Tina Han

k-fold cross validation for hyperparameter tuning

Cross validation is a common technique used to make full use of data sets, reduce overfitting, and for hyper-parameter tuning. A common cross validation technique is k-fold cross validation, where k is the number of times we will "fold" or split the data set. Meaning, if we have a data set of size n and we are applying k-fold cross validation, we would split the data set into k validation sets with n/k samples in each set (give or take a sample if n is not divisible by k). Then we run the algorithm k times, training the data on the $n - n/k$ samples not in the validation set, and then test the generated model with the reserved samples. In this way, we make full use of the given data since we would have the option to not reserve a portion of the data only for testing. However, in some cases, we still might reserve a test set and run cross validation on the training set, then after choosing the best parameters retrain the model with the entire training set, then test the model with the reserved test set. For small data sets, cross validation makes full use of the given data. After using cross validation, we choose the best hyper-parameters to use (e.g. learning rate, smoothing parameter) and then train the model using the entire data set. While cross validation has many benefits, it can be computationally expensive since we run the algorithm multiple times.

Using the IMDB dataset, we will explore using cross validation with different classification algorithms. The IMDB dataset contains 50,000 observations with 1,000 features, which has been split into training and testing data sets each with 25,000 observations, of which half are positive and half are negative. Positive reviews were assigned class 1 and negative reviews were assigned class 0. Each feature in the data set represents a word in a review, and is assigned a binary value 0 if the word did not appear and 1 if it did.

The four classification algorithms we will use are logistic regression, naïve Bayes, random forest and gradient boosted regression trees. Each algorithm will be explained in each section. As a simple demonstration of cross validation, we will use 4-fold cross validation to find the training and testing accuracy in each fold and compare them without any parameter tuning. Then we will use cross validation in a grid search to find the best hyperparameters, and use the best parameters to find the best possible model. From the final model we will compare the training and testing accuracy. Then we will compare the four different algorithms' performances.

The first algorithm we used was logistic regression. Logistic regression labels a data point x using a logistic function to compute the log odds of an output based on an input. First we used the default settings for logistic regression and simply applied 4-fold cross validation to compare the testing and training accuracy between each fold. Figure 1 illustrates the results. We can see that the training and testing accuracy (training accuracy around 0.88 and testing accuracy around 0.85 for all folds) is very similar across each fold, which should not be surprising since the IMDB data set is on the larger side.

Logistic regression applies regularization, and in our case this corresponds to the parameter C. For larger C, there is less regularization, and fits the model well to the training data. A lower C balances the fitting error and complexity. We used cross validation to tune this regularization parameter. We found that letting $C = 7.74$ yielded the best model with a training accuracy of 0.882 and testing accuracy of 0.859. The values are close together and we would not consider this model to be overfit. Interestingly, we should note that these scores are approximately the same as when we used the default $C = 1.0$ in our first cross-validation experiment.

Finally, we can look at the feature importance as determined by logistic regression by considering the magnitude of the coefficients in the logit equation. The top ten most important features with their feature importance score is shown in Figure 2. Recall that the features in the IMDB data set are frequently used words. Unfortunately those words are not explicitly given to us, but we can see that the word corresponding to 0.0.599 is the most important feature, followed by 0.0.244, and so on.

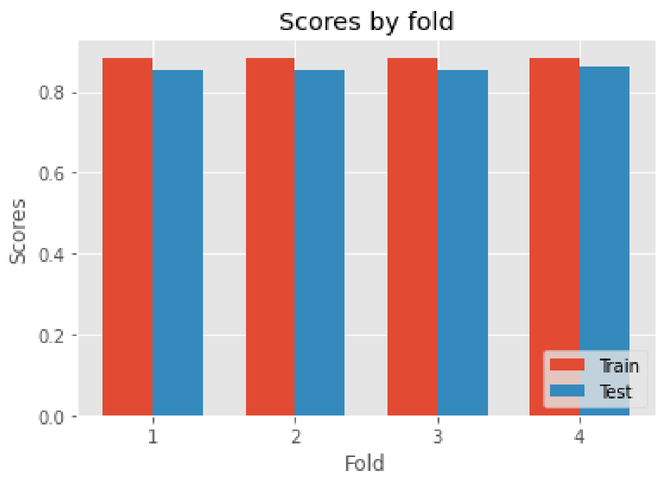

Figure 1: Comparison of accuracy in each fold for logistic regression

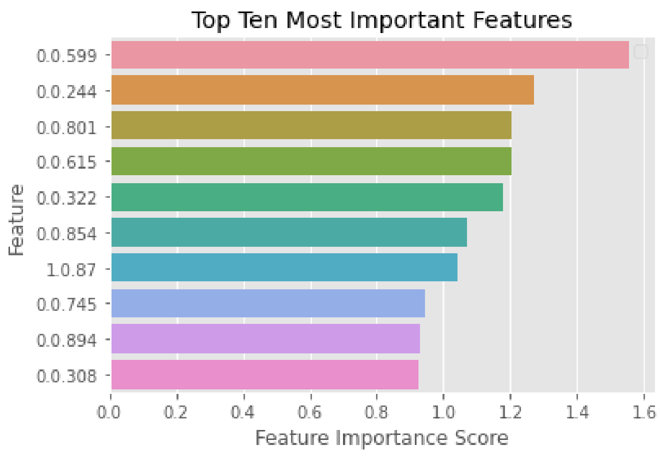

Figure 2: Top ten most important features in logistic regression

Naïve Bayes classifiers are based on Bayes formula given by

$$P(c_j|x) = \frac{P(x|c_j)P(c_j)}{P(x)}$$

where $P(c_j|x)$ is the posterior probability which tells us the probability that a sample belongs to a certain class c_j given the feature values x. And $P(x|c_j)$ is the class conditional probability which represents the probability that an observation belongs to c_j. Then $P(c_j)$ is the prior probability of class c_j which is based on a selected probability density function. In our case, we chose a Bernoulli distribution for our binary feature values. Finally, $P(x)$ is the evidence, or how frequently we will measure a pattern with feature value x. The goal of the naïve Bayes algorithm is to minimize $P(c_j|x)$ based on the training data to create a decision rule. Then a data point is given a class label based on which posterior probability is the greatest, i.e.

$$\text{predicted class} = \arg \max_{j=i,..,k} P(c_j|x)$$

If we have an issue with probabilities equal to zero we can introduce a smoothing parameter (α). In our experiments this is the parameter we fine tune using our cross validation technique in grid search.

We should note that one of the biggest assumptions made in the naïve Bayes classifier is that the data are independent and identically distributed (iid) meaning that they are independently drawn from the same probability distribution. For binary or multinomial data this condition is easily violated, since the IMDB data is binary we must note that the iid condition may not be met.

Similar to logistic regression, we saw that with the default parameters using 4-fold cross validation yielded very similar training and testing accuracy scores, this can be seen in Figure 3. The training accuracy was around 0.82 and testing accuracy around 0.81 for all four folds. When we used 4-fold cross validation in the grid search we found that the best smoothing parameter was $\alpha = 0.01$ which yielded a training accuracy of 0.818 and testing accuracy of 0.815. Again, there is not a large difference between the accuracy using the default $\alpha = 1.0$.

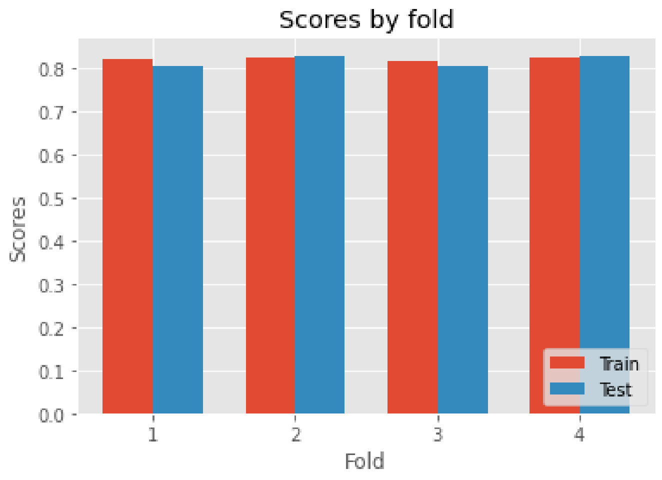

Figure 3: Comparison of accuracy in each fold for naïve Bayes

Random forests are an ensemble method which constructs multiple trees based on subsets of the given data that each "vote" on the classification output. Each tree may overfit on part of the data, but all together they balance one another. The trees are built randomly through a bootstrapping process that randomly selects observations with replacement to build the trees. There are multiple parameters used when building the trees, due to computational limitations, we used grid search and 4-fold cross validation to find the best number of trees to use and max depth, or how many splits each tree has. The more splits, the more complex the tree becomes, as well as more computationally expensive.

First, we used 4-fold cross validation with random forest algorithm with the default parameters and compared the testing and training accuracy. We can see in Figure 4 that across the folds the training and testing accuracy are similar; however, the training accuracy and testing accuracy are quite different. The training accuracy across each fold is 1.0 and have a training accuracy around 0.82. The high training score in comparison to the testing score indicates some overfitting and and that with the default parameters, we have an overly complex model.

Using 4-fold cross validation with the grid search we chose the number of trees to build (n_estimators) and how big the tree could be (max_depth) as the two hyperparameters to fine tune. There were other hyperparameters we could adjust but we decided to only use those two to save on computation time. The grid search showed that the best n_estimators and max_depth was 200 and 100, respectively, which were the largest values we ran. This yielded a training accuracy of 1.0 and testing accuracy of 0.828, which was marginally higher than with the default settings. However, there is likely still some overfitting, and the model is too complex since the training accuracy is still significantly higher than the testing accuracy.

The last thing we checked was the feature importance from the random forest. Figure 4 shows that a few features stood out as the most important. The most important features were 0.0.37, 0.0.178, 1.0.42, 0.0.354, and 0.0.295, again we do not know what words these correspond to. We also note that none of these features appeared in the top ten most important features for logistic regression.

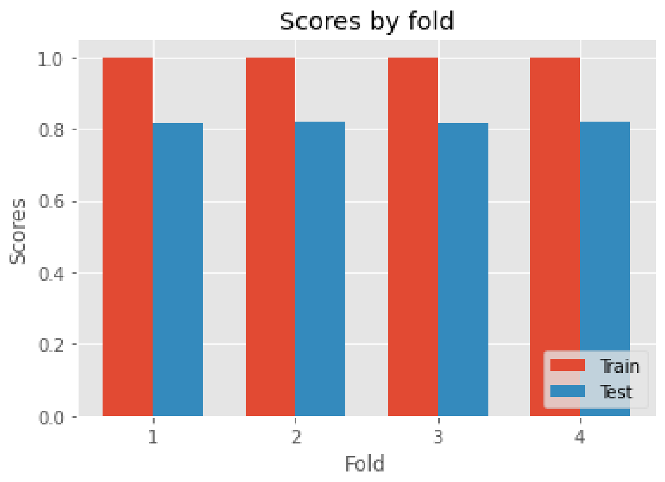

Figure 4: Comparison of accuracy in each fold for random forest

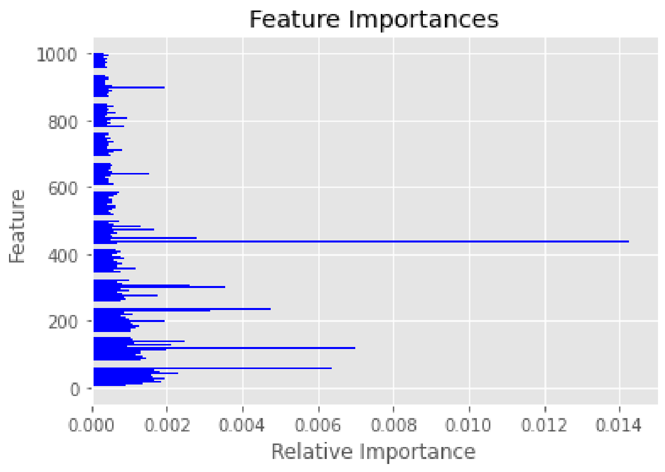

Figure 5: Feature importance in random forest

Similar to random forests, gradient boosted trees are an ensemble method which utilize decision trees. Instead of building each tree independently of one another gradient boosted trees build trees consecutively where each tree corrects the mistakes of the previous one. Also, instead of randomization the algorithm uses heavy pre-pruning with trees of max depth between 1 and 5 which allows for less memory usage and faster predictions. Gradient boosted trees have many of the same parameters as the random forest, but to save on computational expense the only two parameters we will tune are the number of trees (n_estimators) and the learning rate which controls how strongly a tree tries to correct for the previous tree. The higher the learning rate the stronger the correction, but it also raises the complexity of the model.

Like the previous three algorithms we used cross validation with the default parameters to compare the testing and training accuracy across the four folds. We can see in Figure 6 that the training and testing accuracy were similar for all four folds with a training accuracy of about and a testing accuracy of 0.83 and testing accuracy of 0.81. There is no obvious overfitting, and all four models perform about the same.

Next, we used cross validation with a grid search to find the best number of trees (n_estimators) and best learning rate (learning_rate). The best parameters were found to be learning_rate=0.3, n_estimators=100 which were the highest parameter values we tested. This gave a training accuracy of 0.87 and testing accuracy of 0.844, which is an improvement over using the default parameter values. The default n_estimators was 100, but the default learning_rate was 0.1. Thus a bigger allowance for correction yielded better results.

We also checked the feature importance which is shown in Figure 7. Interestingly, the most important features for gradient boosted regression trees were the same as in random forest: 0.0.37, 0.0.178, 1.0.42, 0.0.354, and 0.0.295.

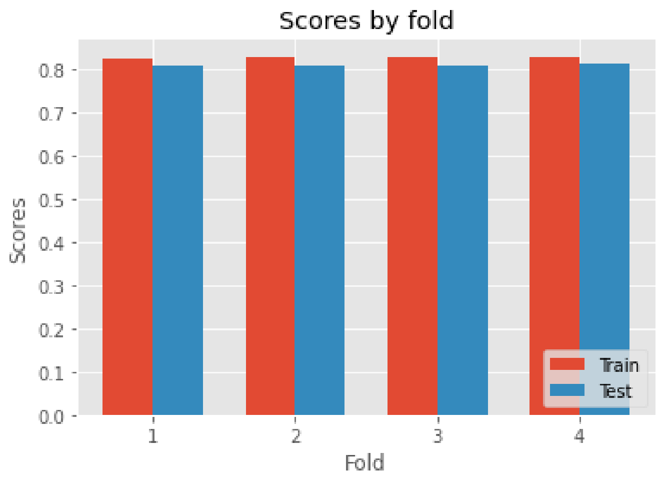

Figure 6: Comparison of accuracy in each fold for gradient boosted regression trees

Table 1 gives the training and testing accuracy for each of the best models from each algorithm. We can see that, in general, they all perform similarly but logistic regression has the best testing accuracy. Comparing the computational complexity of the models makes it even more clear that, with the best parameters we found, logistic regression was the best algorithm for the data. Logistic regression and naïve Bayes were fastest to run a grid search and build the models. Random forest and gradient boosted tree building and grid search had the longest computation time and took up to 100% of the machine's CPU usage for minutes at a time. Furthermore, it is clear that the random forest was overfitting to the training data, and was too complex. The naïve Bayes not only performed the worst but it is also likely that the iid assumption was violated so it would not be a good fit for the data. Overall, on this data set we have seen that simple is better with logistic regression.

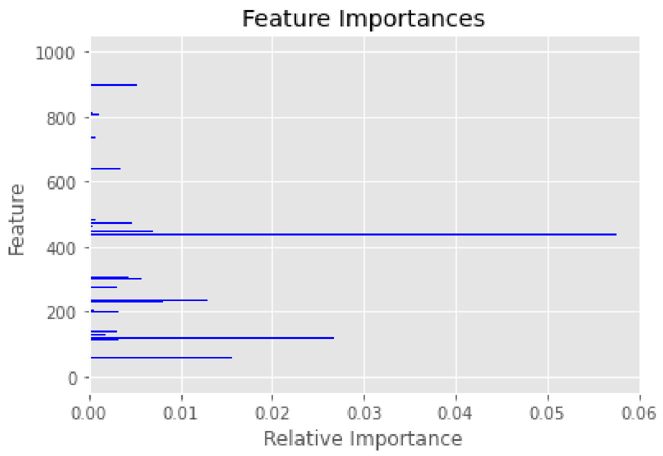

Figure 7: feature importance in gradient boosted regression trees

	Logistic Regression	Naïve Bayes	Random Forest	Gradient Boosted Trees
Training Accuracy	0.882	0.818	1.0	0.870
Testing Accuracy	0.859	0.815	0.832	0.844

Table 1: Train and test accuracy for the four classifiers