Project Overview:

For this project I chose to use a dataset from AirBnb which describes listing data within the Austin, TX area. From this dataset I chose to create two different classification problems. One, a regression problem where I would try to determine the listing price based on a number of features. For problem two, I chose to create a binary classification problem around whether or not a listing was available for booking in the next 90 days.

Data Summary:

The original dataset [1] contains 74 informative and uninformative features. From these 74 features I pruned them to 12 features that I felt were both informative and interpretable. These were:

Feature Name	Data Type	Description
host_is_superhost	bool	Whether a host is a superhost
host_neighbourhood	string	Austin neighborhood listing is in
latitude	float	Latitude of the listing
longitude	float	Longitude of the listing
property_type	string	Type of property (House, apartment, bus, yurt, etc.)
room_type	string	Type of room (room, whole house, whole apt., etc.)
amenities	list of strings	Amenities in listing (bbq, pool, shampoo, etc.)
price	float	Price of the listing
bedrooms	int	Number of bedrooms
beds	int	Number of beds
review_scores_rating	float	Occupant review score rating (0-100)
availability_90	int	Number of available days in the next 90 days

Table 1. Summary of dataset features

From these 12 features, I was able to engineer an additional 28 features and remove some of the originals which would not be useful for our classification task.

Amenities:

I generate dummy variables for different amenities due to the cardinality of the features (389 unique strings). I selected the top amenities that were most associated with differences in price and that still had a significant sample size in the amenities list. These ended up being: baby, fire, bbq, pet and pool.

Property Type:

Property type also had high cardinality with 53 different values. Several of these property types were under represented in the dataset and I decided to remove the listings if there wasn't a large enough sample size behind the feature. I used an arbitrary cutoff of 100 listings per property type. This in turn left me with 9 unique property types: Entire guest suite, Entire townhouse, Entire bungalow, Private room in apartment, Entire guesthouse, Private room in house, Entire condominium, Entire apartment, and Entire house. These were then converted into boolean dummy variables.

Latitude and Longitude:

As discussed in the lectures, even though latitude and longitude are numerical the numbers they represent are much more complicated and there would be little to no relation between them and our classification problems. As such, I took these features and engineered two separate types of features from them. First, I measured the distance (in miles) between the listing latitude and longitude to the center of Austin's downtown. My intuition for this, would be that there is more demand (and a higher price) for listings closer to the city center. Second, I transformed the features into a geohash. A geohash encodes a geographic location into a short string of letters and digits. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape [2]. I used a geohash of precision 5. From this feature, I kept the top 12 most populated geohashes and converted them into dummy variables.

Bedrooms:

Bedrooms also had a high cardinality (14). With fewer samples as the number of bedrooms increases. I kept the original feature, however, I also decided to bin the original feature into five bins to induce some more linearity within the dataset. These were, zero bedrooms, one bedroom, two bedrooms, three bedrooms, and greater than three bedrooms.

Summary:

As we know, the goal of both of our tasks is to approximate a function. In both these projects we are trying to learn a function which can relate our 40 features to our target variable. Our models can't do much with strings, so all strings have been transformed to numerical features. It's important to note that some of our models will aim to model a linear relationship. As such, these models will work better with linear features. One of the reasons we engineered dummy variables is due to the fact that these are linear. Additionally, we also now have several features which are binned, which is also a linear relationship. In the following sections all data is standardized before being used for modeling and all models are cross-validated.

Predicting the Price of a Listing:

Description:

Different listings will go for different prices determined by a number of features that are represented in our dataset. It is also important to note that these prices could be influenced by a variety of outside factors that I did not control for. These would include such things like weather, seasonality, events occurring, etc. Using the dataset described above. My goal was to predict the logged price of a listing. I used the popular machine learning library sklearn [3] for all analysis.

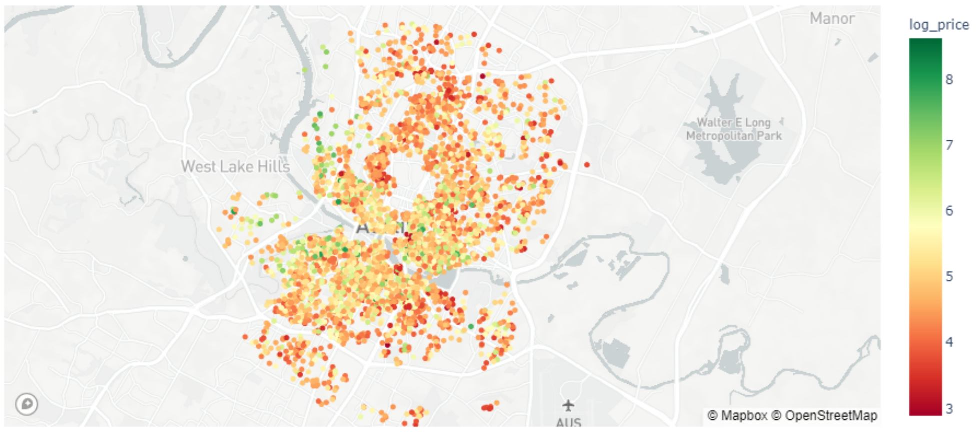

Figure 1. AirBnB listings colored by log of listing price

We have two main types of features: simple numerical features and categorical features which have been transformed into dummy variables. One can see an example of the categorical features in Figure 2, and the numerical features in Figure 3.

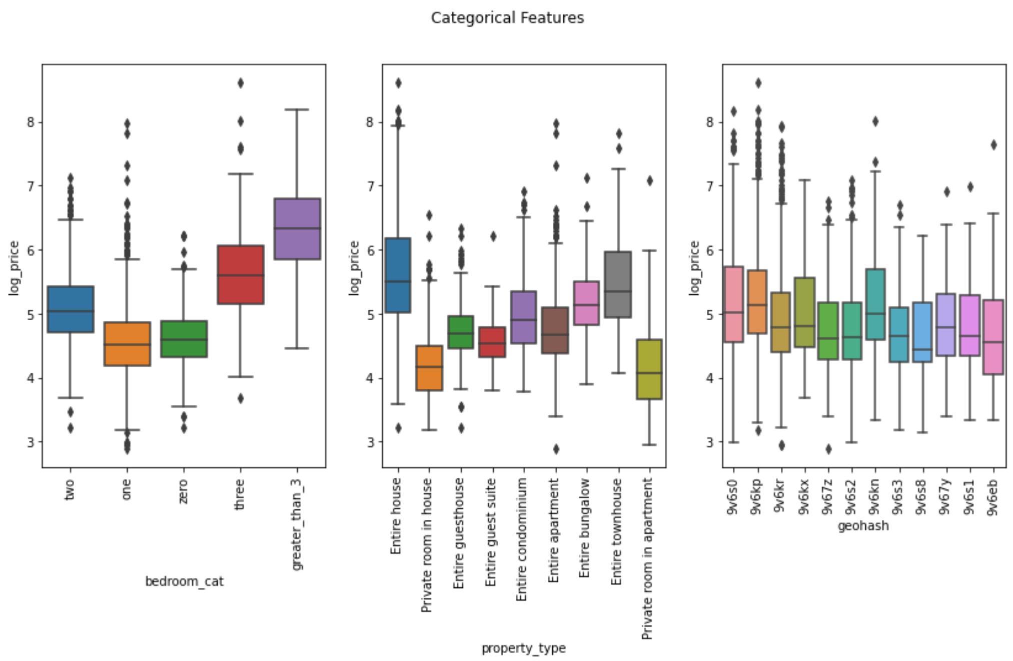

Figure 2. Relationship of categorical features and log price

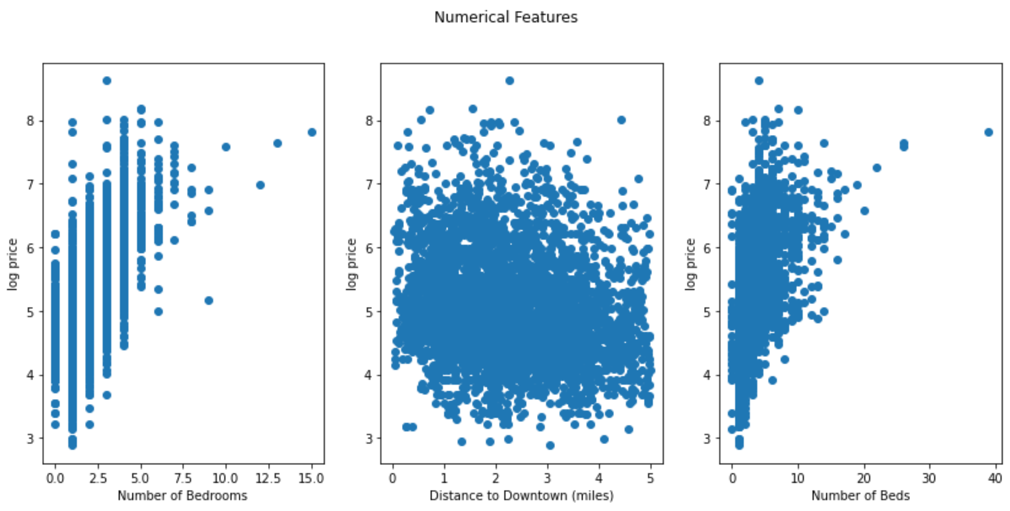

Figure 3. Relationship of numerical features and log price

There are some interesting findings in the underlying data that relates to this particular learning problem. There seems to be some strong relationships between our categorical features and log price as evidenced by the change in the mean between each category. There also seems to be some relationship between log price and the number of beds and bedrooms (this intuitively makes sense as from our own subject expertise we believe larger listings would list for more money.) It's equally interesting that there seems to be no real relationship between log price and distance to downtown.This could suggest that many people are not wanting to be particularly close to downtown and that attractions are elsewhere. For this problem I will be using root mean squared error(RMSE) as a metric for accuracy where: $RMSE = \sqrt{\frac{\sum_{i}^{N}(x_{i} - x_{n})^{2}}{N}}$

Modeling Results:

Decision Trees:

For decision trees we were asked to implement a tree with pruning. From our text, we know about c4.5 and c5.0 from the creator of the IDE3 algorithm which implements some forms of pruning. From [4] I chose to use minimal cost-complexity pruning. This algorithm is relatively simple and is parameterized by one parameter, alpha. Alpha must be greater than 0 and helps my adding a cost to a tree. Essentially, as alpha increases the complexity of the tree decreases.

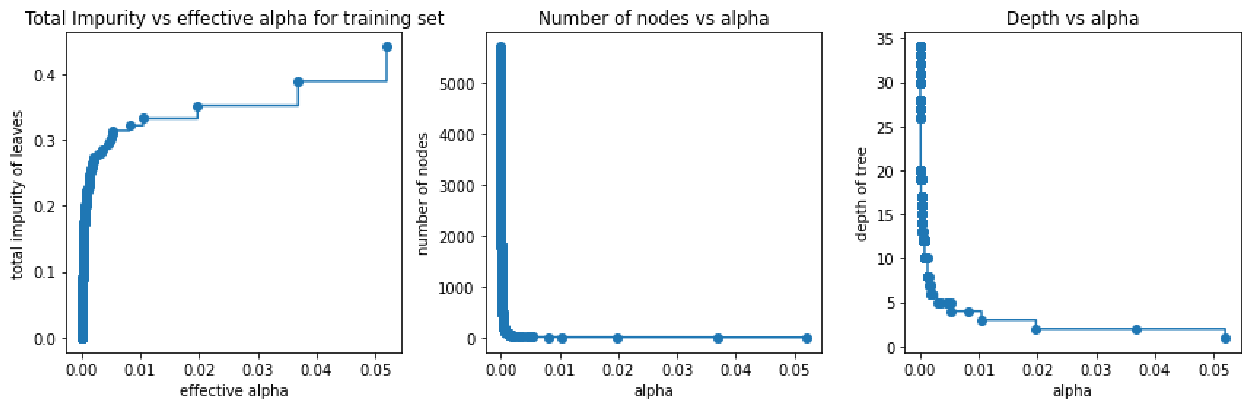

Figure 4. Alpha vs impurity, nodes, and depth of a decision tree

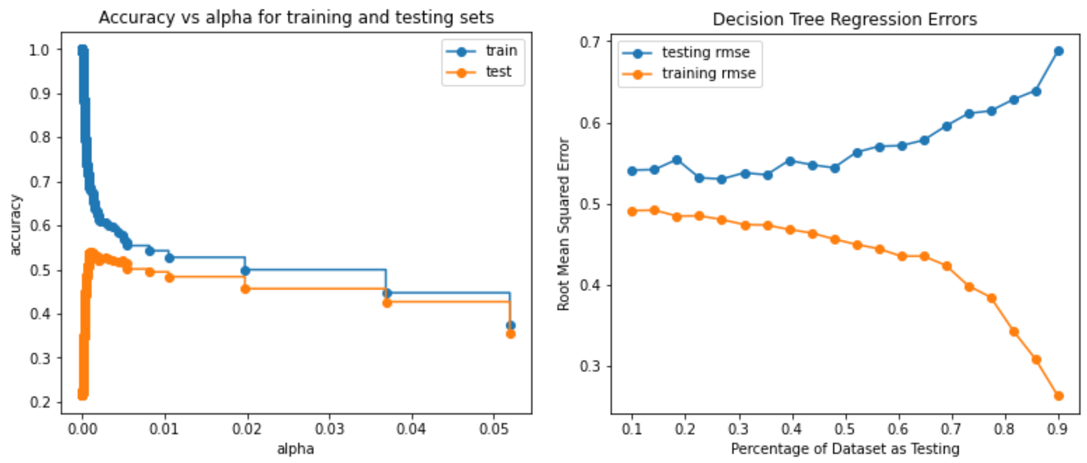

Figure 5. (a) Alpha vs accuracy (b) Accuracy for % of dataset as testing data

Looking at Figure 5 (a) we can see how by varying alpha our training and testing error change. I considered the best value of alpha to be the one which minimizes our testing error. I chose this as my definition of best because one of the goals of a machine learning model is to generalize. The optimal value for alpha was 0.001. Utilizing this value I ran multiple predictions against different percentage splits of the data being testing data. A value of 0.1 on the x-axis for Figure 5(b) means that I split the data using 10% as testing and 90% as training. As one can see the values for accuracy for training and testing are inversely related (ie. as the amount of training data increases, the testing accuracy suffers). It's also very important to note how effective CCP is at avoiding overfitting. Looking at our testing and training RMSE lines they are very close together (<= .05 RMSE) as long as the testing data used is < 60% of the whole dataset. This shows just how useful CCP can be.

Neural Networks:

For my neural network I use a multi-layer perceptron with an adam optimizer and an initial constant learning rate of 0.01. The architecture is simple with one input layer, one hidden layer with 100 units and one output layer. Initial values for the learning rate are very important as a value too small can lead to very slow convergence and large values can lead to unstable training and/or suboptimal weights. We can see this in Figure 6(a).

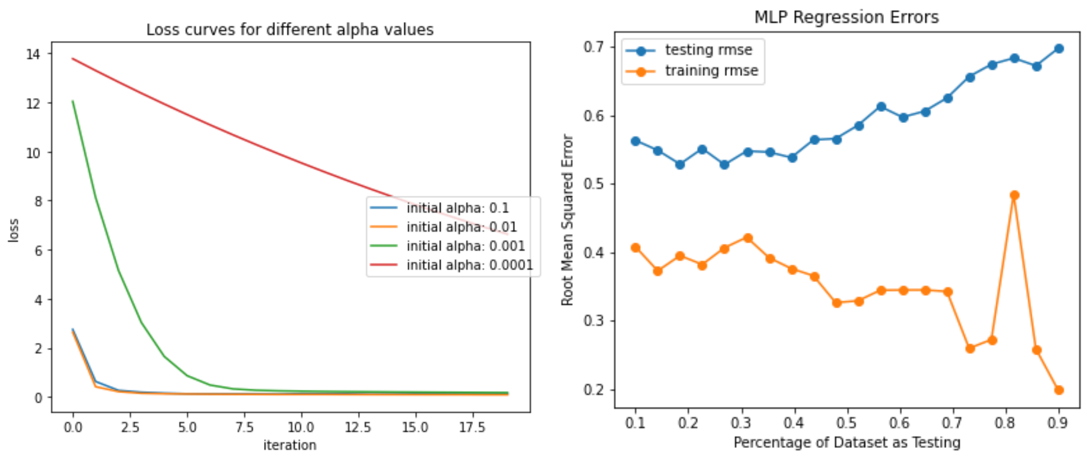

Figure 6. (a) Loss as a function of learning rate (b) Accuracy for % of dataset as testing data

As with the last problem. The higher the % of testing data used the worse the algorithm generalizes. Interestingly enough, for the MLP it looks like the RMSE is quite high when the dataset is at 80% testing. This is most likely due to an outlier not being included. For all models I used a ReLU[5] activation function with the Adam[6] solver. As shown in Figure 6(b), converge is met quickly, for most learning rates in under 100 iterations.

Boosting (Random Forests):

For the boosting algorithm I chose to use a Random Forest. Similar to the decision tree model, I also used the cost-complexity pruning algorithm to tune the model. As one can see in Figure 7 (a), for training, accuracy decreases drastically as alpha increases. This makes sense due to the fact that the random forest easily overfits the data.

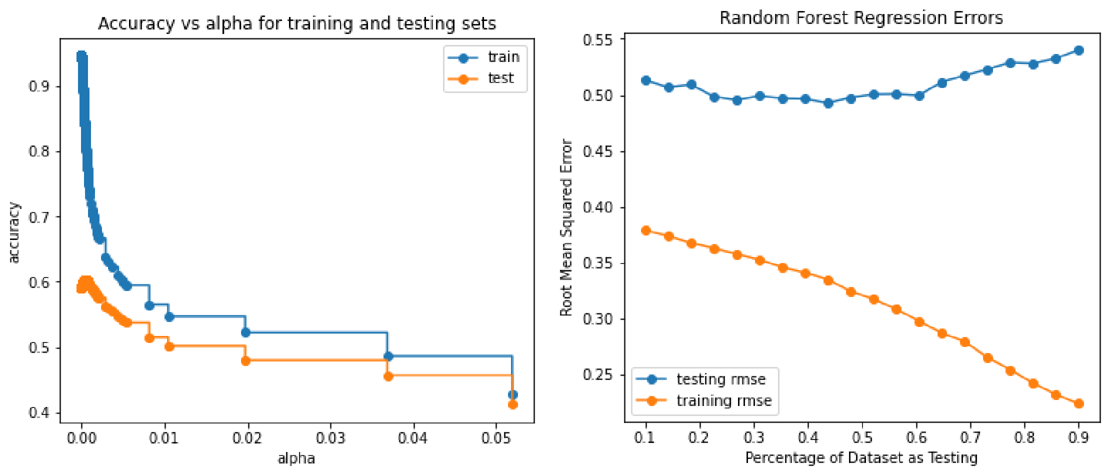

Figure 7. (a) Alpha vs accuracy (b) Accuracy for % of dataset as testing data

The optimal value for alpha was .0004 which interestingly enough is smaller than the Decision Tree model above. Similar to the other plots, the values for accuracy for training and testing are inversely related. I think it's really interesting that the random forest model doesn't do as well with overfitting as one would expect considering that it uses boosting. It appears that the RMSE between testing and training is almost always >10%.

Support Vector Machines:

For the SVM model I chose to look at two different kernels. Specifically, the radial basis function(RBF) kernel from the lectures and a polynomial kernel from the lectures.

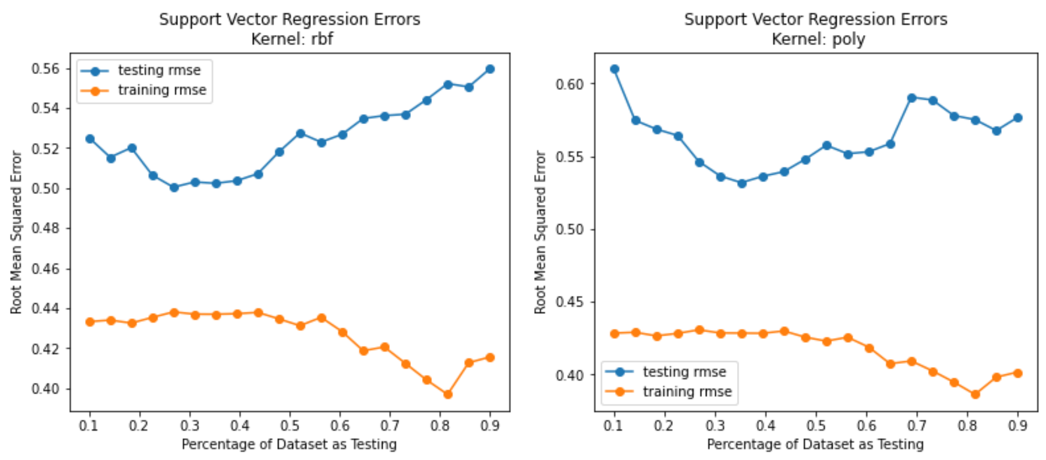

Figure 8: Accuracy by % of dataset as testing data (RBF and polynomial kernels)

The RBF kernel performs better than the polynomial kernel. I found this to be interesting because I purposely engineered features that would have some form of linearity to them. However, I suppose that without these features the difference in accuracy between the two may be higher. Out of the two it looks like the RBF kernel does a better job avoiding overfitting even though the polynomial kernel is only a third degree polynomial. This could be due to the underlying structure of the data as well.

K-Nearest Neighbors:

The experiment I conducted for KNN involved changing the number of neighbors and evaluating the accuracy on different splits of the training and testing data. I chose four different values for k: 2, 4, 6, and 8.

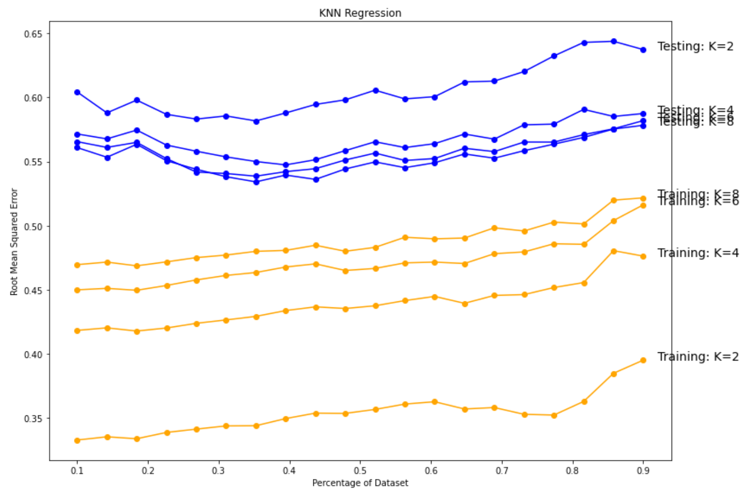

Figure 9. Accuracy by % of dataset as testing data and number of neighbors

I found these models to be particularly interesting. The KNN model using 8 neighbors had the highest in sample error but also the lowest testing error. Conversely, a model with 2 neighbors had the lowest errors on the training set but the highest error on the testing set. Additionally, all other models had training errors that decreased with more data. It seems as if the KNN model actually increases with more data. I found this to be exceedingly odd. One last thing I could have changed was the distance function used by the KNN. For all these examples, I used euclidean.

Model Comparisons:

For every model I followed the same process. I took the original dataset and split it into testing and training portions. Utilizing the above figures, I selected the split percentage which minimized the testing RMSE as this value is the one that enables the model to generalize the best. I then averaged these values to select 35% of the data as testing and the remaining 65% as training. I then selected the hyper parameters which minimized the testing RMSE as well.

Algorithm	Training RMSE	Testing RMSE	Difference	Wall clock times
Decision Trees	15.8	52.7	36.9	750 ms

Multi Layer Perceptron	19.1	52.3	33.2	1min 13s
Random Forest	22.2	49.4	27.2	33.3 s
Support Vector Machine (RBF)	39.7	50	10.3	11 s
K-Nearest Neighbors (K=8)	46.9	53.4	6.5	7.07 s

Table 2. Best training and testing RMSE for all algorithms

As shown in Table 1, all algorithms perform about that same in terms of testing RMSE but drastically different with respect to training RMSE. Even with the tuning of the CCP alpha parameter for the decision tree, it overfits the worst. Additionally, our MLP overfits as well. For the MLP, we could have tuned parameters to make this better by adding dropout or some form of regularization to the network. I added the difference column as a proxy for overfitting.

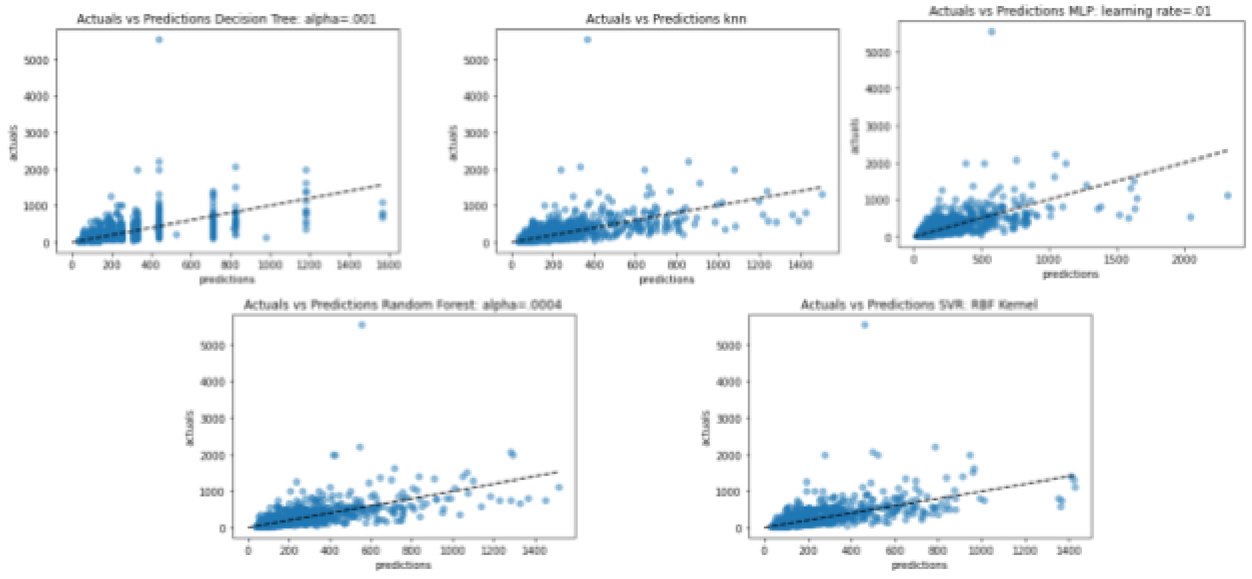

Figure 10. Original price variable vs prediction by algorithm

By definition RMSE is the average of errors across a dataset. As such, this accuracy metric can have issues with outliers. Looking at Figure 10, one can see how there is a massive outlier where the actual price is > $5,000 but the prediction is almost always around $500. For each image in this figure, a perfect prediction would be along the black dotted line. Blue dots closer to the line mean the models are doing a better job predicting. As we can see, the decision tree, due to pruning, has issues with predictions and tends to predict in discrete variables. This is obvious, as we have pruned the tree and therefore it doesn't have as many leaves to predict with. The MLP also overpredicts prices more than the other algorithms. From these plots, it's clear to see that the SVR and Random Forest have the "tightest" predictions. The decision tree trains relatively quickly, which makes sense considering that we are forcing a very pruned tree with alpha. The MLP has by far the slowest training time at one minute where the second slowest is at 33 seconds.

Predicting Listing Availability:

Description:

For the second problem, I chose to predict whether or not a listing would have any availability 90 days out from the date the data was gathered. Importantly, this is not a temporal problem looking at whether a listing will be available in the next 90 days. Instead, it is answering the question, "At this point in time does this listing have availability?". Luckily, this is an easy problem to structure because we have a numerical feature named "availability_90" which we can use to transform the data to answer this binary classification problem. If there exists 0 availability in the next 90 days we denote this value as a 0. If there is any availability we denote this a 1. I used all the same features as problem 1 so please refer to the data summary section for definitions.

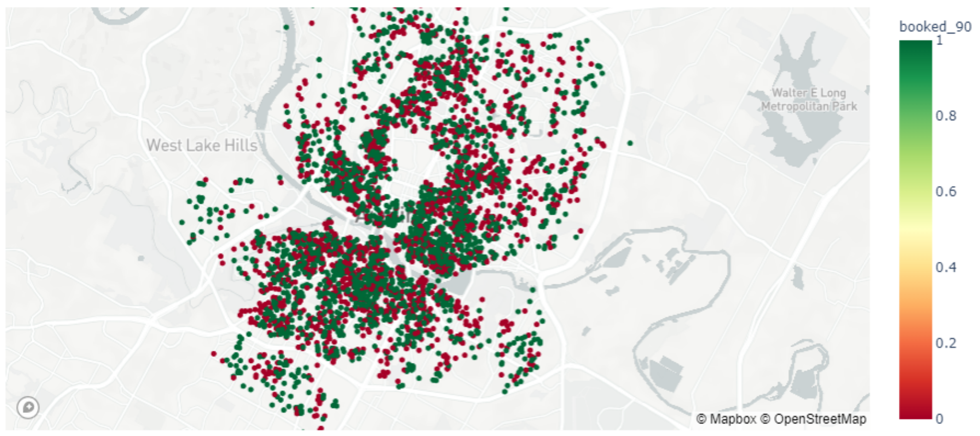

Figure 11. Availability of listings in Austin, Texas. Green is available, Red is unavailable

In problem one we used root mean squared error as a metric for success. In this problem we will be referring to accuracy defined as: $Number\ of\ total\ correct\ predictions\ /\ Total\ number\ of\ predictions$. Once again, all data is standardized and we use cross validation for all reported results.

Modeling Results:

Decision Trees:

Similar to problem one, I decided to use cost-complexity pruning (CCP) in order to prune the tree appropriately. In order to save space, I refer the reader to the section on decisions trees in problem one for a reminder..

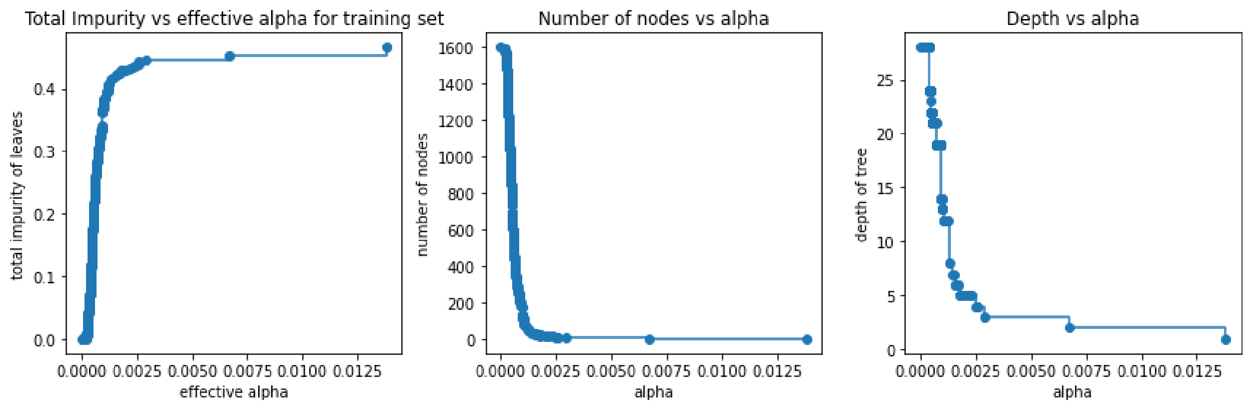

Figure 12: Alpha vs impurity, nodes, and depth of a decision tree

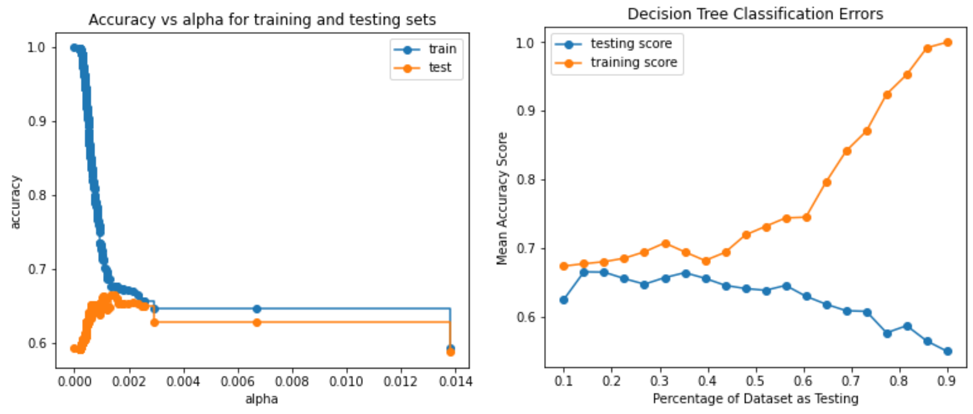

Figure 13. (a) Alpha vs accuracy (b) Accuracy for % of dataset as testing data

Intuitively, impurity increases with alpha and the number of nodes and depth decrease with alpha. Similar to most problems our model overfits the training set relatively easily depending on the amount of data provided at testing. Similar to the last problem, CCP is quite good at avoiding overfitting as the accuracy for both testing and training is <5% away from each other. For the final model I used an alpha of .0001.

Neural Networks:

For problem two I used a relatively straightforward multi-layer perceptron (MLP). Instead of using ReLu again, I decided to use the logistic function from the lectures. Again, I used the Adam optimizer and varied the learning rates as per Figure 14. The architecture is simple with one input layer, one hidden layer with 100 units and one output layer.

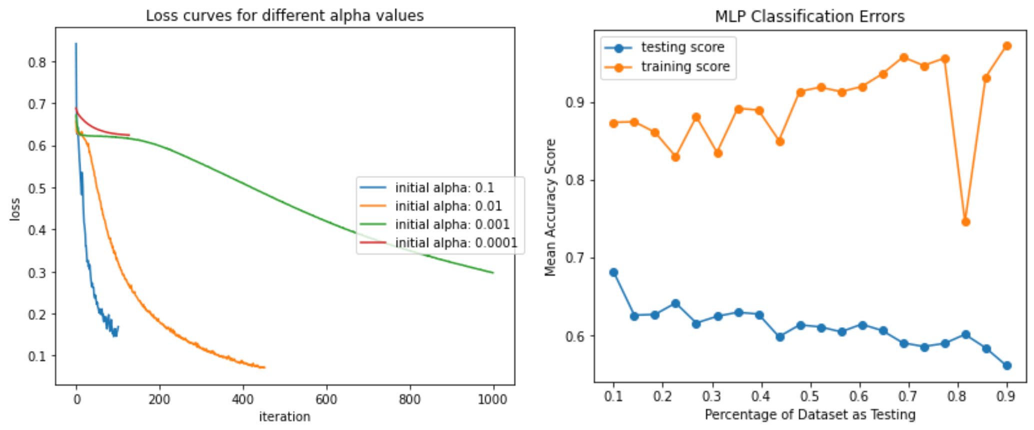

Figure 14. (a) Loss as a function of learning rate (b) Accuracy for % of dataset as testing data

In figure 14(a) it's really interesting to see how our loss function decreases rapidly depending on the initial alpha selected. For alpha set to 0.1 the loss curve looks converges much faster but with higher loss than alpha set to 0.01 while learning rates greater than .0001 don't seem to ever converge. With an alpha of .0001 it looks like the model converges too soon. This model seems to overfit quite a bit as the difference between accuracies is > 10% irregardless of the percentage of data as training. For the final model I used a learning rate of .01.

Boosting (Random Forests):

As before, I used the CCP to prune the random forest. The value for alpha was again lower than that of the decision tree suggesting that the tree doesn't need as much pruning.

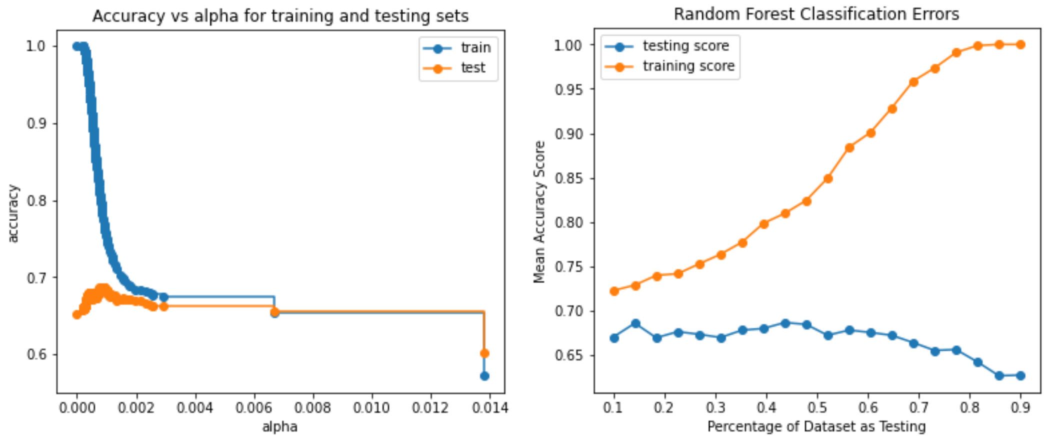

Figure 15.(a) Alpha vs accuracy (b) Accuracy for % of dataset as testing data

From Figure 15(b) one can see how the model does a really good job avoiding overfitting as long < 40% of the dataset is allocated towards testing. For the value of alpha, I again used the value which maximized the testing error of the model utilizing 35% of the data for testing. For my final model, I use an alpha of .0004.

Support Vector Machines:

For the SVM models, I again tested two kernels either linear or radial basis function. Again, it looks like the RBF does a better job as evidenced by the higher accuracy. I decided to test the linear kernel to see how the SVM would handle a strictly linear kernel. The SVM with linear kernel looks to avoid overfitting better than the SVM with the RBF kernel. This is the only example in all of the experiments which show a testing accuracy higher than the training accuracy.

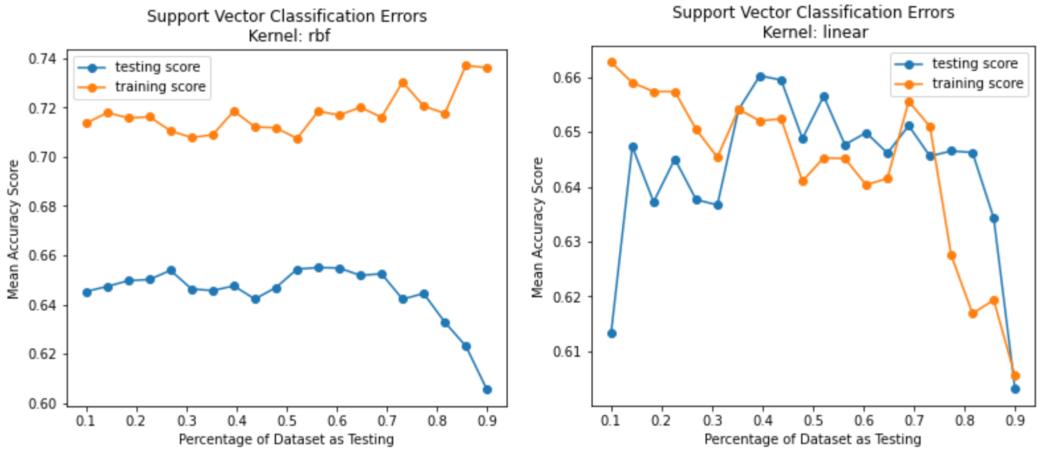

Figure 16: Accuracy by % of dataset as testing data (RBF and polynomial kernels)

In my final model I use the RBF kernel because it has the lowest amount of overfitting and comparable accuracy to the polynomial kernel.

K-nearest Neighbors:

The experiment I conducted for KNN was to change the number of neighbors and evaluate the accuracy on different splits of the training and testing data. I chose four different values for k: 2, 4, 6, and 8 and used euclidean distance as my distance function. I used euclidean because I didn't have any SME to suggest that another metric would work better.

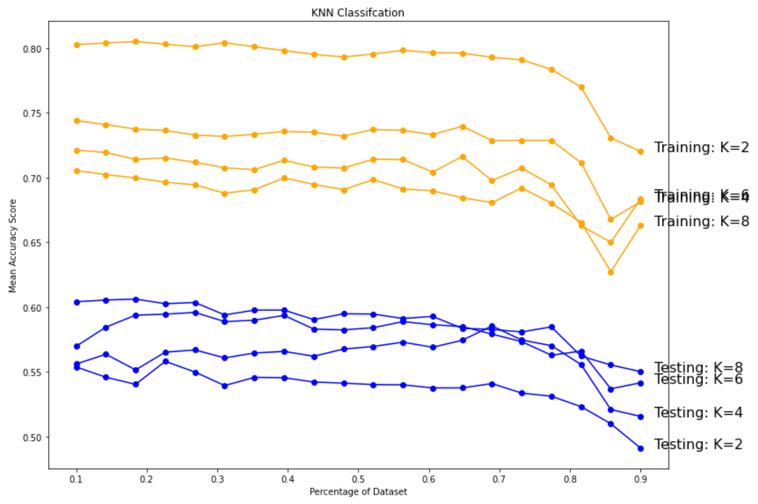

Figure 17. Accuracy by % of dataset as testing data and number of neighbors

Similarly to problem one, the KNN model using 8 neighbors had the highest training error and lowest testing error while a model with 2 neighbors had the lowest training errors on but the highest testing error. Additionally, all other models had training errors that decreased with more data. It seems as if the KNN models accuracy decreases with more data. Technically, this would mean it does a great job at generalizing. In my final model I use K=8.

Model Comparisons:

For every model I followed the same process. I took the original dataset and split it into testing and training portions. I used cross validation to select testing and training datasets allocating 35% for testing and 65% for training.

Utilizing the research on the proper hyperparameters from the above section, I used the aforementioned values that maximized accuracy for my final models.

Algorithm	Training Accuracy	Testing Accuracy	Difference	Clock Time
Decision Trees	69%	59%	10%	236 ms
Multi Layer Perceptron	90%	61.9%	28.1%	2.33 s
Random Forest	76.9%	67.4%	9.5%	741 ms
Support Vector Machine (RBF)	71%	63%	8%	2.71 s
K-Nearest Neighbors (K=8)	69.1%	59.75%	9.35%	902 ms

Table 3. Best training and testing accuracy for all algorithms

From this table, it could be argued what the "best" model is. However, I would select the Random forest because it has a low amount of overfitting which means it generalizes well and it has the highest testing accuracy. I would say that the MLP model is the most deceptive with a 90% training accuracy and a 62.9% testing accuracy. This shows how this particular neural network does a poor job generalizing. That being said, it's not the MLP's fault for overfitting as I didn't provide any form of regularization or layer dropout.

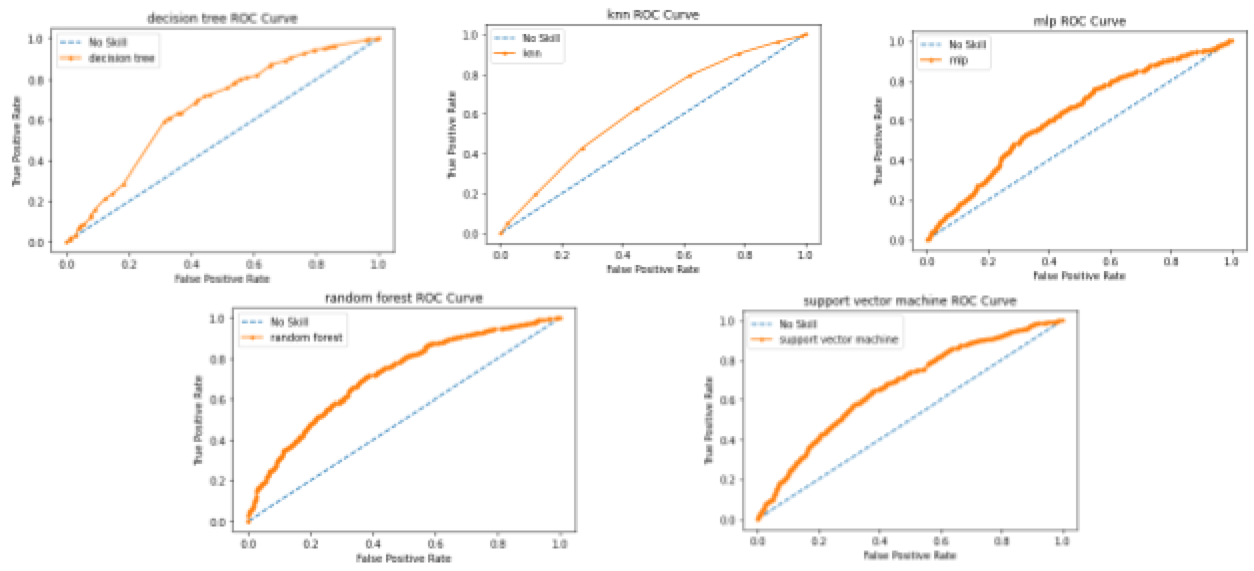

Figure 18. Receiver operating characteristic plots for all algorithms

In Figure 18 I plot the receiver operating characteristic (ROC) curves for all models)[7]. The area under the curve (AUC)[8] is reported in Table 4. Utilizing this accuracy metric, we can state that the random forest model is the best followed by the SVM.

Model	Decision Tree	KNN	MLP	Random Forest	SVM
AUC	0.662	0.622	0.627	0.715	0.668

Table 4. Area under the ROC curve for all algorithms

Lastly, the algorithms have very different training times. The decision tree trains relatively quickly, which, again, makes sense considering we are very restrictive with alpha. The support vector machine and MLP are the slowest.

Comparing Problem One and Two:

As stated in the data summary, I tried to engineer features which were inherently linear as frequently with machine learning algorithms we try to approximate a function. Given the linearity assumptions of multiple algorithms having these linear features may give them a shot. Additionally, as stated by the lectures, it's important to have subject matter expertise in order to select the best algorithms and to have intuition for why algorithms perform the way that they do. In problem one it was clear that the tree-based methods struggled compared to the other methods. I believe this is due to the fact that trees aren't always the best choice for regression problems. When predicting a float the number of leaves could potentially be increased to all possible values in the training set. Conversely, they excelled at the binary classification problem put forth by problem two. For both problems, it appears as if the linearity added in the dataset gave the KNN a fighting chance as it performed comparable with the other models. I also found it very interesting that in Figure 16, the accuracy for testing was higher than the training accuracy for the SVM with linear kernel. It should also be noted that the wall clock times for the classification algorithms is much faster than the regression algorithms on average. Again, I think this illustrates how the machine learning practitioner has to pay attention to how the underlying data looks in order to enable the classifiers to work as intended.

Citations:

[1] Inside airbnb. adding data to the debate. (n.d.). Retrieved February 14, 2021, from http://insideairbnb.com/
[2] Geohash. (2021, February 14). Retrieved February 14, 2021, from https://en.wikipedia.org/wiki/Geohash
[3] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.
[5] Vinod Nair, Geoffrey E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines" 2010.
[6] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.
[7][8] Receiver operating characteristic. (2021, January 14). Retrieved February14, 2021, from https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve