**SVM Predictive Models (Jared)**

**A NOTE ON THE METRIC FOR ASSESSING MODEL QUALITY**

Before diving into the model constructions and comparisons for this method, it’s worth discussing how we assess model quality. In classification, the most common and accepted metric is PCC (percent correctly classified) or prediction error rate (they’re measuring the same thing since 1 = PCC + error rate).

However, for this problem we care much more about identifying customers that are likely to buy whatever product is being marketed. That way, telemarketers can spend their time calling those customers rather than wasting their time on customers that may or may not buy. If we are able to create a model that identifies the “likely to buy” customers, we are doing our job. While a low overall error rate is nice, it’s very secondary to producing a model that gives a low error rate for identifying buying customers.

Therefore, when assessing the different SVM models we will be comparing “percentage of buyers correctly classified” in the test data set, rather than the prediction error on the test data set.

**DATA PRE-PROCESSING**

After having read in the data, I wanted to see if we had a class imbalance issue in our response variable. Lo and behold, we have about 11% of one class (those who bought the product) with the other 89% being the other class (those who didn’t buy the product).

Next, I randomly split the data into a training set (80% of the original data) and a test set (20% of the original data). To be safe, I made sure we still had similar class imbalances in the training and test sets: this was the case, with about 11% still being the minority class in both training and test.

Having seen problems with class imbalance before, I thought it might be a good idea to build models on data other than the original training data. Specifically, a data set that has “upsampled” the original training data, and a data set that has “downsampled” the original training data. In this context, upsampling means keeping all of the majority class observations, and then randomly sampling (with replacement) from the minority class observations until there are equal proportions from each class. Therefore, there will be some repeats of the minority class observations, and the upsampled data set will be larger (more observations) than the data set from which it was constructed. In this context, downsampling means keeping all of the minority class observations, and then randomly sampling (typically without replacement) from the majority class until there are equal proportions from each class. Therefore, there will be some observations from the majority class that are omitted from the downsampled data set, and the downsampled data set will be smaller (fewer observations) than the data set from which it was constructed.

I created both an upsampled version of the training data and a downsampled version of the training data.

**LINEAR KERNEL SVM MODELS**

Parameter tuning:

For linear kernel SVMs there is only one parameter to tune: C. This parameter controls the width of the margin between our separating hyperplanes. Since our original data is fairly large in the context of parameter tuning for an SVM (45,000 observations), I used a subsample of the data to tune parameters. In order to ensure I had a decent number of “buyer” observations (the minority class), I took this subsample from the downsampled training data. Through some iterative manual grid searching, I found the parameter of cost = 1.25 to give the best 10-fold cross-validated accuracy. This was used for constructing the three models below.

Model 1: Trained using original training data

This model took about 4 minutes to run, and gave a prediction error rate of 0.1029369 when predicting onto the test data. This is only slightly better than classifying all observations as “not a buyer”, which would give a test error of just over 0.11. Also, this model did a very poor job of identifying buyers in the test data set: it correctly identified only 18.65079% of buyers.

Model 2: Trained using upsampled training data

This model took about 12 minutes to run, and gave a prediction error rate of 0.1686574 when predicting onto the test data. This is actually worse than just classifying all observations as “not a buyer.” However, as discussed above, what we truly care about is the percentage of buyers correctly classified. Although this model had a poor test error, it did much better at identifying the buyers in the test data set, correctly classifying 84.3254% of them!!

Model 3: Trained using downsampled training data

This model took about 30 seconds to run, and gave a prediction error rate of 0.1640124 when predicting onto the test data. Again, a poor overall test error rate. Similarly to the upsampled model though, it did fairly well at classifying buyers, correctly identifying 84.02778% of them in the test data set.

**GAUSSIAN KERNEL SVM MODELS**

Parameter tuning:

For Gaussian kernel SVMs there are two parameters to tune. In the R package I used, they are “cost” (the C value in the SVM mathematical construction) and the “gamma” parameter (a scaled version of the sigma value in the SVM mathematical construction).

Just as I did for linear kernel SVM tuning, siince our original data is fairly large in the context of parameter tuning for an SVM (45,000 observations), I used a subsample of the data to tune parameters. I used the same 500-observations subsample from the downsampled training data. Through some iterative manual grid searching, I found the parameter of cost = 1000 and gamma = 0.001 to give the best 10-fold cross-validated accuracy. These parameters were used for constructing the three models below.

Model 1: Trained using original training data

This model took about 5 minutes to run, and gave a prediction error rate of 0.09865074 when predicting onto the test data. This is the best test error of all models. BUT, this model did a very poor job of identifying buyers in the test data set: it correctly identified only 28.0754% of buyers.

Model 2: Trained using upsampled training data

This model took about 20 minutes to run, and gave a prediction error rate of 0.1633488 when predicting onto the test data. However, as discussed above, what we truly care about is the percentage of buyers correctly classified. Although this model had a poor test error, it did much better at identifying the buyers in the test data set, correctly classifying 87.40079% of them!!

Model 3: Trained using downsampled training data

This model took about 30 seconds to run, and gave a prediction error rate of 0.1643442 when predicting onto the test data. Again, a poor overall test error rate. Similarly to the upsampled model though, it did fairly well at classifying buyers, correctly identifying 86.60714% of them in the test data set.

**OVERALL VERDICT FOR SVM MODELS**

* From these results, we can see that using training data constructed through a subsampling technique (either upsampling or downsampling) gives far better results than just using original training data. By “far better results” in this context we’re referring to accurate classification of buyers in the test data set, not overall test error rate.
* For both kernels, the upsampled and downsampled training data gave similar results for error rate and buyer classification rate.
* Time consideration: if the goal of this project was to get a decent result in the shortest amount of time, the best result is to use a linear kernel SVM constructed with downsampled data. Only tuning a single parameter (“cost”) was far faster than having to tune two for the Gaussian kernel, and it gave very similar buyer classification accuracy. Also, training a model using the downsampled training data took a negligible amount of time, especially when compared to training on the upsampled data. The downsample-trained and upsample-trained models gave very similarly buyer accuracy, but the upsample-trained models took 20 to 40 times longer to run. On data sets far larger than this, using a smaller subsampled training set might be the only solution that’s computationally feasible. If faced with a problem similar to this in the future, my first SVM model attempt would be trained on some kind of downsampled data and would use a linear kernel. Depending on the nature of the data, some additional accuracy might be gained by using upsampled training data or using a different kernel and tuning parameters, but this initial model would likely give a good baseline result.