**Conclusion**

The business problem that we are trying to solve is helping dispensary owners to determine which strains of cannabis they should stock. Our analysis has provided some valuable insights for this, and we have produced a workable model that will be able to achieve our objective.

In the first section of our analysis we explored the data. Since the features that we had to work with were all text, this involved extensive use of the NLTK library, as well as a number of visualizations, to understand the data set. Apart from giving us an understanding of the types of things that were said about each strain, this section of the analysis gave us two significant insights.

The first is that we would need to transform our target variable to make it more useful to the models. This was required because the users who had submitted their descriptions and scores for each strain had, in general, nearly always chosen a score between 4 and 5 (on a scale of 1-5). This meant that a successful regression would end of predicting nearly the same value for all strains. In order to overcome this, we transformed the target variable – the rating – by separating the samples into bins based on their scores. These bins represent strains that should not be stocked, that should be considered, and that should definitely be stocked. This allowed us to approach the problem as a classification problem rather than a regression problem, and had the added advantage of making our models’ outputs more intelligible to end users. The results are more intelligible because a shop owner would be better served with a class telling him or her to keep or not to keep the strain rather than a number of scores that would in general fall between 4 and 5.

The second insight is that our data set was unbalanced. The data as we found it had nearly 60% of examples in one class. When the models were run using this unbalanced data, they all eventually determined that the best model was one that always picked the overrepresented class, and by doing so achieve 60% accuracy. In order to fix this we oversampled the under represented classes so that the data had a roughly equal balance between the three classes.

The second section built on our insights from the first. Because the features were all text, our approach was different than it would have been for a dataset with numerical features.

First, the data had to be cleaned with NLTK. To do this we wrote a function that made all of the words lower case, removed the stop words, removed punctuation and special characters, stemmed the word, and tokenized them (where applicable).

After this, two data preprocessing methods were tried. The first used scikit-learn’s count vectorizer to turn the cleaned sentences into numerical features, and then used TD-IDF to get the importance of each word in the dataset. The second used gensim’s Word2Vec model to build a one layer neural network to predict probabilities of words appearing near each other in the dataset. With this model built, we extracted the word embeddings – the weights in the hidden layer – for each word, and then reconstructed the feature matrix by taking the mean of each word’s associated vector.

Once the data had been cleaned, we applied the same four models to both preprocessed feature vectors. These models were KNN, Random Forest Classifier, SGD Classifier, and a neural network with Keras. The scikit-learn models were all tuned with a 3 fold grid search on the two datasets, and so were optimized for each of them. The Keras model was built using 4 hidden layers with 5 nodes each, and relu as the activation function. The model was set to run for 50 epochs but with early stopping (patience=3).

All models were evaluated on their training data using 3 fold and 10 fold cross validation. Using this we were able to ensure that the models were not overfitting.

These models performed similarly on both preprocessed datasets. However, the dataset preprocessed with scikit learn had over 10,000 columns in the feature matrix, while the one preprocessed with genism had only 100. Because of this, we determined that the genism method of getting the feature matrix was superior, because it produced comparable results with far less training time.

The accuracies of the models were:

Scikit-learn

The test set accuracy for the SGD model is: 0.49206349206349204

The test set accuracy for the Random Forest model is: 0.8123249299719888

The test set accuracy for the KNN model is: 0.5191409897292251

The test set accuracy for the Keras model is: 0.46872082188461117

Gensim

The test set accuracy for the SGD model is: 0.4565826330532213

The test set accuracy for the Random Forest model is: 0.8356676003734828

The test set accuracy for the KNN model is: 0.5284780578898226

The test set accuracy for the Keras model is: 0.43884220388200545

As can be seen above, the SGD, KNN, and Keras models all performed relatively poorly, and were underfitting the data. Furthermore, they all performed more or less comparably. Train set and test set accuracy were similar in all cases, using both 3 fold and 10 fold cross validation on the training sets. The Keras model was improved to show as high as 70% accuracy on the test set, but this came with a training set accuracy of 99%, and so was clearly an example of overfitting.

However, in both cases the Random Forest model performed well. The accuracy on the training set and test set was comparable, and in fact was higher on the test set.

Our findings from these models is that only the random forest was able to successfully capture the complexity in the feature matrix without overfitting the data. However, its accuracy needs to be taken in perspective. Because the dataset was modified to oversample the under represented classes, it is almost certainly the case that the test set contained some data that was duplicated from the training set. Furthermore, the random forest model performed worst with 3 fold cross validation (around 72%), second worst with the 10 fold cross validation (around 78%), and best on the test set (around 82%). This is odd behaviour and may be related to the fact mentioned above – that some data in the test set is a copy of data in the training set. However, in any case the algorithm attained an accuracy of between 72% and 82%, and so was a successful model.

Precision and recall were also analyzed for all models. The Random Forest model displayed balanced precision and recall – they were similar to each other, and each class had precision and recall similar to the overall accuracy of the model. The tables below show the two metrics for the two methods of preprocessing the data:

Scikit-learn

precision recall f1-score support

0 0.84 0.93 0.88 323

1 0.82 0.75 0.78 403

2 0.78 0.78 0.78 345

accuracy 0.81 1071

macro avg 0.81 0.82 0.81 1071

weighted avg 0.81 0.81 0.81 1071

Gensim

precision recall f1-score support

0 0.87 0.93 0.90 323

1 0.83 0.80 0.81 403

2 0.81 0.80 0.80 345

accuracy 0.84 1071

macro avg 0.84 0.84 0.84 1071

weighted avg 0.83 0.84 0.83 1071

Our conclusion is that the random forest is a good working model and has potential to perform well in the proposed business context. Our recommendation is for dispensary owners to use this algorithm to test proposed strains based on user input so that they can determine which to stock.

In order to improve the model, there are several steps that could be taken. The first is to get more training data, particularly in the under represented classes. This would improve the model’s performance, and would also limit the need to over sample. Secondly, the model would benefit from further hyper-parameter tuning. Given the complexity of the model (high n\_estimators and high max\_depth), the random forest classifier took a very long time to perform grid search. A possible improvement would be to do a randomized search on a greater number of hyper parameters in order to determine where to focus the full grid search.