MILESTONE 2 CMPT 459

Group: Information Gainers
By Sina Khalili (SVM), Logan Militzer (Decision Tree) & Phong Le (Logistical Regression)

LINK TO MILESTONE 2 CODE:

https://github.com/SinaKhalili/kaggle-data-analysis/blob/master/classifiers_svm.ipynb

1) The features our group chose were bedrooms, bathrooms, latitude, longitude, price, special characters, numbers, stop words and uppercase letters. The reason we chose these features was because they are numerical attributes and therefore, they could be handled better by our group's classifiers. Additionally, some of classifiers can only handle numerical data such as a support vector machine. To encode features from the description for the decision tree classifier, we generated new columns about features our group found interesting (for example StopWords, Uppercase characters and numbers). We also used a library that performed binarization on categorical attributes, though this didn't really end up helping. To generate this continuous data, we used a dummy call that automatically set the discrete data into continuous data.

2) The Python library we chose to use was sklearn. The reason we chose sklearn was because there is great documentation on the library. Additionally, it is easier to implement, more compatible than other libraries and has classifier functions for all the classifiers in this assignment. Other libraries we used were standard libraries such as numpy and pandas as both these libraries work well with sklearn. Moreover, they allowed our group to perform various actions such as finding headers, renaming & dropping unnecessary columns to reduce the data set width.

3) We performed cross-validation with a sklearn library function called kfolds and cross_val_score. We used this sub-library of sklearn because it was efficient and provided the ability to quickly change the number of folds computed. Additionally, it allowed us to easily compare the results before and after modifications to the classifiers. The procedure we followed was first identifying the class of interest. The class our group was interested in was "interest_level". First, we binarized the interest level by generating dummy values each interest level. This allowed us to run class through the cross-validation score metric. The cross-validation scores also require the data set without labels. We generated this data set by dropping attributes not in interest for our classifiers. Finally, we modified the number of folds computed to fine tune our cross-validation scores consistently across all the classifiers.

4) The performance our group initially received were low scores for all classifiers; low 60 percentages. Below are the performance analysis observations.

- <u>Support Vector Machine</u>

  The training of the support vector machine required a significant amount of time to train. This may have been due to the combination of a large data set and running the training on a computer without a GPU. To improve run time, we reduced the size of the training data to a point where the support vector machine could create a model in a timely fashion.

  These are the numbers received on our initial support vector machine with validation data set.

  Cross-validation score: ~ 68%

  Kaggle log error score: 1.49068

  These numbers fall in line with our expectations as minimal pre-processing and no modification efforts were attempted to improve the score. Additionally, support vector machines have long runtimes inherently.

- <u>Decision Tree</u>

  The decision tree was quick to compute and easy to interpret. However, we did recognize that it was overfitting with the low score and the number of nodes generated.

  These are the numbers received on our initial decision tree with validation data set.

  Cross-validation score: ~ 63%

  Kaggle log error score: 23.58702

  These numbers initially surprised us, especially the log error score as it was very high. However, we did consider the fact that we did not have much pre-processing and no modification efforts were made to improve the initial score.

- <u>Logistical Regression</u>

  The logistical regression classifier was quick and simple to compute. However, these are the numbers received on our initial logistical regression model with the validation data set.

  Cross-validation score: ~ 69%

  Kaggle log error score: 1.43994

  These numbers fall in line with our expectations as logistic regression is simple computationally and quick to compute. It performed the best out of the 3 classifiers on the initial scorings. We should note that there are limited optimizations that can be applied to the logistical regression classifier.

5) For each classifier, our group performed various techniques to increase the cross-validation score. Below are the modifications and optimizations performed on each classifier.

- <u>Decision Tree</u>

  One of the techniques we performed on the decision tree was to normalize the data set we were working with. We used the min-max scaling normalization method. Normalizing allowed the attributes to have similar weights, but we also had to consider that this process was sensitive to outliers. Therefore, we reduced the data slightly by removing the outliers found in milestone 1. These outliers could be classified as human input error or simply missing values. We also pruned the decision tree to reduce the number of branches and possibilities. Another performance improvement we performed was an exhaustive search from depth 3 to depth 20 to see which decision tree was best. 4 ended up being the best depth. Finally, we changed the number folds performed by the cross-validation score. We analyzed our findings and decided that 5 folds fit our data set the best.

- <u>Support Vector Machine</u>

  Like the decision tree classifier, we performed normalization on the data set. We also used the min-max scaling normalization method for this classifier, though kept our classifier answer as strings. Like decision trees, we changed the number folds performed by the cross-validation score. Our main efforts for improvement began with a large grid search to tune the hyper parameters. We began with changing the 'C' value to each of {0.1,1, 10, 100}, and for every ‘C’ value we tried a gamma value from {1, 0.1, 0.01, 0.001}, and for every ‘C’-gamma pair we tried the four different kernels: ‘rbf’ - radial basis function (which performed the best), the classical polynomial kernel, as well as the sigmoid kernel. We also tried the linear kernel but not for every combination of the values above (only for some as the linear was not very promising). Overall we created around 50 support vector machines to settle on our best one. Our best hyperparameters that we settled on were : the rbf kernel, with a regularization constant ‘C’ of 100, and a gamma of 1.0.

- <u>Logistical Regression</u>

  Like the decision tree and support vector machine classifier, we performed the min-max normalization on the data set. We also ran this model with the data set that removed outliers as min-max normalization is sensitive to this. We also changed the cross-validation score folds to see the one that worked best with the classifier.

6) To ensure our group reduced the possibility of overfitting, we researched and revisited the lectures slides to look for techniques to implement. Ways we reduced overfitting was to ensure the training data was a reasonable size and not too large. We considered if there were features in our data set that we could collectively remove as a group. Moreover, we performed cross-validation by using our initial training data set to create multiple train-test splits. We then used these partitioned training data sets to tune the model by changing the number of folds. Finally, we pruned the decision tree to remove overfitting branches.

7) After the modifications noted in question 5, our classifications improved; some improved more than others. Below are the scores after the modifications and optimizations were performed on each classifier.
   - <u>Support Vector Machine</u>

     Cross-validation score: ~ 69%

     Kaggle log error score: 1.43994

     For the modification and improvement, we performed a grid search. This added significant runtime to the model. After the training the model and running the test data, not much improvement was noted. We believe this was the case because the data set possibly may not be suited for support vector machine classification.

   - <u>Decision Tree</u>

     Cross-validation score: ~ 68%

     Kaggle log error score 0.99459

     Both the cross-validation score and the Kaggle log error score improved significantly once the modifications and improvements were made to the model. We believe it improved a significant amount because of the pruning. Before the modifications and improvements, the model was overfitting. Once the model was pruned, the model became less prone to overfitting and therefore improved both score metrics.

   - <u>Logistical Regression</u>

     Cross-validation score: ~ 69.3%

     Kaggle log error score 1.09819

Oddly enough, after normalization, the cross-validation score did not improve significantly. However, the Kaggle log error score did improve significantly. We believe the Kaggle log error improved a good amount because everything was weighted nicely with the min-max normalization once the outliers were removed.

8) The additional evaluation metric we chose was the f-measure. f-measure measures the precision and recall regarding the accuracy of the classifier via a harmonic mean. Ideally, the f-measure produces a number close to 1. We imported a sklearn metric called f1_score and performed a weighted average on the class test and the class prediction to achieve this evaluation. The classifiers received f-measures as follows below.
   - <u>Support Vector Machine</u>

     f-measure: ~ 57.5
   - <u>Decision Tree</u>

     f-measure: ~ 64
   - <u>Logistical Regression</u>

     f-measure: ~ 58

9) If we were to combine the Kaggle data set with another data set, we would think what would create more value for a particular property when compared to others. We hypothesized that nearby Starbucks locations may influence a property to be higher in interest level. To do this, we would collect a data set that contains Starbucks locations with address, longitude and latitude. An example data set can be seen here: https://www.kaggle.com/jaseibert/analysis-of-starbuck-s-store-locations/data. We would join the data set on longitude and latitude to see if locations near a Starbucks are higher in interest. We would then rerun the models and evaluate the models to see if there is a score improvement.