Learning Problem & Problem of Learning

M. R. Hasan
Readings

• Alpaydin: 2.1, 2.7
Learning Problem & Problem of Learning

• How do the Machine Learning models learn?

• Note learning in ML is **categorically different** from the learning in traditional programming.
Machine Learning

• In traditional programming, **experts write a set of rules** using which we **deduce the result** from a set of observations.

• It’s a **deductive process**.

However, in ML learning happens via **induction**.
What is Induction?

In induction we derive general rules from specific observations.

Let’s compare it with deduction.

Plato is a human being. All human beings are mortal.

Deductive Conclusion: Plato is mortal.

Plato is a human being. Plato is mortal.

Inductive Rule: All human beings are mortal.
Machine Learning

Predicting Future based on Past!
We will discuss the following two questions.

What is the inductive process of learning in ML?

Why is induction based learning hard?
Let’s look at a Machine Learning example to illustrate the **inductive learning process**.

It’s a classification problem (supervised learning).

We want to learn the class \((C)\) of a “family car” **from examples**.
We identified two features that separate a family car from other cars:
- Price
- Engine power

Then, we write an algorithm that learns the rule to differentiate a car (family or not) based on these features.
Machine Learning

• Let us denote price as the **first input attribute** \( x_1 \) (e.g., in U.S. dollars) and engine power as the **second attribute** \( x_2 \) (e.g., engine volume in cubic centimeters).

• Thus we represent each car using **two numeric values**:

\[
\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]

**Labels** denote the type of a car:

\[
\mathbf{r} = \begin{cases} 
1 & \text{if } \mathbf{x} \text{ is a positive example} \\
0 & \text{if } \mathbf{x} \text{ is a negative example}
\end{cases}
\]
Machine Learning

• We plot the training data in the two-dimensional \((x_1, x_2)\) space.

• Here each instance \(t\) is a data point at coordinates \((x_1^t, x_2^t)\) and its type (positive versus negative) is given by \(r^t\).

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]

\[
r = \begin{cases} 
1 & \text{if } x \text{ is a positive example} \\
0 & \text{if } x \text{ is a negative example}
\end{cases}
\]
Machine Learning

• Where do we begin?

• After analyzing the data, we formulate a **hypothesis**:

• A car to be a **family car**, its price and engine power should be in a **certain range**:

**Hypothesis:**

\[(p_1 \leq \text{price} \leq p_2) \text{ AND } (e_1 \leq \text{engine power} \leq e_2)\]

For suitable values of \(p_1, p_2, e_1, \text{ and } e_2\)
The hypothesis is based on the assumption that the class of family car (+ points) can be modeled as a rectangle $C$ in the price-engine power space.

This hypothesis represents our inductive bias. Without the inductive bias learning becomes impossible.
We could have used different hypotheses!

Hypothesis: Rectangle
Parameters: height & width

Hypothesis: Circle
Parameters: center & radius

Hypothesis: Ellipse
Parameters: widths in the two axes

Two axes has the same scale

Two axes need not have the same scale
We have defined the hypothesis class \( H \) (inductive bias) from which we believe \( C \) is drawn.

The hypothesis \( H \): possible set of rectangles.

\[(p_1 \leq \text{price} \leq p_2) \ \text{AND} \ (e_1 \leq \text{engine power} \leq e_2)\]

This hypothesis class (rectangle) defines our Model for Machine Learning.

We create a learning algorithm to find the particular hypothesis (best model), \( h \in H \), to approximate \( C \) as closely as possible.
Inductive Bias = Model = Hypothesis

**Algorithm** to find the *best* ML Model

- **Hypothesis: Rectangle**
  - Parameters: height & width

- **Hypothesis: Circle**
  - Parameters: center & radius

- **Hypothesis: Ellipse**
  - Parameters: widths in the two axes

Two axes need not have the same scale
After choosing the hypothesis class $H$, we need to find **which particular** $h \in H$ is equal, or closest, to $C$ (best model).

For this, we need to find **the four parameters** that define $h$.

In short, we do the following **two tasks in learning**:

- **Define a hypothesis** $H$ (e.g., solution is a rectangle)
- **Create an algorithm** to learn the parameters for choosing the best $h \in H$ that is equal, or closest, to $C$. 
There are two possibilities:

- Find the **most specific hypothesis** $S$, that is the **tightest rectangle**.
- Or find the **most general hypothesis** $G$, that is the **largest rectangle**.

Any $h \in H$ between $S$ and $G$ is a **valid hypothesis** and makes up the **version space**.
There is one problem with the hypothesis $S$.

It’s too obsessed with the training data!

It doesn’t allow us to predict the class of cars that we don’t have in the training examples (e.g., on the boundary).

Our hypothesis should be generalizable!
• Therefore, we choose the hypothesis with the largest margin, for best separation.

Define a hypothesis $H$ (e.g., solution is a rectangle)

Create an algorithm to learn the parameters for choosing an $h \in H$ that provides the largest margin.
Machine Learning

• However, we will **never find the solution** if our hypothesis class $H$ *doesn’t include* $C$.
• So, we need to make sure that $H$ is flexible enough, or has **enough “capacity,”** to learn $C$.

What if $H$ has **huge capacity**? Is it good?
Finally, we **summarize the inductive learning** in ML:

- Define a hypothesis $H$ (e.g., solution is a rectangle) that has **enough capacity**

- Create an algorithm to learn the parameters for choosing an $h \in H$ that provides the **largest margin**.
Finally, we **summarize the inductive learning** in ML:

- **Hypothesis:** Choose an \( h \in H \) that provides the largest margin.
- **Algorithm:** Enough **capacity**
Machine Learning

• We will now discuss the following two questions.

• What is the inductive process of learning in ML?

• Why is induction based learning hard?
Machine Learning

• But how difficult is it to learn a hypothesis (model)?
• Is it possible that we may never learn the hypothesis?
• It could happen if data is scarce.
• For example, only a few instances of family cars will make it difficult to conjure the “Rectangle” hypothesis.

Then, learning gets hard!
Let’s see why learning could be hard?

Consider a simple learning problem (binary classification).
- We want to design an algorithm that can learn a Boolean function from examples.
- In a Boolean function, all inputs (input features) and the output are binary.
- With $d$ inputs, the training set has at most $2^d$ examples.
For example, if we have just one input feature $x_1$ ($d = 1$), then the possible combinations (possible samples) of this feature that we can measure:

- $2^d = 2^1 = 2$

- For each combination (for each $x_1$), there are **two possible functions**: $2^{\text{no. of combinations}}$

<table>
<thead>
<tr>
<th>Feature</th>
<th>Boolean Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$h_1$ $h_2$ $h_3$ $h_4$</td>
</tr>
<tr>
<td>0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 1</td>
</tr>
</tbody>
</table>

Therefore, number of possible functions

$= 2^{\text{no. of combinations}}$

$= 2^{2^d}$

$= 2^2 = 4$

Each function ($h$) represents a possible output (0/1).
Machine Learning

- Since there are 4 possible functions or 4 possible mappings of the input (x) to output 0/1, our model or hypotheses consists of 4 functions.
- A learning algorithm will find the best model (hypothesis) from the observed samples.

Therefore, number of possible functions

\[
\text{number of possible functions} = 2^{\text{no. of combinations}} = 2^{2^d} = 2^2 = 4
\]
If we have just two input features $x_1$ and $x_2$ ($d = 2$), then the possible combinations (possible samples) of this feature that we can measure:

- $2^d = 2^2 = 4$
- For each combination ($x_1$ & $x_2$), the function can map to two possible outcomes: $2^{\text{no. of combinations}}$

Therefore, number of possible functions

$= 2^{\text{no. of combinations}} = 2^{2^d} = 2^4 = 16$

Model or Hypothesis: 16 functions
• We can generalize this observation.
• For d features there are $2^{2^d}$ possible Boolean functions.
• Here each row is a possible combination of the two features.
• Hence, each row represents a training sample.
• In this example, we have 4 possible training samples.

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$h_1$</td>
<td>$h_2$</td>
<td>$h_3$</td>
<td>$h_4$</td>
<td>$h_5$</td>
<td>$h_6$</td>
<td>$h_7$</td>
<td>$h_8$</td>
<td>$h_9$</td>
<td>$h_{10}$</td>
<td>$h_{11}$</td>
<td>$h_{12}$</td>
<td>$h_{13}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Column 3 to the last column represent the output for input combinations of $x_1$ and $x_2$.

There are 16 possible ways to produce output values.

The real world used one way (one function) to create output.

To find that one unique function that real world used, we need to have the set of all possible functions for two features.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>$h_4$</th>
<th>$h_5$</th>
<th>$h_6$</th>
<th>$h_7$</th>
<th>$h_8$</th>
<th>$h_9$</th>
<th>$h_{10}$</th>
<th>$h_{11}$</th>
<th>$h_{12}$</th>
<th>$h_{13}$</th>
<th>$h_{14}$</th>
<th>$h_{15}$</th>
<th>$h_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
• This example tells us **how many training examples** we need to create the set of all possible functions (16).

• We see that to find a **unique function** in case of 2 features, we need at least **4 training examples of all combinations**.
• What if we have **less training data? Or missing data?**
• Then, we will **never know the possible functions** for those instances that we could not get.
• As a consequence, if the unique function resides in that never-known set of possible functions, we **will never find the unique mapping** or function.
• **How many training examples** do we need to create a set of all possible functions (hypotheses or models)?

• In the case of a Boolean function, to end up with a single hypothesis (that the real world used) we need to see all $2^d$ training examples.
• If the training set we are given contains only a small subset of all possible instances, then we might not find the unique solution.
• Let’s say that we have seen only N examples.
• For a d-dimensional input feature, number of possible combinations = 2^d
• The number of combinations of the features we did not see = 2^d – N
• Hence, after seeing N example cases, the number possible functions we could be guess (hypothesize) = 2^{2^d-N}
To get all possible functions ($2^{2^d}$ possible functions) we need $2^d$ training examples (d-dimensional attributes).

If d is large, then for $2^d$ possible combinations we will need lot of data!
Machine Learning

• If we don’t have this amount of data \(2^d\), then our problem is **ill-posed**.

• Because the **data by itself is not sufficient** to find a unique solution.

• Hence, inductive learning gets **hard**.
Machine Learning

• Unfortunately, often times ML problems are ill-posed!
• So, what do we do?

• We should make some extra assumptions to have a unique solution with the data we have.

• We make inductive bias.

• One way we introduce inductive bias is when we assume a hypothesis class $H$. 
For example, in the “family car” learning problem, there are infinitely many ways of separating the positive examples from the negative examples.

We used two inductive biases.

Separator should be a rectangle.

The rectangle should have the largest margin.
Machine Learning

• But we know that each hypothesis class has a certain capacity.
• Therefore, it can learn only certain functions.
• The class of functions that can be learned can be extended by using a hypothesis class with larger capacity, containing more complex hypotheses.
Machine Learning

- For example, the hypothesis class that is a union of two rectangles has higher capacity, but its hypotheses are more complex.

The question now is to decide where to stop.
Machine Learning

• **Choosing between possible H (possible models)** or choosing the right inductive bias is called **model selection**.

• How do we select the best model?
• Split your data into training and test instances.
• Train your model based on possible H.

• Then, see which hypothesis **generalizes** well on the test data!
Machine Learning

• For best generalization, we should match the complexity of the hypothesis class $H$ with the complexity of the function underlying the data.

• If $H$ is less complex than the function, we have underfitting.
Machine Learning

• If there is noise, an overcomplex hypothesis may learn not only the underlying function but also the noise in the data and may make a bad fit.

• This is called overfitting.

So, we need to find a trade-off!
In summary, there is a **trade-off between three factors**:

- The **complexity** (or capacity) of the hypothesis we fit to data.
- The **amount of training data**.
- The **generalization error** on new examples.

You can **never make the optimal trade-off** because of less data!
Machine Learning

• So, bad news:
• There is no universally best model!
• This is sometimes called the no free lunch theorem.
• The reason for this is that a set of assumptions that works well in one domain may work poorly in another.
• And more importantly:

No assumption = No learning
What’s the Recipe of **Successful Learning**?

Data (a lot)  Models or Hypotheses (enough)  Algorithms = Successful Learning