Gideon Blinick

Capstone 1: In-depth Analysis (Machine Learning)

We want to predict credit card default using the 23 features we have available to us containing information about customer demographics and customer financial information.

To do this, we will test 6 different models of classification to see which produces the best results: Logistic Regression, K Nearest Neighbors (KNN) , Support Vector Machine (SVM), Decision Tree, Random Forest, and Naïve Bayes

Before fitting each model to the data we must do two things:

1) Preprocess the data;
2) Split the data into training and test sets

These two steps are the same for every model, and therefore we need only do them once. After completing them we must:

3) Instantiate the model or estimator (this can be combined with the first step into a scikit-learn Pipeline() object);
4) Specify the Hyperparameter space over which we are optimizing our hyperparameters;
5) Create a GridSearchCV() or RandomizedSearchCV() object from the estimator and hyperparameter space;
6) Fit the CV object to the training set (this takes the most time of any of the steps);
7) Create a list of predictions by using the .predict() method on the estimator with the test set;
8) Score the model

We begin by preprocessing the data. We make good use of scikit-learn's Pipeline() object and preprocessing features here. We divide our features into 2 lists, one containing categorical features and the other containing numeric/continuous features. We then apply appropriate transformations: to the continuous features, we apply the MinMaxScaler() transformation so that all values are rescaled to be between 0 and 1; to the categorical features we apply the OneHotEncoder() transformation. These transformations are important because many classifiers like KNN use distance metrics to make classifications. We wrap each of these Pipeline transformations in a ColumnTransformer() object so that they can be applied to their appropriate features.

We split the data using scikit-learn's train_test_split() function. We are now ready to fit our models to the data.

Before doing that, it is important to clarify how we will score each model. The most natural and intuitive metric is accuracy. However, there are 2 problems with using accuracy alone:

1) Our dataset features an imbalance of about 78% to 22% where 78% of the records are labelled as 'no default' and 22% were labeled as 'default'. In classification problems, the greater the imbalance in the dataset, the less informative accuracy is as a metric, since high accuracies can be achieved by predicting the more common label.
2) In the context of our problem, not all mistakes made by a model are equal. There are 2 kinds of errors we can make:
    a. False positives: predicting a person will default when he/she won't

b. False negatives: predicting a person won't default when he/she will

In our problem, our key stakeholder is any credit-extending institution. For such institutions, the cost of a false negative is far higher than the cost of a false positive. Therefore, in our scoring system, we should assign greater value to models that produce fewer false negatives at the cost of more false positives. This trade-off is well-known in classification problems as precision-recall trade-off. In our case we want to maximize recall (the ratio of correctly predicted defaulters to the number of actual defaulters) at the cost of precision (the ratio of correctly predicted defaulters to the total number of predicted defaulters).

So we now know that recall should be our main scoring metric. In addition to accuracy, precision, and recall, we will also keep track of the AUC score for each model, which tracks the trade-off between true-positive rate (higher is better) and false-positive-rate (lower is better)

Finally, before running our models, we create a dummy model that will predict 'no default' for every record in the testing set so we have a natural baseline with which to compare the accuracy, precision, recall, and AUC of our models.

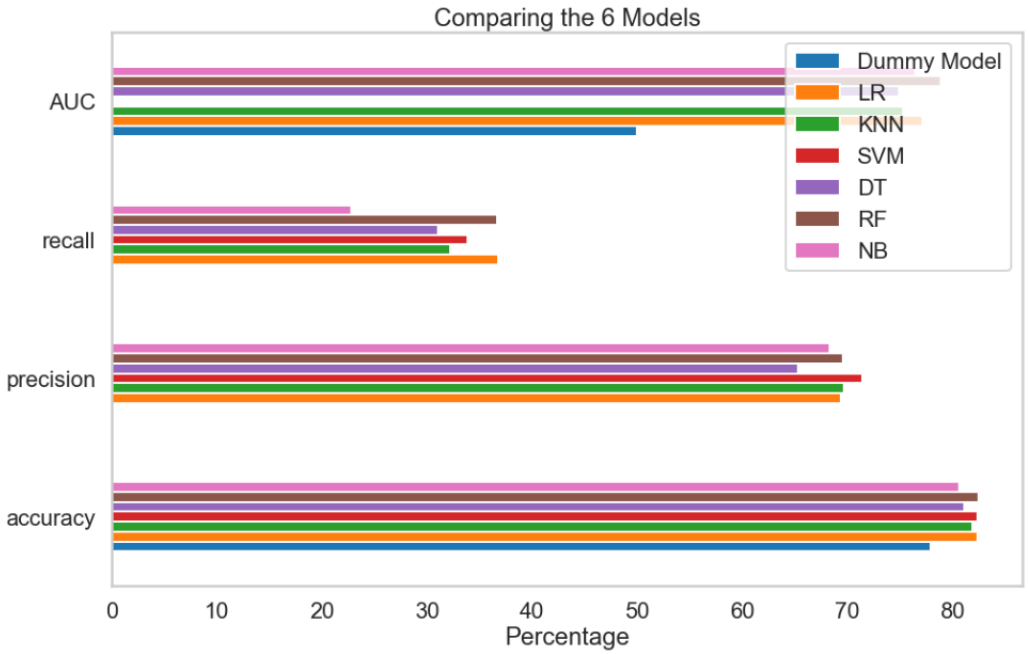
We fit the models to the training data, following the steps above, predict on the test set, and score each of the models, all the while appending each model score to a table tracking each model and its scores.

After fitting the models and scoring them, we obtain a table of our models and their scores:

	Dummy Model	LR	KNN	SVM	DT	RF	NB
accuracy	0.779	0.824	0.819	0.824	0.811	0.825	0.806
precision	0.000	0.694	0.697	0.714	0.653	0.696	0.683
recall	0.000	0.368	0.322	0.339	0.311	0.367	0.228
AUC	0.500	0.771	0.753	NaN	0.749	0.789	0.765
Time to Train	0.045	77.851	278.351	971.990	9.830	125.211	0.062

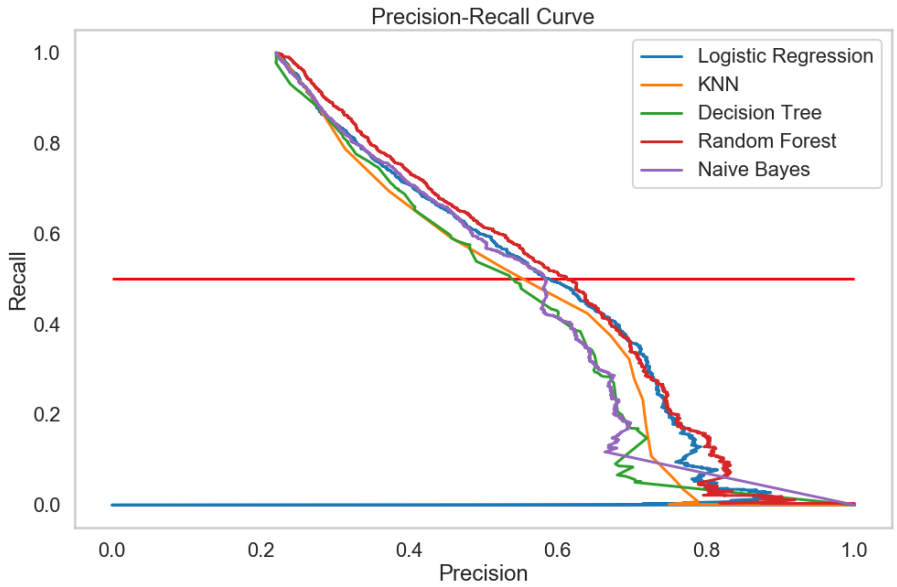
Looking at the results, we can make two key points. First, that all of our models performed better than the dummy model. The dummy model had precision and recall scores of 0 since it never predicted default and had an accuracy of about 78% since that was the percentage of non-default records in our dataset. In contrast, our models achieved accuracies ranging from 80.6% for Naïve Bayes to 82.5% for Random Forest. This means our models are working since they are finding some means by which to distinguish defaulters from non-defaulters. The second point is that by recall alone, Logistic Regression performed the best (36.8%) by a hair over Random Forest (36.7%).

The following chart also provides a helpful way of looking at the performance of the models:



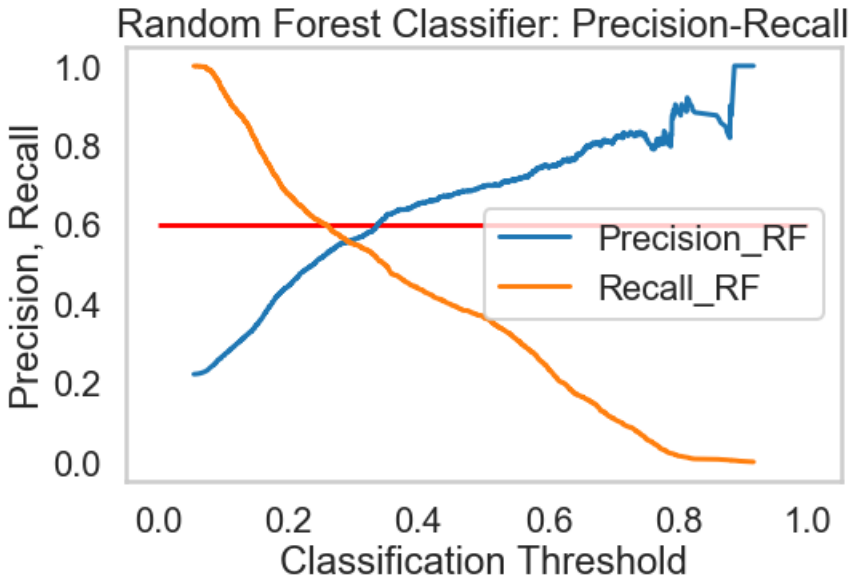
From the table and chart alone, it appears that either Logistic Regression or Random Forest is the best classifier. It is hard to give one the edge since they are so close in all metrics. However, the table and chart don't tell the whole story. What we also need to do is look at a precision-recall graph which shows how recall changes with precision. Different threshold values for the probability at which we classify a customer as defaulting or not will result in different points on the curve. A precision-recall curve gives us a way of standardizing precision so that we can see how each classifier performs in terms of recall at that level of precision.

When we plot precision-recall curves for each of our models (except for SVM – this cannot be done since SVC() in scikit-learn has no predict_proba() method), we obtain the following figure:



We see that the Random Forest model produces the highest recall for a given precision over all the models except for values of precision between about 0.63 and 0.78. In that range, Logistic Regression actually produces higher values of recall. Because the values for precision within our table fell within that range, Logistic Regression had a higher recall then Random Forest in our table. This can be changed by altering the threshold probability value for prediction so that we end up at different points on our precision-recall curves. This will allow us to attain higher values of recall, which we desire.

The figure below shows us how precision and recall change as a function of threshold for the Random Forest Classifier:



We see that at classification values below about 0.3, recall is greater than precision. Therefore, we should change the probability threshold used by our classifier from the default level of 0.5 to achieve greater recall. When we do this and use a level of 0.25, we achieve a recall of 60.6%, precision of 51.5%, and accuracy of 78.6%. Clearly, then, our accuracy (and precision) have suffered as a result of using a new, lower value for threshold, which makes sense. On the other hand, our recall is way up, which is what we really care about. We can also output the confusion matrix that accompanies using a threshold value of 0.25. We notice that it contains more false positives than false negatives, as it should:

PREDICTION TRUE	pay	default	Total
pay	5872	1137	7009
default	785	1206	1991
Total	6657	2343	9000

We conclude that Random Forest performed best. By altering the probability threshold for classification, we are able to achieve higher rates of recall than we are able to with all other classifiers. Since this is our key scoring metric, this makes Random Forest our best model for the particular challenge of classifying customers as defaulting or not.