Classification
Finite Hypothesis Classes

prof. dr Arno Siebes

Algorithmic Data Analysis Group
Department of Information and Computing Sciences
Universiteit Utrecht

Recap

We want to learn a classifier, i.e., a computable function

$$f : \mathcal{X} \to \mathcal{Y}$$

using a finite sample

$$D \sim \mathcal{D}$$

Ideally we would want a function h that minimizes:

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

But because we do not know either f nor $\mathcal{D}$ we settle for a function h that minimizes

$$L_D(h) = \frac{|\{(x_i, y_i) \in D \mid h(x_i) \neq y_i\}|}{|D|}$$

We start with a finite hypothesis class $\mathcal{H}$

Finite isn't that Trivial?

Examples of finite hypothesis classes are

- threshold function with 256 bits precision reals
  - who would need or even want more?
- conjunctions
  - a class we will meet quite often during the course
- all Python programs of at most 10^{32} characters
  - automatic programming aka inductive programming
  - given a (large) set of input/output pairs
  - you don't program, you learn!

Whether or not these are trivial learning tasks, I'll leave to you

- but, if you think automatic programming is trivial, I am interested in your system

It isn't just about theory, but also very much about practice.

The Set-Up

We have

- a finite set $\mathcal{H}$ of hypotheses
- and a (finite) sample $D \sim \mathcal{D}$
- and there exists a function $f : \mathcal{X} \to \mathcal{Y}$ that does the labelling

Note that since Y is completely determined by X, we will often view $\mathcal{D}$ as the distribution for $\mathcal{X}$ rather than for $\mathcal{X} \times \mathcal{Y}$.

The ERM$_{\mathcal{H}}$ learning rule tells us that we should pick a hypothesis h_D such that

$$h_D \in \operatorname*{argmin}_{h \in \mathcal{H}} L_D(h)$$

That is we should pick a hypothesis that has minimal empirical risk

The Realizability Assumption

For the moment we are going to assume that the true hypothesis is in $\mathcal{H}$; we will relax this later. More precisely, we are assuming that

there exists a $h^* \in \mathcal{H}$ *such that* $L_{\mathcal{D},f}(h^*) = 0$

Note that this means that with probability 1

- $L_D(h^*) = 0$

(there are bad samples, but the vast majority is good).

This implies that,

- for (almost any) sample D the ERM$_{\mathcal{H}}$ learning rule will give us a hypothesis h_D for which

$$L_D(h_D) = 0$$

The Halving Learner

A simple way to implement the ERM$_{\mathcal{H}}$ learning rule is by the following algorithm; in which V_t denotes the hypotheses that are still viable at step t

- the first t $d \in D$ you have seen are consistent with all hypotheses in V_t.
- all $h \in V_t$ classify $x_1, \ldots, x_{t-1}$ correctly, all hypotheses in $\mathcal{H} \setminus V_t$ make at least 1 classification mistake

V is used because of version spaces

```
1. $V_1 = \mathcal{H}$
2. For t = 1, 2. ...
    2.1 take $x_t$ from $D$
    2.2 predict majority $(\{h(x_t) \mid h \in V_t\})$
    2.3 get $y_t$ from D (i.e., $(x_t, y_t) \in D$)
    2.4 $V_{t+1} = \{h \in V_t \mid h(x_t) = y_t\}$
```

But, How About Complexity?

The halving learner makes the optimal number of mistakes

- which is good

But we may need to examine every $x \in D$

- for it may be the very last x we see that allows us to discard many members of V_t

In other words, the halving algorithm is

$$O\left(|D|\right)$$

Linear time is OK, but sublinear is better.

Sampling is one way to achieve this

Thresholds Again

To make our threshold example finite, we assume that for some (large) n

$$\theta \in \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}$$

Basically, we are searching for an element of that set

- and we know how to search fast

To search fast, you use a search tree

- the index in many DBMSs

The difference is that we

- build the index on the fly

We do that by maintaining an interval

- an interval containing the remaining possibilities for the threshold (that is, the halving algorithm)

Statistically halving this interval every time

- gives us a logarithmic algorithm

The Algorithm

- $l_1 := -\frac{0.5}{n}, r_1 = 1 + \frac{0.5}{n}$
- for $t = 1, 2, \ldots$
  - get $x_t \in [l_t, r_t] \cap \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}$
    - (i.e., pick again if you draw an non-viable threshold)
  - predict $sign((x_t - l_t) - (r_t - x_t))$
  - get y_t
    - if $y_t = 1$, $l_{t+1} := l_t, r_{t+1} := x_t - \frac{0.5}{n}$
    - if $y_t = -1$, $l_{t+1} := x_t + \frac{0.5}{n}, r_{t+1} := r_t$

Note, this algorithm is only *expected* to be efficient

- you could be getting x_t's at the edges of the interval all the time
  - hence reducing the interval width by $\frac{1}{n}$ only
- while, e.g., the threshold is exactly in the middle

Sampling

If we are going to be linear in the worst case, the problem is:

how big is linear?

That is, at how big a data set should we look

- until we are reasonably sure that we have almost the correct function?

In still other words.

how big a sample should we take to be reasonably sure we are reasonably correct?

The smaller the necessary sample is

- the less bad linearity (or even polynomial) will hurt

But, we rely on a sample, so we can be mistaken

- we want a guarantee that the probability of a big mistake is small

IID

(Note, $\mathcal{X} \sim \mathcal{D}$, $\mathcal{Y}$ computed using the (unknown) function f).
Our data set D is sampled from $\mathcal{D}$. More precisely, this means that we assume that

> *all the $x_i \in D$ have been sampled independently and identically distributed according to $\mathcal{D}$*

- when we sample x_i we do not take into account what we sampled in any of the previous (or future) rounds
- we always sample from $\mathcal{D}$

If our data set D has m members we can denote the iid assumption by stating that

$$D \sim \mathcal{D}^m$$

where $\mathcal{D}^m$ is the distribution over m-tuples induced by $\mathcal{D}$.

Loss as a Random Variable

According to the ERM$_{\mathcal{H}}$ learning rule we choose h_D such that

$$h_D \in \operatorname*{argmin}_{h \in \mathcal{H}} L_D(h)$$

Hence, there is randomness caused by

- sampling D and
- choosing h_D

Hence, the loss $L_{\mathcal{D},f}(h_D)$ is a random variable. A problem we are interested in is

- the *probability* to sample a data set for which $L_{\mathcal{D},f}(h_D)$ is not too large

usually, we denote

- the probability of getting a non-representative (bad) sample by δ
- and we call $(1 - \delta)$ the confidence (or confidence parameter) of our prediction

Accuracy

So, what is a bad sample?

- simply a sample that gives us a high loss

To formalise this we use the accuracy parameter ϵ:

1. a sample D is good if $L_{\mathcal{D},f}(h_D) \leq \epsilon$
2. a sample D is bad if $L_{\mathcal{D},f}(h_D) > \epsilon$

If we want to know how big a sample D should be, we are interested in

- an upperbound on the probability that a sample of size m (the size of D) is bad

That is, an upperbound on:

$$\mathcal{D}^m\left(\{D \mid L_{\mathcal{D},f}(h_D) > \epsilon\}\right)$$

Misleading Samples, Bad Hypotheses

Let $\mathcal{H}_B$ be the set of bad hypotheses:

$$\mathcal{H}_B = \{h \in \mathcal{H} \mid L_{\mathcal{D},f}(h) > \epsilon\}$$

A misleading sample teaches us a bad hypothesis:

$$M = \{D \mid \exists h \in \mathcal{H}_B : L_D(h) = 0\}$$

On sample D we discover h_D. Now note that because of the realizability assumption

$$L_D(h_D) = 0$$

So, $L_{\mathcal{D},f}(h_D) > \epsilon$ can only happen

- if there is a $h \in \mathcal{H}_B$ for which $L_D(h) = 0$

that is, if our sample is misleading. That is,

$$\{D \mid L_{\mathcal{D},f}(h_D) > \epsilon\} \subseteq M$$

a bound on the probability of getting a sample from M gives us a bound on learning a bad hypothesis!

Computing a Bound

Note that

$$M = \{D \mid \exists h \in \mathcal{H}_B : L_D(h) = 0\} = \bigcup_{h \in \mathcal{H}_B} \{D \mid L_D(h) = 0\}$$

Hence,

$$\begin{aligned}
\mathcal{D}^m\left(\{D \mid L_{\mathcal{D},f}(h_D) > \epsilon\}\right) &\leq \mathcal{D}^m(M) \\
&\leq \mathcal{D}^m\left(\bigcup_{h \in \mathcal{H}_B} \{D \mid L_D(h) = 0\}\right) \\
&\leq \sum_{h \in \mathcal{H}_B} \mathcal{D}^m\left(\{D \mid L_D(h) = 0\}\right)
\end{aligned}$$

To get a more manageable bound, we bound this sum further, by bounding each of the summands

Bounding the Sum

First, note that

$$\begin{aligned}\mathcal{D}^m(\{D \mid L_D(h) = 0\}) &= \mathcal{D}^m(\{D \mid \forall x_i \in D : h(x_i) = f(x_i)\}) \\ &= \prod_{i=1}^{m} \mathcal{D}(\{x_i : h(x_i) = f(x_i)\})\end{aligned}$$

Now, because $h \in \mathcal{H}_B$, we have that

$$\mathcal{D}(\{x_i : h(x_i) = y_i\}) = 1 - L_{\mathcal{D},f}(h) \leq 1 - \epsilon$$

Hence we have that

$$\mathcal{D}^m(\{D \mid L_D(h) = 0\}) \leq (1 - \epsilon)^m \leq e^{-\epsilon m}$$

(Recall that $1 - x \leq e^{-x}$).

Putting it all Together

Combining all our bounds we have shown that

$$\mathcal{D}^m\left(\{D \mid L_{\mathcal{D},f}(h_D) > \epsilon\}\right) \leq |\mathcal{H}_B| e^{-\epsilon m} \leq |\mathcal{H}| e^{-\epsilon m}$$

So what does that mean?

- it means that if we take a large enough sample (when m is large enough)
- the probability that we have a bad sample
  - the function we induce is rather bad (loss larger than ϵ)
- is small

That is, by choosing our sample size, we control how likely it is learn we learn a well-performing function. We'll formalize this on the next slide.

Theorem

Let $\mathcal{H}$ be a finite hypothesis space. Let $\delta \in (0,1)$, let $\epsilon > 0$ and let $m \in \mathbb{N}$ such that

$$m \geq \frac{\log\left(|\mathcal{H}|/\delta\right)}{\epsilon}$$

Then, for any labelling function f and distribution $\mathcal{D}$ for which the realizability assumption holds, with probability of at least $1 - \delta$ over the choice of an i.i.d. sample D of size m we have that for every ERM hypothesis h_D:

$$L_{\mathcal{D},f}(h_D) \leq \epsilon$$

Note that this theorem tells us that our simple threshold learning algorithm will in general perform well on a logarithmic sized sample.

A Theorem Becomes a Definition

The theorem tells us that we can

Probably Approximately Correct

learn a classifier from a finite set of hypotheses

- with a sample of logarithmic size

The crucial observation is that we can turn this theorem

- into a definition

A definition that tells us when we

- reasonably expect to learn well from a sample.

PAC Learning (Version 1)

A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property:

- for every $\epsilon, \delta \in (0,1)$
- for every distribution $\mathcal{D}$ over $\mathcal{X}$
- for every labelling function $f : \mathcal{X} \to \{0,1\}$

If the realizability assumption holds wrt $\mathcal{H}, \mathcal{D}, f$, then

- when running A on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$ labelled by f
- A returns a hypothesis $h \in \mathcal{H}$ such that with probability at least $1 - \delta$

$$L_{(\mathcal{D},f)}(h) \leq \epsilon$$

The Details in PAC

As before,

- the realizability assumption tells us that $\mathcal{H}$ contains a true hypothesis.
  - more precisely, it tells us that there exists a $h^* \in \mathcal{H}$ such that $L_{\mathcal{D},f}(h^*) = 0$
- ϵ tells us how far from this optimal results A will be, i.e., it is the *accuracy* – hence Approximately Correct
- δ, the *confidence* parameter, tells us how likely A meets the accuracy requirement – hence, Probably

The function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ determines how many i.i.d. samples are needed to guarantee a probably approximate correct hypothesis

- clearly, there are infinitely many such functions
- we take a minimal one
- it is known as the *sample complexity*

PAC Learning Reformulated

PAC learnability is a probabilistic statement, hence, we can write it as a probability:

$$\mathbb{P}_{D\sim\mathcal{D},|D|\geq m}(L_{(\mathcal{D},f)}(h_D) \leq \epsilon) \geq 1-\delta$$

Note that the probability is a statement over the hypothesis h_D we learn on all (large enough) samples.

If we spell out the loss in this statement, we get

$$\mathbb{P}_{D\sim\mathcal{D},|D|\geq m}(\mathbb{P}_{x\sim\mathcal{D}}(f(x) \neq h_D(x)) \leq \epsilon) \geq 1-\delta$$

in which the inner probability is a statement over a random $x \sim D$

Finite Hypothesis Sets

Our theorem of of a few slides back can now be restated in terms of PAC learning:

> *Every finite hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil$$

And, we even know an algorithm that does the trick: the halving algorithm.