NLP 243 Assignment 1

Ethan Sin
UC Santa Cruz
elsin@ucsc.edu

1 Introduction

The task, given a labeled training dataset and an unlabeled test dataset, is to take a baseline machine learning model and explore different approaches there are to improve the accuracy in the model's ability to predict the labels of the test dataset.

These datasets are comprised of search utterances used to find some information about a movie and each utterance is labeled with the 'CORE RELATIONS', which in general is the category of information that the utterance means to find. For example, one such relation is 'movie_starring_actor' for an utterance like 'who was the main actor in the exorcist', meaning that the utterance is written to find out which actor or actors were in a particular movie. There are 18 possible labels in the dataset, including a 'none' label.

With that in mind, the task we have is a supervised relation extraction problem as we are identifying these core relations using a dataset that has labels that we will train our model on.

2 Models

In all the models tested for this task, the embedding method I used is one-hot encoding. This allowed me to represent my data as numerical for ease of use with different models. It is easy to understand and implement as well, giving someone with a weaker computer science background like me the ability to explore different models with less frustration. It may be worth acknowledging the weakness of one-hot encoding with data like ours. Since the utterances in the dataset are relatively short, the existence of a word alone might not be enough to distinguish between core relations.

It is also worth noting that all models that I am testing are built off of the starter code given to us in section, to vectorize the data and optimize it for use in a model. I adjusted it to build a frequency distribution and include part of speech tagging with NLTK[1] for easy data pre-processing.

2.1 Multinomial Naive Bayes

The first model I experimented with was a Multinomial Naive Bayes classifier. For this and all other models I used the Scikit Learn library[6]. This model uses the Bayes probability that calculates the probability of a class given a document. As we are using one-hot encoding, the features of the document are whether or not a word appears in the document. The model is trained completely on probability, and does not have any significant hyperparameters that can be tuned.

2.2 Decision Tree Classifier

The second model I used was a Decision Tree classifier. This model is trained by building a tree that chooses the next node it travels to based on certain features in the data. It is very prone to overfitting as it can build a complete tree with completely pure leaves to match the validation set. Thus, it has certain hyperparameters such as class weights, controlling the depth of the tree at a certain hard depth or by pruning, and the minimum amount of samples the model needs to see before it makes a split in the tree[4][5].

2.3 Support Vector Classifier

The third model I used was a Support Vector classifier. This maps out the dataset as a set of points based on the features and tries to find the most appropriate hyperplanes to section off the dataset into groups for their labels. There are some relevant hyperparameters such as the C parameter, which determines how much leeway the model has when setting the margins for the hyperplanes, minimizing or allowing more datapoints to remain in an incorrect grouping. There is also the whether or

not you implement the kernel trick and the gamma of the radial basis function[8].

2.4 Logistic Regression

The last model I used was a Logistic Regression classifier. This model finds a mathematically representable 'line' that best fits the data. Unseen datapoints should be able to be estimated by inputting the features of the new data and finding the value of the class at that point in the 'line'[7]. Important hyperparameters are the regularization strength and penalty that are used to prevent the model from fitting too well[3].

3 Experiments

Most of my time experimenting was spent trying to understand how the starter code processed the data features, how each of the models worked, and how the hyperparameters affect the model. As it took time and a lot of various reading and parsing through the code to understand these things, while I compared the four different models I only was able to do more hyperparameter tuning with the Decision Tree classifier. As such, the bulk of this section will talk about that.

For experimenting, I had stuck with the original 80-20 train-test split except randomized for the training and validation set, and I simply used the Kaggle submission as my metric for comparing to an unseen dataset. Initially I did not understood much of how this worked, so in hindsight I now understand I could try different methods to leave unseen data to test the model and to try to control the distribution of labels that are much more uncommon as initially I had not realized how imbalanced the dataset is.

As for data preprocessing, I had implemented the recommended 500 most frequent words only in the vectors and removed stop words.

3.1 Comparing Models

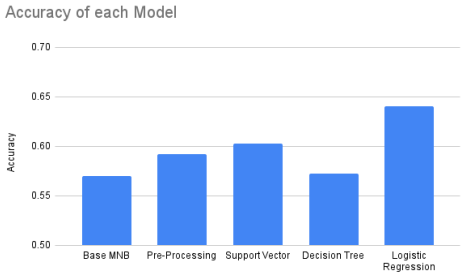
As a baseline, I had compared each model I listed above without any hyperparameter tuning. From there, I began to do research on what hyperparameters existed for Decision Tree classifiers and how to know what values to choose. After I experimented with those, I moved on to other models to test out a few hyperparameters but did not have time to do extensive experimentation with them.

3.2 Decision Tree Classifiers

While trying to understand hyperparameters, I first experimented with clearer strict parameters like max-depth. After that, I adjusted the min_samples_leaf, class_weight, ccp_alpha[2] and min_weight_fraction_leaf to try to prevent overfitting. For the values of these hyperparameters, I found algorithms that would help pinpoint what values to use but I was unable to implement them at this time.

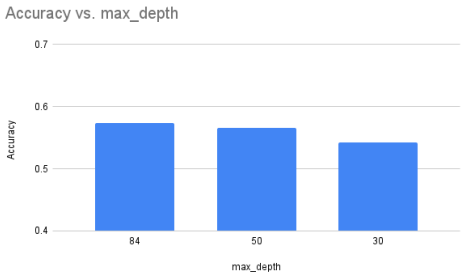
4 Results

Figure 1: Comparing all models



The baseline performance of each model on the test set can be seen above in Figure 1. The most notable things are that preprocessing improved performance by about 3 percent, Logistic Regression performed the best without any tuning, and Decision Tree performed the worst. Despite this, I decided to spend most of my time learning about the Decision Tree's hyperparameters. This is because it seemed like the model that would help me understand tuning hyperparameters the best as it is an easy to understand model and easiest to understand why it would overfit the data because of what I learned in class.

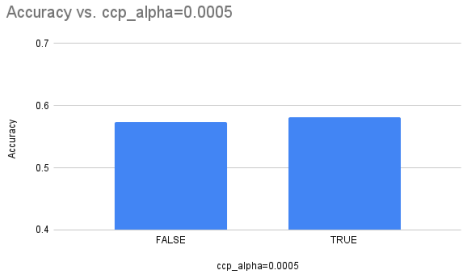
Figure 2: Adjusting max_depth



The hyperparameter I started with first is max depth. This was very easy for me to understand; put a hard stop to how deep the tree can grow as a

crude way to prevent the model from making too many very specific leaves that overfits on features to get perfect accuracy on the validation set. Each increment I tried lowered my final accuracy, however. When I saw this, I decided to move on to test other hyperparameters, but in hindsight I could have tested to see if smaller increments closer to the initial depth 84 could bring up the accuracy at all. This still taught me something valuable, as it got me to realize that highly imbalanced datasets like ours can be very challenging when it comes to overfitting, as the model most likely still needs some nodes that are deeper to learn how to label the utterances that are not part of the most common.

Figure 3: Changing ccp_alpha



I then continued to read up on more ways to tune the tree depth by pruning. I tried methods like min samples leaf to little success, until I got to ccp alpha. I found out that it is a post pruning method that cuts down on computation cost. It gives a metric to search for parts of the tree that can be cut out to yield better results. This was the only method that yielded better performance for me.

Figure 4: Using class_weight='balanced'

I also experimented with class balancing, where each class was given different weights for the generation of the tree inversely proportional to how often they occur in the dataset. I tried this in combination with all the pruning methods I used, and in every situation it yielded lower performance. I suspect that this is due to how rare certain labels are in the dataset, leading the model to overgeneralize more on those labels. If I were to experiment more with this, I would explore more ways to fine-tune the ratios in which the classes were balanced, rather than simply making the weights inversely proportional.

5 Conclusion

In the end, I learned a lot about the whole process of building a model for machine learning and the challenges of a real world dataset. With this assignment in tandem with lectures, I particularly learned a lot about what each of these classifiers do and how they do them, and their strengths and weaknesses along with ways to overcome them. While I did not yield very impressive results, I have a much better idea of how to improve my models accuracy should I face a task like this again.

References

[1] S. Bird, E. Loper, and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[2] S. Dash. Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning? https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-g 2022. [Online; accessed 20-October-2020].

[3] M. Gusarov. Do I need to tune logistic regression hyperparameters? https://medium.com/codex/do-i-need-to-tune-logistic-regression-hyperparamete 2022. [Online; accessed 20-October-2020].

[4] J. Ma. Construct a Decision Tree and How to Deal with Overfitting. https://towardsdatascience.com/construct-a-decision-tree-and-how-to-deal-with-over~:text=To%20prevent%20overfitting%2C%20there%20are,get%20rid%20of%20some%20branches., 2020. [Online; accessed 18-October-2020].

[5] M. Mithrakumar. How to tune a Decision Tree? https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680, 2019. [Online; accessed 18-October-2020].

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] A. Raj. Perfect Recipe for Classification Using Logistic Regression. https://towardsdatascience.com/

the-perfect-recipe-for-classification-using-logistic-regression-f8648e267592, 2020. [Online; accessed 15-October-2020].

[8] S. Yıldırım. Hyperparameter Tuning for Support Vector Machines — C and Gamma Parameters. https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167, 2020. [Online; accessed 20-October-2020].