Exploration of the BERT Model for Pun Detection and Classification

Abstract

The project constructs a Bert model-based pun detection and classification study on the SemEVAL-2017 Task 7 dataset. The model results were compared with those of SVM and Naive Bayes models, which obtained 0.7839, 0.8630 and 0.8337 accuracy respectively. The results of the model are presented visually and analyzed briefly. The advantages and disadvantages of different models are studied and compared, and the low accuracy of Bert model may be due to unbalanced data sets. Finally, the improvement direction and prospect of the model in the future are proposed.

Introduction

A pun is a form of wordplay in which one signifier (e.g., a word or phrase) suggests two or more meanings by exploiting polysemy, or phonological similarity to another signifier, for an intended humorous or rhetorical effect. For example, the first of the following two punning jokes exploits contrasting meanings of the word "interest", while the second exploits the sound similarity between the surface form "propane" and the latent target "profane":

- I used to be a banker but I lost interest.
- When the church bought gas for their annual barbecue, proceeds went from the sacred to the propane.

Conscious or tacit linguistic knowledge – particularly of lexical semantics and phonology – is an essential prerequisite for the production and interpretation of puns. This has long made them an attractive subject of study in theoretical linguistics, and has led to a small but growing body of research into puns in computational linguistics.

For this project, we use a sub-dataset of SemEval-2017 , each context contains an ID, a maximum of one pun, and a feature of puns (0/1) which classifies whether this is a pun or not. We will be fine-tuning the BERT for classifying puns using this generated dataset.Before this, we converted the dataset from xml format to tsv format for subsequent processing.

Related Work

Miller and Turkovic[16] make a case for research into computational methods for detection of puns in a running context. Current approaches to pun interpretation rely on Lesk [17], which considers the highest overlap between the context and gloss in order to return the target sense (two senses) for the pun. Miller and Gurevych [18] performs automatic pun disambiguation in a comprehensive experimental setup.Pun detection is the task of detecting whether there is a pun residing in the given text. The goal of pun location is to find the exact word appearing in the text that implies more than one meanings.Most previous work addresses such two tasks separately and develops separate systems[19]. Compared to the pipeline methods, joint learning has been shown effective [20] since it is able to reduce error propagation and allows information exchange between tasks which is potentially beneficial to all the tasks.

Dataset and Features

The pun detection task at SemEval-2017 Task 7[9] used two manually annotated data sets, we used one of the two and converted it from xml to tsv format as our dataset. The data content has the following characteristics:

- Each item contains an ID, a sentence with at most one pun, and a feature of puns (0/1) which classifies whether this sentence contains a pun or not.
- Each pun (and its latent target) contains exactly one content word (i.e., a noun, verb, adjective, or adverb) and zero or more non-content words (e.g., prepositions or articles). Puns and targets may be multi-word expressions containing only one content word—this includes phrasal verbs such as “take off” or “put up with”.

The data set contains 1780 contexts, of which 1271 (71%) contain a pun. Of the puns, 1098 (86%) have both meanings in WordNet. All puns contain a total of 19461 words, with an average of 10.9 words. For each pun, there are at least 2 words and a maximum of 69 words.

The data set will be randomly divided into training set and test set according to the proportion of pun features (that is, the proportion of training set and test set containing a pun is close to 71% of the total samples), in which the training set accounts for 70% of the total samples.

Methods

For pun detection, we need to identify whether a sentence contains a pun. Formally, the task is modeled as a binary classification problem. Our main method builds on BERT [10], which has obtained state-of-the-art performance on most NLP tasks [11]. More specifically, we need to build a Bert model and fit the model with our training set. Our method first obtains a Bert preprocessor, which is used to convert data into a format that BERT can understand and preprocess our data so that it matches the data BERT was trained on. Specifically, it mainly accomplishes:

- Lowercase our text
- Tokenize it (i.e., "sally says hi" -> ["sally", "says", "hi"])
- Break words into WordPieces[12] (i.e. "calling" -> ["call", "##ing"])
- Map our words to indexes using a vocab file that BERT provides
- Add special "CLS" and "SEP" tokens [13]
- Append "index" and "segment" tokens to each input [11]

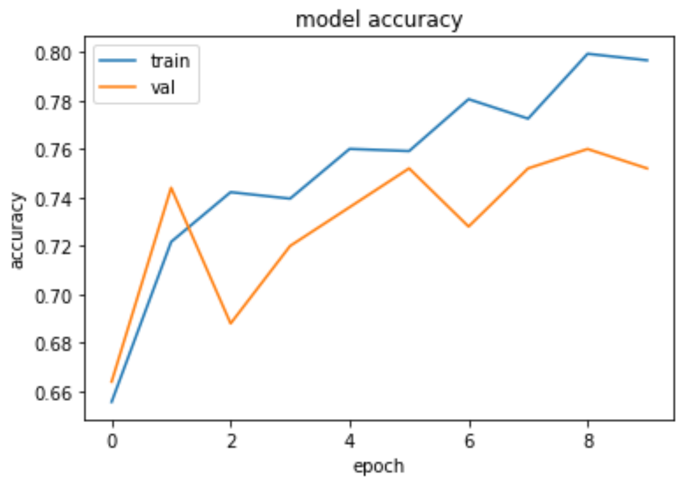
Then, we use pooled_output to pool the output as the last layer hidden-state of the first token of the sequence (classification token) further processed by a Linear layer and a Tanh activation function [14]. Finally, we build the dropout_layer to prevent overfitting and use the activation function Sigmoid [15] to provide the network with nonlinear modeling capabilities.

Experiments

To analyze the functionality of our fine-tuned model of BERT, we also created a Support Vector Machine (SVM) and Naive Bayes Classifier. These three models are combined to have an additional preprocessing layer, for which, we use a CountVectorizer layer [3] from Scikit-Learn for the Machine Learning models (SVM and Naive Bayes) and the corresponding preprocessing layer for our BERT model from TFHub [1].

The pre-saved BERT model that we are using has L=12 hidden layers, which are transformer blocks, a hidden size of H=768, and A=12 attention heads [1]. We picked this version of BERT pre-trained with weights by the original BERT authors [2], to have a model with a high number of layers, while also taking into consideration the training time of the model itself. We used the corresponding layer that performs text preprocessing through the TensorFlow-Text library as well for transfer learning and fine-tuning.

For training our BERT model, we used a batch size of 32, after testing sizes of 32, 64, 128 because we saw that with size = 32, we achieve convergence with a small number of epochs. This was important to us, since fitting BERT to our training data took long periods of time. With our batch size of 32, we plotted validation and test accuracy to see when our algorithm converges, which we used to set our number of epochs at 10. The train vs validation accuracy graph for our fine-tuned BERT is shown below:



For our SVM and Naive Bayes classifiers, we wanted to train the best possible version of these models for a more accurate comparison with BERT. These models consisted of a Scikit-Learn Pipeline made up of a CountVectorizer module and a classifier module (either SVM or Naive Bayes). As a result, we had a number of hyperparameters available for selection for these two models. For this, we decided to use Scikit-Learn's GridSearchCV module [4]. With a five-fold cross validation, the GridSearchCV model exhaustively searches over the given parameters to find the best possible selection of parameters for the input model [4]. The parameters used for Grid Search can be seen in Appendix A.

After performing GridSearch, we found the following best hyperparameters for the two models. We used these to train our final models for the two types of ML.

Algorithm	**Best Hyperparameters**
Support Vector Machine	`{'svm__C': 1, 'svm__kernel': 'linear', 'vectorizer__ngram_range': (1, 2)}`

Multinomial Naive Bayes	`{'naive_bayes__alpha': 1, 'naive_bayes__fit_prior': False, 'vectorizer__ngram_range': (1, 2)}`

To compare our models, the metrics that we used were Precision Score, Recall Score and F1 score - for which we used the Scikit-Learn libraries to perform our calculations. The formulae for these are as follows:

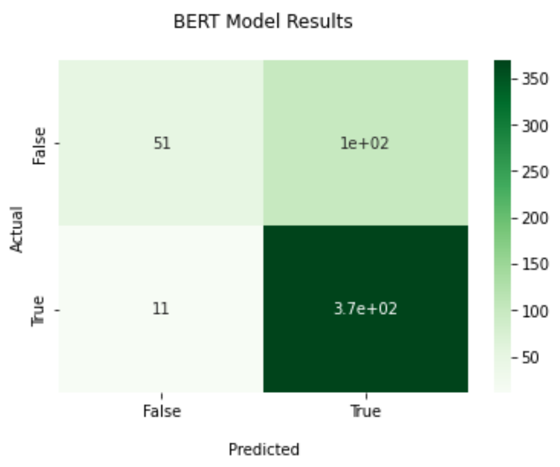
1. Precision Score = tp / (tp + fp) [5]
2. Recall Score = tp / (tp + fn) [6]
3. F1 Score = 2 * (precision * recall) / (precision + recall) [7]

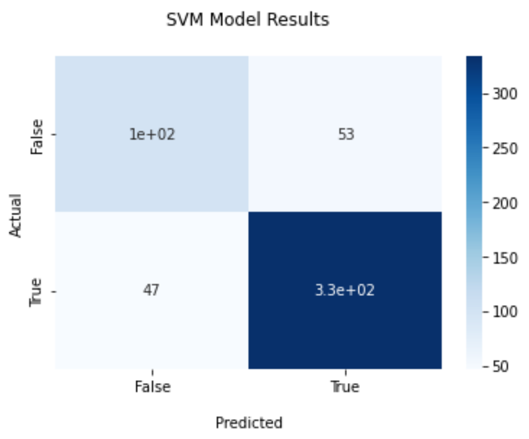
Results & Discussion

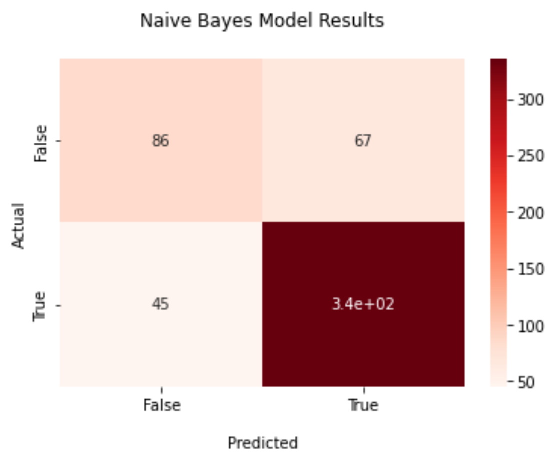
From our conducted experiment, we found some interesting, unexpected results. Here are the calculated metrics for each of the models:

Model	**Precision**	**Recall**	**F1**
BERT	0.7839	0.9711	0.8675
SVM	0.8630	0.8766	0.8698
Naive Bayes	0.8337	0.8819	0.8571

This can be seen also in the plotted confusion matrices:







As you can see, while the BERT model outperforms the other two in Recall significantly, our SVM outperforms the state-of-the-art text classification model for sentiment analysis and humor detection in F1 score and Precision. There could be a number of reasons for SVMs outperforming BERT for pun detection; firstly, the syntax of puns and the words used might be a niche text classification task that BERT does not generalize well to. Another text classification task where SVM beats BERT is legal text classification as seen in the LexGLUE benchmark [8], where SVM models achieve a 1.8% improvement in Terms of Service paperwork. However, this could also be an issue arising with problems in experimental setup. For example, the training dataset is quite unbalanced and has few data points (~1800). Moreover, the preprocessing used across the models is different, which may have led to some issues with BERT.

Conclusion

Through the course of this project, we were able to explore the use of the BERT transformer model towards the purpose of pun detection and classification. Although BERT is the state-of-the-art in many Natural Language Processing (NLP) tasks, including sentiment analysis and humor detection, we see that in this case - the more traditional ML algorithms of SVM and Naive Bayes perform as well and sometimes even better than BERT. There is no clear explanation for this, although it is possible that this task is "too simple or their language not being sufficiently domain-specific to take advantage of the models' pretraining" [8]. In the future, we would explore the cause of the underperformance of BERT and perhaps either improve the preprocessing or use a larger dataset, which could lead to an increased performance in BERT.

Appendices

Appendix A: Parameters used for Grid Search

CountVectorizer (common for both models)

Parameter	Values
NGram Range	`(1, 1), (1, 2), (2, 2)`

Support Vector Machine

Parameter	Values
Kernel	`'linear', 'rbf'`
C (regularization parameter)	`0.1, 1, 10, 100`

Multinomial Naive Bayes

Parameter	Values
Alpha	`0.001, 0.01, 0.1, 1`
Fit_Prior	`True, False`

Bibliography

1. *Tensorflow hub*. [Online]. Available: https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4. [Accessed: 09-May-2022].
2. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional Transformers for language understanding,” *arXiv.org*, 24-May-2019. [Online]. Available: https://arxiv.org/abs/1810.04805. [Accessed: 09-May-2022].
3. “Sklearn: CountVectorizer,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed: 09-May-2022].
4. “Sklearn.model_selection.GRIDSEARCHCV,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed: 09-May-2022].
5. “Sklearn.metrics.precision_score,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score. [Accessed: 09-May-2022].
6. “Sklearn.metrics.recall_score,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score. [Accessed: 09-May-2022].
7. “Sklearn.metrics.f1_score,” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score. [Accessed: 09-May-2022].
8. B. Clavié and M. Alphonsus, “The unreasonable effectiveness of the baseline: Discussing svms in legal text classification,” *arXiv.org*, 22-Oct-2021. [Online]. Available: https://arxiv.org/abs/2109.07234. [Accessed: 09-May-2022].
9. SemEval-2017 Task 7.[Online]. Available: https://alt.qcri.org/semeval2017/task7/. [Accessed: 09-May-2022].
10. Wikipedia contributors. (2022, April 10). BERT (language model). In Wikipedia, The Free Encyclopedia. Retrieved 02:06, May 10, 2022, from https://en.wikipedia.org/w/index.php?title=BERT_(language_model)&oldid=1081939013
11. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
12. WordPiece.[Online]. Available:

https://paperswithcode.com/method/wordpiece.[Accessed: 09-May-2022].
13. google-research/bert.[Online]. Available: https://github.com/google-research/bert.[Accessed: 09-May-2022].
14. BERT.[Online].Available: https://huggingface.co/transformers/v3.0.2/model_doc/bert.html#bertmodel.[Accessed: 09-May-2022].
15. Wikipedia contributors. (2022, April 25). Sigmoid function. In *Wikipedia, The Free Encyclopedia*. Retrieved 02:26, May 10, 2022, from https://en.wikipedia.org/w/index.php?title=Sigmoid_function&oldid=1084599083
16. Towards the automatic detection and identification of English puns. European Journal of Humour Research4(1):59–75.
17. Automatic sense disambiguation using machine readable dictionaries: How totell a pine cone from an ice cream cone. In Proceedings of the 5th Annual International Conference on Systems Documentation. ACM, New York, NY, USA, SIGDOC '86, pages 24–26. https://doi.org10.1145/318723.318728.
18. Wordnet: A lexical database for english. Commun. ACM 38(11):39–41.https://doi.org/10.1145/219717.219748.
19. Nhance at semeval-2017 task 7: A computational approach using word association for puns. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017).
20. Investigating lstms for joint extraction of opinion entities and relations. In Proceedings of ACL.