**Midterm 2**

J. Jedediah Smith

1. A dataset of 10 different types of yeast (single-celled fungi) is available. This dataset contains a total of 1486 instances or patterns. Each instance is made up of 8 numeric attributes (a1, a2,…, a8), which represent the characteristics of the yeast, and 1 output attribute that indicates the type of yeast (TypeA, TypeB,…., TypeK). The number of samples available for each type of yeast is as follows:

|  |  |
| --- | --- |
| **Yeast Type** | **Number of Samples** |
| A | 463 |
| B | 429 |
| C | 244 |
| D | 163 |
| E | 51 |
| F | 44 |
| G | 37 |
| H | 30 |
| I | 20 |
| K | 5 |

You want to build a model to classify the yeast types based on their attributes:

1. What type of algorithms you could use to solve this problem? Describe their advantages and disadvantages.

Since we want to predict Yeast Type, which is distinct rather than continuous, we will want a machine learning model that can do classification. It could be something purpose built or with dual capabilities—either should work.

Theoretically, it would be possible to use something simple like Native Bayes or k-Nearest Neighbor. However, given their simplicity and the assumptions they make, I would hesitate to use them for something as complex as this, where we have 10 difference yeast types. For example, Native Bayes assumes the independence of attributes, which in a scenario like this, will probably not be the case. However, it is known to work well with small training datasets, which could be advantageous here. If we were constrained by our resources, this might be better than nothing. But for the sake of an example, I think we can do better.

Superior choices would be Random Forest and XGBoost, the latter of which is at the forefront of machine learning algorithms. They are both capable of classification, considered to fairly accurate thanks to boosting, and have more parameters we can tweak to improve performance. However, they are also more complex to set up and have proved to be more computationally demanding, on account of their decision trees. Also, XGBoost is known to not perform as well with small number of samples sizes.

1. For each type algorithms that may be used, indicate a possible architecture and parameters that need to be taken into account.

**k-Nearest Neighbors**

The main parameter to look at here is k, the number of groups considered. Its architecture could look something like this:

knn(train = data\_train, test = data\_test, cl =

train\_labels, k = 20)

**Naive Bayes**

The main parameters to look at here are as follows.

* fL = Value for Laplace factor correction, ensures non-zero probability.
* usekernel = Enable to use kernel density estimation instead of gaussian.

The general architecture for a Naive Bayes model could look something like what is outlined below. These aren’t parameters we would want to grid search.

nb <- naiveBayes(data\_train, train\_labels,

fL=0.1, usekernel=FALSE)

test\_pred <- predict(nb, data\_test)

**Random Forest**

The main parameter to look at here are as follows.

* ntree = Number of individual trees.
* mtry = Number of features to randomly sample at each node.
* nodesize = Minimum number of cases allowed in a node.
* maxnodes = Maximum number of nodes allowed.

The ideal architecture would utilize some sort of grid search or hyperparameter tuning. This involves setting a range for each parameter ahead of time, then testing to find the best. For the sake of brevity, I’ve outlined the architecture of a simple random forest function here, but a more complicated structure could be utilized to optimize results.

rf <- randomForest(predictors, data=data\_train, ntree=300,

mtry=12, nodesize=5, maxnodes=12)

test\_pred <- predict(rf, data\_test)

**XGBoost**

The main parameter to look at here are as follows.

* eta = Value of the learning rate, between 0 and 1. Lower better, but takes longer.
* gamma = Minimum splitting by which a node must improve the predictions.
* max\_depth = Maximum levels deep that each tree can grow.
* min\_child\_weight = Minimum impurity needed in a node before another split.
* subsample = Proportion of cases to be randomly sampled for each tree.
* colsample\_bytree = Proportion of predictor variables sampled for each tree.
* nrounds = Number of sequentially built trees in the model.
* eval\_metric = Type of residual error/loss function we’re going to use.

Again, the ideal architecture would use grid search or hyperparameter tuning. This basically involves setting max and min values for each variable, then testing them to get the best ones. I’ve just outlined the architecture of a simple XGBoost function here, but a more complicated structure could be utilized to optimize results.

xgb <- xgboost(data=data\_train, label=train\_labels, eta=1,

gamma=5, max\_depth=5, min\_child\_weight=10,

subsample=1, colsample\_bytree=1, nrounds=20,

eval\_metric=merror)

test\_pred <- predict(xgb, data\_test)

1. RESEARCH QUESTION: Explain the procedure to follow to obtain the best training and test data. (Hint: Notice the number of samples for each yeast type)

The ideal training data needs to be both random and representative. However, if we randomly selected our sets from the total pool of data at our disposal, there is a very real chance we won’t get any examples of the low abundance yeasts like Yeast Type K in one of our sets. This is a problem, because the model won’t be able to predict cases of “K” so easily if it doesn’t realize “K” is a type that appears in the data.

One solution to this problem is to randomly sample within each Yeast Type. For example, rather than taking a random 70% of the total data set all at once, take a random 70% of each individual yeast type. This ensures that the data sets will both be random and representative. The “K” yeast will be in both training and test data, and the samples of K for each are randomly selected.