Confidence Intervals Handout

Background

Whenever we calculate a performance measure from a test set, we are limited to evaluating the model/hypothesis/learner on *only* that *particular* test set. For this worksheet, we will assume we want to calculate overall accuracy for our performance measure, where accuracy is a proportion calculated as:

$$accuracy = \frac{\text{number of correct predictions}}{\text{number of instances in the test set}} = \frac{\text{sum of diagonal in the confusion matrix}}{\text{sum of all entries in the confusion matrix}}$$

Due to the random selection of the test set, the performance measured does not necessarily reflect the exact performance we would achieve with a different randomly chosen test set. For example, assume that we have two randomly chosen test sets. On Test Set 1, the learned model might accurately predict 90 out of the 100 test instances (for an accuracy of 0.9), but on Test Set 2, the same learned model might accurately predict only 85 out of the different 100 test instances (for an accuracy of 0.85). How do we know which one (if either) is correct? What can we really say about the performance of our learned model?

From this example, we only know that the accuracy of the learned model might be somewhere near 0.85 or 0.9 if we were to provide it with an *arbitrary* new test set that is randomly chosen from the set of all possible data. Recall that what we really want to evaluate is how well the learned model generalizes to all possible data, since it learned only from the training data. Performance on a single data set (or two) is only a rough approximation of how well the learned model generalizes, so we need a better way of measuring performance.

One way to do that is to try to find a *range* of performance values that the model probably achieves. For this, we turn to confidence intervals.

A 95% confidence interval provides us with a range of values where we expect the true performance measure to fall on *any* randomly chosen test set, based on the results from a *single* randomly chosen test set. We calculate the 95% confidence interval as:

$$\hat{p} \pm Z_{0.95} * SE = [\hat{p} - Z_{0.95} * SE, \hat{p} + Z_{0.95} * SE]$$

where

- $\hat{p}$ is a performance measure calculated as a proportion (e.g., our *accuracy* measure from above, but possibly *recall* or *precision*, too) from *one* randomly chosen test set,
- $Z_{0.95} = 1.96$ is a value from statistics (explained in class and in the textbook, stop by office hours if you have any questions) that is a good approximation to use as long as you have $n \geq 30$ instances in the test set,
- SE is the *standard error* of proportion $\hat{p}$, calculated as:

$$SE = \frac{s}{\sqrt{n}}$$

- n is the total number of instances in the test set, and
- s is the standard deviation of the values used to find the proportion $\hat{p}$, calculated as:

$$s = \sqrt{\hat{p}(1-\hat{p})}$$

Not only can we use confidence intervals to build estimates of what performance we should expect on *any* test set from *a single* random test set, but we can also use confidence intervals to determine whether one algorithm significantly outperforms another.

In particular, if two confidence intervals for two different algorithms on the same test set overlap, then we say that there are no statistically significant differences in their performances (at the 1-95% = 0.05 level). This means that even though one might have achieved a higher performance than the other, the two performances were close enough to each other that we do not have sufficient evidence to conclude that one truly outperforms the other. Had we chosen a different random test set, we are not certain that the one that performed better on the first test set would do so again.

On the other hand, if the two confidence intervals for the two different algorithms do *not* overlap, then we say that there is a statistically significant difference in their performances (at the 0.05 level), and we conclude that the algorithm with the higher performance is better than the one with the lower performance on this particular problem.

Example

If we consider the following example confusion matrix:

Predicted Labels			
A	B		
40	10	A	Actual
10	40	B	Label

Then the accuracy on the $n = 100$ test instances is:

$$\hat{p} = \frac{AA + BB}{AA + AB + BA + BB} = \frac{40 + 40}{40 + 10 + 10 + 40} = \frac{80}{100} = 0.8$$

so the 95% confidence interval is:

$$\hat{p} \pm 1.96 * \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} = 0.8 \pm 1.96\sqrt{\frac{0.8 * (1-0.8)}{100}} = 0.8 \pm 0.0784 = [0.7216, 0.8784]$$

Thus, if we were to rerun our evaluation on a different randomly chosen test set, we should expect (with 95% confidence) that the accuracy will fall within this confidence interval. Likewise, we also expect (with 95% confidence) that the true accuracy on all possible data to fall within this confidence interval.

Now, assume we also tested another algorithm that achieved a performance with a confidence interval of

$$0.95 \pm 0.05 = [0.90, 1.00]$$

Since the two confidence intervals do *not* overlap, we conclude that the second algorithm statistically significantly outperformed the first (at a 0.05 level).

Notes

1. As n increases, then $SE = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$ decreases. This means that as your test set size increases, your confidence intervals become tighter and tighter. Thus, the calculated performance measure $\hat{p}$ on the randomly chosen test set is closer to the true performance across all possible data.

2. To achieve larger n during testing when given a fixed data set, we have two main choices:

   (a) Use a larger proportion of of our data set for testing. The downside to this approach is that it reduces the amount of data available for training (since we have a fixed data set with a fixed number of instances), and less data for training means we have less data to learn from, often resulting in a worse learned model.

   (b) Use k-fold cross-validation, where we eventually test over every single instance in the data set, causing the n from our confusion matrix to be the same as the size of the whole data set! The primary disadvantage of this approach is that it requires training and testing k times, which takes longer to run. A second disadvantage is that this approach technically generates k distinct models/learners, and not a single model/learner, so it is less clear which model should be used for prediction in production after testing.