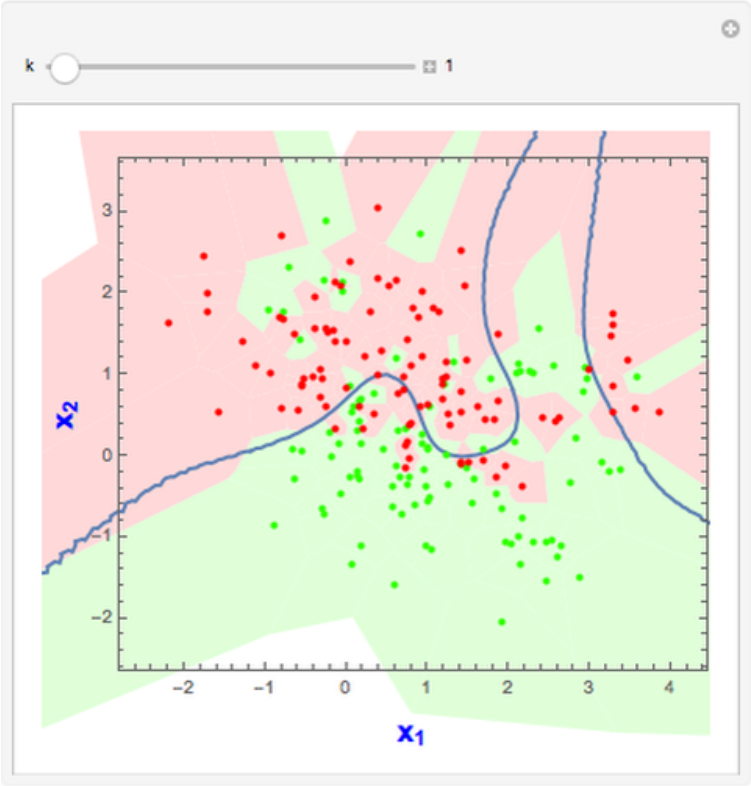


Figure 1: 1-NN classifier for data with 2 classes

The benefit of the k-NN algorithm is that it works nicely with non-linear data. However, during inference, we have to find the k nearest neighbors, which causes linear time blowup (time complexity will be of order $nd + n \log k$). This is undesirable in most situations, as we generally want testing to be fast so that the user does not have to wait for a long duration. Thus, we now move on to parameterised algorithms.

2.3 Simple $\widehat{h}$

A simple function for the classifier is the linear function, i.e. $\widehat{h} = w^T x + b$. Note that this classifier may not work suitably for all possible datasets, and if the given dataset is highly non-linear, then this classifier would perform poorly.

2.3.1 Constant $\widehat{h}$

A possible restriction to linear classifiers is the set of constant classifiers. The best constant classifier $\widehat{h} = c^*$ is given by

$$c^* = \arg\min_c \sum_{(x_i, y_i) \in \mathcal{D}_{\text{Train}}} \mathbb{I}[c \neq y_i]$$

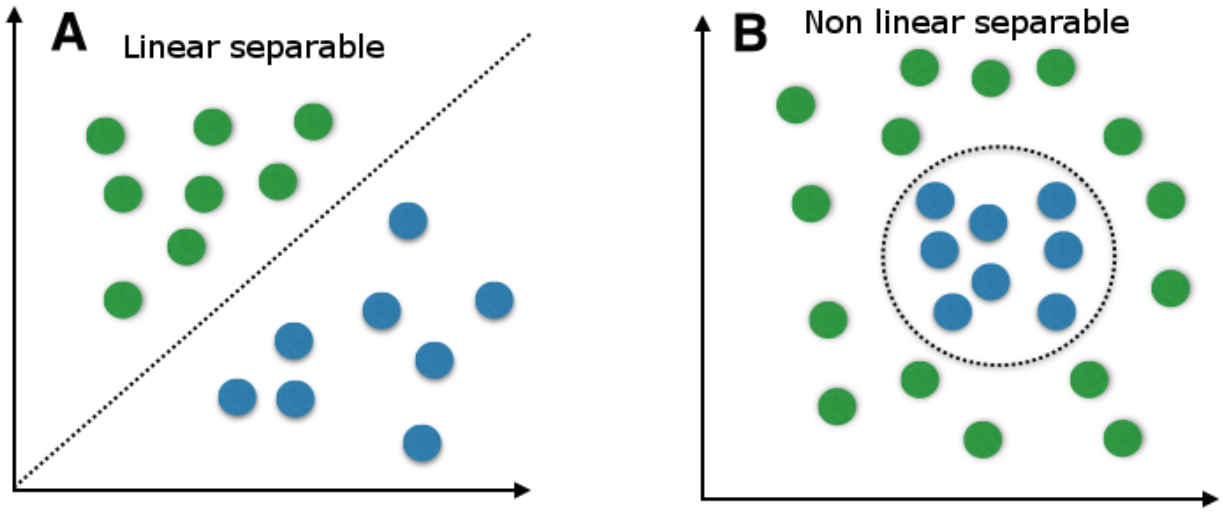


Figure 2: Linear v/s Non-linear data

This can easily be found as the mode label (also known as majority class) of the training data. However, although this classifier is easy to learn, it can perform poorly in the case when the distribution of the training data and the test data differs even slightly.

So our focus is mostly on linear classifiers. Note that we don't go to higher dimensions (like quadratic functions), since the problem at hand is already pretty hard.

2.4 Hypothesis Class

The hypothesis class is the set of classifiers over which we are searching for the optimal. For example, for a linear classifier, this class is given by all possible linear functions, i.e. all possible pairs (w, b) with weight $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$.

3 Error

For any classifier $\widehat{h}$, the error that we aim to minimize is given by

$$\text{Error}(\widehat{h}) = \sum_{(x_j, y_j) \in \mathcal{D}_{\text{Test}}} \mathbb{I}[\widehat{h}(x_j) \neq y_j]$$

Our aim is to minimize the test error, but our classifier must be learned from the training dataset. In particular, the best classifier h^* satisfies

$$h^* = \arg\min_{\widehat{h} \in H} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{Train}}} \mathbb{I}[\widehat{h}(x_i) \neq y_i]$$

Note that, if the set of classifiers over which we are looking is not exhaustive enough, i.e. we are missing some crucial classifiers, then the error for any classifier in H will be extremely large. So our hypothesis class should be able to generalize on $\mathcal{D}_{\text{Test}}$.

3.1 Linear Hypothesis Class

Our aim is to find the optimal pair $\{w^*, b^*\}$ such that

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}[w^T x_i + b \neq y_i]$$

However, since we are searching over all $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, $w^T x_i + b$ is a continuous function, which means that its probability of actually becoming exactly equal to a real number y_i is 0. In such a situation, the above error will become M with probability 1. Thus, we must change our error function.

3.2 Discrete error - Based on Signum Function

Let's assume that there are only 2 classes, -1 and 1, i.e. for all $1 \leq i \leq M$, we have $y_i \in \{-1, 1\}$. One option is to make our prediction 1 if $w^T x_i + b$ is non-negative, and -1 otherwise. This can be written as the classifier $h(x_i) = (w^T x_i + b)$. Thus, we wish to find the parameters

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}[(w^T x_i + b) \neq y_i]$$

Note that the set $\mathcal{D}_{\text{Train}}$ is **scarce** (does not cover a lot of data points), so any discrete classifier will not be able to generalize well over the test set. Hence, it is better to look for **probabilistic classifiers**. Another problem with a discrete classifier is that it will not be possible to find derivatives, which are required during gradient descent.

3.3 Final Proposal for Error - Based on Sigmoid Function

To mimic the discrete classifier, we replace the signum function with the sigmoid function. So our classifier will give the prediction $f(x_i) = \dfrac{1}{1 + e^{-w^T x_i + b}}$. Now we are looking for the parameters

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}\left[\frac{1}{1 + e^{-w^T x_i + b}} \neq \frac{y_i + 1}{2}\right]$$

Since the sigmoid function squeezes our prediction between 0 and 1, we have also changed the true prediction -1 to 0, by shifting $y_i \longrightarrow \dfrac{y_i + 1}{2}$.

Finally, the error function is still discrete, and so we replace it with a continuous function, giving the final classification problem proposal:

$$\boxed{\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \max\left(0, \left(\frac{1}{2} - f(x_i)\right) y_i\right)}$$

4 Solutions to Tutorial 1

Presented below are the solutions to the first tutorial:

4.1 Problem 1

A loss function is a good approximator of $\mathbb{I}[(h(x_i)) \neq y_i]$ if it takes a high value when y_i and $h(x_i)$ are of different signs, while it is close to 0 when they have the same sign.

4.1.1 Part (i)

max{0, 1 - $y_i \cdot h(x_i)$}

If y_i and $h(x_i)$ have the same sign, then $1 - y_i h(x_i)$ is lesser than 1, and so it's max with 0 is close to 0 (or 0 only). On the other hand, when they have different signs, $1 - y_i h(x_i)$ is highly positive, as desired.

Note that the term 1 is also added here, to ensure that small noises do not lead to significant errors in the function output.

y	h(x)	Loss
+1	> 0	0
-1	> 0	> 0
+1	< 0	> 0
-1	< 0	0

Answer - YES

4.1.2 Part (ii)

$$min\{0, 1 - y_i \cdot h(x_i)\}$$

This works complementary to the first subpart. So when y_i and $h(x_i)$ have different signs, the term $1 - y_i h(x_i)$ is positive, making the loss 0. While, in the other case, the loss is either 0 or a negative value. Although this follows the trend to some extent, it does not help in penalizing the bad cases sufficiently. Hence, this is a bad loss function.

Answer - NO

4.1.3 Part (iii)

$$\frac{\exp(-y_i \cdot h(x_i))}{1 + \exp(-y_i \cdot h(x_i))}$$

The given loss function assumes high values ($\rightarrow 1$) when y_i and $h(x_i)$ are of opposite signs, and takes values closer to 0 when they have the same sign. Thus, this is a valid approximator.

Answer - YES

4.1.4 Part (iv)

$$\frac{1}{1+\exp(-y_i \cdot h(x_i))}$$

This behaves the exact reverse of (3) ($\frac{1}{1+\exp(-y_i \cdot h(x_i))} = 1 - \frac{1}{1+\exp(y_i \cdot h(x_i))}$) and clearly its a wrong approximation.

Answer : NO

4.2 Problem 2

The problem is a replication of Problem 1. It simply involves replacing $h(x_i)$ by $w^T x_i + b$ for all x_i in the training dataset. All the trends remain the same as mentioned above. Hence, one possible optimization problem in this case is:

$$\{w^*, b^*\} = \arg\min_{w,b} \sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i + b)\}$$

4.3 Problem 3

Only **options (a) and (b)** are correct. This is because for negative values of $yf(x)$ (i.e. the prediction is different from the true label), the loss $\ell(yf(x))$ must be high. On the other hand, when the predicted and true labels have the same sign, then the loss must be low. Both (a) and (b) satisfy this, while (d) and (e) are complementary to them, and so they don't work. Finally, (c) has a low loss value in both cases, which is not desirable either. These trends are similar to what we explained in problems 1 and 2.

4.4 Problem 4

4.4.1 Part (i)

In our Binary classification Task, the Training set is *imbalanced*, i.e, D_{Train} has $\eta_{Train} = 90\%$ examples of class $+1$, and 10% of class -1. Here we are working with family of constant models $\mathcal{H} = \{+1, -1\}$

We only have access to the Training Set, so to minimize the Training error, we choose $\hat{h}$ to be the *Majority Estimator*, i.e, the Dominant class. So $\hat{h}(x) = +1$

4.4.2 Part (ii)

$$Error(h^*) - Error(\hat{h})$$

If the test set is well balanced (i.e. positive and negative labels both form 50% of the data), then any constant classifier will have a test error of 50%. Since h^* and $\widehat{h}$ are both chosen from the constant hypothesis class, so their error will be equal, giving the result Error(h^*) − Error$(\widehat{h}) = 0$.

4.4.3 Generalized Problem

Suppose the composition of $\mathcal{D}_{\text{Train}}$ is η_{Train} for class $+1$, and the distribution for $\mathcal{D}_{\text{Test}}$ is η_{Test} for class $+1$. Now, we train the constant classifier $\widehat{h}$ on $\mathcal{D}_{\text{Train}}$, and let h^* be the best possible constant classifier. Then $\widehat{h} = \begin{cases} +1 & \text{if } \eta_{\text{Train}} \geq 0.5 \\ -1 & \text{if } \eta_{\text{Train}} < 0.5 \end{cases}$ and $h^* = \begin{cases} +1 & \text{if } \eta_{\text{Test}} \geq 0.5 \\ -1 & \text{if } \eta_{\text{Test}} < 0.5 \end{cases}$.

Then $\text{Error}(h^*) = \min(\eta_{\text{Test}}, 1 - \eta_{\text{Test}})$, and $\text{Error}(\widehat{h}) = \begin{cases} 1 - \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} \geq 0.5 \\ \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} < 0.5 \end{cases}$.

Thus, the final result is

$$\text{Error}(h^*) - \text{Error}(\widehat{h}) = \begin{cases} 2 \times \eta_{\text{Test}} - 1 & \text{if } \eta_{\text{Train}} \geq 0.5, \eta_{\text{Test}} < 0.5 \\ 1 - 2 \times \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} < 0.5, \eta_{\text{Test}} \geq 0.5 \\ 0 & \text{if } (\eta_{\text{Train}} - 0.5) \times (\eta_{\text{Test}} - 0.5) \geq 0 \end{cases}$$

4.5 Problem 5

Weighting is introduced to handle bias from the training dataset. In particular, if we simply minimize the unweighted loss function, then there is a bias towards the majority class of training dataset, assuming the situation when the training set is highly imbalanced, while the test set is nearly balanced. In such a situation, a good approach is to give more weight to the loss from the minority class. Thus the r_i's need to be divided in the opposite ratio, i.e. 0.9 for labels -1 and 0.1 for labels $+1$.

An important point to keep in mind is that this weighing can only be done is we know about the distribution of the test set to some extent. In practice, this is generally not the case, as the test set is only revealed at the end of training. Most ML models work under the assumption that the validation set will be close enough to the test set, to generate reasonable models.

4.6 Problem 6

4.6.1 Part (i)

The table alongside shows the behavior of the cross-entropy loss for the four different situations, i.e., for each of the two values that y can take, whether $h(x)$ is greater than the threshold or not. Observing the loss behavior in each case, we can say that it is indeed a valid loss function.

y	h(x)	Loss
+1	$> 0.5 (\rightarrow 1)$	$\rightarrow 0$
0	$> 0.5 (\rightarrow 1)$	$\rightarrow \infty$
+1	$\leq 0.5 (\rightarrow 0)$	$\rightarrow \infty$
0	$0.5 (\rightarrow 0)$	$\rightarrow 0$

4.6.2 Part (ii)

For the training loss to be 0, when $y_i = 1$, we need $h(x_i)$ also as 1, and similarly, when $y_i = 0$, we need $h(x_i) = 0$. However, for the sigmoid function to attain 0 or 1, we need the argument $w^T x$ to become either $-\infty$ or ∞. Since x is finite (unit-norm), this is only possible when $||w|| = \infty$.

4.6.3 Part (iii)

If $h(x_i) = \dfrac{1}{1+e^{w^T x_i}}$, then $1 - h(x_i)$ is the same as the original model. Thus, we simply need to make the change $h(x_i) \longrightarrow 1 - h(x_i)$ in the loss function. This gives our new cross-entropy loss as

$$\sum_{(x_i,y_i)\in\mathcal{D}_{\text{Train}}} -\{y_i \log(1 - h(x_i)) + (1 - y_i) \log h(x_i)\}$$

4.7 Problem 7

- if $\tau = 0$, model always predicts +1
- if $\tau = 1$, model always predict 0
- if $\tau = 0.5$, the model predicts a mixture of $\{0, 1\}$, but the accuracy is not optimal.

The optimal value of τ can be determined as:

$$\tau^* = argmin_{\tau\in[0,1]} \sum_{(x_i,y_i)\in\mathcal{D}_{\text{Test}}} (Sign(h(x_i) - \tau) \neq y_i)$$

4.8 Problem 8

1. $f(x) = w_1 * x_1 + w_2 * x_2$ Linear in x and linear in w
   First property
   $f(\overline{w}, \overline{x}) = w_1x_1 + w_2x_2$
   $f(\overline{w}, \overline{x} + \overline{y}) = w_1(x_1 + y_1) + w_2(x_2 + y_2)$
   $f(\overline{w}, \overline{x} + \overline{y}) = w_1x_1 + w_1y_1 + w_2x_2 + w_2y_2$
   $f(\overline{w}, \overline{x} + \overline{y}) = f(\overline{w}, \overline{x}) + f(\overline{w}, \overline{y})$
   Second property
   $f(\overline{w}, \alpha\overline{x}) = w_1\alpha x_1 + w_2\alpha x_2$
   $f(\overline{w}, \alpha\overline{x}) = \alpha(w_1x_1 + w_2x_2)$
   $f(\overline{w}, \alpha\overline{x}) = \alpha f(\overline{w}, \overline{x})$
   Similarly, it can be shown for $\overline{w}$

2. $f(x) = w_1x_1^2 + w_2x_2^3$
   Linear in w and non-linear in x

3. $f(x) = w_1 \ln x_1 + w_2e^{x_2}$
   Linear in w and non-linear in x

4. $f(x) = x_1 \ln w_1 + x_2 e^{w_2}$
   Linear in x and non-linear in w

5. $f(x) = w^T x \quad w, x \in \mathbb{R}^d$
   Linear in both w and x

6. $f(x) = w^T x + b$
   If we introduce another notation $x_0 = 1$ such that $\overline{x} = [x_0 x_1 ... x_d]^T$ and call $w_0 = b$ then the above equation becomes $f(x) = [w_0 w_1 ... w_d][x_0 x_1 ... x_d]^T$
   $f(x) = w_{new}^T x_{new}$ is linear now in w_{new} and x_{new}

4.9 Problem 9

L2 loss is defined as:

$$E = \Sigma_{i=1}^{N} (y_i - b - w_1 x_1)^2$$

Taking its gradient with respect to b:

$$\frac{\partial E}{\partial b} = \Sigma_{i=1}^{N} 2 * (y_i - b - w_1 x_1)(-1)$$

Equating this to 0, we get

$$\begin{aligned} & \Sigma_{i=1}^{N} (y_i - b - w_1 x_1) = 0 \\ \implies & n\overline{y} - nb - n\overline{x} w_1 = 0 \\ \implies & b + \overline{x} w_1 = \overline{y} \\ \implies & [\overline{x}] b + [\overline{x}^2] w_1 = \overline{x}\ \overline{y} \end{aligned} \tag{1}$$

Taking L2 Loss gradient with respect to w_1:

$$\frac{\partial f}{\partial w_1} = \Sigma_{i=1}^{N} (y_i - b - w_1 x_1)(-x_i)$$

Equating this to 0, we get

$$\begin{aligned} & \Sigma x_i y_i - b(\Sigma x_i) - w_1(\Sigma x_i^2) = 0 \\ \implies & \Sigma x_i y_i - bn\overline{x} - w_1(\Sigma x_i^2) = 0 \\ \implies & [n\overline{x}] b + [\Sigma x_i^2] w_1 = \Sigma x_i y_i \\ \implies & [\overline{x}] b + [\frac{\Sigma x_i^2}{n}] w_1 = \frac{\Sigma x_i y_i}{n} \end{aligned} \tag{2}$$

Subtracting Equation (2) from Equation (1) gives

$$w_1 = \frac{\frac{\Sigma x_i y_i}{n} - \overline{x}\ \overline{y}}{\frac{\Sigma x_i^2}{n} - \overline{x}^2}$$

$$b = \overline{y} - \overline{x}w_1$$

By the way equation of line was: $y = b + w_1 x$

4.10 Problem 10

We will **design** the **X** matrix in the following way with each x_i vector being the data sample vector,

$$X = \begin{bmatrix} \mathrm{x}_1^T & \mathrm{x}_2^T & \mathrm{x}_3^T & . & . & . & . & . & \mathrm{x}_n^T \end{bmatrix}^T$$

Now with matrix multiplication:

$$Xw = \begin{bmatrix} \mathrm{x}_1^T w & \mathrm{x}_2^T w & \mathrm{x}_3^T w & . & . & . & . & . & \mathrm{x}_n^T w \end{bmatrix}^T$$

$$Y = \begin{bmatrix} \mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 & . & . & . & . & . & \mathrm{y}_n \end{bmatrix}^T$$

Now,

$$||Xw - Y||_2^2 = || \begin{bmatrix} \mathrm{x}_1^T w - y_1 & \mathrm{x}_2^T w - y_2 & \mathrm{x}_3^T w - y_3 & . & . & . & . & . & \mathrm{x}_n^T w - y_n \end{bmatrix}^T ||^2$$

$$\implies ||Xw - Y||_2^2 = \Sigma (X_i^T w - y_i)^2$$

Now,

$$L(w) = ||Xw - Y||_2^2 = (Xw - Y)^T (Xw - Y)$$

$$\implies L(w) = w^T X^T X w - w^T X^T Y - Y^T X W + Y^T Y$$

Now derivative of above equation for loss in terms of **w** gives,

$$\nabla_w [w^T X^T X w - w^T X^T Y - Y^T X W + Y^T Y] = 0$$

$$\implies 2X^T X w - X^T Y - X^T Y + 0 = 0$$

$$\implies X^T X w = X^T Y$$

4.11 Problem 11

So X is the *design matrix*, of training inputs stacked as rows and X.shape = [N, d]
We can prove that if X is a full column rank Matrix, then X^TX is invertible.

- X is full column rank, means all its coulmns are Linearly Independent, $rank(X) = d$, So its Null Space is $\{\mathbf{0}\}$, or simply $Xv = 0 \iff v = 0$
- X^TX is a dxd matrix, to show that X^TX is invertible, we need to show that its *Null Space* $\mathcal{N}(X)$ is $\{\mathbf{0}\}$/ dimension 0.
- So say there exist a $\mathbf{v}$, so that $X^TX\mathbf{v} = 0 \implies \mathbf{v^T}X^TX\mathbf{v} = 0 = (X\mathbf{v})^T(X\mathbf{v}) = ||X\mathbf{v}||_2^2$, So it implies $X\mathbf{v} = \mathbf{0}$, which inturn implies $\mathbf{v} = 0$
- Hence Nullspace of X^TX is $\{\mathbf{0}\}$, so it is full row/column rank. Hence X^TX is invertible.

4.12 Problem 12

M is a Positive Definite Matrix $\iff \mathbf{v^T}M\mathbf{v} > 0 \; \forall \mathbf{v} \in \mathrm{R^n}\backslash\{\mathbf{0}\}$
Determinant of Positive Definite matrix is Non Zero, So its determinant exists.

We need to show $(X^TX + \lambda I)$ is Positive Definite, so consider an arbitary $\mathbf{v} \neq 0$
$\mathrm{v}^T(X^TX + \lambda I)v = v^TX^TXv + \lambda v^Tv = (Xv)^TXv + \lambda v^Tv = ||Xv||_2^2 + \lambda||v||_2^2 > 0$
Hence, $X^TX + \lambda I$ is a Positive Definite Matrix

4.13 Problem 13

Linear regression problem can be modelled as $Y_i = w^Tx_i + \epsilon_i$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
So $Y_i \sim \mathcal{N}(f(x_i), \sigma^2)$, where $f(x_i) = w^Tx_i$

The given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

We need to maximize the Likelihood of Observing the generated y_i data with our Model, and then try to Maximise the Likelihood. $L(w) = P(y|x)$

$$P(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - f(x_i))^2}{2\sigma^2}}$$

So the Likelihood, $L(w) = \prod_i P(y_i|x_i)$, we assume all samples are independently drawn

$$L(w) \propto \prod_i e^{-\frac{(y_i - f(x_i))^2}{2\sigma^2}} = e^{\frac{-1}{2\sigma^2}\sum_i (y_i - f(x_i))^2}$$

So maximizing Likelihood means minimizing $-log(L(.))$ Negative log-likelihood, so

$$argmax_w L(w) \iff argmin_w \sum_i (y_i - f(x_i))^2 = \mathbf{L2}\ loss$$

Hence we can see that Maximizing the likelihood is equivalent to minimizing the L2 loss for the regression problem.

AIML - CS 337

Lecture 9: Convexity, Trainability and Stability

4th september, 2022

Lecturer: Abir De *Scribe: Group 17 and Group 18*

1 Definitions

1.1 Trainability

We define trainabilty in order to emphasize on uses of regularization. Being said that, this has not been directly discussed in class. Trainable loss functions requires us to get a bound on expected value of test loss on any test data derived from same distribution as of train data.

Let A be the algorithm, trained on set S(which is and i.i.d from a distribution P), D denotes the test set derived from same distribution.
For all $D, \epsilon > 0$ there exists M such that for all $m > M$, the following condition holds true

$$\underset{s \in P^m}{E}[L_D(A(s))] \leq \min_w L_D(w) + \epsilon$$

1.2 Stability

Stability is a measure of how the algorithm responds on adding a new data point to the training set. Idea is that we should not give excessive priority to a single data point, more importantly in the cases where the algorithm has already been trained on enough points. This property is also closely related to over-fitting(generating a complex model which reduces loss on train-data, but not on test-data) on data. In some cases stability is also viewed in the terms of drift in model if one of the points is tampered at random. We are going to consider the following metric to view stability.

$$\|A(S \cup x) - A(S)\| \;\; where\; S \in D^m, x \in D$$

We aim at showing that this quantity is bounded and the bound becomes tighter on increasing m tending to 0 for large values of m. and we can increasingly tightly bound this value on expectation.

It is not entirely true that more stable algorithms are better than the less stable ones, consider constant output algorithms for example. A useful algorithm should find a hypothesis that on one hand fits the training set and on the other hand does not over-fit (low structural risk)

1.3 Useful properties of loss functions

There are lot of properties that a loss function can satisfy which can probably boost performance of algorithm depending on problem requirements. The following properties are found in most loss functions and are going to be used in the proof that comes up in the next section.

1.3.1 Convexity

Convexity of a function helps us to reach its minimum using gradient descent. Convexity of differentiable functions from $R \mapsto R$ is defined by using double derivatives whereas for functions from R^d, we define using eigen-values of $\frac{\partial^2 f}{\partial w^2}$.

Condition for a function to be convex given $\frac{\partial^2 f}{\partial w^2}$ exits is that its eigen-values must be non-negative.

Que:- Find all eigen values of $XX^T,\ X \in m \times 1$
Solution:-
$Rank(XX^T) \leq Rank(X) \leq 1$
$Rank(XX^T) \leq 1$
$XX^T\ has\ all\ real\ eigen\ values \geq 0$
{$||x||^2$, 0 {with geometric multiplicity m-1}} (observe that sum of eigen-values is $\|x\|^2$)

1.3.2 Lipschitzness

Let $C \in R^d$, A function $f : R^d \to R^k$ is p-Lipschitz over C if for every $w_1, w_2 \in C$ we have

$$\|f(w_1) - f(w_2)\| \leq p||w_1 - w_2||.$$

Note that Lipschitzness does not guarantee differentiability but differentiable function with bounded derivative is Lipschitz.

1.3.3 Smoothness

A differentiable function is β - smooth if its gradient is β lipschitz.

$$\nabla f(w) = (\frac{\partial f(w)}{\partial w_1}, ..., \frac{\partial f(w)}{\partial w_m})$$
$$\|\nabla f(v) - \nabla f(w)\| \leq \beta \|v - w\|$$

Note: It can be proven that if f is β-smooth, following condition holds

$$f(v) \leq f(w) + \langle \nabla f(w), v - w \rangle + \frac{\beta}{2}\|v - w\|^2$$

2 Convexity of Loss Function

We are given a dataset

$$D = \{(x_i, y_i)\}_{i \in [N]}$$

and we are trying to find a model such that

$$\mathrm{X} \xrightarrow{\mathrm{W}} \mathrm{Y}$$

$$w^T x \approx y$$

If we change x by some value, how much W changes.
If loss function is convex, perturbation in W is also small.

$$l(w^T, x, y) \rightarrow convex$$

Examples of convex functions

- $\sum (y - w^T x)^2$
- $max(0, 1 - y w^T x)$
- $-log(1 + e^{-w^T x})$

Example of Non-convex

- $(y - (w^T x)^2 + (w^T x)^3)^2$ is not convex $w.r.t.\ w$

Deep learning systems have nice tools that finds local minimum of to search for local minima. Convexity is the requirement of single minima i.e. local minima is the global minima.
If loss function is $l(w^T, x, y)$

$$\frac{\mathrm{d}}{\mathrm{d}w}(l(w^T, x, y)) \in \mathbb{R}^{d \times 1}$$

is a vector

$$\frac{\mathrm{d}^2}{\mathrm{d}w^2}(l(w^T, x, y)) \in \mathbb{R}^{d \times d}$$

is a **Hessian** matrix. We require that all its $eigen - values \geq 0$ for $l(w^T, x, y)$ to be convex .
For example
example 1
$l(w) = \sum (y - w^T x)^2$

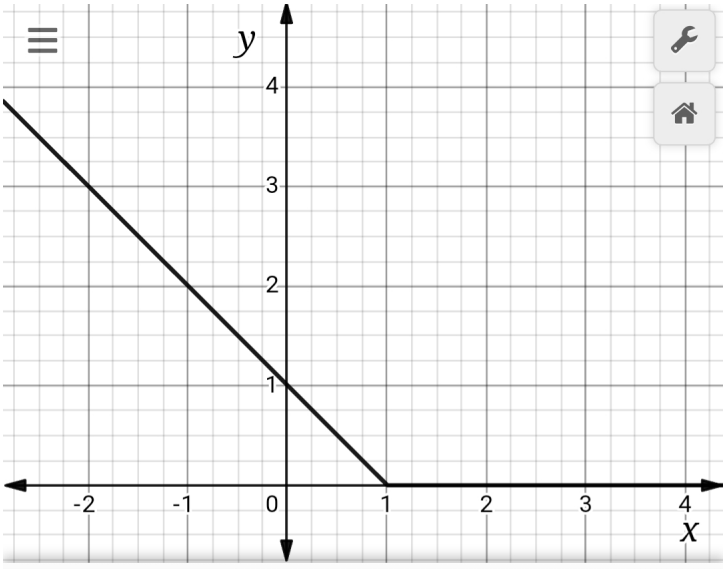
$$\frac{\mathrm{d}}{\mathrm{d}w}(l) = \sum 2(y - w^T x) w$$

$$\frac{\mathrm{d}^2}{\mathrm{d}w^2}(l) = 2XX^T$$

whose eigenvalues are positive (non negative)

Note $\frac{\mathrm{d}^2}{\mathrm{d}w^2}(w^TAw) = A$
XX^T is a rank 1 matrix which implies it has one non zero eigenvalue
Now, $trace(XX^T) =$ sum of eigenvalues$\rightarrow$ eigen values are $\{||x||^2, 0, 0...\}$
Hence, $l(w) = \sum (y - w^Tx)^2$ is a convex function.
example 2
$l(w) = max(0, 1 - yw^Tx)$
It is the maximum of a convex function so it is a convex function.
Or, $\frac{\partial^2}{\partial w^2}$ is zero at almost all points, almost everywhere the $eigen - values$ are zero.

The plot of $max(0, 1 - a)$ where $a = yw^Tx$



3 Regularization and Stability

3.1 Definition

Most RLM algorithms are in the form given below. We have two terms, one defining fitting of model to given examples and the other representing complexity of the model.

$$\arg\min_w \; L_s(w) + R(w)$$

In most scenarios, regularization is used as a tool to reduce over-fitting and provide stability, more on which will be discussed in the section named Regularisation and Stability.

3.2 Uses

There are many uses of introducing regularization but most prominent ones are the following :

- Balance between fitting and stability
- Only some of the functions in the convex smooth domain can be proved to be trainable. But with correct regularization, all smooth convex functions can be shown to be trainable.

Hence, RLM can be used as a general learning rule for convex smooth learning problems.
You may have already intuitively noticed this in previous classes where $XX^T + \lambda$I ($\lambda > 0$) is shown to always have an inverse .

Tikhonov Regularization : $R(w) := \lambda\|w\|^2$
Adding regularizer terms leads to a **strictly convex** loss function.

$$l_\lambda(w, x, y) = \sum (y - w^T x)^2 + \lambda||w||^2$$

$$l_\lambda(w, x, y) = l(w, x, y) + \lambda||w||^2$$

3.3 Assumptions regarding loss function

1. $l_\lambda(w, x, y) = l(w, x, y) + \lambda||w||^2$ is strictly convex, since $\frac{\mathrm{d}^2}{\mathrm{d}w^2}(l_\lambda(w, x, y)) > 0$
2. $\left\|\frac{\mathrm{d}l(w,x,y)}{\mathrm{d}w}\right\|$ is **bounded** , i.e $\left\|\frac{\mathrm{d}l(w,x,y)}{\mathrm{d}w}\right\| \leq$ B
3. $\|W(DUK) - W(D)\| = \mathcal{O}(\frac{1}{|D|})$

Let's prove the assumptions,

$$\frac{\mathrm{d}^2}{\mathrm{d}w^2}(l_\lambda(w, x, y)) = \frac{\mathrm{d}^2}{\mathrm{d}w^2}(l(w, x, y)) + 2\lambda\mathbb{I}$$

whose eigenvalues are **strictly positive**. Since $\lambda > 0$, $2\lambda\mathbb{I}$ has strictly positive eigenvalues, while $l(w, x, y)$ has non negative eigenvalue as shown earlier. Hence $l_\lambda(w, x, y))$ is strictly convex.

$$F(W, S) = \sum_{i=1}^{|S|} l_\lambda(W, x_i, y_i)$$

$$W(S) = \min_W \sum_{i=1}^{|S|} l_\lambda(W, x_i, y_i)$$

First of all , note that $F(W(S \cup K), S) - F(W(S), S) > 0$ because W(S) is the value of W that minimizes F(W,S) .

We want to bound $F(W(S \cup K), S) - F(W(S), S)$. A simple bound on this difference using the notion of Lipschitz continuity is

$$F(W(S \cup K), S) - F(W(S), S) \leq B_\lambda |S| \, \|W(S \cup K) - W(S)\|$$

Now we attempt to form a tighter bound on $F(W(S \cup K), S) - F(W(S), S)$
We can split $F(W(S), S)$ as $F(W(S), S \cup K) - F(W(S), K)$ Note that K is a single data point .

$$\begin{aligned} F(W(S \cup K), S) - F(W(S), S) = &(F(W(S \cup K), S \cup K) - F(W(S), S \cup K)) \\ &+ (F(W(S), K) - F(W(S \cup K), K)) \end{aligned}$$

$F(W(S \cup K), S \cup K) - F(W(S), S \cup K) \leq 0$ because $W(S \cup K)$ is the optimal value of W which minimizes $F(W, S \cup K)$.
F is Lipschitz continuous . Hence ,

$$\begin{aligned} F(W(S), K) - F(W(S \cup K), K) &= (W(S) - W(S \cup K))^T \left. \frac{\partial F}{\partial W} \right|_{W'} \\ &\leq B \, \|W(S \cup K) - W(S)\| \end{aligned}$$

Here we used the Assumption 2 regarding loss function that the first derivative is bounded by some constant B . Hence , we proved that

$$F(W(S \cup K), S) - F(W(S), S) \leq B \, \|W(S \cup K) - W(S)\| \tag{1}$$

Now we try to find a lower bound for the expression $F(W(S \cup K), S) - F(W(S), S)$ using the convexity of the loss function .

First we apply Taylor Series Expansion to $F(W(S \cup K), S)$

$$\begin{aligned} F(W(S \cup K), S) &= F(W(S), S) + (W(S \cup K) - W(S))^T \left. \frac{\partial F}{\partial W} \right|_{W=W(S)} \\ &\quad + \frac{1}{2!}(W(S \cup K) - W(S))^T H(W)(W(S \cup K) - W(S)) \\ &= F(W(S), S) + \frac{1}{2}(W(S \cup K) - W(S))^T H(W)(W(S \cup K) - W(S)) \end{aligned} \tag{2}$$

Where $H(W) = \frac{\partial^2 F}{\partial^2 W}$ is the Hessian Matrix corresponding to F at some W^* lying on the line between W(S) and $W(S \cup K)$. Here we used the idea that $\frac{\partial F}{\partial W}|_{W=W(S)} = 0$ because W(S) is the optimum value which minimizes F(W,S)

Note from Assumption 1 we have that all eigen values of the double derivative of $l(w, x, y)$(loss function with regularization) w.r.t. W are positive. In other words , all eigen values of $\frac{\partial^2 l(W,x,y)}{\partial^2 W}$ are positive .

$$\begin{aligned} F(W, S) &= \sum_{i=1}^{|S|} l_\lambda(W, x_i, y_i) \\ &= \sum_{i=1}^{|S|} l(W, x_i, y_i) + \sum_{i=1}^{|S|} \lambda \, \|W\|^2 = \sum_{i=1}^{|S|} l(W, x_i, y_i) + \lambda |S| \, \|W\|^2 \end{aligned}$$

$$
\begin{aligned}
H(W) &= \frac{\partial^2 F(W,S)}{\partial^2 W} \\
&= \frac{\partial^2 \sum_{i=1}^{|S|} l_\lambda(W, x_i, y_i)}{\partial^2 W} \\
&= \sum_{i=1}^{|S|} \frac{\partial^2 l(W, x_i, y_i)}{\partial^2 W} + 2\lambda|S|I \\
v^T H(W) v &= \sum_{i=1}^{|S|} v^T \frac{\partial^2 l(W, x_i, y_i)}{\partial^2 W} v + 2\lambda|S| v^T v \\
&\geq 2\lambda|S| \left\|v\right\|^2
\end{aligned}
$$

Apply this to (2) , we get

$$
F(W(S \cup K), S) - F(W(S), S) \geq ||W(S \cup K) - W(S)||^2 \lambda |S| \tag{3}
$$

From 1 and 3 , we have

$$
||W(S \cup K) - W(S)||^2 \lambda |S| \leq F(W(S \cup K), S) - F(W(S), S) \leq B \left\|W(S \cup K) - W(S)\right\|
$$

$$
\begin{aligned}
||W(S \cup K) - W(S)||^2 \lambda |S| &\leq B \left\|W(S \cup K) - W(S)\right\| \\
||W(S \cup K) - W(S)|| &\leq \frac{B}{\lambda |S|}
\end{aligned}
$$

Thus we have proved Assumption 3 which states that

$$
\|W(S \cup K) - W(S)\| = \mathcal{O}(\frac{1}{|S|}) \tag{4}
$$

3.4 Regularization and Stability

Stability, as defined earlier is measured by the quantity

$$
\|A(S \cup x) - A(S)\|
$$

where x is new data point added to S.
proving the stability requires us to prove that $\underset{S \in D^m}{\|} A(S \cup x) - A(S)\|$ is bounded and the bound becomes tighter as m increases, converging to 0. We prove this result considering bounds of

$$
E = L(A, S(A \cup x)) - L(A, S(A))
$$

Note that this quantity is greater than 0 for a good algorithm and loss function i.e $E > 0$.

3.4.1 Proving upper bound

Using the Smoothness/ lipschitz

$$L(A, S(A \cup x)) - L(A, S(A)) < B(m)||S(A \cup m) - S(A)||$$

This sure is a bound but not a good one as the factor m destroys the purpose of the bound, which will be clear by the end of this proof .

$$L(A, S(A \cup x)) - L(A, S(A)) = L(A \cup x, S(A \cup x)) - L(x, S(A \cup x)) - (L(A \cup x, S(A)) - L(x, S(A))) \tag{5}$$

$$let\ G(M) = L(A \cup x, S(M)) - L(x, S(M)) \tag{6}$$

Observe that eq 1 can be rewritten as

$$G(A \cup x) - G(A) \tag{7}$$

using smoothness of eq 3 we can write

$$G(S(A \cup x)) - G(S(A)) \leq B||S(A \cup x) - S(x)|| \tag{8}$$

3.4.2 Proving lower bound

This proof uses a non-trivial result from taylor series which will be stated as lemma.

Lemma-1:-

$$f(h_1) \geq f(h_2) + < \nabla f(h_1, h_1 - h_2) > + \frac{1}{2}(h_1 - h_2)^T \nabla^2 f(h^{'}, h_1 - h_2^{'})$$

for some $h^{'}$.

Apply lemma-1 on eq 1, we get

$$\begin{aligned} L(A, S(A \cup x)) - L(A, S(A)) \geq & < \nabla L(A, S(A)), S(A \cup x) - S(A) > \\ & + \frac{1}{2}(S(A \cup x) - S(A))(\nabla^2 l(A, h^{'}) * m)(S(A \cup x) - S(A)) \end{aligned} \tag{9}$$

Observe that $\nabla L(A, S(A)) = 0$, by definition

$$L(A, S(A \cup x)) - L(A, S(A)) \geq \frac{m}{2}(S(A \cup x) - S(A))^T \nabla^2 l(A, h^{'})(S(A \cup x) - S(A)) \tag{10}$$

considering the case of Tiknow regularisation $eig(\nabla^2 L(A, h)) \geq \lambda$ giving

$$L(A, S(A \cup x)) - L(A, S(A)) \geq \frac{m}{2}\lambda||S(A \cup x) - S(A)||^2 \tag{11}$$

using Eq 7 and 4

$$B||S(A \cup x) - S(A)|| \geq \frac{\lambda}{2}\boldsymbol{m}||S(A \cup x) - S(A)||^2 \tag{12}$$

$$\frac{2B}{\lambda m} \geq ||S(A \cup x) - S(A)|| \tag{13}$$

Question:-
On similar lines, derive a bound for following case $||S(A^{'}) - S(A)||$ where $A^{'}$ is set with one case (z_i) tampered to $(z_i^{'})$.

HINT

$$||\overline{S}(A^{'} \cup Z_i) + S(A^{'}) + S(A \cup Z_i^{'}) - S(A)|| = ||S(A^{'}) - S(A)||$$

4 Non Linear Regression

In nonlinear regression, we modeled a given data-set by using a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.
That is why nonlinear regression shows association using a curve, making it nonlinear in the parameter.
In contrast to linear regression, we cannot use the ordinary least squares method to fit the data and estimation of the parameters are also not easy.

Let's think about the solution

$$w = (\lambda I + \sum_{i \epsilon D} X_i X^T)^{-1} \sum_{i \epsilon D} X_i Y_i \tag{14}$$

In linear case, we have a line graph. So, what do we have to change in our approach to fit the non-linear regression?

What is that $X_i X^T$ indicates?

$$X_i X^T = ||X_i||^2$$

Suppose we have two vector [1 2 3]^T and [3 6 9]^T.
Are they similar?
Although its directions are same but these are not the similar points.

If the given two points are close enough that its covariance is atleast 0.7, then we will consider the data set as comparately similar to linear regression otherwise we have to think of the non-linear regression otherwise we have to think of th non-linear regression.

Instead of term $X_i X^T$, we can put the Kernel $K(X_i X_i)$ to get some sort of good approximation to our non-linear data set.

Example
For infinite dimension vector in infinite dimension space :-
The measure of **similarity** between $\vec{X_i}$ and $\vec{Y_i}$ is approximate to

$$e^{-||X_i - X_j||^2}$$

Instead of $X_i X^T$, take the matrix formed by using the above equation for each entry (i,j).
The above equation $e^{-||X_i - X_j||^2}$ can be represented as the dot product of two vectors.

Dot product isn't the good measure of similarity
Other measures are:-

$$e^{-||X_i - X_j||^2}$$
$$\frac{1}{1 + ||X_i - X_j||^2}$$

We can replace $X^T X$ by K in the loss function to get

$$w = (\lambda I + K)^{-1} \sum_{i \epsilon D} X_i Y_i \tag{15}$$

Here, K is the Kernel matrix

$$e^{-||X_i - X_j||^2} => \phi_i^T \phi_j$$

Doing non-linearity in finite dimension to linearity in infinite dimension space

4.1 Stability

It is easy for linear regression but it is bit difficult for non linear regression.
So, we need to apply **deep learning** for those data sets which don't have linearity even in infinite dimension i.e, $\phi_i^T \phi_j$

Example

$$\{\vec{x_i}, y_i | i \epsilon D\}$$

For the above example, linear regression or kernel both didn't work.

We can use deep learning concepts to find some relation between data sets i.e, Y = F($\vec{X}$).
We can approximate any kind of functions for the given set of data that follows non linearity using functions like ReLU , Sigmoid and Tanh.
For example, try to find a linear model for y = log(x) such that x > 0.

AIML - CS 337

Lecture 10: Introduction to Classification using SVM

8th September 2022

Lecturer: Abir De *Scribe: Group 19, Group 20*

We begin by defining the classification problem and then introduce a linear classifier with a bias. We work on ways to find the appropriate parameters for such a model. For a slightly complex and more realistic classification problem, we progress towards the commonly used *hinge-loss* function for SVM.

1 Introduction

We now introduce the classification problem. Our dataset will be of the form $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^{N}$. Similar to regression, the x_i are data points, usually in $\mathbb{R}^k$ for some $k \in \mathbb{N}$. But unlike regression, y_is are now discrete. For a 2 class problem, the set of possible labels can be {+1,-1}, {Cat,Dog} etc. Even if they are written as {+1,-1}, for the purpose of a classification problem we shall always assume there is no partial order among the labels.

Note- Take a *ranking* task where all objects are given labels from {1,2,3,4,5}. These labels are assumed to posses an implicit partial order. This task can thus use this ordering information and is different from our classification problem. In fact, this is called *ordinal regression*.

Let us consider 2 dimensional data points with class labels from {+1,-1} as shown below

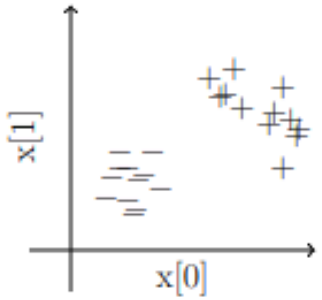


Figure 1: Cluster of positive and negative labels

A few simple classifiers are listed below :

- **Constant model classifier**:- Such a classifier will assign a constant label to every data point. Clearly, if samples from multiple labels are present in the test dataset, it is impossible for a constant model to produce a zero error - even on the training dataset.

- **Unsupervised classifier**:- Unsupervised classifier perform no learning and use heuristics to assign a label to a test data point. For example, an unclassified classifier can find the closest point to the test point in the training dataset and return the class label of this point. Another approach might involve calculating the centroid of all class labels and return the label of the class whose centroid lies closest to the test point. Such models have limited utility.

Supervised learning solutions to the problem follow :

2 Linear Classifier

One way to define a classifier is to introduce a parameter $w \in \mathbb{R}^k$. Then,

$$\mathbf{w}^T x > 0 \;\Rightarrow\; y = +1$$

$$\mathbf{w}^T x < 0 \;\Rightarrow\; y = -1$$

A classifier of this form will fail to provide a significant margin between the classes. To counter this issue, we change the classification rules to the following :

$$\mathbf{w}^T x > +1 \;\Rightarrow\; y = +1$$

$$\mathbf{w}^T x < -1 \;\Rightarrow\; y = -1$$

You might ask how one came up with the magic numbers +1 and -1? These numbers don't matter as w can be scaled appropriately to get the exact same classifier whatever $\{-\delta, +\delta\}$ pair one choses to set the margins.

A major problem with the current classifier is that the origin is never classified to any class label. This is unnecessarily restrictive. For example, consider the following dataset which cannot be seperated by a line passing through the origin but is clearly seperable by a line that does not.

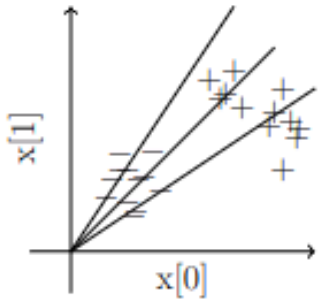


Figure 2: Angularly inseperable clusters

Note- Angularly seperable clusters can be seperated using the current classification model.

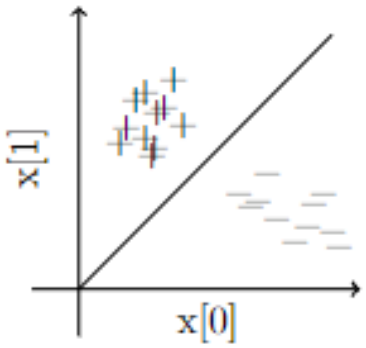


Figure 3: Angularly separated clusters

Solving this issue is simple, clearly we are missing a bias term in our rules :

$$\mathbf{w}^T x + \mathbf{b} > 1 \;\Rightarrow\; y = +1$$

$$\mathbf{w}^T x + \mathbf{b} < -1 \;\Rightarrow\; y = -1$$

Now, our model has 2 parameters to train, $w, b \in \mathbb{R}^k$. We shall discuss methods of finding a working pair of w, b which are capable of correctly classifying a training dataset.

A proposed solution is to find 2 points that are closest to one another and lie in different classes and use the perpendicular bisector between this pair as the seperation boundry. Clearly this is a heuristic and a intrinsically unsupervised approach and a counter example can be easily found -

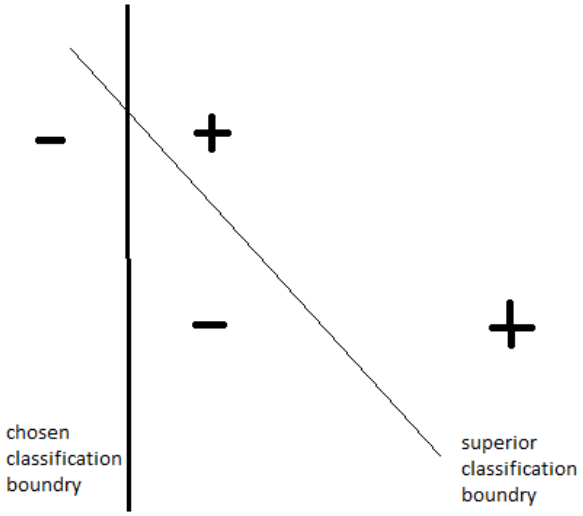


Figure 4: The perpendicular bisector approach does not work.

3 Convex Optimization

Clearly there can be multiple w, b such that the classifier

$$\mathbf{w}^T x + \mathbf{b} > +1 \;\Rightarrow\; y = +1$$
$$\mathbf{w}^T x + \mathbf{b} < -1 \;\Rightarrow\; y = -1$$

classifies the training data reasonably well.

The pair of conditions can be rewritten as $y_i(\mathbf{w}^T x_i + \mathbf{b}) > 1$ if the class labels are fixed to be $\{+1, -1\}$.

Our aim is to find a convex objective function $\mathbf{c}$ for which one can perform :

$$\{\mathbf{w}^*, \mathbf{b}^*\} = argmin\ \mathbf{c}(\mathbf{w}, \mathbf{b}) \quad : \quad \forall i \in \mathbb{D}\ \ y_i(\mathbf{w}^{*T} x_i + \mathbf{b}^*) > 1$$

which captures the gist of the problem we are trying to solve. A few proposals follow :

- Let the perpendicular distance of the i^{th} positive point from a line $\{\mathbf{w}, \mathbf{b}\}$ be $d_i^+(\mathbf{w}, \mathbf{b})$ and for the j^{th} negative point be $d_j^-(\mathbf{w}, \mathbf{b})$

  $$\mathbf{c}(\mathbf{w}, \mathbf{b}) := min_{i \in n_+}\{d_i^+(\mathbf{w}, \mathbf{b})\} \cdot min_{j \in n_-}\{d_j^-(\mathbf{w}, \mathbf{b})\}$$

  Our optimization problem would have been

  $$\{\mathbf{w}^*, \mathbf{b}^*\} = argmax\ \mathbf{c}(\mathbf{w}, \mathbf{b})$$

  We try to maximize this objective function. It is equivalent to minimizing its negative (:

  This objective function tries to increase the distance of both clusters from our classification line. Convince yourself that a sum would not achieve this 2 fold minimization.

  This objective function is not easily optimized.

- Define

  $$\mathbf{c}(\mathbf{w}, \mathbf{b}) = min(min_{i \in n_+}\{d_i^+(\mathbf{w}, \mathbf{b})\}, min_{j \in n_-}\{d_j^-(\mathbf{w}, \mathbf{b})\})$$

  Or

  $$\mathbf{c}(\mathbf{w}, \mathbf{b}) = min_{\forall i \in \mathbb{D}}\{d_i(\mathbf{w}, \mathbf{b})\}$$

  Here, we are interested in maximizing the minimum distance across all points. If this quantity is large, then all other distances will be large.
  In fact, perpendicular distance of the i^{th} point from the line $\mathbf{w}^T x + \mathbf{b} = 0$ is given by $d_i = \frac{|\mathbf{w}^T x_i + \mathbf{b}|}{||\mathbf{w}||}$.
  Since this is a constrained minimization (maximization), we have $y_i(\mathbf{w}^T x_i + \mathbf{b}) > 1$. Thus,

  $$d_i = \frac{|\mathbf{w}^T x_i + \mathbf{b}|}{||\mathbf{w}||} > \frac{1}{||\mathbf{w}||}$$

  For increasing d_i over all i, we increase the lower bound on d_i.

- This leads to the famous regularized loss :

  $$\{\mathbf{w}^*, \mathbf{b}^*\} = argmin\ ||\mathbf{w}||^2 \quad : \quad \forall i \in \mathbb{D}\ \ y_i(\mathbf{w}^T x_i + \mathbf{b}) > 1$$

4 A more realistic task

After seeing how to select a good classifier when multiple are available, it is time to look at a more real world problem. Clearly the set of inequalities

$$\forall i \in \mathbb{D} \;\; y_i(\mathbf{w}^{*T} x_i + \mathbf{b}^*) > 1$$

need not always have a $\mathbf{w}, \mathbf{b}$ pair satisfying them. Consider the following figure :

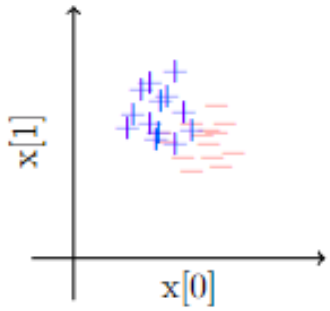


Figure 5: Overlapping clusters. The points cannot be linearly classified.

Dropping constraints, one can simultaneously minimize $\mathbf{c}$ and the missclassification count :

$$|\{i \;:\; |y_i(\mathbf{w}^T x_i + \mathbf{b})| \leq 1\}|$$

However, this doesn't take into consideration the degree of misclassification. The points for which $|y_i(\mathbf{w}^T x_i + \mathbf{b})| \sim 1$ are treated the same as the points for which $|y_i(\mathbf{w}^T x_i + \mathbf{b})| \sim 0$. To counter this, we introduce the hinge loss over $y_i(\mathbf{w}^T x_i + \mathbf{b}) - 1$ and use it to train the SVM.
The final loss function looks like :

$$\mathbf{w}^*, \mathbf{b}^* = argmin_{w,b} \sum_i H(y_i(\mathbf{w}^T x_i + \mathbf{b}) - 1) + \lambda ||w||^2$$

AIML - CS 337

Lecture 11: Soft SVM: Non-Separable Classification

12th September, 2022

Lecturer: Abir De *Scribe: Group 21, Group 22 and TA team*

1 Recap

1.1 The Problem

Our dataset is of the form $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$.
The x_i are data points, usually in $\mathbb{R}^k$ for some $k \in \mathbb{N}$.
The y_is are discrete, and each $y_i \in \{+1, -1\}$.

1.2 Separable Case

In the previous lecture, we assumed that there will always be a hyperplane separating our data, in such a way that, for every point, the classification error will be 0. We saw how we can solve such an instance with the following formulation,

$$\{\boldsymbol{w}^*, b^*\} = ||\boldsymbol{w}||^2, \; s.t \; \forall i \in \mathcal{D}, \; y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) > 1$$

Necessary and Sufficient Condition for separability: Convex Hull corresponding to all the positive points and that corresponding to all the negative points do not intersect.

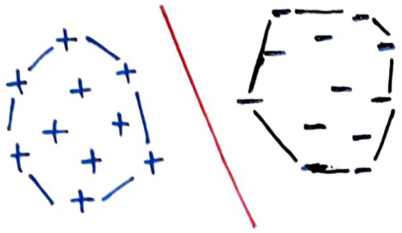


Figure 1: Separable Clusters

1.3 Non-Separable Case

In the case of non-separable instances: we will not be able to find a $\boldsymbol{w}$ such that the condition $y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) > 1$ is satisfied for every point - this is because the convex hulls for positive and negative points intersect, and hence we cannot correctly classify all our points.

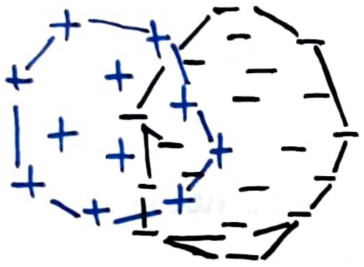


Figure 2: Overlapping Clusters

One way to deal with these is to remove points in the overlap.

Another way is to work with a new formulation where we relax some of our constraints. However, we can't simply handcraft these constraints. We'll have to mathematically model these constraints and try to learn them.

2 Formulation for Non-Separable Instance

We may proceed like this: If the given expression isn't greater than 1 for all (x_k, y_k), there must exist some (x, y) for which it doesn't.

To accommodate this, we may replace the 1 in $y(w^T x + b) > 1 \quad \forall x, y$

by $y(w^T x + b) > 1 - \xi \quad \forall x, y$

We want $\xi > 0$, to be as small as possible. The ξ will be different for different points.

That is, finally, we can write our constraints in the form,

$$y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) \geq 1 - \xi_{(x_i, y_i)}$$

where $\xi_{(x_i,y_i)}$ (known as slack variables) are separate for each (x_i, y_i), which leads us to the following optimization problem,

$$\min_{\boldsymbol{w},\, b,\, \xi_{(\boldsymbol{x_i}, y_i)}} ||\boldsymbol{w}||^2 + c \sum_i \xi_{(x_i, y_i)_+}$$

Note that if $y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) > 1$ i.e., our point is correctly classified, then $1 - y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) < 0$, otherwise $1 - y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) > 0$. So, we can use $\xi_{(x_i,y_i)} = 1 - y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b)$

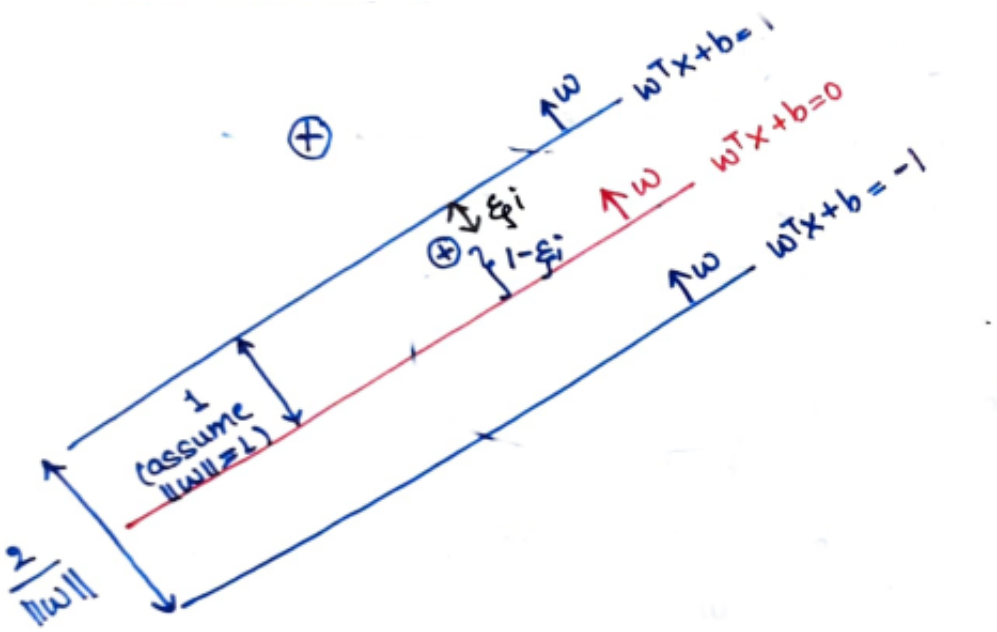
At optimal points, equality is always achieved when minimising $\xi_{(x_i,y_i)}$). This allows us to write our formulation in the following equivalent form,

$$\min_{\boldsymbol{w},\, b} \left(||\boldsymbol{w}||^2 + c \sum_{i \in \mathcal{D}} \max(1 - y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b), 0)\right)$$

If we increase c, the classes would be better separated and might lead to over-fitting.

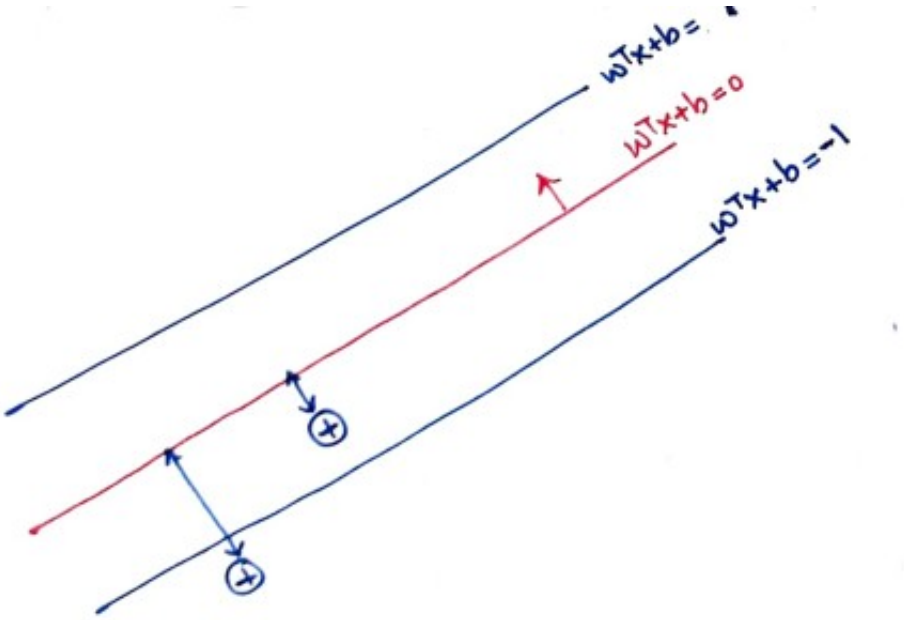
A Geometric Perspective

We have that $y_i(w^T x_i + b) > 1$ is not satisfied for all points of the dataset. If it were satisfied then we would be dealing with the separable case anyway. So, For the case, When $y_i(w^T x_i + b) > 1$ doesn't hold, We are trying to find the minimum $\xi_{(\boldsymbol{x}_i, y_i)}$ s.t. $y_i(w^T x_i + b) \geq 1 - \xi_{(\boldsymbol{x}_i, y_i)}$.
($\xi_{(\boldsymbol{x}_i, y_i)}$ can be simply written as ξ_i)



Note here that for the point that lies above the positive hyperplane,
We have that $y_i(w^T x_i + b) > 1 \Rightarrow \xi_{(\boldsymbol{x}_i, y_i)} = 0$

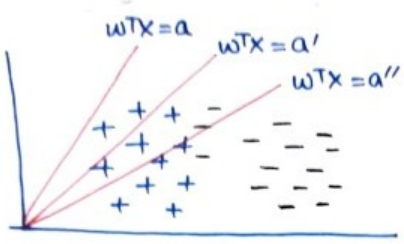
For the other point, which is labelled as $+$ve but is below the said hyperplane, We'll have that $y_i(w^T x_i + b) = 1 - \xi_{(\boldsymbol{x}_i, y_i)}$, in any such case, We'll find that $0 \leq 1 - y_i(w^T x_i + b) \leq 1$
If the case were rather like this :



We'll instead have that $0 \leq 1 - y_i(w^T x_i + b) \geq 1 \Rightarrow \xi_{(\boldsymbol{x}_i, y_i)} \geq 1$

Thus, we may now conclude that the quantity $1 - y_i(w^T x_i + b)$ is negative, When the point is correctly classfied, and positive if its incorrectly specified.

3 Tuning of b



In the separable case we had to use b, because if we forced a hyperplane separating all points to that pass through origin we will always incur some misclassification error.
In non-separable case, we have not regularized b so it is not stable, i.e. change x slightly and b changes significantly. So we try to get rid of b. We are already incurring error due to misclassified points so we allow some error due to b to make our optimisation task easier.

We have two options via preprocessing route.

3.1 Shift the origin

$$\vec{\boldsymbol{X}}_{i\_new} = \vec{\boldsymbol{X}}_{i\_old} - \boldsymbol{E}[\vec{\boldsymbol{X}}]$$

$\boldsymbol{E}[\vec{\boldsymbol{X}}]$: Emperical mean vector incase of data

3.2 Batch Normalization

$$\boldsymbol{X}^{(i)}_{j\_new} = \frac{\boldsymbol{X}^{(i)}_{j\_old} - \hat{\mu}_j}{\hat{\sigma}_j}$$

$\boldsymbol{X}^{(i)}_{j\_new}$: jth feature of ith sample

$\hat{\mu}_j$: Emperical mean over all the samples for jth feature

$\hat{\sigma}_j$: Emperical standard deviation over all samples for jth feature

Batch normalization works because the data along each feature dimension is scaled properly.
Batch normalization also makes the bias tend to 0, training becomes stable and we achieve very good accuracy.
One disadvantage of batch normalization is that mean and standard deviation of each batch may differ significantly if we have lots of data, leading to increase in error. Thus, we have to find the ideal number of data points, which cannot be too high or too low.

4 Dual Formulation

4.1 Convex Optimization

Some background of convex optimization is needed before we move forward with our new formulation of SVM.

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \text{ such that } g(\boldsymbol{w}) \leq \boldsymbol{c} \tag{1}$$

Note that $g(\boldsymbol{w})$ can be a vector and then the inequality would be pointwise ($g_i(w) \leq c_i$)
We can approximate (1) as:

$$\max_{\boldsymbol{\lambda} \geq 0} \min_{\boldsymbol{w}} f(\boldsymbol{w}) + \boldsymbol{\lambda}^\top (g(\boldsymbol{w}) - \boldsymbol{c}) \tag{2}$$

When $f(\boldsymbol{w})$ and $g(\boldsymbol{w})$ both are strictly convex, (1) and (2) are exactly equivalent. This means that either $\boldsymbol{\lambda} = 0$ or $g(\boldsymbol{w}) = \boldsymbol{c}$. This is known as Slater's condition.

4.2 Objective Function

SVM problem at hand:

$$\min_{\boldsymbol{w},b,\xi} \lambda||\boldsymbol{w}||^2 + \sum_{i\in\mathcal{D}} \xi_i$$
$$\text{Constraints: } y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \quad \forall i \in \mathcal{D}$$

Rewriting constraints as $1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b) \leq 0$ and $-\xi_i \leq 0$ to make them of the form $g(w) \leq c$.
Using convex optimisation of (2) we rewrite our formulation as:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \min_{\boldsymbol{w},b,\xi} \lambda||\boldsymbol{w}||^2 + \sum_{i\in\mathcal{D}} \xi_i + \sum_{i\in\mathcal{D}} \alpha_i(1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)) + \sum_{i\in\mathcal{D}} \beta_i(-\xi_i)$$
$$\text{s.t. } \alpha_i \geq 0, \beta_i \geq 0 \ \ \forall i \in \mathcal{D}$$

This is our final optimisation problem in dual space and the function to be maximised is called Lagrangian dual function $g(\boldsymbol{\alpha}, \boldsymbol{\beta})$.
Note : λ used here is different from the λ used in (2) of convex optimisation.

4.3 Optimality Conditions

Differentiating $g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ w.r.t $\boldsymbol{w}, b, \xi_i$ and equating to 0, we get:

$$\frac{\partial g}{\partial \boldsymbol{w}} = 0 \Rightarrow 2\lambda \boldsymbol{w^*} + \sum_{i\in\mathcal{D}} \alpha_i(-y_i \boldsymbol{x_i}) = 0$$
$$\Rightarrow \boldsymbol{w^*} = \sum_{i\in\mathcal{D}} \frac{\alpha_i y_i \boldsymbol{x_i}}{2\lambda}$$
$$\frac{\partial g}{\partial b} = 0 \Rightarrow \sum_{i\in\mathcal{D}} \alpha_i y_i = 0$$
$$\frac{\partial g}{\partial \xi_i} = 0 \Rightarrow 1 - \alpha_i - \beta_i = 0$$
$$\Rightarrow \alpha_i + \beta_i = 1 \quad \forall i \in \mathcal{D}$$

We also see that our objective function is quadratic in $\boldsymbol{w}$ and constraints are linear in $\boldsymbol{w}, \xi_i$. Therefore, all functions are strictly convex and we can apply Slater's condition. Therefore,

$$\alpha_i^*(1 - \xi_i^* - y_i(\boldsymbol{w^*}^\top \boldsymbol{x_i} + b^*)) = 0 \quad \forall i \in \mathcal{D}$$
$$\beta_i^* \xi_i^* = 0 \quad \forall i \in \mathcal{D}$$

Let's look at 2 cases:

1. $\xi_i^* > 0 \Rightarrow$ point is incorrectly classified or it is inside the margin of hyperplanes
   $\Rightarrow \beta_i^* = 0 \Rightarrow \alpha_i^* = 1$.
   α_i^* being high can be seen as penalty being high, which intuitively means that the point is misclassified.

2. $\xi_i = 0 \Rightarrow$ point is on or outside the margin of hyperplanes
   If $1 - y_i(\boldsymbol{w^{*\top}} \boldsymbol{x_i} + b^*) > 0 \Rightarrow \alpha_i^* = 0 \Rightarrow \beta_i^* = 1$. This means that penalty is low and point is correctly classified.
   If $1 - y_i(\boldsymbol{w^{*\top}} \boldsymbol{x_i} + b^*) = 0$, then we cannot comment anything about α_i^*.

4.4 Dual Problem at Optimality

Let us substitute optimal values into g and rewrite our dual problem. We have:

$$\begin{aligned}
\boldsymbol{w^*} &= \sum_{i \in \mathcal{D}} \frac{\alpha_i y_i \boldsymbol{x_i}}{2\lambda}, \ \sum_{i \in \mathcal{D}} \alpha_i y_i = 0, \ \alpha_i + \beta_i = 1, \ \alpha_i \geq 0, \ \beta_i \geq 0 \\
g(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \lambda ||\boldsymbol{w^*}||^2 + \sum_{i \in \mathcal{D}} \xi_i^* (1 - \alpha_i - \beta_i) + \sum_{i \in \mathcal{D}} \alpha_i (1 - y_i(\boldsymbol{w^{*\top}} \boldsymbol{x_i} + b^*) \\
&= \lambda \boldsymbol{w^{*\top}} \boldsymbol{w^*} + \sum_{i \in \mathcal{D}} \xi_i^*(0) + \sum_{i \in \mathcal{D}} \alpha_i - \boldsymbol{w^{*\top}} \sum_{i \in \mathcal{D}} \alpha_i y_i \boldsymbol{x_i} \\
&= \sum_{i \in \mathcal{D}} \alpha_i - \frac{1}{4\lambda} \sum_{i \in \mathcal{D}} \alpha_i y_i \boldsymbol{x_i^\top} \sum_{j \in \mathcal{D}} \alpha_j y_j \boldsymbol{x_j}
\end{aligned}$$

Thus, our dual problem has become:

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}} \sum_{i \in \mathcal{D}} \alpha_i - \frac{1}{4\lambda} \sum_{i \in \mathcal{D}} \sum_{j \in \mathcal{D}} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i^\top} \boldsymbol{x_j} \\
&\text{s.t. } 0 \leq \alpha_i \leq 1 \ \ \forall i \in \mathcal{D} \ \text{ and } \ \sum_{i \in \mathcal{D}} \alpha_i y_i = 0
\end{aligned}$$

We have conveniently got rid of $\boldsymbol{\beta}$ and derived a clean optimization problem.

AIML - CS 337

Lecture 12: Kernel Methods

26/09/2022

Lecturer: Abir De *Scribe: Group 23, Group 25, Group 26*

In this Lecture we discuss the smooth transition from Linear to Non-Linear space of functions. We discuss treatments where f(x) is non-linear in x that is, $f(x) \neq w^T x$, but we can construct it using our prior knowledge. We transform the data into a higher dimensional space and finds out that data is separable when transformed into higher dimensions. We introduce "kernel method", a method of applying SVMs to problems with non linear classification boundaries. kernels are basically measure of similarity between two vectors.

1 Introduction

Remember we had a question in midsem (question 2.d) where the labels are +1 for all x such that |x| > 2 and -1 otherwise. We were asked to provide a 1D transformation $\phi : \mathbb{R} \to \mathbb{R}$ so that SVM applied to the given dataset.

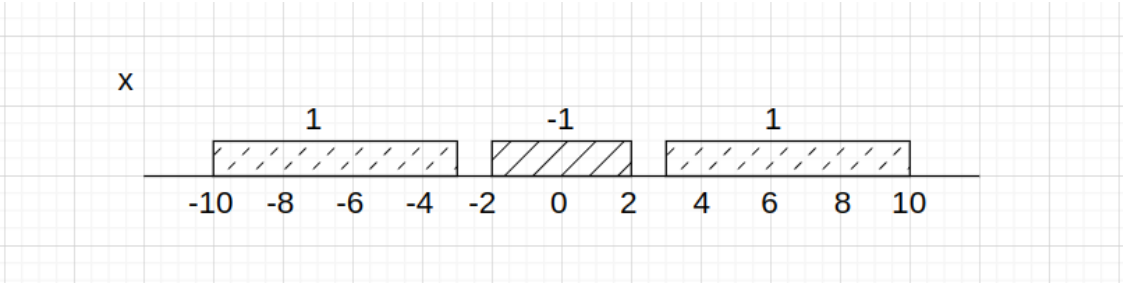


Figure 1: Initial input space : x

As it can be seen that SVM can not be applied to the initial input space.
Now let us first define a mapping $\phi : R \hookrightarrow R^2$ as follows:

$$\phi(x) = (x, x^2)$$

Now our prediction will be $h(\phi(x))$ instead of $h(x)$, where $\phi(x) = x^2$ and h is the learnt classifier, it is possible to apply SVM in the transformed space as seen in the Figure 2.

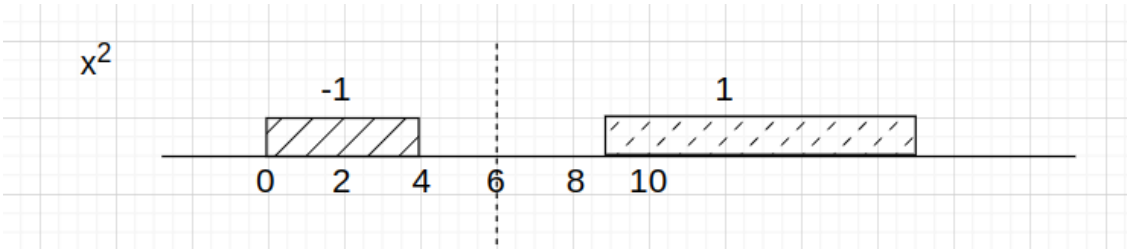


Figure 2: Transformed input space : x^2

The Main Concept behind Kernel Methods is that there is a possibility that, data-points that are not linearly-separable in lower dimensions are linearly-separable in higher dimensions.

2 Kernel Methods

The Basic Algorithm will be:-

- Given some Data Set $S = (\{\boldsymbol{x_i}\}_{i=0}^n, \{y_i\}_{i=0}^n)$, where $\boldsymbol{x_i} \in \mathbb{R}^d$
- Consider a function $\boldsymbol{\phi}$ such that $\boldsymbol{\phi(x)} \in \mathbb{R}^{d'}$ where $d' > d$ (and can even be infinity)
- Create a new Data Set $\hat{S} = (\{\boldsymbol{\phi(x_i)}\}_{i=0}^n, \{y_i\}_{i=0}^n)$
- Train a linear Predictor h over $\hat{S}$
- And then the prediction of any point $\boldsymbol{x_{test}}$ in the test dataset is given by $h(\boldsymbol{\phi(x_{test})})$

Thus the prediction is given by $\boldsymbol{w^T \phi(x)}$ where both $\boldsymbol{w}, \boldsymbol{\phi(x)} \in \mathbb{R}^{d'}$

As d' can reach infinity, this is theoretically possible for non linear functions but calculating and storing $\boldsymbol{w}$ and $\boldsymbol{\phi(x)}$ becomes practically impossible, but the dot product $\boldsymbol{w^T \phi(x)}$ is a scalar.

Here we are enriching the expressive power of halfspaces by first mapping the data into a high dimensional feature space, and then learning a linear predictor in that space. While this approach greatly extends the expressiveness of halfspace predictors, it raises both sample complexity and computational complexity challenges. We tackle this using the method of *kernels*.

A popular choice for the mapping ϕ is a polynomial mapping: $x \to (1, x, x^2, \ldots)$

The Setup:
Given a mapping ϕ we are left to solve the optimization problem:

$$\min_w f(\{y_i\}_i^n, \{\boldsymbol{w^T \phi(x_i)}\}_i^n) + \lambda R(\boldsymbol{w})$$

Where f is a loss function and R is a monotonic Regularization function.

2.1 Transformation

- In a finite case if we sample d dimension vector N times such that $d << N$. .It is with high probability that there will be d vectors that are linearly independent of each other. But as the value of d itself tends to infinity it becomes less and less likely
- In the given case the vectors $\{\boldsymbol{\phi(x_i)}\}_i^n$ are of infinite dimension. So if the optimum solution is $\boldsymbol{w^*}$ then there exists $\{\alpha_i\}_i^n$ such that $\boldsymbol{w^*}$ can be represented as

$$\boldsymbol{w^*} = \sum_{i=0}^{n} \alpha_i * \boldsymbol{\phi(x_i)} \ + \boldsymbol{v}$$

  where $\boldsymbol{v^T \phi(x_i)} = 0$ for all i, that is, **v** is orthogonal to the span of the vectors mapped by the function ϕ.
- Let us define $\boldsymbol{w} = \boldsymbol{w^*} - \boldsymbol{v}$

$$\begin{aligned}
&||\boldsymbol{w}||^2 = ||\boldsymbol{w^*} - \boldsymbol{v}||^2 \\
&||\boldsymbol{w}||^2 = ||\boldsymbol{w^*}||^2 - 2\boldsymbol{w^*}.\boldsymbol{v} + ||\boldsymbol{v}||^2 \qquad \text{As the value } \boldsymbol{w^*}.\boldsymbol{v} = \boldsymbol{v}.\boldsymbol{v} = ||\boldsymbol{v}||^2 \\
&||\boldsymbol{w}||^2 = ||\boldsymbol{w^*}||^2 - 2||\boldsymbol{v}||^2 + ||\boldsymbol{v}||^2 \\
&||\boldsymbol{w}||^2 = ||\boldsymbol{w^*}||^2 - ||\boldsymbol{v}||^2
\end{aligned}$$

- So as norm is a positive function we have $||\boldsymbol{w}|| \leq ||\boldsymbol{w^*}||$
- And since R is non-decreasing, we obtain $R(\boldsymbol{w}) \leq R(\boldsymbol{w^*})$
  (R is monotonic regularization function as defined above)

- Also as $\boldsymbol{w^T\phi(x)} = (\boldsymbol{w^*} - \boldsymbol{v})^T\boldsymbol{\phi(x)} = \boldsymbol{w^{*^T}\phi(x)}$
  (since v is orthogonal to $\boldsymbol{\phi(x)}$, we have $\boldsymbol{v^T\phi(x)}$=0)
- And so,

$$f(\{y_i\}_i^n, \{\boldsymbol{w^T\phi(x_i)}\}_i^n) = f(\{y_i\}_i^n, \{\boldsymbol{w^{*^T}\phi(x_i)}\}_i^n)$$

- We have shown that the loss function is same for both $\boldsymbol{w^*}$ and $\boldsymbol{w}$ and the regularization function is less for $\boldsymbol{w}$ than $\boldsymbol{w^*}$,
- Hence the objective function for $\boldsymbol{w}$ is less than that for $\boldsymbol{w^*}$, but as $\boldsymbol{w^*}$ is the optimum solution, we must have that $\boldsymbol{w}$ is also an optimum solution .
- Hence we have proved that the value

$$\boldsymbol{w} = \sum_{i=0}^{n} \alpha_i * \boldsymbol{\phi(x_i)}$$

  is an optimum solution for the objective function.[1]

Theorem 2.1 (**Representer Theorem**). *Given a mapping from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, there exists a vector $\boldsymbol{\alpha} \in \mathbb{R}^{d'}$ such that $\boldsymbol{w} = \sum_{i=1}^{d'} \alpha_i\boldsymbol{\phi(x_i)}$ is an optimal solution.*

$$\min_w f(\{y_i\}_i^n, \{\boldsymbol{w^T\phi(x_i)}\}_i^n) + \lambda R(\boldsymbol{w})$$

2.2 Kernel

Substituting the fact that $\boldsymbol{w} = \sum_{i=1}^{d'} \alpha_i\boldsymbol{\phi(x_i)}$ is an optimum solution into the value $\boldsymbol{w^T\phi(x)}$

$$\begin{aligned}
\boldsymbol{w^T\phi(x)} &= \left(\sum_{i=1}^{d'} \alpha_i\boldsymbol{\phi(x_i)^T}\right)\boldsymbol{\phi(x)} \\
&= \sum_{i=1}^{d'} \alpha_i\boldsymbol{\phi(x_i)^T\phi(x)} \qquad \text{consider a function } K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{\phi(x_i)^T\phi(x_j)} \\
&= \sum_{i=1}^{d'} \alpha_i K(\boldsymbol{x_i}, \boldsymbol{x})
\end{aligned}$$

[1] Also remember that this was the result that we got by applying the Lagrange multiplier on the objective functions

Here K is called a *Kernel* Function and it denotes *Similarity* between the data points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$.

What does this expression physically mean? We want to make a prediction at a new test point x, then $w^{*T}\phi(x)\ y(x)$ gives the **weighted mean** of labels $y(x_i)$ with weights as similarity

$$S(x, x_i) \ = \ \phi(x_i)^T\phi(x)$$

thus giving more weight to similar neighbours. This essentially represents the idea of K-Means clustering.
We can note that this similarity function can be any form, not necessarily dot product, The term "kernels" is used in this context to describe inner product in the feature space

Implications of this result

- This representation takes us beyond SVM
- w is linear combination of $\phi(x_i)$ thus $w^T\phi(x)$ can be easily calculated.

Some Popular Kernels are:

- **SVM** $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i^T x_j}$
- **Gaussian** $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{\frac{-||\boldsymbol{x_i}-\boldsymbol{x_j}||^2}{\sigma^2}}$
- **k degree polynomial** $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (1 + \langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle)^k$

where $\langle ., . \rangle$ denotes the inner product.

Exercise

Que- Show that RBF can be written as inner product

It is easy to see that RBF(or Gaussian) Kernel satisfies the positive semidefinite condition, we can infact state and prove a possible inner product representation

Consider

$$\phi(x) \ = \ \sum_{n=0}^{\infty} \frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n$$

Observe that

$$< \phi(x), \phi(x^{'}) > = \sum_{n=0}^{\infty} \left(\frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n \right) \left(\frac{1}{\sqrt{n!}} e^{-\frac{x^{'2}}{2}} x^{'n} \right)$$

$$< \phi(x), \phi(x^{'}) > = e^{-\frac{x^2 + x^{'2}}{2}} \sum_{n=0}^{\infty} \left(\frac{(xx^{'})^n}{n!} \right)$$

$$< \phi(x), \phi(x^{'}) > = e^{-\frac{x^2 + x^{'2}}{2} + xx^{'}}$$

$$< \phi(x), \phi(x^{'}) > = e^{-\frac{(x - x^{'})^2}{2}}$$

Que- Do the same for polynomial kernel
Left for reader
Hint: Solution can be checked in book Understanding machine learning chapter Kernel Methods

3 SVM Kernel

Now going back to the dual formulation of Hard-SVM:

$$\max_{\boldsymbol{\alpha} \in R^D : \boldsymbol{\alpha} \geq \mathbf{0}} \left(\sum_{i=1}^{D} \alpha_i - \frac{1}{2} \sum_{i=1}^{D} \sum_{j=1}^{D} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i^T x_j} \right)$$

$$= \max_{\boldsymbol{\alpha} \in R^D : \boldsymbol{\alpha} \geq \mathbf{0}} \left(\sum_{i=1}^{D} \alpha_i - \frac{1}{2} \sum_{i=1}^{D} \sum_{j=1}^{D} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \right)$$

$$= \max_{\boldsymbol{\alpha} \in R^D : \boldsymbol{\alpha} \geq \mathbf{0}} \left(\sum_{i=1}^{D} \alpha_i - \frac{1}{2} (\boldsymbol{\alpha} * \boldsymbol{Y})^{\boldsymbol{T}} \boldsymbol{G} (\boldsymbol{Y} * \boldsymbol{\alpha}) \right)$$

$*$ denotes element-wise multiplication

$\boldsymbol{G} = \{K(\boldsymbol{x_i}, \boldsymbol{x_j})\}_{i,j=1}^{D \times D}$ Known as the **Gram matrix**

Hence, we are optimising for $\boldsymbol{\alpha}$ instead of $\boldsymbol{w}$.

The advantage of working with kernels rather than directly optimizing w in the feature space is that in some situations the dimension of the feature space is extremely large while implementing the kernel function is very simple.

Theorem 3.1 (Mercers's Theorem)**.** $K : [a, b] \times [a, b] \to \mathbb{R}$ *is a kernel if and only if :*

1. *Symmetric :* $K(x, y) = K(y, x)$ *for all* $x, y \in [a, b]$*, and*
2. *Positive Semi-definite :* $\sum_{j=1}^{n} \sum_{j=1}^{n} K(x_i, x_j)c_i c_j \geq 0$ *for all finite sequences of points* $\mathbf{x}_1, ..., \mathbf{x}_n$ *of* $[a, b]$ *and all choices of real numbers* $c_1, ..., c_n$

References

[1] S. B.-D. Shai Shalev-Shwartz. *Understanding Machine Learning, Chapter 16*. Cambridge University Press, 2014.

AIML - CS 337

Lecture 14: Kernel Methods - II

29th Sept 2022

Lecturer: Abir De *Scribe:*

1 Kernel for probability distributions

In the previous lecture we had seen that a Kernel can be defined as an inner product in the feature space as thus:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \tag{1}$$

Extending on this definition let $x \in A, x' \in B$, and x, x' be drawn from probability distributions P_1, P_2 respectively. We can define a Kernel K over the sets A,B as such:

$$K(A, B) = \int_{-\infty}^{\infty} \phi_A^T(x).\phi_B(x')Pr(x, x') \tag{2}$$

$$= \iint_{x \in A, x' \in B} \phi^T(x)\phi(x')dP(x, x') \tag{3}$$

Now, one possible measure of similitarity between the two sets A and B is $P(A \cap B) - P(A)P(B)$. To show that this indeed is a valid kernel, we can show that there exists some ϕ such that $K(A, B) = P(A \cap B) - P(A)P(B)$ and $K(A, B)$ satisfies Equation (3).

Consider $\phi_A(x) = \mathbb{I}_A(x) - P(A)$ where $\mathbb{I}_A(x)$ is the indicator function:

$$\mathbb{I}_A(x) = \begin{cases} 1 & x \in A \\ 0 & \text{otherwise} \end{cases}$$

Substituting in Equation (2) we get:

$$\begin{aligned} K(A, B) &= \int_{-\infty}^{\infty} (\mathbb{I}_A(x) - P(A))(\mathbb{I}_B(x') - P(B))Pr(x, x') \\ &= \int_{-\infty}^{\infty} (\mathbb{I}_A(x)\mathbb{I}_B(x'))Pr(x, x') - P(A) \int_{-\infty}^{\infty} (\mathbb{I}_B(x'))Pr(x, x') \\ &- P(B) \int_{-\infty}^{\infty} (\mathbb{I}_A(x'))Pr(x, x') + P(A)P(B) \int_{-\infty}^{\infty} Pr(x, x') \\ &= P(A \cap B) - P(A)P(B) - P(A)P(B) + P(A)P(B) \\ &= P(A \cap B) - P(A)P(B) \end{aligned}$$

2 Inner Product of Functions

We define inner product of two functions f,g as:

$$\langle f, g \rangle \doteq \int_{x,y} f(x)g(y)dP(x,y)$$

Under this inner product definition, K is Kernel for ϕ.
Properties of inner product:

1. Positive Semidefinite: $\langle f, f \rangle \ \geq 0$
2. Symmetric: $\langle f, g \rangle = \langle g, f \rangle$
3. Linearity: $< \langle c_1 f_1 + c_2 f_2, g \rangle = c_1 \langle f_1, g \rangle + c_2 \langle f_2, g \rangle$

We define norm of function using this inner product definition as:

$$\|f\| \doteq \sqrt{\langle f, f \rangle}$$

3 Finding similarity from loss

Until now, we assumed that Kernel was given to us and we used the kernel as a measure of similarity. But what if we have to find the Kernel if the loss is given.
Let F(w) be defined as follows:

$$F(w) = \sum_{i \in D} l(h_w(x_i), y_i)$$

Given this loss, we now need to find the similarity between points in Dataset.

$$\begin{bmatrix} Loss(x_i|t=0) \\ Loss(x_i|t=1) \\ . \\ . \\ . \end{bmatrix}$$

We make vectors of all x_i's and compare their similarity. We use this to select batches of data which have different training curve. If loss of two points is similar, we can not say that the points themselves are also similar. This is because weights are randomly initialized, so loss cannot be a good measure of similarity. We can instead use gradient of loss to find similarity.
If $X_i \sim X_j$ then $\nabla_w l(h_w(x_i), y_i) \approx \nabla_w l(h_w(x_j), y_j)$ but not the other way round.
Therefore, we can define the kernel as follows:

$$K(x_i, x_j) = E_w[\nabla_w^T l(h_w(x_i), y_i).\nabla_w l(h_w(x_j), y_j)]$$

4 Final Problem

Consider now the following optimization objective:

$$\min_{f\in\Lambda}\sum_{i\in\mathcal{D}}(y_i-f(x_i))^2+\lambda\sum_{i\in\mathcal{D}}f(x_i)^2$$

where f is defined in the vector space Λ of functions generated by the set

$$\{k(x_i,\cdot)\}_{i\in\mathcal{D}}$$

which is a linear subspace of $\mathbb{R}^{\mathcal{X}}$, where $x_i\in\mathcal{X}$ for all $i\in\mathcal{D}$ and $k(\cdot,\cdot)$ is the kernel function defined $\mathcal{X}\times\mathcal{X}\xrightarrow{k}\mathbb{R}$. This vector space is also equipped with the following inner product:

$$\left\langle\sum_{i\in\mathcal{D}}\alpha_i k(x_i,\cdot),\sum_{j\in\mathcal{D}}\beta_j k(x_j,\cdot)\right\rangle=\sum_{i\in\mathcal{D}}\sum_{j\in\mathcal{D}}\alpha_i\beta_j k(x_i,x_j)$$

That the above is an inner product space is easily verified. Indeed, for all $g\in\Lambda$, there are $\alpha_i\in\mathbb{R}$ for all $i\in\mathcal{D}$ such that $g=\sum_{i\in\mathcal{D}}\alpha_i k(x_i,\cdot)$.

$$\langle g,g\rangle=\sum_{i\in\mathcal{D}}\sum_{j\in\mathcal{D}}\alpha_i\alpha_j k(x_i,x_j)\geq 0$$

since $k(\cdot,\cdot)$ is a kernel and therefore is positive semidefinite. Linearity in both operands of $\langle\cdot,\cdot\rangle$ is implicit from the definition and finally symmetry of $\langle\cdot,\cdot\rangle$ follows from the symmetry of $k(\cdot,\cdot)$.

As a result, we may rephrase the objective as

$$\min_{\alpha\in\mathbb{R}^{|\mathcal{D}|}}\sum_{i\in\mathcal{D}}\left(y_i-\sum_{j\in\mathcal{D}}\alpha_j k(x_j,x_i)\right)^2+\lambda\sum_{i\in\mathcal{D}}\left(\sum_{j\in\mathcal{D}}\alpha_j k(x_j,x_i)\right)^2$$

We have

$$\begin{aligned}\sum_{i\in\mathcal{D}}f(x_i)^2&=\sum_{i\in\mathcal{D}}\left(\sum_{j\in\mathcal{D}}\alpha_j k(x_j,x_i)\right)^2\\&=\sum_{i\in\mathcal{D}}\left(\sum_{j\in\mathcal{D}}\sum_{k\in\mathcal{D}}\alpha_j\alpha_k k(x_j,x_i)k(x_k,x_i)\right)\end{aligned}$$

Finally, we note that the norm of f in the aforementioned inner product space is given by

$$\begin{aligned}\left\langle\sum_{i\in\mathcal{D}}\alpha_i k(x_i,\cdot),\sum_{i\in\mathcal{D}}\alpha_j k(x_j,\cdot)\right\rangle&=\sum_{i\in\mathcal{D}}\sum_{j\in\mathcal{D}}\alpha_i\alpha_j k(x_i,x_j)\\&=\alpha^T G\alpha\end{aligned}$$

where $G=\Big[k(x_i,x_j)\Big]_{|\mathcal{D}|\times|\mathcal{D}|}$ and $\alpha=\begin{bmatrix}\alpha_1 & \cdots & \alpha_{|\mathcal{D}|}\end{bmatrix}^T$.

5 Homework Problem

Problem. Show that the following kernel is positive semidefinite:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right)$$

Solution. Note the following equality:

$$\int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{(x-z)^2}{\sigma^2}\right) \frac{1}{\sigma\sqrt{\pi}} \exp\left(-\frac{(y-z)^2}{\sigma^2}\right) \, dz = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right)$$

which is equivalent to the following (assuming a Euclidean norm):

$$\int_{\mathbb{R}^n} \left(\frac{1}{\sigma}\sqrt{\frac{2}{\pi}}\right)^n \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{\sigma^2}\right) \exp\left(-\frac{\|\boldsymbol{y} - \boldsymbol{z}\|^2}{\sigma^2}\right) \, d\boldsymbol{z} = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right)$$

Let $\{\boldsymbol{x}_i\}_{i \in \mathcal{D}}$ be a set of data points. Then, for any sequence of real numbers $\{c_i\}_{i \in \mathcal{D}}$, we have

$$\begin{aligned}
&\sum_{i \in \mathcal{D}} \sum_{j \in \mathcal{D}} c_i c_j \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) \\
&= \left(\frac{1}{\sigma}\sqrt{\frac{2}{\pi}}\right)^n \int_{\mathbb{R}^n} \sum_{i,j \in \mathcal{D} \times \mathcal{D}} c_i c_j \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{z}\|^2}{\sigma^2}\right) \exp\left(-\frac{\|\boldsymbol{x}_j - \boldsymbol{z}\|^2}{\sigma^2}\right) \, d\mathbf{z} \\
&= \left(\frac{1}{\sigma}\sqrt{\frac{2}{\pi}}\right)^n \int_{\mathbb{R}^n} \left[\sum_{i \in \mathcal{D}} c_i \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{z}\|^2}{\sigma^2}\right)\right]^2 \, d\mathbf{z} \\
&\geq 0
\end{aligned}$$

which is obviously non-negative. This completes the proof. ■

AIML - CS 337

Lecture 14: Kernel Methods

3rd October 2022

Lecturer: Abir De *Scribe: Group 29 and Group 30*

1 Recap : SVM formulation

Recall the discussion of earlier classes

$$w^*_{svm} = \frac{\sum_i^{|D|} \alpha_i y_i x_i}{2\lambda}$$

This is linear in x, and the $dim(x) = dim(w) < \infty$ To generalise this, suppose we make it non-linear in x, but it's linear in some $\phi(x)$ which can be ∞-dimensional. Previously the similarity mechanism involved $x_i^T x$. The new similarity mechanism uses the kernel formulation $K(x_i, x)$ for e.g $K(x_i, x) = e^{-||x_i - x||^2}$. Formally, this new "similarity measure" must have some properties, which are discussed later.

2 Mathematics

We continue with our discussion on kernel methods/tricks in this lecture with more rigorous mathematics.

2.1 Inner Product Space

An inner product space (over reals) is a vector space $\mathcal{V}$ and an inner product, which is a mapping

$$\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathcal{R}$$

that has the following properties $\forall x, y, z \in \mathcal{V}$ and $a, b \in \mathcal{R}$:

- Symmetry: $\langle x, y \rangle = \langle y, x \rangle$
- Linearity: $\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle$
- Positive-definiteness: $\langle x, x \rangle \geq 0$ and $\langle x, x \rangle = 0 \iff x = 0$

For an inner product space, we define norm as $\|x\| = \sqrt{\langle x, x \rangle}$

2.2 Hilbert Space

A *Hilbert Space* is an inner product space that is complete and seperable with respect to the norm defined by the inner product.A space is called complete if all Cauchy Sequences in the space converge. Examples of Hilbert spaces include :

1. $\mathbb{R}^n$ is an Hilbert space for the Euclidean norm. The dot-product is defined as with $\langle a, b\rangle = a^T b$, the vector dot product of a and b.
2. The space l_2 of square summable sequences, with inner product $\langle x, y\rangle = \sum_{i=0}^{\infty} x_i y_i$

Definition 2.1. Kernel
Let $\mathcal{X}$ be a non-empty set. A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there exists a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}$ such that $\forall x, x' \in \mathcal{X}$,

$$K(x, x') = \langle \phi(x), \phi(x')\rangle_{\mathcal{H}}$$

If we are given a function of two arguments, $K(x, x')$, the following can be used to determine if it is a valid kernel.

1. Find a feature map. But this may not be obvious sometimes, and the feature map may not be unique.
2. Can use a direct property of the function which is positive definiteness. The following lemma gives a sufficient and necessary condition.

Lemma 2.2. *Let $\mathcal{H}$ be a Hilbert space, $\mathcal{X}$ a non-empty set and $\phi : \mathcal{X} \to \mathcal{H}$. A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implements an inner product in $\mathcal{H}$ if and only if it is positive semidefinite; namely $\forall (x_1, \ldots, x_n) \in \mathcal{X}^n$, the Gram matrix $G_{i,j} = K(x_i, x_j)$, is a positive semidefinite matrix.*

Proof. $\implies$ (If K implements an inner product then the Gram matrix is positive semidefinite)

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n}\langle a_i\phi(x_i), a_j\phi(x_j)\rangle_{\mathcal{H}} \\ &= ||\sum_{i=1}^{n} a_i\phi(x_i)||^2_{\mathcal{H}} \geq 0\end{aligned}$$

$\impliedby$ For this direction, define the space of functions over $\mathcal{X}$ as $\mathbb{R} = \{f : \mathcal{X} \to \mathbb{R}\}$ For each $x \in \mathcal{X}$ let $\phi(x)$ be the function $x \mapsto K(\cdot, x)$. Define a vector space by taking all linear combinations of elements of the form $K(\cdot, x)$. Define an inner product on this vector space to be

$$\langle \sum_i \alpha_i K(\cdot, x_i), \sum_j \beta_j K(\cdot, x'_j)\rangle = \sum_{i,j} \alpha_i \beta_j K(x_i, x'_j)$$

This is a valid inner product since it is symmetric (because K is symmetric), it is linear, and it is positive definite. Clearly,

$$\langle \phi(x), \phi(x') \rangle = \langle K(\cdot, x), K(\cdot, x') \rangle = K(x, x').$$

□

2.3 Projection Theorem & Properties

Theorem 2.3. *Let $\mathcal{H}$ be a Hilbert space and $\mathcal{M}$ be a closed subspace of $\mathcal{H}$. Then for any $x \in \mathcal{H}$, there exists a unique $m_0 \in \mathcal{M}$ for which*

$$\|x - m_0\| \leq \|x - m\| \forall m \in \mathcal{M}$$

This m_0 is called the projection of x onto $\mathcal{M}$. Furthermore, $m_0 \in \mathcal{M}$ is the projection of x onto M iff

$$x - m_0 \perp \mathcal{M}$$

Theorem 2.4. *Let $\mathcal{M}$ be a closed subspace of $\mathcal{H}$. For any $x \in \mathcal{H}$, let m_0 be the projection of x onto $\mathcal{M}$. Then*

$$\|m_0\| \leq \|x\|$$

with equality only when $m_0 = x$.

3 Generalised objective function

Consider

$$\min_{w} \; l(\{w^T \phi(x_i)\}_{i \in D}, \{y_i\}_{i \in D}) + \lambda R(||w||) \tag{1}$$

where $l : \mathbb{R}^{|D|} \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically non-decreasing function.

Theorem 3.1. ***Representer Theorem***

Assume that ϕ is a mapping from $\mathcal{X}$ to a Hilbert space. Then, there exists a vector $\alpha \in \mathbb{R}^{|D|}$ such that $w = \sum_{i=1}^{|D|} \alpha_i \phi(x_i)$ is an optimal solution of equation 1

Proof. Let w^* be an optimal solution of Equation 1. Because w^* is an element of a Hilbert space, we can rewritew^* as

$$w* = \sum_{i=1}^{|D|} \alpha_i \phi(x_i) + u$$

where $\langle u, \phi(x_i) \rangle = 0$ for all i. Set $w = w^* - u$. Clearly, $||w^*||^2 = ||w||^2 + ||u||^2$, thus $||w|| < ||w^*||$. Since R is non-decreasing we obtain that $R(||w||) < R(||w^*||)$. Additionally, for all i we have that

$$\langle w, \phi(x_i) \rangle = \langle w^* - u, \phi(x_i) \rangle = \langle w^*, \phi(x_i) \rangle,$$

hence

$$l(\{w^T\phi(x_i)\}_{i\in D}, \{y_i\}_{i\in D}) = l(\{w^{*T}\phi(x_i)\}_{i\in D}, \{y_i\}_{i\in D})$$

We have shown that the objective of Equation 1 at w cannot be larger than the objective at w^* and therefore w is also an optimal solution. Since $w = \sum_{i=1}^{|D|} \alpha_i \phi(x_i)$ we conclude our proof. □

Form of f is

$$\begin{aligned} f(x) &= w^{*T}\phi(x_i) \\ &= \sum_{i=1}^{|D|} \alpha_i \phi^T(x_i)\phi(x) \end{aligned}$$

Here $\phi^T(x_i)\phi(x)$ is like a similarity measure If $\phi(\cdot)$ is ∞-dimensional, we can write it as

$$f(x) = \sum_{i=1}^{|D|} \alpha_i \sum_{j=0}^{\infty} \phi(x_i)[j]\phi(x)[j]$$

Hence, if $\phi(\cdot)$ is ∞-dimensional, it is not feasible to code w as it has the same dimension as ϕ. So, we try to represent the objective function in functional form or through the kernel formulation.

3.1 Objective in terms of Kernel

Writing $w = \sum_{j=1}^{|D|} \alpha_j \phi(x_j)$, we have that for all i

$$\langle w, \phi(x_i)\rangle = \langle \sum_{j=1}^{|D|} \alpha_j \phi(x_j), \phi(x_i\rangle = \sum_{j=1}^{|D|} \alpha_j \langle \phi(x_j), \phi(x_i)\rangle.$$

Similarly,

$$||w||^2 = \langle \sum_{j=1}^{|D|} \alpha_j \phi(x_j), \sum_{j=1}^{|D|} \alpha_j \phi(x_j)\rangle = \sum_{i,j=1}^{|D|} \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j)\rangle.$$

Let $K(x, x') = \langle \phi(x), \phi(x')\rangle$ be a function that implements the kernel function with respect to the feature space. Hence, instead of solving Equation 1, we can solve the equivalent problem

$$\min_{\alpha\in\mathbb{R}^{|D|}} l(\{\sum_{j=1}^{|D|} \alpha_j K(x_j, x_i)\}_{i\in D}, \{y_i\}_{i\in D}) + \lambda R(\sqrt{\sum_{i,j=1}^{|D|} \alpha_i \alpha_j K(x_j, x_i)}) \tag{2}$$

3.2 Objective in terms of functional form

$$f(x) = \sum_i \alpha_i K(x_i, x)$$

Function f forms a vector space

$$f_1, f_2 \in V \implies af_1 + bf_2 \in V$$

$$0 \in V, \text{ by putting } \alpha_i = 0 \quad \forall i$$

Now define an inner product

$$\langle f, g \rangle_H = \sum \alpha_i^f \alpha_j^g K(x_i, x_j) \tag{3}$$

From the properties of inner product space, for 3 to be true, $\sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \geq 0$ and $K(x_i, x_j) = K(x_j, x_i) \quad \forall i, j$

These are in accordance with the earlier lemma which we proved.

$$\begin{aligned}
||w||^2 &= \langle w, w \rangle \\
&= \langle \sum_{i=1}^{|D|} \alpha_i \phi(x_i), \sum_{i=1}^{|D|} \alpha_j \phi(x_j) \rangle = \sum_{i,j=1}^{|D|} \alpha_i \alpha_j \langle \phi(x_i) \phi(x_j \rangle \\
&= \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) = \langle f, f \rangle \\
&= ||f||^2
\end{aligned}$$

Hence, the objective function becomes

$$\min_w \; l(\{f(x_i)\}_{i \in D}, \{y_i\}_{i \in D}) + \lambda R(||f||)$$

4 Kernel Matrix & Prediction function

4.1 Kernel Matrix

We define $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

Definition 4.1. We define the kernel matrix for a kernel k on a set $\{x_1, ..., x_n\}$ is

$$K = (k(x_i, x_j))_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \in \mathcal{R}^{n \times n}$$

4.2 Prediction Function

Consider the minimizer $w = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ according to the representer theorem. Then for a given x, we define the prediction function as

$$\begin{aligned} f(x) &= \langle w, \phi(x) \rangle \\ &= \sum_{i=1}^{n} \alpha_i \langle \phi(x_i), \phi(x) \rangle \\ &= \sum_{i=1}^{n} \alpha_i k(x_i, x) \end{aligned}$$

5 Different forms of Objective Function

5.1 In terms of Kernel Matrix and α

Consider $w = \sum_{i=1}^{n} \alpha_i \phi(x_i)$. Then we have for norm

$$\begin{aligned} \|w\|^2 &= \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \\ &= \alpha^T K \alpha \end{aligned}$$

Similarily, predictions on the training points have a particular simple form:

$$\begin{aligned} \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} &= \begin{pmatrix} \alpha_1 k(x_1, x_1) + \cdots + \alpha_n k(x_1, x_n) \\ \vdots \\ \alpha_1 k(x_n, x_1) + \cdots + \alpha_n k(x_n, x_n) \end{pmatrix} \\ &= \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \\ &= K\alpha \end{aligned}$$

Hence our generalised objective function can be reduced to using the knowledge that minimizer lies in the span of $\phi(x_1), ..., \phi(x_n)$

$$\min_{\alpha \in \mathcal{R}^n} R(\sqrt{\alpha^T K \alpha}) + L(K\alpha)$$

This is the kernelized objective function

5.2 In terms of prediction function

Recall that $f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, x_i)$. Now we define a dot product of f and another function $g(\cdot) = \sum_{i}^{m'} \beta_j k(\cdot, x'_j)$ as follows

$$\langle f, g \rangle := \sum_{i}^{m} \sum_{j}^{m'} \alpha_i \beta_j k(x_i, x'_j)$$

Now we try to find the condition on kernel k, such that f belongs to Hilbert space so that we can define norm of f.
Symmetry can be seen as follows:

$$\langle f, g \rangle = \sum_{j=1}^{m'} \beta_j f(x'_j) = \sum_{i=1}^{m} \alpha_i g(x_i)$$

This implies $\langle f, g \rangle = \langle g, f \rangle$ if $k(x_i, x_j) = k(x_j, x_i)$.
Positive definiteness can be seen as follows:

$$\langle f, f \rangle = \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j) \geq 0 \; \forall \alpha_i, \alpha_j \in \mathcal{R}$$

This property holds true when the kernel matrix K is positive semi-definite.
Similarly, linearity is also true without any further assumption on the kernel. Hence $\|f\|^2 = \langle f, f \rangle = \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j) = \|w\|^2$. Hence we can substitute $\|w\|^2$ with $\|f\|^2$ with the given properties of K. Hence our generalised loss function becomes

$$\min_f R(\|f\|) + L(f(x_1), f(x_2), ..., f(x_n))$$

Note: If $\forall x |f(x)| \leq M_x \|f\|_H$ then $\exists f(x) = \sum_i \alpha_i k(x_i, x)$

5.3 Need for such substitution

If $\phi(x)$ has a very large or ∞ dimension, it is impossible to code w as it has the same dimension as $\psi(x)$. So we can either go with the kernel matrix or make our analysis on the prediction function, both of which are independent of the dimension of $\phi(x)$. This is a useful tool for analysing the correctness of RBF kernel where $\phi(x)$ is of infinite dimension.

6 Reproducing kernel Hilbert spaces

For a Hilbert space $\mathcal{H}$ of real-valued functions on $\mathcal{X}$, and for any point $x \in \mathcal{X}$, the evaluation functional at x is defined as the map $L_x : \mathcal{H} \mapsto \mathbb{R}$ such that for all functions $f \in \mathcal{H}$,

$$L_x(f) = f(x). \tag{4}$$

In this setting, $\mathcal{H}$ is called a reproducing kernel Hilbert space if for all $x \in \mathcal{X}$, L_x is bounded, i.e. there is some finite constant M such that

$$|L_x(f)| = |f(x)| \leq M \|f\|_{\mathcal{H}}. \tag{5}$$

(Equivalently, for all $x \in \mathcal{X}$, L_x is continuous at any $f \in \mathcal{H}$.)

7 Example problem

7.1 Problem Statement

Consider the functions $h : \mathbb{N} \to [1 \dots m]$ and $\mathcal{E} : \mathbb{N} \to \pm 1$

$$a^{h,\mathcal{E}}(x)[i] = \sum_{j \text{ s.t } h(i)=j} \mathcal{E}(j) x_j$$

Then prove that

$$\mathop{\mathbb{E}}_{h,\mathcal{E}\sim\ \mathcal{U}(.)}[\langle a^{h,\mathcal{E}}(x), a^{h,\mathcal{E}}(x')\rangle] = \langle x, x'\rangle$$

7.2 Solution

$$\mathop{\mathbb{E}}_{h,\mathcal{E}\sim\ \mathcal{U}(.)}[\langle a^{h,\mathcal{E}}(x), a^{h,\mathcal{E}}(x')\rangle] = \mathop{\mathbb{E}}_{h,\mathcal{E}\sim\ \mathcal{U}(.)}[\sum_{j;h(i)=j} \sum_{j';h(i)=j'} \mathcal{E}(j)\mathcal{E}(j') x_j x'_{j'}]$$

Note that since h and $\mathcal{E}$ are sampled from uniform distributions, for $j \neq j'$

$$\mathbb{E}[\mathcal{E}(j)\mathcal{E}(j')] = 0$$
$$\text{and when } j = j', \mathbb{E}[\mathcal{E}(j)\mathcal{E}(j)] = 1$$

Therefore the expectation simplifies to

$$\mathop{\mathbb{E}}_{h,\mathcal{E}\sim\ \mathcal{U}(.)}[\sum_{j=j'}(1) * x_j x'_{j'} + 0] = \mathbb{E}[\sum_j x(j)x'(j)] = \mathbb{E}[\langle x, x'\rangle] = \langle x, x'\rangle$$

8 Mercer's Theorem

Theorem 8.1. *A "symmetric" function $k(x, x')$ can be expressed as an inner product*

$$k(x, x') = \langle \psi(x), \psi(x')\rangle$$

for some ψ if and only if K (kernel matrix) is positive semi-definite (and symmetric).

AIML - CS 337

Lecture 15: Kernels and Gaussian Processes

6th October 2022

Lecturer: Abir De *Scribe: Group 31 and Group 32*

1 Prologue

By now, we have studied various kernel tricks which can be for separating data having non-linear relationship by simply defining an appropriate Gram matrix representing the kernel. Further, the trick can be extended to non parametric regression[2], classification and PCA(kernel PCA[1]) as well. In this lecture we look at another application of kernels in the context of Gaussian Processes and how to deal with smaller training sets to still give fair results.

2 Problem

Consider the standard linear regression model as follows:

$$w^{\text{regression}} \rightarrow \min \left[\sum_{i \in D} (y_i - w^T x_i)^2 \right]$$

The solution to the above problem is:

$$w^{\text{regression}} = (\sum_{i \in D} x_i x_i^T)^{-1} \cdot (\sum_{i \in D} x_i y_i)$$

The predictions are made using function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, $f(x_i) = w^T \cdot x_i$ Notice that when we substitute an input data from training set,

$$f(x_i) \neq y_i$$

An alternative approach to this could be to obtain a distribution on the function we are trying to predict such that every point in the training data must have exactly the same output in the hypothesis as the training label. More precisely, we would like to design a non linear estimator f to model the training data with the additional restriction that $\forall x_i \in D \; f(x_i) = y_i$; for the other points $x \notin D$, $f(x)$ is a random variable with an associated probability distribution, while having certain guarantees on accuracy on test set and assuming train and test set are from same distribution. We can visualise such a function as shown in the figure below:

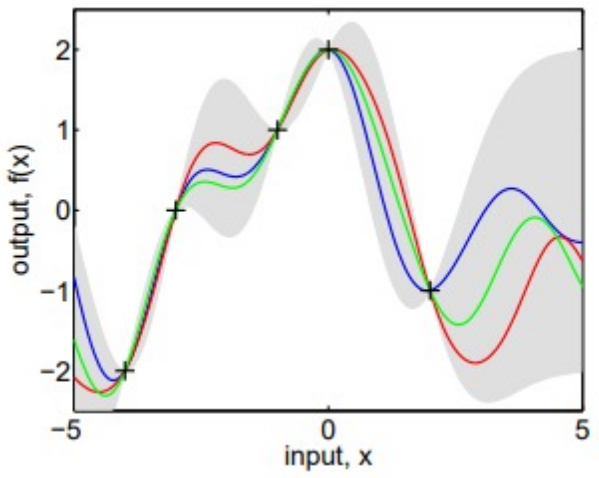


Figure 1: Graphical Representation

Here in this figure you can see that the points marked as + are the points in our dataset, for which the output is exactly one value while it is a distribution (as given by the shaded area) for all the other points

3 Gaussian Process

Gaussian processes are a method for non parametric estimation to provide confidence on the seen data and some kind of distribution on unseen data. For any subset of the training data, we must have that the joint prior distribution of this subset is normally distributed for some mean and covariance matrix. For any subset $\{x_1...x_m\}$ of the training data, the prior distribution follows:

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \quad \sim \quad \mathcal{N}(\vec{\mu}(x_1, \ldots, x_m), \Sigma(x_1, \ldots, x_m))$$

where $\vec{\mu}$ and Σ are deterministic functions.

On introducing a new data point into any subset of the training data, we expect the resulting conditional distribution to also follow the normal distribution. For the data point x^*

$$f(x^*)|(f(x_1), \ldots f(x_m), x^*) \sim \mathcal{N}(\vec{\mu}(x_1, \ldots, x_m, x^*), \Sigma(x_1, \ldots, x_m, x^*))$$

As described earlier, we expect that if a new data point introduced is already in the training data, then we expect that $f(x^*)$ takes the value that was present in the training set. This means that for any x^* such that $x^* \in \{x_1, \ldots, x_m\}$

$$f(x^*)|(f(x_1), \ldots f(x_m), x^*) \sim \mathcal{N}(f(x^*), 0)$$

Our aim is to design a matrix Σ that satisfies such a property i.e. posterior for any point in training data must have zero variance.

Let $X_D = [X_1^d X_2^d \dots X_n^d]^T$ denote the points in train set and $X_T = [X_1^t X_2^t \dots X_n^t]^T$ denote the points in test set. $f(X_D)$ and $f(X_T)$ denote random variables depending on the input.

$$f(X) = \begin{pmatrix} f(X_1^d) \\ f(X_2^d) \\ . \\ . \\ f(X_n^t) \end{pmatrix}$$

These random variables are dependent on each other, and we need to model the dependency between them. We model $f(X)$ as multi-variate Gaussian distribution. Therefore,

$$\begin{bmatrix} f(X_D) \\ f(X_T) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X_D) \\ m(X_T) \end{bmatrix}, \begin{bmatrix} k(X_D, X_D) & k(X_D, X_T) \\ k(X_D, X_T)^T & k(X_T, X_T) \end{bmatrix} \right)$$

Here, $m(\cdot)$ is a function denoting mean, and $k(\cdot, \cdot)$ is a kernel function used for creating the covariance matrix. The reason we are using kernel function here is because we want to model some sort of similarity between the random variables, high correlation implying higher similarity. With this model, we can determine the prior distribution of the random variables $f(\cdot)$

$$P(f(X)) = P\left(\begin{bmatrix} f(X_D) \\ f(X_T) \end{bmatrix} \right) = \frac{1}{\text{(some constant)} \cdot \det(K)^{0.5}} \exp(-0.5 f(X)^T \Sigma^{-1} f(X))$$

Our aim now is to get distribution of $f(X_T)$ given the train predictions $\{y_i\}$ and X_T. for which we can use Bayes' rule.

3.1 Evaluating the posterior

We would model the predictions $Y = [y_1 y_2 \dots y_n]^T$ as

$$Y = I \cdot f(X_D)$$

A more general model would be to include additive Gaussian noise such that $Y = f(X_D) + \eta$. But for simplicity, we assume noise to be zero. Before the actual derivation, we note two important results.

3.1.1 Gaussian Marginalisation Rule

If we marginalize out variables in a multivariate Gaussian distribution, the result is still a Gaussian distribution. Mathematically, if $X = [X_1, X_2, \dots X_n]^T$ is a multi-variate Gaussian random variable($\sim \mathcal{N}(\mu, \sigma)$), then any subset of X is a multi-variate Gaussian and the mean and covariance is given by $(A\mu, A\sigma A^T)$. A can be constructed by using e_i^T as rows. For example, if $n = 3$, and the subset is constructed using $[X_1\ X_3]$, then,

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3.1.2 Conditional Rule for multi-variate Gaussian

Intuitively, if we start with a Gaussian distribution and update our knowledge given the observed value of one of its components(that is, find conditional probability distribution), then the resulting distribution is still Gaussian! Mathematically,
Let $[x\ y]$ jointly form multi variate Gaussian random variable,

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix} \right)$$

Here Σ_{ab} represents covariance matrix between random vectors a and b and $f(\cdot)$ represents the PDF.

$$f(x|y) = \frac{f(x,y)}{f(y)}$$

Now, we will substitute $f(x,y)$ with the expression for multi-variate Gaussian distribution($\mathcal{N}(\mu, \Sigma)$), and $f(y)$ with $\mathcal{N}(\mu_y, \Sigma_{yy})$. Simplifying the equations, we get

$$f(x|y) = \mathcal{N}(\Sigma_{xy}\Sigma_{yy}^{-1}y, \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx})$$

(μ is assumed to be zero for simplicity)
To get the detailed derivation, please see [3]

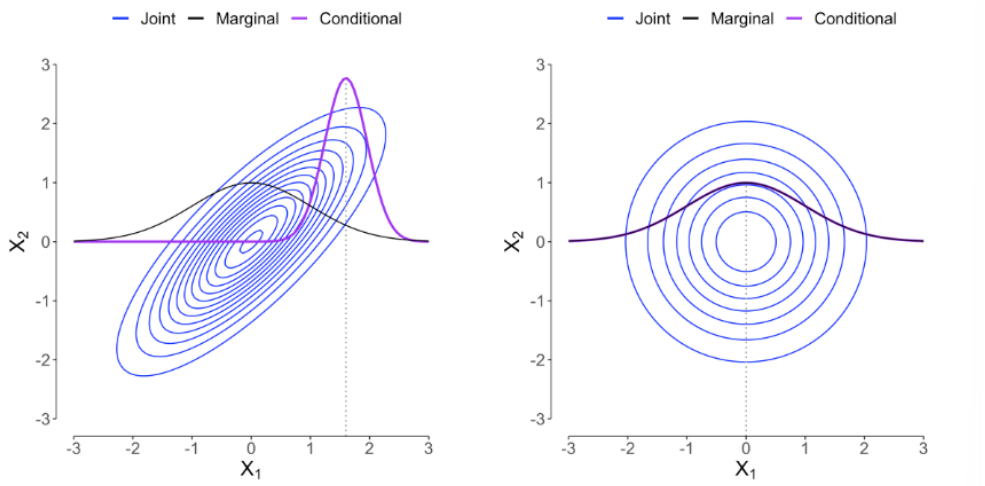


Figure 2: Joint, Marginal, Conditional for bivariate Gaussian. Source[3]

3.2 Getting to the posterior

We modelled $Y = f(X_D)$, now's the time to use it.

$$\begin{bmatrix} f(X_T) \\ Y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(X_T) \\ \mu(X_D) \end{bmatrix}, \begin{bmatrix} k(X_T, X_T) & k(X_T, X_D) \\ k(X_T, X_D)^T & k(X_D, X_D) \end{bmatrix} \right)$$

Using the conditional rule described above,

$$P(f(X_D)|Y) = \mathcal{N}(\mu_{posterior}, \sigma^2_{posterior})$$

where

$$\mu_{posterior} = \mu X_T + k(X_T, X_D)(k(X_D, X_D))^{-1}(Y - \mu(X_D))$$
$$\sigma^2_{posterior} = k(X_T, X_T) - k(X_T, X_D)(k(X_D, X_D))^{-1}k(X_T, X_D)^T$$

4 Aftermath[1]

.

4.1 Posterior mean

For the sake of investigation, let's assume $\mu(\cdot) = 0$.
Now, if $X_D = X_T$, $\mu_{posterior} = Y$ which is desirable because we want the mean for predictions at training points to be the same as the given predictions in train set. The mean value at a single test location, say x_i^t, is a weighted sum of all the observations Y. The weights are defined by the kernel between the test location x_i^t and all training locations in X.

4.2 Posterior variance

Observe that if $X_D = X_T$, we get $\sigma_{posterior} = 0$. This means that the prediction for a point in train set is exactly the mean, which in turn is the Y of the training set.

5 Conclusion

We started with a train set $\{(x_i^d, y_i^d)\}$ and test inputs $\{x_i^t\}$, and devised a function that would yield no error for inputs which are in train set, and low errors on other points. The described method is particularly useful for low data situations. A detailed study of Gaussian processes can be found in the reference [4].

[1] pun intended (credits Group 31)

References

[1] *Kernel PCA*. URL: https://www.geeksforgeeks.org/ml-introduction-to-kernel-pca/.

[2] *Non-parametric regression*. URL: https://en.wikipedia.org/wiki/Nonparametric_regression.

[3] *Properties of multi-variate Gaussian*. URL: https://fabiandablander.com/statistics/Two-Properties.html.

[4] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

AIML - CS 337

Lecture 16: Mixture Models

10th October 2022

Lecturer: Abir De *Scribe: Group 33, Group 34*

1 Prologue

Mixture models are useful when we have a large dataset with heterogenous data.
Typical models like regression assume that data is generated by adding noise to a central distribution and is centered around a single mode. However, we can have multimodal spatial characteristics as well, like :

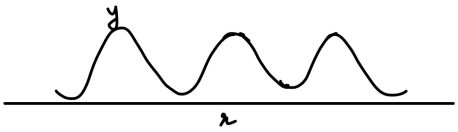


Figure 1: Multi modal spatial characteristics

We will be discussing problems where we are given a dataset and are asked to partition it into a (given) certain number of classes, taking some similarity measure(s) or probabilistic models as a basis. It's the latter part where mixture models come into picture!
We will first investigate the former version a bit and then move on to mixture models.

2 Problem Statement

Given a dataset D, segregate it into k clusters $\{C_1, C_2 \cdots C_k\}$. More formally (and for the sake of introducing notations for the rest of the document)

$$\text{Given a dataset, } D = \{x_1, x_2, \ldots, x_n\} \text{ of } n \text{ points}$$

$$\text{Our task is to find a labelling function } z : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, k\}$$

Here, $z(i) = j$ means that the point x_i belongs to the cluster C_j. For convenience, we will often write z_i to denote $z(i)$.
Now that we have defined our problem,let us approach it in a couple of settings.

3 Objective functions

One recurrent theme in machine learning is to view our model as a function optimizing a reasonable objective function in a reasonable hypothesis class. Throughout this section, our hypothesis class for z will be simply the set of all functions from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, k\}$. As for the objective functions, here are a few candidates proposed in class:

Maximising pair wise distances between points belonging to different clusters

$$\max_{z} \sum_{\substack{i,j \\ z(i) \neq z(j)}} ||x_i - x_j||$$

The intuition behind this optimization problem is to in a sense keep different cluster points as far as possible in the Euclidean space. This function for any z is a bit expensive to compute as We have to iterate over all $\binom{k}{2}$ pairs of clusters and compute sum of distances between all the pair of points in which one point lies in one cluster and the other in other.

Minimizing pair wise distances belonging to same cluster

$$\min_{z} \sum_{\substack{i,j \\ z(i) = z(j)}} ||x_i - x_j||^2$$

The intuition here is to minimize the total sum of pairwise distances of points belonging to the same cluster and thus in a way ensuring that the points in the same cluster are as close to each other as possible. Call this problem P_0.

Point-wise distances instead of pair-wise distances

$$\min_{\substack{z \\ \bar{x_1}, \bar{x_2}, \ldots, \bar{x_k}}} \sum_{c=1}^{c=k} \sum_{z_i = c} ||x_i - \bar{x_c}||^2$$

The intuition behind this is to think of clusters as being identified by the points $\bar{x_c}$ and trying to make all the points as close to the representative point of the cluster they belong to.
It actually turns out that $\bar{x_c}$ must be the mean of points lying in cluster C_c since sum of squares of distances from a point to all the points of a given finite set of points is minimized (uniquely) at their mean! Call this optimization problem P_1.

Going from hard constraints to soft constraints

$$\min_{\mu_1, \mu_2, \ldots, \mu_n} \sum_{c=1}^{c=k} \sum_{i,j} \mu_{ic} \mu_{jc} ||x_i - x_j||^2$$

Here, μ_i's are probability distributions over the set $\{1, 2, \ldots, k\}$ and μ_{ic} denotes the probability of point i belonging to cluster c.

Note that for this objective function, our focus is to attach probabilities to a point belonging to a cluster instead of deterministically assigning clusters to points.
The intuition behind this objective function is to find those μ_i's for which the expected value of sum of squares of distances between points of same cluster is as small as possible and thus in a way, points of same cluster are $expectedly$ as close to each other as possible.
Call this problem P_2.

$$\min_{\substack{\mu_1,\mu_2,\ldots,\mu_n \\ \bar{x_1},\bar{x_2},\ldots,\bar{x_n}}} \sum_{c=1}^{c=k} \sum_{i=1}^{i=n} \mu_{ic} ||x_i - \bar{x_c}||^2$$

Here, the meaning of μ_i's the same as in P_2. $\bar{x_i}$'s can be viewed as representatives of clusters as in P_1. The intuition behind this problem is that we want to make the points as close to their cluster representatives as possible, in expectation.
Call this problem P_3.

Note that it is linear in μ_{ic} and hence at optimization condition it will lie at the boundary, i.e. it will either be 0 or 1. We also know that for a fixed i, $\sum_c \mu_{ic} = 1$. Hence, at the optimization condition, for a fixed i, μ_{ic} will be 1 for a particular c_0 and 0 for the rest. This can be interpreted as point i belonging to cluster c_0.

$$\mu_{ic} = \begin{cases} 1, & \text{if } c = \arg\min_{c'} ||x_i - \bar{x_{c'}}||^2 \\ 0, & \text{otherwise} \end{cases}$$

We therefore do the optimization in two steps where we first fix $\bar{x_c}$ making it a linear optimization problem owing to the above condition. Followed by which we optimize for the mean values $\bar{x_c}[\forall c \in 1, 2 \ldots k]$.

Hence we can say that at optimal condition $P_1 \equiv P_3$! Note however that we can't say $P_0 \equiv P_2$ at optimal since it is a case of quadratic optimization and we can't say μ will hit boundary conditions.

Also, in (5), at optimal, $\bar{x_c} = \sum_{i \in c} x_i / n_c$. This is because the optimal condition for $\min_{x_c} \sum ||x_i - \bar{x_c}||^2$ reduces to x_c being the mean of the points.

Following is the k-means algorithm for optimization of the objective function P_0, i.e., without prior probability of distributions of the cluster buckets.

Algorithm 1: The k-Means Clustering Algorithm

input: $\chi \subset \mathbb{R}^n$; Number of clusters k
initialize: Randomly chosen initial centroids $\mu_1, \ldots, \mu_k$
repeat until convergence
$\forall i \in [k]$ set $C_i = \{x \in \chi : i = argmin_j ||x - \mu_j||\}$
(break ties in some arbitrary manner)
$\forall i \in [k]$ update $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

Now, we will change our focus to probabilistic models.

4 Probabilistic setting

In this section, we will look at this problem from a probabilistic sampling perspective. We will assume that the data points $x_1, x_2, \ldots, x_n$ are respectively sampled from iid instances $X_1, X_2, \ldots, X_n$ of a random variable X which can be thought of as some sort of a mixture of k random variables $Y_1, Y_2, \ldots, Y_k$ (and hence the term, mixture models) over the sample space from which D is sampled. Sampling from X is done as follows:- first we pick a random number c from a probability distribution π over $\{1, 2, \ldots, k\}$ and then pick a random number x from Y_c and associate x with the cluster c. The Y_i's are generally assumed to be coming from the same class of distribution but with different parameters. Let θ be the set of parameters parametrising Y_i's and π.
Observe that π_c(short for $\pi(c)$) will be the probability of a random point sampled from X belonging to Cluster C_c. Also, let Z_i be the random variable describing the cluster to which x_i belongs to and define π_{ic} as the probability that x_i belongs to cluster c, i.e. $\pi_{ic} = P(Z_i = c|X_i = x_i, \theta)$. Using Baye's rule, we can write π_{ic} as:

$$\begin{aligned}
\pi_{ic} &= P(Z_i = c|X_i = x_i, \theta) \\
&= \frac{P(X_i = x_i|Z_i = c, \theta)P(Z_i = c|\theta)}{P(X_i = x_i)} \\
&= \frac{P(X_i = x_i|Z_i = c, \theta)P(Z_i = c|\theta)}{\sum_{j=1}^{j=k} P(X_i = x_i|Z_i = j, \theta)P(Z_i = j|\theta)} \\
&= \frac{P(Y_c = x_i|\theta)\pi_c}{\sum_{j=1}^{j=k} P(Y_j = x_i|\theta)\pi_j}
\end{aligned} \tag{1}$$

Now that we have our probabilistic model ready, we can can apply a host of probabilistic tools like MLE, Bayesian inference, etc to estimate θ.
The likelihood function will be:

$$\begin{aligned}
\mathcal{L}(\theta|\vec{X} = \vec{x}) &= P(\vec{X} = \vec{x}|\theta) \\
&= \sum_{\vec{z} \in \{1,2,\ldots,k\}^n} P(\vec{X} = \vec{x}|\vec{Z} = \vec{z}, \theta)P(\vec{Z} = \vec{z}|\theta) \\
&= \sum_{\vec{z} \in \{1,2,\ldots,k\}^n} \prod_{i=1}^{i=n} (P(X_i = x_i|Z_i = z_i, \theta)P(Z_i = z_i|\theta)) \\
&= \sum_{\vec{z} \in \{1,2,\ldots,k\}^n} \prod_{i=1}^{i=n} (P(Y_{z_i} = x_i|\theta)\pi_{z_i} \\
&= \mathbb{E}_{\vec{Z}} P(Y_{z_i} = x_i|\theta)
\end{aligned} \tag{2}$$

Note that here $\vec{x} = \{\boldsymbol{x}_1, x_2, \ldots, x_n\}$ and similarly other vector notations are defined. We can maximize the above likelihood function to obtain maximum likelihood estimate for θ and one possible approach to cluster can be to assign that cluster c to a point x_i ,for which π_{ic} is maximum as per the estimated θ.

AIML - CS 337

Lecture 17: Regression

15-10-2022

Lecturer: Abir De *Scribe: Group 35 & 36*

We have already discussed about Gaussian Processes in the previous lectures. In this lecture we continue on it and draw comparisons between Linear Regression which is parametric and Gaussian Process which is not parametric. At the end of the lecture we make use of Gaussian Processes to improve upon K-Means Clustering Algorithm. To begin with let us recall Linear Regression

1 Linear Regression

Consider the dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in R^d, y_i \in R$. On applying Linear Regression on this dataset we get loss as follows

$$L(\boldsymbol{\omega}) = \sum_{i=1}^{N} (y_i - \boldsymbol{\omega}^T x_i)^2 + \lambda ||\boldsymbol{\omega}||^2$$

In vectorized form we can write loss as

$$L(\boldsymbol{\omega}) = ||\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\omega}||^2 + \lambda ||\boldsymbol{\omega}||^2$$

where $\boldsymbol{y} \in R^N, \boldsymbol{X} \in R^{d\times N}, \boldsymbol{\omega} \in R^d$. On minimizing the loss function we get

$$\begin{aligned}
\nabla_{\boldsymbol{w}} L(\boldsymbol{\omega}) &= 0 \\
\nabla_{\boldsymbol{w}} ((\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\omega})^T(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\omega}) + \lambda\boldsymbol{\omega}^T\boldsymbol{\omega}) &= 0 \\
\nabla_{\boldsymbol{w}} (\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}^T\boldsymbol{\omega} - \boldsymbol{\omega}^T\boldsymbol{X}\boldsymbol{y} + \boldsymbol{\omega}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{\omega} + \lambda\boldsymbol{\omega}^T\boldsymbol{\omega}) &= 0 \\
-2\boldsymbol{X}\boldsymbol{y} + 2\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{\omega} + 2\lambda\boldsymbol{\omega} &= 0 \\
(\boldsymbol{X}\boldsymbol{X}^T + \lambda I)\boldsymbol{\omega} &= \boldsymbol{X}\boldsymbol{y}
\end{aligned}$$

Proof for invertiblity of $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ when $\lambda > 0$
Consider

$$\begin{aligned}
\boldsymbol{v}^T(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\boldsymbol{v} &= \boldsymbol{v}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v} + \lambda\boldsymbol{v}^T\boldsymbol{v} \\
&= ||\boldsymbol{X}\boldsymbol{v}||^2 + \lambda||\boldsymbol{v}||^2 \\
&> 0 \ \text{ when} ||\boldsymbol{v}|| > 0
\end{aligned}$$

Thus $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is a positive definite matrix and hence invertible.
Similarly proof for invertibility of $\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I}$ can be done and we will use that in a later section.

When $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is invertible we have

$$\boldsymbol{\omega} = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$$
$$y_{pred}^{LR} = \mathbf{x}_*^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$$

2 Bayesian Linear Regression

We have already seen how to obtain the probability distribution $P(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y})$ in the previous lecture on Gaussian Process. Although this was not discussed in class, this alternate way would help us relate Gaussian Process with Linear Regression. Let f($\mathbf{x}$) = $\mathbf{x}^T\mathbf{w}$ and $\mathbf{y} = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$.

$$P(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{x_i}, \mathbf{w}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{(y_i - \mathbf{x}_i^T\mathbf{w})^2}{2\sigma_n^2}}$$
$$= \frac{1}{(2\pi\sigma_n^2)^{\frac{n}{2}}} e^{-\frac{|\mathbf{y} - \mathbf{X}^T\mathbf{w}|^2}{2\sigma_n^2}} = \mathcal{N}(\mathbf{X}^T\mathbf{w}, \sigma_n^2 I)$$

Consider a prior on $\mathbf{w}$ as $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$

$$\text{posterior} = \frac{\text{likelihood} \times \text{ prior}}{\text{marginal likelihood}} \qquad P(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{P(\mathbf{y}|\mathbf{w}, \mathbf{X})P(\mathbf{w})}{P(\mathbf{y}|\mathbf{X})}$$

Note that $P(\mathbf{y}|\mathbf{X})$ is independent of $\mathbf{w}$. Thus we have

$$P(\mathbf{w}|\mathbf{X}, \mathbf{y}) \propto e^{-\frac{(\mathbf{y} - \mathbf{X}^T\mathbf{w})^T(\mathbf{y} - \mathbf{X}^T\mathbf{w})}{2\sigma_n^2}} e^{-\frac{\mathbf{w}^T\Sigma_p^{-1}\mathbf{w}}{2}}$$
$$\propto e^{-\frac{(\mathbf{w} - \overline{\mathbf{w}})^T(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1})(\mathbf{w} - \overline{\mathbf{w}})}{2}}$$

Let $A = \frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1}$ and $\overline{\mathbf{w}} = \frac{1}{\sigma_n^2}(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1})^{-1}\mathbf{X}\mathbf{y}$

$$P(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \mathcal{N}(\frac{1}{\sigma_n^2}A^{-1}\mathbf{X}\mathbf{y}, A^{-1})$$
$$P(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int P(f_*|\mathbf{x}_*, \mathbf{w})P(\mathbf{w}|\mathbf{X}, \mathbf{y})d\mathbf{w}$$
$$= \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^T A^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^T A^{-1}\mathbf{x}_*)$$

Note that $\overline{\mathbf{w}} = \frac{1}{\sigma_n^2}(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1})^{-1}\mathbf{X}\mathbf{y} = (\mathbf{X}\mathbf{X}^T + \sigma_n^2\Sigma_p^{-1})^{-1}\mathbf{X}\mathbf{y}$. Comparing it with the result of first section we obtain

$$\boxed{\lambda I = \sigma_n^2\Sigma_p^{-1}}$$

2.1 Bayesian Linear Regression to Gaussian Regresion

Going to higher dimensional space, we can just replace $\mathbf{x}$ by $\phi(\mathbf{x})$. Define $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\Sigma_p\phi(\mathbf{x})$. As Σ_p is positive definite we can find a symmetric matrix $\Sigma_p^{1/2}$ (using SVD) so that $(\Sigma_p^{1/2})^2 = \Sigma_p$. Define $\psi(\mathbf{x}) = \Sigma_p^{1/2}\phi(\mathbf{x})$ Thus $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x}).\psi(\mathbf{x}')$. Also let $\mathbf{k}_* = [\mathbf{k}(\mathbf{x}_*, \mathbf{x}_1), \mathbf{k}(\mathbf{x}_*, \mathbf{x}_2), ...\mathbf{k}(\mathbf{x}_*, \mathbf{x}_N)]^T$. Substituting the above values in the obtained formula of $P(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y})$ along with some mathematical manipulations [1] we obtain

$$P(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}\mathbf{y}, \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}\mathbf{k}_*)$$

This is the exact same equation which we had obtained earlier for Gaussian Processes.

3 Gaussian Processes

Recall that in Gaussian Processes we have the following

$$P(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}\mathbf{y}, \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}\mathbf{k}_*)$$
$$y_{pred}^{GP} = \mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}\mathbf{y}$$

where $\mathbf{k}_* = [\mathbf{k}(\mathbf{x}_*, \mathbf{x}_1), \mathbf{k}(\mathbf{x}_*, \mathbf{x}_2), ...\mathbf{k}(\mathbf{x}_*, \mathbf{x}_N)]^T$. Also observe that if $\sigma_n = 0$ for a $\mathbf{x}_* = \mathbf{x}_i \in D$ the value of $\mathbf{k}_*^T(K + \sigma_n^2 I)^{-1}$ is a row vector with all values zero except the i^{th} index which has value one (Think in terms of matrix multiplication of KK^{-1} and focus on the i^{th} row of the output). Thus the mean of the distribution(prediction) for a point in training dataset is the true label and the variance at that point is zero. Note that y_{pred}^{GP} is a linear combination of observations $\mathbf{y}$. Another way to look at this equation is to see it as a linear combination of N kernel functions, each one centered on a training point, by writing

$$y_{pred}^{GP} = \sum_{i=1}^{N} \alpha_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}_*)$$

This equation can be arrived at pretty simply.

$$\begin{aligned}
\mathbf{k}_*^T &= [\mathbf{k}(\mathbf{x}_*, \mathbf{x}_1), \mathbf{k}(\mathbf{x}_*, \mathbf{x}_2), ...\mathbf{k}(\mathbf{x}_*, \mathbf{x}_N)] \\
\boldsymbol{\alpha} &= (K + \sigma_n^2 I)^{-1}\mathbf{y} \\
y_{pred}^{GP} &= \mathbf{k}_*^T \boldsymbol{\alpha} \\
&= \sum_{i=1}^{N} \alpha_i \mathbf{k}(\mathbf{x}_i, \mathbf{x}_*)
\end{aligned}$$

4 Linear Regression with $\lambda = 0$

. Without regularisation the solution of Linear Regression looks like

$$y_{pred}^{LR} = \mathbf{x}_*^T (\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{y}$$

For $\sigma_n = 0$ and kernel $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}.\mathbf{x}'$ we have

$$\begin{aligned}
y_{pred}^{GP} &= \mathbf{k}_*^T (K + \sigma_n^2 \boldsymbol{I})\boldsymbol{y} \\
&= \mathbf{x}_*^T \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{y}
\end{aligned}$$

Normally we don't expected Linear Regression to completely fit the training dataset whereas we know that Gaussian Process(with $\sigma_n = 0$) fits the training dataset completely. Let us try if we can show that $y_{pred}^{LR} = y_{pred}^{GP}$. In class we gave the following argument.

$$\begin{aligned}
\mathbf{X}(\mathbf{X}^T\mathbf{X}) &= (\mathbf{X}\mathbf{X}^T)\mathbf{X} \\
(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}$$

The second equation is obtained by multiplying first equation with $(\mathbf{X}\mathbf{X}^T)^{-1}$ from left and $(\mathbf{X}^T\mathbf{X})^{-1}$ from right. Thus the second equation shows the equivalence of y_{pred}^{LR} and y_{pred}^{GP}. But the above derivation would be valid only if both $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T\mathbf{X}$ are invertible which requires that d=N and $\mathbf{X}$ is invertible. This is definetly not a great achievement. We show a more stronger result below which shows the equality of both if $d >> N$.

There is a high probability that $\mathbf{X}$ (which is a $d \times N$ matrix) is rank N i.e. there is a high probability we get N linearly independent vectors out of d vectors($d >> N$). If $\mathbf{X}$ is rank N , then $\mathbf{X}^T\mathbf{X}$ would be rank N and invertible (Tutorial 1 Problem 11)

Proof : $\mathbf{X}$ is rank N. This is equivalent to saying $\mathbf{X}\mathbf{v} = \mathbf{0} \iff \mathbf{v} = \mathbf{0}$. To show that $\mathbf{X}^T\mathbf{X}$ is invertible , we need to show that its Null Space is $\{\mathbf{0}\}$.

$$\begin{aligned}
\mathbf{X}^T\mathbf{X}\mathbf{v} &= 0 \\
\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} &= 0 \\
(\mathbf{X}\mathbf{v})^T(\mathbf{X}\mathbf{v}) &= 0 \\
||\mathbf{X}\mathbf{v}|| &= 0 \\
\mathbf{X}\mathbf{v} &= 0 \\
\mathbf{v} &= 0
\end{aligned}$$

Thus $\mathbf{X}^T\mathbf{X}$ is full rank and invertible. From the first section we know that $\mathbf{X}\mathbf{y} = \mathbf{X}\mathbf{X}^T\mathbf{w}_*$

$$\begin{aligned}
\mathbf{X}\mathbf{y} &= \mathbf{X}\mathbf{X}^T\mathbf{w}_* \\
\mathbf{X}^T\mathbf{X}\mathbf{y} &= \mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{w}_* \\
(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\mathbf{y} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{w}_* \\
\mathbf{y} &= \mathbf{X}^T\mathbf{w}_*
\end{aligned}$$

The equation above would have many solutions for $\mathbf{w}_*$ (The dimension of Null Space of $\mathbf{X}^T$ is d-N). Consider a particular solution : $\mathbf{w} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{y}$
Thus $y^{LR}_{pred} = \mathbf{x}_*^T\mathbf{w} = \mathbf{x}_*^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{y} = y^{GP}_{pred}$

d is a rough proxy for capacity of the model. Thus for large d Linear Regression will interpolate(overfit) and memorize all points in the training dataset and thus for large d Linear Regression performs similar to Gaussian Process.

Summarizing the discussion above if $\lambda = 0$ and ($d >> N$ or ($d \to \infty$)) Linear Regression is same as Gaussian Process,i.e. Linear Regression on infinite space is same as Gaussian Process . If $\lambda = 0$ and $d < N$ the model does not have enough capacity to memorize all the points and hence Linear Regression is not equal to Gaussian Process.

5 Linear Regression with $\lambda > 0$

From the first section, we already know

$$y^{LR}_{pred} = \mathbf{x}_*^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$$

For $\sigma_n^2 = \lambda$ and kernel $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}.\mathbf{x}'$ we have

$$\begin{aligned}
y^{GP}_{pred} &= \mathbf{k}_*^T(K + \sigma_n^2\boldsymbol{I})\boldsymbol{y} \\
&= \mathbf{x}_*^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda I)^{-1}\boldsymbol{y}
\end{aligned}$$

Now we will show that $y_{pred}^{LR} = y_{pred}^{GP}$ for the given case

$Proof :-$ Using **X** = **X**I = I**X** for all matrices **X**

$$\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}) = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})\boldsymbol{X}$$

From our earlier discussion we know that both $(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})$ and $(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})$ are invertible. Multiying $(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}$ from right and $(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}$ from left, we will get

$$(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}$$

Thus we can write y_{pred}^{LR} as

$$\begin{aligned} y_{pred}^{LR} &= \mathbf{x}_*^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y} \\ &= \mathbf{x}_*^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} \\ &= y_{pred}^{GP} \end{aligned}$$

Thus we have shown that when regularised, even for finite dimension Linear Regression and Gaussian Process are equivalent.

6 Interpretation of λ

We know that $\lambda I = \sigma_n^2\Sigma_p^{-1}$. Thus λ is proportional to inverse of variance of **w** (of prior distribution) when there is no data. If the data is good and we put λ=0 it means that we have confidence on data and hence allow the model to choose from all **w** as posterior of **w** would have variance **0** (Dirac Delta). But when data is not enough(d comparable to N) we lack confidence on **w** and thus we set $\lambda > 0$ telling the model to pick **w** from a distribution thereby preventing overfitting on the training data.

6.1 Posterior of weights using GP

Gaussian Process Regression gives distribution of $f_* = f(\mathbf{x}_*) = \mathbf{x}_*^{\mathbf{T}}\mathbf{w}$ as:

$$Pr(f_*|\mathbf{x}_*, X, \mathbf{y}) = \mathcal{N}(\mathbf{k}_*{}^T(K + \sigma^2 I)^{-1}\mathbf{y}, k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*{}^T(K + \sigma^2 I)^{-1}\mathbf{k}_*) \tag{1}$$

where $k(.,.)$ is kernel function, $K(X, X_*)$ denotes $N \times N_*$ matrix of co-variances evaluated for every pair and $\mathbf{k}_*$ denotes the vector of co-variances between the test point and the N training points. We can get this expression through function-space view as described in [**?**].

Note Mean prediction is a linear combination of observations $\mathbf{y}$, this is sometimes referred to as a linear predictor. Another way to look at this equation is to see it as a linear combination of N kernel functions, each one centered on a training point, by writing

$$\bar{f}_* = \sum_{i=1}^{N} \alpha_i k(\mathbf{x_i}, \mathbf{x}_*) \tag{2}$$

where $\alpha = (K + \sigma^2 I)^{-1}\mathbf{y}$.

Now we want to compute posterior for weights i.e. $Pr(\mathbf{w}|X, \mathbf{y})$ using Gaussian Process Regression. But GPR gives us $Pr(\mathbf{x_*^T w}|\mathbf{x}_*, X, \mathbf{y})$. To obtained desired results, we can have $\mathbf{x}_* \in \{(\mathbf{1_i})_{i=1}^d\}$ where $\mathbf{1_i}$ is $d \times 1$ vector with all entries 0 except i^{th} entry which is 1. For $\mathbf{x}_* = \mathbf{1_i}$, we will get distribution for w_i. We can do the above evaluation in vectorized form also by giving input as I_d which is Identity matrix of size $d \times d$ (some manipulations must be done since input is changed from vector to matrix). Posterior of weights takes exact same form as equation **??** with $\bar{\mathbf{w}} = K(I_d, X)(K(X, X) + \sigma^2 I)^{-1}\mathbf{y}$ for $K(X, X) = X^T X$ and $\lambda = \sigma^2$. Prediction $\hat{y}$ using GPR is following which is same as that of LR:

$$\begin{aligned} \hat{y} &= \hat{\mathbf{x}}^T \bar{\mathbf{w}} = \hat{\mathbf{x}}^T X(X^T X + \sigma^2 I)^{-1}\mathbf{y} \\ \hat{y} &= \hat{\mathbf{x}}^T (XX^T + \lambda I)^{-1} X\mathbf{y} \end{aligned} \tag{3}$$

Homework Exercise Prove that predicted variance cannot be less than data variance in GPR.

6.2 Summarizing equivalence of LR and GP

$d < N, \lambda = 0$	Linear Regression $\neq$ Gaussian Process Regression
$d < N, \lambda > 0$	Linear Regression $=$ Gaussian Process Regression
$d \to \infty, \lambda = 0$	Linear Regression $=$ Gaussian Process Regression
$d \to \infty, \lambda > 0$	Linear Regression $=$ Gaussian Process Regression

Observation is that Linear Regression is equivalent to Gaussian Process Regression in all aspects except when $d < N, \lambda = 0$.

7 Is GP good enough?

If we have small training and test dataset, then should we go for Deep Learning or is GP good enough? Well, GP turns out be sufficient for such data sets, and DL is not required.

So, in which conditions GP will perform good/bad? GP works well if it gives low variance on test set for points close to training dataset points.

What if GP gives high variance? Then GP has no confidence, it is as good as no prediction. Solution is to combine K-Means Clustering and GP to enhance GP.

7.1 K-means clustering

Consider cluster of data points, to classify a new point $\mathbf{x}'$, we find $\mathbf{x}^*$ such that $\|\mathbf{x}' - \mathbf{x}\|$ is minimum across all $\mathbf{x_i}$ for $\mathbf{x}^*$. Predicted cluster y' for $\mathbf{x}'$ will be $y(x^*)$.

Note Gaussian Process tries to induce based on labels (target values) but K-means clustering operates on features (example values).

7.2 Combining Gaussian Process with Clustering

To summarise, for small number of points, rather than applying GP directly, we can do some operation and then perform GP on the dataset. We will be continuing discussion on this in further classes.

References

[1] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

AIML - CS 337

Lecture 18: Tutorial 2 (Gaussian Processes)

16 October, 2022

Lecturer: Abir De *Scribe:*

In this tutorial, we explore the theory behind Gaussian Processes, and solve a few problems based on them.

1 Basics of Bayesian Theory

Consider a Binomial Random Variable (i.e. an experiment involving repeated N Bernoulli trials) that denotes toss of a coin repeated N times. We report an estimate for the bias of the coin p as $\hat{p} = \mathbb{E}[\#\text{heads}]$. This is referred to as Maximum Likelihood Estimate. When N is small, this estimate is very poor and thus we see Bayesian theory that gives us reasonable estimates in low data regimes.

1.1 Prior

The concept of a prior distribution is that it gives us some additional information, or in a way, it is our initial belief, about the parameters involved in the distribution. For example, for the above mentioned problem, we would choose the prior to be a **Beta distribution** with parameters α and β chosen such that $\alpha = \beta = c$, where c is some constant, such that the mean of the prior distribution is $\frac{1}{2}$, and variance is some suitable fraction.

1.2 Likelihood

The likelihood tells us more about the data and the probability associated with the input instance, given the parameters (conditional probability). We wish to use the data to find (or more precisely, **estimate**) the parameters that better fit the data. To estimate the parameters, we use **Baye's rule** of conditional probability.

1.3 Estimation

We compute the posterior distribution by taking the product of the prior distribution with the likelihood, thus computing the conditional probability of parameters given the data. Mathematically:

$$P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)}$$

Here, θ represents the set of parameters. The LHS represents the **posterior**, the RHS numerator represents product of **likelihood** and **prior**, and the RHS denominator, which is primarily for normalisation (so that the distribution integrates to 1), represents the **marginal** on the data, which is essentially $\sum_{\theta} P(D|\theta) \cdot P(\theta)$. We then determine the estimate using the posterior distribution (maximum value estimation).

1.4 Degeneracy

If we have not seen much data, the estimate essentially falls back to the prior distribution. On the other hand, if there is lot of data, the posterior estimate tends to the data, and prior effectively is ignored. These degenerate cases also occur if there is extreme precision (variance tends to 0) or unreliability (variance is large) in the prior, because the estimate that we find varies roughly in the form:

$$\mu_{MAP} = \frac{N\mu_D\sigma_P^2 + \mu_P\sigma_D^2}{N\sigma_P^2 + \sigma_D^2}$$

Consider the cases when $N \to 0$, $N \to \infty$, $\sigma_P \to 0$ and $\sigma_P \to \infty$.

2 Gaussian & Multi-variate Gaussian

Analogous to the one-dimensional Gaussian distribution denoted by $X \sim \mathcal{N}(\cdot|\mu, \sigma)$, we have the multi-variate Gaussian distribution $X \in \mathbb{R}^n \sim \mathcal{N}(\cdot|\vec{\mu}, \Sigma^{n\times n})$, where $\Sigma^{n\times n}$ is the covariance matrix defined as $\Sigma(i,j) = cov(X_i, X_j)$. For parametrizing an MVG distribution, we use $m(X)$ as the estimate for μ and $K(x, x')$ as the estimate for Σ where the kernel function $K(\cdot,\cdot)$ is chosen keeping properties of the covariance matrix in mind (symmetric and positive semi-definite).

3 Gaussian Processes

We introduce the problem statement for fitting a Gaussian Process to a dataset. Given the training data $D = (\mathbf{x}_i, y_i)_{i=1}^N$, we make a few basic assumptions:

- $y = f(x) + \epsilon; \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$
- Prior $[f(\mathbf{x}_i), \ldots, f(\mathbf{x}_N)] \sim \mathcal{N}(0, K)$
- Target $[y_1, \ldots, y_N|\mathbf{f}] \sim \mathcal{N}(\mathbf{f}, \sigma^2)$

Theoretically, we consider that ϵ is **irreducible noise** that we do not model (not in our control). We assume a prior on the relation between x and y. The task, which is sampling values of the function f, reduces to conditional probability involving the data. The problem is thus essentially:

Question. For a test point $\mathbf{x}^*$, find the label y^*. Since noise is zero-mean, we will proceed to find $P(f^*|\mathbf{x}^*, D)$. The labels y^* will then follow $\mathcal{N}(\cdot|f^*, \sigma_\epsilon)$.

4 Determining $P(f^*|\mathbf{x}^*, D)$

To find the distribution $P(f^*|\mathbf{x}^*, D)$, we introduce f and then use Baye's rule:

$$\begin{aligned} P(f^*|\mathbf{x}^*, D) &= \int_{f_1,\ldots,f_N} P(f^*, f_1, \ldots, f_N|\mathbf{x}^*, D)d\mathbf{f} \\ &= \int_{\mathbf{f}} P(f^*|\mathbf{f}, \mathbf{x}^*, D) \cdot P(\mathbf{f}|\mathbf{x}^*, D)d\mathbf{f} \end{aligned}$$

4.1 Data Likelihood $P(\mathbf{f}|\mathbf{x}^*, D)$

Question. Can you remove some terms from the conditioning set?
Ans: Analysing the expression, we can conclude that for the function f, given the data D, x^* is not relevant. For determining $P(\mathbf{f}|D)$, we need $P(D|\mathbf{f})$ and then using the prior $P(\mathbf{f})$, we can find the data likelihood using Baye's rule. We have, $P(D|\mathbf{f}) = P(\{(x_i, y_i)\}|\mathbf{f})$.

Question. Simplify this further and obtain an expression for the data likelihood. Note that the expression should not involve $\mathbf{x}$.
Ans: Since we are under supervised setting, $\mathbf{x}$ is fixed and so can be moved over to conditions. Now given $\mathbf{f}$, y does not depend on $\mathbf{x}$.

$$\begin{aligned} P(D|\mathbf{f}) &= P(\{(x_i, y_i)\}|\mathbf{f}) \\ &= P(\{y_i\}|\{x_i\}, \mathbf{f}) \\ &= P(\{y_i\}|\mathbf{f}) \end{aligned}$$

By analytically evaluating, we get the data likelihood as:

$$P(\mathbf{f}|D) = \mathcal{N}(K(K + \sigma^2 I)^{-1}\mathbf{y}, \sigma^2 K(K + \sigma^2 I)^{-1})$$

Question. When will the data likelihood be maximum? When will the GP perfectly fit the dataset?
Ans: When the standard deviation is high, the data does not fit at all and the estimate falls back to the prior. On the other hand, when $\sigma \to 0$, we trust the data to the fullest, K is invertible and so the data fits perfectly.

4.2 Joint Distribution $P(\mathbf{f}, f^*)$

Define $\mathbf{k} \equiv [\mathcal{K}(\mathbf{x}^*, \mathbf{x}_1), \ldots, \mathcal{K}(\mathbf{x}^*, \mathbf{x}_N)]^T$. Then the joint distribution of $[\mathbf{f}\ f^*]^T$ is:

$$\begin{bmatrix} \mathbf{f} \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K & \mathbf{k} \\ \mathbf{k}^T & \mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix}\right)$$

4.3 Marginal and Conditional

For multi-variate Gaussian Processes, we can determine the marginal and conditional distribution using the joint distribution using the following properties:

$$P_{X,Y} \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix}\mu_X \\ \mu_Y\end{bmatrix}, \begin{bmatrix}\Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY}\end{bmatrix}\right)$$

We have $X \sim \mathcal{N}(\mu_X, \Sigma_{XX})$ and $Y \sim \mathcal{N}(\mu_Y, \Sigma_{YY})$ then:

$$X|Y \sim \mathcal{N}(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y), \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX})$$

$$Y|X \sim \mathcal{N}(\mu_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \mu_X), \Sigma_{YY} - \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XY})$$

Using these, we can find out the marginal $P(f^*)$ and conditional $P(f^*|\mathbf{f})$ distributions. Coming back to the integral:

$$P(f^*|\mathbf{x}^*, D) = \int_{\mathbf{f}} P(f^*|\mathbf{f}, \mathbf{x}^*, D) \cdot P(\mathbf{f}|\mathbf{x}^*, D) d\mathbf{f}$$

Question. Why is the result of the above integration Gaussian?
Ans: Since both would be Gaussian, and the data likelihood is also a Gaussian, the resulting distribution for $P(f^*|\mathbf{x}^*, D)$ after integration will also be a Gaussian. Solving the integration gives us:

$$P(f^*|\mathbf{x}^*, D) = \mathcal{N}\left(\mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{y}, \mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{k}\right)$$

Question. Do the labels $\mathbf{y}$ affect the variance of f^*?
Ans: We can see that variance $= \mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{k}$
As you can see, the variance of posterior distribution does not have any term related to labels, we can safely say that answer is No.

4.4 1-Nearest Neighbor Regressor

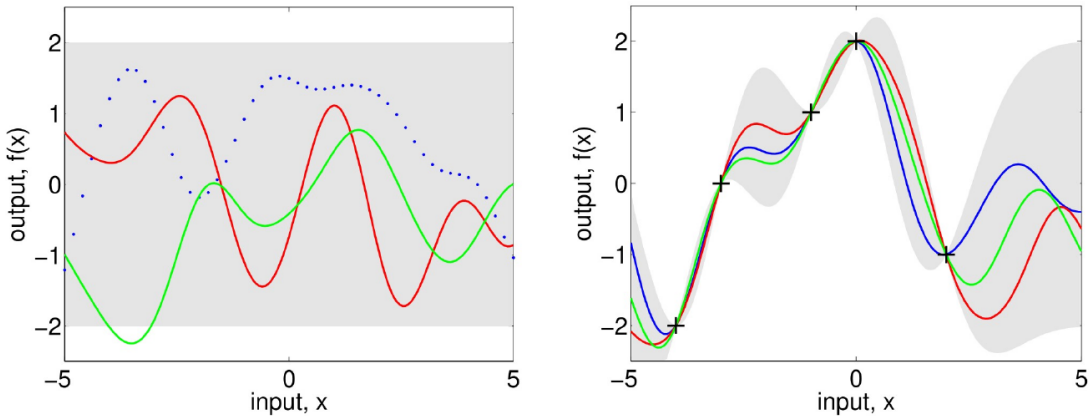
Question. How should $\mathbf{k}$ look like for the above equation to act as a 1-nearest neighbor regressor?
Ans: Suggestion from a student: we can use the Euclidean distance function $d(\mathbf{x}_i, \mathbf{x}_j)$. We then add a threshold β defined as $\beta_i = d(\mathbf{x}^*, \mathbf{x}_i) + \epsilon_i$. We can then define the kernel function as:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 & d(\mathbf{x}_i, \mathbf{x}_j) < \beta_j \\ 0 & otherwise \end{cases}$$

The Gaussian Process for a one-nearest neighbor regressor would be of the form:

$$GP\left(m(\mathbf{x}) = \mathbf{0}, \mathcal{K}(\mathbf{x}, \mathbf{x}') = e^{\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2}\right)}\right)$$

5 Sampling Procedure



(a), prior (b), posterior

Figure 2.2: Panel (a) shows three functions drawn at random from a GP prior; the dots indicate values of y actually generated; the two other functions have (less correctly) been drawn as lines by joining a large number of evaluated points. Panel (b) shows three random functions drawn from the posterior, i.e. the prior conditioned on the five noise free observations indicated. In both plots the shaded area represents the pointwise mean plus and minus two times the standard deviation for each input value (corresponding to the 95% confidence region), for the prior and posterior respectively.

To generate samples $\mathbf{x} \sim \mathcal{N}(\mathbf{m}, K)$ with arbitrary mean $\mathbf{m}$ and covariance generating multivariate matrix K using a scalar Gaussian generator (which is readily available in many Gaussian samples programming environments) we proceed as follows: first, compute the Cholesky decomposition L of the positive definite symmetric covariance matrix $K = LL^T$, where L is a lower triangular matrix. Then generate $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, I)$ by multiple separate calls to the scalar Gaussian generator. Compute $\mathbf{x} = \mathbf{m} + L\mathbf{u}$, which has the desired distribution with mean $\mathbf{m}$ and covariance $L\mathbb{E}[\mathbf{u}\mathbf{u}^T]L^T = LL^T = K$ (by the independence of the elements of $\mathbf{u}$).

6 Problems

6.1 GP with RBF Kernel

Consider a Gaussian Process $f(x) \sim GP(0, K)$ where K is the RBF kernel $K(x_1, x_2) = e^{\left(-\frac{(x_1-x_2)^2}{2}\right)}$ where $x \in \mathbb{R}$ and mean is 0. We have one data point $D = \{x_0 = 0, y_0 = -1\}$. Answer the following:

(a) What is $-2\mu(x) + 4\sigma^2(x)$, where $\mu(x)$ and $\sigma(x)$ are mean and standard deviation of the posterior distribution $P(f(x)|D)$ when:

- $x = \sqrt{\log(4)}$
- $x \to \infty$

(b) Let $y_1 = f(x = \sqrt{\log(4)})$ and $y_2 = f(x = -\sqrt{\log(4)})$ be 2 random variables. What is $\alpha + \beta$ where $\alpha, \beta = \arg\max P(y_1, y_2|D)$?

Solution:

(a) We know $P(f^*|\mathbf{x}^*, D) = \mathcal{N}\left(\mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{y}, \mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{k}\right)$.
Here as there is only one data point (x_0, y_0)

$$\mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{y} = \frac{\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0)\mathbf{y}_0}{(1 + \sigma^2)} = \mu(x)$$

$$\mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{k} = 1 - \frac{\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0)^2}{(1 + \sigma^2)} = \sigma^2(x)$$

Now using $\mathbf{y}_0 = -1$

$$-2\mu(x) + 4\sigma^2(x) = 4 + \frac{\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0)(2 - 4(\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0))}{1 + \sigma^2}$$

For $\mathbf{x}^* = \sqrt{\log(4)}$, we get $\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0) = \frac{1}{2}$ and for $\mathbf{x}^* \to \infty$, we get $\mathcal{K}(\mathbf{x}^*, \mathbf{x}_0) = 0$.
Putting the values back, we get $-2\mu(x) + 4\sigma^2(x) = \mathbf{4}$ for both cases.

(b) Given the symmetry for positive and negative values of x, both y_1 and y_2 follow $\mathcal{N}(\mu = -1, \sigma^2 = 0.5)$. Hence the joint distribution will also be a Gaussian, and the position of maximum value will be the mean, which will be given as (using the formula mentioned in 1.4):

$$\mu = \frac{-1 \times 0.5 + -1 \times 0.5}{0.5 + 0.5} = -1$$

Since the distribution has circular symmetry, $\alpha = \beta = -1$, giving us the answer $\mathbf{-2}$.

6.2 Fitting GP on Data

Suppose we want to sample a point $\mathbf{x}$ such that a GP that is fit on D exhibits the least variance there. Formulate the objective that gives us this.

Solution:

We sample the point $\mathbf{x}$ which corresponds to minimum of the variance of the posterior distribution:

$$\arg\min_{x^*} \left(\mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(K + \sigma^2 I)^{-1}\mathbf{k}\right)$$

Qualitatively, it is very likely that this point will lie close to a dense cluster of the training data and hence exhibit lowest variance.

6.3 1-D Regressor GP

We are estimating a 1-D regression function $f(x)$ as a Gaussian Process $GP(m(x), K(x, x'))$ where the kernel function $K(x, x') = \sigma_f^2 e^{\left(\frac{(x-x')^2}{2\tau}\right)}$ and $m(x) = 0$. Each $y_i = f(x_i) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Assume training data $D = \{(\mathbf{x}_i, y_i\}_{i=1}^N$. Express briefly in qualitative terms the shape of the mean of $f(x|D)$ posterior to seeing the training data D in terms of the following properties:

(a) Noise σ^2 is very large

(b) Length scale τ changes from 10^{-4} to 10^4

Solution:

(a) The posterior falls back to the prior since data is unreliable (very noisy).
Mathematically, we know $P(\mathbf{f}|D) = \mathcal{N}(K(K + \sigma^2 I)^{-1}\mathbf{y}, \sigma^2 K(K + \sigma^2 I)^{-1})$.
Hence the mean = $(K(K + \sigma^2 I)^{-1}\mathbf{y}$ and variance = $\sigma^2 K(K + \sigma^2 I)^{-1})$
If noise is large then $\sigma^2 I$ term dominates K hence mean $= (K(K + \sigma^2 I)^{-1}\mathbf{y} \approx \frac{K\mathbf{y}}{\sigma^2} \approx \mathbf{0}$.
Using the same argument variance = $\sigma^2 K(K + \sigma^2 I)^{-1}) \approx \frac{\sigma^2 KI}{\sigma^2} \approx K$.
So $P(\mathbf{f}|D) \approx \mathcal{N}(\mathbf{0}, \mathbf{K})$ which is equal to prior.

(b) As τ increases, because of kernel function having reduced flexibility (smaller range of values), the fluctuations will reduce, distribution becomes dense, and so the GP becomes very smooth.

AIML - CS 337

Lecture 19: Mixture Models

20/10/2022

Lecturer: Abir De *Scribe: Course Team*

1 Introduction

We observe a data set $D = \{X_i\}_{i=1}^N$, where each $X_i = x_i$ is being sampled from one of the K mixture components.

Each of the mixture component is a multivariate Gaussian density with its own parameters $\theta_k = \{\mu_k, \sum_k\}$

$$p_k(x_i|\theta_k) = \frac{1}{(2\pi)^{d/2}|\sum_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^t \sum_k^{-1}(x-\mu_k)}$$

We now have to estimate the parameters of the K mixture components, θ_k and the mixture weights, which represent the probability that a randomly selected $\bar{x}$ was generated by k^{th} component, $\pi_k = P(c(\bar{x}) = k)$, where $\sum_{k=1}^K \pi_k = 1$.

2 Computing posterior distribution $P(c(X_i) = k|X_i)$

Using initial estimates for ω, we obtain the posterior in the following way -

$$P(c(X_i) = k \mid X_i, \omega) = \frac{P(\boldsymbol{X_i} \mid c(X_i) = k, \theta_k).P(c = k)}{\sum_m P(\boldsymbol{X_i} \mid c(X_i) = m, \theta_m)P(c = m)} = \frac{N(X_i; \theta_k)\pi_k}{\sum_m N(X_i; \theta_m)\pi_m}$$

This follows a direct application of Bayes rule. These membership weights reflect the uncertainty, given $X_i = x_i$ and ω, about which of the K components generated vector $X_i = x_i$.

3 Maximum Likelihood Estimation

The complete set of parameters for a mixture model with K components is -

$$\omega = \{\pi_1, \pi_2, ..\pi_K, \theta_1, .., \theta_K\}$$

We now maximize the likelihood of data, $P(D) = P(X_1 = x_1, X_2 = x_2, ..., X_N = x_N)$ w.r.t ω.

$$P(D) = \prod_{i=1}^N P(\boldsymbol{X_i} = \boldsymbol{x_i})$$

$$\implies \log(P(D)) = \sum_{i=1}^N \log(P(\boldsymbol{X_i} = \boldsymbol{x_i}))$$

We know that marginal probability of X_i is,

$$P(\boldsymbol{X}_i = \boldsymbol{x}_i) = \sum_{k=1}^{K} P(\boldsymbol{X_i} = \boldsymbol{x_i} \mid c(X_i) = k) P(c = k)$$

$$\implies P(\boldsymbol{X}_i = \boldsymbol{x}_i) = \sum_{k=1}^{K} P(\boldsymbol{X_i} = \boldsymbol{x_i} \mid c(X_i) = k) \pi_k$$

Using the above,

$$\log(P(D)) = \sum_{i=1}^{N} \log(\sum_{k=1}^{K} P(\boldsymbol{X_i} \mid c(X_i) = k) \pi_k)$$

Differentiating the above w.r.t π_k, μ_k and $\sum_k$, we obtain the new parameters (and using the equation presented in Section 2)-

$$\text{Let } N_k = \sum_{i=1}^{N} P(c(X_i) = k | X_i, \omega)$$

$$\pi_k^{new} = \frac{N_k}{N}$$

$$\mu_k^{new} = (\frac{1}{N_k}) \sum_{i=1}^{N} X_i . P(c(X_i) = k | X_i, \omega)$$

$$\sum_k^{new} = (\frac{1}{N_k}) \sum_{i=1}^{N} P(c(X_i) = k | X_i, \omega) . (X_i - \mu_k^{new}) . (X_i - \mu_k^{new})^t$$

4 Iterative Procedure for Parameter Estimation

We now work on choosing a suitable initial prior for π_k. Entropy of a distribution is defined as $-\sum_{i=1}^{N} P(\boldsymbol{X_i}) * \log(P(\boldsymbol{X_i}))$ where $\boldsymbol{X_i}$ are random variables of the distribution. In a K-means cluster distribution, we have $\pi_1, \pi_2, .., \pi_K$. In order to maximize the randomness, we assign each one of the random variables probability 1/K.

Now, using the above initial prior for π_k, and some initial parameter estimates θ_k, we derive the posterior $P(c(X_i) = k | X_i)$ (membership weights) as presented in Section 2.

Using these new membership weights, we calculate the new π_k, μ_k and $\sum_k$ using the equations given at the end of Section 3 (derived by differentiating the log likelihood).

Using these new parameter estimates, we calculate the new membership weights and repeat the steps again until the value of likelihood of data converges.

Log likelihood of data - $\log \prod_{i=1}^{N} P(\boldsymbol{X_i}) = \sum_{i=1}^{N} \log(\sum_{k=1}^{K} P(\boldsymbol{X_i} \mid c(\boldsymbol{X_i}) = k) P(c = k))$

Let $P_\omega = P(\boldsymbol{X_i} \mid c(\boldsymbol{X_i}) = k), P_c = P(c = k \mid \boldsymbol{X_i})$

$\omega = \omega^{t-1}$

At time t, $\max_{\omega} \sum_{i=1}^{N} \log(\sum_{k=1}^{K} P_\omega P_c(\omega^{t-1}))$ will give us the new parameter estimates ω

5 Representation in terms of Expectation

We can also represent the likelihood of data $\{\prod_{i=1}^{N} P(\boldsymbol{X_i})\}$ as below.

Now, $P(\boldsymbol{X}) = \sum_{Z} P(X|Z)P(Z)$

implies $P(\boldsymbol{X}) = \mathbf{E}_{\boldsymbol{Z}}[P(\boldsymbol{X}|\boldsymbol{Z})]$

Hence, $P(\boldsymbol{X_i}) = \mathbf{E}_c[P(\boldsymbol{X_i} \mid c)]$

$\prod_{i=1}^{N} P(\boldsymbol{X_i}) = \prod_{i=1}^{N} \mathbf{E}_c[P(\boldsymbol{X_i} \mid c)]$

Now, $\prod_{i=1}^{N} \sum_{k=1}^{K} P(\boldsymbol{X_i}|\boldsymbol{c} = \boldsymbol{k}) P(\boldsymbol{c} = \boldsymbol{k})$

is equal to, $\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \sum_{k_3=1}^{K} .. \sum_{k_N=1}^{K} (\prod_{i=1}^{N} P(\boldsymbol{X_i}|\boldsymbol{c} = \boldsymbol{k_i}) P(\boldsymbol{c} = \boldsymbol{k_i}))$

$\prod_{i=1}^{N} P(\boldsymbol{X_i}) = \mathbf{E}_{(k_1,k_2,k_3...,k_N)}[\prod_{i=1}^{N} P(\boldsymbol{X_i} \mid c = k_i)]$

6 Mixture Model to K-Means iterative algorithm

Entropy of a distribution is defined as $S(X) = \sum_{i=1}^{N} P(X_i) \log(P(X_i))$ where X_i are random variables of the distribution. Entropy is maximised when all of these probabilities are equal (easily proved with differentiation).

So we set the prior $w_k = 1/K$ for all k initially to maximise entropy in K-Means. We set random initial parameter estimates θ. In addition, for later iterations we set $P(c = k) = \boldsymbol{I}(c = k)$ where $\boldsymbol{I}$ is the delta function.

Using maximum likelihood estimation of data, we calculate the new parameters and weights and stop when the likelihood converges.

AIML - CS 337

Lecture 20: Gradient Descent

27 October 2022

Lecturer: Abir De *Scribe: Group 41 & Group 42*

Till now, we haven't discussed minimization of **Supervised Learning Algorithms yet**. In this lecture, we cover the optimisation technique **Gradient Descent**; the working of the algorithm, the cause of stochastic behaviour of gradient descent and how to utilise this stochastic nature to our advantage. We also see applications of gradient descent in an upcoming technology known as *MLaaS* (*M*achine *L*earning *a*s *a S*ervice). This lecture is mainly theoretical in nature where many concepts are just introduced at a very high level without getting into the Maths and detailed theory involved.

1 Introduction

Optimisation is a central requirement in machine learning. For instance, consider we have a dataset $\mathcal{D}$ with the i^{th} data point $(x_i, y_i) \in \mathcal{D}$. Then, for a given model $f(x) = \theta^T x_i$ loss function, we have

$$L(\theta) = \sum_{i \in \mathcal{D}} L(\theta^T x_i, y_i) \tag{1}$$

And, we seek to solve the following optimisation problem

$$\min_{\theta} L(\theta) \tag{2}$$

Let us represent the initial value of the model (the starting point) as θ^0. We seek the following assurance from our optimization algorithm

$$\theta^0 \xrightarrow{???} \theta^* = \arg\min_{\theta} L(\theta)$$

Thus, whatever the initial value of the weights/learnable parameter (θ^0), we obtain θ^*

We now answer some crucial questions about θ^0 such as:

- How do we choose θ^0 and what effect does this choice have on our results?
- Do we converge to 'the' minima on starting from any θ^0?

2 The Gradient Descent Setup

Gradient Descent begins with a starting value, θ^0. Then, at each intermediate value of the model, it computes the gradient of the loss function with respect to the model parameters and updates the

model a fixed constant (called learning rate) times the gradient in the 'direction' opposite to that of the gradient. Thus, we express

$$\theta^t = \theta^{t-1} - \eta_t \nabla_\theta L(\theta) \tag{3}$$

where η_n is the learning rate at time step 't' and θ^t represents the model weights/parameters after 't' iterations and we terminate when a suitable convergence condition is met. The value of θ thus obtained is claimed to be θ^*

The aim of gradient descent (GD) algorithms is to minimize function l with respect to parameters $\mathbf{w}$ by updating them towards negative gradient. Whether the algorithm converges to a global or local minimum, or at all, depends on the initialization of parameters, step size (= learning rate), and properties of the loss function such as convexity.
The pseudocode for GD is as follows :

Algorithm 1 Gradient Descent

$\mathbf{w}_0 \leftarrow \mathbf{r}$ ▷ $\mathbf{r}$ is a randomly initialised vector
$i \leftarrow 0$
while ($\mathbf{w}$ `has not converged`) **do**
    $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \eta_i \cdot \text{Grad}(\ell, \mathbf{w}_i, \mathbf{x}, \mathbf{y})$
    $i \leftarrow i + 1$
end while

Here, Grad$(\cdot)$ is substituted according to the algorithms being GD, Mini-batch GD or SGD. Note that the hyperparameters $\mathbf{w}_0$ and η can be tuned and affect the speed as well as convergence result of the algorithm (in finite steps).

2.1 Vanilla/Batch Gradient descent

When the number of training samples is relatively small, an effective method to update the parameters is to compute the gradient using the whole training set at each iteration:

$$\begin{aligned}\mathbf{w}_{i+1} &= \mathbf{w}_i - \eta_i \nabla_w l(\mathbf{w}_i) \\ &= \mathbf{w}_i - \eta_i \sum_{(x,y)\in\mathcal{D}} \nabla_w \ell(\mathbf{w}_i, \mathbf{x}, \mathbf{y})\end{aligned}$$

where η_i is the learning rate. Note that the learning rate does not necessarily have to be constant during the search. Even though batch GD converges with comparably small number of iterations, calculating the gradient with a large training set can be computationally very inefficient.

Note: This is also referred to as Batch Gradient Descent in multiple texts.

2.2 Mini-batch gradient descent

In mini-batch GD, the gradient is computed by using a smaller subsets of the training set at each iteration. If $B_1, B_2, ... \subset \mathcal{D}$, is a partition of the training set, then the parameters are updated by

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \sum_{(x,y)\in \boldsymbol{B}_i} \nabla_w \ell(\mathbf{w}_i, \mathbf{x}, \mathbf{y})$$

It is likely that the algorithm does not converge before using all the batches, and in this case the batches are shuffled and re-iterated.
When the size of dataset is large it is a difficult job to store the gradients calculated using all of the training instances. This is a limitation caused due to the **GPU memory**.

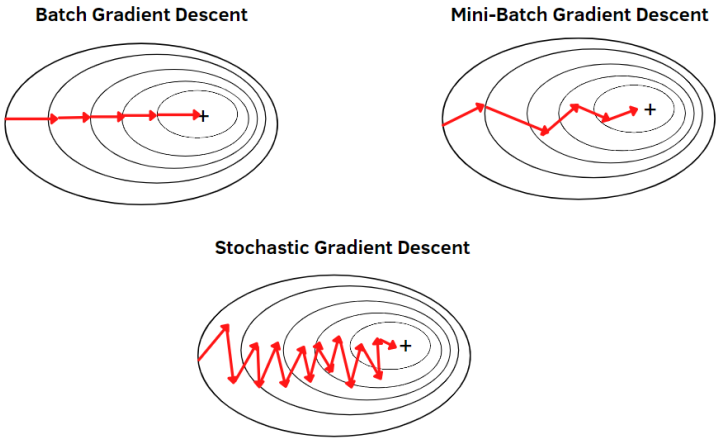
To resolve this issue, mini-batch GD is adopted with a feasible batch size (practically, it is around 10-20 for Computer Vision applications with 360p image resolutions and 2-4 in NLP applications involving documents). It improves the speed of convergence as well since quicker (but noisier) updates are made to the parameters.

2.3 Stochastic gradient descent

In stochastic GD, the training set is shuffled and the gradient is computed using a single (random) instance at each step. In every iteration, the training set is randomly shuffled and such updates are performed for all instances :

$$\mathbf{w} = \mathbf{w} - \eta_i \nabla_w \ell(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

where $(\mathbf{x}, \mathbf{y})$ is the random instance for one of the updates. In this algorithm, it is not needed that the updates are along the direction of gradient vector but the expectation of updates (for every pass over the training set) is parallel to the gradient.



The figure above compares the three algorithms in their descent trends. To recall, Batch GD considers all of the training set, Mini-Batch GD considers small batches of training instances, Stochastic GD considers only single instances for gradient updates. Thus, the direction along which the update vector (Δw term) points and the smoothness of the descent can be explained.

2.4 Hyper-parameters

There are 2 hyperparameters which gradient descent needs, namely the starting point θ^0 and the learning rate η_t (possibly variable as a function of t)

2.4.1 Choice of Learning Rate η

This famous graph below[1] that shows how a learning rate that is too big or too small affects the loss during training.

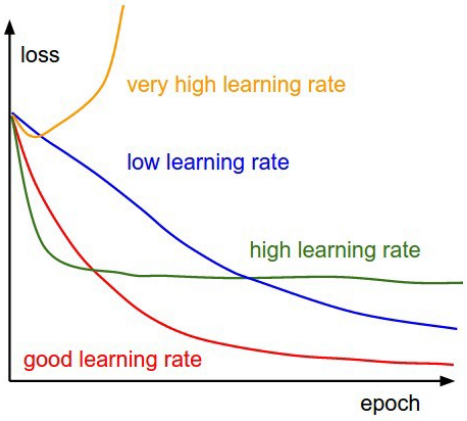


Figure 1: Choice of Learning Rate η

With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line).

2.4.2 Choice of θ^0

We observe, and can show, that for a given learning rate, the various values of θ^0 can give different values of θ^*. An important question lies ahead. Suppose we choose various values of the starting model, say

$$\Theta^0 = \{\theta_1^0, \theta_2^0 \cdots \theta_n^0\}$$

and run gradient descent to obtain the corresponding optimal values (for which loss is minimised) as

$$\Theta^* = \{\theta_1^*, \theta_2^* \cdots \theta_n^*\}$$

Should we choose the best value in Θ^* or the average value and why?

Claim 2.1. *In case of various optimal values θ^* corresponding to various starting values, θ^0, we choose the best value of θ^* (for which loss is minimised)*

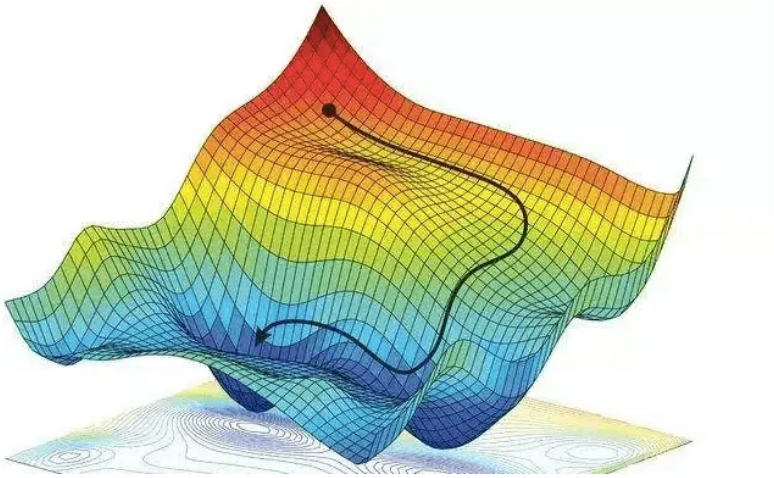


Figure 2: Plot of Loss(z-axis) vs θ_1(x-axis), θ_2(y-axis)

A justification to this is that, the choice of initial value can result in the gradient descent algorithm converging to a local minima and not being able to identify a "better" minima. Thus, the choice of various starting points allows the algorithm to explore various minima points and then choosing the best allows us to choose the more optimal value.
Note: The comparisons are done on the basis of performance on the **validation set**.

2.5 Initialisation During Coding

- Let us assume that we run optimisation multiple times, starting with random $\mathbf{w}_0^1, \mathbf{w}_0^2, \ldots$ each time resulting in validation errors as $\epsilon_1, \epsilon_2, \ldots$
  Question: How do we +decide which $\mathbf{w}_0^i$ to rely upon?
  Claim: Choose that $\mathbf{w}_0^i$ which gives least validation error ϵ_i.
  Justification: As explained above, lower validation error means better generalisation and weights giving least error signify the (yet) best found minima.

- In mini-batch gradient descent fix the initial weight parameter $\mathbf{w}_0$ and train over various batch sequences $\{\mathcal{B}_0^i, \mathcal{B}_1^i, \mathcal{B}_2^i, \ldots\}_{i=0}^N$
  Assume that $\mathbf{w}_0$ generates error ϵ_i for the ith sequence
  Question: While comparing various initialisations of $\mathbf{w}_0$, what metric should we use to judge them? Best error i.e. $\min(\epsilon_i)$ or average error i.e. $\bar{\epsilon}$?
  Claim: Use the average error i.e. $\bar{\epsilon}$ and not $\min(\epsilon_i)$ as a judging metric
  Justification: Relying on the sequence of batches is equivalent to using validation for training the model and this results in overfitting. Average error is equivalent to $\mathbb{E}[\epsilon]$ when a large #batches are tried and is a better way to judge the parameter.

- **Question**: Where exactly do ML libraries initialise the weights?

They would be initialised for an instance of the model when the model object constructor is called. The parameters can also be sampled from various distributions.[3]

```
class MyModel(self, ...):

    def __init__:
        self.linear = nn.Linear(in_features, out_features)
        #This is the line where initialization occurs with RANDOM weights

    def initialize(self):
        self.linear.weight.data = torch.randn(size)
        self.linear.bias.data = torch.randn(size)
        #This is the function where we initialize weights and biases
    according to the distribution of our choice

if __name__ == "__main__":
    model_instance = MyModel(...)
    #Writing this is fine, it still allows random initialization of
    weights
    model_instance.initialize()
    #This is optional, we write it only if we want to initialize weights
    with a distribution of our choice
```

We have mentioned that `Torch` automatically initializes weights of the Linear Layer when we create an instance of `nn.Linear`. One might ask what distribution did `Torch` use when it initialized the weights.
`Torch` uses $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$ to sample the weights.
Practically, we often observe a certain randomness in our results even when we use the same hyperparameters and model. What are the sources of this randomness?

1. When we use **Batch Gradient Descent**, we use the gradients of the *entire dataset* to move one step. That is, we will compute the gradient of loss with contribution from each point of the dataset and move in the direction opposite to this computed gradient. In such a scenario, the only source of randomness in a model (assuming we are not using other stochastic techniques such as **dropout**) is the *initialization of* θ^0
2. When we use **Mini-Batch Gradient Descent**, the choice of batches is also a source of randomness which has an effect on the final outcome. In each iteration, we choose different partitions of the data set to make batches and these partitions are often random, which results in non-determinism.

3 Mini-Batch Gradient Descent

Above, we would compute the loss function over the entire dataset, however, this suffers from one major problem; it is **slow to converge**[2]. Hence, it is prudent to use batching; training on chunks of the dataset at a time.

3.1 Dependence on Batch Order

We observe that the various orders of batches can give different values of θ^*. An important question lies ahead. Suppose we choose various batch orders of the starting model, say

$$\Omega = \{O_1, O_2 \cdots O_n\}$$

and run gradient descent to obtain the corresponding optimal values (for which loss is minimised) as

$$\Theta^* = \{\theta_1^*, \theta_2^* \cdots \theta_n^*\}$$

Should we choose the best value in Θ^*? Or the average value and why?

Claim 3.1. *In case of various optimal values θ^* corresponding to various batch orders, θ^0, we choose the average value of θ^* (for which loss is minimised)*

A justification to this is that, the choice of 'best' batch order depends on the validation dataset and can result in the gradient descent algorithm overfitting on the validation dataset. Even if we employ early stopping to prevent overfitting on the training set, we will overfit on the validation set, since we're trying to 'train' the batch choice on the validation set. For it to generalise well, we choose the average over all the optimal values obtained through the various batch orders.

We should have a **high generalization** on the validation set. If on changing batches, we get a huge change in validation error, that means the model is **not stable with respect to batches** and in that case, one should change the model or tune some other hyperparameters.

3.1.1 Questions to Think about on this topic

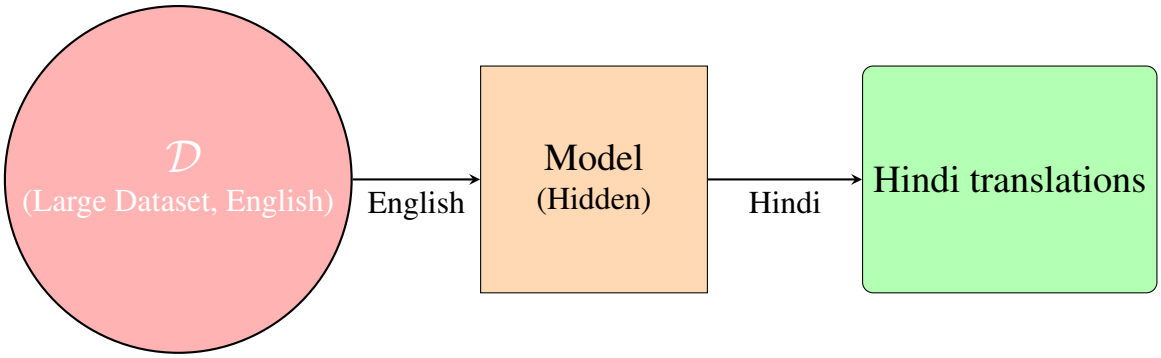
1. Can you prove an equivalence between L-2 Regularization and mini-batch gradient descent? Mini-batch gradient descent can also be used to prevent overfitting like L-2 regularization. We can say that under certain conditions, mini-batch gradient descent gives us similar results on running optimization algorithms. Can you find this strong connection?

2. Suppose you have a 24 Gigabyte GPU and you wish to run a training algorithm with 500 epochs at max. Assume that other parameters are given, then how would you choose your batch choice?

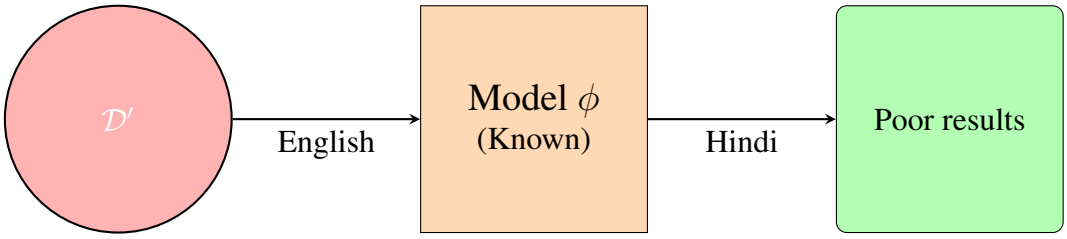
4 Machine Learning as a Service

Have you ever used a service which has an online chatbot? Say, on a banking website which employ models which use NLP techniques in creating such chatbots. MLaaS is a cloud computing service in which clients can use powerful models for their own use without having the hassle of research and development.

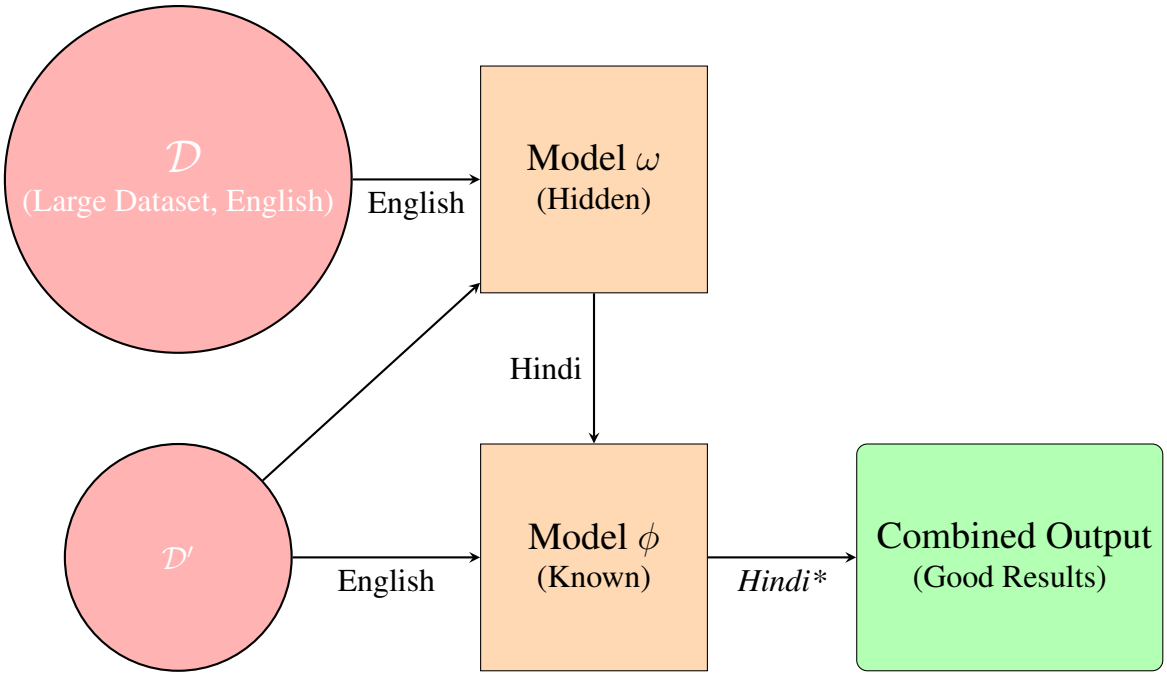
We will work with the simplified example of an online service which performs a certain task, say Google Translate, a service which takes an English sentence and converts it to Hindi.
The task is performed by a model M trained on a large dataset $\mathcal{D}$.



Now, we want to train another model *M'* on a smaller dataset $\mathcal{D}'$ supplementing training using model *M*, which acts as a black box since we do not have access to the internal structure and parameters of the model itself. The structure of *M* is the company's trade secret. We also do not have a dataset as large as Google does!



We wish to utilise the output of the model M during the training of M' to improve the results we would have got only using M'.
Our goal is to get a 'Gold Standard' output, which we call *Hindi. Note that this is different from the translations which just Google's model gives, which is why we were employing our model in the first place. Ideal output matching *Hindi** cannot be obtained with just our data and model alone as well though.**



$$\theta = [\phi, \omega]$$

but Learnable parameters are just ϕ

The issue is that, we cannot backpropagate through M since the gradient of the loss with respect to the parameters of M is not known, i.e. even though ϕ are known, ϕ and thus θ aren't. Are main problem is the lack of knowledge about the functional form of ω due to which we cannot compute gradients with respect to them. So what is the solution?

4.0.1 Approach 1: Perturbation

A proposed solution was that we can perturb the inputs to M a little, thus effectively estimating the loss function around the inputs allowing us to backpropagate without knowing the gradient. This is like numerical estimation of the derivative. However, this is **very costly** since we need a huge number of queries to get smooth derivatives, numerical derivative computation is computationally expensive and thus **not a feasible solution**.

4.0.2 Approach 2: Gaussian Processes

The idea is to model M using a Gaussian process (any Bayesian model can suffice), i.e. replace the entire black box by a Gaussian Process. We use this to obtain correct results and reduce uncertainty. The procedure is roughly as follows

- Select an example sentence s through $\mathcal{D}$, pass it through M and get loss

- Make Gaussian process and train *M'* to fine tune for sentences *s'* close to original example sentence s

Using Gaussian Processes saves us the entire hassle of computing derivatives, since no derivatives are involved in Gaussian Process, thus solving our problem with backpropagation.

Note that for a large amount of data, the **interpolative nature** of Gaussian Process almost approximates the black box we are trying to model.

4.1 Prompt Engineering

This is a very interesting and upcoming area of NLP. Usually a model is designed for one task. But, we can make models which can generalise to many tasks. Say, a model is trained on some set of tasks (using 1 million training samples) but tested on other task, not in the set of tasks in the training set. It is possible to provide some examples (10^1) and retrain to make it 'adjust' to the new task.

As an example, suppose we are designing a complex model which processes a set of questions which are of a similar type:

1. Who is the Chancellor of Germany? *Ans: Olaf Scholz*
2. When did Armstrong land on the moon? *Ans: 1969*
3. When was Obama born? *Ans: 1961*

We have a 'training' data set of 1 million questions and label answers. While testing, now we give our model some 10 test examples which are nothing like what we've seen before.
As an example, we prompt the model, "What is the sentiment of the people towards an X political party?" Although our model will not be able to answer it accurately from the training it has received, when we retrain it with **just these 10 new 'test examples' (since they are not from an actual test set in the conventional set)**, the model is seen to perform surprisingly well on unseen examples.

Have you ever heard about the godly transformer model **GPT-3**?
It is a NL model that can do absolutely any task given to it in writing. There's a huge variety of tasks that it can perform. Keeping digression aside, here the input that we provided to the model "*Find the derivative of log(x)*" is called the **prompt** and then processes it.
In prompt engineering, the description of the task is embedded in the input, e.g., as a question instead of it being implicitly given. Prompt engineering typically works by converting one or more tasks to a prompt-based dataset and training a language model with what has been called "prompt-based learning" or just "prompt learning". Prompt engineering may work from a large "frozen" pretrained language model and where only the representation of the prompt is learned, with what has been called "prefix-tuning" or "prompt tuning".
Another way is, instead of a single sentence as a prompt, give few examples of **question-answer**

[1]Not exact numbers. Just an order of magnitude

pairs followed by the main question as your prompt. Then, the model interpretes the examples and owing to its huge recalling capability, is able to correctly answer your question. This is called **Few Shot Learning**.

Thus, there exists a certain robustness in the model by which when we train on a very small 'test set', we will be able to **update our weights** and still get very good answers for unseen questions. This is an interesting topic of current research known as **Prompt Engineering**.

5 Questions to ponder

- Why the configuration/permutation of the sequence of mini-batches is not a hyperparameter but the number of bathces (in the sequence) is?
- Show that Gradient Descent with L2 regularization is equivalent to Mini-batch Gradient Descent without L2 regularization with some appropriate batch size.
- If you have a GPU with 24GB RAM and you will train a model for 500 iterations at max, then what should be the batch size you would use?

6 Conclusion

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. It is a helpful tool in the machine learning toolkit which also works moderately well in practice. There exist many gradient descent algorithms[4] which further optimise gradient descent. We use gradient descent to optimise models in machine learning services and prompt engineering as well

References

[1] *CS231n: Deep Learning for Computer Vision*. Stanford University.

[2] B. Fehrman, B. Gess, and A. Jentzen. Convergence rates for the stochastic gradient descent method for non-convex objective functions. 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[4] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs, 2016.

AIML - CS 337

Lecture 21: Generative Models

7th November 2022

Lecturer: Abir De *Scribe: Group 43*

1 Introduction

So far, we have looked at various discriminative algorithms. It is important to understand the difference between Discriminative models and Generative models. Discriminative models draw boundaries in the data space, while generative models try to model how data is placed throughout the space. A generative model focuses on explaining how the data was generated, while a discriminative model focuses on predicting the labels of the data.

2 What to expect from a Generative Model

Let's say we are dealing with images. Given a set of images, $I = \{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_N}\}$, a Generative model tries to learn the underlying distribution from which the data might have been obtained. However, this is not an easy task. We also expect the Generative model to generate new images similar to a given image $\mathbf{x}'$, i.e the model should be able to sample an output image using the conditional distribution given an input image $\mathbf{x}'$. A few obvious challenges are : How do we approximate the underlying distribution? and how do we calculate the similarity between an input $\mathbf{x}$ and a Generated output $\mathbf{x}'$?

3 Universal approximator of a distribution

Goal To learn a function $P : \mathbb{R}^d \rightarrow [0, 1]$ satisfying $\int_{-\infty}^{\infty} P(\mathbf{x})d\mathbf{x} = 1$ and capable of capturing any distribution. For example, we would like $P_\theta(.)$ to capture different distributions by changing the parameter θ.

A possible approach For each $\mathbf{x}$, we can model the distribution as a Normal distribution with mean and covariance matrices taken as a function of θ and $\mathbf{x}$. We can model the mean and covariance as the output of some Neural network. We can write

$$P_\theta(\mathbf{x}) \sim \mathcal{N}(\mu_\theta(\mathbf{x}), \Sigma_\theta(\mathbf{x}))$$

The Neural Network should to be able to learn $\mu_\theta(\mathbf{x})$ and $\Sigma_\theta(\mathbf{x})$ to enable the Normal distribution to capture the desired distribution.

How do we train this model? We can use the KL divergence to measure the similarity between two probability distributions. Here we will be comparing the empirically generated PDF and the PDF obtained by the Generative model.

KL divergence For distributions P and Q of a continuous Random variable, the KL divergence is defined to be

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

It is a measure of how one probability distribution P is different from a second, reference probability distribution Q.

4 Generative procedure : Latent Variable Model

Let $\mathbf{x}$ denote the variable that represents our data and assume that $\mathbf{x}$ is generated from a latent variable z that is not directly observed. Here, z can be thought of an *encoded* representation of $\mathbf{x}$ and is analogous to a seed for generation. We assume that we can easily sample the latent variables according to some probability density function $P(z)$ defined over some high dimensional space $\mathcal{Z}$. Then, say we have a family of deterministic functions $f(z;\theta)$, parameterized by a vector θ in some space Θ, where $f : \mathcal{Z} \times \Theta$. f is deterministic, but if z is random and θ is fixed, then $f(z;\theta)$ is a random variable in the space $\mathcal{X}$. We wish to optimize θ such that we can sample z from $P(z)$ and, with high probability, $f(z;\theta)$ will be *like* the $\mathbf{x}$'s in our dataset.
To accommodate the notion of likelihood, we replace these deterministic functions with a probability distribution. The total probability of $\mathbf{x}$ being observed can then be expressed as

$$P(\mathbf{x}) = \int P(\mathbf{x}|z;\theta)P(z)dz$$

For example, we can assume that both z and $\mathbf{x}|z$ are distributed normally. Then the genrative process can be summarized as

$$z \sim \mathcal{N}(\mathbf{0}, I)$$
$$\mathbf{x}|z \sim \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$$

Here, $\mu_\theta(z)$ and $\Sigma_\theta(z)$ can be thought of as the output of some Neural Network which takes in z as an input.
The model should learn to increase $P(\mathbf{x})$ given some training data, i.e., the generative model should learn to make the training data more likely.

Training Let's say we have some training data $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}$. As mentioned above, we focus on maximizing the **Likelihood** of the training data. The likelihood of training data for a generative

model can be written as

$$P(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}) = \prod_{i=1}^{N} P(\mathbf{x_i}) = \prod_{i=1}^{N} \int P_\theta(\mathbf{x_i}|z)P(z)dz$$

So, we can write the log likelihood as

$$LL(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}) = \sum_{i=1}^{N} \log \int P_\theta(\mathbf{x_i}|z)P(z)dz$$

Using *Jensen's* Inequality,

$$\sum_{i=1}^{N} \log \mathbb{E}_z[P_\theta(\mathbf{x_i}|z)] \geq \sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$$

So, we can finally write

$$\sum_{i=1}^{N} \log \int P_\theta(\mathbf{x_i}|z)P(z)dz \geq \sum_{i=1}^{N} \int [\log P_\theta(\mathbf{x_i}|z)]P(z)dz = \sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$$

Based on the above inequality, we can instead aim to maximise $\sum_{i=1}^{N} \mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$.
We can compute $\mathbb{E}_z[\log P_\theta(\mathbf{x_i}|z)]$ approximately. For a given i, first obtain a large number of samples $s_{i1}, s_{i2}, ..., s_{1S}$ for z and compute $\frac{1}{S}\sum_{j=1}^{S} \log P_\theta(\mathbf{x_i}|s_{ij})$. Thus we finally try to maximize

$$\sum_{i=1}^{N} \sum_{j=1}^{S} \frac{\log P_\theta(\mathbf{x_i}|s_{ij})}{S}$$

5 Conditional generation

So far we have seen the generation of a sample based on the training data without any conditions. Suppose, we now modify the problem : Given a sample $\mathbf{x}^*$, generate a sample $\mathbf{x}$ which is most similar to $\mathbf{x}^*$. In other words, we would like to maximize

$$P(\mathbf{x}|\mathbf{x}^*) = \int P(\mathbf{x}|z)P(z|\mathbf{x}^*)dz$$

The first quantity $P(\mathbf{x}|z)$ within the integral is known. The second part can be calculated as

$$P(z|\mathbf{x}^*) = \frac{P(\mathbf{x}^*|z)P(z)}{P(\mathbf{x}^*)} = \frac{P(\mathbf{x}^*|z)P(z)}{\int_{\mathcal{Z}} P(\mathbf{x}^*|s)P(s)ds}$$

Thus, the required probability becomes

$$P(\mathbf{x}|\mathbf{x}^*) = \int P(\mathbf{x}|z)\frac{P(\mathbf{x}^*|z)P(z)}{\int_{\mathcal{Z}} P(\mathbf{x}^*|s)P(s)ds}dz$$

All quantities in the above equation can be computed using known information. However, it is expensive to compute it in general.
Alternatively, we can model $P(z|\mathbf{x}^*)$ separately. It means that we need a new function $Q_{\mathbf{x}}(z)$ which can take a value of $\mathbf{x}$ and give us a distribution over z values that are most likely to produce $\mathbf{x}$. The space of z values that are likely under Q should be much smaller than the space of all z values which were all previously equally likely under the prior $P(z)$.

Training $Q_{\mathbf{x}}(z)$ A separate neural network can be used to train $Q_{\mathbf{x}}(z)$. Our objective can be to minimise the KL divergence between $Q_{\mathbf{x}}(z)$ and $P(z|\mathbf{x})$

$$\begin{aligned}
\mathrm{KL}(Q_{\mathbf{x}}(z), P(z|\mathbf{x})) &= \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log Q_{\mathbf{x}}(z) - \mathbb{E}_{z\sim Q_{\mathbf{x}}}\left(\log \frac{P(\mathbf{x}|z)P(z)}{P(\mathbf{x})}\right) \\
&= \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log Q_{\mathbf{x}}(z) - \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log P(\mathbf{x}|z) - \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log P(z) + \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log P(\mathbf{x}) \\
&= \mathrm{KL}(Q_{\mathbf{x}}(z), P(z)) - \mathbb{E}_{z\sim Q_{\mathbf{x}}} \log P(\mathbf{x}|z) + \mathbb{E}_{z\sim Q_{\mathbf{x}}} P(\mathbf{x})
\end{aligned}$$

Here, our goal is to minimise the KL divergence between $Q_{\mathbf{x}}(z)$ and $P(z|\mathbf{x})$. Alternatively, we aim to maximize $\mathbb{E}_{z\sim Q_{\mathbf{x}}} \log P(\mathbf{x}|z) - \mathrm{KL}(Q_{\mathbf{x}}(z), P(z))$. The first part in this quantity aims to maximize the likelihood, whereas the second part tries to minimize the KL divergence between $Q_{\mathbf{x}}(z)$ and $P(z|\mathbf{x})$. The expressed we obtained here is also referred to as the Evidence Lower Bound(**ELBO**) function [1]. Thus, our objective function will be $\sum_{i=1}^{N} \mathbb{E}_{z\sim Q_{\mathbf{x_i}}} \log P(\mathbf{x_i}|z) - \mathrm{KL}(Q_{\mathbf{x_i}}(z), P(z))$. We aim to maximize ELBO and identify the underlying parameters which give us the maximum.

Please refer to [2] and [3] for more info on Variational Auto encoders

References

[1] Understanding variational autoencoders. `https://en.wikipedia.org/wiki/Evidence_lower_bound`.

[2] C. Doersch. Tutorial on variational autoencoders. `https://arxiv.org/pdf/1606.05908.pdf`, 2016.

[3] J. Rocca. Understanding variational autoencoders. `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`, 2019.