Notes from Lectures

Inductive learning

* Given a training set of examples of form (x, f(x)), can we return a function h that approximates f.
* X is the input and f(x) is the output and h are the hypothesis
* The reality is that we will never know what that function f is, so we search for a hypothesis that approximates f. f is the function we are trying to discover, and it is not something we have as a ground truth.
* Difference between deductive and inductive learning
  + Deductive learning is where you are given a rule (a concept) and you are also provided with examples. Afterwards, you would be asked to apply this rule to these examples.
  + Inductive learning is the opposite as you are given some examples and you need to figure out the concept (or rule or function) that explains these examples. THIS IS WHAT MACHINE LEARNING DOES.

Supervised Learning

* 2 types of problems
  + Classification- the range (output space) of F will be categorical.
  + Regression- the output space of F is continuous.
  + These differences are only based on the output type.
* Nature of domain (input space) of f does not matter.

Classification

Problem: Will you enjoy an outdoor sport based on the weather?

Training set:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sky | Humidity | Wind | Water | Forecast | EnjoySport |
| Sunny | Normal | Strong | Warm | Same | Yes |
| Sunny | High | Strong | Warm | Same | Yes |
| Sunny | High | Strong | Warm | Change | No |
| Sunny | High | Strong | Cool | Change | yes |
| X | | | | | F(x) |

Possible Hypotheses:

* H1- S= Sunny -> enjoySport = yes
* H2: Wa = cool or F = same -> enjoySport = yes

Note: We do not know F(x)

Examples (whether they are classification or regression examples)

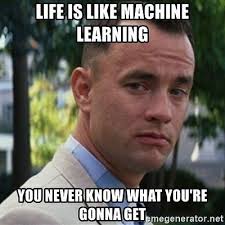
|  |  |  |  |
| --- | --- | --- | --- |
|  | Domain | Range | Classification or Regression |
| Spam Detection | Text is the input. This could be represented as a one-hot vector or as vector embeddings as we are working with words. Essentially, a vector of words | Spam or Legit | Classification as we only have two categories |
| Stock Prediction | Time series of prices | One possibility: up or down  Other possibility: positive real numbers | Classification in first one, Regression in second one |
| Speech Recognition | Audio Signal | Words | The intermediate representations are continuous as you represent words in continuous values but the output is discrete. So, its classification |
| Digit Recognition | Images | Digits in the range of 0-9 | Classification problem |
| Housing evaluation (Estimate of price of a house) | Features of the house like Bedrooms, Sq Foot space and etc. These features can be discrete or continuous | Price of the house (Positive real numbers) | Regression Problem |
| Weather Prediction | Sensor data including satellite imagery | {Rain, Sun}, and temperature | Classification for rain and sun and regression for temperature. |

How large should the output space should be for it to be a classification or a regression problem?

* The distinction lies in the orderness and magnitude of the elements. For example- when we consider the stock prediction example, we see that different values such as 53,54, and 55 have an order and that their relative magnitude matters as bigger numbers might portray a different scenario than that of lower numbers. However, in tasks like speech recognition where the continuous values of words doesn’t matter as all words are treated the same, we are able to recognize this task as more of a categorical labeling task.

Hypothesis Space

* Hypothesis Space H
  + Set of all hypotheses h that the learner may consider
  + Learning is a search through hypothesis space
* Objective: find h that minimizes
  + Misclassification
  + Or more generally some error function with respect to the training examples. These examples come from an underlying distribution and we can **assume** that the test data points also come from this distribution.
  + This can be thought of as an optimization problem where we have some objective and we have to minimize this objective. However, this might prompt you to say, “So, Machine Learning is simply an Optimization Problem and this course in ML should really be a course in Optimization?”. However, the main goal of Machine learning is generalizability. This means that when we minimize this objective and find a suitable, optimized hypothesis, we have to test it by seeing how it generalizes on unseen examples. This differentiates ML from Optimization. However, hardcore optimizationists would say that perhaps “ML is just an ill-defined problem as judging the performance of an algorithm on unseen data points is illogical”. However, this is where Machine Learning gets FUN!!!!



Because here we have to combine optimization with statistics to reach a suitable hypothesis which can perform well on unseen data.

* What about unseen examples? (This is often called “test data” which is used for testing the generalizability of our algorithms).

Generalization

* A good hypothesis will generalize well
  + i.e. predict unseen examples correctly
* Usually…
  + Any hypothesis h found to approximate the target function f well over a sufficiently large set of training examples will also approximate the target function well over any unobserved examples
* In other words, suppose we possess a large set of data points about wind speed, temperature, power input, and rotor speed and our task is to predict a target temperature of the rotor then our hypothesis would probably involve a combination of these 4 features (input values) and the hypothesis space would be all of the polynomials (if that is how our hypothesis is presented) from degree 0 to 4. Then, in order to determine whether we have a sufficiently good hypothesis that is approximating the underlying function f, that helped to produce this data, we would have to first check its performance on the training data set. If our hypothesis performs well over that dataset (which we are assuming is large) then we would be able to confidently state that our hypothesis represents the underlying distribution well. This would mean that it would perform well on future examples as well.

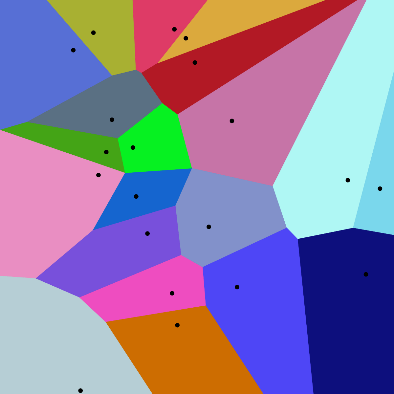
The Grand Idea behind this is….

We **assume** that our data comes from a distribution and through machine learning we take a suitable hypothesis and tune it to make it come close to this hidden, underlying distribution function.

Inductive Learning

* Goal: find an h that agrees with f on training set
  + h is consistent if it agrees with f on all examples
* CAVEAT: finding a consistent hypothesis is not always possible and not desirable as well
  + Insufficient hypothesis space. (NOT ALWAYS POSSIBLE CASE)
    - E.g.- it is not always possible to learn exactly f(x) = ax + b + xsin(x) when H = space of polynomials of finite degree. This function is not part of the space because the sin function (when represented with the taylor series) will be represented by infinite terms whose degrees also grow infinitely large. This might be a bit contrived and one might ask, “what about in practice?”. Example- in tasks like object detection, we might often search for a very large and complex function because the functions that we ourselves use to see pictures are also believed to be extremely complex. This might lead us to the space of neural networks but often practitioners find that even the space of neural nets is insufficient and we suffer from having an insufficient hypothesis space.
  + Noisy Data (
    - E.g.- in weather prediction, identical conditions may lead to rainy and sunny days. Sensor noise might produce noisy data which would make finding a consistent hypothesis a tough task. This would mean that we are unable to find a function that is able to explain our data and so we might declare this problem to be “unrealizable”. A quick fix solution might prompt us to expand our hypothesis space to be as large as possible and have a higher chance of making the problem realizable but this is not necessarily good.
* A learning problem is realizable if the hypothesis space contains the true function otherwise it is unrealizable.
  + Difficult to determine whether a learning problem is realizable since the true function is not known.
* It is possible to use a very large hypothesis space
  + Example: H = class of all Turing machines
* But there is a tradeoff between expressiveness of a hypothesis class and complexity of finding a good hypothesis. This means that when we expand our hypothesis class (make it more expressive) then there is a higher chance of finding the right hypothesis but the time complexity, space complexity, and data complexity (the data that we require to find such a hypothesis) increase as well.

Nearest Neighbor Classification

* Classification function
  + h(x(input)) = y(output)x\* (where yx\* is the label associated with the nearest neighbor)
  + x\* = argminx’ d(x,x’)
* Distance Measures : d(x,x’)
  + L1: Euclidean Distance Measure
  + L2 Distance
  + …. LP Distance. But how does all of this happen in terms of classification?
* Voronoi Diagrams
  + Partition implied by nearest neighbor fn h
    - Assuming Euclidean Distance
  + Pictured above is a Voronoi diagram of a dataset and if we run a classifier for a test example, then we would first compute the Euclidean Distance from that example and then place it in the weighted space of that closest example. This is a 1-NN classifier. Notice that the boundaries are straight lines because we are partitioning the space near two examples in half and this produces straight lines.
* Nearest Neighbors classifier works well but tends to be unstable due to noise in data which could be due to noise in measuring devices or something else. So, how do we make NN classifier more robust?
* Idea: Assign most frequent label among k-nearest neighbors
  + Let knn(x) be the k-nearest neighbors of x according to distance d
  + Label: yx ← mode ({yx’|x’ ∈ knn(x)}). The most frequent label according to the labels in K NN that are being considered.
  + One might immediately think what would be the effect of different values of K and which value would be optimal (would it be 3,5,7,9 or something else)?
  + Well, if our value of K is one then we get a partitioning that is extremely similar to that of the Voronoi Diagram. If we use K=3 then we get a much more suitable partition. But with a K= n partition we get a less robust partition as we would classify everything in the same way (by choosing the most frequent class in the dataset).
  + The Tradeoff is between performance and number of neighbors which is quite INTUITIVE.

Performance of a Learning Algorithm

* A learning algorithm is good if it produces a hypothesis that does a good job of predicting classifications of unseen examples.
* We usually verify our performance with a test set
  + The usual process of ML projects is to start by collecting a large set of data points
  + Divide those data points into two **DISJOINT** sets: train and test set.
  + Learn hypothesis h with training set
  + Measure performance of correctly classified examples by h in the test set.
* Effect of K
  + Best K depends on type of problem and the amount of training data.

(INSERT PICTURE OF DIAGRAM)

Underfitting and Overfitting

* Underfitting occurs when an algorithm find a hypothesis h with training accuracy that is lower than the future accuracy of some other hypothesis h’. Idea is that we are not finding the best possible hypothesis.
* Amount of underfitting of h:
  + Max {0, maxh’ futureAccuracy(h’) – trainAccuracy(h)} = {0, maxh’ testAccuracy(h’) – trainAccuracy(h)}
* Common cause:
  + Classifier is not expressive enough which means that our hypothesis space is not expressive enough. Essentially, it is like looking for F(x) = sinx in a space of finite polynomials.
* Overfitting occurs when an algorithm finds a hypothesis h with higher training accuracy than its future accuracy. The hypothesis fits the noise in the data which is bad for future predictability.
* Amount of overfitting of h:
  + max{ 0, trainAccuracy(h) – futureAccuracy(h)} = max { 0, trainAccuracy(h) – testAccuracy(h)}. FutureAccuracy is the true underlying accuracy of the hypothesis.
* Common causes:
  + Classifier is too expressive.
  + Noisy Data
  + Lack of Data

Choosing K

* How should we choose K?
  + Ideally: select K with highest future accuracy
  + Alternative: select K with highest test accuracy
* Problem: since we are choosing K based on test set, the test set effectively becomes part of the training set when optimizing K. Hence, we cannot trust anymore the test set accuracy to be representative of future accuracy.
* Solution: split data into training, validation and test sets
  + Training set: compute nearest neighbor
  + Validation set: optimize hyperparameters such as K. The validation (process of deciding if numerical results quantifying hypothesized relationships between variables are acceptable as descriptions of data) process essentially becomes a way for us to optimize our hyperparameters. In other ML algorithms, we will encounter many hyperparameters, so this split will become really important.
  + Test set: measure performance

Choosing K based on Validation set

Let k be the number of neighbors

For k =1 to max # of neighbors

Hk 🡨 train(k, trainingData)

accuracy­k 🡨 test(h­k, validationData)

k\* 🡨 argmaxk accuracyk

h 🡨 train(k\*, trainingData U validationData)

accuracy 🡨 test(h, testData)

return k\*, h, accuracy

Robust Validation

* How can we ensure that validation accuracy is representative of future accuracy?
  + Validation accuracy becomes more reliable as we increase the size of the validation set
  + However, this reduces the amount of data left for training
* Popular solution: Cross Validation!!!



* Repeatedly split training data into two parts, one for training and one for validation. Report the average validation accuracy.
* K-fold Cross Validation: split training data in k equal size subsets. Run k experiments, each time validating on one subset and training on the remaining subsets. Compute the average validation accuracy of the k experiments.
* Picture:
* If we perform a 4-cross fold validation. Then we would divide up our data into 4 segments (A,B,C,D). In first iteration, we would train on A,B,C and validate on D. Then we would train on A,B,D and validate on C. Then we would train on A,C,D and validate on B. Then we would train on B,C,D and validate on A. There are essentially K chose K-1. However, as we increase K, we could assign different ratios for training and validating. So, if we are performing a 32-CV then we could do 28 folds for training and 4 for validating. An important to thing to note is that we never optimize our hyperparameters through the test set. The main purpose of CV is so that we are able to optimize our hyperparameters on the validation split of the training data. Another thing to note is that, usually, we train multiple times on our training set (EPOCHS) and set our hyperparameters accordingly.
* Cross Fold Validation is more robust in giving us an assessment of our hyperparameters in comparison to just Splitting our data in a train and validation set.

Selecting the number of Neighbors by CV

Let k be the number of neighbors

Let k’ be the number of trainingData splits

For k = 1 to max # of neighbors

For i = 1 to k’ do (where i indexes trainingData splits)

hki 🡨 train (k, trainingData1…i-1, i+1, …., k’)

accuracyki 🡨 test (hki,trainingDatai)

accuracyk 🡨 average ({accuracyki} for all i)

k\* 🡨 argmaxk­ accuracyk

h 🡨 train (k\*, trainingData1…k’)

accuracy 🡨 test (h, testData)

return k\*, h, accuracy

Weighted K-Nearest Neighbour

* We can often improve KNN by weighting each neighbor based on some distance measure
  + w(x,x’) inversely proportionate to (1/distance(x,x’))
* Label
  + Yx 🡨 argmaxy ∑{x’ | x’ member of knn(x) /\ y = yx’} w(x,x’)
  + Where knn(x) is the set of K nearest neighbors of x

KNN for Regression

* Let yx be a real value instead of a categorical label
* KNN regression