Week 4 - Hyperparameter Tuning

This week you are going to build off-of the logistic regression model to produce a *regularized* logistic regression. In logistic regression, you can tune the bias and variance of your estimator by penalizing the coefficient weights. This allows you to tune the model to your specific data problem to achieve a lower expected prediction error. The typical ways to penalize the coefficients are to use the L1 norm (LASSO), L2 norm (Ridge), or a linear combination of L1 and L2 norms (elastic net). For this problem, you are going to use an L2 penalty.

Now that you are using a regularized logistic regression, you have to tune the regularization hyperparameter λ to the data. This parameter determines the strength of the penalty term, and when $\lambda = 0$ the model is unregularized. By changing λ, you will change the bias and variance of your estimator which will impact your predictive performance. You will use the training data to determine which value of λ works best. This process is called *hyperparameter tuning*.

In order to tune the λ hyperparameter, you need a way to estimate the prediction error on unseen data. Two common ways to do this are bootstrapping and cross-validation. In this exercise, we will be using k-fold cross-validation to choose our hyperparameters. If you are interested in learning more about cross-validation we recommend reading the *scikit-learn* documentation on the topic.

Grid Search

1. `[Ind]` Instantiate a standard scaler to scale each feature in the data to a standard normal distribution. This is essential in regularized logistic regressions because they are *not* scale invariant. If we do not do this, then features with larger values will be more heavily penalized by the regularization term.

2. `[Ind]` Instantiate a new SMOTE object.

3. `[Ind]` Instantiate a logistic regression object like in the previous week, but this time assign it an L2 penalty.

4. `[Grp]` Combine the scaler, SMOTE, and logistic regression objects into a pipeline. Explain why we need to do this rather than scaling and resampling all of the training data before our cross-validation.

5. `[Ind]` Run a grid search cross-validation to evaluate different hyperparameter choices for C in the logistic regression, where C is the multiplicative inverse of the λ parameter we discussed earlier. You will need to decide what values of C to try, the number of folds k, and the scoring metric. We recommend talking to your group mentor about considerations for these choices.

Evaluate Cross-Validation Results

After you run the grid search cross-validation, you should interpret the results. You are going to create a figure to visualize the results like the one shown in Figure 2.1.

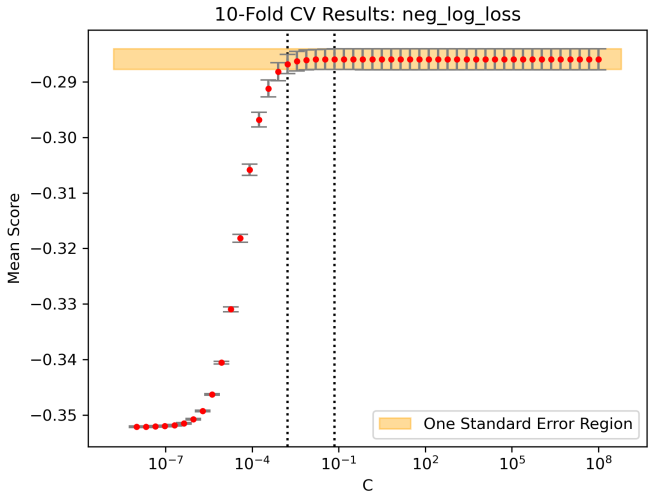

Figure 2.1: Cross-validation results from the logistic regression model. Error bars indicate one standard error of the mean.

1. `[Ind]` Write a function that takes the `GridSearchCV.cv_results_` from your grid search as an argument.

2. `[Ind]` In the function, plot the mean score as a function of C.

3. [Ind] Now have the function return the value for C that achieves the maximum score. If there are ties, choose the C with the smallest value (most regularized).

At this point, you have enough information to pick a good hyperparameter; however, you have only used the mean of the score without incorporating the variance of the score across the folds. Over the next few steps, you will incorporate the variance of the cross-validation results to apply a common heuristic in logistic regression called the one standard error rule.

1. [Ind] In the same function as before, calculate the standard error of the mean score by dividing the standard error of the scores by the square root of the number of folds.[1]
2. [Ind] In the plot, add error bars that show the standard error of the mean scores.
3. [Ind] In the function, in addition to the C that corresponds to the best mean score, return the smallest C that is within one standard error of the optimal C. This choice of C gets you a model that is more regularized than the model with the best mean score but at a statistically similar performance. This is a heuristic called the one standard error rule that chooses to have a simpler model at the cost of slightly worse performance.
4. [Ind] In the plot, add a band that shows the region for the one standard error rule (see Figure 2.1). Pyplot's fill_between may be useful.
5. [Ind] Add the resulting plot to your document and write a few sentences interpreting the plot. You might want to think about how regularization relates to performance and which C you would choose (one standard-error vs maximum).

Inference on the Test Set

Now that you have done hyperparameter selection, you will fit a final model and provide predictions on the test set.

1. [Grp] Pick which C you want to use from your grid search. This should be one of the two values that you return from the function you wrote.
2. [Grp] Fit the model to all the training data with your chosen C and do not forget to scale the data! Explain why you should fit your model to all the training data at this step.
3. [Grp] Report the performance of the model on the validation data using the function you wrote previously.
4. [Grp] Now use the model to predict the labels on the test data and submit the results to your group mentor. Make sure to keep the original order of the test data in your submission.

[1] There are a couple caveats here that should be addressed. First of all, this calculation of standard error is not entirely correct because it assumes that the fold scores are independent; however, due to the overlap in training data across the folds, this assumption is incorrect. Secondly, if you use the "std_test_score" from `cv_results_` then you will be using a biased estimate of the sample standard deviation that underestimates the true population standard deviation. You can partially address this with Bessel's correction.