**Method and Results of Experiments**

Rachel Rosenberg

<https://github.com/rrosenb1/rrosenb1_msia490_2019/tree/project>

The method I took for picking the best type of model to use was characterized by attempting to build several different kinds of models and then pick the best-performing one to further tune. In this paper, I will discuss specific pre-processing methodology, feature creation, multi-modeling methods, and tuning methods.

**Preprocessing**

For the initial modeling phase, where I built several kinds of models, I used the exact same dataset each time. This is a corpus of 600,000 blog posts from 20,000 different blogger.com authors in 2004. It is labelled with, and I am predicting with, each author’s zodiac sign.

To clean for modeling, I manipulated the data to have one row per post (rather than one row per author), created the label as a category, and then performed standard NLP preprocessing steps of stripping, converting to lowercase, removing numbers, removing stop words, and lemmatizing. I originally stemmed words, but found that I could train much faster and without performance loss if I lemmatized with the NLTK package instead.

**Feature Creation**

I used TF-IDF word vectors as features for these models, as is standard practice in text classification modeling. A large part of the experimentation I did involved manipulating TF-IDF parameters; I manipulated min\_df, the normalization method (L1 or L2), and ngram\_range, keeping a constant max\_df of 0.8. I also scaled the

**Multi-Modeling**

For the initial modeling phase, I used about 140,000 rows of this dataset, vectorized with min\_df = 100, unigrams only, balanced datasets, and L2 regularization, in order to keep everything constant. I split into training and testing sets, with 33% of the data going to testing, and built each type of model once. Results seen below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model Name** | **Accuracy** | **F1-Score** | **Notes** |
| Logistic Regression | 0.371 | 0.370 |  |
| Decision Tree Classifier | 0.314 | 0.314 |  |
| Random Forest Classifier | 0.321 | 0.319 | Very long training time |
| Support Vector Machine | 0.368 | 0.367 | Standard for NLP applications |
| K Nearest Neighbors | 0.267 | 0.257 |  |
| MLPC Classifier | 0.265 | 0.105 | SKLearn built-in neural network model |

I also investigated confusion matrices for each model type to ensure that there were no concerning patterns or model behaviors. For each model type, these came out fairly homogeneous, with a large number of misclassified samples evenly distributed. However, the MLPC Classifier (the Sklearn-based Neural Network) classified everything as Type 2 (the Earth type), because classes were imbalanced and there was no class-balance parameter. It is important to make sure that misclassified samples are evenly distributed to rule out unbalanced class errors. Unbalanced classes are also the reason that I tracked F1-Score in addition to accuracy.

Overall, the **logistic regression** model performed the best of these 6 types. I moved forward with this model for parameter tuning.

**Final Model Tuning**

To tune the logistic regression model, I first manipulated TF-IDF parameters of min\_df, ngram\_range, and regularization type. Results below (green denotes accuracy > 40%, orange denotes accuracy < 35%):

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **F1-Score** |
| {'min\_df': 25, 'ngram\_range': (1, 1), 'penalty': 'l1'} | 0.360 | 0.358 |
| {'min\_df': 25, 'ngram\_range': (1, 1), 'penalty': 'l2'} | 0.404 | 0.404 |
| {'min\_df': 25, 'ngram\_range': (1, 2), 'penalty': 'l1'} | 0.353 | 0.345 |
| {'min\_df': 25, 'ngram\_range': (1, 2), 'penalty': 'l2'} | 0.400 | 0.400 |
| {'min\_df': 25, 'ngram\_range': (1, 3), 'penalty': 'l1'} | 0.363 | 0.354 |
| {'min\_df': 25, 'ngram\_range': (1, 3), 'penalty': 'l2'} | 0.403 | 0.403 |
| {'min\_df': 50, 'ngram\_range': (1, 1), 'penalty': 'l1'} | 0.356 | 0.351 |
| {'min\_df': 50, 'ngram\_range': (1, 1), 'penalty': 'l2'} | 0.391 | 0.390 |
| {'min\_df': 50, 'ngram\_range': (1, 2), 'penalty': 'l1'} | 0.355 | 0.349 |
| {'min\_df': 50, 'ngram\_range': (1, 2), 'penalty': 'l2'} | 0.395 | 0.394 |
| {'min\_df': 50, 'ngram\_range': (1, 3), 'penalty': 'l1'} | 0.351 | 0.344 |
| {'min\_df': 50, 'ngram\_range': (1, 3), 'penalty': 'l2'} | 0.392 | 0.391 |
| {'min\_df': 75, 'ngram\_range': (1, 1), 'penalty': 'l1'} | 0.350 | 0.347 |
| {'min\_df': 75, 'ngram\_range': (1, 1), 'penalty': 'l2'} | 0.382 | 0.382 |
| {'min\_df': 75, 'ngram\_range': (1, 2), 'penalty': 'l1'} | 0.349 | 0.344 |
| {'min\_df': 75, 'ngram\_range': (1, 2), 'penalty': 'l2'} | 0.383 | 0.382 |
| {'min\_df': 75, 'ngram\_range': (1, 3), 'penalty': 'l1'} | 0.350 | 0.344 |
| {'min\_df': 75, 'ngram\_range': (1, 3), 'penalty': 'l2'} | 0.381 | 0.381 |
| {'min\_df': 100, 'ngram\_range': (1, 1), 'penalty': 'l1'} | 0.350 | 0.346 |
| {'min\_df': 100, 'ngram\_range': (1, 1), 'penalty': 'l2'} | 0.375 | 0.374 |
| {'min\_df': 100, 'ngram\_range': (1, 2), 'penalty': 'l1'} | 0.348 | 0.343 |
| {'min\_df': 100, 'ngram\_range': (1, 2), 'penalty': 'l2'} | 0.380 | 0.379 |
| {'min\_df': 100, 'ngram\_range': (1, 3), 'penalty': 'l1'} | 0.351 | 0.347 |
| {'min\_df': 100, 'ngram\_range': (1, 3), 'penalty': 'l2'} | 0.378 | 0.377 |

The pattern here is that lower min\_df, bigrams or trigrams, and L2 regularization make the best combinations. There are a few cases here where bigrams outperform trigrams.

Knowing this, I moved into the final tuning step to tune the model parameters. Keeping previous best parameters of min\_df = 25, ngram\_range = (1,3), and penalty = L2, I now manipulated max\_iterations, C (regularization strength), and multi-class solution method. I varied max\_iterations at values of 100, 200, and 500, C at values of 0.5, 1, and 5, and multi-class methods of ‘ovr’ and ‘auto’. The best combination was max\_iterations = 100, C = 1, and multi-class method = ‘ovr’. The accuracy of this model was 0.405 and the F1-Score was 0.405. The confusion matrix was as follows:

[[4596 2177 1997 2438]

[2336 4970 2227 2602]

[1987 2003 3990 2183]

[2377 2400 2142 4743]]

The pattern here is that…

**Conclusion**

Overall, the best model to fit this dataset is