Statistical NLP: course notes

Çağrı Çöltekin — SfS / University of Tübingen

2020-05-25

These notes are prepared for the class *Statistical Natural Language Processing* taught in Seminar für Sprachwissenschaft, University of Tübingen.

6 *Evaluating and tuning machine learning systems*

So far, we defined evaluation metrics for both regression and classification. For regression, we typically use either *root mean squared error* (RMSE) as an indication of average error made by the system, or, *coefficient of determination* (R^2) as an indication of model fit. For classification, we have defined *accuracy* as well as *precision*, *recall* and their harmonic mean F_1 *score*. We also constantly repeated that, with any machine learning method, the aim is to perform well on new, unseen data points. A complex model can always memorize the answers we expect on the training data, resulting in *overfitting*. An overfitted model will perform well on the training data, but worse on the test instances.

We have also discussed, *regularization*, a way to counteract overfitting. In this short lecture, we will discuss a few more issues related to overfitting, and common practices for choosing better models – models with small test error.

6.1 Bias and variance of an estimator

There are two important quantities of interest that affect procedures of model tuning, or choosing good models among a family of models.

Bias of an estimator is the difference between the estimate and its true value. It is simply the difference between the expected estimate of the model parameters and the true, ideal, model parameters.

$$\text{bias}(\hat{\boldsymbol{w}}) = \mathrm{E}[\hat{\boldsymbol{w}}] - \boldsymbol{w} \tag{6.1}$$

An estimator with bias 0 is called an *unbiased* estimator. Even if we have an unbiased estimator, it does not mean that we will have a good estimate. The above formula tells us that on average, an unbiased estimator's estimate will be the same as the true parameter value. An individual estimate may, and typically does, diverge from the true value of the parameter. Put it another way, if we use an unbiased estimator many times to estimate the parameters, the average estimate will converge to the true parameter value in the limit.

Variance is another important property of an estimator. All else

being equal, we prefer estimators with low bias. However, as we hinted above, bias is not the only concern. Bias indicates a tendency in the limit, which is reassuring if we had infinite amount of data (and power to process them). However, we generally get only one or a limited number of chances to estimate a model. And even if we have an unbiased estimator, there are no guarantees that the single estimate we get is not far from the true parameter value. As we know, the expected divergence of an individual data point (in this case parameter estimate) from its expected value is its variance. Formally,

$$\text{var}(\hat{\boldsymbol{w}}) = \mathrm{E}\left[(\hat{\boldsymbol{w}} - \mathrm{E}[\hat{\boldsymbol{w}}])^2\right] .$$

Hence, as well as being low-bias, we want our estimators to have low variance.

Before discussing the bias and variance of estimators further, let us make the above discussion more concrete with an example. Figure 6.1 presents parameter values from two regression estimates for the same data. The top plot shows the estimates obtained with a *ordinary least squares regression* (OLS) estimation. The bottom plot shows least squares regression with L2 regularization (ridge regression). Both estimates are performed on the same 1 000 draws from the true regression model $y = 2x_1 + 2x_2$ with added Gaussian noise. Each gray dot on both plots represents parameters w_1 and w_2 estimated using a random draw from this model. Thanks to synthetic data, we know the true parameter values $(2, 2)$ which is marked on the plots with a cross. And the mean of the parameter estimates are indicated with a circle.

Note that the OLS estimate (top plot in Figure 6.1) has a low bias: the average of the estimated parameters are closer to the true values. In fact, the OLS is an unbiased estimator, if we repeat the estimation more, the average would get closer and closer to the true values. It is also known to be the unbiased estimator with the lowest variance. However, the variance of the OLS is high compared to the ridge regression estimate (the bottom plot), which clearly exhibits more bias. Since the ridge regression tries to minimize the parameter values together with the training error, the average estimate is biased towards the origin $(0, 0)$. Yet, its variance is moch lower, making it less likely for an individual estimate to fall too far from the true parameters. In the experiment plotted in Figure 6.1, the variances of OLS estimates are over 100 times more than the variances of their regularized estimates.

As it should be clear from the discussion above, we want low-bias, low-variance estimators. However, as it turns out, this is a trade off. Low variance comes with the cost of high bias, and low bias comes with the cost of high variance.

Bias and variance is also related to overfitting. An estimator with high variance is likely to overfit to the data. An estimator with high bias is, as expected, likely to underfit, not able to learn the true parameters. Bias and variance are properties of an estimator. However,

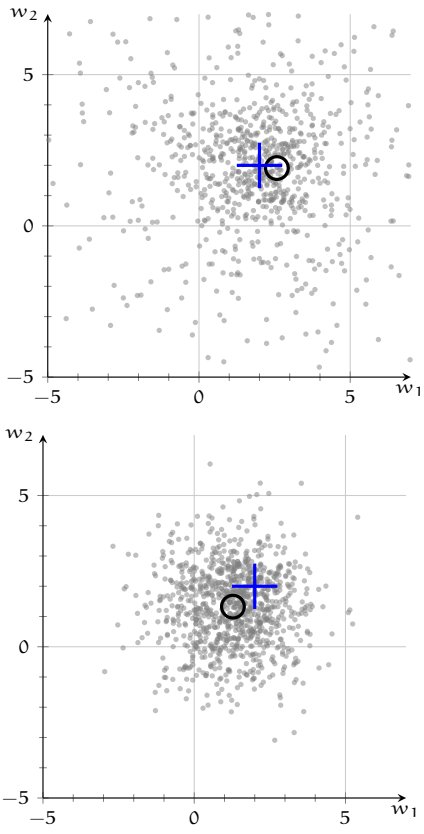

Figure 6.1: A demonstration of bias and variance through a regression problem with two predictors. Each dot in the plots indicates an estimate of the coefficients of a regression equation sampled using three $(\boldsymbol{x}, y)$ pairs from the equation $y = 2x_1 + 2x_2$ with added Gaussian noise. The values on the top plot are estimated using ordinary least-squares regression, while the bottom plot includes an L2 regularization term with regularization parameter 10. The true values of parameters $(2, 2)$ is indicated by a cross on the plots. The circles indicate average of the estimates.

it also interacts with model complexity. Models with high complexity (e.g., many parameters) tend to have high variance, while simpler models exhibit low variance, but high bias.

6.2 *Model selection and tuning*

The best way to prevent overfitting and high variance is more data. However, this is often not a (cheap) choice. In many practical use cases, we have a limited amount of data. In almost any use of supervised machine learning systems, we face with the task of selecting a model among a set of models, with possibly different characteristics or architectures. However, even with a single model family, e.g., even if we are using regression, there are some aspects of the model we want to tune. For a regression model, this could involve eliminating some of the predictors (hence their coefficients) to simplify the model, using a particular regularization method (e.g., L1 or L2), and/or choosing the best regularization strength. Note that all of these has to be fixed at the time of training. Hence, we cannot (in general) learn the same method used for learning the parameters. Such parameters, ones that needs to be fixed outside the training procedure, are called *hyperparameters*. And, the task of selecting a model can be considered as tuning these hyperparameters.

As noted many times earlier, the whole point of the exercise is to estimate a good model that does not overfit. We want a model that makes fewer mistakes on unseen data. In other words, our ultimate aim is to reduce the test error. In practice, we do not know the labels for the data points that our model will be used on. As a result, we do not know the test error. However, we can estimate the test error on the labeled data set we have at hand. The crucial point here, however, is to make sure that the part of the data we use for estimating the test error and the part of the data that we use for training do not overlap. If we tune the hyperparameters on the training data, we would also be 'overfitting' them to the training set. Hence, in practice we use a portion of data, often called *development set* or *held-out data* for testing different models/hyperparameters, while using the other rest of the data, *training set*, for training each of the alternative models.

As long as the distribution of the new/unseen data is the same as the distribution of the training/development set, this procedure works well. However, we would not be making use of a part of the annotated data. Furthermore, we would be tuning our hyperparameters on a fixed part of the data set, which may not be a good proxy for the test set, especially if the size of the data is small. The hyperparameters tuned on a single fixed validation set may also result in 'overtuning' to the validation data used.

K-FOLD CROSS VALIDATION is technique that allows to use the complete data for training and tuning. In k-fold cross validation, we repeat each step of the tuning process (e.g., for each set of hyperparameter values we explore) k times. At each repetition (fold), we hold

a different $1/k^{th}$ part of the data out as the development set, and we use the rest of the data as training set. Figure 6.2 demonstrates the way data would be split for 5-fold cross validation. Typically, we take the average performance/error metric from all folds as our estimate of performance/error on the test data. Hence, we choose the hyperparameter setting whose k-fold performance is the best. If we also average over the parameter values learned in each fold, we often arrive at a parameter estimate with less variance.

Typical values for k in k-fold cross validation used in practice are 5 or 10. A special case of k-fold cross validation, where k is equal to the number of data points is called *leave-one-out cross validation*. In general, the choice of number k in cross validation shows a trend similar to bias–variance trade-off we discussed. Remember that our aim with k-fold cross validation is to estimate the test error. A large k results in smaller held-out data, causing a more varied performance score, but a less biased estimate of the test error/performance, since we would be averaging over many scores. A small k, on the other hand, result in larger validation sets and low variance, but it will also bias the estimates towards ones that work best on these small number of validation sets.

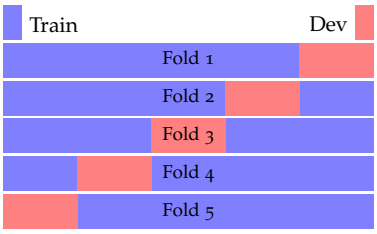

Figure 6.2: A schematic description of k-fold cross validation. Each row correspond to an experiment where the part of the data (marked red) is held-out, and the rest (blue segments) is used for training.

A practical issue with held-out data (and cross validation) in classification is the distribution of class labels across the splits. To keep the class distribution of training and held-out data the same often shuffling it before the split is sufficient. However, if some of the class labels are rare there we may end up with splits where some class labels are not represented in the training or held-out sections of the data. To prevent this, a methods called *stratified* split is used. The idea is simply to keep the class distribution same in both training and held-out parts of the data by splitting the parts of the data with the same class distribution.

6.3 *Comparing with a baseline*

Even if we tune our system to the best of our ability, the question of whether the model is doing anything useful cannot simply be answered by evaluating a single model. This is where a *baseline model* is useful. In the simplest case, we expect our models to perform better than trivial baselines. A common trivial baseline is a *random baseline* which determines the outcome randomly. For classification, for example, we would randomly assign one of the class labels to each test instance. Another common choice for classification is the *majority class* baseline, which will definitely give a trivial baseline better than the random baseline in case of class imbalance.

In many problems, however, there are other trivial baselines that perform better than a random or majority class baseline. For many cases there often are existing non-trivial solutions, or *state-of-the-art* results. If such a state-of-the-art baseline exist, it is more informative compare our models with a such a baseline or a known result from the relevant literature. This sort of comparison often requires yet an-

other split of the annotated data set. Even if we are using a held-out development set, since the hyperparameters are tuned on this data set, the results on the development set is bound to be better than that is expected on new test instances. In a way, over-tuning on a development set is likely 'overfit' the hyperparameters. In such cases the results on a *test set*, which ideally is used only once to report the results, is more objective for comparing different models' performances on the task. This is particularly common in research, where it is also a common practice to use a designated test set that facilitates the comparison of the methods suggested by different groups or people.

Comparing with a baseline or a state-of-the-art model allows us to interpret our performance metrics. If we observe an improvement over an appropriate baseline, then we can be certain that our model is doing something (more) useful. However, there is one more question often neglected in CL/NLP literature. The question is whether the improvement we observe can be by chance or not. This is where *statistical significance tests* are used in many areas of research. In natural language processing, the test set sizes we use are generally large, as a result even small differences tend to be statistically significant. However, it is in general a good idea to test the differences explicitly. We will not discuss the statistical significance testing in this class. It is a very well established field. When comparing the performance difference between two (or more) models, it is important to use an appropriate procedure to show that the differences observed are unlikely to be by chance.

6.4 Summary

This short lecture reviewed a few topics/practices about evaluating machine learning models. Although brief, these are very important for any applied ML research and practice. As Richard Feynman famously put it, "The first principle is that you must not fool yourself and you are the easiest person to fool." The lesson from the present lecture is trying to make sure not to fool ourselves when we evaluate our ML systems. We discussed only some of the tools or practices for this purpose. Most machine learning text books (e.g., Hastie, Tibshirani, and Friedman 2009; Bishop 2006) cover the topics discussed in this class.

Bibliography

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN: 978-0387-31073-2.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer series in statistics. Springer-Verlag New York. ISBN: 9780387848587. URL: http://web.stanford.edu/~hastie/ElemStatLearn/.