Machine Learning Methods for Document Classification on Humor Detection

1st Jichen Li
Dept. of Electrical Engineering
Stevens Institute of Technology
Hoboken, USA
jli218@stevens.edu

2nd Yi Rong
Dept. of Electrical Engineering
Stevens Institute of Technology
Hoboken, USA
yrong4@stevens.edu

3rd Hao Zhao
Dept. of Applied Mathematics
Stevens Institute of Technology
Hoboken, USA
hzhao33@stevens.edu

***Abstract*—In this project, we will use BERT, Naive Bayes(NB), Support Vector Machine(SVM) and XGBoost to describe a novel approach for detecting humor in short texts. BERT, Bidirectional Encoder Representations from Transformers, which is a newly proposed language representation model, BERT has been proved to be very powerful that obtains the state-of-the-art results on almost all natural language processing tasks. Naive Bayes, a classification technique based on Bayes' Theorem with an assumption of independence among predictors. SVM, a supervised learning model with associated learning algorithms that can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. While XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework.**

***Index Terms*—BERT, Naive Bayes, Support Vector Machine, XGBoost, humor detection**

I. Introduction

One of the latest directions of artificial intelligence is related to humor, in recent years, Comedy based computing such as the joke writing computer, STANDUP - System to Augment Non-Speakers'Dialogue Using Puns; SASI, a sarcasm-detecting program; and DEviaNT, the Double Entendre via Noun Transfer program. As these progress humor is injected into the computer-generated sentence reponses, and humor detection becomes an indispensable and important step.

This paper aims at utilizing BERT, Naive Bayes, Support Vector Machine and XGBoost for humor detection. One approach builds on using BERT sentence embedding in a neural network, where, given a text, our method first obtains its token representation from the BERT tokenizer, then, by feeding tokens into the BERT model, it will gain BERT sentence embedding. Finally, it will pass sentence embedding as input to a two-layered neural network to predict the target value.

For Naive Bayes, Support Vector Machine and XGBoost, we use these alternative ways in text classification. Naive Bayes is a series of probabilistic algorithms, which uses probability theory and Bayes' Theorem to predict text labels. They are probabilistic, which means they calculate the probability of each tag for a given text and then output the tag with the highest one. They get these probabilities by using Bayesian theorem, which describes the probability of a feature based on a prior knowledge of the conditions that may be associated with the feature.

Support Vector Machine(SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text

XGBoost is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Another approach PRADO, a novel projection attention neural network that combines trainable projections with attention and convolutions. Which shows the effectiveness of the trainable projection model in finding semantically similar phrases and reaching high performance while maintaining compact size. It combines trainable projections with attention and convolutions allowing it to capture long range dependencies and making it a powerful and flexible approach for long text classification.

II. Related Work

For decades, learning widely applicable word representation has been an active research field, including non neural methods and neural methods. Pre-training word embedding is an indispensable part of modern NLP system. Compared with the embedding learning from scratch, it has a great improvement.

With the progress NLP, researchers applied the more advanced methods to evaluate the task, Humor detection. This includes using statistical and N-gram analysis [6], Regression Trees [7], Word2Vec combined with K-NN Human Centric Features [8], and Convolutional Neural Networks [9].

With the popularity of transfer learning, a lot of research has focused on using pre-trained models for different tasks of NLP. Transfer learning in NLP, especially models like ULMFiT [11], Allen AI's ELMO [12], and Google's BERT [2], which focus on storing knowledge gained from training one problem and apply it to another related problem (usually after fine-tuning on a small amount of data).

The first transfer learning method in Natural Language Processing (NLP) was Universal Language Model Fine-tuning for Text Classification (ULMFiT) [11]. They trained a language model on a preliminary dataset, such as Wikitext,and then fine tuned it on a small training set to complete the new task. In addition to significantly outperforming many state-of-the-art tasks, their sample models are also accomplished by training only 100 tagged samples, which perform as well as the old models trained with large amounts of data.

ELMo [12] includes task-specific architectures and uses the pre-trained representations as add-on features [12]. It is a deep expression of cultural words, and can model complex features of word use (e.g., syntax and semantics). They added these representations to the existing models and showed improvements to six popular NLP tasks.

BERT [2] uses a multi-layer bidirectional transformer encoder, which consists of several encoders stacked together and can learn deep bi-directional representations. Similar to previous transfer learning methods, it pre-trains on unlabeled data to fine tune various tasks (such as "question answering") later. Initially with two model sizes ($BERT_{BASE}$ and $BERT_{LARGE}$) and 11 latest results were obtained. Since then, it has been pre-trained and fine-tuned for several tasks and languages, and has introduced several Bert-based architectures and model sizes (for example, multilingual Bert, Roberta [13], Albert [14], and videobert [15]).

The focus of Reference [9] is to detect humor in jokes by using transformer architecture. They solved this problem by building a model that could identify humorous jokes based on ratings from popular Reddit r / Jokes threads (13884 negative and 2025 positive). They show that their approach outperforms all previous work on selected tasks, with an F-measure of 93.1% for the puns dataset and 98.6% for the short joke dataset.

There are emerging tasks related to humor detection. Reference [16] focuses on predicting humor through the use of audio information, so only using audio data can achieve 0.750 AUC. A large number of studies have focused on humor detection in non-English texts, such as Spanish [17] [18], Chinese and English-Hindi [19].

III. Our Solutions

We have succeed in running Naive Bayes, SVM, XGBoost abd BERT for humor detection task. And we still try to run PRADO model in the future as a new direction.

A. Dataset

We are using the Annamoradnejad dataset of 200k short texts for humor detection, [1] which can be found on Kaggle. The dataset contains 200k formal short texts labeled as humorous and not humorous. The short texts were scrawled from Reddit communities. The dataset is compiled as a single CSV file with no additional information about each text (such as the source, date, etc).

The texts are deliberately chosen to have compatible statistics (text length, word counts, etc.), making it less likely to detect humor based on simple analytic models without understanding the underlying latent connections.

Humor Detection	
How do you confuse hellen keller? give her a basketball and tell her to read it.	True
Are bradley cooper and suki water house back together?	False
I heard they wanted to do an asian version of drive but there's already a movie called crash.	True
7 ways to make your home happier and more relaxing in the new year	False

Fig. 1. Examples for humor detection

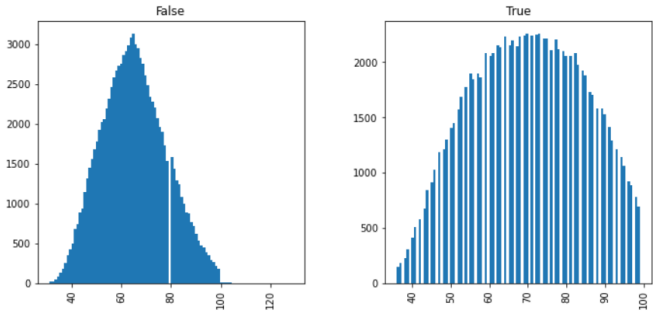

Fig. 2. Character Length of the texts

The dataset contains 100k humor text and 100k non-humor text. The character length for each text is shown in figure 2. For both humor and non-humor text, the character lengths of texts are approximately normally distributed with the mean length centered around 70 characters. Table 1 below shows the top 15 most common words and their counts for both humor and non-humor texts. We filtered out common stop words (i.e. "the", "and", etc.) to make the result more meaningful.

TABLE I
Most Common Words

Humor		Non Humor	
Word	Count	Word	Count
call	8174	trump	5951
like	7530	photos	5147
whats	5563	new	4918
get	5503	video	3321
im	4984	donald	2910
dont	4602	us	2801
say	4563	says	2424
one	4047	day	2315
people	3858	make	1919
know	3142	one	1807
cant	2752	best	1810
man	2547	get	1757
joke	2537	people	1637
got	2514	5	1616
make	2463	women	1613

To run our models, the data is split into 80%(160k) training and 20%(40k) testing rows, where each class has been separated roughly in a 50-50 split.

B. Pre-Processing

For some of our models, we can't directly input the raw text data into the models. We need to first transform the raw data into an understandable format for NLP models. Data preprocessing is a proven method of resolving such issues. This will help us get better results through the classification algorithms. For our case, we used three different pre-processing techniques.

First, we performed tokenization. Tokenization is the process of breaking our text up into words, symbols, or other meaningful elements as tokens. The list of tokens replaces the raw input for further processing. In our specific case, we used the NLTK Library's *word_tokenize* and *sent_tokenize* methods to break the short text into a list of words respectively.

Our second process is word stemming/lemmatization. This process reduces the inflectional forms of each word into a root. Lemmatization is very similar to stemming. A stemmer operates on a single word without knowing the context, and therefore cannot differentiate words that have different meanings depending on the part of speech. However, stemmers are easier to implement and run faster. For our data, we used the NLTK Library's WordNetLemmatizer.

The final process we used is word vectorization. Word vectorization is the process of turning text data into numerical vectors. There is an abundance of different methods for word vectorization. Here, we are using the most popular one among them Term Frequency-Inverse Document (TF-IDF). Without going into math details, TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection

Besides the three main processes, we also performed some other minor processes to our data, including changing all text to lowercase, eliminating stopwords, etc.

C. Naive Bayes

Sometimes the simplest solutions are usually the most powerful ones, and Naive Bayes is a good example of that. Naive Bayes is a simple classification method based on Bayes rule that works particularly well in NLP problems.

$$P(\mathbf{A}|\mathbf{B}) = \frac{P(\mathbf{B}|\mathbf{A})P(\mathbf{A})}{P(\mathbf{B})}, \tag{1}$$

Naive Bayes relies on very simple representation of document, bag of words representation. We assume that every word in a sentence is independent of the other ones. This means that we're no longer looking at entire sentences, but rather at individual words. This also means that the position of each word does not matter. "How do you confuse Hellen Keller" is the same as "Keller Hellen confuse you do how".

$$P(\text{How do you}) = P(\text{How}) \times P(\text{do}) \times P(\text{you}), \tag{2}$$

This assumption is very strong but super useful. It's what makes this model work well with little data or data that may be mislabeled. The next step is just applying this to the Bayes theorem.

$$P(\text{How do you}|\text{humor}) = P(\text{How}|\text{humor}) \times P(\text{do}|\text{humor}) \times P(\text{you}|\text{humor}) \tag{3}$$

The final step is just to calculate the probability of humor and non humor and determine the outcome based on which one turns out to be larger. In this step, we also performed Laplace smoothing for words that has not been seen in the training data. Using the sklearn Multinomial Naive Bayes model, we were able to achieve a final accuracy of 87.3%.

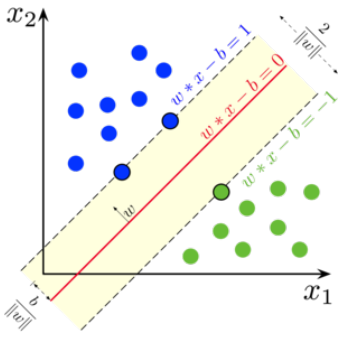

Fig. 3. SVM

D. SVM

Support Vector Machines (SVM) is a supervised machine learning algorithm, which has been used successfully in many Natural Language Processing (NLP) tasks. Many NLP problems will transform the problem into a multi-class classification task; then convert the multiclass problem into several binary classification problems; SVM works extremely well in these problems. Given training data, SVM learns a classification hyperplane in the feature space which has the maximal distance (or margin) to all the training examples. As a result, SVM tends to have better generalization on unseen data than other distance-based learning algorithms such as KNN. Another advantage of the SVM is that, by using different types of kernel functions, the SVM can explore different kinds of combinations of the given features without increasing computational complexity, making SVM the optimal classifier.

Furthermore, usually, NLP problems have high dimensional but sparse feature vectors. Namely, positive and negative data are distributed into two distinctly different areas of the feature space. This is extremely helpful for SVM to search a hyperplane in the feature space and the main reason why SVM can achieve very good results in NLP problems. Such high dimensional representation is achieved by exploring the kernel functions to map the input features into higher dimensions.

In our project, we used the SVM model from sklearn and tried different parameters. Comparing the final results of the different SVM models, we found the most important parameter for the SVM is the kernel function. For our specific data, a linear kernel proves to be the best with a final accuracy of 88.3%.

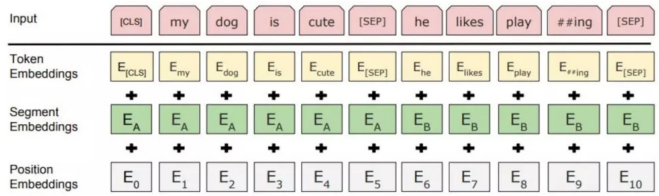

Fig. 4. Transform inputs into token embeddings, segmentation embeddings and position embeddings

E. XGBoost

XGBoost is an algorithm that has dominated applied machine learning and Kaggle competitions for classification problems. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. XGBoost stands for eXtreme Gradient Boosting, was developed by Tianqi Chen with the goal of "to push the limit of computations resources for boosted tree algorithms".

XGBoost is famous for its executional speed and model performance. Generally, XGBoost is fast. Especially when compared to other implementations of gradient boosting. It utilizes system features such as parallelization, distributed computing, cache optimization, and etc. to maximize the computing power of the machine. XGBoost also yields good model performance. It dominates structured or tabular datasets on classification problems. It is the go-to algorithms for Kaggle competitions.

Though XGBoost isn't specially designed for NLP problems. We still believed it could be a potential solution to our problem. We used the xgboost library and after tuning the different parameters, we were able to achieve an accuracy of 88.1%.

F. BERT

BERT is a Transformer-based model for NLP tasks which is created in 2018 by Jacob Devlin and his colleagues from Google. It shows amazing results in the top level test of machine reading comprehension SQuAD1.1 and achieved state-of-the-art performance on many NLP tasks [2]. Today it's still the best model for many NLP tasks.

Humor, usually defined as the quality which appeals to a sense of the ludicrous or absurdly incongruous, is caused by logical conflict in language. So the relationship and logic between every words can reflect the essence of humor. The best way to describe humours text in computer language is distributed representation.

One hot encoding is used to encode short text with little amount of vocabulary. However, for large range of words, researchers use word embedding which transform words into vector. The benefits of word embedding is decreasing the complication of vector space's dimension to avoid the 'Curse of dimension' and also keep the relationship between every words.

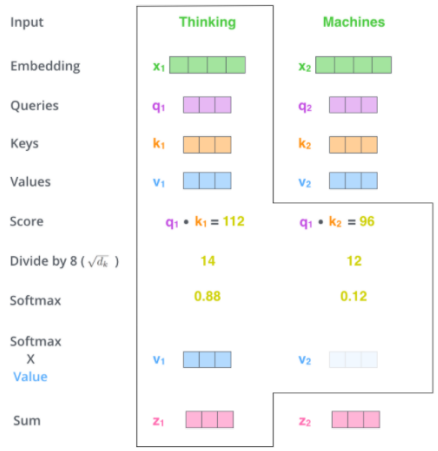

Fig. 5. The detailed calculation process of Attention encoder

BERT has two steps in framework: pre-training and fine-tuning. During pre-training, the model encode every word into three embeddings. The model first uses tokenizer to transform input's words into token embedding, which is just assign a matrix to each words. Then it encodes a word into segmentation embedding, which describe whether these words belong to the same sentence.At last the BERT encodes every word into position embedding, which keep the information of the location of every word in the sentence. After that, some random token embeddings are replaced by 'Mask'. The algorithm predicting the words covered by Mask is called 'Masked LM'(MLM). Unlike left-to-right or right-to-left direction, BERT predicts Mask begin from both left and right at the same time. The key step is called self-Attention.

Attention is a function module in BERT. The Attention module include 6 encoder and 6 decoder steps. For every encoder step, the module first encode the inputs into embedding. Then set 3 matrix called Query, Key and Values. The embedding multiplies 3 matrix individually and get 3 result called Q, K and V. Using the following formula to deal with Q, K and V:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

This is the process of one encoder. In this step, Query means the information that you ask. K is the information that relate to Query. Q·K is the similarity between each other.

After pre-training step, the vocabulary has been transformed into vectors and formed a space. the distance between each vector represent the relationship of each word.

As for fine-tuning, BERT model first used pre-trained parameters, and all parameters using from downstream tasks. Each downstream task has a separate fine-tuned models, even if they have been initialized with the same pre-trained parameters.

Our method first obtains the text token representation from the BERT tokenizer, using a maximum sequence length of 100

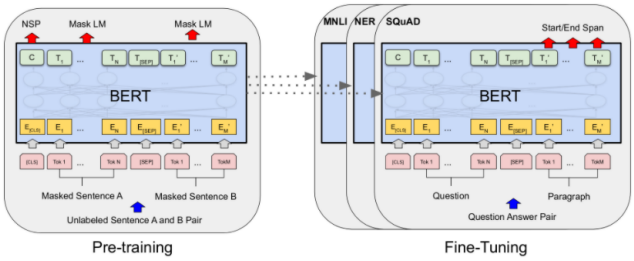

Fig. 6. Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.

(the maximum sequence length of BERT is 512). Then, by feeding tokens as input to the BERT model, it will generate BERT sentence embeddings (768 hidden units). The model embeds sentences as the input of a two-layer neural network (dropout=0.2, Activation = sigmoid) to predict a single target value. Finally, we applied target prediction that produces a binary result.

BERT provides two pre-trained general types ($BERT_{BASE}$ and $BERT_{LARGE}$), they use separate data source and training scale. In our proposed method, we use small ($BERT_{BASE}$) has the following characteristics: 12 layers, 768 hidden, 12 heads, 110 meters parameters, according to lowercase English text for training.

The BERT model performed well in humor detection, we clean up the dataset and choose randomly one tenth data which contains 20k texts for preliminary experiment. The data is randomly divided into 80% of training data and 20% of testing data. And the pre-train model we choose is small BERT-size which contain 12 layers, 768 hidden units and 12 the Learning rate we set is 2e-5. For final result, the accuracy is 97.9%. The result shows that BERT model can detect humor in text precisely.

IV. Comparison of our results

After finishing four algorithm, we get the finally result.

As shown in the Table III, BERT achieves the best accuracy among the four algorithms, which is 0.98. The other three are almost the same, all close to 0.88.

TABLE II
RESULTS OF BERT

	values
auc	0.9799
eval accuracy	0.9800
f1 score	0.9803
false negatives	37.0000
false positive	44.0000
loss	0.0933
precision	0.9786
recall	0.981942
true negatives	1960.0000
true positives	2012.0000
global step	2026.0000

The advantage of BERT is that it reach the state-of-the-art performance on this task. However, the training time is also the longest. It cost nearly 8 hours to run 10000 steps. On the other hand, although Naive Bayes yields the worst results, its training and classification time are extremely fast. It only takes a few minutes to train a Naive Bayes model, and seconds to find out the result. SVM and XGBoost also requires significantly less time than the BERT algorithm. Therefore, if we are only looking for accuracy disregarding the computational power limit, we should choose the BERT method. However, if we only have limited computational power, Naive Bayes may be the better choice.

V. Future research directions

In the BERT model, at the fine-tuning step, the feature extraction step and the training of classifier all together determine the final accuracy of the solution. One main factor that may affect the performance is the design of the classifier. As the embedding of each word extracted by BERT having a size of 768, we really need a bigger network to perform the classification. Another fact that may cause the failure of the BERT model is the variance of the length of the statement-answer pairs. In the training dataset, the longest one has 100 words, while some others may have only 0 words. Thus, some samples are padded with too many zeros. This approach may be effective on dataset with more training samples. We leave it in our future work. While another approach is PRADO. In 2019, Google AI team published a neural architecture called PRADO, which at the time achieved state-of-the-art performance on many text classification problems, using a model with less than 200K parameters. While most models use a fixed number of parameters per token, the PRADO model used a network structure that required extremely few parameters to learn the most relevant or useful tokens for the task. And because of this characteristic, PRADO cost less time to train NLP model and can handle complex tasks without large GPU. However, the accuracy of PRADO is lower than BERT. The advantage of PRADO is very obvious that many researchers are trying to improve it.

Figure 5 shows the overall architecture of network PRADO . It consists of a projected embedding layer, a convolutional and attention encoder mechanism and a final classification layer [4].

Most NLP tasks requiring large RAM or GPU for training. However, in many cases, a lightweight engine has more advantage. PRADO is such a lightweight model. The base idea of PRADO is compress the vector space further, since some words like 'a', 'the' do not express meaningful information.

TABLE III
COMPARISON OF OUR RESULTS

	NB	SVM	XGBoost	BERT
Accuracy	0.873	0.883	0.881	**0.980**
F1	0.874	0.883	0.881	**0.980**
Precision	0.882	0.886	0.882	**0.979**
Recall	0.866	0.880	0.880	**0.982**

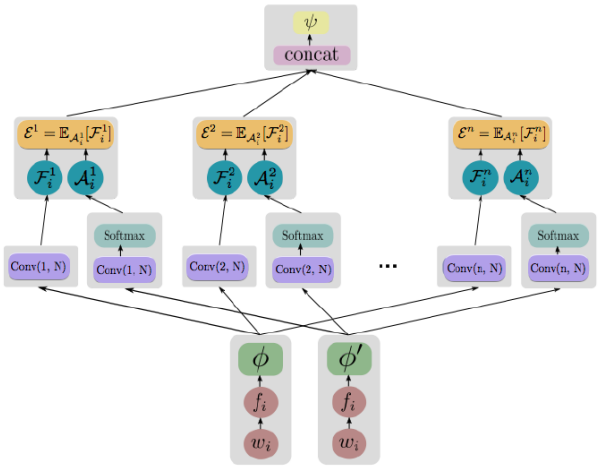

Fig. 7. PRADO Model Architecture

The PRADO focus more on important words and ignore others and it captures contextual information for long-text classification while maintaining a very low memory footprint.

We still working on improve PRADO algorithm. Google AI team used QRNN(Quasi-recurrent neural networks) combine PRADO and got comparable results with BERT. We decide to dive further into this direction later.

VI. Conclusion

In this project, we have proposed several approaches for document classification on humor detection.

Because of the difficulty of NLP tasks, training NLP models takes a very long time and most of the algorithms require GPU to accelerate the training progress to ensure the accuracy of the task. BERT, for example, reaching the highest accuracy while taking the longest training time. The Naive Bayes, SVM and XGBoost are not as complicated as BERT, but they cannot be compared to the performance of BERT. So the most critical question is can we find an algorithm to absorb the advantages of both, while overcoming the disadvantages of both. PRADO may give us a hope. However, PRADO, as a new type of lightweight model, still has some difficulties to overcome.

References

[1] Annamoradnejad, I. (2020). ColBERT: Using BERT Sentence Embedding for Humor Detection. arXiv preprint arXiv:2004.12765.

[2] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT.

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. NIPS.

[4] Prabhu Kaliamoorthi, Sujith Ravi, & Zornitsa Kozareva.(2019). PRADO: Projection Attention Networks for Document Classification On-Device.

[5] Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2017). Quasi-Recurrent Neural Networks. ArXiv, abs/1611.01576.

[6] Taylor, J. M., Mazlack, L. J. (2004). Computationally recognizing word-play in jokes. In Proceedings of the Annual Meeting of the Cognitive Science Society (Vol. 26, No. 26).

[7] Purandare, A., Litman, D. (2006, July). Humor: Prosody analysis and automatic recognition for f* r* i* e* n* d* s. In Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (pp. 208-215).

[8] Yang, D., Lavie, A., Dyer, C., Hovy, E. (2015, September). Humor recognition and humor anchor extraction. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 2367-2376).

[9] Weller, O., Seppi, K. (2019). Humor detection: A transformer gets the last laugh. arXiv preprint arXiv:1909.00252.

[10] 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.

[11] Howard, J., Ruder, S. (2018). Universal language model fine-tuning for text classification. arXiv preprint arXiv:1801.06146.

[12] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365.

[13] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[14] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.

[15] Sun, C., Myers, A., Vondrick, C., Murphy, K., Schmid, C. (2019). Videobert: A joint model for video and language representation learning. In Proceedings of the IEEE International Conference on Computer Vision (pp. 7464-7473).

[16] Chiruzzo, L., Castro, S., Etcheverry, M., Garat, D., Prada, J. J., Rosá, A. (2019). Overview of HAHA at IberLEF 2019: Humor Analysis based on Human Annotation. In IberLEF@ SEPLN (pp. 132-144).

[17] Ismailov, A. (2019). Humor Analysis Based on Human Annotation Challenge at IberLEF 2019: First-place Solution. In IberLEF@ SEPLN (pp. 160-164).

[18] Giudice, V. (2019). Aspie96 at HAHA (IberLEF 2019): Humor Detection in Spanish Tweets with Character-Level Convolutional RNN. In IberLEF@ SEPLN (pp. 165-171).

[19] Khandelwal, A., Swami, S., Akhtar, S. S., Shrivastava, M. (2018). Humor detection in english-hindi code-mixed social media content: Corpus and baseline system. arXiv preprint arXiv:1806.05513.