Evaluation of Models and Sources of Error in the ML Pipeline

CS 360 Machine Learning

Week 3, Day 1

February 6, 2024

Contents

1 Evaluation of models

As a quick review, we've seen that the performance on training data overestimates the accuracy in reality. That's why we use the held aside test data, something the model didn't see, to evaluate the model. Ideally the training data and test data should both be drawn from the same distribution. In deployment, that data should ideally also be drawn from the same distribution. In reality, we don't know what these distributions are, but our goal is to think about how well the training data and testing data distributions match. Looking at an example, we see that as we increase the model complexity, the training accuracy keeps going up, but at some point the *testing* accuracy starts to go down.

Let's try to make this description more concrete. Consider a hypothesis (model) h with training error $\text{error}_{train}(h)$ and the error over all possible datasets denoted $\text{error}_{\mathbb{D}}(h)$. This evaluation over all possible datasets is hypothetical. We can now say that a hypothesis h *overfits* the training data if there exists another hypothesis h' such that:

$$\text{error}_{train}(h) \leq \text{error}_{train}(h') \quad \text{AND}$$

$$\text{error}_{\mathbb{D}}(h) > \text{error}_{\mathbb{D}}(h')$$

Here we imagine we have a fixed amount of data, though in the real world we might consider an ongoing collection of data and associated ongoing refinement of the model. (The process of continually incorporating such data into the training and testing process is tricky and requires being careful not to train on test data; one common way to handle this is to retrain in batches.)

1.1 Loss functions

We've considered common notions of accuracy and error, here we formalize them.

Definition 1 (Zero-one loss)**.**

$$\ell(y, \hat{h}) = \begin{cases} 0 & \textit{if } y = \hat{y} \\ 1 & \textit{otherwise} \end{cases} \tag{1}$$

Definition 2 (Squared loss)**.**

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

Zero-one loss is simple accuracy - is the prediction right for binary of multi-class prediction. For regression, we've also considered squared loss. We could also consider the absolute loss for regression, but it's not differentiable everywhere, so we can't always make use of it.

These are the functions we want to use when we want to minimize our error. We think about the overall problem we're solving in the following way. Given a loss function ℓ, a sample of data D from an unknown distribution of all data $\mathbb{D}$, and a hypothesis space $H = \{h | h : X \to Y\}$. Our goal is to find a function $f(X) \to y$ that minimizes the error over $\mathbb{D}$ with respect to ℓ. We call this the generalization error.

We assume that the distributions are drawn i.i.d, meaning that they are drawn independently and identically distributed from all of the data; all data points are similarly probable to occur. (There are exceptions to this assumption such as time series data, structured data, and active learning).

The *generalization error* is thus defined as the expected value:

$$\mathbb{E}_{(x,y)} \left[\ell(y, \hat{f}(x)) \right]$$

where the first part of the loss function represents the true values while the second part is the predicted values.

Assuming that we're considering a regression problem, this means that $y = f(x) + \varepsilon$ where f is some underlying true model of the real phenomena that created the data we're considering and ε is some amount of error from that true model, which is assumed to have a mean of 0. Both f and ε are unknown, while y is assumed to be observable. In a regression problem we're trying to predict $\hat{y} = \hat{f}(x)$ based on a training process.

Our goal is to minimize the expected value of the loss, as described above. Our training data is only a sample from the true distribution, which sets up a dilemma in minimizing based on the true distribution. What are the sources of error here; why might learning fail?

Inductive bias is when we made assumptions at the beginning that led us down a poor path. Preferring one classification over another, for example, when we know what the data looks like and have a preconceived hypothesis that makes use of our background knowledge or assumptions, is known as inductive bias. When picking an algorithm for a dataset, we need to be aware of this bias – we might not know what the right solution is.

We might also have other issues that lead to errors, such as noise in the training data (e.g. typos, scientific measurement noise, etc.). The available features may be insufficient to model the phenomena we're interested, such as an x-ray that doesn't capture the medical issues of interest. There may be a mismatch between what you want to learn and the data you actually have. There may also be an issue where even a "correct" prediction is up for interpretation. A learning algorithm may also not be able to accurately model the data. We come back to these concerns later in this class.

First, we ask: what about sources of errors that are fundamental to the learning process itself?

1.2 The Bias-Variance Tradeoff

You may have seen this as a concept in CS 260, but we'll make it more formal here. We'll consider the expected value of the mean-squared error (MSE) of a regression problem, i.e., the loss function:

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

(Squaring it makes the loss function differentiable.) If I want to find the expected value of this loss $\mathbb{E}[(y - \hat{y})^2]$ I know that y in my mental model (described above) is $f(x) + \varepsilon$, and by definition $\hat{y} = \hat{f}(x)$, which gives:

$$\mathbb{E}[(y - \hat{y})^2] = \mathbb{E}[f(x) + \varepsilon - \hat{f}(x)]^2$$

When you square this and rearrange terms you end up with:

$$= \mathbb{E}[f(x) - \hat{f}(x)]^2 + Var(\varepsilon)$$

Reducible Error | Irreducible Error

where $Var(\varepsilon)$ is the variance of the data error term ε and we have assumed that its mean is 0. The left part is known as the *reducible error* and represents the error terms due to the choice of model, while the right part of the above is known as *irreducible error* and includes the error inherent to the data.

Using the trick of adding and subtracting the same thing (so we're adding zero but it allows us to break apart the terms), we get:

$$= \mathbb{E}\left[f(x) - \mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)] - \hat{f}(x)\right]^2 + Var(\varepsilon)$$

After squaring and rearranging terms, this gives:

$$= \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + \mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right)\right]^2 + Var(\varepsilon)$$

Bias | Variance | Irreducible Error

Recall that this is an expected value over different possible training datasets, and our goal is to have a model close to the "true" model. The resulting part on the left is known as *bias* in a statistical sense and to the right of that is the model's variance. Note that the variance only depends on the trained model. I'd like it to be small, so that the phenomena is predictably estimated from whatever training set I have.

What about bias – do we want it to be low or high? We also want this to be low. Recall that we want the overall error to be low. As you increase the flexibility (complexity) of the model, the bias goes down, but the variance may go up. This is known as the *bias-variance tradeoff*:

$$= bias(\hat{f})^2 + Var(\hat{f}) + Var(\varepsilon)$$

This tradeoff has three components: 1) the bias, which we can't measure, 2) the variance of the model, which we can measure, and 3) the variance of the noise, which we usually also don't know. These are the three components that make up the total error, and there's a limit to how much we can reduce the overall error. As flexibility (model complexity) goes up, bias will go down, but variance will go up. As you become less flexible, the bias will go up, but the variance will go down. Remember that the variance we're considering is how different the model is when you consider different training data. I.e., we're considering what would happen if we happened to be given a different training data sampled from the true distribution $\mathbb{D}$ and how that might increase the error of a resulting trained model. This helps to measure how much your model is overfitting to the noise. Note that there's also the noise term that can't be removed - this assumes that there is some measurement error or other noise that's inherent to the data.

1.3 Cross-validation

How can we make the general approach to machine learning somewhat better? One general approach is to split your data into 70% training data, 10% development data (validation data), and 20% test data. For each possible setting of our hyperparameters, we train a model using that setting of hyperparameters on the training data, and compute the model's error rate on the development data. From the above collection of models, choose the one that achieved the lowest error rate on the development data. Finally, do evaluation of the model on the test data.

In practice, unfortunately this process might be repeated many times if that final evaluation step on the test data doesn't get results that you like. But the problem with this is that you've now used the test data as part of your model selection. This violates our running edict: **don't touch the test data!**

One question / concern that arises that might lead to this type of error is: why did we choose that specific "split" of the data into training and validation sets? In principle, we might think, we should do this multiple times since performance may be different for each split.

We could use k-fold cross-validation (e.g., $k = 10$) to handle this concern. In this case, we randomly partition the full dataset of n instances into k disjoint subsets. This is a way of using all of the training data to also do validation – note that the test data is still separated and held out for later testing. In this case, for each partition, we learn on the training data and validate on the validation partition. We do this for each of the different partitions and report the summary statistics (e.g., accuracy) over all of these splits, with each hyperparameter choice evaluated using the summary statistic computed by fixing the hyperparameter and evaluating across all splits. We would then choose the model with the best validation performance, chosen over all these hyperparameter choices, to run on the test data. This allows us to compare models in a more robust way. Note that k will be chosen based on how much data there is, so that the training and validation sets have enough data to be effective; often people choose $k = 10$ in practice, but it's better to choose this based on the size of your data.

2 Sources of error in a machine learning pipeline

Accuracy and other traditional error measures focus on evaluating a trained model against the test data, using confusion matrices and associated error measures. However, recent critiques of deployed models have considered the potential societal impact when models are deployed in ways that directly impact people and suggest the need for evaluation beyond these traditional approaches.

2.1 Fairness measures

As you may have discussed in CS 106, a criminal risk assessment model was created which identified defendants as low or high risk for recidivism based on a label measuring rearrest of individuals within 2 years. It was found, in an article by Propublica, that this risk assessment essentially gave the benefit of the doubt to white defendants—it was more likely to label them low-risk even if they went on to reoffend—while it did the opposite with Black defendants, labeling them high-risk even when they didn't reoffend. Key to this analysis was a table showing the rates at which white and Black defendants received misclassifications—an analysis similar to, but not quite the same as, a confusion matrix analysis.

In another case, a prominent academic article and accompanying project by Joy Boulamwini and Timnit Gebru demonstrated that commercially deployed gender classification algorithms based on facial photos were more likely to work incorrectly for darker skinned women than for white people regardless of gender or for men regardless of skin color. Again, the analysis rested on misclassification rates *per demographic group*, such as the true positive rate for darker skinned women.

When considering the potential civil rights impact of machine learning models on employment, the U.S. Equal Employment Opportunity Commission has put out technical assistance making it clear that civil rights laws still apply when models or other algorithms are used in a hiring context. This includes a general rule-of-thumb known as the "four-fifths rule" that compares selection rates per demographic group to determine if they are substantially different from each other. It is calculated based on the per-demographic group predictions.

These measures, designed to identify potential discrimination in the outcomes of machine learning models, can all be calculated based on modified confusion matrices that are calculated per demographic group. While a general confusion matrix includes the counts for each item in a test set, a confusion matrix for a fairness analysis includes only those examples from a specific demographic group in that matrix.

Table 1: Confusion matrices for an example where a model is selecting people to be recommended to a hiring committee based on existing employee data and an analysis is being done to determine whether men are advantaged in the process. Top: classifications of examples that represent men. Bottom: classifications of examples that represent non-men.

		Predicted Class	
		Don't Hire	DO Hire
Test Data Label	Wasn't Hired	TN_{men}	TP_{men}
	WAS Hired	FN_{men}	TP_{men}

		Predicted Class	
		Don't Hire	DO Hire
Test Data Label	Wasn't Hired	TN_{non}	FP_{non}
	WAS Hired	FN_{non}	TP_{non}

Based on these confusion matrices, we can calculate the four-fifths ratio:

$$\frac{\text{rate of hiring for non-men}}{\text{rate of hiring for men}} \geq \frac{4}{5}$$

In an ideal world, these rates would be equal, but the "four-fifths" part of the rule comes from the idea that there can be some leeway for less than exactly equal rates. In terms of our confusion matrices, this is calculated as:

$$\frac{\left(\frac{FP_{non}+TP_{non}}{\text{total non-men}}\right)}{\left(\frac{FP_{men}+TP_{men}}{\text{total men}}\right)}$$

Note that this ratio is calculated without considering the test data label, i.e., entirely based on the predictions. In a hiring context, we can see how this might make sense. Consider a company that was historically discriminatory—it likely uses its historical employment data as the training and test data. In this case, matching new predicted classifications to historical data and attempting to equalize those based on historical demographic groups may not be desired, and could even be discriminatory. The four-fifths rule considers only the columns of these per-demographic group confusion matrices.

When considering what it means to be "fair" in the case of misclassification, we may again want to see that various error measures are equal when calculated for each demographic group as usual based on the separated confusion matrices. For example, we may want to see that the ratio of true positive rates is

equal to 1.

$$\frac{\text{true positive rate for non-men}}{\text{true positive rate for men}} = 1$$

$$\frac{\left(\frac{TP_{non}}{TP_{non}+FN_{non}}\right)}{\left(\frac{TP_{men}}{TP_{men}+FN_{men}}\right)} = 1$$

Or we might want to consider false positive rates, or another error measure.

$$\frac{\text{false positive rate for non-men}}{\text{false positive rate for men}} = 1$$

Once we've computed the per-demographic group confusion matrices, most fairness measures are ratios or differences of various entries.

2.2 Sources of error and model cards

The traditional machine learning pipeline when considered in deployment first creates a model based on training data and a chosen model type and then deploys the model in the real world when given an example and generates a prediction. However, in a real-world deployment scenario we need to consider the assumptions made throughout the pipeline. Each of these assumptions could be potential sources of error.

One key assumption is the reliance on machine learning in the first place—it's important to interrogate whether machine learning is even the right approach to solve a problem! Perhaps there's a better solution with different technology or outside the bounds of computer science entirely.

Another assumption that can commonly lead to errors in the real world is that the trained model is appropriate to the given real world task. Conditions, which lead to training and test data, could change in the real world or the model could be deployed in a new context that doesn't match the conditions it was designed for.

In order to identify these and other assumptions and hopefully prevent deployment errors, one approach is to state these assumptions explicitly. Model cards are one approach to transparency reporting about machine learning models that has been widely adopted by industry (including examples from Google, Hugging Face, Amazon, and Jigsaw). Key to this approach is the explicit identification and reporting of intended uses of the model based on its training data and process as well as uses that are known to be out-of-scope.