

Machine Learning for Greenhorns

RNDr. Milan Straka, Ph.D.

January 18, 2022

Contents

| | | |
|----------|----------------|-----------|
| 1 | Lecture | 4 |
| 1.1 | Q | 4 |
| 1.2 | Q | 4 |
| 2 | Lecture | 5 |
| 2.1 | Q | 5 |
| 2.2 | Q | 5 |
| 2.3 | Q | 6 |
| 2.4 | Q | 6 |
| 3 | Lecture | 7 |
| 3.1 | Q | 7 |
| 3.2 | Q | 8 |
| 3.3 | Q | 8 |
| 3.4 | Q | 9 |
| 3.5 | Q | 9 |
| 3.6 | Q | 10 |
| 3.7 | Q | 11 |
| 4 | Lecture | 11 |
| 4.1 | Q | 11 |
| 4.2 | Q | 11 |
| 4.3 | Q | 12 |
| 4.4 | Q | 12 |
| 4.5 | Q | 12 |
| 4.6 | Q | 13 |
| 4.7 | Q | 13 |
| 4.8 | Q | 14 |
| 5 | Lecture | 15 |
| 5.1 | Q | 15 |
| 5.2 | Q | 15 |
| 5.3 | Q | 16 |
| 5.4 | Q | 17 |
| 5.5 | Q | 18 |
| 5.6 | Q | 18 |
| 6 | Lecture | 19 |
| 6.1 | Q | 19 |
| 6.2 | Q | 19 |
| 6.3 | Q | 19 |
| 6.4 | Q | 20 |
| 6.5 | Q | 21 |
| 6.6 | Q | 22 |
| 7 | Lecture | 22 |
| 7.1 | Q | 22 |
| 7.2 | Q | 23 |
| 7.3 | Q | 23 |
| 7.4 | Q | 23 |

| | | |
|-----------|----------------|-----------|
| 7.5 | Q | 24 |
| 7.6 | Q | 25 |
| 7.7 | Q | 25 |
| 8 | Lecture | 26 |
| 8.1 | Q | 26 |
| 8.2 | Q | 26 |
| 8.3 | Q | 26 |
| 8.4 | Q | 27 |
| 8.5 | Q | 27 |
| 8.6 | Q | 28 |
| 8.7 | Q | 28 |
| 8.8 | Q | 29 |
| 8.9 | Q | 30 |
| 9 | Lecture | 31 |
| 9.1 | Q | 31 |
| 9.2 | Q | 31 |
| 9.3 | Q | 32 |
| 9.4 | Q | 32 |
| 9.5 | Q | 33 |
| 9.6 | Q | 34 |
| 9.7 | Q | 34 |
| 9.8 | Q | 34 |
| 9.9 | Q | 34 |
| 9.10 | Q | 35 |
| 10 | Lecture | 35 |
| 10.1 | Q | 35 |
| 10.2 | Q | 36 |
| 10.3 | Q | 37 |
| 10.4 | Q | 37 |
| 10.5 | Q | 37 |
| 11 | Lecture | 38 |
| 11.1 | Q | 38 |
| 11.2 | Q | 38 |
| 11.3 | Q | 38 |
| 11.4 | Q | 39 |
| 11.5 | Q | 39 |
| 11.6 | Q | 39 |
| 12 | Lecture | 39 |
| 12.1 | Q | 39 |
| 12.2 | Q | 39 |
| 12.3 | Q | 39 |
| 13 | Lecture | 39 |
| 13.1 | Q | 39 |
| 13.2 | Q | 40 |
| 13.3 | Q | 40 |

| | |
|------------------|----|
| 13.4 Q | 40 |
| 13.5 Q | 40 |
| 13.6 Q | 40 |
| 13.7 Q | 40 |
| 13.8 Q | 40 |
| 13.9 Q | 40 |
| 13.10Q | 41 |
| 13.11Q | 41 |
| 13.12Q | 41 |

1 Lecture

1.1 Q

Define the prediction function of a linear regression model and write down L_2 -regularized mean squared error loss. [5]

Answer: Prediction function of linear regression:

$$y(\mathbf{x}; \mathbf{w}, b) = x_1 w_1 + x_2 w_2 + \dots + x_D w_D + b = \sum_{i=1}^D x_i w_i + b = \mathbf{x}^T \mathbf{w} + b$$

The \mathbf{w} are usually called *weights* and b is called *bias*.

Alternatively via matrix:

Using an explicit bias term in the form of $y(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + b$.

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = \begin{bmatrix} w_1 x_{11} + w_2 x_{12} + b \\ w_1 x_{21} + w_2 x_{22} + b \\ \vdots \\ w_1 x_{n1} + w_2 x_{n2} + b \end{bmatrix}$$

With extra 1 padding in \mathbf{X} and an additional b weight representing the bias.

$$\begin{bmatrix} x_{11} & x_{12} & 1 \\ x_{21} & x_{22} & 1 \\ \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} w_1 x_{11} + w_2 x_{12} + b \\ w_1 x_{21} + w_2 x_{22} + b \\ \vdots \\ w_1 x_{n1} + w_2 x_{n2} + b \end{bmatrix}$$

L_2 -regularized mean squared error loss, MSE:

L_2 regularization (also called weighted decay) penalizes models with large weights:

$$\frac{1}{2} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Alternatively via matrix:

$$\frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

1.2 Q

Starting from the unregularized sum of squares error of a linear regression model, show how the explicit solution can be obtained, assuming $(\mathbf{X}^T \mathbf{X})$ is regular. [10]

In order to find a minimum of $\frac{1}{2} \sum_i^N (\mathbf{x}_i^T \mathbf{w} - t_i)^2$, we can inspect values where the derivative of the error function is zero, with respect to all weights w_j .

$$\frac{\partial}{\partial w_j} \frac{1}{2} \sum_i^N (\mathbf{x}_i^T \mathbf{w} - t_i)^2 = \frac{1}{2} \sum_i^N (2(\mathbf{x}_i^T \mathbf{w} - t_i) x_{ij}) = \sum_i^N x_{ij} (\mathbf{x}_i^T \mathbf{w} - t_i)$$

Therefore, we want for all j that $\sum_i^N x_{ij} (\mathbf{x}_i^T \mathbf{w} - t_i) = 0$. We can write all the equations together using matrix notation as $\mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t}) = 0$ and rewrite to

$$\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{t}.$$

The matrix $\mathbf{X}^T \mathbf{X}$ is of size $D \times D$. If it is regular, we can compute its inverse and therefore

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}.$$

Answer:

2 Lecture

2.1 Q

Define expectation $\mathbb{E}(f(x))$ and variance $\text{Var}[f(x)]$ of a discrete random variable. Then define the bias of an estimator and show that estimating an expectation using a single sample is unbiased. [5]

Answer:

Definition 2.1 (Expected value). $\mathbb{E}(f(x))$ with respect to discrete probability distribution $P(x)$:

$$\mathbb{E}_{X \sim P}[f(x)] = \sum_x P(x) \cdot f(x)$$

Linearity:

$$\mathbb{E}[\alpha X + \beta Y] = \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y]$$

Definition 2.2 (Variance). Definition of $\text{Var}[f(x)]$:

Variance measures how much the values of a random variable differ from its mean $\mu = \mathbb{E}[x]$.

$$\text{Var}(x) \stackrel{\text{def}}{=} \mathbb{E} \left[(x - \mathbb{E}[x])^2 \right], \text{ or more generally,}$$
$$\text{Var}(f(x)) \stackrel{\text{def}}{=} \mathbb{E} \left[(f(x) - \mathbb{E}[f(x)])^2 \right].$$

Show that estimating an expectation using a single sample is unbiased:

An **estimator** is a rule for computing an estimate of a given value, often an expectation of some random value(s).

For example, we might estimate *mean* of random variable by sampling a value according to its probability distribution.

Bias of an estimator is the difference of the expected value of the estimator and the true value being estimated:

$$\text{bias} = \mathbb{E}[\text{estimate}] - \text{true estimated value}.$$

If the bias is zero, we call the estimator **unbiased**, otherwise we call it **biased**.

As an example, consider estimating $\mathbb{E}_P[f(x)]$ by generating a single sample x from P and returning $f(x)$. Such an estimate is unbiased, because $\mathbb{E}[\text{estimate}] = \mathbb{E}_P[f(x)]$, which is exactly the true estimated value.

2.2 Q

Describe standard gradient descent and compare it to stochastic (i.e., online) gradient descent and minibatch stochastic gradient descent. [5]

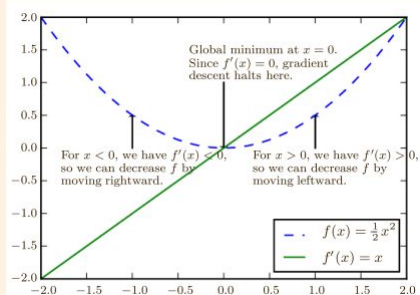
Assuming we are minimizing an error function

$$\arg \min_{\mathbf{w}} E(\mathbf{w}),$$

we may use **gradient descent**:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w})$$

The constant α is called a **learning rate** and specifies the "length" of a step we perform in every iteration of the gradient descent.



Answer:

Consider an error function computed as an expectation over the dataset:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \nabla_{\mathbf{w}} \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} L(\mathbf{y}(\mathbf{x}; \mathbf{w}), t).$$

- **(Regular) Gradient Descent:** We use all training data to compute $\nabla_{\mathbf{w}} E(\mathbf{w})$ exactly.
- **Online (or Stochastic) Gradient Descent:** We estimate $\nabla_{\mathbf{w}} E(\mathbf{w})$ using a single random example from the training data. Such an estimate is unbiased, but very noisy.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \nabla_{\mathbf{w}} L(\mathbf{y}(\mathbf{x}; \mathbf{w}), t) \text{ for randomly chosen } (\mathbf{x}, t) \text{ from } \hat{p}_{\text{data}}.$$

- **Minibatch SGD:** The minibatch SGD is a trade-off between gradient descent and SGD – the expectation in $\nabla_{\mathbf{w}} E(\mathbf{w})$ is estimated using m random independent examples from the training data.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \frac{1}{m} \sum_{i=1}^m \nabla_{\mathbf{w}} L(\mathbf{y}(\mathbf{x}_i; \mathbf{w}), t_i) \text{ for randomly chosen } (\mathbf{x}_i, t_i) \text{ from } \hat{p}_{\text{data}}.$$

2.3 Q

Formulate conditions on the sequence of learning rates used in SGD to converge to optimum almost surely. [5]

Assume that we perform a stochastic gradient descent, using a sequence of learning rates α_i , and using a noisy estimate $J(\mathbf{w})$ of the real gradient $\nabla_{\mathbf{w}} E(\mathbf{w})$:

$$\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha_i J(\mathbf{w}_i).$$

It can be proven (under some reasonable conditions; see Robbins and Monro algorithm, 1951) that if the loss function L is convex and continuous, then SGD converges to the unique optimum almost surely if the sequence of learning rates α_i fulfills the following conditions:

$$\forall i : \alpha_i > 0, \quad \sum_i \alpha_i = \infty, \quad \sum_i \alpha_i^2 < \infty.$$

Note that the third condition implies that $\alpha_i \rightarrow 0$.

For non-convex loss functions, we can get guarantees of converging to a *local* optimum only. However, note that finding a global minimum of an arbitrary function is *at least NP-hard*.

Answer:

2.4 Q

Write an L_2 -regularized minibatch SGD algorithm for training a linear regression model, including the explicit formulas of the loss function and its gradient. [5]

Answer: Loss function and gradient:

To apply SGD on linear regression, we usually minimize one half of mean squared error:

$$E(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (y(\mathbf{x}; \mathbf{w}) - t)^2 \right] = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (\mathbf{x}^T \mathbf{w} - t)^2 \right].$$

If we also include L_2 regularization, we get

$$E(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (\mathbf{x}^T \mathbf{w} - t)^2 \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

We then estimate the expectation by a minibatch \mathbf{b} of examples as

$$\sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} \left(\frac{1}{2} (\mathbf{x}_i^T \mathbf{w} - t_i)^2 \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2,$$

which gives us an **estimate of a gradient**

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} \left((\mathbf{x}_i^T \mathbf{w} - t_i) \mathbf{x}_i \right) + \lambda \mathbf{w}.$$

Algorithm:

The computed gradient allows us to formulate the following algorithm for solving linear regression with minibatch SGD.

Input: Dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \mathbb{R}^N)$, learning rate $\alpha \in \mathbb{R}^+$, L_2 strength $\lambda \in \mathbb{R}$.

Output: Weights $\mathbf{w} \in \mathbb{R}^D$ which hopefully minimize regularized MSE of linear regression.

- $\mathbf{w} \leftarrow \mathbf{0}$
- repeat until convergence (or until our patience runs out):
 - sample a batch \mathbf{b} (either uniformly randomly; or we may want to process all training instances before repeating them, which can be implemented by generating a random permutation and then splitting it to batch-sizes chunks)
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} ((\mathbf{x}_i^T \mathbf{w} - t_i) \mathbf{x}_i) - \alpha \lambda \mathbf{w}$

3 Lecture

3.1 Q

Define binary classification, write down the perceptron algorithm and show how a prediction is made for a given example. [5]

Answer: Binary classification:

Binary classification is a classification in two classes.

To extend linear regression to binary classification, we might seek a threshold and then classify an input as negative/positive depending whether $y(\mathbf{x}; \mathbf{w}) = \mathbf{x}^T \mathbf{w} + b$ is smaller/larger than a given threshold.

Zero value is usually used as the threshold, both because of symmetry and also because the bias parameter acts as a trainable threshold anyway.

- Consider two points on the decision boundary. Because $y(\mathbf{x}_1; \mathbf{w}) = y(\mathbf{x}_2; \mathbf{w})$, we have $(\mathbf{x}_1 - \mathbf{x}_2)^T \mathbf{w} = 0$, and so \mathbf{w} is orthogonal to every vector on the decision surface – \mathbf{w} is a normal of the boundary.
- Consider \mathbf{x} and let \mathbf{x}_\perp be orthogonal projection of \mathbf{x} to the boundary, so we can write $\mathbf{x} = \mathbf{x}_\perp + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$. Multiplying both sides by \mathbf{w}^T and adding b , we get that the distance of \mathbf{x} to the boundary is $r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}$.
- The distance of the decision boundary from origin is therefore $\frac{|b|}{\|\mathbf{w}\|}$.

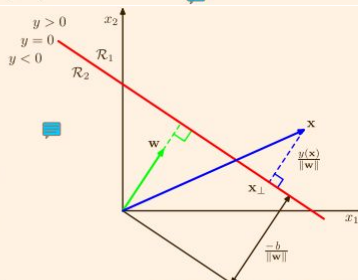


Figure 4.1 of Pattern Recognition and Machine Learning

Perceptron algorithm:

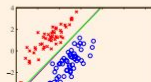
The perceptron algorithm is probably the oldest one for training weights of a binary classification. Assuming the target value $t \in \{-1, +1\}$, the goal is to find weights \mathbf{w} such that for all train data,

$$\text{sign}(y(\mathbf{x}_i; \mathbf{w})) = \text{sign}(\mathbf{x}_i^T \mathbf{w}) = t_i,$$

or equivalently,

$$t_i y(\mathbf{x}_i; \mathbf{w}) = t_i \mathbf{x}_i^T \mathbf{w} > 0.$$

Note that a set is called linearly separable, if there exists a weight vector \mathbf{w} such that the above equation holds.



The perceptron algorithm was invented by Rosenblatt in 1958.

Input: Linearly separable dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \{-1, +1\}^N)$.

Output: Weights $\mathbf{w} \in \mathbb{R}^D$ such that $t_i \mathbf{x}_i^T \mathbf{w} > 0$ for all i .

- $\mathbf{w} \leftarrow \mathbf{0}$
- until all examples are classified correctly, process example i :
 - $y \leftarrow \mathbf{x}_i^T \mathbf{w}$
 - if $t_i y \leq 0$ (incorrectly classified example):
 - $\mathbf{w} \leftarrow \mathbf{w} + t_i \mathbf{x}_i$

We will prove that the algorithm always arrives at some correct set of weights \mathbf{w} if the training set is linearly separable.

Consider the main part of the perceptron algorithm:

- $y \leftarrow \mathbf{x}_i^T \mathbf{w}$
- if $t_i y \leq 0$ (incorrectly classified example):
 - $\mathbf{w} \leftarrow \mathbf{w} + t_i \mathbf{x}_i$

We can derive the algorithm using on-line gradient descent, using the following loss function

$$L(y(\mathbf{x}; \mathbf{w}), t) \stackrel{\text{def}}{=} \begin{cases} -t\mathbf{x}^T \mathbf{w} & \text{if } t\mathbf{x}^T \mathbf{w} \leq 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, -t\mathbf{x}^T \mathbf{w}) = \text{ReLU}(-t\mathbf{x}^T \mathbf{w}).$$

In this specific case, the value of the learning rate does not actually matter, because multiplying \mathbf{w} by a constant does not change prediction and does not change the loss derivative. Note that the second condition is crucial; the first holds also for logistic regression, but there learning rate matters.

3.2 Q

Show that the perceptron algorithm is an instance of stochastic gradient descent. Why are the learning rates not needed (i.e., why does not the result of the training depend on the learning rate)? [5]

Consider the main part of the perceptron algorithm:

- $y \leftarrow \mathbf{x}_i^T \mathbf{w}$
- if $t_i y \leq 0$ (incorrectly classified example):
 - $\mathbf{w} \leftarrow \mathbf{w} + t_i \mathbf{x}_i$

We can derive the algorithm using on-line gradient descent, using the following loss function

$$L(y(\mathbf{x}; \mathbf{w}), t) \stackrel{\text{def}}{=} \begin{cases} -t\mathbf{x}^T \mathbf{w} & \text{if } t\mathbf{x}^T \mathbf{w} \leq 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, -t\mathbf{x}^T \mathbf{w}) = \text{ReLU}(-t\mathbf{x}^T \mathbf{w}).$$

In this specific case, the value of the learning rate does not actually matter, because multiplying \mathbf{w} by a constant does not change prediction and does not change the loss derivative. Note that the second condition is crucial; the first holds also for logistic regression, but there learning rate matters.

Answer:

3.3 Q

Define entropy, cross-entropy, Kullback-Leibler divergence, and prove the Gibbs inequality (i.e., that KL divergence is non-negative). [5]

Entropy

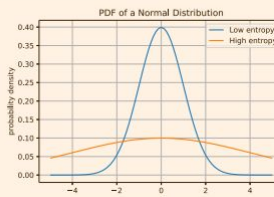
Amount of **surprise** in the whole distribution.

$$H(P) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{x} \sim P} [I(\mathbf{x})] = -\mathbb{E}_{\mathbf{x} \sim P} [\log P(\mathbf{x})]$$

- for discrete P : $H(P) = -\sum_{\mathbf{x}} P(\mathbf{x}) \log P(\mathbf{x})$
- for continuous P : $H(P) = -\int P(\mathbf{x}) \log P(\mathbf{x}) \, d\mathbf{x}$

Note that in the continuous case, the continuous entropy (also called *differential entropy*) has slightly different semantics, for example, it can be negative.

From now on, all logarithms are *natural logarithms* with base e .



Answer:

Cross-Entropy

$$H(P, Q) \stackrel{\text{def}}{=} -\mathbb{E}_{\mathbf{x} \sim P} [\log Q(\mathbf{x})]$$

Gibbs Inequality

- $H(P, Q) \geq H(P)$
- $H(P) = H(P, Q) \Leftrightarrow P = Q$

Proof: Consider $H(P) - H(P, Q) = \sum_x P(x) \log \frac{Q(x)}{P(x)}$.

Using the fact that $\log x \leq (x - 1)$ with equality only for $x = 1$, we get

$$\sum_x P(x) \log \frac{Q(x)}{P(x)} \leq \sum_x P(x) \left(\frac{Q(x)}{P(x)} - 1 \right) = \sum_x Q(x) - \sum_x P(x) = 0.$$

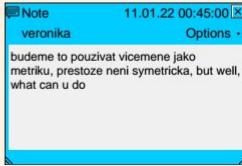
For the equality to hold, $\frac{Q(x)}{P(x)}$ must be 1 for all x , i.e., $P = Q$.

Kullback-Leibler Divergence (KL Divergence)

Sometimes also called **relative entropy**.

$$D_{\text{KL}}(P||Q) \stackrel{\text{def}}{=} H(P, Q) - H(P) = \mathbb{E}_{x \sim P}[\log P(x) - \log Q(x)]$$

- consequence of Gibbs inequality: $D_{\text{KL}}(P||Q) \geq 0$
- generally $D_{\text{KL}}(P||Q) \neq D_{\text{KL}}(Q||P)$



3.4 Q

Define data distribution, empirical data distribution and likelihood. [5]

Definice 2.5 (Rozdělení náhodné veličiny). Buď $X : \Omega \rightarrow \mathbb{R}$ náhodná veličina. Pravděpodobnostní míra P_X na borelovských podmnožinách \mathbb{R} splňující pro každé $a \in \mathbb{R}$ rovnost $P_X(-\infty, a] = \mathbb{P}[X \leq a]$ se nazývá **rozdělení náhodné veličiny** X .

Pokud se nechceme trápit teorií míry, stačí nám vědět, že P_X je definována pro každý interval typu (a, b) , $[a, b)$, $(a, b]$ i $[a, b]$ a splňuje vlastnosti pravděpodobnostní míry, čili se dá rozšířit na všechna spočetná sjednocení i průniky intervalů (a dále na borelovské množiny, což už nás pro začátek tolik nepálí).

Čeho si však všimneme je

- P je pravděpodobnost definovaná na základním prostoru (Ω, \mathcal{F}) .
- P_X je obrazem míry P v zobrazení X a jde o pravděpodobnost definovanou na prostoru $(\mathbb{R}, \mathcal{B})$. Hodnoty $P_X(A)$ pro $A \in \mathcal{B}$ jsou dány identitou $P_X(A) = \mathbb{P}[X \in A]$.

Answer:

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ be training data drawn independently from the data-generating distribution p_{data} .

We denote the **empirical data distribution** as \hat{p}_{data} , where

$$\hat{p}_{\text{data}}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{|\{i : \mathbf{x}_i = \mathbf{x}\}|}{N}$$

Let $p_{\text{model}}(\mathbf{x}; \mathbf{w})$ be a family of distributions.

- If the weights are fixed, $p_{\text{model}}(\mathbf{x}; \mathbf{w})$ is a probability distribution.
- If we instead consider the fixed training data \mathbf{X} , then

$$L(\mathbf{w}) = p_{\text{model}}(\mathbf{X}; \mathbf{w}) = \prod_{i=1}^N p_{\text{model}}(\mathbf{x}_i; \mathbf{w})$$

is called the **likelihood**. Note that even if the value of the likelihood is in range $[0, 1]$, it is not a probability, because likelihood is not a probability distribution.

Pozn.: explicitly mentioned Gauss/normal distribution

Normal (or Gaussian) Distribution

Distribution over real numbers, parametrized by a mean μ and variance σ^2 :

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

For standard values $\mu = 0$ and $\sigma^2 = 1$ we get $\mathcal{N}(x; 0, 1) = \sqrt{\frac{1}{2\pi}} e^{-\frac{x^2}{2}}$.

3.5 Q

Describe maximum likelihood estimation, as minimizing NLL, cross-entropy and KL divergence. [10]

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ be training data drawn independently from the data-generating distribution p_{data} . We denote the empirical data distribution as \hat{p}_{data} and let $p_{\text{model}}(\mathbf{x}; \mathbf{w})$ be a family of distributions.

The **maximum likelihood estimation** of \mathbf{w} is:

$$\begin{aligned} \mathbf{w}_{\text{MLE}} &= \arg \max_{\mathbf{w}} p_{\text{model}}(\mathbf{X}; \mathbf{w}) = \arg \max_{\mathbf{w}} \prod_{i=1}^N p_{\text{model}}(\mathbf{x}_i; \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^N -\log p_{\text{model}}(\mathbf{x}_i; \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(\mathbf{x}; \mathbf{w})] \\ &= \arg \min_{\mathbf{w}} H(\hat{p}_{\text{data}}(\mathbf{x}), p_{\text{model}}(\mathbf{x}; \mathbf{w})) \\ &= \arg \min_{\mathbf{w}} D_{\text{KL}}(\hat{p}_{\text{data}}(\mathbf{x}) \| p_{\text{model}}(\mathbf{x}; \mathbf{w})) + H(\hat{p}_{\text{data}}) \end{aligned}$$

Answer:

MLE can be easily generalized to a conditional case, where our goal is to predict t given \mathbf{x} :

$$\begin{aligned} \mathbf{w}_{\text{MLE}} &= \arg \max_{\mathbf{w}} p_{\text{model}}(t|\mathbf{X}; \mathbf{w}) = \arg \max_{\mathbf{w}} \prod_{i=1}^m p_{\text{model}}(t_i|\mathbf{x}_i; \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^m -\log p_{\text{model}}(t_i|\mathbf{x}_i; \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(t|\mathbf{x}; \mathbf{w})] \\ &= \arg \min_{\mathbf{w}} H(\hat{p}_{\text{data}}, p_{\text{model}}(t|\mathbf{x}; \mathbf{w})) \\ &= \arg \min_{\mathbf{w}} D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}}(t|\mathbf{x}; \mathbf{w})) + H(\hat{p}_{\text{data}}) \end{aligned}$$

where the conditional entropy is defined as $H(Y|X) = H((X, Y)) - H(X)$, so $H(p) = \mathbb{E}_{(\mathbf{x}, t) \sim p} -\log p(t|\mathbf{x})$ and $H(p, q) = \mathbb{E}_{(\mathbf{x}, t) \sim p} -\log(q(t|\mathbf{x}; \mathbf{w})) = \mathbb{E}_{\mathbf{x} \sim p} H(p(t|\mathbf{x}), q(t|\mathbf{x}))$.

The resulting *loss function* is called **negative log likelihood**, or **cross-entropy** or **Kullback-Leibler divergence**.

3.6 Q

Considering binary logistic regression model, write down its parameters (including their size) and explain how is prediction performed (including the formula for the sigmoid function). Describe how we can interpret the outputs of the linear part of the model as logits. [5]

An extension of perceptron, which models the conditional probabilities of $p(C_0|\mathbf{x})$ and of $p(C_1|\mathbf{x})$. **Logistic regression** can in fact handle also more than two classes, which we will see on the next lecture.

Logistic regression employs the following parametrization of the conditional class probabilities:

$$\begin{aligned} p(C_1|\mathbf{x}) &= \sigma(\mathbf{x}^T \mathbf{w} + b) \\ p(C_0|\mathbf{x}) &= 1 - p(C_1|\mathbf{x}), \end{aligned}$$

where σ is a **sigmoid function**

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

Answer: It can be trained using the SGD algorithm.

We denote the output of the "linear part" of the logistic regression as

$$\tilde{y}(\mathbf{x}; \mathbf{w}) = \mathbf{x}^T \mathbf{w},$$

and the overall prediction as

$$y(\mathbf{x}; \mathbf{w}) = \sigma(\tilde{y}(\mathbf{x}; \mathbf{w})) = \sigma(\mathbf{x}^T \mathbf{w}).$$

The logistic regression output $y(\mathbf{x}; \mathbf{w})$ models the probability of class C_1 , $p(C_1|\mathbf{x})$.

To give some meaning to the output of the linear part $\tilde{y}(\mathbf{x}; \mathbf{w})$, starting with

$$p(C_1|\mathbf{x}) = \sigma(\tilde{y}(\mathbf{x}; \mathbf{w})) = \frac{1}{1 + e^{-\tilde{y}(\mathbf{x}; \mathbf{w})}},$$

we arrive at

$$\tilde{y}(\mathbf{x}; \mathbf{w}) = \log \left(\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})} \right) = \log \left(\frac{p(C_1|\mathbf{x})}{p(C_0|\mathbf{x})} \right),$$

which is called a **logit** and it is a logarithm of odds of the probabilities of the two classes.

3.7 Q

Write down an L_2 -regularized minibatch SGD algorithm for training a binary logistic regression model, including the explicit formulas of the loss function and its gradient. [10]

To train the logistic regression, we use MLE (the maximum likelihood estimation). Its application is straightforward, given that $p(C_1|\mathbf{x}; \mathbf{w})$ is directly the model output $y(\mathbf{x}; \mathbf{w})$.

Therefore, the loss for a batch $\mathbf{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ is

$$E(\mathbf{w}) = \frac{1}{N} \sum_i -\log(p(C_{t_i}|\mathbf{x}_i; \mathbf{w})).$$

Input: Input dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \{0, +1\}^N)$, learning rate $\alpha \in \mathbb{R}^+$.

- $\mathbf{w} \leftarrow 0$
- until convergence (or until patience is over), process batch of N examples:
 - $\mathbf{g} \leftarrow \frac{1}{N} \sum_i -\nabla_{\mathbf{w}} \log(p(C_{t_i}|\mathbf{x}_i; \mathbf{w}))$
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}$

Answer:

| Name | Activation | Distribution | Loss | Gradient |
|---------------------|-------------|--------------|--|---------------------------------|
| linear regression | identity | Normal | NLL \propto MSE | $(y(\mathbf{x}) - t)\mathbf{x}$ |
| logistic regression | $\sigma(y)$ | Bernoulli | NLL $\propto \mathbb{E} - \log(p(t \mathbf{x}))$ | $(y(\mathbf{x}) - t)\mathbf{x}$ |

Pozn.: primým vypočtem

4 Lecture

4.1 Q

Define mean squared error and show how it can be derived using MLE. [5]

During regression, we predict a number, not a real probability distribution. In order to generate a distribution, we might consider a distribution with the mean of the predicted value and a fixed variance σ^2 – the most general such a distribution is the normal distribution.

Answer:

Therefore, assume our model generates a distribution $p(t|\mathbf{x}; \mathbf{w}) = \mathcal{N}(t; y(\mathbf{x}; \mathbf{w}), \sigma^2)$.

Now we can apply MLE and get

$$\begin{aligned} \arg \max_{\mathbf{w}} p(\mathbf{t}|\mathbf{X}; \mathbf{w}) &= \arg \min_{\mathbf{w}} \sum_{i=1}^N -\log p(t_i|\mathbf{x}_i; \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} -\sum_{i=1}^N \log \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(t_i - y(\mathbf{x}_i; \mathbf{w}))^2}{2\sigma^2}} \\ &= \arg \min_{\mathbf{w}} -N \log(2\pi\sigma^2)^{-1/2} - \sum_{i=1}^N -\frac{(t_i - y(\mathbf{x}_i; \mathbf{w}))^2}{2\sigma^2} \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^N \frac{(t_i - y(\mathbf{x}_i; \mathbf{w}))^2}{2\sigma^2} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N (t_i - y(\mathbf{x}_i; \mathbf{w}))^2. \end{aligned}$$

4.2 Q

Considering K -class logistic regression model, write down its parameters(including their size) and explain how is prediction performed (including the formula for the softmax function). Describe how we can interpret the outputs of the linear part of the model as logits. [5]

To extend the binary logistic regression to a multiclass case with K classes, we:

- generate K outputs, each with its own set of weights, so that for $\mathbf{W} \in \mathbb{R}^{D \times K}$,

$$\bar{\mathbf{y}}(\mathbf{x}; \mathbf{W}) = \mathbf{x}^T \mathbf{W}, \text{ or in other words, } \bar{y}(\mathbf{x}; \mathbf{W})_i = \mathbf{x}^T (\mathbf{W}_{*,i})$$

- generalize the sigmoid function to a softmax function, such that

$$\text{softmax}(y)_i = \frac{e^{y_i}}{\sum_j e^{y_j}}.$$

Answer:

Using the softmax function, we naturally define that

$$p(C_i|\mathbf{x}; \mathbf{W}) = \mathbf{y}(\mathbf{x}; \mathbf{W})_i = \text{softmax}(\bar{\mathbf{y}}(\mathbf{x}; \mathbf{W}))_i = \text{softmax}(\mathbf{x}^T \mathbf{W})_i = \frac{e^{(\mathbf{x}^T \mathbf{W})_i}}{\sum_j e^{(\mathbf{x}^T \mathbf{W})_j}}$$

Considering the definition of the softmax function, it is natural to obtain the interpretation of the linear part of the model $\bar{\mathbf{y}}(\mathbf{x}; \mathbf{W})$ as **logits**:

$$\bar{\mathbf{y}}(\mathbf{x}; \mathbf{W})_i = \log(p(C_i|\mathbf{x}; \mathbf{W})) + c.$$

The constant c is present, because the output of the model is *overparametrized* (the probability of for example the last class could be computed from the remaining ones). This is connected to the fact that softmax is invariant to addition of a constant:

$$\text{softmax}(\mathbf{y} + c)_i = \frac{e^{y_i + c}}{\sum_j e^{y_j + c}} = \frac{e^{y_i}}{\sum_j e^{y_j}} \cdot \frac{e^c}{e^c} = \text{softmax}(\mathbf{y})_i.$$

Pozn.: maybe add explanation for the logits

4.3 Q

Write down an L_2 -regularized minibatch SGD algorithm for training a K -class logistic regression model, including the explicit formulas of the loss function and its gradient. [10]

| | | | | |
|--------------------------------|------------|-------------|--|--|
| multiclass logistic regression | softmax(y) | categorical | NLL $\propto \mathbb{E} - \log(p(t \mathbf{x}))$ | $(\mathbf{y}(\mathbf{x}) - \mathbf{1}_t) \mathbf{x}^T$ |
|--------------------------------|------------|-------------|--|--|

Answer:

To train K -class classification, analogously to the binary logistic regression we can use MLE and train the model using minibatch stochastic gradient descent:

- Input:** Input dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \{0, 1, \dots, K-1\}^N)$, learning rate $\alpha \in \mathbb{R}^+$.
Model: Let \mathbf{w} denote all parameters of the model (in our case, the parameters are a weight matrix \mathbf{W} and maybe a bias vector \mathbf{b}).
- $\mathbf{w} \leftarrow 0$
 - until convergence (or until patience is over), process batch of N examples:
 - $\mathbf{g} \leftarrow \frac{1}{N} \sum_i \nabla_{\mathbf{w}} - \log(p(C_{t_i}|\mathbf{x}_i; \mathbf{w}))$
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}$

L_2 -regularization for the update:

$$\text{weights} = \text{weights} - \text{args.learning_rate} \cdot \text{gradient}$$

4.4 Q

Prove why are decision regions of a multiclass logistic regression convex. [5]

Note that the decision regions of the binary/multiclass logistic regression are convex (and therefore connected). To see this, consider \mathbf{x}_A and \mathbf{x}_B in the same decision region R_k .

Any point \mathbf{x} lying on the line connecting them is their linear combination, $\mathbf{x} = \lambda \mathbf{x}_A + (1 - \lambda) \mathbf{x}_B$, and from the linearity of $\bar{\mathbf{y}}(\mathbf{x}) = \mathbf{x}^T \mathbf{W}$ it follows that

$$\bar{\mathbf{y}}(\mathbf{x}) = \lambda \bar{\mathbf{y}}(\mathbf{x}_A) + (1 - \lambda) \bar{\mathbf{y}}(\mathbf{x}_B).$$

Given that $\bar{\mathbf{y}}(\mathbf{x}_A)_k$ was the largest among $\bar{\mathbf{y}}(\mathbf{x}_A)$ and also given that $\bar{\mathbf{y}}(\mathbf{x}_B)_k$ was the largest among $\bar{\mathbf{y}}(\mathbf{x}_B)$, it must be the case that $\bar{\mathbf{y}}(\mathbf{x})_k$ is the largest among all $\bar{\mathbf{y}}(\mathbf{x})$.

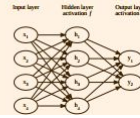
Figure 4.3 of Pattern Recognition and Machine Learning

Answer:

4.5 Q

Considering a single-layer MLP with D input neurons, H hidden neurons, K output neurons, hidden activation f and output activation a , list its parameters (including their size) and write down how is the output computed. [5]

Assume we have an MLP with input of size D , weights $\mathbf{W}^h \in \mathbb{R}^{D \times H}$, $\mathbf{b}^h \in \mathbb{R}^H$, hidden layer of size H and activation f with weights $\mathbf{W}^y \in \mathbb{R}^{H \times K}$, $\mathbf{b}^y \in \mathbb{R}^K$, and finally an output layer of size K with activation a .



Answer:

~~In order to compute the gradient of the loss L with respect to all weights,~~

We now extend the model by adding a **hidden layer** with activation f .

- The computation is performed analogically:

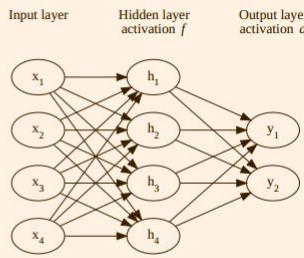
$$h_i = f\left(\sum_j x_j w_{j,i}^h + b_i^h\right),$$

$$y_i = a\left(\sum_j h_j w_{j,i}^y + b_i^y\right),$$

or in matrix form

$$\mathbf{h} = f(\mathbf{x}^T \mathbf{W}^h + \mathbf{b}^h),$$

$$\mathbf{y} = a(\mathbf{h}^T \mathbf{W}^y + \mathbf{b}^y),$$



and for batch of inputs $\mathbf{H} = f(\mathbf{X}\mathbf{W}^h + \mathbf{b}^h)$ and $\mathbf{Y} = a(\mathbf{H}\mathbf{W}^y + \mathbf{b}^y)$.

4.6 Q

List the definitions of frequently used MLP output layer activations (the ones producing parameters of a Bernoulli distribution and a categorical distribution). Then write down three commonly used hidden layer activations (sigmoid, tanh, ReLU). [5]

Output Layer Activation Functions

- regression:**
 - identity activation: we model normal distribution on output (linear regression)
 - $\exp(x)$: we model Poisson distribution on output (Poisson regression)
- binary classification:**
 - $\sigma(x)$: we model the Bernoulli distribution (the model predicts a probability)

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

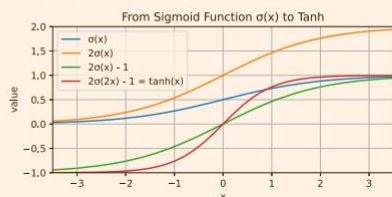
- K -class classification:**
 - softmax(x)**: we model the (usually overparametrized) categorical distribution

$$\text{softmax}(\mathbf{x}) \propto e^{\mathbf{x}}, \quad \text{softmax}(\mathbf{x})_i \stackrel{\text{def}}{=} \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Answer:

Hidden Layer Activation Functions

- no activation (**identity**): does not help, composition of linear mapping is a linear mapping
- σ (but works suboptimally – nonsymmetrical, $\frac{d\sigma}{dx}(0) = 1/4$)
- tanh**
 - result of making σ symmetrical and making derivation in zero 1
 - $\tanh(x) = 2\sigma(2x) - 1$
- ReLU**
 - $\max(0, x)$
 - the most common non-linear activation used nowadays

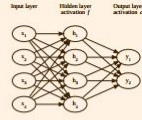


4.7 Q

Considering a single-layer MLP with D input neurons, a ReLU hidden layer with H units and softmax output layer with K units, write down the formulas of the gradient of all the MLP parameters (two weight matrices and two bias vectors), assuming input \mathbf{x} , target t and negative log likelihood loss. [10]

Answer: General steps to calculate the gradient:

Assume we have an MLP with input of size D , weights $\mathbf{W}^h \in \mathbb{R}^{D \times H}$, $\mathbf{b}^h \in \mathbb{R}^H$, hidden layer of size H and activation f with weights $\mathbf{W}^y \in \mathbb{R}^{H \times K}$, $\mathbf{b}^y \in \mathbb{R}^K$, and finally an output layer of size K with activation a .



In order to compute the gradient of the loss L with respect to all weights, you should proceed gradually:

- first compute $\frac{\partial L}{\partial \mathbf{y}}$,
- then compute $\frac{\partial \mathbf{y}}{\partial \mathbf{y}_{in}}$, where \mathbf{y}_{in} are the inputs to the output layer (i.e., before applying activation function a ; in other words, $\mathbf{y} = a(\mathbf{y}_{in})$),
- then compute $\frac{\partial \mathbf{y}_{in}}{\partial \mathbf{W}^y}$ and $\frac{\partial \mathbf{y}_{in}}{\partial \mathbf{b}^y}$, which allows us to obtain $\frac{\partial L}{\partial \mathbf{W}^y} = \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{y}_{in}} \cdot \frac{\partial \mathbf{y}_{in}}{\partial \mathbf{W}^y}$ and analogously $\frac{\partial L}{\partial \mathbf{b}^y}$,
- followed by $\frac{\partial \mathbf{y}_{in}}{\partial \mathbf{h}}$ and $\frac{\partial \mathbf{h}}{\partial \mathbf{h}_{in}}$,
- and finally using $\frac{\partial \mathbf{h}_{in}}{\partial \mathbf{W}^h}$ and $\frac{\partial \mathbf{h}_{in}}{\partial \mathbf{b}^h}$ to compute $\frac{\partial L}{\partial \mathbf{W}^h}$ and $\frac{\partial L}{\partial \mathbf{b}^h}$.

In our case we just substitute our given activation functions (code with the same activation functions just to check, on paper much easier):

```
weights = [generator.uniform(size=[train_data.shape[1], args.hidden_layer], low=-0.1, high=0.1),
generator.uniform(size=[args.hidden_layer, args.classes], low=-0.1, high=0.1)]
biases = [np.zeros(args.hidden_layer), np.zeros(args.classes)]

gradients_w = [np.zeros(weights[0].shape), np.zeros(weights[1].shape)]
gradients_b = [np.zeros(biases[0].shape), np.zeros(biases[1].shape)]

for idx in chunk:
    hidden, output = forward(train_data[idx])

    onevec = np.zeros(args.classes)
    onevec[train_target[idx]] = 1

    L__grad__y_in = softmax(hidden.T.dot(weights[1]) + biases[1]) - onevec

    gradients_b[1] += L__grad__y_in
    gradients_w[1] += np.outer(hidden, L__grad__y_in)

    L__grad__h = weights[1] @ L__grad__y_in

    L__grad__h_in = L__grad__h * np.array([h > 0 for h in hidden])

    gradients_b[0] += L__grad__h_in
    gradients_w[0] += np.outer(train_data[idx], L__grad__h_in)

gradients_w[0] /= len(chunk)
gradients_w[1] /= len(chunk)
gradients_b[0] /= len(chunk)
gradients_b[1] /= len(chunk)

weights[0] = weights[0] - args.learning_rate * gradients_w[0]
weights[1] = weights[1] - args.learning_rate * gradients_w[1]
biases[0] = biases[0] - args.learning_rate * gradients_b[0]
biases[1] = biases[1] - args.learning_rate * gradients_b[1]
```

4.8 Q

Formulate the Universal approximation theorem. [5]

Let $\varphi(x)$ be a nonconstant, bounded and nondecreasing continuous function. (Later a proof was given also for $\varphi = \text{ReLU}$ and even for any nonpolynomial function.)

For any $\varepsilon > 0$ and any continuous function $f : [0, 1]^D \rightarrow \mathbb{R}$, there exists $H \in \mathbb{N}$, $\mathbf{v} \in \mathbb{R}^H$, $\mathbf{b} \in \mathbb{R}^H$ and $\mathbf{W} \in \mathbb{R}^{D \times H}$, such that if we denote

$$F(\mathbf{x}) = \mathbf{v}^T \varphi(\mathbf{x}^T \mathbf{W} + \mathbf{b}) = \sum_{i=1}^H v_i \varphi(\mathbf{x}^T \mathbf{W}_{*,i} + b_i),$$

where φ is applied elementwise, then for all $\mathbf{x} \in [0, 1]^D$:

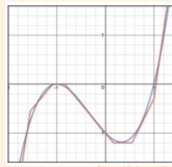
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon.$$

Answer:

Proof not required, sketch of it added for fun:

Sketch of the proof:

- If a function is continuous on a closed interval, it can be approximated by a sequence of lines to arbitrary precision.



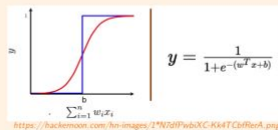
$$\begin{aligned} n_1(x) &= \text{ReLU}(-5x - 7.7) \\ n_2(x) &= \text{ReLU}(-1.2x - 1.3) \\ n_3(x) &= \text{ReLU}(1.2x + 1) \\ n_4(x) &= \text{ReLU}(1.2x - 2) \\ n_5(x) &= \text{ReLU}(2x - 1.1) \\ n_6(x) &= \text{ReLU}(5x - 5) \end{aligned}$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

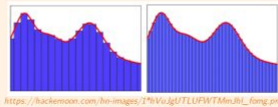
- However, we can create a sequence of k linear segments as a sum of k ReLU units – on every endpoint a new ReLU starts (i.e., the input ReLU value is zero at the endpoint), with a tangent which is the difference between the target tangent and the tangent of the approximation until this point.

Sketch of the proof for a squashing function $\varphi(x)$ (i.e., nonconstant, bounded and nondecreasing continuous function like sigmoid):

- We can prove φ can be arbitrarily close to a hard threshold by compressing it horizontally.



- Then we approximate the original function using a series of straight line segments



5 Lecture

5.1 Q

How do we search for a minimum of a function $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}$ subject to equality constraints $g_1(x) = 0, \dots, g_m(x) = 0$? [5]

Let $f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ be a function, which has a minimum (or a maximum) in x subject to equality constraints $g_1(x) = 0, \dots, g_m(x) = 0$. Assume that f, g_1, \dots, g_m have continuous partial derivatives and that the gradients $\nabla_x g_1(x), \dots, \nabla_x g_m(x)$ are linearly independent.

Then there exist $\lambda_1 \in \mathbb{R}, \dots, \lambda_m \in \mathbb{R}$, such that the **Lagrangian function**

$$\mathcal{L}(x, \lambda) \stackrel{\text{def}}{=} f(x) - \sum_{i=1}^m \lambda_i g_i(x)$$

has zero gradient in both x and λ .

Answer:

This strategy of finding constrained minima is known as the **method of Lagrange multipliers**.

Example:

Assume we want to find a categorical distribution p_1, \dots, p_n with maximum entropy.

Then we want to minimize $-H(p)$ under the constraints

- $p_i \geq 0$ for all i ,
- $\sum_{i=1}^n p_i = 1$.

Ignoring the first constraint for the time being, we form a Lagrangian

$$\mathcal{L} = \left(\sum_i p_i \log p_i \right) - \lambda \left(\sum_i p_i - 1 \right).$$

Computing the derivative with respect to p_i and setting it equal to zero, we get

$$0 = \frac{\partial \mathcal{L}}{\partial p_i} = 1 \cdot \log(p_i) + p_i \cdot \frac{1}{p_i} - \lambda = \log(p_i) + 1 - \lambda.$$

Therefore, all $p_i = e^{\lambda-1}$ must be the same, and the constraint $\sum_{i=1}^n p_i = 1$ yields $p_i = \frac{1}{n}$.

5.2 Q

Prove which categorical distribution with N classes has maximum entropy. [5]

Answer: Softmax

Let $\mathbf{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ be training data of a K -class classification, with $\mathbf{x}_i \in \mathbb{R}^D$ and $t_i \in \{1, 2, \dots, K\}$.

We want to model it using a function $\pi : \mathbb{R}^D \rightarrow \mathbb{R}^K$ so that $\pi(\mathbf{x})$ gives a distribution of classes for input \mathbf{x} .

We impose the following conditions on π :

- $\forall 1 \leq k \leq K : \pi(\mathbf{x})_k \geq 0$,
- $\sum_{k=1}^K \pi(\mathbf{x})_k = 1$,
- $\forall 1 \leq k \leq K : \sum_{i=1}^N \pi(\mathbf{x}_i)_k \mathbf{x}_i = \sum_{i=1}^N [t_i == k] \mathbf{x}_i$.

There are many such π , one particularly bad is

$$\pi(\mathbf{x}) = \begin{cases} \mathbf{1}_{t_i} & \text{if there exists } i : \mathbf{x}_i = \mathbf{x}, \\ \mathbf{1}_0 & \text{otherwise,} \end{cases}$$

where $\mathbf{1}_i$ is a one-hot encoding of i (vector of zeros, except for position i , which is equal to 1).

Therefore, we want to find a more **general** π – consequently, we turn to the principle of maximum entropy and search for π with maximum entropy.

We want to minimize $-\sum_{i=1}^N H(\pi(\mathbf{x}_i))$ given

- $\forall 1 \leq i \leq N, \forall 1 \leq k \leq K : \pi(\mathbf{x}_i)_k \geq 0$,
- $\forall 1 \leq i \leq N : \sum_{k=1}^K \pi(\mathbf{x}_i)_k = 1$,
- $\forall 1 \leq j \leq D, \forall 1 \leq k \leq K : \sum_{i=1}^N \pi(\mathbf{x}_i)_k \mathbf{x}_{i,j} = \sum_{i=1}^N [t_i == k] \mathbf{x}_{i,j}$.

We therefore form a Lagrangian (ignoring the first inequality constraint):

$$\begin{aligned} \mathcal{L} = & \sum_{i=1}^N \sum_{k=1}^K \pi(\mathbf{x}_i)_k \log(\pi(\mathbf{x}_i)_k) \\ & - \sum_{j=1}^D \sum_{k=1}^K \lambda_{j,k} \left(\sum_{i=1}^N \pi(\mathbf{x}_i)_k \mathbf{x}_{i,j} - [t_i == k] \mathbf{x}_{i,j} \right) \\ & - \sum_{i=1}^N \beta_i \left(\sum_{k=1}^K \pi(\mathbf{x}_i)_k - 1 \right). \end{aligned}$$

We now compute partial derivatives of the Lagrangian, notably the values

$$\frac{\partial}{\partial \pi(\mathbf{x}_i)_k} \mathcal{L}.$$

We arrive at

$$\frac{\partial}{\partial \pi(\mathbf{x}_i)_k} \mathcal{L} = \log(\pi(\mathbf{x}_i)_k) + 1 - \mathbf{x}_i^T \boldsymbol{\lambda}_{*,k} - \beta_i.$$

Setting the Lagrangian to zero, we obtain

$$\pi(\mathbf{x}_i)_k = e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1}.$$

Such a form guarantees $\pi(\mathbf{x}_i)_k > 0$, which we did not include in the conditions.

In order to find out the β_i values, we turn to the constraint

$$\sum_k \pi(\mathbf{x}_i)_k = \sum_k e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1} = 1,$$

from which we get

$$e^{\beta_i} = \frac{1}{\sum_k e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k} - 1}},$$

yielding

$$\pi(\mathbf{x}_i)_k = e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1} = \frac{e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k}}}{\sum_{k'} e^{\mathbf{x}_i^T \boldsymbol{\lambda}_{*,k'}}} = \text{softmax}(\mathbf{x}_i \boldsymbol{\lambda})_k.$$

5.3 Q

Consider derivation of softmax using maximum entropy principle, assuming we have a dataset of N examples $(x_i, t_i), x_i \in \mathbb{R}^D, t_i \in \{1, 2, \dots, K\}$. Formulate the three conditions we impose on the searched $\pi : \mathbb{R}^D \rightarrow \mathbb{R}^K$, and write down the Lagrangian to be maximized. [10]

Let $\mathbf{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\}$ be training data of a K -class classification, with $\mathbf{x}_i \in \mathbb{R}^D$ and $t_i \in \{1, 2, \dots, K\}$.

We want to model it using a function $\pi : \mathbb{R}^D \rightarrow \mathbb{R}^K$ so that $\pi(\mathbf{x})$ gives a distribution of classes for input \mathbf{x} .

We impose the following conditions on π :

- $\forall 1 \leq k \leq K : \pi(\mathbf{x})_k \geq 0,$
- $\sum_{k=1}^K \pi(\mathbf{x})_k = 1,$
- $\forall 1 \leq k \leq K : \sum_{i=1}^N \pi(\mathbf{x}_i)_k x_i = \sum_{i=1}^N [t_i == k] x_i.$

Answer:

There are many such π , one particularly bad is

$$\pi(\mathbf{x}) = \begin{cases} \mathbf{1}_{t_i} & \text{if there exists } i : \mathbf{x}_i = \mathbf{x}, \\ \mathbf{1}_0 & \text{otherwise,} \end{cases}$$

where $\mathbf{1}_i$ is a one-hot encoding of i (vector of zeros, except for position i , which is equal to 1).

Therefore, we want to find a more **general** π – consequently, we turn to the principle of maximum entropy and search for π with maximum entropy.

We want to minimize $-\sum_{i=1}^N H(\pi(\mathbf{x}_i))$ given

- $\forall 1 \leq i \leq N, \forall 1 \leq k \leq K : \pi(\mathbf{x}_i)_k \geq 0,$
- $\forall 1 \leq i \leq N : \sum_{k=1}^K \pi(\mathbf{x}_i)_k = 1,$
- $\forall 1 \leq j \leq D, \forall 1 \leq k \leq K : \sum_{i=1}^N \pi(\mathbf{x}_i)_k x_{i,j} = \sum_{i=1}^N [t_i == k] x_{i,j}.$

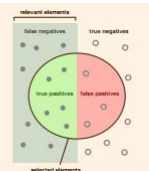
We therefore form a Lagrangian (ignoring the first inequality constraint):

$$\begin{aligned} \mathcal{L} = & \sum_{i=1}^N \sum_{k=1}^K \pi(\mathbf{x}_i)_k \log(\pi(\mathbf{x}_i)_k) \\ & - \sum_{j=1}^D \sum_{k=1}^K \lambda_{j,k} \left(\sum_{i=1}^N \pi(\mathbf{x}_i)_k x_{i,j} - [t_i == k] x_{i,j} \right) \\ & - \sum_{i=1}^N \beta_i \left(\sum_{k=1}^K \pi(\mathbf{x}_i)_k - 1 \right). \end{aligned}$$

5.4 Q

Define precision (including true positives and others), recall, F_1 score and F_β score (we stated several formulations for F_1 and F_β scores; any of them will do). [5]

| | Target positive | Target negative |
|--------------------|---------------------|---------------------|
| Predicted positive | True Positive (TP) | False Positive (FP) |
| Predicted negative | False Negative (FN) | True Negative (TN) |

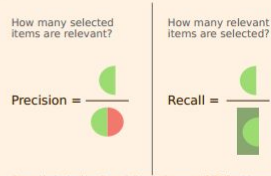


Answer:

In some cases, we are mostly interested in positive examples.

We define **precision** (percentage of correct predictions in predicted examples) and **recall** (percentage of correct predictions in the gold examples) as

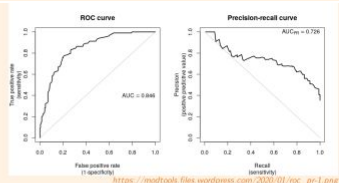
$$\begin{aligned} \text{precision} &= \frac{TP}{TP + FP}, \\ \text{recall} &= \frac{TP}{TP + FN}. \end{aligned}$$



The precision and recall go “against each other”: increasing the classifier threshold usually increases recall and decreases precision, and vice versa.

We therefore define a single **F_1 score** as a harmonic mean of precision and recall:

$$\begin{aligned} F_1 &= \frac{2}{\text{precision}^{-1} + \text{recall}^{-1}} \\ &= \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \\ &= \frac{TP}{TP + FP + TP + FN}. \end{aligned}$$



The F_1 score can be generalized to F_β score, which can be used as a metric when recall is β times more important than precision; F_2 favoring recall and $F_{0.5}$ favoring precision are commonly used.

The formula for F_β is

$$\begin{aligned}
 F_\beta &= \frac{1 + \beta^2}{\text{precision}^{-1} + \beta^2 \text{recall}^{-1}} \\
 &= \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \\
 &= \frac{TP}{TP + FP + \beta^2(TP + FN)}.
 \end{aligned}$$

5.5 Q

Explain the difference between micro-averaged and macro-averaged F_1 score. [5]

To extend F_1 -score to multiclass classification, we expect one of the classes to be *negative* and the others *different kinds of positive*. For each of the positive classes, we compute the same confusion matrix as in the binary case (considering all other labels as negative ones), and then combine the results in one of the following ways:

- **micro-averaged F_1** (or just **micro F_1**): we first sum all the TP, FP and FN of the individual binary classifications and compute the final F_1 -score (this way, the frequency of the individual classes is taken into account);
- **macro-averaged F_1** (or just **macro F_1**): we first compute the F_1 -scores of the individual binary classifications and then compute an unweighted average (therefore, the frequency of the classes is ignored).

Answer:

5.6 Q

Describe k-nearest neighbors prediction, both for regression and classification. Define L_p norm and describe uniform, inverse and softmax weighting. [5]

Regression

To perform regression when k nearest neighbors have values t_i and weights w_i , we predict

$$t = \sum_i \frac{w_i}{\sum_j w_j} \cdot t_i.$$

Classification

For uniform weights, we can use **voting** during prediction – the most frequent class is predicted (with ties broken arbitrarily).

Otherwise, we weight the categorical distributions $t_i \in \mathbb{R}^K$ (with the training target classes represented using one-hot encoding), predicting a distribution

$$t = \sum_i \frac{w_i}{\sum_j w_j} \cdot t_i.$$

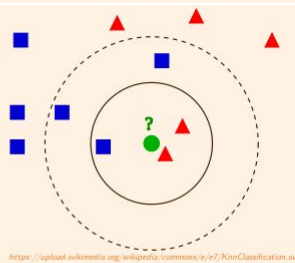
The predicted class is then the one with largest probability, i.e., $\arg \max_k \sum_i w_i t_{i,k}$.

Answer:

Several hyperparameters influence the behaviour of the prediction phase:

- **k**: consider k most similar training examples (higher k usually decrease variance, but increase bias);
- **metric**: a function used to find the nearest neighbors; common choices are metrics based on L_p norms (with usual values of p being 1, 2, 3, ∞). For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$, the distance is measured as $\|\mathbf{x} - \mathbf{y}\|_p$, where

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p};$$



- **weights**: optionally, more similar examples can be considered with bigger weights:
 - **uniform**: all k nearest neighbors are considered equally;
 - **inverse**: the weight of an example is proportional to the inverse of distance;
 - **softmax**: the weights are proportional to softmax of negative distances.

6 Lecture

6.1 Q

Define a kernel based on a feature map $\varphi: \mathbb{R}^D \rightarrow \mathbb{R}^F$, and write down the formulas for

1. a polynomial kernel of degree d ,
2. a polynomial kernel of degree at most d ,
3. an RBF kernel. [5]

We define a **kernel** corresponding to a feature map φ as a function

$$K(\mathbf{x}, \mathbf{z}) \stackrel{\text{def}}{=} \varphi(\mathbf{x})^T \varphi(\mathbf{z}).$$

Answer:

- **Polynomial kernel of degree d** , also called *homogenous polynomial kernel*

$$K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z})^d,$$

corresponds to a feature map returning all combinations of exactly d input features.

- **Polynomial kernel of degree at most d** , also called *nonhomogenous polynomial kernel*

$$K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z} + 1)^d,$$

corresponds to a feature map generating all combinations of up to d input features.

- **Gaussian Radial basis function (RBF) kernel**

$$K(\mathbf{x}, \mathbf{z}) = e^{-\gamma \|\mathbf{x} - \mathbf{z}\|^2},$$

corresponds to a scalar product in an infinite-dimensional space; it is a combination of polynomial kernels of all degrees. Assuming $\gamma = 1$ for simplicity, we get

6.2 Q

Define a kernel and write down the mini-batch SGD training algorithm of dual formulation of kernel linear regression. Then describe how predictions for unseen data are made. [10]

We can formulate an alternative linear regression algorithm (a so-called **dual formulation**):

Input: Dataset $(\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \mathbb{R}^N)$, learning rate $\alpha \in \mathbb{R}^+$.

- set $\beta_i \leftarrow 0$
- compute all values $\mathbf{K}_{i,j} = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$
- until convergence (or patience runs out), process a minibatch of examples with indices \mathbf{b} :
 - simultaneously for all $i \in \mathbf{b}$ (the β_j on the right side must not be modified during the batch update):
 - $\beta_i \leftarrow \beta_i - \frac{\alpha}{|\mathbf{b}|} \left(\sum_j \beta_j \mathbf{K}_{i,j} - t_i \right)$

The predictions are then performed by computing

$$y(\mathbf{z}) = \varphi(\mathbf{z})^T \mathbf{w} = \sum_i \beta_i \varphi(\mathbf{z})^T \varphi(\mathbf{x}_i).$$

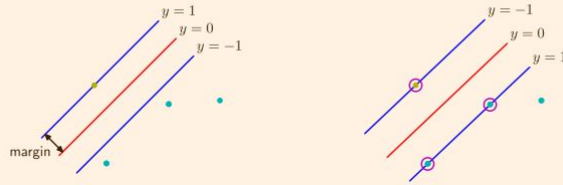
Answer:

6.3 Q

Derive the primary formulation of hard-margin SVM (the value to minimize, the constraints to fulfill) as a maximum-margin classifier. [5]

Let us return to a binary classification task. The perceptron algorithm guaranteed finding some separating hyperplane if it existed (but it could find quite a bad one).

We now consider finding the one with **maximum margin**.



Answer:

Assume we have a dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{t} \in \{-1, 1\}^N$, a feature map φ and a model

$$y(\mathbf{x}) \stackrel{\text{def}}{=} \varphi(\mathbf{x})^T \mathbf{w} + b.$$

We already know that the distance of a point \mathbf{x}_i to the decision boundary is

$$\frac{|y(\mathbf{x}_i)|}{\|\mathbf{w}\|} \quad \text{assuming } y \text{ classifies all } \mathbf{x}_i \text{ correctly} \quad \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|}.$$

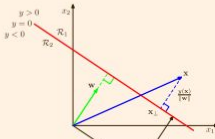


Figure 4.1 of Pattern Recognition and Machine Learning

We therefore want to maximize

$$\arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\varphi(\mathbf{x}_i)^T \mathbf{w} + b)].$$

However, this problem is difficult to optimize directly.

Because the model is invariant to multiplying \mathbf{w} and b by a constant, we can decide that for the points closest to the decision boundary, it will hold that

$$t_i y(\mathbf{x}_i) = 1.$$

Then for all the points we will have $t_i y(\mathbf{x}_i) \geq 1$ and we can simplify

$$\arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\varphi(\mathbf{x}_i)^T \mathbf{w} + b)].$$

to

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{given that } t_i y(\mathbf{x}_i) \geq 1.$$

6.4 Q

How do we search for a minimum of a function $f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}$ subject to an inequality constraint $g(\mathbf{x}) \geq 0$? Formulate both the variant with KKT conditions and the variant with the λ maximization, and prove that they are equivalent. [10]

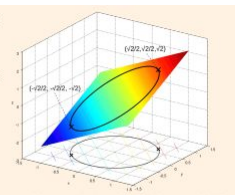
Given a function $f(\mathbf{x})$, we can find its minimum with respect to a vector $\mathbf{x} \in \mathbb{R}^d$, by investigating the critical points $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$.

We can even incorporate constraints of form $g(\mathbf{x}) = 0$ by forming a Lagrangian

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

and again investigating the critical points $\nabla_{\mathbf{x}, \lambda} \mathcal{L}(\mathbf{x}, \lambda) = 0$.

We now describe how to include inequality constraints $g(\mathbf{x}) \geq 0$.



https://upload.wikimedia.org/wikipedia/commons/0/0d/Lag

Answer: The λ maximization:

Our goal is to find a minimum of $f(\mathbf{x})$ subject to a constraint $g(\mathbf{x}) \geq 0$.

We start by again forming a Lagrangian $f(\mathbf{x}) - \lambda g(\mathbf{x})$.

The optimum can either be attained for $g(\mathbf{x}) > 0$, when the constraint is said to be **inactive**, or for $g(\mathbf{x}) = 0$, when the constraint is said to be **active**. In the inactive case, the minimum is again a critical point of the Lagrangian with the condition $\lambda = 0$.

When the minimum is on a boundary, it corresponds to a critical point with $\lambda \neq 0$ – but note that this time the sign of the multiplier matters, because the minimum is attained only when the gradient of $f(\mathbf{x})$ is oriented **into** the region $g(\mathbf{x}) \geq 0$. We therefore require $\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$ for $\lambda > 0$.

In both cases, $\lambda g(\mathbf{x}) = 0$.

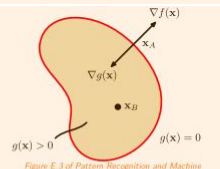
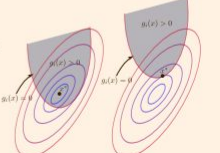


Figure E.3 of Pattern Recognition and Machine Learning



https://upload.wikimedia.org/wikipedia/commons/5/5d/let

The KKT conditions variant:

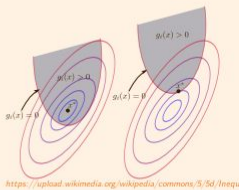
Let $f(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$ be a function, which has a minimum in \mathbf{x} subject to an inequality constraint $g(\mathbf{x}) \geq 0$. Assume that both f and g have continuous partial derivatives and that $\frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}) \neq 0$.

Then there exists a $\lambda \in \mathbb{R}$, such that the **Lagrangian function**

$$\mathcal{L}(\mathbf{x}, \lambda) \stackrel{\text{def}}{=} f(\mathbf{x}) - \lambda g(\mathbf{x})$$

has zero gradient in \mathbf{x} and the following conditions hold:

$$\begin{aligned} g(\mathbf{x}) &\geq 0, \\ \lambda &\geq 0, \\ \lambda g(\mathbf{x}) &= 0. \end{aligned}$$



These conditions are known as **Karush-Kuhn-Tucker (KKT)** conditions.

It is easy to verify that if we have the minimum \mathbf{x} and λ fulfilling the KKT conditions $g(\mathbf{x}) \geq 0, \lambda \geq 0, \lambda g(\mathbf{x}) = 0$, the Lagrangian \mathcal{L} has a **maximum** in λ :

- if $g(\mathbf{x}) = 0$, then \mathcal{L} does not change when changing λ ,
- if $g(\mathbf{x}) > 0$, then $\lambda = 0$ from the KKT conditions, which is a maximum of \mathcal{L} .

On the other hand, if we have the minimum \mathbf{x} , $\lambda \geq 0$ and \mathcal{L} has a maximum in λ , all the KKT conditions must hold:

- if $g(\mathbf{x}) < 0$, then increasing λ would increase \mathcal{L} ,
- if $g(\mathbf{x}) > 0$, then decreasing λ increase \mathcal{L} , so $\lambda = 0$.

Maximizing Given $f(\mathbf{x})$

If we instead want to find the constrained maximum of $f(\mathbf{x})$, we can search for the minimum of $-f(\mathbf{x})$, which results in the Lagrangian $f(\mathbf{x}) + \lambda g(\mathbf{x})$, which we *minimize* with respect to λ .

6.5 Q

Starting from primary hard-margin SVM formulation, derive the dual formulation (the Lagrangian L , the required conditions, the KKT conditions). [10]

Answer: Hard margin model formulation:

We already know that the distance of a point \mathbf{x}_i to the decision boundary is

$$\frac{|y(\mathbf{x}_i)|}{\|\mathbf{w}\|} \stackrel{\text{assuming } y \text{ classifies all } \mathbf{x}_i \text{ correctly}}{=} \frac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|}.$$

We therefore want to maximize

$$\arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\varphi(\mathbf{x}_i)^T \mathbf{w} + b)].$$

Because the model is invariant to multiplying \mathbf{w} and b by a constant, we can decide that for the points closest to the decision boundary, it will hold that

$$t_i y(\mathbf{x}_i) = 1.$$

Then for all the points we will have $t_i y(\mathbf{x}_i) \geq 1$ and we can simplify

$$\arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_i [t_i (\varphi(\mathbf{x}_i)^T \mathbf{w} + b)].$$

to

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1.$$

The Lagrangian, conditions, KKT conditions:

In order to solve the constrained problem of

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1,$$

we write the Lagrangian with multipliers $\mathbf{a} = (a_1, \dots, a_N)$ as

$$\mathcal{L} = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i a_i [t_i y(\mathbf{x}_i) - 1].$$

Setting the derivatives with respect to \mathbf{w} and b to zero, we get

$$\begin{aligned} \mathbf{w} &= \sum_i a_i t_i \varphi(\mathbf{x}_i), \\ 0 &= \sum_i a_i t_i. \end{aligned}$$

Substituting these to the **Lagrangian**, we want to maximize

$$\mathcal{L} = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j)$$

with respect to a_i subject to the constraints $a_i \geq 0$ and $\sum_i a_i t_i = 0$, using the kernel $K(\mathbf{x}, \mathbf{z}) = \varphi(\mathbf{x})^T \varphi(\mathbf{z})$.

The solution will fulfil the **KKT conditions**, meaning that

$$a_i \geq 0, \quad t_i y(\mathbf{x}_i) - 1 \geq 0, \quad a_i (t_i y(\mathbf{x}_i) - 1) = 0.$$

Therefore, either a point \mathbf{x}_i is on a boundary, or $a_i = 0$. Given that the prediction for \mathbf{x} is $y(\mathbf{x}) = \sum_i a_i t_i K(\mathbf{x}, \mathbf{x}_i) + b$, we only need to keep the training points \mathbf{x}_i that are on the boundary, the so-called **support vectors**. Therefore, even though SVM is a nonparametric model, it needs to store only a subset of the training data.

6.6 Q

Considering hard-margin SVM, define what a support vector is, and how predictions are performed for unseen data. [5]

The solution will fulfil the **KKT conditions**, meaning that

$$a_i \geq 0, \quad t_i y(\mathbf{x}_i) - 1 \geq 0, \quad a_i (t_i y(\mathbf{x}_i) - 1) = 0.$$

Therefore, either a point \mathbf{x}_i is on a boundary, or $a_i = 0$. Given that the prediction for \mathbf{x} is $y(\mathbf{x}) = \sum_i a_i t_i K(\mathbf{x}, \mathbf{x}_i) + b$, we only need to keep the training points \mathbf{x}_i that are on the boundary, the so-called **support vectors**. Therefore, even though SVM is a nonparametric model, it needs to store only a subset of the training data.

Answer:

7 Lecture

7.1 Q

Write down the primary formulation of soft-margin SVM using the slack variables (the value to minimize, the constraints to fulfil). [5]

Until now, we assumed the data to be linearly separable – the **hard-margin SVM** variant. We now relax this condition to arrive at **soft-margin SVM**. The idea is to allow points to be in the margin or even on the *wrong side* of the decision boundary. We introduce **slack variables** $\xi_i \geq 0$, one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\mathbf{x}_i) \geq 1, \\ |t_i - y(\mathbf{x}_i)| & \text{otherwise.} \end{cases}$$

Therefore, $\xi_i = 0$ signifies a point outside of margin, $0 < \xi_i < 1$ denotes a point inside the margin, $\xi_i = 1$ is a point on the decision boundary, and $\xi_i > 1$ indicates the point is on the opposite side of the separating hyperplane.

Therefore, we want to optimize

$$\arg \min_{\mathbf{w}, b} C \sum_i \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

Answer:

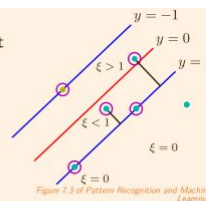
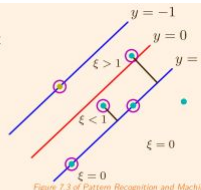


Figure 7.3 of Pattern Recognition and Machine Learning.

7.2 Q

Starting from primary soft-margin SVM formulation, derive the dual formulation (the Lagrangian L , the required conditions, the KKT conditions). [10]

Until now, we assumed the data to be linearly separable – the **hard-margin SVM** variant. We now relax this condition to arrive at **soft-margin SVM**. The idea is to allow points to be in the margin or even on the *wrong side* of the decision boundary. We introduce **slack variables** $\xi_i \geq 0$, one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\mathbf{x}_i) \geq 1, \\ |t_i - y(\mathbf{x}_i)| & \text{otherwise.} \end{cases}$$


Therefore, $\xi_i = 0$ signifies a point outside of margin, $0 < \xi_i < 1$ denotes a point inside the margin, $\xi_i = 1$ is a point on the decision boundary, and $\xi_i > 1$ indicates the point is on the opposite side of the separating hyperplane.

Therefore, we want to optimize

$$\arg \min_{\mathbf{w}, b} C \sum_i \xi_i + \frac{1}{2} \|\mathbf{w}\|^2 \text{ given that } t_i y(\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

Answer:

To solve the soft-margin variant, we again create a Lagrangian, this time with two sets of multipliers $\mathbf{a} = (a_1, \dots, a_N)$ and $\boldsymbol{\mu} = (\mu_1, \dots, \mu_N)$:

$$\mathcal{L} = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i - \sum_i a_i [t_i y(\mathbf{x}_i) - 1 + \xi_i] - \sum_i \mu_i \xi_i.$$

Solving for the critical points and substituting for \mathbf{w} , b and $\boldsymbol{\xi}$ (obtaining an additional constraint $\mu_i = C - a_i$ compared to the previous case), we obtain the Lagrangian in the form

$$\mathcal{L} = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j),$$

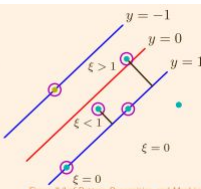
which is identical to the previous case, but the constraints are a bit different:

$$\forall i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$

7.3 Q

Write down the primary formulation of soft-margin SVM using the hinge loss. [5]

Until now, we assumed the data to be linearly separable – the **hard-margin SVM** variant. We now relax this condition to arrive at **soft-margin SVM**. The idea is to allow points to be in the margin or even on the *wrong side* of the decision boundary. We introduce **slack variables** $\xi_i \geq 0$, one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\mathbf{x}_i) \geq 1, \\ |t_i - y(\mathbf{x}_i)| & \text{otherwise.} \end{cases}$$


Answer:

Note that the slack variables can be written as

$$\xi_i = \max(0, 1 - t_i y(\mathbf{x}_i)),$$

so we can reformulate the soft-margin SVM objective using the **hinge loss**

$$\mathcal{L}_{\text{hinge}}(t, y) \stackrel{\text{def}}{=} \max(0, 1 - ty)$$

to

$$\arg \min_{\mathbf{w}, b} C \sum_i \mathcal{L}_{\text{hinge}}(t_i, y(\mathbf{x}_i)) + \frac{1}{2} \|\mathbf{w}\|^2.$$

Such formulation is analogous to a regularized loss, where C is an *inverse* regularization strength, so $C = \infty$ implies no regularization, and $C = 0$ ignores the data entirely.

7.4 Q

Describe the high-level overview of the SMO algorithm (the test whether the KKT conditions hold, how do we select the a_i and a_j to update, what is the goal of updating the a_i and a_j , how

do we detect convergence; but without the update of a_i, a_j, b themselves). [5]

At its core, the SMO algorithm is just a coordinate descent. It tries to find a_i maximizing \mathcal{L} while fulfilling the KKT conditions – once found, an optimum has been reached, given that for soft-margin SVM the KKT conditions are sufficient conditions for optimality (for soft-margin SVM, the loss is convex and the inequality constraints are not only convex, but even affine). However, note that because of the $\sum a_i t_i = 0$ constraint, we cannot optimize just one a_i , because a single a_i is determined from the others. Therefore, in each step, we pick two a_i, a_j coefficients and try to maximize the loss while fulfilling the constraints.

- loop until convergence (until $\forall i : a_i < C \Rightarrow t_i y(\mathbf{x}_i) \geq 1$ and $a_i > 0 \Rightarrow t_i y(\mathbf{x}_i) \leq 1$)
 - for i in $\{1, 2, \dots, N\}$:
 - choose $j \neq i$ in $\{1, 2, \dots, N\}$
 - $a_i, a_j \leftarrow \arg \max_{a_i, a_j} \mathcal{L}(a_1, a_2, \dots, a_N)$, while respecting the constraints:
 - $0 \leq a_i \leq C, 0 \leq a_j \leq C, \sum_i a_i t_i = 0$

Answer:

Maximizing $\mathcal{L}(\mathbf{a})$ with respect to a_i and a_j then amounts to maximizing a quadratic function of a_j , which has an analytical solution.

7.5 Q

Describe the part of the SMO algorithm which updates a_i and a_j to maximize the Lagrangian. If you explain how is the update derived (so that if I followed the instructions, I would come up with the update rules), you do not need to write explicit formulas. [10]

At its core, the SMO algorithm is just a coordinate descent. It tries to find a_i maximizing \mathcal{L} while fulfilling the KKT conditions – once found, an optimum has been reached, given that for soft-margin SVM the KKT conditions are sufficient conditions for optimality (for soft-margin SVM, the loss is convex and the inequality constraints are not only convex, but even affine). However, note that because of the $\sum a_i t_i = 0$ constraint, we cannot optimize just one a_i , because a single a_i is determined from the others. Therefore, in each step, we pick two a_i, a_j coefficients and try to maximize the loss while fulfilling the constraints.

- loop until convergence (until $\forall i : a_i < C \Rightarrow t_i y(\mathbf{x}_i) \geq 1$ and $a_i > 0 \Rightarrow t_i y(\mathbf{x}_i) \leq 1$)
 - for i in $\{1, 2, \dots, N\}$:
 - choose $j \neq i$ in $\{1, 2, \dots, N\}$
 - $a_i, a_j \leftarrow \arg \max_{a_i, a_j} \mathcal{L}(a_1, a_2, \dots, a_N)$, while respecting the constraints:
 - $0 \leq a_i \leq C, 0 \leq a_j \leq C, \sum_i a_i t_i = 0$

Answer:

In soft-margin SVM, we try to maximize

$$\mathcal{L} = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Maximizing $\mathcal{L}(\mathbf{a})$ with respect to a_i and a_j then amounts to maximizing a quadratic function of a_j , which has an analytical solution.

There we have to have a negative second derivative (we want the maximum):

We already know that $a_i = t_i(\zeta - a_j t_j)$.

To find a_j maximizing \mathcal{L} , we use the formula for locating a vertex of a parabola

$$a_j^{\text{new}} \leftarrow a_j - \frac{\partial \mathcal{L} / \partial a_j}{\partial^2 \mathcal{L} / \partial a_j^2},$$

which is in fact one Newton-Raphson iteration step.

Denoting $E_j \stackrel{\text{def}}{=} y(\mathbf{x}_j) - t_j$, we can compute the first derivative as

$$\frac{\partial \mathcal{L}}{\partial a_j} = t_j(E_i - E_j),$$

and the second derivative as

$$\frac{\partial^2 \mathcal{L}}{\partial a_j^2} = 2K(\mathbf{x}_i, \mathbf{x}_j) - K(\mathbf{x}_i, \mathbf{x}_i) - K(\mathbf{x}_j, \mathbf{x}_j).$$

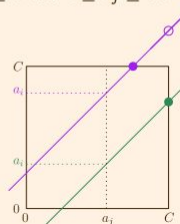
If the second derivative is negative, we know that the vertex is really a maximum, in which case we get

$$a_j^{\text{new}} \leftarrow a_j - t_j \frac{E_i - E_j}{2K(\mathbf{x}_i, \mathbf{x}_j) - K(\mathbf{x}_i, \mathbf{x}_i) - K(\mathbf{x}_j, \mathbf{x}_j)}.$$

However, our maximization is constrained – it must hold that $0 \leq a_i \leq C$ and $0 \leq a_j \leq C$.

Recalling that $a_i = -t_i t_j a_j + \text{const}$, we can plot the dependence of a_i and a_j . If for example $-t_i t_j = 1$ and $a_j^{\text{new}} > C$, we need to find the “right-most” solution fulfilling both $a_i \leq C$ and $a_j \leq C$. Such a solution is either:

- when a_j^{new} is clipped to C , as in the green case in the example,
- when a_j^{new} is clipped so that $a_i^{\text{new}} = C$ (the purple case in the example), in which case $a_j^{\text{new}} = a_j + (C - a_i)$.



7.6 Q

Describe the part of the SMO algorithm which updates b to maximize the Lagrangian. If you explain how is the update derived (so that if I followed the instructions, I would come up with two b candidates and a rule how to utilize them), you do not need to write explicit formulas. [10]

Answer: After updating a_i, a_j (if we updated them at all), we proceed to update the bias (that is the same for all values a and needs to be updated so that the conditions still hold with the new a updates):

To arrive at the bias update, we consider the KKT condition that for $0 < a_j^{\text{new}} < C$, it must hold that $1 = t_j y_j(\mathbf{x}_j) = t_j [(\sum_l a_l^{\text{new}} t_l K(\mathbf{x}_j, \mathbf{x}_l)) + b^{\text{new}}]$. Combining it with the fact that $(\sum_l a_l t_l K(\mathbf{x}_j, \mathbf{x}_l)) + b = E_j + t_j$, we obtain

$$b_j^{\text{new}} = b - E_j - t_i(a_i^{\text{new}} - a_i)K(\mathbf{x}_i, \mathbf{x}_j) - t_j(a_j^{\text{new}} - a_j)K(\mathbf{x}_j, \mathbf{x}_j).$$

Analogously for $0 < a_i^{\text{new}} < C$ we get

$$b_i^{\text{new}} = b - E_i - t_i(a_i^{\text{new}} - a_i)K(\mathbf{x}_i, \mathbf{x}_i) - t_j(a_j^{\text{new}} - a_j)K(\mathbf{x}_j, \mathbf{x}_i).$$

Finally, if $a_j^{\text{new}}, a_i^{\text{new}} \in \{0, C\}$, we know that all values between b_i and b_j fulfil the KKT conditions. We therefore arrive at the following update for bias:

$$b^{\text{new}} = \begin{cases} b_i^{\text{new}} & \text{if } 0 < a_i^{\text{new}} < C, \\ b_j^{\text{new}} & \text{if } 0 < a_j^{\text{new}} < C, \\ (b_i^{\text{new}} + b_j^{\text{new}})/2 & \text{otherwise.} \end{cases}$$

7.7 Q

Describe the one-versus-one and one-versus-rest schemes of constructing a K -class classifier by combining multiple binary classifiers. [5]

There are two general approaches for building a K -class classifier by combining several binary classifiers:

- **one-versus-rest** scheme: K binary classifiers are constructed, the i -th separating instances of class i from all others; during prediction, the one with highest probability is chosen
 - the binary classifiers need to return calibrated probabilities (not SVM)
- **one-versus-one** scheme: $\binom{K}{2}$ binary classifiers are constructed, one for each (i, j) pair of class indices; during prediction, the class with the majority of votes wins (used by SVM)

However, voting suffers from serious difficulties, because there usually exist regions which are ambiguous.

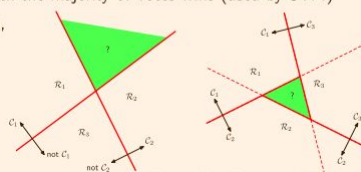


Figure 4.2 of Pattern Recognition and Machine Learning

Answer:

8 Lecture

8.1 Q

Explain how is the TF-IDF weight of a given document-term computed. [5]

We already know how to represent images and categorical variables (classes, letters, words, ...).
 Now consider a problem of representing a whole *document*.
 We usually represent a document as a **bag of words** – we create a feature space with a dimension for every unique word (or for character sequences), called **term**.
 However, there are many possible ways how the values of the terms might be set.

Answer:

Commonly used ways of setting the term values:

- **binary indicators**: 1/0 depending on whether a term is present in a document or not;
- **term frequency (TF)**: relative frequency of a term in a document;

$$TF(t; d) = \frac{\text{number of occurrences of } t \text{ in the document } d}{\text{number of terms in the document } d}$$

- **inverse document frequency (IDF)**: we could also represent a term using self-information of a probability of a random document containing it (therefore, terms with lower document probability have higher weights);

$$IDF(t) = \log \frac{\text{number of documents}}{\text{number of documents containing } t \text{ (optionally } + 1)} = I(P(d \ni t))$$

- **TF-IDF**: empirically, product **TF · IDF** is a feature reflecting quite well how important is a term to a document in a corpus (used by 83% text-based recommender systems in 2015).

8.2 Q

Define conditional entropy, mutual information, write down relation between them, and finally prove that mutual information is zero if and only if the two random variables are independent (you do not need to prove statements about D_{KL}). [5]

Consider two random variables x and y with distributions $x \sim X$ and $y \sim Y$.
 The **conditional entropy** $H(Y|X)$ can be naturally considered an expectation of a self-information of $Y|X$, so in the discrete case,

$$H(Y|X) = \mathbb{E}_{x,y} [I(y|x)] = - \sum_{x,y} P(x,y) \log P(y|x).$$

In order to assess the amount of information *shared* between the two random variables, we might consider the difference

$$H(Y) - H(Y|X) = \mathbb{E}_{x,y} [-\log P(y)] - \mathbb{E}_{x,y} [-\log P(y|x)] = \mathbb{E}_{x,y} \left[\log \frac{P(x,y)}{P(x)P(y)} \right].$$

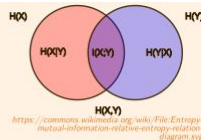
We can interpret this value as

How many bits of information will we learn about Y when we find out X ?

Answer:

Let us denote this quantity as the **mutual information** $I(X; Y)$:

$$I(X; Y) = \mathbb{E}_{x,y} \left[\log \frac{P(x,y)}{P(x)P(y)} \right].$$



- The mutual information is symmetrical, so

$$I(X; Y) = I(Y; X) = H(Y) - H(Y|X) = H(X) - H(X|Y).$$

- It is easy to verify that

$$I(X; Y) = D_{KL}(P(X, Y) \| P(X)P(Y)).$$

Therefore,

- $I(X; Y) \geq 0$,
- $I(X; Y) = 0$ iff $P(X, Y) = P(X)P(Y)$ iff the random variables are independent.

8.3 Q

Show that TF-IDF terms can be considered portions of a suitable mutual information. [5]

Let \mathcal{D} be a collection of N documents and \mathcal{T} collections of terms.

Our assumption is that whenever we need to draw a document, we do it uniformly randomly. Therefore,

- $P(d) = 1/|\mathcal{D}|$ and $I(d) = H(\mathcal{D}) = \log |\mathcal{D}|$,
- $P(d|t) = 1/|\{d \in \mathcal{D} : t \in d\}|$ and $I(d|t) = H(\mathcal{D}|\mathcal{T} = t) = \log |\{d \in \mathcal{D} : t \in d\}|$,
- $I(d) - I(d|t) = H(\mathcal{D}) - H(\mathcal{D}|\mathcal{T} = t) = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in d\}|} = IDF(t)$.

Finally, we can compute the mutual information $I(\mathcal{D}; \mathcal{T})$ as

$$I(\mathcal{D}; \mathcal{T}) = \sum_{d,t} P(d) \cdot P(t|d) \cdot (I(d) - I(d|t)) = \frac{1}{|\mathcal{D}|} \sum_{d,t} TF(t; d) \cdot IDF(t).$$

Therefore, summing all TF-IDF terms recovers the mutual information between \mathcal{D} and \mathcal{T} , and we can say that each TF-IDF carries a "bit of information" attached to a document-term pair.

Answer:

8.4 Q

Show that L_2 -regularization can be obtained from a suitable prior by Bayesian inference (from the MAP estimate). [5]

Frequently, the mean is assumed to be zero, and the variance is assumed to be σ^2 . Given that we have no further information, we employ the maximum entropy principle, which provides us with $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, \sigma^2)$. Then

$$\begin{aligned} \mathbf{w}_{\text{MAP}} &= \arg \max_{\mathbf{w}} p(\mathbf{X}|\mathbf{w})p(\mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \prod_{i=1}^N p(\mathbf{x}_i|\mathbf{w})p(\mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^N (-\log p(\mathbf{x}_i|\mathbf{w}) - \log p(\mathbf{w})). \end{aligned}$$

By substituting the probability of the Gaussian prior, we get

$$\mathbf{w}_{\text{MAP}} = \arg \min_{\mathbf{w}} \sum_{i=1}^N -\log p(\mathbf{x}_i|\mathbf{w}) - \frac{1}{2} \log(2\pi\sigma^2) + \frac{\|\mathbf{w}\|^2}{2\sigma^2},$$

which is in fact the L_2 -regularization.

Answer:

Pozn.: normal distribution

Normal (or Gaussian) Distribution

Distribution over real numbers, parametrized by a mean μ and variance σ^2 :

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

For standard values $\mu = 0$ and $\sigma^2 = 1$ we get $\mathcal{N}(x; 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

8.5 Q

Write down how is $p(C_k|\mathbf{x})$ approximated in a Naive Bayes classifier, explicitly state the Naive Bayes assumption, and show how prediction is performed. [5]

Answer: Context:

So far, our classifiers were so-called **discriminative** and had a form

$$p(C_k|\mathbf{x}) = p(C_k|x_1, x_2, \dots, x_D).$$

Instead, we might use the Bayes' theorem and rewrite to

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})}.$$

Then, classification could be performed as

$$\arg \max_k p(C_k|\mathbf{x}) = \arg \max_k \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} = \arg \max_k p(\mathbf{x}|C_k)p(C_k).$$

when maximizing with respect to k , $p(\mathbf{x})$ is independent of k and acts only as a constant

Therefore, instead of modeling $p(C_k|\mathbf{x})$, we model

- the prior $p(C_k)$ according to the distribution of classes in the data, and
- the distribution $p(\mathbf{x}|C_k)$.

Our $p(C_k|\mathbf{x})$ approximated by Naive Bayes and the assumption + prediction:

Modeling the distribution $p(\mathbf{x}|C_k)$ is however difficult – \mathbf{x} can be high-dimensional high-structured data.

Therefore, the so-called **Naive Bayes classifier** assumes that

all x_d are independent given C_k ,

so we can rewrite

$$p(\mathbf{x}|C_k) = p(x_1|C_k)p(x_2|C_k, x_1)p(x_3|C_k, x_1, x_2) \cdots p(x_D|C_k, x_1, x_2, \dots)$$

to

$$p(\mathbf{x}|C_k) = \prod_{d=1}^D p(x_d|C_k).$$

Notice that modeling $p(x_d|C_k)$ is substantially easier because it is a distribution over a single-dimensional quantity.

8.6 Q

Considering a Gaussian naive Bayes, describe how are $p(x_d|C_k)$ modeled (what distribution and which parameters does it have) and how we estimate it during fitting. [5]

Gaussian Naive Bayes

In Gaussian naive Bayes, we expect a continuous feature to have normal distribution for a given C_k , and model $p(x_d|C_k)$ is modeled as a normal distribution $\mathcal{N}(\mu_{d,k}, \sigma_{d,k}^2)$.

Assuming we have the training data \mathbf{X} together with K -class classification targets \mathbf{t} , the “training” phase consists of estimating the parameters $\mu_{d,k}$ and $\sigma_{d,k}^2$ of the distributions $\mathcal{N}(\mu_{d,k}, \sigma_{d,k}^2)$ for $1 \leq d \leq D$, $1 \leq k \leq K$, employing the maximum likelihood estimation.

Now let feature d and class k be fixed and let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N_k}$ be the training data corresponding to the class k . We already know that maximum likelihood estimation using N_k samples drawn from a Gaussian distribution $\mathcal{N}(\mu_{d,k}, \sigma_{d,k}^2)$ amounts to

$$\arg \min_{\mu_{d,k}, \sigma_{d,k}^2} \frac{N_k}{2} \log(2\pi\sigma_{d,k}^2) + \sum_{i=1}^{N_k} \frac{(x_{i,d} - \mu_{d,k})^2}{2\sigma_{d,k}^2}.$$

Answer:

Setting the derivative with respect to $\mu_{d,k}$ to zero results in

$$0 = \sum_{i=1}^{N_k} \frac{-2(x_{i,d} - \mu_{d,k})}{2\sigma_{d,k}^2},$$

which we can rewrite to $\mu_{d,k} = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{i,d}$.

Similarly, zeroing out the derivative with respect to $\sigma_{d,k}^2$ gives

$$0 = \frac{N_k}{2\sigma_{d,k}^2} - \frac{1}{2(\sigma_{d,k}^2)^2} \sum_{i=1}^{N_k} (x_{i,d} - \mu_{d,k})^2,$$

from which we obtain $\sigma_{d,k}^2 = \frac{1}{N_k} \sum_{i=1}^{N_k} (x_{i,d} - \mu_{d,k})^2$.

However, the variances are usually smoothed (increased) by a given constant α to avoid too sharp distributions (in Scikit-learn, the default value of α is 10^{-9} times the largest variance of all features).

The choice among the Gaussian, Bernoulli and multinomial naive Bayes depends on the feature values.

- If we expect the individual feature values to be roughly normally distributed, Gaussian NB is an obvious choice.
- To use multinomial NB, the features should roughly follow the multinomial distribution – i.e., they must be non-negative, be interpretable as “counts” and “compete” with each other.
 - Note that the feature can be real-valued (the multinomial distribution can be extended to real-value observations using the Γ function).
- In order to use Bernoulli NB, the features *must* be binary. However, an important difference is that contrary to the multinomial NB, the **absence of features** is also modeled by the $(1 - p_{d,k})$ term; the multinomial NB uses $p_{d,k}^0 = 1$ in such case.

8.7 Q

Considering a Multinomial naive Bayes, describe how are $p(\mathbf{x}|C_k)$ modeled (what distribution and which parameters does it have) and how we estimate it during fitting. [5]

The last variant of naive Bayes we will describe is the **multinomial naive Bayes**, where $p(\mathbf{x}|C_k)$ is modeled to be multinomial distribution, $p(\mathbf{x}|C_k) \propto \prod_d p_{d,k}^{x_{i,d}}$ probability of each term to the power of its appearances

Similarly to the Bernoulli NB case, we can write the log likelihood as

$$\log p(C_k|\mathbf{x}) + c = \log p(C_k) + \sum_d x_d \log p_{d,k} = b_k + \mathbf{x}^T \mathbf{w}_k.$$

Answer:

As in the previous cases, we turn to the maximum likelihood estimation in order to find out the values of $p_{d,k}$. We start with the log likelihood

$$\sum_{i=1}^{N_k} \log \left(\prod_d p_{d,k}^{x_{i,d}} \right) = \sum_{i,d} x_{i,d} \log p_{d,k}.$$

To maximize this quantity with respect to a probability distribution $\sum_d p_{d,k} = 1$, we need to form a *Lagrangian*

$$\mathcal{L} = \sum_{i,d} x_{i,d} \log p_{d,k} + \lambda \left(1 - \sum_d p_{d,k} \right).$$

Setting the derivative with respect to $p_{d,k}$ to zero results in $0 = \sum_{i=1}^{N_k} \frac{x_{i,d}}{p_{d,k}} - \lambda$, so

$$p_{d,k} = \frac{1}{\lambda} \sum_{i=1}^{N_k} x_{i,d} = \frac{\sum_{i=1}^{N_k} x_{i,d}}{\sum_{i=1}^{N_k} \sum_d x_{i,d}}, \text{ where } \lambda \text{ is set to fulfill } \sum_d p_{d,k} = 1.$$

Denoting $n_{d,k}$ as the sum of features x_d for a class C_k , the probabilities $p_{d,k}$ could be therefore estimated as

$$\times \quad p_{d,k} = \frac{n_{d,k}}{\sum_j n_{j,k}}.$$

However, for the same reasons as in the Bernoulli NB case, we also use the **Laplace smoothing**, i.e., utilize a Dirichlet prior $\text{Dir}(\alpha + 1)$, and instead use

$$p_{d,k} = \frac{n_{d,k} + \alpha}{\sum_j (n_{j,k} + \alpha)} = \frac{n_{d,k} + \alpha}{(\sum_j n_{j,k}) + \alpha D}$$

with pseudo-count $\alpha > 0$.

The choice among the Gaussian, Bernoulli and multinomial naive Bayes depends on the feature values.

- If we expect the individual feature values to be roughly normally distributed, Gaussian NB is an obvious choice.
- To use multinomial NB, the features should roughly follow the multinomial distribution – i.e., they must be non-negative, be interpretable as “counts” and “compete” with each other.
 - Note that the feature can be real-valued (the multinomial distribution can be extended to real-value observations using the Γ function).
- In order to use Bernoulli NB, the features *must* be binary. However, an important difference is that contrary to the multinomial NB, the **absence of features** is also modeled by the $(1 - p_{d,k})$ term; the multinomial NB uses $p_{d,k}^0 = 1$ in such case.

8.8 Q

Considering a Bernoulli naive Bayes, describe how are $p(x_d|C_k)$ modeled (what distribution and which parameters does it have) and how we estimate it during fitting. [5]

When the input features are binary, the $p(x_d|C_k)$ might be modeled using a Bernoulli distribution

$$p(x_d|C_k) = p_{d,k}^{x_d} \cdot (1 - p_{d,k})^{(1-x_d)}.$$

We can therefore write

$$p(C_k|\mathbf{x}) \propto \left(\prod_{d=1}^D p_{d,k}^{x_d} \cdot (1 - p_{d,k})^{(1-x_d)} \right) p(C_k),$$

and by computing a logarithm we get

$$\log p(C_k|\mathbf{x}) + c = \log p(C_k) + \sum_d \left(x_d \log \frac{p_{d,k}}{1 - p_{d,k}} + \log(1 - p_{d,k}) \right) = b_k + \mathbf{x}^T \mathbf{w}_k,$$

where the constant c does not depend on C_k and is therefore not needed for prediction

Answer:

$$\arg \max_k \log p(C_k|\mathbf{x}) = \arg \max_k b_k + \mathbf{x}^T \mathbf{w}_k.$$

To estimate the probabilities $p_{d,k}$, we turn again to the maximum likelihood estimation. The log likelihood of N_k samples drawn from Bernoulli distribution with parameter $p_{d,k}$ is

$$\sum_{i=1}^{N_k} \log(p_{d,k}^{x_{i,d}} (1 - p_{d,k})^{1-x_{i,d}}) = \sum_{i=1}^{N_k} (x_{i,d} \log p_{d,k} + (1 - x_{i,d}) \log(1 - p_{d,k})).$$

Setting the derivative with respect to $p_{d,k}$ to zero, we obtain

$$0 = \sum_{i=1}^{N_k} \left(\frac{x_{i,d}}{p_{d,k}} - \frac{1 - x_{i,d}}{1 - p_{d,k}} \right) = \frac{1}{p_{d,k}(1 - p_{d,k})} \sum_{i=1}^{N_k} ((1 - p_{d,k})x_{i,d} - p_{d,k}(1 - x_{i,d})),$$

giving us $p_{d,k} = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{i,d}$.

We could therefore estimate the probabilities $p_{d,k}$ as

$$\times \quad p_{d,k} = \frac{\text{number of documents of class } k \text{ with nonzero feature } d}{\text{number of documents of class } k}.$$

However, if a feature d is always set to one (or zero) for a given class k , then $p_{d,k} = 1$ (or 0). That is impractical because the resulting classifier would give probability zero to inputs with the opposite value of such feature.

Therefore, **Laplace or additive smoothing** is used, and the probability $p_{d,k}$ estimated as

$$p_{d,k} = \frac{\text{number of documents of class } k \text{ with nonzero feature } d + \alpha}{\text{number of documents of class } k + 2\alpha}$$

for some pseudo-count $\alpha > 0$.

here we have it twice (the documents where there is the term and where there isn't)

Note that even if this technique has a special name, it corresponds to using a **maximum a posteriori** estimate, using $\text{Beta}(\alpha + 1, \alpha + 1)$ as a prior distribution.

The choice among the Gaussian, Bernoulli and multinomial naive Bayes depends on the feature values.

- If we expect the individual feature values to be roughly normally distributed, Gaussian NB is an obvious choice.
- To use multinomial NB, the features should roughly follow the multinomial distribution – i.e., they must be non-negative, be interpretable as “counts” and “compete” with each other.
 - Note that the feature can be real-valued (the multinomial distribution can be extended to real-value observations using the Γ function).
- In order to use Bernoulli NB, the features *must* be binary. However, an important difference is that contrary to the multinomial NB, the **absence of features** is also modeled by the $(1 - p_{d,k})$ term; the multinomial NB uses $p_{d,k}^0 = 1$ in such case.

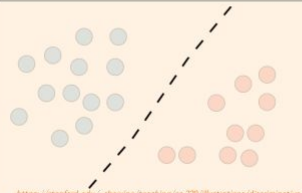
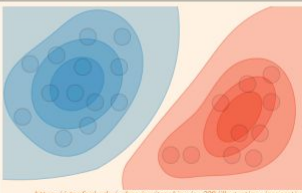
8.9 Q

Describe the difference between a generative and a discriminative model, the strengths of these models, and explain why is logistic regression and multinomial/Bernoulli naive Bayes called a generative-discriminative pair. [5]

So far, all our classification models (but naive Bayes) have been **discriminative**, modeling a **conditional distribution** $p(t|\mathbf{x})$ (predicting some output distribution).

On the other hand, the **generative models** estimate **joint distribution** $p(t, \mathbf{x})$, often by employing Bayes' theorem and estimating $p(\mathbf{x}|t) \cdot p(t)$. They therefore model the probability of the data being generated by an outcome, and only transform it to $p(t|\mathbf{x})$ during prediction.

Answer:

| | Discriminative Model | Generative Model |
|----------------|---|---|
| Goal | Estimate $P(t \mathbf{x})$ | Estimate $P(t, \mathbf{x}) = P(\mathbf{x} t)P(t)$ |
| What's learned | Decision boundary | Probability distribution of the data |
| Illustration |  |  |

- Empirically, **discriminative models perform better in classification tasks**, because modeling the decision boundary is often much easier than modeling the data distribution.
- On the other hand, **generative models can recognize anomalies/outliers/out-of-distribution data** (when the input example has low probability under the data distribution).
- The term **generative** comes from a (theoretical) possibility of "generating" random instances of \mathbf{x} and t . However, just being able to evaluate $p(\mathbf{x}|\mathbf{t})$ does not necessarily mean there is an **efficient procedure** of actually sampling (generating) \mathbf{x} .
 - In recent years, generative modeling combined with deep neural networks created a new family of **deep generative models** like VAE or GAN, which can in fact efficiently generate samples from $p(\mathbf{x})$.

Given that

- **multinomial/Bernoulli naive Bayes fits $\log p(C_k|\mathbf{x})$ as a linear model**, and
- **a logistic regression also fits $\log p(C_k|\mathbf{x})$ as a linear model**.

multinomial/Bernoulli NB and logistic regression form a so-called **generative-discriminative** pair.

Several theorems are known about this generative-discriminative pair (for proofs see the 2002 paper *On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes* by NG and Jordan):

- If the assumed model in naive Bayes is correct, then both logistic regression and naive Bayes converge to the same performance.
- **Asymptotically, logistic regression is always better or equal to the naive Bayes.**
- Let $\epsilon > 0$ be given and let the model contain D features. therefore with fewer rich training examples, NB is a good choice
 - **Logistic regression can reach the optimal error up to ϵ with $\Omega(D)$ training examples.**
 - **naive Bayes can reach the optimal error up to ϵ with $\Omega(\log(D))$ examples.**

9 Lecture

9.1 Q

Prove that independent discrete random variables are uncorrelated. [5]

We define **covariance** of two random variables x, y as

$$\text{cov}(x, y) = \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])].$$

Answer:

Two random variables x, y are **uncorrelated** if $\text{cov}(x, y) = 0$; otherwise, they are **correlated**.

Note that **two independent random variables are uncorrelated**, because

$$\begin{aligned} \text{cov}(x, y) &= \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] && \text{the } x \text{ and } y \text{ happen independently} \\ &= \sum_{x,y} P(x, y)(x - \mathbb{E}[x])(y - \mathbb{E}[y]) && \text{hence on average both } x \text{ and } y \text{ are their respective} \\ &= \sum_{x,y} P(x)P(y)(x - \mathbb{E}[x])(y - \mathbb{E}[y]) && \text{mean value and we get 0} \\ &= \sum_x P(x)(x - \mathbb{E}[x]) \sum_y P(y)(y - \mathbb{E}[y]) \\ &= \mathbb{E}_x[x - \mathbb{E}[x]] \mathbb{E}_y[y - \mathbb{E}[y]] = 0. \end{aligned}$$

However, **dependent random variables can be uncorrelated** – random uniform x on $[-1, 1]$ and $y = |x|$ are not independent (y is completely determined by x), but they are uncorrelated.

9.2 Q

Write down the definition of covariance and Pearson correlation coefficient ρ , including its range. [5]

We define **covariance** of two random variables x, y as

$$\text{cov}(x, y) = \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])].$$

Answer:

There are several ways to measure correlation of random variables x, y .

Pearson correlation coefficient, denoted as ρ or r , is defined as

$$\rho \stackrel{\text{def}}{=} \frac{\text{cov}(x, y)}{\sqrt{\text{Var}(x)}\sqrt{\text{Var}(y)}}$$

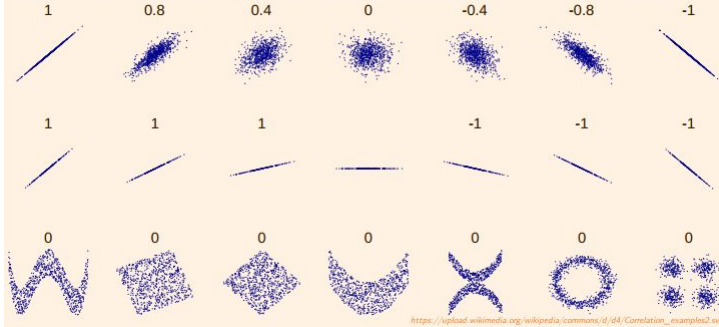
$$r \stackrel{\text{def}}{=} \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}},$$

where:

- ρ is used when the full expectation is computed (population Pearson correlation coefficient);
- r is used when estimating the coefficient from data (sample Pearson correlation coefficient);
 - \bar{x} and \bar{y} are sample estimates of mean.

The value of Pearson correlation coefficient is in fact normalized covariance, because its value is always bounded by $-1 \leq \rho \leq 1$ (and the same holds for r).

Pearson correlation coefficient quantifies **linear dependence** of the two random variables.

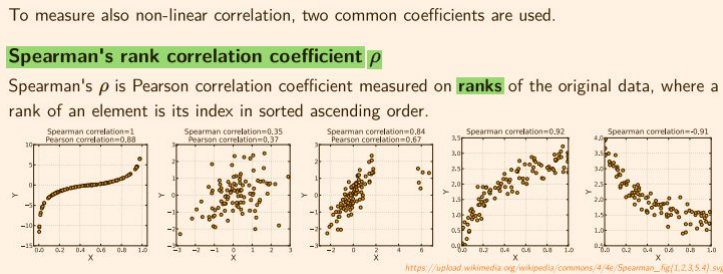


Pozn. (Σ , not for the exam):

Alternatively, the desired inequality can be obtained by applying the Cauchy-Schwarz inequality $\langle u, v \rangle \leq \sqrt{\langle u, u \rangle} \sqrt{\langle v, v \rangle}$ on $\langle x, y \rangle \stackrel{\text{def}}{=} \mathbb{E}[xy]$.

9.3 Q

Explain how are the Spearman's rank correlation coefficient and the Kendall rank correlation coefficient computed (no need to describe the Pearson correlation coefficient). [5]



Answer:

Kendall rank correlation coefficient τ

Kendall's τ measures the amount of **concordant pairs** (pairs where y increases/decreases when x does), minus the **discordant pairs** (where y increases/decreases when x does the opposite):

$$\tau \stackrel{\text{def}}{=} \frac{|\{\text{pairs } i \neq j : x_j > x_i, y_j > y_i\}| - |\{\text{pairs } i \neq j : x_j > x_i, y_j < y_i\}|}{\binom{n}{2}}$$

$$= \frac{\sum_{i < j} \text{sign}(x_j - x_i) \text{sign}(y_j - y_i)}{\binom{n}{2}}.$$

There is no clear consensus whether to use Spearman's ρ or Kendall's τ , but I believe Kendall's τ is a bit more preferred. First, $\frac{1+\tau}{2}$ can be interpreted as a probability of a concordant pair, and Kendall's τ converges to a normal distribution faster.

As defined, the range of Kendall's $\tau \in [-1, 1]$. However, if there are ties, its range is smaller – therefore, several corrections (not discussed here) exist to adjust its value in case of ties.

9.4 Q

Considering an averaging ensemble of M models, prove the relation between the average mean squared error of the ensemble and the average error of the individual models, assuming the

model errors have zero mean and are uncorrelated. [10]

Answer: Context (we want to ultimately 'cancel out' the errors):

Ensembling is combining several models with a goal of reaching higher performance.

The simplest approach is to train several independent models and then combine their outputs by averaging or voting.

The terminology varies, but for classification:

- **voting (or hard voting)** usually means predicting the class predicted most often by the individual models,
- **averaging (or soft voting)** denotes averaging the returned model distributions and predicting the class with the highest probability.

The main idea behind ensembling is that if models have uncorrelated errors, then by averaging model outputs the errors will cancel out.

The averaged MSE:

If we denote the prediction of a model y_i on a training example (\mathbf{x}, t) as $y_i(\mathbf{x}) = t + \epsilon_i(\mathbf{x})$, so that $\epsilon_i(\mathbf{x})$ is the model error on example \mathbf{x} , the mean square error of the model is

$$\mathbb{E}[(y_i(\mathbf{x}) - t)^2] = \mathbb{E}[\epsilon_i^2(\mathbf{x})].$$

Considering M models, we analogously get that the mean square error of the ensemble is

$$\mathbb{E}\left[\left(\frac{1}{M} \sum_i \epsilon_i(\mathbf{x})\right)^2\right].$$

Because

$$\sum \frac{1}{M} y_i = \sum \frac{1}{M} (t + \epsilon_i(x)) = t + \frac{1}{M} \sum \epsilon_i(x)$$

and hence the error of the ensemble is the average of the individual errors. The MSE is then

$$\mathbb{E}\left[\left(\frac{1}{M} \sum \epsilon_i(x)\right)^2\right].$$

Finally, assuming that the individual errors ϵ_i have zero mean and are *uncorrelated*, we get that $\mathbb{E}[\epsilon_i(\mathbf{x})\epsilon_j(\mathbf{x})] = 0$ for $i \neq j$, and therefore,

$$\mathbb{E}\left[\left(\frac{1}{M} \sum_i \epsilon_i(\mathbf{x})\right)^2\right] = \mathbb{E}\left[\frac{1}{M^2} \sum_{i,j} \epsilon_i(\mathbf{x})\epsilon_j(\mathbf{x})\right] = \frac{1}{M} \mathbb{E}\left[\frac{1}{M} \sum_i \epsilon_i^2(\mathbf{x})\right],$$

so the average error of the ensemble is $\frac{1}{M}$ times the average error of the individual models.

9.5 Q

In a regression decision tree, state what values are kept in internal nodes, define the squared error criterion and describe how a leaf is split during training (without discussing splitting constraints). [5]

Assume we have an input dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{t} \in \mathbb{R}^N$. At the beginning, the decision tree is just a single node and all input examples belong to this node. We denote $I_{\mathcal{T}}$ the set of training example indices belonging to a node \mathcal{T} .

For each leaf, our model predict the average of the training examples belonging to that leaf, $\hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i$.

We use a **criterion** $c_{\mathcal{T}}$ telling us how *uniform* or *homogeneous* are the training examples belonging to a node \mathcal{T} – for regression, we employ the sum of squares error between the examples belonging to the node and the predicted value in that node; this is proportional to the variance of the training examples belonging to the node \mathcal{T} , multiplied by the number of the examples. Note that even if it is not *mean* squared error, it is sometimes denoted as MSE.

$$c_{SE}(\mathcal{T}) \stackrel{\text{def}}{=} \sum_{i \in I_{\mathcal{T}}} (t_i - \hat{t}_{\mathcal{T}})^2, \text{ where } \hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i.$$

Answer:

To split a node, the goal is to find a feature and its value such that when splitting a node \mathcal{T} into \mathcal{T}_L and \mathcal{T}_R , the resulting regions decrease the overall criterion value the most, i.e., the difference $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$ is the lowest. split by brute force trying all features and 'all' values. In reality we want to start in a node with low homogeneity of examples and increase it while decreasing the overall criterion value.

9.6 Q

In a K -class classification decision tree, state what values are kept in internal nodes, define the Gini index and describe how a node is split during training (without discussing splitting constraints). [5]

For multi-class classification, we predict such class most frequent in the training examples belonging to a leaf \mathcal{T} .

To define the criterions, let us denote the average probability for class k in a region \mathcal{T} as $p_{\mathcal{T}}(k)$.

For classification trees, one of the following two criterions is usually used:

- **Gini index**, also called **Gini impurity**, measuring how often a randomly chosen element would be incorrectly labeled if it was randomly labeled according to $\mathbf{p}_{\mathcal{T}}$:

$$c_{\text{Gini}}(\mathcal{T}) \stackrel{\text{def}}{=} |\mathcal{I}_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k)(1 - p_{\mathcal{T}}(k)),$$

Answer:

The Gini index is the minimized value of the square loss $\mathcal{L} = \sum_{i \in \mathcal{I}_{\mathcal{T}}} (p - t_i)^2$. (see 'For binary classification, derive the Gini index from a squared error loss. [10]' card)

9.7 Q

In a K -class classification decision tree, state what values are kept in internal nodes, define the entropy criterion and describe how a node is split during training (without discussing splitting constraints). [5]

For multi-class classification, we predict such class most frequent in the training examples belonging to a leaf \mathcal{T} .

To define the criterions, let us denote the average probability for class k in a region \mathcal{T} as $p_{\mathcal{T}}(k)$.

For classification trees, one of the following two criterions is usually used:

Answer:

- **Entropy Criterion**

$$c_{\text{Entropy}}(\mathcal{T}) \stackrel{\text{def}}{=} |\mathcal{I}_{\mathcal{T}}| \cdot H(\mathbf{p}_{\mathcal{T}}) = -|\mathcal{I}_{\mathcal{T}}| \sum_{\substack{k \\ p_{\mathcal{T}}(k) \neq 0}} p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k).$$

9.8 Q

For binary classification, derive the Gini index from a squared error loss. [10]

Recall that $\mathcal{I}_{\mathcal{T}}$ denotes the set of training example indices belonging to a leaf node \mathcal{T} , let $n_{\mathcal{T}}(0)$ be the number of examples with target value 0, $n_{\mathcal{T}}(1)$ be the number of examples with target value 1, and let $p_{\mathcal{T}} = \frac{1}{|\mathcal{I}_{\mathcal{T}}|} \sum_{i \in \mathcal{I}_{\mathcal{T}}} t_i = \frac{n_{\mathcal{T}}(1)}{n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1)}$.
(the probability of 1)

Consider sum of squares loss $L(p) = \sum_{i \in \mathcal{I}_{\mathcal{T}}} (p - t_i)^2$.

By setting the derivative of the loss to zero, we get that the p minimizing the loss fulfils $|\mathcal{I}_{\mathcal{T}}|p = \sum_{i \in \mathcal{I}_{\mathcal{T}}} t_i$, i.e., $p = p_{\mathcal{T}}$.

The value of the loss is then

$$L(p_{\mathcal{T}}) = \sum_{i \in \mathcal{I}_{\mathcal{T}}} (p_{\mathcal{T}} - t_i)^2 = n_{\mathcal{T}}(0)(p_{\mathcal{T}} - 0)^2 + n_{\mathcal{T}}(1)(p_{\mathcal{T}} - 1)^2$$

$$= \frac{n_{\mathcal{T}}(0)n_{\mathcal{T}}(1)^2}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))^2} + \frac{n_{\mathcal{T}}(1)n_{\mathcal{T}}(0)^2}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))^2} = \frac{(n_{\mathcal{T}}(1) + n_{\mathcal{T}}(0))n_{\mathcal{T}}(0)n_{\mathcal{T}}(1)}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))}$$

$$= (n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))(1 - p_{\mathcal{T}})p_{\mathcal{T}} = |\mathcal{I}_{\mathcal{T}}|p_{\mathcal{T}}(1 - p_{\mathcal{T}}).$$

(the probability of choosing zero)

Answer:

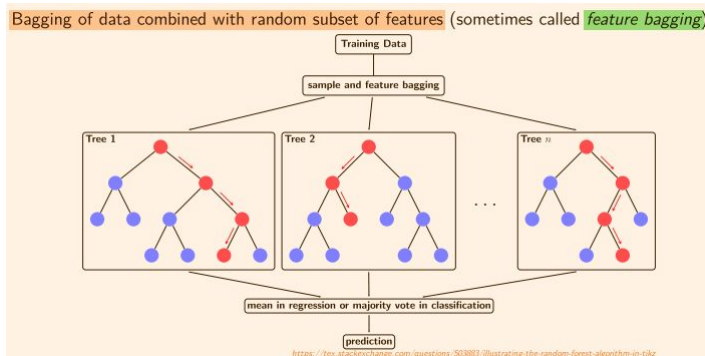
9.9 Q

For K -class classification, derive the entropy criterion from a non-averaged NLL loss. [10]

Answer: lecture 9, 1:28:00 cca, TODO

9.10 Q

Describe how is a random forest trained (including bagging and random subset of features) and how is prediction performed for regression and classification. [5]



Answer:

Bagging

Every decision tree is trained using bagging (on a bootstrapped dataset).

Random Subset of Features

During each node split, only a random subset of features is considered, when finding the best split. A fresh random subset is used for every node.

Extra Trees

The so-called extra trees are even more randomized, not finding the best possible feature value when choosing a split, but considering uniformly random samples from a feature's empirical range (minimum and maximum in the training data).

10 Lecture

10.1 Q

Write down the loss function which we optimize in gradient boosting decision tree during the construction of t^{th} tree. Then define g_i and h_i and show the value $w_{\mathcal{T}}$ of optimal prediction in node \mathcal{T} . [10]

Answer: Context:

Considering a regression task first, we define the overall loss as

$$\mathcal{L}(\mathbf{W}) = \sum_i \ell(t_i, y(\mathbf{x}_i; \mathbf{W})) + \sum_{t=1}^T \frac{1}{2} \lambda \|\mathbf{W}_t\|^2,$$

where

- $\mathbf{W} = (\mathbf{W}_1, \dots, \mathbf{W}_T)$ are the parameters (leaf values) of the trees;
- $\ell(t_i, y(\mathbf{x}_i; \mathbf{W}))$ is an per-example loss, $(t_i - y(\mathbf{x}_i; \mathbf{W}))^2$ for regression;
- the λ is the usual L_2 regularization strength.

To construct the trees sequentially, we extend the definition to

$$\mathcal{L}^{(t)}(\mathbf{W}_t; \mathbf{W}_{1:t-1}) = \sum_i \left[\ell(t_i, y^{(t-1)}(\mathbf{x}_i; \mathbf{W}_{1:t-1}) + y_t(\mathbf{x}_i; \mathbf{W}_t)) \right] + \frac{1}{2} \lambda \|\mathbf{W}_t\|^2.$$

Answer: In each update we are creating a new tree, so traditional SGD update would be too expensive and Newton's (or modified) method is used in our approach:

However, a more principled approach was later suggested. Denoting

$$g_i = \frac{\partial \ell(t_i, y^{(t-1)}(\mathbf{x}_i))}{\partial y^{(t-1)}(\mathbf{x}_i)}$$

and

$$h_i = \frac{\partial^2 \ell(t_i, y^{(t-1)}(\mathbf{x}_i))}{\partial y^{(t-1)}(\mathbf{x}_i)^2},$$

we can expand the objective $\mathcal{L}^{(t)}$ using a second-order approximation to

$$\mathcal{L}^{(t)}(\mathbf{W}_t; \mathbf{W}_{1..t-1}) \approx \sum_i \left[\ell(t_i, y^{(t-1)}(\mathbf{x}_i)) + g_i y_t(\mathbf{x}_i) + \frac{1}{2} h_i y_t^2(\mathbf{x}_i) \right] + \frac{1}{2} \lambda \|\mathbf{W}_t\|^2.$$

where the loss can be rewritten with g_i, h_i as:

Recall that we denote the indices of instances belonging to a node \mathcal{T} as $I_{\mathcal{T}}$, and let us denote the prediction for the node \mathcal{T} as $w_{\mathcal{T}}$. Then we can rewrite

$$\begin{aligned} \mathcal{L}^{(t)}(\mathbf{W}_t; \mathbf{W}_{1..t-1}) &\approx \sum_i \left[g_i y_t(\mathbf{x}_i) + \frac{1}{2} h_i y_t^2(\mathbf{x}_i) \right] + \frac{1}{2} \lambda \|\mathbf{W}_t\|^2 + \text{const} \\ &\approx \sum_{\mathcal{T}} \left[\left(\sum_{i \in I_{\mathcal{T}}} g_i \right) w_{\mathcal{T}} + \frac{1}{2} \left(\lambda + \sum_{i \in I_{\mathcal{T}}} h_i \right) w_{\mathcal{T}}^2 \right] + \text{const}. \end{aligned}$$

By setting a derivative with respect to $w_{\mathcal{T}}$ to zero, we get

$$0 = \frac{\partial \mathcal{L}^{(t)}}{\partial w_{\mathcal{T}}} = \sum_{i \in I_{\mathcal{T}}} g_i + \left(\lambda + \sum_{i \in I_{\mathcal{T}}} h_i \right) w_{\mathcal{T}}.$$

Therefore, the optimal weight for a node \mathcal{T} is

$$w_{\mathcal{T}}^* = - \frac{\sum_{i \in I_{\mathcal{T}}} g_i}{\lambda + \sum_{i \in I_{\mathcal{T}}} h_i}.$$

10.2 Q

Write down the loss function which we optimize in gradient boosting decisiontree during the construction of t^{th} tree. Then define g_i and h_i and the criterion used during node splitting. [10]

Answer: In each update we are creating a new tree, so traditional SGD update would be too expensive and Newton's (or modified) method is used in our approach:

However, a more principled approach was later suggested. Denoting

$$g_i = \frac{\partial \ell(t_i, y^{(t-1)}(\mathbf{x}_i))}{\partial y^{(t-1)}(\mathbf{x}_i)}$$

and

$$h_i = \frac{\partial^2 \ell(t_i, y^{(t-1)}(\mathbf{x}_i))}{\partial y^{(t-1)}(\mathbf{x}_i)^2},$$

we can expand the objective $\mathcal{L}^{(t)}$ using a second-order approximation to

$$\mathcal{L}^{(t)}(\mathbf{W}_t; \mathbf{W}_{1..t-1}) \approx \sum_i \left[\ell(t_i, y^{(t-1)}(\mathbf{x}_i)) + g_i y_t(\mathbf{x}_i) + \frac{1}{2} h_i y_t^2(\mathbf{x}_i) \right] + \frac{1}{2} \lambda \|\mathbf{W}_t\|^2.$$

Recall that we denote the indices of instances belonging to a node \mathcal{T} as $I_{\mathcal{T}}$, and let us denote the prediction for the node \mathcal{T} as $w_{\mathcal{T}}$. Then we can rewrite

$$\begin{aligned} \mathcal{L}^{(t)}(\mathbf{W}_t; \mathbf{W}_{1..t-1}) &\approx \sum_i \left[g_i y_t(\mathbf{x}_i) + \frac{1}{2} h_i y_t^2(\mathbf{x}_i) \right] + \frac{1}{2} \lambda \|\mathbf{W}_t\|^2 + \text{const} \\ &\approx \sum_{\mathcal{T}} \left[\left(\sum_{i \in I_{\mathcal{T}}} g_i \right) w_{\mathcal{T}} + \frac{1}{2} \left(\lambda + \sum_{i \in I_{\mathcal{T}}} h_i \right) w_{\mathcal{T}}^2 \right] + \text{const}. \end{aligned}$$

By setting a derivative with respect to $w_{\mathcal{T}}$ to zero, we get

$$0 = \frac{\partial \mathcal{L}^{(t)}}{\partial w_{\mathcal{T}}} = \sum_{i \in I_{\mathcal{T}}} g_i + \left(\lambda + \sum_{i \in I_{\mathcal{T}}} h_i \right) w_{\mathcal{T}}.$$

Therefore, the optimal weight for a node \mathcal{T} is

$$w_{\mathcal{T}}^* = - \frac{\sum_{i \in I_{\mathcal{T}}} g_i}{\lambda + \sum_{i \in I_{\mathcal{T}}} h_i}.$$

Substituting the optimum weights to the loss, we get

$$\mathcal{L}^{(t)}(\mathbf{W}) \approx -\frac{1}{2} \sum_T \frac{(\sum_{i \in I_T} g_i)^2}{\lambda + \sum_{i \in I_T} h_i} + \text{const},$$

which can be used as a splitting criterion.

10.3 Q

How is the learning rate used during training and prediction of a gradient boosting decision tree? [5]

Answer: Furthermore, gradient boosting trees frequently use:

- data subsampling: either bagging, or (even more commonly) utilize only a fraction of the original training data for training a single tree (with 0.5 being a common value),
- feature subsampling;
- **shrinkage:** multiply each trained tree by a learning rate α , which **reduces influence of each individual tree** and leaves space for future optimization.

If one model is responsible for most of the work, in case it was wrong, the small corrections of subsequent (or other) models wouldn't have the strength to correct it, hence we limit it to only eliminate a certain amount of the errors. So we still train it to correct as much as possible and then we take this correction and scale it (e.g. by 0,1):

$$y(t) = \sum \alpha y_i$$

10.4 Q

For a K -class classification, describe how to perform prediction with a gradient boosting decision tree trained for T time stamps (how the individual trees perform prediction and how are the $K \cdot T$ trees combined to produce the predicted categorical distribution). [5]

Answer: In multiclass classification it is easier to generate K trees, each for one logit (one class), so tree 1 will be tasked with saying whether the example belongs to class one or not, not to which other class it belongs in case it doesn't belong to class one. This is the decision capacity we want to maximize for each tree.

For multiclass classification, we need to model the full categorical output distribution. Therefore, for each "timestep" t , we train K trees $\mathbf{W}_{t,k}$, each predicting a single value of the linear part of a generalized linear model.

Then, we perform prediction by

$$\text{softmax}(\mathbf{y}(\mathbf{x}_i)) = \text{softmax} \left(\sum_{t=1}^T y_{t,1}(\mathbf{x}_i; \mathbf{W}_{t,1}), \dots, \sum_{t=1}^T y_{t,K}(\mathbf{x}_i; \mathbf{W}_{t,K}) \right),$$

Our trees form '2D matrix' where the final prediction is performed as the softmax of the sums of the columns (sums for each class - logits for each class). These sums are the input of softmax, that gives us a distribution.

10.5 Q

Considering a K -class classification, describe which individual trees (and in which order) are created during gradient boosted decision tree training, and what per-example loss is used for training every one of them (expressed using predictions of the already trained trees). You do not need to describe the training process of the individual trees themselves. [10]

Answer: In the first step we train K trees, one for each class and these try to decide the predictions. These predictions are then used in softmax to generate the distribution which we'll try to correct. In $t = 2$ we create K more trees, each one of which will try to correct the logits for their given class so that the sums help the final distribution the most, si at every t we create K trees that we train in parallel (gradient is different for every part of the model, that work together after each timestamp).

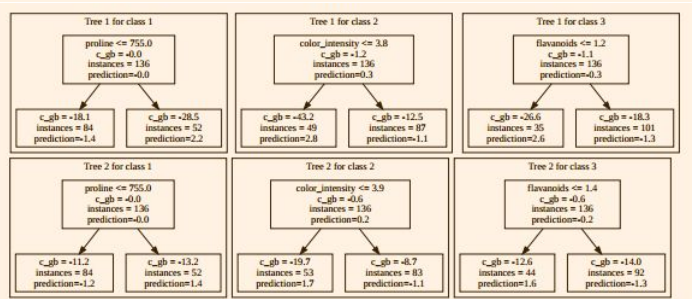
For multiclass classification, we need to model the full categorical output distribution. Therefore, for each "timestep" t , we train K trees $\mathbf{W}_{t,k}$, each predicting a single value of the linear part of a generalized linear model.

Then, we perform prediction by

$$\text{softmax}(\mathbf{y}(\mathbf{x}_i)) = \text{softmax}\left(\sum_{t=1}^T y_{t,1}(\mathbf{x}_i; \mathbf{W}_{t,1}), \dots, \sum_{t=1}^T y_{t,K}(\mathbf{x}_i; \mathbf{W}_{t,K})\right),$$

and the per-example loss is defined analogously as

$$\ell(t_i, \mathbf{y}(\mathbf{x}_i)) = -\log(\text{softmax}(\mathbf{y}(\mathbf{x}_i))_{t_i}).$$



11 Lecture

11.1 Q

When deriving the first principal component, write the value of the variance we aim to maximize, both without and with the covariance matrix (and define the covariance matrix). [5]

Answer:

11.2 Q

When deriving the first M principal components, write the value of the reconstruction loss we aim to minimize using all but the first M principal components, both without and with the covariance matrix (and define the covariance matrix). [10]

Answer:

11.3 Q

Write down the formula for whitening (sphering) the data matrix \mathbf{X} , and state what mean and covariance does the result has. [5]

Answer:

11.4 Q

Explain how to compute the first M principal components using the SVD decomposition of the data matrix \mathbf{X} , and why it works. [5]

Answer:

11.5 Q

Write down the algorithm of computing the first M principal components of the data matrix \mathbf{X} using the power iteration algorithm. [10]

Answer:

11.6 Q

Describe the K-means algorithm, including the kmeans++ initialization. [10]

Answer:

12 Lecture

12.1 Q

Define the multivariate Gaussian distribution of dimension D . [5]

Answer:

12.2 Q

Show how to sample from a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with a full covariance matrix, by using random samples from $\mathcal{N}(0, \mathbf{I})$ distribution. [5]

Answer:

12.3 Q

Describe the constant surfaces of a multivariate Gaussian distribution with (1) $\sigma^2 \mathbf{I}$ covariation, (2) a diagonal covariation matrix, (3) a full covariation matrix. [5]

Answer:

13 Lecture

13.1 Q

Considering a Gaussian mixture with K clusters, explain how we represent the individual clusters and write down the likelihood of an example \mathbf{x} for a given Gaussian mixture. [5]

Answer:

13.2 Q

Write down the log likelihood of an N -element dataset for a given Gaussian mixture model with K components. [5]

Answer:

13.3 Q

Considering the algorithm for Gaussian mixture clustering, write down the E step (how to compute the responsibilities) and the M step (how to update the means, covariances and priors of the individual clusters). [10]

Answer:

13.4 Q

Write down the MSE loss of a regression problem, and formulate the bias-variance trade-off, i.e., the decomposition of expected MSE loss (with respect to a randomly sampled test set) into bias, variance and irreducible error terms. [10]

Answer:

13.5 Q

Considering statistical hypothesis testing, define type I errors and type II errors (in terms of the null hypothesis). Finally define what a significance level is. [5]

Answer:

13.6 Q

Explain what a test statistic and a p-value are. [5]

Answer:

13.7 Q

Write down the steps of a statistical hypothesis test. [5]

Answer:

13.8 Q

Explain the differences between a one-sample test, two-sample test and a paired test. [5]

Answer:

13.9 Q

When considering multiple comparison problem, define the family-wise error rate, and formulate the Bonferroni correction, which allows to limit the family-wise error rate by a given α . [5]

Answer:

13.10 Q

For a trained model and a given test set with N examples and metric E , write how to estimate 95% confidence intervals using bootstrap resampling. [5]

Answer:

13.11 Q

For two trained models and a given test set with N examples and metric E , explain how to perform a paired bootstrap test that the first model is better than the other. [5]

Answer:

13.12 Q

For two trained models and a given test set with N examples and metric E , explain how to perform a random permutation test that the first model is better than the other with a significance level α . [5]

Answer:

List of Theorems

| | | |
|-----|---------------------------------------|---|
| 2.1 | Definition (Expected value) | 5 |
| 2.2 | Definition (Variance) | 5 |

List of Theorems