Fake it Till You Make it: Fake Review Detection and Generation

Jose Giron, Jonathan Gomes-Selman, Nikita Demir
Department of Computer Science
Stanford University
jmgiron@stanford.edu, jgs8@stanford.edu, ndemir@stanford.edu

1 Introduction

We live in a time of fake news and misinformation. As the use of digital platforms in daily life continues to grow so does the potential for malicious digital information to skew perception. Increasingly, online reviews have become customers' primary influence in their decision making. Research has shown that a 1 star change in a business's reviews translates to anywhere between a 5 and 9 percent change in revenue [8]. However, the nature of online ratings means that it's easy to manipulate reviews, and businesses have been quick to notice. Sites have popped up online that promise gleaming reviews for a person's business or worse, negative reviews for the competition. Thus, there is a clear need for algorithms that can reliably detect when a review is fake. This problem is particularly challenging because most fake reviews are created by humans and therefore are nearly indistinguishable from the real ones. To tackle this problem compare and contrast different fake review classifiers. The input to our algorithms are a combination of word features (unigram, bigram, trigram counts) and reviewer behavioral features (average review rating, average length of review, etc..). We then apply an ensemble of different machine learning techniques, such as NaiveBayes, SVM with linear kernel, and Random Forests in addition to more complex deep models like LSTM, FastText, and CNN.

In addition to classification, we implement a context-specific review generator based on the open source OpenNMT-py seq-2-seq model. The generator takes in a list of contexts such as desired review rating and type of food in order to output a convincing fake review. We then evaluate the success of our generated results to test the efficacy of using generated models applied to reviews as well as to test the genarlizability of our classifiers.

2 Related Work

The importance of reviews for commerce and user experience has made detecting fake reviews a hot research topic. Previous approaches primarily compare and contrast two features for classification: word or text based features and metadata / review centric based features. Papers have presented varying result quality when utilizing just text based features for classification. For example, Wang et al. reports a best accuracy and corresponding f-scores of 80 % using SVM models with unigram feature vectors [4]. In contrast, Mukherjee et al. provide an in depth argument against the efficacy of utilizing strictly text based features due to very similar word distributions between fake and non fake reviews, showing peak accuracy of 65% when using SVMs trained on unigrams[7]. Additionally, previous NLP based models primarily utilize simple Machine Learning models such as SVM and Logisitic regression, thus motivating the exploration of new state of the art Deep Learning models for classification.

Considering the behavioral features of reviews has shown promising results. Both Mukherjee et al. and Wang et al. demonstrate significant classification improvement when considering behavioral features over text based features [7][8]. However, a previous CS229 paper, Zhang et al. provides conflicting results in this domain [9]. They shows little predictive power in the behavioral features, showing not nearly as stark differences in the behavioral features between fake and non-fake reviewers as Mukherjee et al.

We believe that the great discrepancy between the results for different papers presents a strong additional motivation for our work. These past approaches appear to use very similar datasets (i.e. Yelp datasets for fake / non fake reviews in Chicago which they unfortunately do not cite), yet they get very different results; therefore, not only do we look to better understand the potential for developing classification models for fake review detection, but we also look to test the replicability of past research that seems contradictory. Moreover, we specifically highlight the past work of Zhang et al. Although we initially took inspiration from the success of their models, upon further review we identify one key limitation in their results: their feature engineering methods. We see that the important features used for classification are the top 100 unigrams and top 100 bigrams with greatest count differences between the fake and real reviews. Because these features are calculated over the entire dataset and then used during cross validation, information is not being held out for the respective validation set. In fact, the engineering of these features implicitly gives the classifier information about the validation/test set, allowing greatly inflated accuracy and f-scores.

Lastly, for the task of generating context specific restaurant reviews we draw inspiration from the use of sequence-2-sequence models. We reference work by Juuti et al. outlining a principled approach for generating context specific reviews using the openNMT software [2]. We look to implement and improve portions of their described methods to both qualitatively analyze the results and quantitatively analyze the ability of our classifiers to detect computer generated fake reviews.

3 Dataset

We used two datasets during our project, one to explore the problem of fake review classification and the other for generation of context specific fake reviews.

3.1 YelpZip - Classification

The YelpZip dataset contains 61,541 fake and not fake yelp reviews for restaurants in Chicago, of which 53,400 are real and 8,141 are fake. The dataset is compiled using Yelp's proprietary algorithm, which flags reviews believed to be fake. Although the labels provided by Yelp are not guaranteed to be correct, studies have shown that Yelp's algorithm is quite robust and, thus, we treat the labels as ground truth [1]. In addition to a binary class label, each review contains a small set of metadata describing that review. For a given review the metadata includes: creation date, reviewer's ID, product ID, and the star rating.

Because of the large category imbalance, only 13.2% fake reviews, we employ uniformly undersampling over the majority class (real reviews) to produce a reduced dataset of 16,282 examples with an equal distribution of fake and not fake reviews.

As we discuss later, even for a knowledgeable human reader, there is very little discernible difference between a fake and real review. To illustrate, we include an example, can you spot the difference?

> Great ambiance, amazing food. The staff is wonderful. I have gone three weeks in a row and plan on going again soon. A great summer get-a-way!
>
> Delicious food, great value, excellent service and byob. What else do you possibly need? The dinner specials are recommended. They are a lot of food and the pad Thai is exceptional!

Spoiler alert: the first review is fake and the second is real.

3.2 Yelp Dataset - Generation

For the problem of context specific review generation, because we only train on real reviews we expand our dataset to the much larger Yelp Open Dataset [9]. This dataset contains a corpus of 3.5 millions restaurant reviews and accompanying restaurant specific information. For training our sequence-2-sequence model, we create a parallel dataset of (context, review) pairs where the context of a review is an array containing [review rating, restaurant name, city, state, restaurant specific tags (e.g. Diner, Korean)]. Note, during pre-processing we tokenize each review to remove non-ascii characters, extra whitespace, and separate out punctuation.

4 Features

For our baseline models (Linear Regression, SVM, Naive-Bayes, Random Forests), we use unigrams count vectors to represent each review. We chose to use count vectors because we saw little improvement using more complex techniques eg. Tf-idf vectors.

For our more complex models (FastText, CNN, LSTM), we use pre-trianed word embeddings (Glove-100d) to represent the reviews and further updated the word embeddings during training [10]. We also tokenize our data using spaCy to remove non-ascii characters, extra whitespace, and separate out punctuation.

Based on the conclusion from Mukherjee et al. we also consider 5 behavioral features which compare how fake vs real reviewers behave. These are maximum content similarity between any two reviews by a reviewer (measured using cosine similarity); average review length; maximum number of reviews in a given day; percent of positive reviews; average rating deviation which looks at the average amount a review deviates from other reviews in a given product (ranges from 0-4 in a 5 star system).

We were surprised to see that using these features on our dataset the real and fake reviews were almost indistinguishable. As shown in Figure 1 the word distributions for the two classes are nearly identical. This shows that the fake and real reviews largely cannot be distinguished by word features. In addition, we found that behavioral features as shown in Figure 2 have much less of a spread between fake and real reviewers than that reported by Mukherjee el al. which suggests that they use a different and more distinguishable dataset.

Figure 1: Word cloud distributions for real reviews on the left and fake reviews on the right.

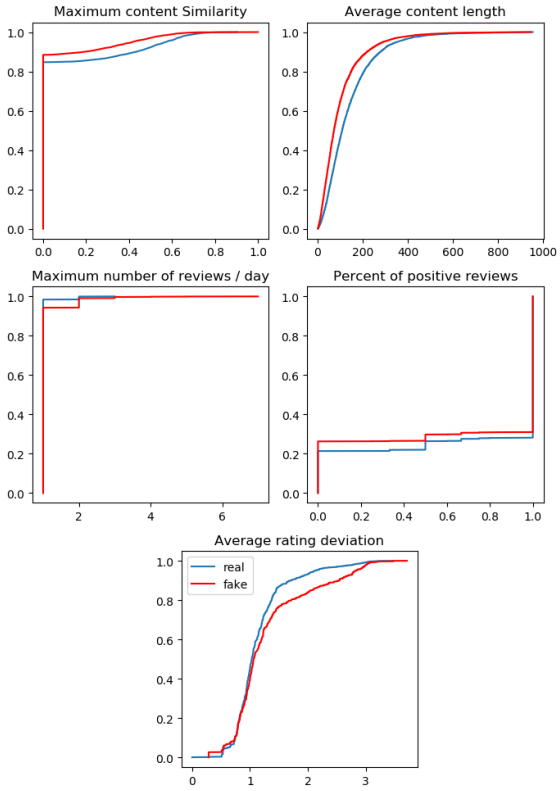


Figure 2: CDF plots of different behavioral features

5 Method

We propose two primary avenues of exploration: Classification and Generation. Although different, each of these avenues share distinct features and build off of each other. We present an assortment of Machine Learning methods in order to explore these domains.

5.1 Classification

We evaluate the performance of several different learning algorithms of increasing complexity. As a baseline we compare Multinomial Naive Bayes and logistic regression.

Multinomial Bayes: We use a unigram language model to model $p(x_1, x_2, ..., x_n|y)$ as $\prod_{i=1}^{n} p(x_i|y)$ using the Naive Bayes assumption of conditional independence given y. We then train our model to maximize the joint likelihood:

$$\mathcal{L}(\phi_y, \phi_{j|y=1}, \phi_{j|y=0}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)})$$

Namely, we estimate the prior distribution of y ϕ_y, the probability of seeing a word j given a review is fake $\phi_{j|y=1}$,and similarly, the probability of seeing a word j given a review is real $\phi_{j|y=0}$.

Logistic Regression: In logistic regression we look to learn $P(y = 1|x)$, that is the probability that the review is fake given the review contents. We represent the reviews using sklearn's CountVectorizer to produce word counts vectors. We then train using sklearn's Logisitc Regression implementation which performs coordinate ascent using L2 regularization and the limited-memory BFGS optimizer, which uses a compressed version of the inverse hessian matrix to guide ascent. The objective function we maximize is the following, where $\sigma(x)$ is the sigmoid function, $y^{(i)}$ is the label for the review $x^{(i)}$, and our regularization term $\lambda = 1$.

$$J(\mathbf{w}) = \sum_{i=1}^{n} [y^{(i)} \log(\sigma(x^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(x^{(i)}))] + \frac{\lambda}{2} ||\mathbf{w}||_2^2$$

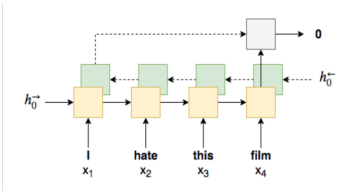
SVM: We use sklearn's SGDClassifier with hinge loss and L2 regularization, which is equivalent to a linear SVM using stochastic gradient descent. We use a linear kernel so that we can use stochastic gradient descent and make training feasible. Our feature vector is a counts vector of the top 1000 words across the training set. The objective of SVM is to create a hyperplane that separates true and false examples by minimizing the following objective function.

$$\min_{w,b} \frac{1}{2} ||w||^2$$
$$s.t. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, ..., m$$

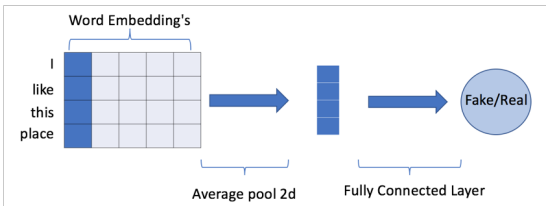
Random Forest: A Random Forest is an ensemble methods that creates a forest of binary decision trees of certain depth and outputs the mode of the decision of those trees. Each tree is trained on a random subset of the input features and tries to classify by minimizing, in our case, the gini loss at each split. We used sklearn's RandomForestClassifier to classify the reviews using behavioral data and word count vectors (using sklearn's CountVectorizer). We used 100 decision trees of depth 2.

LSTM Neural Network: Recurrent neural network (RNN) models have emerged as very powerful auto-regressive models for sequential data such as text. At a high level, recurrent neural networks recursively process information in a sequential patterns, allowing for previous information to inform the present and future. RNNs look to model data, $P(x) = P(x_1)P(x_2|x_1)...p(x_t|x_{t-1})$, using the chain rule of probability, where the network implicitly models the conditioning variables as the current hidden state h_t of the network. An LSTM is a special type of RNN that introduces additional structure to allow for the learning of long-term dependencies. The LSTM network introduces an additional memory term c_t along with "gates" - (function operations) to forget and remember certain important information.

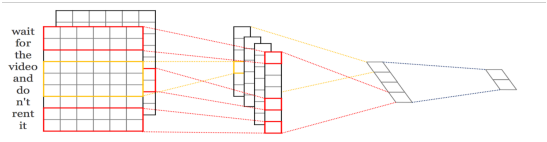
We use a 2 layer, bidirectional LSTM with 512 hidden units in each LSTM cell. We initially encode input tokens with corresponding Glove 100d vectors and then allow for further training of the embedding layer[2]. For classification, we want to model $P(y = 1|x_1, x_2, ..., x_t)$. We thus treat the final forward and backward hidden units h_{tf}, h_{tB} as encodings of the data and pass the concatenated vector $[h_{tf}, h_{tb}]$ through a linear output layer with sigmoid activation. During training we then optimize BCELoss (as described above for logistic regression) with an Adam optimizer and utilize dropout = 0.5 to control against over fitting.



FastText: FastText is a rather simple but powerful deep model developed by Joulin et al at Facebook research [5]. It manages to reach state-of-the-art text classification accuracy with very fast training time. FastText is comprised of a word embedding layer (initialized with glove-100d), a 2d average pool layer, and a fully connected layer. The output is then passed through a sigmoid function and rounded to compute the predicted class. Intuitively, this model averages the word embeddings of a review to get a document vector that can be passed through a fully connected layer and then trained with BCELoss.



CNN: We also apply apply a CNN based model for text classification based on results published by Yoon Kim [6]. The CNN, like FastText, takes in a 2d grid of words and embeddings (intialized with glove-100d). We then apply different sized 2d convolutional filters over the embedding matrix. We use convolutional filters of size $[c \times 100]$, where we vary height c of the filters as $c = [3, 4, 5]$ and the width is the length of the embedding vectors. Max pooling is then applied to the output of each convolutional filter and the resulting single unit blocks are concatenated. Lastly, we apply a fully connected layer to this feature vector and pass the output through the sigmoid activation function. Intuitively, the height of each convolutional filter tries to capture information about n-grams in the text, where n = filter height. We train our model with BCELoss with dropout = 0.5 to reduce overfitting.



5.2 Generation

In order to generate context specific restaurant reviews we take inspiration from an approach explored by Juuti et al [2]. This approach uses a sequence-2-sequence model to translate from a given context to the corresponding review. Sequence-2-sequence models have shown great promise in learning how to translate from a given source to a corresponding target, with application to domains such as image captioning and machine translation. As

described above, the inputs / context for our model is comprised of a sequence of descriptor tokens for a review - [rating, restaurant name, city, state, restaurant tags] - and the output / target is the matching review. We use the open source sequence-2-sequence model openNMT-py [2].

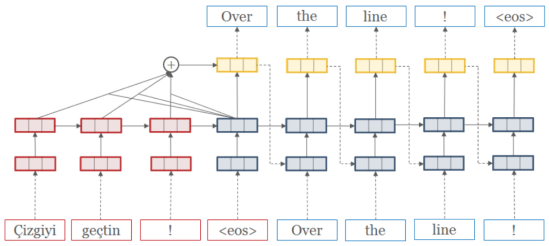


Figure 3: Sequence-2-Sequence model architecture

The openNMT-py model allows for great flexibility in defining architecture parameters, optimization algorithms, and other training hyper-parameters. As shown in figure 3, the sequence-2-sequence model includes an encoding RNN network that encodes the source sequence into a compressed vector representation that is then fed into a decoder RNN network, which translates the encoding into the target sequence. Given a source C, the encoding network maps C to a vector representation that is passed to the decoding network. The decoding network then looks to represent $P(R|C)$ where R is the corresponding review for the source C. The decoder models the probability distribution over the each output token r_t. Namely, the decoder defines a distribution over all possible output tokens for a given time step t, then predicts the corresponding token using the log softmax function (gives log probability for each token). Thus the model defines the probability of an output sequence using the chain rule as follows:

$$P(R|C) = \prod_{t=0}^{n} p(r^t|X, r^1, ..., r^{t-1})$$

During training we minimize the negative log likelihood of the translated phrase. Thus, our loss function consists of the combined log softmax loss over the output distribution for each token $\hat{r^t}$, where r^t is a one hot encoding for the target token:

$$L(\hat{r^t}, r^t) = -\sum_i r_i^t log(\hat{r^t}_i) \quad (1)$$

$$L(R) = -\sum_t L(\hat{r^t}, r^t) \quad (2)$$

One additional component to the model is an attention layer, which looks to solve the challenge of trying to represent the entire source information by a single fixed-length vector. This attention layer allows the decoder to have a more global reference of the source context, as each output unit is connected to all the outputs of the source network and learns to weight the importance of each source output.

Because the neural decoder model looks to generate an output that maximizes the log probability given a particular source, a long recognized problem in neural generation models is lack of diversity in generation[2] [3]. As we show later, generated reviews tend to follow very contrived formulations with little creativity. In an attempt to encourage diversity, we implement several techniques explored by [] to penalize repetition and encourage non-trivial generation. Specifically, we penalize certain words at each step of the generative process by directly perturbing the log probability outputs of the decoder model. As motivated by [] we introduce three penalty parameters α, λ, b. Given the distribution $logp$ over a the output vocabulary we modify $logp$ as follows. Firstly, we sample a (0, 1) value from a Bernoulli distribution parameterized by b for each vocab term. Then, for each term that had a one sampled we apply a penalize $logp_i$ as $logp_i = logp_i + \lambda * \alpha^t$, where t represents how far we are into the sentence. We can interpret this penalty as a factor encouraging the generation to randomly explore novel completions, with increased emphasis at the start of the sentence. The second penalty term we add is for each candidate word that we have generated previously we add a penalty of λ to encourage our geneartion to avoid repeating previous words.

6 Results

6.1 Evaluation

We evaluate our results using a variety of metrics. The two main sources of numerical evaluation for classification success that we focus on are accuracy and f-score, where we use f-score as a proxy for evaluating both precision and recall. For the simpler algorithms, we use 5 fold cross validation and present the averaged results. We also provide the confusion matrix for the best model, Naive-Bayes, as the sum of the confusion matrices for each fold to get the result across the entire set.

For our more complex models, we used a .7/.15/.15 train-valid-test random split. We used the validation set to tune hyperparameters. Results from the validation set also helped us notice that we should use early stopping and dropout to prevent overfitting. The results we present below are from our test set.

6.2 Classification results

Model	Accuracy	F Score
Naive-Bayes (words)	0.665	0.595
Naive-Bayes (words + behaviors)	0.678	0.676
Logistic Regression (words)	0.660	0.660
Logistic Regression (words + behaviors)	0.630	0.630
Random Forests (behaviors)	0.650	0.645
Random Forests (words + behaviors)	0.620	0.618
FastText (words)	0.680	0.649
CNN (words)	0.688	0.628
LSTM (words)	0.644	0.617

Table 1: Average test accuracy, precision, recall and F Score percentage

	real pred	fake pred
real label	4793	3348
fake label	1968	6173

Table 2: Confusion matrix for Naive-Bayes using word and behavior features

Overall our classification results were less than we were hoping when compared with other previous work done in the area. However, our results corresponded with our feature analysis wherein we showed that both the word distributions and the behavior feature distributions between the real and fake reviews were very similar. Given that our human baselines only averaged around

50% on the same dataset (i.e. random guessing) we were impressed to see better results. Although we were not able to replicate the results of related works in the field, we showed in our feature analysis that these related works had fundamentally more amicable datasets.

We also interestingly observed similar performance across all of our models. We hypothesized that this was due to an inherent upper boundary on how well the reviews could be distinguished in our dataset. Further analysis showed that the different algorithms produced similar predictions on the same test dataset. For example the predictions of Naive Bayes with word and behavior features was 92.38% similar to Naive Bayes with just words. However, comparing word feature Naive Bayes and word feature Logistic Regression we saw a prediction similarity of around 77.86%. This suggests that maybe some ensemble of the two models could perform better. We implemented a basic ensemble voting classifier that would take the argmax of sum over the probabilities outputted by both models and didn't see any better results. Further evidence for the existence of an upper bound to classification based on word and behavioral features comes from Table 2, where we see that we produce a significant amount of fake positive classifications. This suggests that there are significant similarities in the word and behavioral distributions of real and fake reviews, which cause many real reviews to be misclassified as positive.

6.3 Generation Results

To evaluate generated results we combine qualitative and quantitative metrics. Firstly, look qualitatively at the results, as a human expert is capable of evaluating many characteristics of what makes a review realistic, such as grammar, fluidity, and believability. We compare the results of the neural translation model with and without any diversity penalty. We select penalizing parameters: $b = 0.6, \lambda = -4, \alpha = 0.66$ and also compare the log probability of the review.

Context: [4, Yellow, Door, Bistro, Calgary, AB, American, (New), Bistros, Restaurants, Breakfast, 'Brunch', 'Canadian']

No Penalty: The food was great. The service was very good. I would definitely come back.

Log(r|c) = -22.6364

With penalty: Had brunch here with my wife. Food was great. Service was a little slow, but overall a great experience.

Log(r|c) = -34.4819

Context: [1, Benjarong, Authentic, Thai, Cuisine, Las, Vegas, NV, Thai, Restaurants]

No Penalty: Worst Thai food I've ever had!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Log(r|c) = -23.9030

With penalty: Not good at all. I ordered pad thai and it tasted like rubber. I will never go back.

Log(r|c) = -32.1670

Qualitatively we see that the penalized reviews are much more diverse and also do a better job avoiding falling into the trap of repeating the same token many times (i.e. the '!'). Additionally, we can quantitatively see that our penalization approach has affected the generation by pushing the model to generate reviews with lower overall log likelihood but greater resulting linguistic creativity.

After qualitatively inspecting the generated reviews we wanted to compare our trained classifiers ability to classify our generated reviews. We generate 5000 reviews using our penalized and non-penalized models. Then we feed them as input to the Naive Bayes classifier trained on the Yelp Fake Review dataset (i.e. not trained of any computer generated reviews). Somewhat surprisingly we obtain the following results.

Type	Accuracy
No Penalty	0.9836
Penalty	0.9154

On one hand we see that our classification model is able to obtain quite high accuracy despite never being exposed to computer generated reviews. On the other hand, we see that the reviews generated with diversity penalty do appear to be more realistic as they are harder for the classifier to identify correctly. Therefore, these results present promise for both domains!

7 Conclusion and Future Work

7.1 Conclusion

The goal of our work was to apply and compare modern classification algorithms on the popular problem of detecting fake Yelp reviews. Preliminary research showed that others had found promising results from text analysis using relatively simple model and there remained roomed for improvement through the use of cutting-edge techniques. However, we found that results were relatively uniform across most models and furthermore, we weren't able to replicate many results produced by other groups due to differences in datasets. We found that our CNN produced the highest accuracy but we think that our most successful model was Naive-Bayes using both words and behavioral features, which reached an F Score of 0.676. The relatively low accuracies of our models demonstrates the challenge nature of this problem especially treated as a purely language based classification problem. Nevertheless, based on a human baseline of roughly 50% our models are able to learn some representation of the two classes.

We were also very impressed by the results of our review generator. Given a particular context we were able to generated fairly believable restaurant reviews that specifically reflect the context provided. Moreover, we demonstrate success in increasing the diversity of generated reviews by penalizing parts of the generation process. Given generated reviews we were also able to test the generalizability of our classification models. Tested on our most successful model, Naive Bayes, we achieve very high accuracy in detecting the computer generated reviews. Although this demonstrates a need for improvement in our generative model, we do see that the classification model is less accurate when detecting the penalized reviews signalling a promising direction of research for further improvement.

7.2 Future Work

One thing we did not have a chance to explore more were various ensemble methods. Since there seemed to be some difference in the classifications produced by different models perhaps model ensemble methods could be trained to produce more impactful results. In particular, it would be interesting to combine the outputs of one of our implemented deep model with a Neural Net trained on behavior features. Our dataset was also limited especially in terms of metadata detail, which we think could provide major improvements for behavioral feature classifications.

8 Contributions

We all contributed an equally large amount of work.

References

[1] Yao, Yuanshun. Automated Crowdturfing Attacks and Defenses in Online Review Systems in arXiv, 2017
[2] Juuti, Mika, et al. "Stay On-Topic: Generating Context-specific Fake Restaurant Reviews." arXiv preprint arXiv:1805.02400 (2018).
[3] Li, Jiwei, Will Monroe, and Dan Jurafsky. "A simple, fast diverse decoding algorithm for neural generation." arXiv preprint arXiv:1611.08562 (2016).
[4] Wang, Xinyue, et al. "Identification of fake reviews using semantic and behavioral features." 2018 4th International Conference on Information Management (ICIM). IEEE, 2018.
[5] Joulin, Armand, et al. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).
[6] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).
[7] Mukherjee, A., Venkataraman, V., Liu, B. and Glance, N. 2013. What Yelp Fake Review Filter might be Doing? ICWSM. (2013).
[8] Luca, Michael. "Reviews, reputation, and revenue: The case of Yelp. com." (2016).
[9] https://www.yelp.com/dataset/challenge
[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation
[11] https://github.com/N-Demir/Yelp-reviews