MILESTONE 3 CMPT 459

Group: Information Gainers
By Sina Khalili (XGBoosting), Logan Militzer (Random Forest) & Phong Le (K Nearest Neighbours)

Link to code: https://github.com/SinaKhalili/kaggle-data-analysis

1) The classifiers our group chose was XGBoosting, Random Forest and K Nearest Neighbours. We chose these classifiers because we found them interesting; especially XGBoosting, since it's one of the more efficient and popular boosting algorithms. It is known as a gradient boosting algorithm since it uses gradient descent. Additionally, we researched these classifiers and thought they would fit well for our data set because they're more modern techniques to classify.

2) The features our group chose were bedrooms, bathrooms, latitude, longitude, price, special characters, numbers, stop words and uppercase letters. We dropped the rest of the attributes as we didn't find these useful for our classifiers. Like milestone 2, the main reason we chose these features was because they are numerical attributes. Our classifiers handle numerical attributes better than categorical ones. During milestone 1, we encoded features from the description by generating new columns. These were features our group was interesting in. These features were, for example StopWords, Uppercase characters and numbers. To convert categorical attributes to numerical attributes, we performed binarization. We used a library to follow through with this. This library made dummy values automatically. This proved to be useful when attempting to tune the parameters for the random forest classifier.

3) We performed cross-validation with the sklearn library. The functions we used were called kfolds and cross_val_score. Additionally, these functions provided efficient computation and gave us the ability to change the number of folds computed. 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. 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 moderate scores for all classifiers; high 60 to low 70 percentages. Below are the performance analysis observations.
   - <u>XGBoosting</u>

     On the initial training and validation dataset, with a very basic XGBoosting algorithm we were able to get around 71% accuracy, which was pretty in line with what was expected. Though it does seem like it may have been overfitting, since the training accuracy was sitting at much higher accuracy of ~80% which was concerning. This was revealed in full when we tried to see the kaggle score, which was a disappointing 2.02 - surely not what we want with such a modern technique! This is where we realized the true depth of the XGBoosting algorithm. Just in the sklearn library there are over *ten* hyperparameters to tune (though some of them have only a few options), namely however these were the learning rate, the depth, the estimators, and the objective. This was going to require some more in-depth knowledge and work:

These are the numbers received on our initial XGBoosting classifier with validation data set.

Cross-validation score: ~ 71%

Kaggle log error score: 2.02

These numbers fall in line with our expectations as minimal pre-processing and no modification efforts were attempted to improve the score. One benefit of the XGBoosting is the ability to quickly train, every classifier only took a few seconds (~10) to get created – this is because the “eXtreme” in XGBoosting means it does a lot to improve running time such as caching, parallelizing and GPU use when available.

- Random Forest

The random forest was quick to compute initially because not many parameters were set nor tuned for the training data. We did note that this score could be improved by fine tuning the parameters or by training a more consistent data set.

These are the numbers received on our initial random forest with the validation data set.

Cross-validation score: ~ 72%

Kaggle log error score: 6.23692

These numbers did not surprise us as no tuning we performed on the random forest. However, we were slightly surprised with the cross-validation score being moderate in score rather than lower 60’s like milestone 2. We were also surprised with the moderate cross-validation score and the poor Kaggle log error score.

- K Nearest Neighbours

The KNN classifier was quick and simple to compute. However, these are the numbers received on our initial K Nearest Neighbours model with the validation data set and 5-fold cross-validation.

Cross-validation score: ~ 66%

Kaggle log error score: 11.29661

The Kaggle score is not surprising as the worst performance of the 3 chosen ones because KNN has this “curse of dimensionality”; a large feature set means more data, and larger dimensionality means more opportunity for larger magnitude data points to dominate smaller ones, for example, like price and bedrooms respectively. Also, it turns out that high dimensionality spaces makes Euclidean distance less useful because any point requires its neighbour to be close in every single dimension.

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>XGBoosting</u>

    Here is where the real work began. First, we did a research into all the various hyperparameters for the XGBoosting algorithm. Then we used a RandomSearchCV, that is, a large number of parameters which were in inputted into the training and randomly chosen and cross validated. After many rounds of this, a best model is chosen based on the score and used as the main model. Using this, we fit 5 different folds of 25 candidates, giving us 125 total fits, and we performed that 5 times, for a total of 600 XGBoosting classifiers. We took only the best of all of those which gave a model with subsample of 0.4, regulation lambda of 2, regulation alpha of 0.5, 250 estimators, learning rate of 0.001 and many other attributes which you can find on the notebook on github. Using this new model, we were able to get slightly better performance on the training data – 72% accuracy – but more importantly were able to improve the Kaggle log loss score, but only slightly to 1.88. This was again slightly disappointing since XGBoosting was supposed to be much stronger than this. The next approach was to again use a grid search this time, which tries all possible combination of a few parameters, and this time we used a GPU to save some time. With this level of training, we were able to reduce the log loss of the only slightly again to our best score of 1.22 - still way below what we thought would be possible with XGBoosting. So afterwards after doing some research we found an autoML which uses genetic learning to create XGBoosting models called “Teapot”. After running the autoML for a while, we used its model, which was also quite bad and didn’t really improve our score at all. In a final research, we stumbled upon a notebook on the Kaggle website in which XGBoosting was used properly and with those same parameters got ~0.56 log loss, however since this was not found by us, we don’t consider it our best score. This was just to show that XGBoosting could achieve really good numbers if researched properly.

  - <u>Random Forest</u>

One of the techniques we performed to improve the scores was by generating a min-max normalized data set. We ensured that this data set was cleaned, and outliers were removed. We did this because this normalization technique is sensitive to outliers. To further improve the scores, we performed tuning of the parameters of the classifier. This was achieved by testing different combinations of parameters. The parameters we were mainly tuning was n_estimators, min_samples, min_samples_leaf, max_features, max_depth and bootstrap. We determined these were the best parameters to tune after a lot of testing. However, this was intensive on our computers as this classifier liked having more estimators. However, a problem began to arise as we tested and tuned the parameters. This problem was that our computers, especially the Logan's computer, began constantly crashing as it couldn't handle all the computations. There was a point that we had to stop this tuning as it could have been potentially be harming the computer with how hot, how long and how much computing power was being used during these extreme lengths of time. We believe we could further improve the tuning of the parameters if a more power computing platform was used. Ultimately, we learned that there is a trade-off we need to consider; classifier performance and computing efficiency.

- K Nearest Neighbours

  Using min-max normalization on the data set equally scales it to make sure that higher magnitude data points less dominant than lower ones. We also cleaned out any outliers and missing values that could negatively affect min-max scaling. We then tried to tweak the hyperparameter k, the number of neighbours in KNN, and reduce the dimensionality of the dataset by selecting correlated features via Pearson's correlation matrix. This gave us bathrooms as the main feature along with prices, and Numbers. Setting n_neighbors to 10 along with normalization of values gives us approximately 68% on the cross val score, a 2% increase from question 4. Although this score did not significantly improve, the Kaggle log loss score did immensely, which we will see in question 7.

6) To reduce the risk of overfitting, our group researched various ways on the internet. Additionally, we refreshed ourselves on the lecture slides seek the techniques to implement to reduce this risk of overfitting. Firstly, we learned via lecture and via the lecture slides was to generate a training data set that was a reasonable in size; not too large but also not too small. We also reduced the width of the data set by dropping attributed our group didn't find useful like mentioned in question 1. Moreover, it was recommended to performed cross-validation on our training data set. This cross-validation created multiple train-test splits for the classifiers. Finally, we changed the number of folds by using the partitioned training data sets as a guideline to tune the models by changing the

number of folds. In XGBoosting, overfitting was certainly an issue, especially when it came to some of the grid search CV where the chosen model would give a really good f1-micro score but perform really poorly on the Kaggle website. To remedy this, lots of cross validation was done.

7) After the modifications noted in question 5, our classifications improved. Below are the scores after the modifications and optimizations were performed on each classifier.

- <u>XGBoosting</u>

    Cross-validation score: ~ 80%

    Kaggle log error score: 1.22010

    Again, as noted above we did a lot to try and improve the XGBoosting score such as: tuning the hyperparameters via running a randomized search cross validator, running a grid search, running a gpu-enhanced grid search, using an autoML, and doing more research into the XGBoosting. Changing the scoring function from f1-weighted to f1-micro helped slightly. After all that we found a proper implementation on Kaggle which achieved 0.56 log loss, but since this isn't our finding, we chose 1.2201 as our final best score. We realize that XGBoosting is very powerful but only with the correct amount of knowledge and experimentation.

- <u>Random Forest</u>

    Cross-validation score: ~ 74%

    Kaggle log error score 2.08911

    Both the cross-validation score and the Kaggle log error score improved after performing the modifications and improvements in question 5. We hoped for them to improve more given all the computing time invested into the classifier. We do think we could further improve the score but unsure if it would be worth the computing time given the small improvement noted above.

- <u>K Nearest Neighbours</u>

    Cross-validation score: ~ 68%

    Kaggle log error score 1.42207

    After our modifications, the cross-validation score did not significantly increase. However, I want to highlight the log error score. It dropped from 11.29661 to 1.42207, approximately an 8x improvement. Reducing the number of features whilst keeping the same amount of data entries also reduces the chance of our model overfitting and computing complexity, therefore eliminating the "curse of dimensionality" explained in question 4.

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. There are a few types of f-measure for categorical values, for example 'weighted', 'micro' and 'macro'. To see the accuracy of our own we used 'micro', since this is a multi-class classification setup, we preferred micro as there was a class imbalance (much more low interest posts than other categories).
   - <u>XGBoosting</u>

     f-measure (micro): ~ 0.86 (best)
   - <u>Random Forest</u>

     f-measure (micro): ~ 0.73
   - <u>K Nearest Neighbours</u>

     f-measure (micro): ~ 0.69

9) Using these more modern classifiers was exciting for us as we could see using these in the real world. We did learn that these classifiers require more time to compute and tune but if you have a powerful enough machine, we are confident that you could achieve a score with a Kaggle log error less than 0.6. We learned is there is a significant trade off; classifier performance and computing efficiency. You can't necessarily choose both, but you can determine a fair trade-off that fits the required application. We also learned that exciting methods of data mining which are popular on Kaggle, such as XGBoosting, are not popular because they are easy to use, but because they are powerful only after the technique has been mastered – and hence require quite an amount of research in order to use efficiently. KNN outperformed Logistic Regression (LR) because of the dimensionality reduction. In milestone 2, we did not specify a max_iter for LR, and the model was not able to converge within its default value because of such a large feature set. Therefore, the regression coefficients are not as meaningful as they could be, resulting in a worse performance.