Chapter 2: Limits of Learning

2.1 Data Generating Distributions

Bayes Optimal Classifier: it is a function that basically finds the optimal solution, for example whether the player will enjoy the game or not considering the whether is sunny and hot. In this case, we basically find the probability that the player will enjoy given that the weather is hot and sunny and probability that the player will not enjoy considering the same conditions.

After that, we just find the probability that the player will not enjoy and the probability that the player will enjoy, whichever is greater is the correct answer and that is basically how the Bayes classifier works.

Inductive Bias: in terms of data that narrow down the relavant concept, what type of solutions are we more likely to prefer? For example, a test set one shows a picture of birds and the second test set shows a picture of non-birds. But some people might see it that it is a bird vs non-bird and that it is fly vs non-fly. In other words, inductive bias is the assumptions that people make to generalize from the training data to unseen instances.

An example of an inductive bias is a **shallow decision tree** where there is a predefined amount of depths d in the tree and beyond that, we just make guesses on what a test data will be based on those predefined features we got. In other words, we cannot add more features to the tree and we just rely on the ones that we have.

A **Partify function** is the opposite because it wants to add more features whenever it encounters one and then make the predictions.

2.3 Not Everything is Learnable in ML

Noise: It is the things that might go wrong in the algorithm. For example, the student wrote a typo in his review or maybe he put a one star review by mistake when he meant a five-star review and so on.

Sometimes the features are not sufficient, for example you are deciding whether a patient has cancer. You look at their family history, Xray, symptoms, but you did not look to see if he has tumors.

Some examples might not even have one single correct answer. If you are building a safe web search engine that removes explicit content, some people that might consider a content explicit others might find it fine.

Underfitting: Sometimes you have little to no features which means that you do not really have the features needed to predict the right thing. Similarly, **overfitting** is the idea that you are focusing way too much on idiosyncracies rather than what the algorithm is supposed to fit. You are focusing way too much on the noice to fit rather than what it's supposed to fit. We are aiming between underfitting and overfitting. An analogy could be a student not studying for an exam and a student studying past exams but not actually learning or understanding the concepts.

But, how would you know which model is best? Simple, we need to decide which of these models have the minimum error.

First of all, how would you train a model? Suppose you have 1000 examples of pottery ratiings. You take 800 as **training data** which is the data that you will train your model, and you will set aside 200 as **test data**. After you model based on the training data, you can test your model based on the test data and then you test your model's performance by calculating the **test error** which is how accurate your predictions they were.

You must remember that you never touch your test data ever! Touching it beforehand will ruin everything.

Remember that paramters are just internal values or weights that the model learns from training data. If you are deciding which model is better than the other, you cannot use the test data and the error to decide which one is better because remember, you must not touch the test data unless you have a final model! Instead, you can have 700 as training data, 200 as test data, and 100 as **development data** and you can train different models based on the development data. Pick the model with the lowest error rate and peform the test data on that model and calculating the test error on them.

5.3 Feature Pruning and Normalization

Pruning is the idea that, for example, a word like "groovy" appeared only once in a document that has 100,000 words, does that mean that we should keep that word as a feature? The best answer should be no because we might be overfitting our model if we kept that word.

So, if a word only appeared a bunch of k times, then we could just remove that word from consideration. k is dependant on you and what you decide to be the cut-off point.

So, we might remove things based on whether they have *low variance* or not. But, most occurent words are low variant as well as least occurent words. Clearly, pruning too much might remove the important stuff.

Which is why it is important to **normalize** data so that they are consistent in some way. There is **feature normalization** and you adjust it the same way across all examples, and then there is **example normalization** which is where you adjust each example individually.

In feature normalization, there is two types of things that you could do, first you could do **centering**, where you move the entire data set so they are centered around the origin, and then there is **scaling** (one of them has to hold true):

- Each feature has variance 1 across the training data.
- Each feature has a maximum absolute value 1 across the training data.

Centering makes sure that no feature is arbitrarily large and scaling all features have the same scale.

- Scaling: $x_{n,d} \leftarrow x_{n,d} - \mu$.
- Variance scaling: $x_{n,d} \leftarrow x_{n,d}/\sigma_d$.
- Absolute scaling: $x_{n,d} \leftarrow x_{n,d}/r_d$.

Here, $x_{n,d}$ refers to the dth feature of example n. μ is the average and σ is the variance.

However, in example nortmalization, you view examples one at a time. You need to scale in a way that the length of each example vector is exactly just one. A simple transformation could be the following:

$$x_n \leftarrow x_n / \|x_n\|$$

The main advantge of this is that you can make comparisons easy across different data sets.