Chapter 5
Classification

Slides (pdf) Video (Panopto)

So far, we have talked about Regression analysis for continuous output values such as house prices, where a house price can be any number in a range. We will now transition to the next application of machine learning, Classification. Now, our output values are no longer continuous , rather they are distinct *classes*.

Definition 5.0.1: Classification

Classification is an application of machine learning where a class label is predicted based on some input data.

Examples of classification are classifying emails as spam or not, classifying animals as cats or dogs, classifying product reviews as positive or negative. Notice that in all of these scenarios, our outputs are not numerical values along a range, so regression no longer applies here. This means that for our quality metric, Mean-Squared Error is also no longer a viable option because we do not have predicted numerical values.

We will examine classification through a restaurant review example. Suppose you have a collection of different restaurant reviews and you want to classify them as positive or negative. You would like to extract the sentences from the review, feed them as input to a classifier model, and let the model tell you if the review is good or bad. How would you go about this?

5.1 Simple Threshold Classifier - Implementation 1

The first way we could solve this problem is by using a Simple Threshold Classifier. In this type of classifier, we do **Sentiment Analysis**: We will curate a list of positive and negative words, and then classify each of the reviews by the frequency of positive words and the frequency of negative words. Our lists may look something like this:

Positive: "great", "awesome", "good", "amazing"
Negative: "bad", "terrible", "disgusting", "sucks"

If a sentence has more positive words than negative words, we will assign it the output value $y = +1$, to indicate that this is a *positive review*. Otherwise, we will assign it the output value $y = -1$, to indicate that this is a *negative review*. For example, if our review was "Sushi was great, food was awesome, but the service was terrible", we count two positive words "great" and "awesome", and one negative word "terrible". Since there are more positive words than negative words, we assign this review the output $+1$, and classify it as a positive review.

There are unfortunately a number of limitations with this approach. First off, how do we get the list of positive and negative words? If we want a robust list covering the entire dataset, do we have to go through each review ourselves and painstakingly write this list up from scratch? And also, what about words that have varying degrees of sentiment? Would you say "awesome" has a higher sentiment than "good"? And what about the adverb "not"? If we say "not good" we mean negative sentiment, but "good" was on the list of positive words. As you can see, the Simple Threshold Classifier may not suffice for our problem, which calls the need for a better approach.

5.2 Linear Classifier - Implementation 2

We will try to account for the issue of varying sentiment in a Linear Classifier approach. Unlike the Simple Threshold Classifier, the Linear Classifier does not require us to write a list of positive and negative words ourselves. Instead, it will *learn*, using labeled training data, the weights of different words. These weights will be used to score a sentence. Suppose we have trained a model that takes in reviews x and their ratings y, and uses the words in each as features to predict the review's sentiment $\hat{y}$. Let the following be the learned weights for each word:

Word	Weight
good	1.0
great	1.5
awesome	2.7
bad	-1.0
terrible	-2.1
awful	-3.3
restaurant, the, we, where, ...	0.0
...	...

Figure 5.0.1: Learned weights representing sentiments for words found in the restaurant reviews.

Now let's see how these weights play out on our example:

"Sushi was great, food was awesome, but the service was terrible"

We will calculate the sentiment score of this input using the frequencies of each word:

$$Score(x) = \text{"awesome"} * 1 + \text{"great"} * 1 + \text{"terrible"} x1$$
$$Score(x) = 2.7 * 1 + 1.5 * 1 + -2.1 * 1$$
$$Score(x) = +2.1$$

As the score here is positive, we decide this review is positive. We call this classification method "Linear" because our output is a linear weighted sum of inputs. Note here that our threshold is 0, and in the unlikely event we achieved a score of exactly 0, we would choose an arbitrary class for this review.

Can you think of some limitations for this approach? We still have not really resolved the case "not good" meaning not good, and "not not good" meaning good. Also, sometimes a word's sentiment relies on surrounding words. For example, saying "a bit sad" is less negative than just "sad". Thus, a linear classifier cannot learn complex models very well. Though we will not go into it much in this textbook, **Natural Language Processing (NLP)** is a field of machine learning that is used for more complex analyses of language.

Model: $\hat{y}^{(i)} = sign\left(Score\left(x^{(i)}\right)\right)$

$$Score(x_i) = w_0 h_0\left(x^{(i)}\right) + w_1 h_1\left(x^{(i)}\right) + \ \ldots \ + w_D h_D\left(x^{(i)}\right)$$

$$= \sum_{j=0}^{D} w_j h_j(x^{(i)})$$

$$= w^T h(x^{(i)})$$

We will also use the notation

$$\hat{s}^{(i)} = Score\left(x^{(i)}\right) = w^T h\left(x^{(i)}\right)$$

$$\hat{y}^{(i)} = sign(\hat{s}^{(i)})$$

Figure 5.0.2: Notation for linear classifiers.

5.3 Decision Boundaries

We can visualize the effect of the weighted words on determining if a review is positive or negative via illustrating the decision boundary. For simplicity's sake, suppose we only had two nonzero-weighted features, "awful" and "awesome". Our decision boundary would look like so:

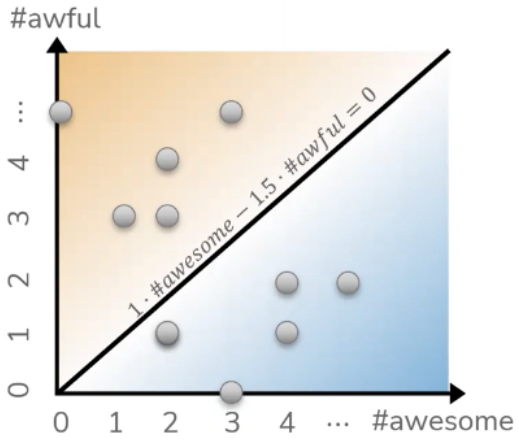

Figure 5.0.3: A linear decision boundary between two features.

Since we have more than two nonzero-weighted features, the decision boundary for the example would be a higher-dimensional graph with linear boundaries. If we needed a more complex, nonlinear decision boundary, then we would need a more complex model. These are examples of more complex decision boundaries:

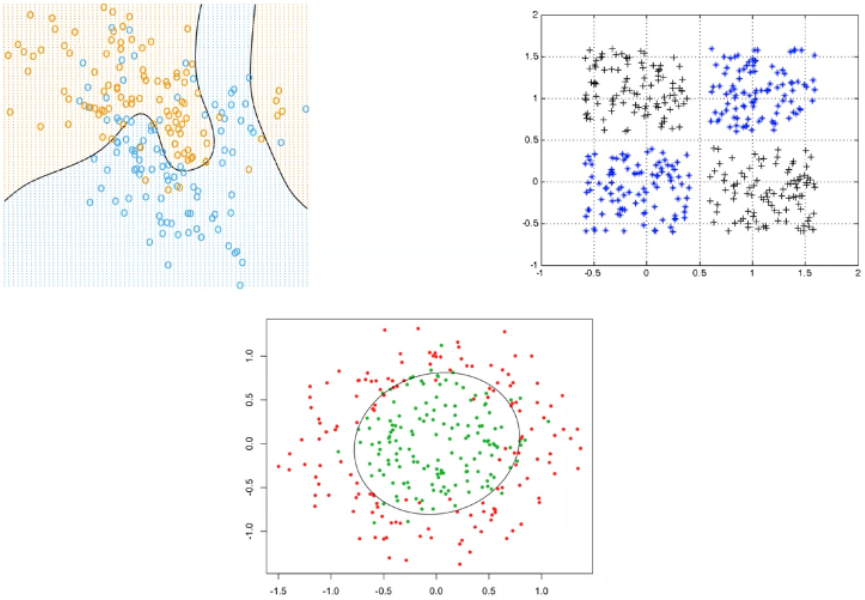

Figure 5.0.4: Visualization of nonlinear decision boundaries produced by more complex classification models.

5.4 Classification Error

As we mentioned earlier, Mean-Squared Error is no longer applicable to Classification, because we aren't trying to fit a predictor to some datapoints by reducing each point's error from the predictor. This is because our output values aren't continuous numerical values, they are classes. So, we calculate error in a much simpler way:

$$Error = (\text{wrong classifications})/(\text{total classifications})$$

$$Error = \frac{1}{n}\sum_{i=1}^{n}(y_i \neq \hat{y_i}) \tag{5.0.1}$$

where n is the total number of classifications.

A "wrong" classification would be if the true label y of a review was positive, but we predicted $\hat{y}$ to be negative, or if the true label y of a review was negative, but we predicted $\hat{y}$ to be positive. Thus, for our accuract calculation we have:

$$Accuracy = 1 - Error = (\text{correct classifications})/(\text{total classifications})$$

5.5 Classification Accuracy

When calculating the accuracy of a classification model, it is important to note that since we are outputting distinct classes, not all correct classifications are due to the model's own learning. Some are due to random chance. For example if you had two output classes that were equally likely and your model was simply randomly guessing, it could achieve 0.5 accuracy. If all classes are equally likely to be chosen, then random-chance accuracy is $1/k$ classes.

Accuracy as a metric should be taken with some precaution. If you were trying to classify emails as spam or not spam, and you made a Dummy Classifier that always outputted "spam" no matter what, you could get 90% accuracy, simply because you might have 90% of your inbox filled with spam! We call this Dummy Classifier a Majority Class Classifier, because it classifies everything as whatever the majority is. At the bare minimum, your classification model should be higher than a Majority Class Classifier.

Evidently, if there is a class imbalance we might get a higher accuracy than random chance when this is not actually a reliable baseline to compare our model performance to. This is where a **confusion matrix** is helpful.

Definition 5.0.2: Confusion Matrix

A **Confusion Matrix** is a table comparing all the true positive, false positive, false negative, and true negative values from the output of a model.

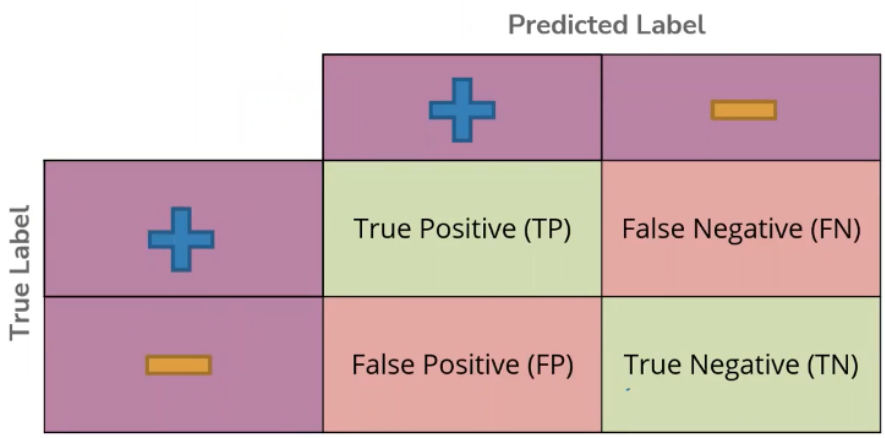

Figure 5.0.5: A confusion matrix for binary classification.

Is a false negative or a false positive worse? The answer relies on your application. If we are building a model to detect spam email, a false negative would result in an annoying spam email in your inbox, but a false positive would mean that an important email is now lost. If we are building a model to detect a disease, a false negative means that the disease goes untreated and a patient could suffer greatly while a false positive means that a healthy patient wasted some money on a treatment. Depending on the consequences of false predictions of our models, we will have to make decisions on whether or not the model is ethical. In later chapters we will discuss the ethical consequences of erroneous or discriminatory models on different demographic groups further in depth.

5.6 Learning Theory

How does adding more data affect the error rate of our models? Previously, we discussed a limitation of the phrase "not good". While this has a positive word "good" in it, ultimately it should not count positively towards the score. So for our language analysis we might look at two words at a time instead of one. We refer to one word as a "unigram" and two words as a "bigram". In general, the more data the better. However, it is interesting to note that if we parse our sentences differently, the advantage of more data changes drastically. A model that uses bigrams instead of unigrams is a more complex model, so it will

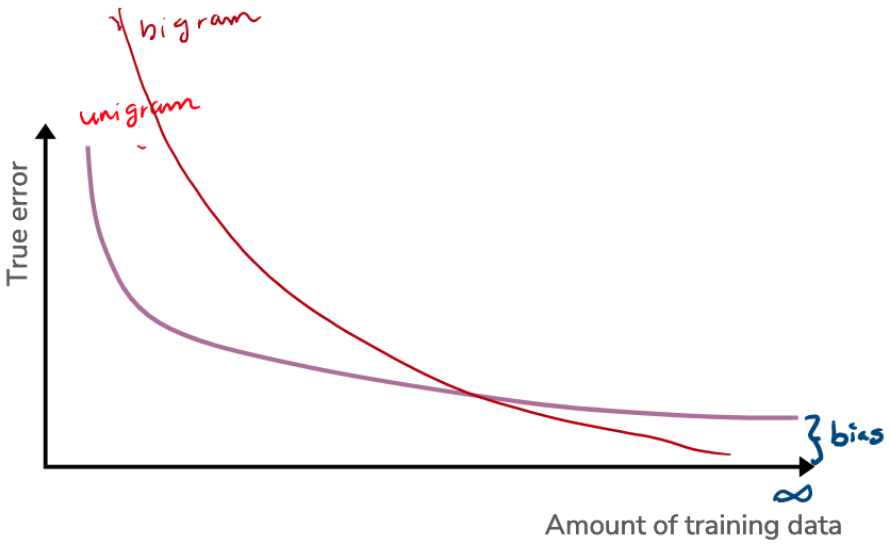

Figure 5.0.6: The learning curve for different data parsing types.

do worse than a unigram model with less data (due to overfitting to the limited training set), but after a sufficient amount of data, it will start doing much better.

5.7 Changing the Threshold

If some types of error, such as false negatives or false positives are more unacceptable than the other in our context, it might be a good idea to change our scoring threshold in the decision boundary. If a doctor wants to lower the risk of making a false negative prediction for the safety of her patients, she could tune her model's threshold to be closer to negative ∞. This way, a score less than 0 is required to output "negative", so while we increase the likelihood of false positives, at least we decrease the likelihood of false negatives, which are more dangerous in the medical world.

Conversely, if I want to lower my chances of missing out on Aritzia's latest Winter sale in my inbox, I would tune my spam-detector model's threshold to be closer to positive ∞. I want there to be less instances of detecting spam when the email is actually important, so by increasing the score threshold, the score must be higher to be marked as spam.