**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.

**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 | 0.373 | 0.372 |  |
| Decision Tree Classifier | 0.319 | 0.316 |  |
| Random Forest Classifier | 0.329 | 0.323 | Very long training time |
| Support Vector Machine |  |  | Standard for NLP applications |
| K Nearest Neighbors |  |  |  |
| MLPC Classifier |  |  | 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. 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:

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **F1-Score** |
| {'min\_df': 25, 'ngram\_range': (1, 1), 'penalty': 'l1'} | 0.471 | 0.402 |
| {'min\_df': 25, 'ngram\_range': (1, 1), 'penalty': 'l2'} | 0.562 | 0.557 |
| {'min\_df': 25, 'ngram\_range': (1, 2), 'penalty': 'l1'} | 0.523 | 0.441 |
| {'min\_df': 25, 'ngram\_range': (1, 2), 'penalty': 'l2'} | 0.581 | 0.565 |
| {'min\_df': 25, 'ngram\_range': (1, 3), 'penalty': 'l1'} | 0.496 | 0.425 |
|  |  |  |
|  |  |  |

The pattern here is that lower min\_df, higher ngram\_range, and L2 regularization make the best combinations.

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. Results below:

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **F1-Score** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

The pattern here is that …

**Conclusion**

Overall, the best model to fit this dataset is