Lecture 4

Leif Bauer

bauer96@purdue.edu.

1 Review of Last Lecture

In the last lecture we covered the general machine learning pipeline, classifiers, and the hypothesis space which is picked by minimizing the loss function. Ideally we will find a function that minimizes the expected error.

$$\min_{h \in H} E_{(x,y)\sim Pr} L(h(x), y)$$

But in practice we don't know the underlying distribution. Instead, we take an empirical average over the points in the data set, and minimize the error.

$$\min_{h \in H} \frac{1}{N} \sum_{(x_i, y_i) \in D_{train}} L(h(x_i), y_i)$$

However, this can make arbitrary errors, so we include a test data set D_{test} on which our model is evaluated. Since we want to optimize the hyper-parameters of our model we optimize our model on a validation set D_{val}. We evaluate the results with the test set only after we finish optimizing the model with our validation set.

Recipe:

For each hyper-param

Train using D_{train}

Train using D_{val}

Choose the model with the smallest L_{val}

Report the loss using D_{test}

2 Cross Validation

There are several critiques to this type of machine learning recipe, for example there may be errors in the data set. One option in this situation is to split the data set into four parts, each with their own training and validation sets with loss functions: L^1_{train},L^1_{val},L^2_{train},L^2_{val},L^3_{train},L^3_{val},L^4_{train}, L^4_{val}.

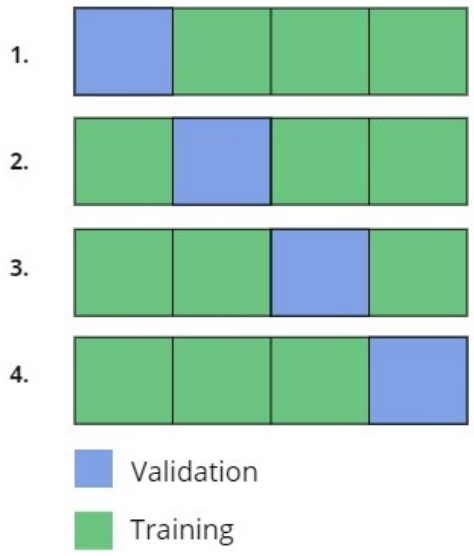

Figure 1: Cross validation data sets

We then train the data on three parts of the data set, and we use the other part to validate. Using this method we will get the minimized validation loss functions which we average together to get our model:

$$L_{val} = \frac{1}{4}(L^1_{val} + L^2_{val} + L^3_{val} + L^4_{val})$$

An advantage to this method is that each set is used in both validation and training. However, it is important to remember that the testing set is still set aside and is separate from the validation set. We must evaluate the model in a fair way.

3 Examples

3.1 Predicting the stock market using S&P 500

Suppose we are given a feature set (date, previous day's S&P 500, interest rate, gold price...) $\rightarrow$ Today's S&P 500. We need to build a regressor to predict the score based on the given features. The stock prices are shown in figure 2. Let's say we split the data into 60% training, 20% validation, and 20% testing. It turns out, this type of splitting is not a very good way to

split, because it will randomly shuffle pieces of the data. The model may just be training itself to remember the date and predict the average S&P on that week. This is very possible, and when it tries to predict tomorrows stock price, it will do a bad job, because it hasn't seen that date yet. What we need to do is use all of the data from say 1910-2000 for training and use the data from 2000-2010 for testing, and since it hasn't seen the test data, it won't just remember the date. You

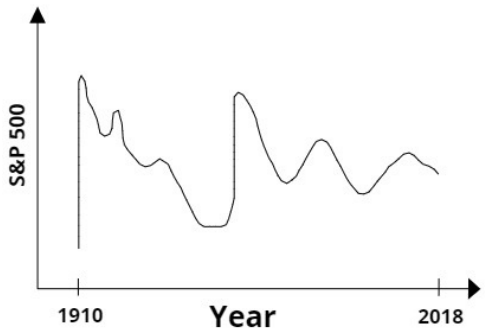

Figure 2: Temporal S&P 500 data

may be thinking, 'Why don't we just forget the date feature?', but even then we should still not do random shuffling. Suppose the interest rate is high until the 1980s and then afterwards it goes negative. This would tell the neural net that the date is after the 1980s. So, in a sense it is still using the date information, and random shuffling would continue to cause issues. In general if you have temporal data with a past and future, you should think about if it is a good idea to split data versus randomly shuffling data.

3.2 European Parliament and the Ethics of Neural Nets

The European Parliament is currently discussing several ethical concerns with neural nets, and recently passed a law that says you cannot use ethnic properties as features in a neural net to predict things. However, this is not a good solution because other features may be captured by the neural net that could still capture ethnic properties (e.g. how fast you run, how fat you are, your height, etc.). These types of changes do not make us safe from ethical concerns.

3.3 Tennis Player Example from Prerequisite Check

Suppose you are given a feature set (height, weight, etc. ...) $\rightarrow$ Tennis Players, and suppose we have 10 labeled students in our class, 40 unlabeled students in our class, and 1000 labeled students not in our class. How do we use this data? What we should do is train on the 1000 labeled students not in our class, and use the 10 labeled students from our class as the validation set. Then the 40 unlabeled students are the test set. What we should not do is combine the labeled students into 1010 and randomly shuffle to create a validation set. We want to put the 10 from our class in the validation set because they will better predict our class. The training

set are essentially unrelated. This is a problem because we typically assume that all data sets are from the same data distribution when they really are not. Can we still do this if the data sets are unrelated? Not in a classical model, we will have good training but terrible tests. This is a field of active research called domain adaptation.

4 K-NEAREST NEIGHBOR

Consider as an example the Iris dataset which is used to classify flowers. For our case we'll consider only two types of flowers the setosa (+) and the versicolor (-). The classification is done based on sepal length and petal width. Suppose we have a new item, shown in the box. Which

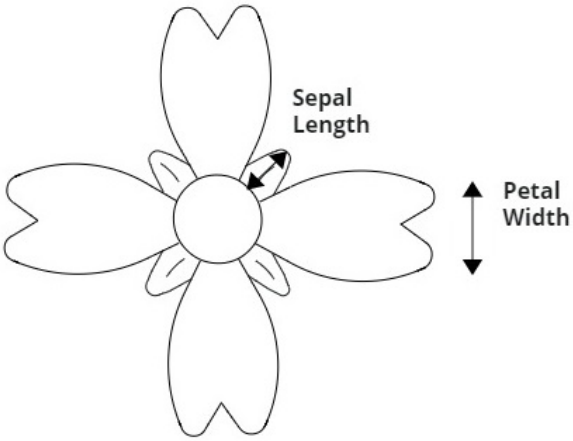

Figure 3: Iris features

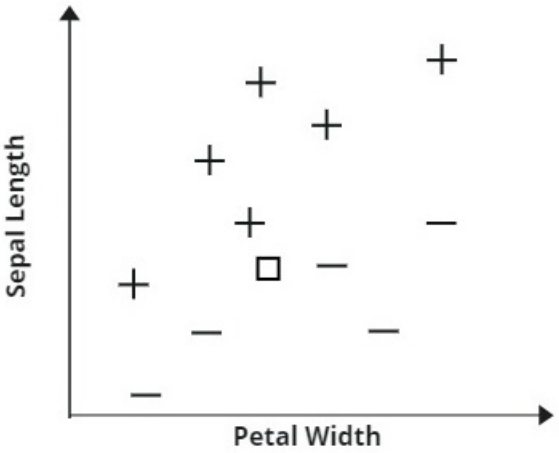

Figure 4: Graph of iris features

side is it closer to? Let's try putting it into the category based on its nearest neighbor. Suppose you have a data set D_{Train} with c labels.

$$D_{Train} = \{(x_1, y_1), ..., (x_n, y_n)\}$$
$$x_i \in \mathbb{R}^n \quad y_i \in [c] = \{1, 2, ..., c\}$$

We want to find a function to map the D dimensional space to c labels

$$f : \mathbb{R}^D \to [c]$$

4.1 1-NEAREST NEIGHBOR

The 1-nearest neighbor is defined using this equation

$$nn(x) = \arg\min_{n \in [N]} ||x - x_n||_2 = \sqrt{\sum_{d=1}^{n} (x_{1d} - x_{nd})^2}$$

where $||\quad||_2$ is the Euclidean distance. This function will return the nearest neighbor for a given input vector x. For classification we use

$$y = h_1(x) = y_{nn}(x)$$

which is the y returned from the nearest neighbor. Essentially, this makes predictions based on the nearest neighbor's label. The line which marks positions of equal distance between two

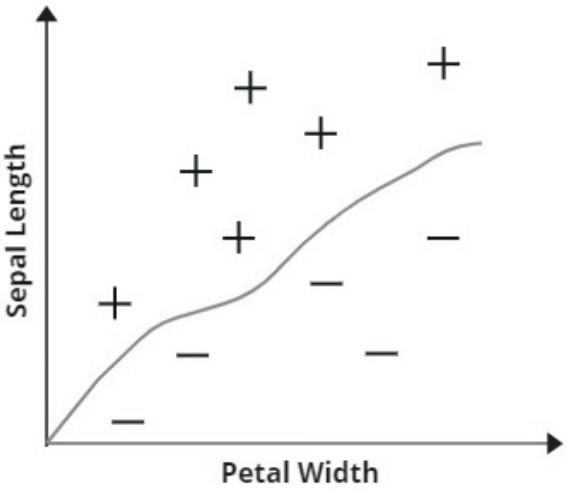

Figure 5: Graph of iris features

classes is called the decision boundary. The classifier makes its decisions based on which side of the boundary it is on. There are also other types of distance measures such as $||\quad||_0, ||\quad||_1, ||\quad||_\infty$. A more commonly used method is called k-nearest neighbor. The high level idea is to look at the k-nearest neighbors around the point. These k neighbors then vote to decide which label it gets. As an exercise, can you write this down in mathematical language?

5 THE IMPORTANCE OF NORMALIZATION

Consider this data set

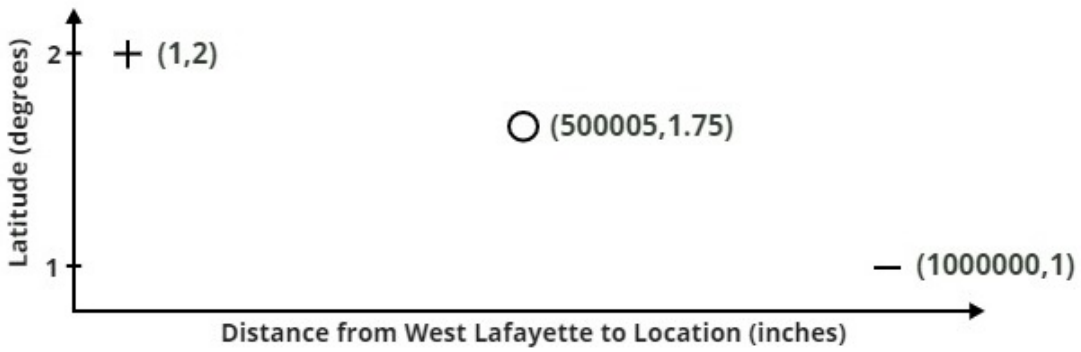

Figure 6: Non-normalized data

The middle point is closer to (-) from Euclidean distance. Actually, no matter where you are on the y-axis, it will not make a difference. This is what happens when you don't normalize data. This is especially common for linear models when you don't normalize each dimension.

6 BAYES OPTIMAL CLASSIFIER

Consider the expected risk:

$$R(h) = E_{(x,y)\sim Pr} L(h(x), y)$$

where h is the classifier and $L(h(x), y) = \mathbb{1}(h(x) \neq y)$. Let's say we have backdoor info that we shouldn't know, but somehow we know which is $Pr(y|x)$. What kind of classifier do we use? Suppose we write a classifier that outputs the category with the largest conditional probability:

$$f^*(x) = \arg\max_{c\in[C]} Pr(c|x)$$

This is called the Bayes optimal classifier. It has a very nice property:

$$R(f^*) \leq R(f) \quad \forall f$$

Essentially what this says is that the Bayes optimal classifier risk is less than or equal to the risk of any other classifier. We will get to the proof in the next class, but essentially it is due to how we calculate the risk:

$$\begin{aligned} R(f^*) &= E_{x\sim Pr} E_{y\sim Pr(y|x)}[\mathbb{1}(h(x) \neq y)] \\ &= E_{x\sim Pr}[1 - \max_{c\in[C]} Pr(c|x)] \end{aligned}$$

In the next lecture we will also prove that the nearest neighbor classifier is *almost* Bayesian optimal.

$$R(f^*) \leq \lim_{n \to \infty} E[R(f_N)] \leq 2R(f^*)$$

Which was proven by a theorem by Cover and Hart. This is very good because we don't have hidden information $Pr(y|x)$.