Predicting Success of Literary Fiction

Joy Zhong and Will Zhou

Machine Learning and Statistical Data Analysis, Winter 2015

1 Introduction

In this study we use random forests to build a model that predicts the success of literary works with up to 76% accuracy. While researchers have studied similar topics such as readability in academic articles [1] and quantitative assessment of English fluency in text [2], the contrast between good and excellent writing in fiction is a relatively novel task. We closely model our problem based on a 2013 study that used SVM's to predict the success of literary works with up to 84% accuracy for a single genre [3].

One motivation for predicting the success of novels comes from the needs of publishers to quickly distinguish potentially popular books from the thousands of manuscripts they receive. Additionally, by using random forests–which are generally considered more interpretable than SVM's–we are able to isolate certain features of text such as vocabulary, syntax, and sentiment that correlate with successful or popular writing. Although there exists a wealth of literature on good writing based on qualitative observations, there are fewer studies based on quantitative analyses, perhaps because of the complexity of the problem and the numerous factors that play into the success of a novel. We hope that our work contributes to a growing literature on providing quantitative analyses of what constitutes "good" and successful writing.

2 Methodology

2.1 Data

In order to directly compare results with [3], we use the same dataset from the Project Gutenberg database of novels available at [4]. The dataset is organized by genre (Adventure Stories, Fiction, Historical Fiction, Love Stories, Mystery, Science Fiction, and Short Stories), with each genre containing 100 novels (half success, half failure) for a total of 700 novels overall. Another reason we chose this particular dataset was the access it provided to full texts of published novels. Because most modern novels are unavailable for free due to copyright issues, the public domain novels from Project Gutenberg were an appropriate alternative.

We use the download count of a novel from Project Gutenberg as the metric for success. Following [3], novels with over 80 downloads are classified as successes while novels with under 25 downloads are classified as failures. Success novels have a median of 225 downloads while failure novels have a median of 10 downloads. Training and test set comprise 80% and 20% of the data respectively and we perform four-fold cross-validation on the training set.

2.2 Feature Selection

We use tree-based feature selection to select the top 100 most informative tf-idf unigrams. Other simple metrics include the average number of words per sentence and the average word length. More linguistically informed features we used were the average number of POS (part-of-speech) tags per sentence (e.g. average number of verbs or adjectives per sentence). POS tags are generated according to the tags outlined in the Penn Treebank Project [5].

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Figure 1: Information gain metric, where S is the set of training examples and A is the feature to sort on.

We also use higher-level features, performing LDA (Latent Dirichlet Analysis) for topic modeling, along with genre, sentiment analysis, and readability metrics as measured by the Gunning Fog index and Flesch-Kincaid test. Finally, we measured the averaged log-likelihood of each text in relation to a corpus trained on parts of Project Gutenberg and the Brown Corpus.

2.3 Random Forest Algorithm

We separate the construction of random forests into two steps: building individual decision trees and inserting randomness in how all trees are grown in order to obtain a well-varied forest of trees.

The algorithm we wrote for building individual decision trees was the bulk of our implementation. The core part of growing these trees is deciding which feature to "branch" on during learning, and there are a number of metrics to do this. Among the most common are the Gini impurity metric and information gain metric. We used the information gain metric as shown in Figure 1 to determine the best feature to branch on at each split. For more implementation details, see Appendix below.

We insert randomness into the construction of each decision tree in two ways. The first method is through bagging: before constructing each tree, we randomly select a fraction of the data (with or without replacement) for the tree to be trained on. The second method is by randomly constraining the features that can be selected at each branch of the decision tree. When cross-validating, we tune the aforementioned two hyperparameters along with the maximum depth of each decision tree. The final, cross-validated hyperparameters for the random forest were a max depth of 13, bagging with replacement using the full dataset, and randomly selecting 10% of features to branch on at each step.

2.4 SVM Comparison

The SVM we used as a comparison to our random forest was an SVM using a radial-basis kernel. When cross-validating the SVM, we found that in general, the SVM had a large positive bias in classification (*i.e,* the SVM had too many false positives in predicting successes). To ameliorate this issue, we selected the best-tuned SVM based not on accuracy, but F1 score, which is the harmonic mean between precision and recall. We found this removed the tendency in positive bias, and the final hyperparameters for the SVM were a slack penalty C of 1000 and a gamma of .00093 for the radial basis kernel.

3 Results and Discussion

Our main results, which show overall accuracy on our four-folds of training data and our test set are in Figure 2. Accuracy on training data hovers around 70% using random forests and is generally slightly worse using SVM's. However, accuracy on the test set for random forests is 76%, which is significantly better than the 69% given by SVM. Interestingly, test accuracy is even higher using random forests than cross-validation accuracy. This may be due to the fact that tuning the hyperparameters of the random forest did not significantly boost accuracy in the final test set, and thus the extra 20% of data on the test set likely drives the increased performance, especially since our dataset is small.

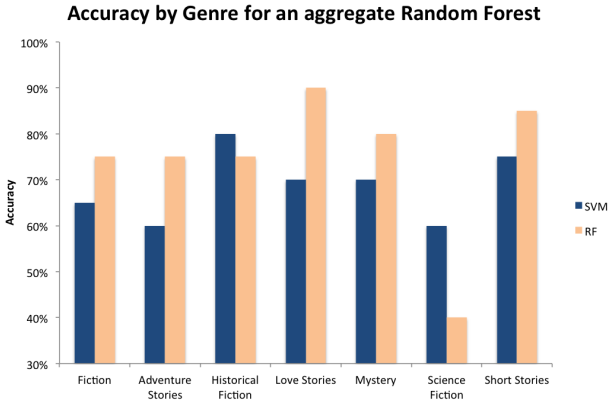

Figure 2: Overall Accuracy

3.1 Accuracy within each Genre

We note that the 84% accuracy given in [3] for SVM's is for a single genre, whereas our benchmark SVM accuracy is across all genres. To give a better comparison and insight on how random forests and SVMs performed across genres, we plot their accuracies in Figure 3 (left). The resulting graph demonstrates that on the high-end, our SVM benchmark approaches 80% accuracy for historical fiction and short stories which is comparable to the high-end accuracies in the previous study.

More importantly, our results show extremely high accuracies for random forests across most genres–90% for love stories and 85% for short stories, which suggests that random forests are perhaps very well-suited for predicting the success of literary fiction. Strikingly, random forests only achieve 40% accuracy on the science fiction genre while they achieve around 80% accuracy averaged across all other genres. This may indicate that science fiction as a genre is the most different in terms of what features predict literary success. If we were to look at the primary audience across genres, it is likely that science fiction readers would differ most in their preferences in a good novel. In addition, science fiction is often very different in writing style from the other genres.

Another way to analyze how well our features and learning algorithm predict within each genre is by training a separate random forest for each genre. The weakness in this approach is that less training data is used for each forest, and in this particular case, the approach only allows 80 training examples and 20 test examples for each genre. Thus, we caution that the accuracies of this method are less reliable.

Figure 3 (right) shows the accuracy for each genre when separate random forests and SVM's are trained on each genre. In general, it appears that accuracy is lower following this approach, although a separately trained random forest on science fiction does achieve better performance. On the other hand, love stories, which previously had a 90% accuracy, are no longer classified well. Either we do not have enough training data for this approach, or significant cross-genre information indicative of success is lost when training separate random forests.

3.2 Feature Importances

Besides training an algorithm for predicting literary success, another primary motivation for this study is to determine which types of features are most helpful in identifying well-written novels, and which features are indicative of success or failure. To analyze the features by these measures, we first look at how helpful features were based on their average information gain in random forests. Secondly, for select features we run univariate analyses to determine

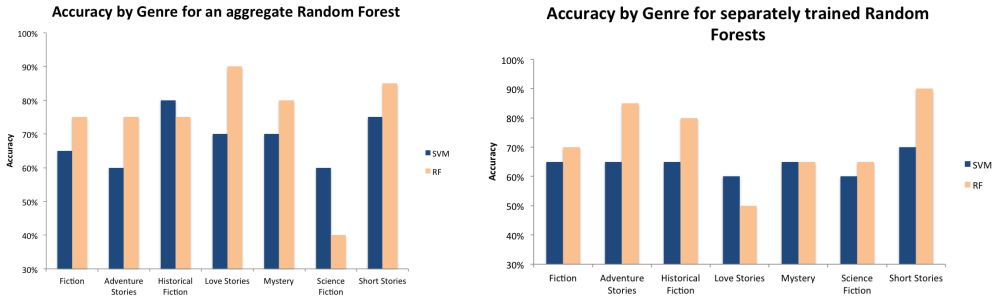

Figure 3: Accuracy by Genre

Most useful	Information Gain	Least useful	Information Gain
Foreign word	0.0166	Proper noun, plural	0.0017
Verb, base form	0.0165	Symbol	0.0025
Possessive ending	0.0160	Pos. Wh-pronoun	0.0059
Interjection	0.0144	Adv, Comparitive	0.0074
Cardinal number	0.0142	Adjective	0.0077

Figure 4: Most and Least Informative POS Tags

whether the features are generally positively or negatively correlated with success. In reality the relationship between each feature and success can be non-linear.

3.2.1 Simpler Features, Readability Metrics & Log-likelihood

Unsurprisingly, our simpler features did not appear to help much in classification once we added more complex features, as their average information gain was low at around .005, which was only a bit more informative than some of the least informative part of speech tags. Additionally, readability metrics were similarly unhelpful, which is consistent with [3], as the previous study finds very little difference in FOG index and Flesch index for successful and unsuccessful novels. The general unhelpfulness of readability metrics likely explains why average log-likelihood of words in the novel texts was not important as well, as [1] finds that metric to be very correlated with readability.

3.2.2 Part-of-Speech Tags & Sentiment

The average proportion of part-of-speech tags in each sentence of the text was among one of the most important feature sets in our random forest. In fact, apart from unigrams, only part-of-speech tags and sentiment appeared to affect the predictive accuracy in the learning algorithm. In Figure 4, we show the most important and least important part of speech tags based on information gain. Foreign words, verbs in their base form, possessive endings, interjections, and the presence of numbers were all important, while adverbs and adjectives were among some of the least informative. The latter finding may indicate that the proportion of adverbs and adjectives does not reveal much about the quality of a novel.

To determine how the more important features actually affect a novel's success, we ran univariate analyses on each feature. First, we determine the median difference in average proportion of a particular part of speech tag between successes and failures (successes minus failures) and secondly, we determine the correlation coefficient of each feature with success. The results are shown in Figure 5.

Unexpectedly, we find that cardinal numbers are strongly correlated with success. This

Univariate Analysis of Select Features	Median Difference in Proportion	Correlation Coefficient
Cardinal Number	0.016	0.156
Verb - Base form	-0.090	-0.038
Sentiment	-0.010	-0.138
Possessive Ending	-0.0070	-0.118
Interjection	-0.0018	-0.149

Figure 5: Univariate Analysis of Important Features

finding is unusual, as numbers would intuitively seem be more important in data-driven articles or texts but only fictional works are included in our dataset. We also find that verbs in their base form, possessive endings, interjections, and sentiment are negatively correlated with success.

The relationship between sentiment and success is especially puzzling, as one might hypothesize that positive novels would be more popular with readers than negative novels. However we find that sentiment is not only very important in terms of information gain but is also strongly negatively correlated with success. This result is better explained when we look at the range of sentiment values across the whole dataset, which actually spans from neutral sentiment to positive sentiment, with very few novels having negative sentiment. Thus the negative correlation may indicate that more neutral novels are successful while overly positive novels are subject to failure.

3.2.3 LDA and Genre

The features discussed previously are all related to writing style, while LDA and genre are related to the content of a novel. Therefore, one might reason that these features would provide additional, non-duplicative information for predicting successful novels. In contrast, we find genre to be by far the least informative features in our dataset, with average information gains of less than .001 (in comparison, our simpler features had average information gains of roughly .005). LDA was more useful based on this metric, but did not contribute to overall test accuracy in our final results. In fact, our random forest only misclassified one more example when tested on a feature set that excluded LDA and genre (for brevity, we omit a table or graph of these results as they are roughly equivalent to our presented results in Figure 2). This margin of underperformance is unlikely to be significant as a different random seed for our learning algorithm would easily yield a slightly different result.

There are a couple interpretations for these findings. One interpretation is that our features as an aggregate are successful in revealing successful writing irrespective of genre or high-level topics, as evidenced by the lack of helpfulness in these content-based features. However, it may also be the case that these features actually duplicate much of the information present within unigrams, as it is unclear whether the top 100 tf-idf unigrams already serve as a good proxy for the content of a novel.

3.2.4 Unigrams

Unigrams were one of our strongest drivers in predictive performance, but paradoxically, no individual unigrams appeared to be indicative of success. In fact, average information gain based off of individual words was low, and only a few unigrams (though, new, brought, began, end, having) reappeared in the top 100 most informative tf-idf unigrams across our cross-validated data.

As we can see from Figure 6, which shows some of our most informative unigrams, there does not seem to be a clear pattern for what words were important.

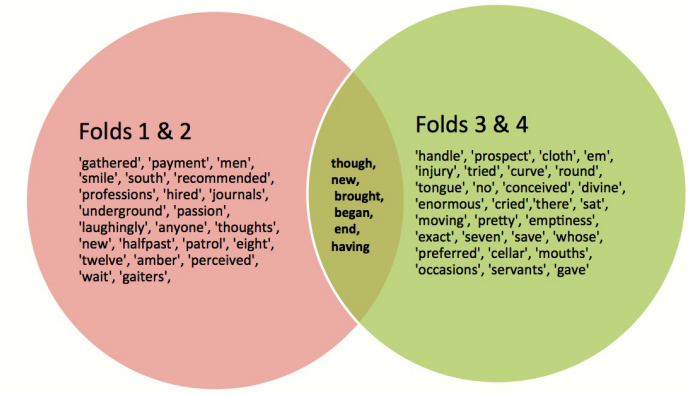

Figure 6: Most Informative Unigrams

4 Conclusion

In this study we learned a random forest model on different pieces of literary fiction for two primary motivations, firstly to present a predictive model for literary success and secondly to provide clarity on the writing characteristics that best define a successful novel. We find moderate success in both aspects, achieving an accuracy of 76% across genres despite having a very limited set of data, while also establishing unique insight on how specific parts of speech, sentiment, and vocabulary affect novel success.

Importantly, we show that a random forest model trained on our aggregate dataset does well predicting literary success for all genres except for science fiction, which suggests that separate features or models may be more important for science fiction. A random forest model also consistently outperforms SVM's across all our tests, so future researchers may find it useful to benchmark text classification algorithms using such tree ensemble methods– especially given that our random forest did not necessarily require tuning to achieve high accuracies.

Additionally, a more basic topic-based model (LDA) and genre did not provide any more information than unigrams in predicting literary success, which suggests that this content-based information is either not helpful or subsumed within the information unigrams already provide. We find that the proportion of part-of-speech tags, unigrams, and sentiment are the most important features within our model. Within parts of speech, the most important tags appear to be the proportion of numbers, foreign words, interjections, verbs in their base form, and possessive endings, which with the exception of numbers are all negatively correlated with success.

References

[1] Emily Pitler and Ani Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality. In Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing.

[2] Ani Nenkova, Jieun Chae, Anne Louis, and Emily Pitler, Structural Features for Predicting the Linguistic Quality of Text: Applications to Machine Translation, Automatic Summarization and Human-Authored Text, Empirical Methods in Natural Language Generation: Data Oriented Methods and Empirical Evaluation, 2010.

[3] Vikas Ganjigunte Ashok, Song Feng, and Yejin Choi. 2013. Success with style: Using writing style to predict the success of novels. In Proceedings of EMNLP.

[4] http://www3.cs.stonybrook.edu/s̃ongfeng/success/

[5] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

A Implementation Details

Our project code consists of two main parts–code to extract features from the text in the novels directory (saving these feature matrices to .pkl files) and code to run training and testing data. All written code can be found in the scripts directory.

The code breakdown is as follows:

A.1 Miscellaneous Files

Entirely written by us, these files are just quickly hacked up files to experiment with or look at data:

- data_features.py
- parseArpa.py

A.2 Feature Extraction

All features are coded manually with the exception of LDA, Sentiment analysis, and POS tagging (we do manually manipulate POS data after it is tagged on our own). However, in many cases we make use of the nltk library to tokenize the text in terms of sentences or detect syllable count.

The external libraries can be downloaded by first downloading the pip installation tool, from: `https://pypi.python.org/pypi/pip`. The libraries required thereafter are lda, nltk, numpy, scipy, sklearn, and TextBlob, which can be installed by typing pip install [name].

- features.py – main feature extraction
- pos_tags.py – used to save average POS tags per a sentence to save time
- trainAndTest.py – misnomer (the file only saves feature matrices into .pkl files)

A.3 Training and Testing

We write all the code for training and testing, including the implementation for a random forest. However, we use the built-in SVM and RF as comparisons or to quickly see rough results. The only external library in this case should be the sklearn library, which we use to tune and test built-in SVM's and built-in random forests.

- RandomForest.py – contains random forest code that uses decisionTree.py to grow trees. This file is also our main driver for tuning and testing.
- decisionTree.py – contains main code for growing decision trees
- TreeNode.py – class to model a decision tree node

A.4 Code Samples

We've included below the bulk of our algorithm implementation, including code snippets from the growth of individual decision trees and the random forest class.

decisionTree.py

```
1 # Recursively grow decision trees
2 MAX_DEPTH = 10
3 def growDecisionTree(X, y, depth = 0, mtry = None):
```

```
4         if mtry == None:
5                 mtry = X.shape[1]
6
7         X = np.copy(X)
8         y = np.copy(y)
9
10        majority = np.argmax(np.bincount(y))
11
12        if depth >= MAX_DEPTH or np.bincount(y)[majority] == len(y):
13                return TreeNode(category = majority)
14
15        bestFeature, bestSplit = chooseBestFeatureSplit(X, y, mtry)
16
17        leftIndices = np.where(X[:,bestFeature] <= bestSplit)[0]
18        rightIndices =  np.setdiff1d(np.array(range(len(y))),
              leftIndices)
19
20        node = TreeNode(bestFeature, bestSplit)
21        node.left = growDecisionTree(X[leftIndices,:],
              y[leftIndices], depth + 1)
22        node.right = growDecisionTree(X[rightIndices,:],
              y[rightIndices], depth + 1)
23
24        return node
25
26 def chooseBestFeatureSplit(X, y, mtry):
27        allFeatures = np.array(range(X.shape[1]))
28        tryFeatures = np.random.choice(allFeatures, mtry, replace =
              False)
29        tryFeatures = np.sort(tryFeatures)
30        bestFeature = None
31        bestSplit = None
32        bestInfoGain = -np.inf
33        for feature in tryFeatures:
34                infoGain, split = calcBestSplit(feature, X, y)
35                if bestInfoGain < infoGain:
36                        bestInfoGain = infoGain
37                        bestFeature = feature
38                        bestSplit = split
39
40        return bestFeature, bestSplit
41
42 def calcBestSplit(feature, X, y):
43        bestSplit = None
44        bestInfoGain = -np.inf
45        uniqueVals = np.unique(X[:, feature])
46        for i in range(len(uniqueVals) - 1):
47                split = uniqueVals[i]
48                infoGain = calcInfoGain(split, feature, X, y)
49                if bestInfoGain < infoGain:
50                        bestInfoGain = infoGain
51                        bestSplit = split
52
53        return bestInfoGain, bestSplit
54
55 # Calculates negative entropy, because in maximizing info gain
56 # we are really minimizing entropy
57 def calcInfoGain(split, feature, X, y):
58        leftIndices = np.where(X[:, feature] <= split)[0]
```

```
59          rightIndices =  np.setdiff1d(np.array(range(len(y))),
                leftIndices)
60
61          labelA = y[leftIndices]
62          labelB = y[rightIndices]
63
64          # weights of class A and class B
65          wA = len(labelA) / len(y)
66          wB = 1 - wA
67          entropy = wA * calcEntropy(labelA) + wB * calcEntropy(labelB)
```

randomForest.py

```
1  class RandomForest:
2          def __init__(self, trees = 100, fraction = .3, mtry = 10,
               replace = True):
3              self.trees = trees
4              self.fraction = fraction
5              self.forest = None
6              self.mtry = mtry
7              self.replace = replace
8
9          def fit(self, X, y):
10                 obsNum = X.shape[0]
11                 bags = [np.random.choice(obsNum, int(self.fraction *
                       obsNum),
12                  replace = self.replace) for x in range(self.trees)]
13                 self.forest = [dt.growDecisionTree(X[idx], y[idx],
14                         mtry = self.mtry) for idx in bags]
15
16         def predict(self, Xtest):
17                 if self.forest == None:
18                         "Data has not yet been trained"
19                         return None
20
21                 votes = np.array([dt.predictLabels(Xtest, root) for
                       root in self.forest])
22                 predictions = np.round([np.mean(votes[:,i]) for i in
                       range(Xtest.shape[0])])
23
24                 return np.array(predictions)
```