MaMa Summary

November 10, 2023

Contents

1 Classification Learning

1.1 vocabulary

- domain set X: set of possible inputs
- classes or label set Y: target of classification
- data points or samples: $x \in X$
- features or attributes: entries of x
- training set $S \subseteq X$: pairs (x, y) of data points x and labels y
- classifier h: function $X \to Y$

1.2 loss and error

Let h be a classifier $h : X \to Y$.

- loss function: $l(y, y') \geq 0$ for $y, y' \in Y$.
- zero-one loss:
$$l_{0-1}(y, y') = \begin{cases} 0, & \text{if } y = y' \\ 1, & \text{if } y \neq y' \end{cases}$$
- training error:
$$L_S(h) = \frac{1}{|S|} \sum_{(x,y) \in S} l(y, h(x))$$

Example 1.2.1. *Let $X = \mathbb{R}^2$ and $Y = \{-1, 1\}$. The easiest function is a linear classifier that splits X in two parts and characterizes data points accordingly. For the general case of $X = \mathbb{R}^n$, $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ we have:*

$$h_{w,b} = sgn(w^T x + b)$$

You can get rid of the bias b by adapting the training set a little:

$$\overline{S} = (\begin{pmatrix} x \\ 1 \end{pmatrix}, y) : (x, y) \in S$$

where the new linear classifier is defined by

$$\overline{h}_{\overline{w}}(x) = sgn(\overline{w}^T \overline{x})$$

with $\overline{w} = \binom{w}{b}$.

<u>When is it possible to achieve training error 0?</u> (wrt zero-one-loss)
This is precisely then the case, when $\exists w \in \mathbb{R}^n$ s.t. $\forall (x, y) \in S$

- if $y = 1$ then $w^T x \geq 0 \leftrightarrow$ if $y = 1$ then $w^T x > 0$
- if $y = -1$ then $w^T x < 0$

Both can be generalized by saying

$$y \cdot w^T x > 0$$

or

$$\exists w : y \cdot w^T x \geq 1 \; \forall (x, y) \in S$$

A training set with 0 training error is called separable.

1.3 <u>logistic regression</u>

Logistic regression computes a linear classifier.
<u>direct way:</u>
Look for a $w \in \mathbb{R}^n$ such that the training error is minimised, i.e.

$$\min_{w \in \mathbb{R}^n} \frac{1}{|S|} \sum_{(x,y) \in S} h_{0-1}(y, h_w(x))$$

Definition 1.3.1 (logistic function)**.** The logistic function $\phi_{sig} : \mathbb{R} \to [0, 1]$ is defined by

$$z \mapsto \frac{1}{1 + e^{-z}}$$

We then solve

$$\min_{w \in \mathbb{R}^n} \frac{1}{|S|} \sum_{(x,y) \in S} -\log_2(\phi_{sig}(y w^T x))$$

Lemma 1.3.2. *For all training sets* $S \subseteq \mathbb{R}^n \times \{-1, 1\}$ *it holds that*

$$\frac{1}{|S|} \sum_{(x,y) \in S} h_{0-1}(y, h_w(x)) \leq \frac{1}{|S|} \sum_{(x,y) \in S} -log_2(\phi_{sig}(y w^T x))$$

1.4 <u>training error and real life</u>

A classifier is only good, if it performs good on new data. To test your classifier you need to evaluate the classifier on data it hasn't seen during training. This is called the test error. The question now is, how the data should be split into test and training data. This is a difficult question and should be evaluated for each use case separately. A part from the training set should also be used for validation if there are degrees of freedom in the chosen classifier.

1.5 Quadratic classifier

This is another binary classifier. In this case wa search for a matrix $U \in \mathbb{R}^{n\times n}$ of weights, a vector $w \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$. The classifier is then defined by

$$h(x) = sgn\left(\sum_{i,j=1}^{n} u_{i,j}x_ix_j + \sum_{i=1}^{n} w_ix_i + b\right) = sgn(x^Tux + w^Tx + b)$$

To simplify things we redefine the training set S by $\overline{S}$ containing all samples (x, y) with $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ and y analogously defined. Redefine x by

$$\overline{x} = (x_1^2, x_1x_2, x_1x_3, .., x_n^2, .., x_n, 1)$$

Doing this will transform the quadratic classifier into a linear one.

1.6 nearest neighbour classifier

The training set again is defined by $S \subseteq X \times Y$. When adding a new data-point, we check what class the nearest data point belongs to and add the new point to the same class. In the case of k-nearest-neighbour we check the classes of the k nearest points. The problem is that the training set must be in memory the entire time. This makes the learning potentially very slow.

1.7 decision tree classifier

The classifier of a decision tree works as follows:

1. Set $v = r$ where r is the tree's root
2. while v is not a leaf do
3. Let (i, j) be the decision rule of v
4. If $x_i \leq t$ set $v = v_L$ else set $v = v_R$
5. end while

6. Output the class $c(v)$ of v

The following now describes how a tree can be learned.
The idea is to start with a single node r where the class $c(r)$ is just the majority class in S.
Iteratively decide for each leaf v, if it is beneficial to split it into two child nodes. Let S_v be the path of the training set that, following the existing decisions, ends up in v. The split into S_L and S_R would then be defined by a feature i and a threshold t. We define the gain of the split by

$$gain(v, i, t) = \gamma(S_v) - \left(\frac{|S_L|}{|S_R|}\gamma(S_L) + \frac{|S_R|}{|S_L|}\gamma(S_R) \right)$$

where γ is a inhomogeneity measure. This can be defined in multiple ways.

Definition 1.7.1. The inhomogeneity can be defined by the training error

$$\gamma(S_v) = 1 - \max_{y \in Y} p(y, S_v)$$

where

$$p(y, S_v) = \frac{|\{(x, y') \in S_v :\ y' = y\}|}{|S_v|}$$

Definition 1.7.2 (gini impurity)**.** This is used in scikit-learn.

$$\gamma(S_v) = 1 - \sum_{y \in Y} p(y, S_v)^2$$

Definition 1.7.3 (entropy)**.**

$$\gamma(S_v) = - \sum_{y \in Y} p(y, S_v) \cdot \log_2(y, S_v)$$

1.8 loss functions

So far, we only looked at the zero-one-loss $l_{0-1}(y, y') = \begin{cases} 1, \ y \neq y' \\ 0, \ y = y' \end{cases}$ where y is the true class and y' is the prediction.
Consider the use case of a spam filter. This is a binary classifier that checks whether a new mail is spam or not. This admits two kind of errors:

- false-positive: good mail is classified as spam
- false-negative: spam mail is classified as good

The case of false-positive is more serious in this example. This needs to be applied to the loss function.

$$l(y,y') = \begin{cases} 0, \ y = y' \\ 10, \ \text{if } y \text{ is good and } y' \text{ spam} \\ 1, \ \text{if } y \text{ is spam and } y' \text{ is good} \end{cases}$$

Example 1.8.1 (loss functions in regressions)**.** *Consider a predictor $h : \mathbb{R}^n \to \mathbb{R}$. The (mean)square-loss is defined by*

$$l(y,y') = (y - y')^2$$

and the (mean) absolute loss

$$l(y,y') = |y - y'|$$

1.9 A statistical model

Let X and Y be sets as above. We define a probability distribution D on $X \times Y$ with the following assumptions

- D is unknown
- if D were defined on X only, then $\mathbb{P}[p] = \begin{cases} \text{large if picture shows cat or dog} \\ \text{small if not} \end{cases}$ where p is a picture. Instead D is defined on $X \times Y$ because there might be some uncertainty in X
- iid: data points of the training set are drawn from D independently
- D is fixed

A classifier $h^* : X \to Y$ works well on new data if

$$L_D(h^*) = \mathbb{E}_{(x,y)\sim D}[l(y, h^*(x))]$$

is small. This is called true risk or generalization error. For classification using the zero-one-loss this simplifies to

$$L_D(h^*) = \mathbb{P}_{(x,y)\sim D}[h^*(x) \neq y]$$

1.10 Bayes error and classifier

The Bayes error is the smallest error that any classifier can achieve.

$$\varepsilon_{bayes} = \inf_h L_D(h)$$

A Bayes classifier h^* such that $L_D(h^*) = \varepsilon_{bayes}$ is called a Bayes classifier.

Theorem 1.10.1. *For all classifiers $h : X \to Y$ it holds that*

$$L_D(h) \geq L_D(h_{bayes})$$

2 PAC learning

Let H be a set of classifiers. Draw a training set S from D. The goal is to minimize $h_S = \arg\min_{h \in H} L_S(h)$. This is called the method of empirical risk minimization.

2.1 empirical risk minimisation

Lemma 2.1.1 (Hoeffding's inequality)**.** *Given independent random variables $X_1, ..., X_m : \Omega \to [a, b]$ with $\mathbb{E}[X_i] = \mu$ for all $i \in [n]$. Then*

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right|\right] \leq 2 \cdot \exp\left(-2\frac{m\varepsilon^2}{(b-a)^2}\right)$$

We now want to figure out the connection between $L_S(h)$ and $L_D(H)$. Let $|S| = m$ and $S = \{(x_1, y_1), ...(x_m, y_m)\}$. Define $X_i = l(y_i, h(x_i))$. Then the training error is given by

$$\frac{1}{m} = \sum_{i=1}^{m} X_i = L_S(h)$$

Furthermore, $\mathbb{E}[X_i] = \mathbb{E}_{(x,y)\sim D}[l(y, h(x))] = L_D(h)$. Assuming zero-one-loss, Hoeffing implies

$$\mathbb{P}[|L_S(h) - L_D(h)| \geq \varepsilon] \leq 2 \cdot \exp(-2m\varepsilon^2) := \delta$$

Then $\varepsilon = \sqrt{\frac{\ln(\frac{2}{\delta})}{2m}}$

Theorem 2.1.2. *With probability $\geq 1 - \delta$ it holds that*

$$|L_S(h) - L_D(h)| \leq \sqrt{\frac{\ln(\frac{2}{\delta})}{2m}}$$

This does not however imply that with high probability $L_S(h_S) \approx L_D(h_S)$. It is however true for the test-set: with probability $1-\delta$ it holds that $|L_T(h_S) - L_D(h_S)| \leq \sqrt{\frac{\ln(\frac{2}{\delta})}{2|T|}}$.

2.2 error decomposition

Assume we have a training set S and a test set T and the training error h_S has been minimised by empirical risk minimisation. We are interested in the generalisation error $L_D(h_s)$. For example assume a training error of 9% and a test error of 12%. We can decompose the unknown generalisation error as follows:

$$L_D(h_S) = \underbrace{(L_D(h_S) - L_T(h_S))}_{\text{small if } T \text{ reasonably large}} + \underbrace{(L_T(h_S) - L_S(h_S))}_{\text{generalisation gap,12\%}-9\%} + \underbrace{L_S(h_S)}_{9\%}$$

A large generalisation gap means that the classifier learns the training set and not the underlying distribution D. That means the classifier is overfitting.

2.3 finitely many classifiers

Lemma 2.3.1. *Let H be a set of classifiers. Assume that for every $\varepsilon, \delta > 0$, there is an m s.t. when drawing a sample S of size at least m then, with a probability at least $1-\delta$ it holds that*

$$\sup_{h \in H} \left| L_D(h) - LS_(H) \right| \leq \frac{\varepsilon}{2}$$

Then with probability at least $1-\delta$ it holds that

$$L_D(h_S) \leq \inf_{h \in H} L_D(h) + \varepsilon$$

These assumptions are called the uniform convergence property.

Theorem 2.3.2. *Let $\varepsilon, \delta > 0$ and H be a finite class of classifiers. Let the training set S have size $m \geq \frac{2}{\varepsilon^2} \ln \frac{2|H|}{\delta}$. Then with probability at least $1-\delta$ it holds that*

$$L_D(h_S) \leq \min_{h \in H} L_D(h) + \varepsilon$$

This proves that empirical risk minimisation works.

2.4 probably approximately correct (pac)

Let H be a set of classifiers. H is agnostically PAC learnable if
(informal) there is a learning algorithm that provided the training set is large enough, returns an almost optimal classifier.

(formal) $\exists m_H : (0,1)^2 \to \mathbb{Z}_+$ and a learning algorithm A (think risk minimisation) s.t. $\forall \varepsilon, \delta \in (0,1)$ and $\forall$ probability distributions D on $X \times Y$ if S with $|S| \geq m_H(\varepsilon, \delta)$ is iid drawn from D then with probability $\geq 1 - \delta$ it holds that

$$L_D(A(S)) \leq \inf_{h \in H} L_D(h) + \varepsilon$$

Classes H that are finite are PAC-learnable. Also classes that satisfy the uniform convergence property are PAC-learnable as well.

2.5 VC-dimension

Very informally this dimension gives an idea of how powerful a class of classifiers can be. The VC-dimension is only defined for binary classifiers, say with classes 0 and 1. Let H be a set of binary classifiers (for example axis-parallel rectangle classifiers, H $= \{I_R :$ R axis parallel rectangle$\}$, so $I_R(x) = \begin{cases} 1, x \in R \\ 0, x \notin R \end{cases}$). Formal definition: We say that H shatters a set $C \subset X$ if

$$\{h|_C : h \in H\} = H|_C \overset{!}{=} \{f : C \to \{0,1\}\} \Leftrightarrow |H|_C| = 2^{|C|}$$

The VC-dimension of H is then the largest d s.t.

$$\exists C \subseteq X \text{ of } |C| = d \text{ s.t. } C \text{ is shattered by } H$$

The dimension of H is infinite if $\forall k \in \mathbb{Z}_+ \; \exists C \subseteq X$ of $|C| = X$ that is shattered by H.
It is easy to see that the VC-dimension of the axis parallel rectangle classifier is at least 4 and in fact it is exactly for as we will see below.
Consider any set of 5 points in $\mathbb{R}^2$. Let p_1 be the point with the largest y-coordinate and p_2 be the point with the smallest y-coordinate. p_3 has the largest x coordinate and p_4 has the smallest. Notice that each pair of points may be identical. Finally, there is a fifth point q that is somewhere in the rectangle spanned by those four points. If we give points p_1 to p_4 the class 1 and q class 0, then we can't shatter any set of five points implying that VC-dim ≤ 4.

Example 2.5.1. *Let $X = \mathbb{R}$. Define*

$$H = \{h_{[a,b]} : a \leq b\}$$

where

$$h_{[a,b]}(x) = \begin{cases} 1 \; x \in [a,b] \\ 0; \;\; else \end{cases}$$

Obviously $VC(H) \geq 2$ *because there is always an interval that contains two points. Indeed the dimension is exactly 2 because if we have* $x_1 < x_2 < x_3$ *with classes 1, 0 and 1, there is no classifier that is correct.*

The class of homogeneous linear classifiers in $\mathbb{R}^d$

$$h_w : x \mapsto sgn(w^T x)$$

has VC-dimension of d.

2.6 Fundamental Theorem of PAC-learning

Theorem 2.6.1. *Let* H *be a set of binary classifiers. Then* H *is PAC-learnable iff the VC dimension of* H *is finite.*

Theorem 2.6.2. *Let* H *be a set of binary classifiers with* $VC(H) = d < \infty$*. Then* $\exists c > 0$ *s.t.* $\forall \varepsilon, \delta > 0$ *it holds that with probability at least* $1 - \delta$

$$L_D(h_S) \leq \inf_{h \in H} L_D(h) + \sqrt{c \frac{d + \log(\frac{1}{\delta})}{m}}$$

Definition 2.6.3. Let H be a set of binary classifiers. We define the growth function

$$\tau : \mathbb{N} \to \mathbb{N}$$

with

$$\tau(n) = \max_{C \subseteq X, |C| = m} |H|_C| = \max |\{h|_c : C \to \{0,1\} : h \in H\}|$$

Lemma 2.6.4. *Let* H *be a set of binary classifiers with* $VC(H) = d < \infty$*. Then*

$$\tau(m) \leq \sum_{i=0}^{d} \binom{m}{i} \; \forall m \in \mathbb{N}$$

In particular if

$$m \geq d + 1$$

then $\tau(m) \leq (d+1) \cdot m^{d+1}$.

Theorem 2.6.5. *Let H be a set of binary classifiers. Then for every $\delta > 0$ with probability of at least $1-\delta$ over the choice $S \sim D^m$ of the training set it holds that*

$$\sup_{h\in H} |L_D(h) - L_S(h)| \leq \frac{4+\sqrt{\log(\tau_H(2\cdot m))}}{\delta\sqrt{2\cdot m}}$$

Lemma 2.6.6. *Let H be a set of binary classifiers. Then for every $\delta > 0$ with probability at least $1-\delta$ over the choice $S \sim D^m$ of the training set it holds that*

$$\sup_{h\in H} |L_D(h) - L_S(h)| \leq \frac{8+2\sqrt{\log(\tau_H(2\cdot m))}}{\delta\sqrt{m}}$$

Note: The lemma is a weaker bound than the one in above theorem and can thus be ignored.

Theorem 2.6.7. *Let H be a set of binary classifiers with the domain set X of VC-dimension $\geq 2m$ and a learning algorithm A. Then there is a distribution D on $X \times \{0,1\}$ s.t.*

1. *$\exists h^* \in H$ with $L_D(h^*) = 0$*
2. *with probability $\geq \frac{1}{7}$ and a choice of training set S with $|S| = m$ we have:*

$$L_D(A(S)) \geq \frac{1}{8}$$

2.7 VC-dimension of linear classifiers

Theorem 2.7.1. *The class of linear classifiers in $\mathbb{R}^d$*

$$H = \{x \mapsto sgn(w^T x) : \ w \in \mathbb{R}^d\}$$

has VC-dimension d.

2.8 neural networks

For an input vector x, weights w and a bias b the output of an artificial neuron is $w^T x + b$. A classification network is made up of an input layer, the hidden layer where the neurons are and the output layer. In such a case, the weight vector w would be a weight matrix $W^{(1)}$. The each neuron then has a weight to the output neuron which also has a bias $b^{(2)}$ so the final output is then given by $sgn(W^{(2)}h + b^{(2)})$ where h is the output of the hidden layer. One can notice that this is simply an affine function. Hence neurons are augmented by an activation function, e.g. $\text{ReLU}(x) = \max(0, x)$ (=Rectified linear unit).

If the activation function is non-linear one can calculate more difficult stuff than with normal affine functions.
Activation functions $\sigma : \mathbb{R} \to \mathbb{R}$ are applied component-wise. It is important to note, that neurons on the same layer must share the same activation function. The necessity of activation functions becomes inherently apparent, when one tries to compute XOR with a neural network. Even though XOR is not separable, one can compute the XOR function using a neural network, if an activation function like ReLU is used.
Modern neural networks consist of many hidden layers, each equipped with ReLU while the output layer usually uses the logistic function or softmax. A logistic function output does not give a simple class but a confidence level instead. This is a difference to the classical classification task. Quite similarly, one can use a softmax function defined by

$$z \mapsto \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

This gives a distribution instead.