Learning Conjunctions; Tools for Probabilistic Analysis

These notes are slightly edited from scribe notes in previous years.

1 Learning Over Logical Domains

In this part we cover one more concrete example of a learnable class. This time the input space is multi-dimensional and the concepts are defined logically.

Examples are elements of \( \{0, 1\}^n \) (i.e. a bit string of length n)

We have n Boolean variables \( x_1, \ldots, x_n \)

\( x_i \) takes values in \( \{0, 1\} \)

E.g.

<table>
<thead>
<tr>
<th>Example</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101011</td>
<td>( \oplus )</td>
</tr>
<tr>
<td>11011011</td>
<td>( \oplus )</td>
</tr>
<tr>
<td>01011011</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>00011100</td>
<td>( \ominus )</td>
</tr>
</tbody>
</table>

Hypothesis A conjunction of literals (i.e. a variable or a negated variable)

E.g. \( x_1 \land \neg x_3 \land x_7 \)

Observations For \( \oplus \) examples:

1. The set of literals that are true (=1) in all \( \oplus \) examples include all literals that are in the true concept \( c \).

2. If a literal is false in any \( \oplus \) example, then it is surely not in the true concept.

The Elimination Algorithm

1. Initially, \( hypothesis = x_1 \land \ldots \land x_n \land \neg x_1 \land \ldots \land \neg x_n \)

2. Take \( m = \frac{2n}{\epsilon} \ln \frac{2n}{\delta} \) I.I.D. examples
   - ignore \( \ominus \) examples
   - remove any literal which is false in any positive example

Theorem: The Elimination Algorithm PAC learns the class of conjunctions.

For any literal \( z \), we define \( P_z = Pr_D( z = 0 \text{ and true concept } c = 1) \)
Example: \( c = x_1 \land \neg x_3 \land x_7 \). \( P_{x_1} = 0 \), since if \( x_1 \) is false in the example, then \( c \) is false also. (This holds for any & all literals in the true concept \( c \))

**Definition:** A literal \( z \) is a bad literal if \( P_z > \frac{\epsilon}{2n} \)

The theorem follows from the next two lemmas.

**Lemma 1:** If we remove all bad literals (or the hypothesis does not include any), then \( \text{err}(\text{hyp}) < \epsilon \)

**Proof:** Notice that by part 1 of the Observations, there is only one type of error - that is error due to including unnecessary literal(s) in the hypothesis. We have:

\[
\text{err}(\text{hyp}) = P_r[\text{some unnecessary literal(s) in hyp = 0 AND c(x) = 1}]
\]

We use a simple Union Bound:

\[
\leq \sum_{z \in \text{hyp}} P_z \\
\leq 2n \frac{\epsilon}{2n} \\
= \epsilon
\]

**Lemma 2:** With probability at least \( 1 - \delta \) all bad literals are removed by the algorithm.

**Proof:** If a literal \( z \) is bad, then:

\[
Pr[z \text{ is not removed after 1 example}] = 1 - P_z \\
\leq 1 - \frac{\epsilon}{2n}
\]

\[
Pr[z \text{ is not removed after } m \text{ examples}] \leq (1 - \frac{\epsilon}{2n})^m \\
\leq e^{-\frac{\epsilon}{2n} \cdot \frac{2n}{\epsilon} \ln \frac{2n}{\delta}} \\
= \frac{\delta}{2n}
\]

\[
Pr[\text{there is a bad literal but we don’t remove it}] \leq 2n \frac{\delta}{2n} = \delta
\]

## 2 Tools for Probabilistic Analysis

This part introduces some inequalities, and illustrates abstract scenarios where they can be used. These building blocks will be useful in analyzing learning problems and models.

We have already used the following useful inequality

\[
1 + x < e^x \text{ for all } x \in \mathbb{R}
\]

\[
1 - x < e^{-x} \text{ for all } x \in \mathbb{R}
\]

2.1 Markov’s Inequality

If \( x \) is a positive random variable (\( x \geq 0 \)) then

\[
Pr[x \geq \alpha] \leq \frac{E[x]}{\alpha}
\]
e.g. $\alpha = KE[x]$ then $Pr[x \geq KE[x]] \leq \frac{1}{K}$

Proof

$$E[x] = \int_{-\infty}^{\infty} xp(x) \, dx$$

$$= \int_{0}^{\infty} xp(x) \, dx \quad \text{(because } x \geq 0\text{)}$$

$$= \int_{0}^{\alpha} xp(x) \, dx + \int_{\alpha}^{\infty} xp(x) \, dx$$

$$\geq 0 + \int_{\alpha}^{\infty} xp(x) \, dx \quad \text{(because } \int_{0}^{\alpha} xp(x) \, dx \geq 0\text{)}$$

$$\geq 0 + \alpha \int_{\alpha}^{\infty} p(x) \, dx$$

$$E[x] \geq \alpha Pr[x \geq \alpha]$$

which implies

$$Pr[x \geq \alpha] \leq \frac{E[x]}{\alpha}$$

### 2.2 Chernoff/Hoeffding Bound

Suppose $x_1, x_2, \ldots, x_n$ are bounded independent random variables with $x_i \in [a, b]$ and $E[x_i] = p$ for all $i$. Let $\hat{p} = \frac{1}{n} \sum_{i=1}^{n} x_i$. Hoeffding’s bound is as follows

$$Pr[|\hat{p} - p| > \epsilon] \leq 2e^{-2n\epsilon^2/[b-a]^2}$$

Some variants of this bound are given in chapter 9 of [KV] and appendix B of [SB]. These bounds are often referred to as “Chernoff Bounds” in the literature.

The bound is useful, for example, when we estimate the rate of Heads for a coin. Here $x_i$ is discrete with values 0 or 1, the $x_i$’s are IID, and $E[x_i] = p$ is the probability of getting Head. However, as we will see later in the course, the bound is useful much more broadly as it applies to any type of random variable as long as its range is bounded in a known interval $[a, b]$.

We will discuss the proof in the next lecture. Here we demonstrate the use of this inequality in various contexts.

**Example 0**

$p = 0.7$, $n = 200$

$$Pr[\hat{p} \notin [0.6, 0.8]] = Pr[|\hat{p} - p| > 0.1] \quad \text{(i.e. } \epsilon = 0.1\text{)}$$

$$\leq 2e^{-2(200)(\frac{1}{100})}$$

$$= 2e^{-4}$$

$$\approx \frac{1}{40}$$

If $n = 2000$ then $Pr[\hat{p} \notin [0.6, 0.8]] \approx \left(\frac{1}{30}\right)^{10}$
Example 1

Consider a coin with \( Pr(\text{Head}) = \frac{1}{4} \) and the potential of our observation \( \hat{p} \) to mislead us when flipping the coin 200 times.

According to Markov’s Inequality

\[
Pr[\hat{p} > \frac{1}{2}] \leq \frac{E[|x|]}{\frac{1}{2}}
\]

\[
= \frac{\frac{1}{4}}{\frac{1}{2}}
\]

\[
= \frac{1}{2}
\]

According to Hoeffding’s Bound

\[
Pr[\hat{p} > \frac{1}{2}] < Pr[|\hat{p} - p| > \frac{1}{4}]
\]

\[
\leq 2e^{-2(200)(\frac{1}{16})}
\]

\[
< 3 \times 10^{-11}
\]

We see that Hoeffding’s bound is much tighter.

Example 2

Consider two coins: coin1 with \( p_1 = \frac{1}{2} \) and coin2 with \( p_2 = \frac{1}{4} \). We do not know which coin is which. Design an experiment to pick coin1.

- **Algorithm**:
  Pick any coin, flip it \( n \) times and calculate \( \hat{p} \). If \( \hat{p} \geq \frac{3}{8} \) then this is coin1, otherwise it is coin2.

  We claim that with \( Pr \geq 1 - \delta \) the algorithm picks the coin correctly. There are two (symmetric) cases depending on which coin is being flipped by the algorithm. Assume that the algorithm is using coin1, then to make a mistake we have, \( p_1 = \frac{1}{2} \) and \( \hat{p}_1 \geq \frac{3}{8} \) therefore \( \alpha = \frac{1}{8} \).

  Applying Hoeffding’s inequality:

  \[
  Pr[|p_1 - \hat{p}_1| > \frac{1}{8}] < 2e^{-2n\frac{1}{16}}
  \]

  \[
  2e^{-\frac{n}{32}} < \delta
  \]

  \[
  n > 32 \ln \frac{2}{\delta}
  \]

  The case when the algorithm uses coin2 is similar. Thus, regardless of the coin being used by the algorithm, with probability \( \geq (1 - \delta) \) coin1 can be distinguished correctly.

Example 2b

Consider two coins: coin1 with \( p_1 = \frac{1}{2} \) and coin2 with \( p_2 < \frac{1}{2} \). We do not know which coin is which. Design an experiment to identify coin1.

- There exists no solution for this problem, as we cannot know \( p_2 \) in advance. Since there is no guaranteed gap between the two probabilities \( p_1 \) and \( p_2 \), they cannot be distinguished apart.
Example 3
Consider \( k \) coins with \( Pr[\text{Head}] = p_i \) for each coin \( i \). \( p_1 \leq \alpha \). We do not know anything about \( p_2, p_3, \ldots, p_k \). Design an experiment to pick a coin with \( p_i \leq 2\alpha \). Notice that we allow a gap between the guaranteed \( p_1 \leq \alpha \) and required \( \leq 2\alpha \).

- **Algorithm:**
  Flip each coin \( n \) times and calculate \( \hat{p}_i \) for each coin. Pick the coin with minimum \( \hat{p}_i \)

We want to guarantee that any coin with \( p_i > 2\alpha \) is not chosen. This is implied if all \( \hat{p}_i \) are within \( \frac{\alpha}{2} \) of \( p_i \):

\[
\hat{p}_1 < \alpha + \frac{\alpha}{2} \\
\hat{p}_i > 2\alpha - \frac{\alpha}{2}
\]

Thus under this condition coin 1 is preferred over any coin \( i \) with \( p_i > 2\alpha \). Therefore, if the algorithm fails then at least one of \( |p_i - \hat{p}_i| \) and \( |p_1 - \hat{p}_1| \) is \( > \frac{\alpha}{2} \).

We next show that when using \( n > \frac{2}{\alpha^2} \ln \left( \frac{2k}{\delta} \right) \), the algorithm which returns the minimum \( \hat{p} \) picks a good coin (which has \( p_i \leq 2\alpha \)) with probability \( \geq (1 - \delta) \).

For each coin,

\[
Pr[|p_i - \hat{p}_i| > \frac{\alpha}{2}] < 2e^{-2n\frac{\alpha^2}{4}} < \frac{\delta}{k}
\]

where the last inequality is implied if

\[
2e^{-2n\frac{\alpha^2}{4}} < \frac{\delta}{k} \\
-2n\frac{\alpha^2}{4} < \ln \left( \frac{\delta}{2k} \right) \\
n > \frac{2}{\alpha^2} \ln \left( \frac{2k}{\delta} \right)
\]

By union bound this implies,

\[
Pr[\exists i \text{ s.t. } |p_i - \hat{p}_i| > \frac{\alpha}{2}] \leq \frac{k\delta}{k} = \delta
\]

Example 4: Randomized Polynomial Time Algorithms
RP is the set of decision problems that have the following kind of algorithm,

- If input \( \in \text{No} \) \( \Rightarrow \) Always say No.
- If input \( \in \text{Yes} \) \( \Rightarrow \) with probability \( \geq \frac{3}{4} \) say Yes.

Can we boost the confidence of an RP algorithm? i.e. replace \( \frac{3}{4} \) with \( 1 - \delta \). The confidence of an RP algorithm can be boosted by running it \( k \) times and,

5
• if the algorithm says Yes at least once ⇒ Say yes.
• otherwise say No.

The number of trials needed to achieve the desired confidence can be computed as follows,

\[ Pr[\text{algorithm says No \(k\) times}] \leq \frac{1}{4} < \delta \]

\[ \frac{1}{2^k} < \delta \]

\[ k > \frac{1}{2} \log_2 \frac{1}{\delta} \]

Example 5: Bounded Probabilistic Polynomial Time Algorithms

BPP is the set of decision problems that have the following kind of algorithm,

• If input ∈ No ⇒ with probability \(\geq \frac{3}{4}\) say No.
• If input ∈ Yes ⇒ with probability \(\geq \frac{3}{4}\) say Yes.

Can we boost the confidence of BPP algorithms? i.e. replace \(\frac{3}{4}\) with \(1 - \delta\). The confidence of a BPP algorithm can be boosted by running it \(k\) times and calculating \(\hat{p} = \text{fraction of times the algorithm said Yes}\),

• if \(\hat{p} \geq \frac{1}{2}\) say Yes.
• otherwise say No.

The output would be wrong only when \(|p - \hat{p}| > \frac{1}{4}\), this can be used to calculate the number of trials needed to achieve the desired confidence,

\[ Pr[|p - \hat{p}| > \frac{1}{4}] < 2e^{-2n \frac{4}{16}} < \delta \]

\[ n \geq 8 \ln \frac{2}{\delta} \]