**Final Project Technical Report, Classifying House Prices**

**Introduction**

The classification project that we partook in was certainly one of the most rewarding tasks of the year. That being said, it was quite difficult. This may have been, at least in part, because of the dataset we chose to classify. Our table is essentially a collection of houses that contains listings of various real estate attributes. So, what was the task we chose? We decided that it would be a good idea to attempt to predict house prices! We discovered a great many things along the way, including how bad of an idea it is to run Naïve Bayes over all the attributes in the dataset. We also found out that ensemble classifiers, interestingly enough, have the highest accuracy, and support vector machines (at least with this data) have a far less impressive accuracy.

**Data Analysis**

The housing dataset that we classified is a CSV file from Kaggle. It contains a total of 1460 instances and has a total of eighty attributes, not including the house ID, which we ultimately decided to leave unused. The attribute that serves as the label in our project is the very last column of the dataset, and that is the one titled “Sale Price”.

There are a few curious statistics regarding this dataset. First off, the sale prices are almost all in the lower range. This was easy to see once we discretized the dataset, since almost all of the instances (actual, not predicted) were placed in the lowest bins by our discretization algorithm.

**Classification Results**

So, what is the amazing classifier that we elected to use on this dataset? We chose the one and only support vector machine from SKLearn. The general approach we used to classify the data with our support vector machine was similar to the strategy we employed in our other assignments throughout the semester.

First, our classifier cleans the data so that there are no “NA” values. We ultimately elected to do this in several steps, which are listed below. The program eventually replaces the NAs with the mode of the various attributes.

Next, the classifier discretizes the table. This is also a multi-step process, and we eventually had to write an ugly boolean function to implement it. The function is\_continuous() determines if an attribute is continuous, and if this results to true, the attribute is converted to a categorical one. Since the class label itself is a continuous attribute, we had to include sale\_prices = np.log1p(sale\_prices) in order to normalize the prices so they could be more effectively fit. Lastly, we chose to split the continuous attributes into ten bins, mainly because we wanted a number that wasn’t too large or too small.

Since implementing all of the features of a support vector machine would be ridiculously complicated, we ultimately decided to just use SKLearn’s library to implement one. Even so, we still feel that we learned a lot about this classification technique. There are two key steps. The first step is fitting the data, and in order to do this, the program creates an x and y axis, somewhat like a graph. The y-axis contains the class, and the x-axis contains the rest of the attributes. Then, the data is fit with a call to svm’s SVC function. An important parameter for SVC is the C parameter, which penalizes slack variables that can lead to miscalculation. A higher value of C leads to more variables being penalized, so a well-chosen C can make or break this classifier’s performance.

The second step that the program takes to classify the data is to fit the data using the clf, which it basically does by generating a best fit line through the new data points.

To test our classifier’s predictive ability, our program uses the classic approach of stratified k-fold cross validation, in which k folds of the original data are generated, and then each one is used in turn as the test set for a total of k “runs” of the classifier. The accuracy of each run is recorded, then at the end, the average accuracy is calculated. The average accuracy is the metric we used to rate our classifier’s performance.

We compared our new classifier to several old ones, including a Naïve Bayes classifier, a KNN classifier, and an ensemble classifier. In Naïve Bayes, the classification of instances is based on probabilities, and probabilities require a huge amount of computation. We were forced to implement this algorithm with significant attribute selection, since the program took more than an hour to run Naïve Bayes without it.

Our KNN classifier turned out to be the second slowest, since it also requires tons of computation. The basic idea is this: for each instance to be classified, calculate the k nearest neighbors amongst the training set using the Euclidean distance algorithm, and then use a majority vote to pick the class label.

For our ensemble, we chose to use a random decision forest. Our ensemble generates N trees, takes the M most accurate of those trees, then votes from amongst all of their predictions in order to choose the class label. We chose to use the TDIDT method to implement this algorithm.

Our program has some surprising results regarding the accuracies of these classifiers. We both expected the support vector machine’s accuracy to be the highest, but that was not the case. Our support vector machine has the *lowest* accuracy of all the classifiers, coming it at a measly 60 – 70 percent. Naïve Bayes and KNN come next, and are nearly tied at about 82 percent accuracy, while the ensemble classifier has by far the best accuracy at around 90 percent.

**Conclusion**

Ultimately, our project consisted of cleaning the dataset, discretizing continuous attributes, creating a plot with SKLearn’s support vector machine, fitting the data to that plot for predictions, and finally, using stratified k-fold cross validation to compute the accuracy. The dataset proved to be challenging for us in several ways: the number of attributes, the presence of NA values, and lastly, the need to normalize the sale price. There was ultimately more work we had to do *before* implementing the classifier than there was for implementing the classifier itself. Our classifier’s performance wound up being less than satisfactory, and we definitely expected its accuracy to be higher. The low accuracy could either be a result of this particular dataset, or it could also be due to our lack of experience using support vector machines. We were unsure exactly how to choose the best C and gamma values for the SVC function, and we believe that if we had more time, and we figured out how to optimize these values, the support vector machine’s accuracy could improve drastically. But at the end of the day, we still learned a lot, and now we have a basic understanding of a new classifier to take with us into the real world.