CS109b Final Project - Group 24

- Tyler Finkelstein
- James Fallon
- Aaron Myran
- Ihsaan Patel

Feature Engineering

In addition to the features described in the previous milestone, we attemped to create features from the text fields (`overview`,`tagline`,`plot`,`plot_outline`, and `mpaa_rating`) by clustering them based on key words.

To do this, we first transformed each text field into a tf-idf matrix which gets word counts for each record and then scales down the word counts based on the occurence of the word in the entire training set. As part of the tf-idf transformation we:

- "stemmed" the words ("fishing", "fished", "fisher" would all be reduced to "fish")
- filtered out common english terms ("a", "the", etc.)
- used n-grams of lengths 1,2,and 3 ("John","John likes", "John likes movies", etc.)
- limited the max dataset frequency of a term to 10% (essentially the size of a genre as there are 11 genres) and the minimum datasest frequency to 5 (based on previous models we'd seen)

With the transformation complete we applied k-means clustering to the tf-idf matrix, tuning the number of clusters based on whether the cluster key-words made intuitive sense with the genres in that cluster. For instance, `overview_cluster_3` had key words of "murderer", "town", "killer", "investigator", and "detective", and the top genres in this cluster were Thriller, Horror, and Drama-Thriller. For `overview_cluster_4`, which had key words of "Love", "falls", "woman","man", and "story", the top genres were "Drama-Romance", "Drama", and "Comedy-Romance".

All the code and clusters can be viewed here: https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/Text%20Analysis.ipynb (https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/Text%20Analysis.ipynb)

Credit for some of the code and general approach goes to: http://brandonrose.org/clustering (http://brandonrose.org/clustering)

Model Implementation

Random Forest

We implemented a Random Forest Classifier on our data, which essentially grows decision trees while randomizing the subset of predictors used to grow those trees. The model than takes the mode of the outcomes of all the trees and records that class as the model's prediction.

With the sklearn package, we tuned a random forest classifier using grid search with 5-fold cross-validation on the following parameters:

- `n_estimators`: the number of trees grown in the model. We did not vary this and set it at 500 under the assumption that more trees would improve model accuracy (with the trade-off being the amount of time taken to run the model)
- `max_features`: the number of features in the subset used to train the tree. The more features selected, the more likely the model is to overfit. We used the 3 options provided by the sklearn package - auto, sqrt, and log2.
- `min_samples_leaf`: the numeber of samples in a particular branch at which the tree stops growing. The lower the number, the more likely the tree is to overfit. We used 1, 10, and 100.
- `class_weight`: the sklearn package offers built-in options to deal with class imbalance which is an issue for us, so we tuned the model on 3 parameters - None, balanced, and balanced_subsample.
- `max_depth`: the depth to which individual decision trees are grown, with higher values increasing the chance of overfitting. We used 10, 100, and None for this parameter.

We chose the f1 score as the metric to maximize for our grid search, which resulted in the following model being chosen:

- `max_features` = auto
- `min_samples_leaf`= 1
- `class_weight` = balanced_subsample
- `max_depth` = 100

XGBoost

There is a lot of buzz that XGBoost is often a very strong classifier, and better than the Random Forest model. XGBoost is a GBM (generalized boosted model), and like RF, it uses decision trees. However, instead of completely randomizing the predictors in those trees, it learns from each previous tree and includes regularization.

Using the xgboost and sklearn packages, we tuned a XGBoost classifier using grid search with 5-fold cross-validation on the following parameters, as our research indicated these to be most crucial:

- gamma : Minimum loss reduction required to make a further partition on a leaf node of the tree.
- min_child_weight : Minimum sum of instance weight(hessian) needed in a child.
- max_depth : Maximum tree depth for base learners.
- subsample : Subsample ratio of the training instance.
- colsample_bytree : Subsample ratio of columns when constructing each tree.
- reg_alpha : L1 regularization term on weights

We also selected a relatively slow learning rate and large number of trees to be used.

Again, we chose the f1 score as the metric to maximize for our grid search, which resulted in the following model being chosen:

- `gamma` = 0.2
- `min_child_weight`= 0
- `max_depth` = 5
- `subsample` = 0.6
- `colsample_bytree` = 0.9
- `reg_alpha` = 0.1
- `eta` = 0.1
- `n_estimators` = 1000

Model Results

We evaluated model performance on our testing set (30% of the original dataset or ~3,000 records)

Random Forest

The model performed significantly better than a random classifier, with performance metrics on the test set as follows:

- `accuracy` = 0.4284
- `precision` = 0.4244
- `recall` = 0.4284
- `f1` = 0.3945

We visualized the model's performance using both confusion matrices and ROC curves. The confusion matrix for the test set showed that the model performed well on genres like Documentary, Drama, and Family, but more poorly on genres like Comedy - Drama, Comedy - Romance, and Drama - Thriller. More problematic was that the model's performance on the training set yielded 100% accuracy, indicating potential overfitting during training. However, when we ran the model using smaller amounts of predictors (filtering based on feature importance), all of the performance metrics mentioned above declined as well. This bears further investigation.

To see the model and notebook visualizations, please see: https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/Random%20Forest%20Model.ipynb (https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/Random%20Forest%20Model.ipynb)

XGBoost

The XGBoost model had the following results:

- `Training Accuracy` = 100%. Across 11 classes, this was somewhat concerning.
- `Test Accuracy` = 41%
- `Precision Score` = 40%
- `Recall Score` = 41%

The most impactful features were `popularity`, `overview_length`, `plot_length`, `plot_outline_length`, and `cast`. It is surprising that `overview_length`, `plot_length`, and `plot_outline_length` had such high feature importance.

Additionally, it's important to consider that our 41% accuracy was achieved across 11 classes, and that these classes have overlap. For example, it's not surprising that it's hard to distinguish between Comedy and RomCom.

To see the model and notebook visualizations, please see: https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/XGBoost%20Model.ipynb (https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/XGBoost%20Model.ipynb)

Model Differences

While random forest grows trees independently of other trees, xgboost looks to minimize the residuals produced by the model by growing additional trees based on the residuals of the previous trees. This can lead xgboost to perform better than random forest if tuned correctly but also makes it more prone to overfit if the model fits the residuals too well. Tuning the XGBoost model takes longer as well since trees are not grown separately and there are more parameters to tune.

Model Improvement

There are a few ways we could improve these models:

- `Tune on More Parameters`: We could look tune more parameters on the models, both by including completely new parameters (varying the `criterion` in the random forest model) or increasing the number of values for existing parameters (testing 500, 1000, and 2000 `max_features` for random forest)
- `Include More Features`: We did attempt some PCA on a subset of the movie posters but did not do it for the entire dataset. IMDb also offers more features for the movies so we could include those. We could also try more clustering techniques, including agglomerative ones, on the text variables and see if the results improve performance.
- `Change Genre Configuration`: Since we combined various genre classes together based on personal judgement, we might have combined some incorrectly making differences between the genres harder to distinguish than they are in reality.
- `Consider Outputting Probability Distributions`: Due to the overlap between our genre categories (as some are defined as intersections of others), it might be interesting to see what the model's probability distribution by category looks like. For example, is it confident that movies whose actual label contains 'Comedy' fall under some category containing 'Comedy'? Or is it also predicting that a movie whose actual label contains 'Comedy' is likely to fall into 'Documentary' too?

PCA and SVM

As noted above, we began to explore the posters data by using PCA and multi-class prediction with SVM on a random subset of the movie posters (400 train and 400 test). The performance was poor - about 19% test accuracy with tuned linear and radial kernels - and the computation time was very high. We view this exploratory analysis as a useful benchmark or point of comparison for the deep learning models.

The Rmd code is available here: https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/PCA-SVM.Rmd (https://github.com/All-Star-Vipers/CS109B-Final-Project/blob/master/Milestone%203/PCA-SVM.Rmd)

And the HTML output from the Rmd file is available at this link: https://all-star-vipers.github.io/CS109B-Final-Project/Milestone%203/PCA-SVM.html (https://all-star-vipers.github.io/CS109B-Final-Project/Milestone%203/PCA-SVM.html)