

Don't Eat the Onion:

Detecting “The Onion” vs non-Onion Headlines

Team: Liam Bush, Kevin Zhang, Jacob Feigenberg, Eduardo Gonzalez. **TA:** Kunli Zhang

1) Abstract

For this project, we study the problem of detecting satirical news headlines from "The Onion" compared to non-Onion headlines leveraging a dataset of 24,000 headlines sourced from The Onion and r/NotTheOnion. With the rise of misinformation and the potential consequences of mistaking satire for reality, this project is important since it can help people distinguish between the two. Our primary target contribution is the development of a high-accuracy machine learning model specifically tuned to detect satirical headlines from the Onion, which often resemble real headlines, as a new domain (Application). Second, we would like to discuss various advantages or pitfalls in different model families, like simple logistic regression, in detecting satirical headlines (Analysis). Through the project, we have developed effective 1D CNN and LSTM models both achieving ~87% test accuracy, and found most interestingly that the use of 1D convolutions were able to effectively outperform at this task from what we attribute to the capturing of fuller context within headlines data.

2) Introduction

Problem: “The Onion” (hereby simply, Onion) is a newspaper organization that publishes satirical news articles. As a famous satire publication, Onion headlines/articles have often mistakenly been taken seriously. One such instance was in May 2015, when former FIFA vice president Jack Warner mistakenly spread the Onion headline “FIFA Frantically Announces 2015 Summer World Cup In United States” as real news on Facebook. We also see more serious implications of this (discussed in “Motivation”). There is even a dedicated subreddit, r/AteTheOnion, with nearly 600K members focused on those who failed to see through The Onion’s satire. As avid readers of The Onion, we decided on the following task: *We aim to detect “The Onion” compared to non-Onion headlines. Our model will be able to output/classify, for a given headline, whether it is an Onion or non-Onion headline.*

The inputs to the system are headlines in text form, and the outputs will be a binary classification of whether the headline is Onion or non-Onion. We are using data from a dataset of 24,000 total Onion and non-Onion article headlines, with the non-Onion articles being “Onion-like” sourced from r/NotTheOnion (subreddit dedicated to real headlines that may appear satirical). The breakdown of the dataset is 15,000 non-Onion headlines, and 9,000 Onion headlines. The dataset is linked here:

<https://github.com/lukefeilberg/onion>. We will be splitting this into a 60/20/20 train/validation/test split, with the validation set used as required for any hyperparameter selection. Our aim is to build the best classification system possible, and foresee creating/tuning several iterations of model types/parameters to achieve this objective and contribute relative to previous work (see later sections).

To evaluate our proposed system, we will measure log loss between our predicted Onion/non-Onion label and actual labels for training, and for further evaluation/test, consider metrics such as classification accuracy, precision, recall, and/or F1 scores where appropriate.

Motivation: While the classification of articles as Onion vs non-Onion may seem frivolous at first glance, we believe the implications are actually deep. The very fact that entire subreddits have been dedicated to both the failure to identify Onion articles (r/AteTheOnion) or to real headlines that seem Onion-esque (r/NotTheOnion) highlights an important point: Information, specifically what is real/fake, exaggerated/accurate, and satire/serious, actually walks a fine line between each contrast. In an age where information spreads quicker than ever, systems that can discern between these contrasts will also become more important. As such, a high performing system in this project offers a step forward in the fight against the spread of misinformation, even if such misinformation is more light-hearted in nature. We

also believe that the flip side to this is to ensure that the idea of curbing misinformation does not inadvertently become a euphemism for censorship. As students, we commit to ethical research practices, prioritizing that our system will have limited risk for any chance of user harm, of which we see limited chances of within our specific aim of a classification problem.

3) Background

A few works inspired our contribution to the issue of headline satire detection and, more broadly, to the issue of misinformation:

Automatic Satire Detection: Are You Having a Laugh? <https://aclanthology.org/P09-2041.pdf>. This work by Clint Burfoot and Timothy Baldwin of the University of Melbourne used SVMs and specially constructed features to delineate satirical news articles from real news, giving us a reference point to which we can compare complicated learned features (deep learning) and a simpler bag-of-words approach. The authors also emphasized the importance of headlines as a feature in classifying satirical news, supporting our approach of focusing on Onion headlines.

Metaheuristics Based Long Short Term Memory Optimization for Sentiment Analysis <https://www.sciencedirect.com/science/article/pii/S1568494622008432#fig7>. This work by researchers Mehtab Suddle and Maryam Bashir sought to optimize the architecture of LSTMs. While the study used genetic algorithms in addition to LSTMs, it found the most success with higher dropout rates, which is reflected in our implementation of an LSTM model.

4) Summary of Our Contributions

Our main contribution will be the development of a machine learning model specifically tuned to detect satirical headlines from the Onion, which often resemble real headlines. That is, by exploring different model architectures we will apply an existing machine learning algorithm to a new domain (*Application*). This differs from prior work by attempting to detect satire or misinformation just from a headline; since readers often get most of their information from headlines (and skim the rest of the article), it is critical that, with the help of our model(s), they can discern between fake and real headlines. Additionally, we hope to expose advantages or flaws in using different model families, like simple logistic regression, in detecting satirical headlines (*Analysis*).

5) Detailed Description of Contributions.

Our approach to investigating how to classify satirical headlines begins with EDA and works towards more complex, higher capacity models. Firstly, some work completed was already discussed and outlined in the *Project Check-in* submitted earlier, which can be referenced in the attachments beyond the 6-page writeup (see *Project Check-in* and its corresponding Appendix). We avoid repeated discussion of such work here for brevity, including but not limited to: EDA and word distribution analysis (e.g. produced word clouds and confusion matrices) to understand dataset properties, preprocessing (e.g. bag of words and tokenizing with existing techniques like spaCy and Byte Pair Encoding), and initial logistic regression.

For additional work since, we used these features to train decision tree-based models/ensembles (e.g. Random Forest) and boosted variations of these (Adaboost). We also plan on tweaking existing CNN (1D architecture designed for text classification) and LSTM architectures to provide a higher capacity model (in hope of higher predictive power). This will get us to our main contributions, namely, producing a model that can predict satirical headlines and demonstrating the superiority of certain model families over others in text-related classification problems. The latter contribution is supported by plenty of research [Kalaivani, 2021]. We outline methods and architectures used before additional discussion in the subsequent section:

5.1 Methods

1) BPE Logistic Regression (*Baseline*): Used as our baseline, as BPE allowed us to quickly get a

tokenization output without too much pre-planning. We didn't do any text preprocessing. 2) spaCy

Logistic Regression: We then used spaCy, a rule-based tokenizer. Preprocessing here includes lowercasing, removing URLs/special characters, and removing stop words.

3) spaCy Logistic Regression with stopwords: We then continued with spaCy, but with preprocessing identical as above but keeping stop words.

4) Bag of Words Logistic Regression with stopwords: We then experimented with bag of words, with preprocessing identical as above.

5) Bag of Words Logistic Regression (no pre-processing): For the last logistic regression model, we used bag of words, without text preprocessing at all.

6) Decision Tree: After logistic regression, we experimented with tree-based models like Decision Trees to explore a higher capacity model family. Since they partition features into hierarchical structures based on simple decision rules, Decision Trees can capture complex relationships between features and labels, and require minimal data preprocessing.

7) Decision Tree (after tuning hyperparameters with Grid Search): After finding out which way to tokenize/pre-process the input resulted in the best Decision Tree, we used Grid Search to tune hyperparameters. Some of the hyperparameters tuned included max depth, min samples split, min samples leaf, and max features.

8) Decision Tree (after tuning hyperparameters with Randomized Search): Same as above but using Randomized Search rather than Grid Search, which is faster to execute and gives us a broader (though non-exhaustive) search.

9) Random Forest: Following Decision Trees, we experimented with Random Forest, since its power as an ensemble technique aggregating multiple decision trees could potentially improve the accuracy we were achieving by helping ameliorate overfitting symptoms. Random Forest handles high-dimensional feature spaces and can effectively deal with noisy/correlated features.

10) Random Forest (after tuning hyperparameters with Grid Search): Used to tune hyperparameters as we did with Decision Trees. Some parameters tuned included number of estimators, maximum depth, minimum samples split, minimum samples leaf, etc.

11) Random Forest Tree (after tuning hyperparameters with Randomized Search): Same as above but using Randomized Search.

12) 1D CNN: We then applied a 1D CNN model to attempt to classify the dataset. We decided this was an appropriate technique to explore, as the convolutional operation may work towards better capturing positional relationships within sentence segments.

13) LSTM: Aiming for improvements from the CNN model, we opted for a sequential model, using an LSTM as it was more likely to overcome the issues of exploding and vanishing gradients in training RNNs. We started with a high dropout rate, as supported by Suddle and Bashir's work discussed in part 3.

14) AdaBoost (after Grid Search): We then employed AdaBoost to experiment with reweighting hard-to-classify headlines. It's an ensemble method that combines multiple weak classifiers to create a strong classifier. We then optimized using Grid Search to fine-tune parameters.

15) GPT Models (gpt-3.5-turbo, finetuned gpt-3.5-turbo, gpt-4, and gpt-4o): As novel techniques, we finally explored several GPT versions to assess text classification capabilities. Initial tests with gpt-3.5-turbo. We then fine tuned this model on our dataset to enhance its accuracy in differentiating Onion and non-Onion headlines. Further, we tested gpt-4 and gpt-4o.

5.2 Experiments and Results

The key question our experiments will try to answer is whether it is possible to build a high performing classifier for Onion or non-Onion headlines. Our alternate hypothesis for this research question is that it is possible, and that an accuracy of at least 90% is achievable on the dataset. In addition to considering accuracy scores (which will define our alternate/null hypotheses), we will also consider other possible performance metrics such as, but not necessarily limited to, precision, recall, and/or F1 scores. These will also be evaluated in the context of certain task-specific objectives, such as what the weighting of precision vs recall in the context of a misinformation classifier more generally should be.

Note: The results of logistic regression were discussed in the Project Check-in, so we only briefly mention

accuracy here to avoid redundancy.

1) BPE Logistic Regression (*Baseline*): *74% accuracy*

2) spaCy Logistic Regression: *82% accuracy*

3) spaCy Logistic Regression with stopwords: *84% accuracy*

4) Bag of Words Logistic Regression with stopwords: *85% accuracy*

5) Bag of Words Logistic Regression (no pre-processing): *85.4% accuracy*

6) Decision Tree: We ran decision trees with default parameters on each of the ways to tokenize/preprocess the input mentioned above. The most accurate decision tree turned out to be the one run on the Bag of Words with no pre-processing, which is consistent with the behavior we saw with the Logistic Regression models. *It achieved a 75.1% accuracy.*

7) Decision Tree (after tuning hyperparameters with Grid Search/Randomized Search): It was interesting to find out that hyperparameter tuning yielded a worse model. After doing Grid Search the *accuracy was 72.7%*, whereas Randomized Search gave a model with *accuracy of 72.5%*. This is likely because the feature search space was not big enough in Grid Search, and likely because we need more iterations for Randomized Search. However, after applying these techniques for a while it became clear that to do any broader search would require computational resources that we didn't have, since cells were already taking close to an hour to terminate with the current feature search space.

8) Random Forest: Same as Decision Trees, we ran Random Forests with default parameters on each of the ways to tokenize/preprocess the input. It was interesting to find out that the best performing one this time was text preprocessing with spaCy keeping stopwords with *81.5% accuracy*, with bag of words with no pre-processing at a close second place with *81% accuracy*.

9) Random Forest (after tuning hyperparameters with Grid Search/Randomized Search): In this case hyperparameter tuning did result in a better model. While the cells took a long time to run (about an hour and a half each), Grid Search provided hyperparameters that resulted in a *82.2% accuracy*, and Randomized Search (ran for 100 iterations) provided hyperparameters that *resulted in a 82.3% accuracy*.

10) 1D CNN: Our 1D CNN model is characterized as follows: We first split into a 60/20/20 train/test/val split. We then used Tokenizer to tokenize our text data with a word list length of 10,000, and also padded input sentences accordingly to be fed into the network. Our model consists first of an Embedding layer of embedding length 300, followed by a 128 filter count and 5 kernel-size 1D convolution, max pool, fully connected with 64 nodes (with 0.5 dropout), and an output sigmoid node. We also trained using the Adam

optimizer, with early stopping based on validation loss. *This method achieved test accuracy of 87%, and F1-score of 0.82.*

11) LSTM: Similarly to the CNN model, we used a 60/20/20 train/test/val split. We used Tokenizer to tokenize our text data with a word list length of 10,000, and also padded input sentences accordingly to be fed into the network. Our model consists first of an Embedding layer of embedding length 128, reusing the embedding length from the paper by Suddle and Bashir in part 3. We started with a high recurrent dropout rate of 0.8, as recommended by Suddle and Bashir, but after tuning, we arrived at a recurrent dropout of 0.3 for the single LSTM layer (as many units as embedding length), followed by a single fully connected layer with sigmoid nonlinearity. Again, we used Adam as the optimizer on binary cross entropy loss. *This achieved a test accuracy of 87% and F1-score of 0.82.*

12) AdaBoost: We applied the AdaBoost classifier to headlines, enhancing text classification with TF-IDF vectorization combined with 'stop_words_count' and 'original_length' features. The data was split into 80% training and 20% testing sets, and further refined by selecting the top 100 features through chi-squared tests. Optimized via GridSearchCV for parameters like 'n_estimators' and 'learning_rate', *the model achieved an accuracy of 76.6%*. This streamlined approach significantly improved our ability to distinguish between 'Onion' and 'Not Onion' headlines, effectively demonstrating the utility of ensemble methods in text classification.

13) GPT Models: We integrated GPT models (gpt-3.5-turbo, finetuned gpt-3.5-turbo, gpt-4, and gpt-4o) to classify headlines as 'Onion' or 'Not Onion' via the OpenAI API, using a custom function to automate and standardize queries to each. Each model's performance was assessed under a strict time constraint of 300 seconds for batch processing, ensuring efficiency. The models processed headlines with varying effectiveness: gpt-4 achieved the *highest accuracy at 68.4%*, followed by the fine tuned gpt-3.5-turbo at

64.7%, the standard gpt-3.5-turbo at 62.3%, and gpt-4o at 60.5%. This evaluation highlighted the differences in model capabilities to discern complex text nuances in a controlled testing environment.

Note on Pre-trained Embeddings: Although we originally planned to use pre-trained embeddings (e.g. Word2Vec) with which to train a deep learning model, we ultimately decided that headline data have idiosyncratic properties that may not be well captured with generic word embeddings trained on a larger corpus, and that with a dataset size of 24,000 consisting of data with fairly unique characteristics (e.g. relatively smaller vocabulary corpus, short sentence length), we can attempt to train our own embeddings from scratch. As a result, we trained embedding layers through the Keras library.

For a complete table of results, see Appendix 1.

Results Discussion: As we mentioned in the check-in report, for logistic regression models, the best performing one is also the one with the least pre-processing and no BPE/spaCy tokenization. This is likely due to pre-processing resulting in the removal of stopwords, special characters, capitalization, links, etc., which turn out to be useful in differentiating an Onion vs non-Onion headline. Hence, the model that does not remove these has additional context and is better at the classification task. Further, it appears that tokenization (like spaCy) modifies headlines by further removing some non-stopwords (e.g., the word “someone”) and by conjugating verbs into their root form. For example, the original headline “My Doctor Told Me I Should Vaccinate My Children, But Then Someone Much Louder Than My Doctor Told Me I Shouldn’t” will become “doctor tell vaccinate child louder doctor tell not” after preprocessing and spaCy tokenization. As we can see a lot of context is lost, and we believe this explains why a simple bag of words without any preprocessing is the best model among logistic regression.

There are a couple of factors that may explain the logistic regression models outperforming decision trees and random forest. The most likely appears to be insufficient hyperparameter tuning (e.g. n_estimators, depth). In future projects it would be interesting to further explore hyperparameter tuning, but it’s clear to us after this experience that much more compute power is needed to do this effectively than the default Google Collab compute units. While we attempted hyperparameter tuning for a significant time, execution times were so large that it wasn’t possible to explore this as much as we desired.

Finally, to answer our *Analysis* questions of why certain models outperformed, we believe that our top performing models 1D CNN and LSTM outperformed other model families for several reasons. For the 1D CNN, our custom learned word embeddings combined with convolution operations helped capture the definitional/spatial context of words, with a kernel size of 5 being sufficient for short headlines with average lengths of ~12 words. Similarly, we believed that an embedding + LSTM model would capture the sequential nature of the headlines, as RNNs and LSTMs are often used for this reason for NLP. Compared with other model families like logistic regression using bag of words, we believe this allowed better capturing of inter-word relationships with headline sentences.

6) Compute/Other Resources Used

For the GPT baseline classification models in our project, we utilized the OpenAI API, employing models gpt-3.5-turbo, finetuned gpt-3.5-turbo, gpt-4, and gpt-4o. Apart from this, we generally relied on the compute provided through Google Colab.

7) Conclusions

To recap, in this project, we attempted to train a model that could discern between satirical and real-world news headlines, as proxied with Onion headlines for the former. This proved to be a challenging task, as we unfortunately could not reject our null hypothesis of only being able to achieve less than 90% accuracy on this dataset. This reinforces the opinion that it is hard to spot fake news both as a human or with machine learning, and further stresses the importance of machine learning research in fake news and in satire in general. Despite this, we believe our research offers a building block for future exploration by

comparing the performance of different model families on this problem.

The 1D CNN model performed the best on this dataset, followed closely by the LSTM and the logistic regression (no preprocessing). From these results, we first learned that a 1-dimensional “spatial” model can capture similar information about natural language as can a sequential model like an LSTM. Additionally, we were surprised that the logistic regression model performed similarly to the higher capacity models, possibly due to insufficient hyperparameter tuning. Finally, as was discussed earlier, we learned that word removal often hurts model performance, as removing stop words or other words often leaves out important context information from the headlines.

For the future, given more time and compute, we would perform a more thorough hyperparameter search for each model, creating more robust models that could be used commercially. In addition, we would try to build a CNN-LSTM model, combining the spatial capacity of the former with the sequential capacity of the later, potentially building a more complete model of headlines. Touching on the issue of ethics and broader social and environmental impact, we want to ensure that the idea of curbing misinformation does not become a euphemism for censorship. We would need to determine whether it is more important to identify more fake news (sensitivity) or to limit censoring real information (specificity). Because this is a tradeoff, we would consult with experts in politics, social dynamics, and journalism to develop an ideal mix between the two metrics. Information is spreading quicker than ever, and unfortunately, so is misinformation; we hope that our work with satirical headlines has implications for a more robust classifier of fake news, so that users can be protected from misinformation online.

Other Prior Work / References (apart from Sec 3)

M. S. Kalaivani, S. Jayalakshmi, “Comparative Analysis of Convolutional Neural Network and LSTM in Text-Based Sentiment Classification,” IEEE 2021.

Broader Dissemination Information:

Your report title and the list of team members will be published on the class website. Would you also like your pdf report to be published?

No

If your answer to the above question is yes, are there any other links to github / youtube / blog post / project website that you would like to publish alongside the report? If so, list them here.

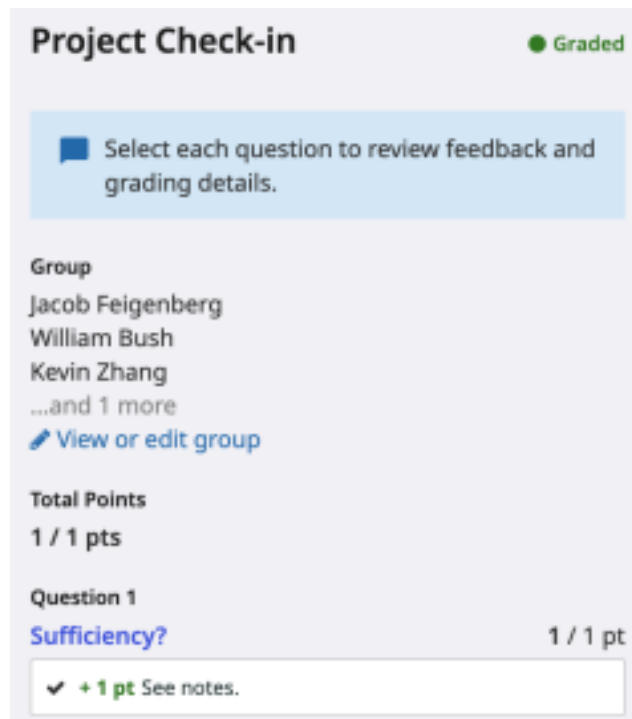
N/A

Full Work Plan / Work Report (completed in green)

| Person(s) | Task | Apr 9 - Apr 15 | Apr 16 - Apr 22 | Apr 23 - Apr 29 | Apr 30 - May 6 | May 7 - May 15 |
|--------------|-------------------------------|----------------|-----------------|-----------------|----------------|----------------|
| Kevin, Jacob | Review Related Works | | | | | |
| Kevin, Jacob | Project Check-in Writeup | | | | | |
| Liam | Data Preprocessing | | | | | |
| Liam | EDA, Bag of words | | | | | |
| Eduardo | Logistic Regression | | | | | |
| Eduardo | Modeling with Spacy, BytePair | | | | | |
| Eduardo | Tree-based Models | | | | | |
| Jacob, Kevin | CNN | | | | | |
| Jacob, Kevin | LSTM | | | | | |
| Liam | Adaboost/XGBoost | | | | | |
| Liam | ChatGPT Baseline | | | | | |
| Everyone | Final Writeup/Presentation | | | | | |

(Exempted from page limit) Attach your midway report here, as a series of screenshots from Gradescope, starting with a screenshot of your main evaluation tab, and then screenshots of each page, including pdf comments. This is similar to how you were required to attach screenshots of the proposal in your midway report.

Gradescope



The screenshot shows the 'Project Check-in' interface in Gradescope. At the top, it says 'Project Check-in' with a green 'Graded' status. Below this is a blue box with the instruction: 'Select each question to review feedback and grading details.' Underneath, the 'Group' is listed as Jacob Feigenberg, William Bush, Kevin Zhang, and one more person, with a link to 'View or edit group'. The 'Total Points' section shows '1 / 1 pts'. The 'Question 1' section is titled 'Sufficiency?' and shows '1 / 1 pt'. At the bottom, there is a green checkmark and '+1 pt See notes.'

TA Feedback (via email)



The screenshot shows an email from Kunli Zhang (kunli@seas.upenn.edu) to Eduardo, kevin@wharton.upenn.edu, me, and William. The email is dated Thursday, April 25, 11:15 AM. The body of the email says: 'Hi all, Thanks for submitting your project check in. The writeup looks great! I only have one point of feedback: for your final write up, it might be useful to include some sort of summary table comparing the different models you have. You can also discuss why you think certain models performed better or worse.' The email ends with 'Best, Kunli' and a yellow Kunli logo.

Project Check-in

Don't Eat the Onion:

Detecting "The Onion" vs non-Onion Headlines

Team: Jacob Feigenberg, Kevin Zhang, Eduardo Gonzalez, Liam Bush. **TA:** Kunli Zhang

1) Introduction

Problem: "The Onion" (hereby simply, Onion) is a newspaper organization that publishes satirical news articles. As a famous satire publication, Onion headlines/articles have often mistakenly been taken seriously. One such instance was in May 2015, when former FIFA vice president Jack Warner mistakenly spread the Onion headline "FIFA Frantically Announces 2015 Summer World Cup In United States" as real news on Facebook. We also see more serious implications of this (discussed in "Motivation"). There is even a dedicated subreddit, *r/AteTheOnion*, with nearly 600K members focused on those who failed to see through The Onion's satire. As avid readers of The Onion, we decided on the following task:
We aim to detect "The Onion" compared to non-Onion headlines. Our model will be able to output/classify, for a given headline, whether it is an Onion or non-Onion headline.

The inputs to the system are headlines in text form, and the outputs will be a binary classification of whether the headline is Onion or non-Onion. We are using data from a dataset of 24,000 total Onion and non-Onion article headlines, with the non-Onion articles being "Onion-like" sourced from *r/NotTheOnion* (subreddit dedicated to real headlines that may appear satirical). The breakdown of the dataset is 15,000 non-Onion headlines, and 9,000 Onion headlines. The dataset is linked here: <https://github.com/lukefeilberg/onion>. We will be splitting this into a 60/20/20 train/validation/test split, with the validation set used as required for any hyperparameter selection. Our aim is to build the best classification system possible, and foresee creating/tuning several iterations of model types/parameters to achieve this objective and contribute relative to previous work (see later sections).

To evaluate our proposed system, we will measure log loss between our predicted Onion/non-Onion label and actual labels for training, and for further evaluation/test, consider metrics such as classification accuracy, precision, recall, and/or F1 scores where appropriate.

Motivation: While the classification of articles as Onion vs non-Onion may seem frivolous at first glance, we believe the implications are actually deep. The very fact that entire subreddits have been dedicated to both the failure to identify Onion articles (*r/AteTheOnion*) or to real headlines that seem Onion-esque (*r/NotTheOnion*) highlights an important point: Information, specifically what is real/fake, exaggerated/accurate, and satire/serious, actually walks a fine line between each contrast. In an age where information spreads quicker than ever, systems that can discern between these contrasts will also become more important. As such, a high performing system in this project offers a step forward in the fight against the spread of misinformation, even if such misinformation is more light-hearted in nature. We also believe that the flip side to this is to

ensure that the idea of curbing misinformation does not inadvertently become a euphemism for censorship. As students, we commit to ethical research practices, prioritizing that our system will have limited risk for any chance of user harm, of which we see limited chances of within our specific aim of a classification problem.

2) How We Have Addressed Feedback From the Proposal Evaluations

Kunli's feedback on our initial proposal was very helpful, especially his note, "I am not sure about the datasets you are using. It seems that the journal entries are already in string data. Where will you be getting your handwritten journal data from?" Having reviewed the feedback, our group realized that our initial first proposal, "Analyzing mental health through sentiment analysis on handwritten journaling entries", lacked the necessary dataset to be able to fully evaluate our pipeline performance from handwritten journal entries to sentiment analysis scoring. As such, we subsequently met Kunli during his Office Hours, where we discussed several alternatives, eventually mutually agreeing that the classification between Onion and non-Onion article headlines offers a challenging and interesting project idea which we are now implementing. The most important piece of feedback that we received is that we need to work with a proper dataset; we selected data that would be the best fit for this classification task and will introduce it in later sections.

Lastly, another piece of feedback Kunli offered which, *time permitting*, can be explored as an additional feature is to develop a user interface that allows users to input headlines and receive classification results. This interactive feature will be powered by a model trained on a dataset comprising both satirical and genuine news headlines. The model architecture will be enhanced to handle queries efficiently and provide accurate classifications. This allows us to further work towards our stated aim within "Motivation" to consider the problem of classifying Onion vs non-Onion articles as a sub-problem of the general fight against misinformation, and would permit us to better spread awareness of the problem through a more light-hearted application.

3) Prior Work We are Closely Building From

A few works inspired our contribution to the issue of headline satire detection and, more broadly, to the issue of misinformation.

Automatic Satire Detection: Are You Having a Laugh? <https://aclanthology.org/P09-2041.pdf>.

This work by Clint Burfoot and Timothy Baldwin of the University of Melbourne used SVMs and specially constructed features to delineate satirical news articles from real news, giving us a reference point to which we can compare complicated learned features (deep learning) and a simpler bag-of-words approach. The authors also emphasized the importance of headlines as a feature in classifying satirical news, supporting our approach of focusing on Onion headlines.

Fake News Detection: A Deep Learning Approach.

<https://scholar.smu.edu/cgi/viewcontent.cgi?article=1036&context=datasciencereview>. This work by Aswini Thota, Priyanka Tilak, Simrat Ahluwalia, and Nibrat Lohia of Southern Methodist University gave us inspiration on how to use pre-trained word embeddings (like those from Google's Word2Vec) to feed into our own neural networks.

4) What We are Contributing

Our main contribution will be the development of a machine learning model specifically tuned to detect satirical headlines from the Onion, which often resemble real headlines. That is, by tuning and using an LSTM, we will apply an existing machine learning algorithm to a new domain (application). This differs from prior work by attempting to detect satire or misinformation just from a headline; since readers often get most of their information from headlines (and skim the rest of the article), it is critical that, with the help of our model(s), they can discern between fake and real headlines. Additionally, we hope to expose flaws in using other models, like CNNs (designed for images), in detecting satirical headlines (analysis).

5) Detailed Description of Each Proposed Contribution, Progress Towards It, and Any Difficulties Encountered So Far

5.1 Methods

Our approach to investigating how to classify satirical headlines will begin with exploratory data analysis and work from simpler to more complex, higher capacity models. Much of this is reiterated below, but we've looked at word distributions and produced word clouds and confusion matrices to understand the properties of this headline dataset. The next step was preprocessing: creating our own bag of words as well as tokenizing with existing techniques like spaCy and Byte Pair Encoding. These handcrafted features were then fed into a logistic regression classifier to get our first results (discussed below). Next, we plan to use these features to train decision tree-based ensembles (random forest) and boosted variations of these models (Adaboost, XGBoost). Similarly to Thota et al. (Section 3), we plan on using pre-trained word embeddings to feed into neural networks; for this deep learning stage, we also plan on tweaking existing CNN (1D architecture designed for text classification) and LSTM architectures to provide a higher capacity model (in hope of much higher predictive power). This will get us to our main contributions, namely, producing a LSTM model that can predict satirical headlines and demonstrating the LSTMs over CNNs in text-related classification problems. The latter contribution is supported by plenty of research [Kalaivani, 2021].

5.2 Experiments and Results

The key question our experiments will try to answer is whether it is possible to build a high performing classifier for Onion or non-Onion headlines. Our alternate hypothesis for this research question is that it is possible, and that an accuracy of at least 90% is achievable on the dataset. In addition to considering accuracy scores (which will define our alternate/null hypotheses), we will also consider other possible performance metrics such as, but not necessarily limited to, precision, recall, and/or F1 scores. These will also be evaluated in the context of certain task-specific objectives, such as what the weighting of precision vs recall in the context of a misinformation classifier more generally should be.

Our results thus far are as follows:

1. Firstly, our experiment begins with better understanding our dataset. To do this, we utilized various techniques such as plotting word distributions, word clouds, and

confusion matrices to better ascertain the dataset's distributions. *Please see Appendix 1 for some results.*

2. We also worked on preprocessing our data. Some examples of this are removing stop words, and also exploring bag of words and tokenization techniques like spaCy or Byte Pair Encoding. *Please see Appendix 2 for some results.*
3. Finally, we begin to set up actual prediction models for our classification task. We decided that a simple *baseline* for our task will be to utilize logistic regression, specifically with Byte Pair Encoding (BPE) as an initial attempt before further iteration. We also varied the level of preprocessing. A summary of results so far are as follows:
 - a. Baseline: Logistic regression, BPE: We used this as our baseline as BPE allowed us to quickly get a tokenization output without too much pre-planning. We didn't do any text preprocessing. This achieved a test accuracy of 0.74. *Please see Appendix 3 for some results.*
 - b. Iteration 1: We then considered a rule-based tokenizer, specifically spaCy. Preprocessing here comprised of lowercasing, removing URLs/special characters, and removing stop words. This achieved a test accuracy of 0.82. *Please see Appendix 4 for some results.*
 - c. Iteration 2: We then continued with spaCy, but with preprocessing identical as Iteration 1 but keeping stop words. This achieved a test accuracy of 0.84. We believe the higher accuracy may indicate going forward that stop words provide additional context to the classifier for Onion articles. *Please see Appendix 5 for some results.*
 - d. Iteration 3: We then experimented with bag of words, with preprocessing identical as Iteration 2. This achieved a test accuracy of 0.85. *Please see Appendix 6 for some results.*
 - e. Iteration 4: Finally, we tried bag of words, without text preprocessing. This achieved a test accuracy of 0.854. We believe the higher accuracy indicates that removed strings such as links may provide additional context to the classifier. *Please see Appendix 7 for some results.*

6) Risk Mitigation Plan

To build a viable project within the remaining time, we plan to iterate through various model types (e.g. LSTM, tree-based models, etc.) in order to move towards our objective of the highest performing classifier possible. At each iteration, we will consider the pros/cons of prior approaches before considering next steps to maximize efficiency. For example, if we find prior model iterations tend to overfit, we will take this learning into account before deciding on next model types or hyperparameters going forward. With our group having made considerable progress already in EDA, preprocessing, and baseline model building, we have created a "simplified setting" with early results to continue building off of. While we do not believe compute will become a bottleneck, in the scenario this occurs, we will reduce the size of our training set by subsampling to ensure on-time project completion. Finally, if any other unforeseen headwind occurs, we plan to firstly communicate with our TA mentor, before then considering approaches such as further dataset modification, model complexity downgrading, or reframing objectives.

Other Prior Work / References (apart from Sec 3)

M. S. Kalaivani, S. Jayalakshmi, "Comparative Analysis of Convolutional Neural Network and LSTM in Text-Based Sentiment Classification," IEEE 2021.

Full Work Plan (completed in green)

| Person(s) | Task | Apr 9 - Apr 15 | Apr 16 - Apr 22 | Apr 23 - Apr 29 | Apr 30 - May 6 | May 7 - May 13 (2 day buffer before final deadline) |
|--------------|--|----------------|-----------------|-----------------|----------------|--|
| Kevin, Jacob | Review Related Works | | | | | |
| Kevin, Jacob | Project Check-in Writeup | | | | | |
| Liam | Data Preprocessing | | | | | |
| Liam | EDA, Constructing bag of words featurization | | | | | |
| Eduardo | Logistic Regression | | | | | |
| Eduardo | Modeling with Spacy, BytePair | | | | | |
| Eduardo | Tree-based Models | | | | | |
| Jacob, Kevin | CNN | | | | | |
| Jacob, Kevin | LSTM | | | | | |
| Liam | Adaboost/XGBoost | | | | | |
| Liam | Baseline test on ChatGPT API | | | | | |
| Everyone | Final Writeup and Presentation | | | | | |

Gradescope

Project Proposal

Graded

Select each question to review feedback and grading details.

Group

William Bush

Kevin Zhang

Jacob Feigenberg

...and 1 more

View or edit group

Total Points

1 / 1 pts

Question 1

Sufficiency?

1 / 1 pt

✓ 1 pt See notes.

TA Feedback (via email)

Kunli Zhang <kunli@seas.upenn.edu>

Mon, Mar 25, 11:23AM ☆ ↶ ⋮

to William, Eduardo, kevinylz, me ↵

Hi William, Eduardo, Kevin, and Jacob,

I am Kunli and I will be your 4190/5190 project TA. If at any point during your project you have any questions, please feel free to reach out. Here is your preliminary **feedback** from your project **proposal**: analysing mental health through sentiment analysis on handwritten journaling entries.

You have clearly articulated the problem that you wish to investigate. Your project is feasible and well articulated.

I am not sure about the datasets you are using. It seems that the journal entries are already in string data. Where will you be getting your handwritten journal data from?

There is a wealth of models available for this type of project. Be sure to investigate the possibility of fine-tuning pre-trained neural networks and models for NLP.

Lastly, I'm not sure what a "continuous sentiment output" might look like. Be sure to clearly explain what this is when implementing the project going forward.

This project looks really interesting and I'm looking forward to seeing your results!

If you have any questions please let me know!

Kunli

Analyzing mental health through sentiment analysis on handwritten journaling entries

Team: Kevin Zhang, Eduardo Gonzalez, Jacob Feigenberg, Liam Bush

Task T: We aim to analyze mental health states based on sentiment analysis of journaling entries. Our model will be able to convert images of users' handwritten journaling entries to text, after which sentiment analysis is performed to analyze their mental health states.

Experience E: We are using data from a dataset with 1500 journal entries from 500 respondents to train a model on sentiment analysis of text (<https://www.kaggle.com/datasets/madhavmalhotra/journal-entries-with-labelled-emotions/data>). This will be split into a training set, validation set, and test set for the model. We will also use another dataset containing handwritten text so that we can build a full pipeline from written journal entries to estimated sentiment (<https://www.kaggle.com/code/dromosys/handwriting-recognition-cnn/input>). This will also undergo a train-val-test split for the other model.

Performance Metrics P: We will measure cross entropy loss between our predicted emotion labels and actual labels through a test set. Furthermore, we may also implement a form of regression to classify intensity of positive and negative emotions, in which case we would also add a mean square error performance metric to measure performance on regression.

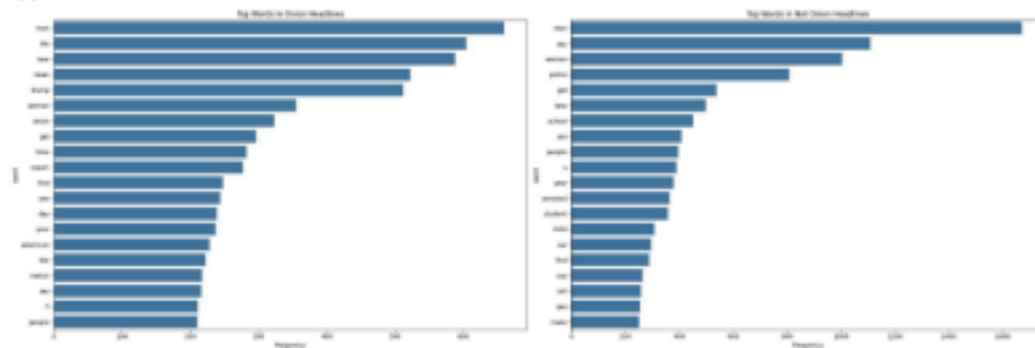
Prior Work:

1. [Kalluri Shashank](#), "Deep Learning Based Sentiment Analysis", Blekinge Institute of Technology 2023. (This paper studies text data coming from various social media platforms and performs sentiment analysis on the entries).

Nature of Main Proposed Contribution(s):

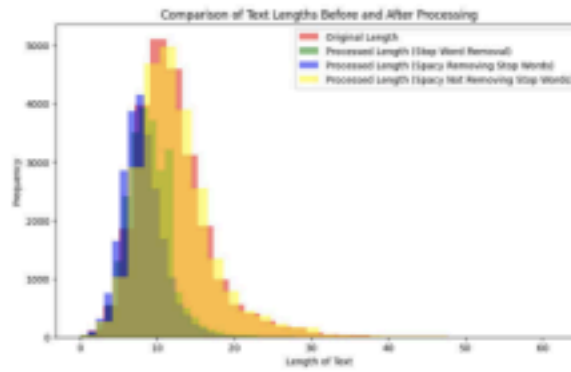
Our main contribution will be the development of a machine learning model specifically tuned to detect mental health labels in personal journal entries. This differs from prior work by focusing on unsolicited, naturalistic text data from individuals over time, providing insights into daily life that clinical transcripts might not capture.

Why We Care: We're diving into this project because we believe in the power of merging tech with mental health. By analyzing journal entries for mood trends, we can create something that could make a difference. It allows us to spot early signs of stress or depression and do





Appendix 2



Appendix 3

```
1 # Initialize Byte Pair Encoding tokenizer
2 tokenizer = ByteLevelBPETokenizer()
3 texts = data['text'].tolist()
4 tokenizer.train_from_iterator(texts)
5
6 def tokenize_with_bpe(text):
7     encoding = tokenizer.encode(text)
8     return ' '.join(encoding.tokens)
9
10 # Vectorize the text data using the BPE tokenizer
11 vectorizer = CountVectorizer(tokenizer=tokenize_with_bpe)
12
13 # Fit and transform the vectorizer on your text data
14 X_bpe = vectorizer.fit_transform(data['text'])
15
16 # Split the data into training and testing sets
17 X_train_bpe, X_test_bpe, y_train_bpe, y_test_bpe = train_test_split(X_bpe, data['label'], test_size=0.2, random_state=42)
18
19 # Initialize the Logistic Regression model
20 model_bpe = LogisticRegression(max_iter=10000)
21
22 # Train the model
23 model_bpe.fit(X_train_bpe, y_train_bpe)
24
25 # Predicting the Test set results
26 y_pred_bpe = model_bpe.predict(X_test_bpe)
27
28 # Evaluating the model
29 print("Performance with Byte Pair Encoding and Logistic Regression:")
30 print(classification_report(y_test_bpe, y_pred_bpe))
31 score_bpe = accuracy_score(y_test_bpe, y_pred_bpe)
32 print("Model accuracy with Byte Pair Encoding and Logistic Regression: ", score_bpe)
```

```

Performance with Byte Pair Encoding and Logistic Regression:
precision    recall  f1-score   support

     0       0.76    0.86    0.81     3018
     1       0.70    0.53    0.60     1782

 accuracy          0.74     4800
 macro avg       0.73    0.70    0.71     4800
weighted avg       0.74    0.74    0.73     4800

Model accuracy with Byte Pair Encoding and Logistic Regression: 0.7414583333333333

```

Appendix 4

```

Performance with spaCy preprocessing:
precision    recall  f1-score   support

     0       0.85    0.88    0.86     3018
     1       0.78    0.73    0.75     1782

 accuracy          0.82     4800
 macro avg       0.81    0.80    0.81     4800
weighted avg       0.82    0.82    0.82     4800

Model accuracy with spaCy preprocessing: 0.8233333333333334

```

Appendix 5

```

Performance with spaCy preprocessing including stop words:
precision    recall  f1-score   support

     0       0.86    0.90    0.88     3018
     1       0.82    0.75    0.78     1782

 accuracy          0.84     4800
 macro avg       0.84    0.83    0.83     4800
weighted avg       0.84    0.84    0.84     4800

Model accuracy with spaCy preprocessing including stop words: 0.8447916666666667

```

Appendix 6

```

Performance on original text Basic Pre-Processing (just URLs, special characters, HTML stuff):
precision    recall  f1-score   support

     0       0.86    0.90    0.88     3018
     1       0.82    0.76    0.79     1782

 accuracy          0.85     4800
 macro avg       0.84    0.83    0.84     4800
weighted avg       0.85    0.85    0.85     4800

Model accuracy on original Basic Pre-Processing (just URLs, special characters, HTML stuff): 0.8497916666666667

```

Appendix 7

```

Performance on original text without spaCy and stop words removal:
precision    recall  f1-score   support

     0       0.86    0.91    0.89     3018
     1       0.83    0.76    0.79     1782

 accuracy          0.85     4800
 macro avg       0.85    0.83    0.84     4800
weighted avg       0.85    0.85    0.85     4800

Model accuracy on original text without spaCy and stop words removal: 0.85375

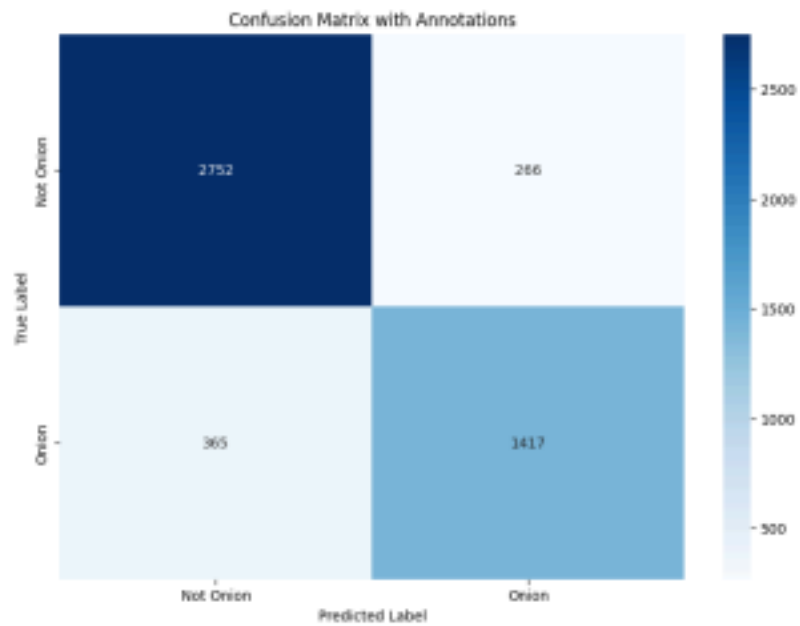
```

Appendix 1:

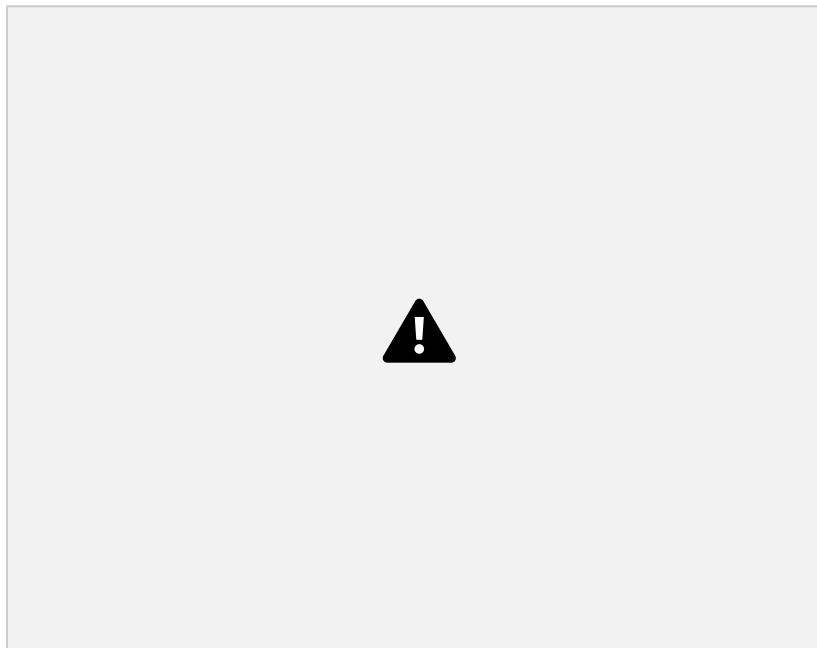
| Results Summary Table | Accuracy | Precision | Recall | F1 |
|--|---------------|-----------|-----------|-----------|
| BPE Logistic Regression (<i>Baseline</i>) | 74.1% | 74% | 74% | 73% |
| spaCy Logistic Regression | 82.3% | 82% | 82% | 82% |
| spaCy Logistic Regression with stopwords | 84.4% | 84% | 84% | 84% |
| Bag of Words Logistic Regression with stopwords | 85.0% | 85% | 85% | 85% |
| Bag of Words Logistic Regression (no pre-processing) | 85.4% | 85% | 85% | 85% |
| Decision Tree | 75.1% | 75% | 75% | 75% |
| Decision Tree (after Grid/Randomized Search) | 72.7% / 72.5% | 73% / | 73% / 73% | 73% / 72% |
| Random Forest | 81.5% | 72% | 82% | 81% |
| | | 82% | | |
| Random Forest (after Grid/Randomized Search) | 82.2% / 82.3% | 84% / 84% | 82% / 82% | 81% / 81% |
| 1D CNN | 86.9% | 83.9% | 79.9% | 81.9% |
| LSTM | 86.5% | 80.6% | 83.8% | 82.2% |
| Ada Boost (after Grid Search) | 76.6% | 76% | 76% | 77% |

| | | | | |
|------------------------|-------|-------|-------|-------|
| gpt-3.5-turbo | 62.3% | 63.1% | 95.8% | 76.1% |
| Fintuned gpt-3.5-turbo | 64.7% | 62.4% | 64.8% | 58.2% |
| gpt-4 | 68.4% | 71.9% | 81.0% | 76.2% |
| gpt-4o | 60.5% | 74.2% | 55.5% | 63.5% |

Appendix 2: CNN Confusion Matrix



Appendix 3: CNN ROC Curve



Appendix 4: LSTM Confusion Matrix



Appendix 5: LSTM ROC Curve

