Lecture 1: What does it mean to learn?

CSE 427: Machine Learning

Md Zahidul Hasan

Lecturer, Computer Science
BRAC University

Spring 2023

Learning From Experience

CGPA	GRE	Publications	Admission Offers
3.5	321	3	VT, UC Riverside
4.0	332	4	MIT, UCB, CMU
3.7	321	0	NYU, UMass
3.89	324	1	Purdue, UMass
3.75	329	2	NYU, Yale
3.3	330	0	WSU
3.23	311	1	WSU, FIU

There are various groups and threads on social media that help people with undergraduate/graduate admissions. People post their profiles and the list of universities they got admission offers from. Future candidates can learn from the experiences shared therein and can decide which universities to aim for.

Applications

1. speech recognition, speech synthesis, speaker verification
2. document classification, spam detection
3. image recognition, object detection, pet and face detection
4. machine translation, parsing, parts of speech tagging
5. recommendation systems, stock market predictions
6. generative art, music synthesis, AI story generator
7. computational biology: drug discovery, protein function prediction
8. medical diagnosis: skin cancer detection, eye disease detection
9. games: chess, go, checker
10. self-driving cars, traffic prediction
11. online fraud detection, anomaly detection

Elements of Learning

Instances: Examples or instances are the rows of a dataset. For example, in the previous example, a student's profile is an instance.
Features: Features or attributes are the columns of a dataset except the last one. For example, a student's profile consists of CGPA, GRE, number of publications, TOEFL, etc.
Labels: The last column of the dataset is called the label set. In the given example, the label set contains the list of universities that offered admission. The goal of supervised learning is to learn this column.
Feature Vector: A row can be expressed as a single vector excluding the label. In the above scenario there are 7 students, i.e. $x_1 = (3.5, 321, 3), \cdots, x_7 = (3.23, 311, 1)$. And the labels can be expressed as $y_1, y_2, \cdots, y_7$.
Training Data: Set of instances used to train a learning algorithm. If the training sample has m instances, it can be expressed as $S = ((x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m))$.

Elements of Learning contd.

Hyperparameters: Free variables that are independent of the learning algorithm.
Validation Data: Examples used to tune the hyperparameters of the learning algorithm.
Testing Data: Examples used to evaluate the performance of a learning algorithm. The learning algorithm doesn't have access to the testing data during the learning process.
Domain: The domain is a pool of instances from where we build the dataset. For example, CGPA can be any real number from 2 to 4, the GRE score is an integer between 260 and 340, the number of publications can be any non-negative integer. Hence a domain for the student profiles can be $\mathcal{D} = \mathcal{A} \times \mathcal{B} \times \mathcal{C}$ where $\mathcal{A}$ is the domain for CGPA, $\mathcal{B}$ is the domain for GRE, $\mathcal{C}$ is the domain for number of publications.
Concept: A concept is what a learning algorithm is trying to learn. Most of the time it's a function from $\mathcal{D}$ to another set.

Elements of Learning contd.

For example, in the case of learning universities, $f(3.5, 321, 3) = \{VT, UC Riverside\}$

Hypothesis: A hypothesis is our assumption about the concept. For example, our learning algorithm thinks that with a 3.5 CGPA, 321 GRE score, and 3 publications, a student can get offers from universities like Virginia Tech, Indiana University, and John Hopkins University. Our hypothesis as well could be a function from $\mathcal{D}$ to a vector of universities like the following $h(3.5, 321, 3) = \{VT, IU, JHU\}$.

Loss Function: During testing, we need to determine the performance of the learning algorithm. If our learning algorithm is a binary classifier that classifies an email into two categories {SPAM, non-SPAM}, then if a testing instance is correctly classified, the loss is 0. Otherwise, it's 1. Let's say we are trying to estimate the rent of a house given a feature vector x. $f(x)$ is the actual rent and $h(x)$ is our estimate. The loss or error could be defined as $|f(x) - h(x)|$ or $(f(x) - h(x))^2$.

Elements of Learning contd.

Empirical Error: Given a concept f, and a training sample $S = (x_1, x_2, \cdots, x_m)$ of size m, then the empirical error of a hypothesis h is defined by:

$$R_S(h) = \frac{\sum_{i=1}^{m} 1_{f(x_i)=h(x_i)}}{m}$$

Generalization Error: The training samples are only a partial view of the data. The true error or generalization error is the error of our learning algorithm if we could test it against all the data instances in the domain. Any instance x is sampled from the domain $\mathcal{D}$. It's defined by:

$$R(h) = \mathbb{E}_{x \sim \mathcal{D}}[1_{f(x)=h(x)}]$$

Machine Learning Tasks

Some of the most common machine learning tasks are the following:

Classification: This involves assigning a label to each data instance. If we are classifying documents based on their category, then we need to assign each document a label such as $\{$sports, politics, entertainment, science, $\cdots\}$. If we are trying to classify a document based on whether it gives off a positive vibe or not, then the labels should be $\{$POSITIVE, NEGATIVE$\}$.

Classification with missing inputs: If we want to diagnose a patient's disease based on their history, bodily features and test results, it could be possible that some people can't afford some expensive medical tests. In that case, we end up with features that have no value for a particular patient. Even in those cases, we need to be able to diagnose their illnesses. In the usual learning scenario, we learn one function to classify the testing samples. In this case, we might need to learn 2^n different functions for the classification where n is the total number of features.

Machine Learning Tasks contd.

Regression: Unlike classification, we are not assigning a label to each instance here. We want to estimate a value which is commonly a real number. For example, we might be trying to estimate the rent of an apartment based on the total square feet, number of rooms, proximity to the city, utility facilities etc.
Transcription: Transcription converts unstructured data into textual forms. For example, optical character recognition, speech to text etc.
Ranking: Given a search prompt, a search engine generates a list of webpages which need to be sorted for users' convenience.
Clustering: Clustering is applied to identify groups of similar entities such as community detection in social media or streaming services.
Manifold learning: Manifold learning is the task of reducing the feature space intelligently so that learning can occur more efficiently.

Machine Learning Tasks contd.

Anomaly detection: We can use machine learning to detect unusual behaviors from a credit card owner to identify theft.
Synthesis and sampling: With synthesis and sampling, we can generate new music or new artwork by learning from the past works of an artist.
Imputation of missing values: If the value of some features are lacking, we can use our wisdom gained from experience to impute the missing values.
Denoising: Suppose, a clean example x is observed to be a corrupted $\bar{x}$ due to some noisy process. From experience, given $\bar{x}$, we can predict the original clean x.
Probability mass/density estimation: In this scenario, we are just trying to learn $p(x)$ for an instance x.

Learning Scenarios

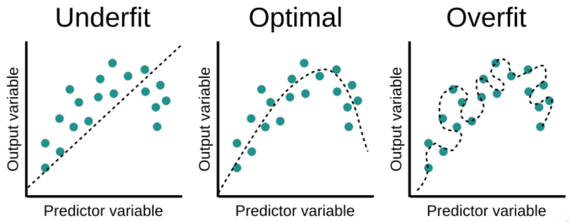
Supervised learning: In this case, we are always given either a set of labels or a value or some ranking. Classification, ranking, regression, etc are supervised tasks.
Unsupervised learning: In unsupervised learning, there is no column for labels. Clustering and dimensionality reduction are some of the unsupervised learning techniques.
Semi-supervised learning: Semi-supervised learning is ideal when it's expensive to get labeled data. In this case, we can learn from the labels that are given, and utilize similarity measures as in unsupervised learning to make predictions.
Transductive inference: Just like the semi-supervised scenario, we get some labeled data and some unlabeled data. But in the case of induction, we learn from the training set to make general predictions. We make no assumption about the test data. But in this case, the test data are the unlabeled instances only. The goal is to minimize error in the test samples only.

Learning Scenarios contd.

Offline learning: In the offline scenario, all the learning happens before testing. It learns from the training set before it is tested.
Online learning: In online learning, the training and testing phases are intertwined. Here, the algorithm makes a prediction when it encounters a data instance. Then it learns from the mistake.
Reinforcement learning: In reinforcement learning, the environment doesn't provide an explicit label set or any kind of supervision. But the agent receives a reward or punishment for any action. The agent's goal is to maximize reward in the long run.
Active learning: In this scenario, the algorithm (student) carefully chooses which data points to include during learning and then queries a teacher about its label. The goal is to get results comparable to the supervised learning scenario with fewer data points because sometimes getting labels can be expensive.

The Risk of Empirical Risk Minimization

Since the testing data isn't available to the learner during training, the only tool to measure performance is the empirical error. But the training data could give us a very misleading picture of the whole domain. Thus the learner could be misled trying to wholly adapt to the training samples. The learning paradigm that tries to minimize empirical risk is called the ERM paradigm. In the image below, we can see that the rightmost predictor most accurately describes the training data. But the true distribution of the data might not follow this peculiar curve. So, to be safe, the second predictor is our best option.



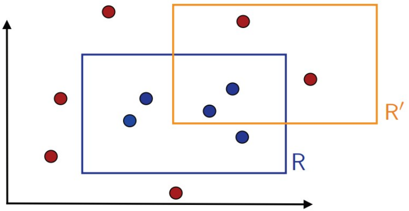
ERM with Inductive Bias

In the previous slide, we saw that the ERM approach could be very misleading. If we put no restriction on the set of possible hypotheses, we could end up with the most bizarre hypothesis that's just perfect for the training data but fails miserably during testing. So, only if we could find a good class of non-bizarre hypotheses called H, we could just choose one that minimizes the empirical error over the training set S. And that hypothesis is called ERM_H.

$$ERM_H(S) \in \arg\min_{h \in H} R_S(h)$$

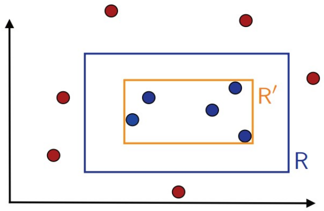
This choice of hypotheses introduces a bias towards a particular set of predictors. This is called inductive bias. ERM with inductive bias is one of the remedies for overfitting. The more restricted a hypothesis class is, the more biased our learner is. The less restricted a hypothesis class is, the more likely it is that we will end up with an overfitting hypothesis. In this course, we will learn how to find a good hypothesis set.

ERM Application: Axix-aligned Rectangles



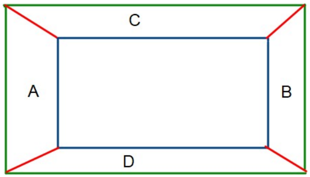
Suppose, a military school is looking for applicants that have their weight and height within a specific range. Let's say an accepted candidate's height is x and their weight is y. So, for some x_1, x_2, y_1, y_2, $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$. But to an outsider, x_1, x_2, y_1, y_2 are hidden. They can only see which applicants were taken and which were discarded. In the above diagram, the blue rectangle is the true decision boundary. Can we find an algorithm so that given any training sample, it can find a hypothesis that can perform arbitrarily well depending on the sample size with a very high probability?

Axis-aligned Rectangles contd.



An obvious choice of H would be the set of all axis-aligned rectangles. If our learning algorithm chooses the rectangle h with the minimum area that contains all the blue points, then definitely $R_S(h) = 0$. If we test this rectangle with a test set, the only way it can make mistakes is by classifying the points that fall outside $h = R'$ but inside R as red. We want to find an upper bound for the likelihood of this kind of error. We want that $Pr[R(h) > \epsilon] \leq \delta$ for arbitrarily small but positive ϵ and δ.

Axis-aligned Rectangles contd.



Let's connect the corners of the outer and inner rectangles with red lines. Let's say, the probability that a point falls into region A is a. Similarly define b, c, d. So, the total probability of making a mistake is $a + b + c + d$.

$Pr[R(h) > \epsilon] = Pr[a + b + c + d > \epsilon]$
$\leq Pr[a \geq \frac{\epsilon}{4} \vee a \geq \frac{\epsilon}{4} \vee a \geq \frac{\epsilon}{4} \vee a \geq \frac{\epsilon}{4}]$
$\leq Pr[a \geq \frac{\epsilon}{4}] + Pr[b \geq \frac{\epsilon}{4}] + Pr[c \geq \frac{\epsilon}{4}] + Pr[d \geq \frac{\epsilon}{4}]$ (union bound)
$\leq 4Pr[a \geq \frac{\epsilon}{4}] \leq 4(1 - \frac{\epsilon}{4})^m \leq 4e^{\frac{-m\epsilon}{4}} = \delta$ ($e^x \geq 1 + x, \quad \forall x \in \mathbb{R}$)
Solving for m gives, $m \geq \frac{4}{\epsilon} \log \frac{4}{\delta}$. So, if we want 99% accuracy with 99% confidence, then we only need $m = \frac{4}{0.01} \log \frac{4}{0.01} = 1040$ points.

ERM with Finite Hypothesis Classes

The most obvious way to restrict a hypothesis class is to put restrictions on its size. For now, we focus on hypothesis classes that are finite in size.

Consistency Assumption

A hypothesis class H is said to be consistent if there exists a hypothesis h so that $R(h) = 0$. This is also called the realizability assumption. Obviously $R_S(h) = 0$ for any sample S.

I.I.D. Assumption

We will also assume that the instances of the dataset are independent of each other and have an identical probability distribution.

Learning Guarantees for Consistent and Finite Hypothesis Classes

Learning Guarantee: Finite and Consistent Scenario

Let H be a finite and consistent hypothesis set that contains the target concept c. Let A be an algorithm such that for any sample S, it returns a hypothesis $h \in H$ so that $R_S(h) = 0$. Then we can say that for any positive $0 < \epsilon, \delta < 1$, there exists a minimal sample size $m \geq \frac{1}{\epsilon}(\log |H| + \log \frac{1}{\delta})$ so that if we take any sample of size m, then the algorithm will return such a hypothesis h so that $Pr[R(h) \leq \epsilon]$ can hold with at least $1 - \delta$ probability.

Proof: $Pr[\exists h \in H : R_S(h) = 0 \wedge R(h) > \epsilon]$
$\leq \sum_{h \in H} Pr[R_S(h) = 0 \wedge R(h) > \epsilon]$ (Union bound)
$\leq \sum_{h \in H} Pr[R_S(h) = 0 | R(h) > \epsilon]$ $(P(A \wedge B) \leq P(A|B))$
$\leq |H|(1-\epsilon)^m \leq |H|e^{-\epsilon m} = \delta.$ $(e^x \geq 1 + x, \quad \forall x \in \mathbb{R})$
Solving for m yields the desired bound.