Classifying Review Sentiment with Bag-of-Words Features

Natural Language Processing (NLP)

Hezekiah Branch

Tufts University, CS 135: Machine Learning, Professor Marty Allen
Submitted 19 December 2021

Summary

In this paper, we explore three types of machine learning models (Logistic Regression, Multilayer Perceptron, and Text Graph Convolutional Network) for the problem of text classification on a dataset of product reviews. Each review is attached to a binary score of 0 or 1, representing a positive or negative semantic value. These three model classes are hypertuned and cross-validated to search for the most optimal set of weights and hyperparameter configurations that best minimize loss, resulting in the lowest error and highest AUROC for an unseen testing dataset without publicly-available annotations. The problem of designing a robust preprocessing pipeline is also explored for investigating the relevance of order and representation during the learning phase.

Keywords: Natural language processing, machine learning, preprocessing, text classification, sentiment analysis

Introduction

In the field of information retrieval, there is a significant need to be able to successfully partition data points into distinct categories, hopefully without an unruly amount of human supervision. A popular example of an analytical approach to such a problem is the revisit to DBSCAN clustering, as seen with the Gan and Tao (2015) paper on p-Approximate DBSCAN. This proposal held the first proven solution to the previously-thought impossible problem of an O(n) unsupervised clustering algorithm for any dimensionality *d*. However, there is still the popular problem of classification for supervised data where we have access to features and labels that can be leveraged for predicting values on unseen data. In this project, we look specifically at text classification where we want to predict the sentiment of a particular collection of words (i.e. document).

The first task that we deal with is designing and building a robust preprocessing pipeline to handle text that may be unclean or unstructured. Additionally, we explore three potentially useful binary classifiers: Logistic Regression, Multilayer Perceptron, and the state-of-the-art Text Graph Convolutional Network. The “north star” metrics that will be used to evaluate the performance of these algorithms are:

(1) accuracy, (2) error on unseen test data, and (3) AUROC score on unseen test data, though other metrics will be discussed.

Preprocessing Pipeline

Rule-Based Preprocessing with spaCy

In order to find the most effective representation of the sentiment data, I decided to take advantage of the spaCy library for the following preprocessing steps: (1) punctuation removal, (2) stop word removal, (3) lemmatization, (4) part of speech (POS) tagging, and (5) stemming. The simplest way to conduct these steps was to use the rule-based matching system provided by the spaCy library. The idea was to experiment with different combinations and orders of preprocessing, an important problem that I observed in the papers I found on text classification (Kruczek et al., 2020; Violos et al., 2018). For each text classification, I experimented with various configurations of these 5 steps during cross-validation. The most important conclusion from this step was that the order mattered most for bag of words representation in comparison to term frequency-inverse document frequency (TF-IDF) representation. The results of which are available in the results section.

Table 1: Comparison of various parts of speech from training set of n=2400 documents

```
Oh oh INTJ False False
and and CCONJ True False
I I PRON True False
forgot forgot VERB False False
to to PART True False
also also ADV True False
mention mention VERB False False
the the DET True False
weird weird ADJ False False
color color NOUN False False
effect effect NOUN False False
it it PRON True False
has have VERB True False
on on ADP True False
your your PRON True False
phone phone NOUN False False
. . PUNCT False True
```

Bag of Words Representation

When designing the bag of words, I decided to keep the full set of words that appear in a single set and apply a dimensionality reduction method to determine the most pertinent words for the vocabulary. I also wanted to keep exclamation marks as a potential amplifier of sentiment (when not used in the context of a stop word). My cleaning function used the order of checking for stop words, punctuation, and non-alphanumeric characters before generating tokens. I quickly found this process to be time-consuming and decided to rely on the sklearn implementation to build the bag of words.

A potential pitfall that I encountered in the literature is the infamous lack of relational mapping between tokens in a bag of words given that the representation heavily relies on frequency. A sort of hack is to increase the magnitude of n to increase the likelihood of capturing possible connections. As such, I decided to create n-grams with n ≤ 3 (i.e. unigrams, bigrams, and trigrams) to compare model accuracy across varying types of n-gram.

The dimensions of the n-gram representations are as follows:

Unigram	2400x34146 sparse matrix with 53755 stored elements
Bigram	2400x17000 sparse matrix with 26608 stored elements
Trigram	2400x22208 sparse matrix with 24327 stored elements

Table 2: Dimensions of bag of n-grams models

Unfortunately, none of the models showed improvement outside of a mixed bag of unigrams and trigrams where each document $n_i \in \{1, 3\}$.

Term Frequency-Inverse Document Frequency

The term frequency-inverse document frequency (TF-IDF) representation remained the stand-out method for feature extraction, representing documents with a normalized word analyzer. For this project, I leveraged a univectorizer, bivectorizer, and trivectorizer similar to the unigram, bigram, and trigram approach in the bag of words representation. The vectorizers were built using a constant set of parameters that are as follows:

TF-IDF Parameter	Significance
lowercase=True	Change all characters to lowercase for uniformity
analyzer='word'	Analyze documents by word (English)
norm='l2'	Normalize using Euclidean distance for cosine similarity
use_idf=True	Frequency reweighting
smooth_idf=True	Smooth weights to prevent zero division

Table 3: Constant set of parameters used to build vectorizers for TF-IDF representation in sklearn

Originally, this method called for a count vectorizer followed by a TF-IDF Transformer model. However, leveraging the TF-IDF vectorizer was far more efficient and replaced the previous method with exact results. Unfortunately, this method did not safely remove English language stop words.

Method

Logistic Regression

For the initial binary classification model, I built a logistic regression model with 10 splits and varying levels of penalization, starting with the model that did not use any classification. The best set of hyperparameters was searched for using Randomized Search cross-validation, followed by Grid Search cross-validation. Initial results did not show a bias toward the grid method as finding the most optimal set of hyperparameters. The model development phase included three stages for each of the 3 types of vectorizers. As a result, all subsequent models were searched for using Randomized Search cross-validation exclusively. In total n=900 models were generated, though only 510 were fit due to conflicts in the hyperparameter space. The conflicts during search for the best performing representation, *unigram vectorizer with TF-IDF representation*, were as follows:

Table 4: Conflicts of hyperparameter configurations

Combination	Failure Quantity
newton-cg solver and l1 penalty	30 fits
lbfgs solver and l1 penalty	30 fits
liblinear solver and none penalty	60 fits
newton-cg solver and elasticnet penalty	30 fits
liblinear solver and elasticnet penalty	30 fits

Each logistic regression classifier was trained on the entire training dataset and chosen on best score and best set of parameters, as outputted by the Randomized Search cross-validation search and a Repeat Stratified K-Fold cross-validation evaluation. Trigrams had a close second in performance where bigrams performed the worst. The unigram TF-IDF output had a best Score: 0.81791 and best set of hyperparameters: {'solver': 'lbfgs', 'penalty': 'l2', 'C': 1}. With an increase of C penalization to 100, the leaderboard test performance showed an 0.16 error rate and 0.84 AUROC.

Below, is an example of all hyperparameters tested on logistic regression models *after* incorporating a split on training data (with no change to chosen hyperparameters to tune) in a later phase:

Table 5: Schedule of hyperparameters for logistic

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_solver	param_penalty	param_C	params	mean_train_score	std_train_score
0	0.101840	0.026455	0.000974	0.002262	lbfgs	none	1	{'solver': 'lbfgs', 'penalty': 'none', 'C': 1}	0.981667	0.000918
1	0.054309	0.005016	0.000575	0.001591	newton-cg	none	0.001	{'solver': 'newton-cg', 'penalty': 'none', 'C'...	0.981667	0.000918
8	0.071199	0.018740	0.000133	0.000426	lbfgs	none	0.1	{'solver': 'lbfgs', 'penalty': 'none', 'C': 0.1}	0.981667	0.000918
4	0.004613	0.002912	0.000356	0.000709	liblinear	l2	1	{'solver': 'liblinear', 'penalty': 'l2', 'C': 1}	0.981296	0.001382
9	0.011710	0.002775	0.000706	0.002096	newton-cg	l2	0.001	{'solver': 'newton-cg', 'penalty': 'l2', 'C': ...	0.979835	0.001564
5	0.002788	0.003641	0.000341	0.000950	liblinear	l2	0.0001	{'solver': 'liblinear', 'penalty': 'l2', 'C': ...	0.977037	0.001062

This configuration differs only from the previously best shown model in the choice for no penalty. This 0.981667 mean train score is high when splitting on the training data to generate test data. However, this data splitting choice showed clear lack of performance on the test data. Clearly, this set of parameters is misleading given the unwisely-sought design choice of splitting an already small data set. At the time of submission, the highly penalized logistic regression classifier was a top text classification winner both in this search space and at the #12 position on the leaderboard.

Multilayer Perceptron

Multilayer perceptron (MLP) models were fitted on TF-IDF matrices using the same approach as logistic regression, a combination of Randomized Search cross-validation for searching and evaluation with Repeat Stratified K-Fold cross validation for evaluation. The best models maintained 400 maximum iterations and used a limited memory BFGS (L-BFGS) solver. This solver relies on inverse Hessian matrices with limited updates, which may indicate a usefulness for a dataset of this type that has very few columns in its input, but a pattern of sparse matrices when generating its TF-IDF representations for the corpus. The sparsity approach of this solver may be a good pairing with the inputted representations given the solver's ability to handle high-dimensional matrices, even when there are non-differentiable components,

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_solver	param_max_iter
5	2.700843	0.436425	0.001847	0.000483	lbfgs	400
8	10.571233	0.565745	0.001623	0.002109	sgd	1000
9	2.862373	1.223768	0.002199	0.002600	lbfgs	800
2	39.877799	4.089589	0.002593	0.001164	lbfgs	400
4	25.070523	1.080219	0.003546	0.000870	adam	800
7	24.821764	1.398746	0.004074	0.002232	adam	1000
0	61.577916	8.712731	0.004064	0.002369	adam	200
1	1.281721	0.102541	0.001545	0.002297	sgd	200
3	11.060734	1.264423	0.001537	0.000582	adam	800
6	7.909126	0.229101	0.004167	0.000522	lbfgs	200

	param_learning_rate_init	param_learning_rate	param_hidden_layer_sizes	param_alpha	mean_train_score	std_train_score
5	0.1	invscaling	(7, 7)	0.01	1.000000	0.000000
8	0.1	constant	(7,)	0.1	1.000000	0.000000
9	0.1	invscaling	(7, 7)	0.001	0.999738	0.001413
2	0.0001	adaptive	(100,)	10	0.998904	0.000455
4	1000	adaptive	(100, 100)	0.00001	0.546929	0.035486
7	1000	constant	(100, 100, 100)	10	0.501775	0.008583
0	0.1	invscaling	(100, 100, 100)	100	0.500000	0.000000
1	100	constant	(7,)	0.0001	0.500000	0.000000
3	0.0001	constant	(7,)	0.1	0.500000	0.000000
6	0.01	adaptive	(100, 100, 100)	0.00001	0.500000	0.000000

Table 6: Schedule of hyperparameters for top 10 Multilayer Perceptron (MLP) models

An unintended but useful conclusion from splitting the training data in search of annotations for the unannotated test set was the distinctive lack of performance between n-gram ($n \leq 3$) models for MLP models. For each type of n-gram TF-IDF representation, the performance did not change for text classification of the training data.

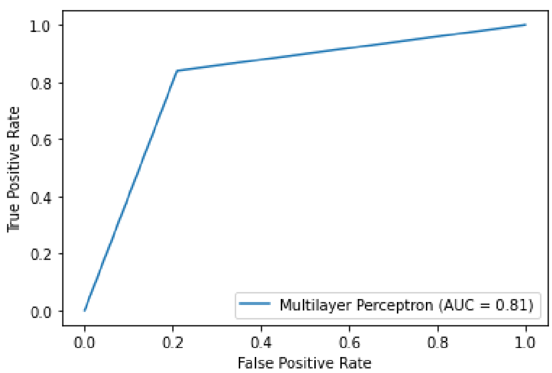

Figure 1: AUROC receiver plot of MLP model

Text Graph Convolutional Network

The third and last model used for text classification was a TensorFlow implementation of the Yao et al. (2019) paper on building a Text Graph Convolutional Network (Text GCN) for text classification. The benefit of this model is that it maps edges between corpus elements based on PMI (point-mutual information) then weights edges using TF-IDF. This mapping allows for semantic relations to be built in the representation of documents, improving the quality of information during retrieval. This state-of-the-art algorithm had the highest training accuracy of all models with 0.89120 training accuracy and 0.82500 validation accuracy on the training dataset. The lofty initial parameters for this graph convolutional neural network include an ADAM optimizer, a learning rate of 0.02, 200 training epochs, 200 units in the hidden layer, 0.5 dropout rate, 0 weight decay (i.e. weight for L2 loss on embedding matrix), tolerance of 10 epochs for early stopping, and a Maximum Chebyshev polynomial degree of 3.

,

Surprisingly, however, the Text GCN model showed the **weakest performance** on the test dataset with error=0.515 and AUROC=0.485. This model is the only one to remove stop words and builds a graph of features before training. The algorithm ran for 27 epochs before early stopping. The drastic difference in performance between the training phase and the test phase suggests that this network was overfitting to the training data, a possible reflection of the difference in size and availability of classes between this dataset and the ones leveraged in the original paper.

Table 7: Output of training accuracy versus loss and validation accuracy versus loss for Text GCN

```
2021-12-19 17:08:23.923640: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP
Epoch: 0001 train_loss= 1.09802 train_acc= 0.45324 val_loss= 1.05709 val_acc= 0.77500 time= 0.11606
Epoch: 0002 train_loss= 1.05613 train_acc= 0.78148 val_loss= 0.99723 val_acc= 0.78333 time= 0.03149
Epoch: 0003 train_loss= 0.99658 train_acc= 0.80000 val_loss= 0.92635 val_acc= 0.77917 time= 0.04690
Epoch: 0004 train_loss= 0.92344 train_acc= 0.80185 val_loss= 0.85295 val_acc= 0.77500 time= 0.03685
Epoch: 0005 train_loss= 0.84819 train_acc= 0.82222 val_loss= 0.78456 val_acc= 0.78333 time= 0.03776
Epoch: 0006 train_loss= 0.77745 train_acc= 0.82731 val_loss= 0.72574 val_acc= 0.77500 time= 0.04685
Epoch: 0007 train_loss= 0.71506 train_acc= 0.82824 val_loss= 0.67696 val_acc= 0.78750 time= 0.03124
Epoch: 0008 train_loss= 0.66075 train_acc= 0.83380 val_loss= 0.63614 val_acc= 0.78333 time= 0.04209
Epoch: 0009 train_loss= 0.61637 train_acc= 0.83889 val_loss= 0.60066 val_acc= 0.77917 time= 0.04239
Epoch: 0010 train_loss= 0.57545 train_acc= 0.83611 val_loss= 0.56837 val_acc= 0.77917 time= 0.04676
Epoch: 0011 train_loss= 0.53586 train_acc= 0.83750 val_loss= 0.53833 val_acc= 0.80417 time= 0.03798
Epoch: 0012 train_loss= 0.50398 train_acc= 0.83565 val_loss= 0.51073 val_acc= 0.80417 time= 0.04679
Epoch: 0013 train_loss= 0.47254 train_acc= 0.83889 val_loss= 0.48664 val_acc= 0.80417 time= 0.03784
Epoch: 0014 train_loss= 0.44551 train_acc= 0.84120 val_loss= 0.46623 val_acc= 0.80833 time= 0.03132
Epoch: 0015 train_loss= 0.41914 train_acc= 0.83750 val_loss= 0.44975 val_acc= 0.81250 time= 0.04688
Epoch: 0016 train_loss= 0.39567 train_acc= 0.84954 val_loss= 0.43689 val_acc= 0.81250 time= 0.03776
Epoch: 0017 train_loss= 0.38050 train_acc= 0.84583 val_loss= 0.42706 val_acc= 0.81667 time= 0.05072
Epoch: 0018 train_loss= 0.36437 train_acc= 0.85093 val_loss= 0.41872 val_acc= 0.81667 time= 0.03388
Epoch: 0019 train_loss= 0.34544 train_acc= 0.86157 val_loss= 0.41203 val_acc= 0.82917 time= 0.04688
Epoch: 0020 train_loss= 0.33596 train_acc= 0.86435 val_loss= 0.40808 val_acc= 0.83333 time= 0.03125
Epoch: 0021 train_loss= 0.32904 train_acc= 0.86713 val_loss= 0.40603 val_acc= 0.83333 time= 0.05346
Epoch: 0022 train_loss= 0.32077 train_acc= 0.86528 val_loss= 0.40630 val_acc= 0.83750 time= 0.03125
Epoch: 0023 train_loss= 0.31191 train_acc= 0.86852 val_loss= 0.40952 val_acc= 0.83750 time= 0.05338
Epoch: 0024 train_loss= 0.30574 train_acc= 0.87870 val_loss= 0.41053 val_acc= 0.82917 time= 0.04185
Epoch: 0025 train_loss= 0.29736 train_acc= 0.88333 val_loss= 0.40903 val_acc= 0.83750 time= 0.04244
Epoch: 0026 train_loss= 0.28593 train_acc= 0.88333 val_loss= 0.41172 val_acc= 0.82500 time= 0.03129
Epoch: 0027 train_loss= 0.27965 train_acc= 0.89120 val_loss= 0.41409 val_acc= 0.82500 time= 0.04688
Early stopping...
Optimization Finished!
```

However, there are still clear benefits for leveraging this graph-based neural network given the efficiency of the algorithm, multiple data visualizations, the successful mapping of documents in a multilayer network, and the creation of both word and document embeddings with consistent preprocessing. These embeddings were generated in a fashion that outperforms current sklearn implementations given the ability to tune for stop word removal and use punctuation as an amplifier for sentiment, a goal previously proposed in the design section of this document.

The original 2019 paper mentions a “robustness of Text GCN to less training data in text classification” which clearly was not observed in this project. However, further inspection of the version of the network that the team publicly released shows a need to update a few of the solvers for the latest version of TensorFlow and a few bugs in the preprocessing code. These errors may be at fault for such unexpected results in comparison to the results provided in the original paper.

Results

The results of the experiment indicate that the Ockham’s razor approach to text classification, such as our regularized Logistic Regression model, is still a prize-winning solution to handling a very important problem in the space of information retrieval. The logistic regression model and the MLP models were the best performers for the test data, while the Text GCN model showed the best performance during training. Below, we see an output of key performance differences between the two higher performing models:

Table 2: Comparison of top performing text classifiers (Logistic versus Multilayer Perceptron)

	Logistic	Multilayer Perceptron
mean_fit_time	0.10184	2.700843
std_fit_time	0.026455	0.436425
mean_train_score	0.981667	1.0
std_train_score	0.000918	0.0

Despite a higher mean training score and the 0.0 standard deviation train score, the MLP model may also be prone to overfitting. The regularization of the logistic regression model, when fit to the entire training dataset and not the double-split subset of data, shows much better consistency and flexibility, likely due to the previously discussed advantages of its solver.

Discussion

The results clearly show that there is a strong practicality for building linear models with non-linear solvers, especially when taking into account regularization and careful attention to preprocessing. This effect showed to be especially pertinent when dealing with a small amount of corpus data that could benefit from being represented as a more machine-friendly representation (e.g. bag of words, bag of n-grams, tf-idf, OneHotEncoding, etc.). Additionally, the results from this project indicate the usefulness of deep learning models for uncovering complex relationships in data. However, such complex methods take a tremendous amount of time to tune and interpret, taking a hit for practicality. Future problems may also want to look into challenging the effectiveness of Text GCNs for datasets with limited annotation or chaining together multiple GCNs for sentiment analysis.

Conclusion

Given the three model types created (Logistic Regression, Multilayer Perceptron, Text Graph Convolutional Network), Logistic Regression with heavy regularization showed the best performance and practicality for binary classification of sentiment (0 negative, 1 positive) for a dataset with limited annotation. A well-driven preprocessing pipeline is also critically important for text classification.

Acknowledgements

Many thanks to Professor Marty Allen, all of the course staff, and the students that made this semester such an exciting learning experience. Looking forward to more experimenting and imagineering in the future.

References

- Chen, J., Huang, H., Tian, S., & Qu, Y. (2009). Feature selection for text classification with Naïve Bayes. *Expert Systems with Applications*, *36*(3), 5432-5435.
- Gan, J., & Tao, Y. (2015, May). DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (pp. 519-530).
- Kruczek J., Kruczek P., Kuta M. (2020) Are n-gram Categories Helpful in Text Classification?. In: Krzhizhanovskaya V. et al. (eds) Computational Science – ICCS 2020. ICCS 2020. Lecture Notes in Computer Science, vol 12138. Springer, Cham. https://doi.org/10.1007/978-3-030-50417-5_39
- Violos, J., Tserpes, K., Varlamis, I., & Varvarigou, T. (2018). Text Classification using the N-gram graph Representation model over high frequency data streams. *Frontiers in Applied Mathematics and Statistics*, *4*, 41.
- Yao, L., Mao, C., & Luo, Y. (2019, July). Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 33, No. 01, pp. 7370-737