Final Project Report: Fake Reviews Detection (Group 16)

Visaj Nirav Shah
ID: 1225369210
Arizona State University
Tempe, USA
vshah47@asu.edu

Hastin Himanshubhai Modi
ID: 1225543982
Arizona State University
Tempe, USA
hmodi5@asu.edu

Mohil Devan Khimani
ID: 1225636802
Arizona State University
Tempe, USA
mkhimani@asu.edu

Shivani Shailesh Nandani
ID: 1225446014
Arizona State University
Tempe, USA
snandani@asu.edu

Shivam Milind Bodiwala
ID: 1225418324
Arizona State University
Tempe, USA
sbodiwal@asu.edu

Rahil Ashish Shah
ID: 1225475355
Arizona State University
Tempe, USA
rshah72@asu.edu

***Abstract*—The prevalence of fake reviews poses a significant challenge for online platforms that rely on reviews to inform consumer decisions and for consumers who rely on the opinions of other users to make decisions regarding buying a product. To address this problem, various machine learning and natural language processing techniques have been developed to automatically detect fake reviews. This project compares and analyzes various supervised fake review detection techniques, including traditional feature-based methods and modern machine learning-based approaches, such as deep neural networks. 2 types of datasets - one where the fake reviews are generated using GPT-2 [1] [2] and another where they are generated using back translation (English - French - English) - are used. A comparative analysis of various techniques involving traditional and modern machine learning methods such as deep neural networks has been made and their strengths and limitations are discussed in detecting fake reviews. Logistic Regression and CNN + Attention are some of the top performers among the variety of architectures experimented with. A user interface that accepts input and displays the model prediction is also developed.**

***Index Terms*—Fake review, Deep neural networks, Classification, Back translation, GPT-2**

I. Introduction

Online platforms have become a popular medium for customers to share their experiences with products and services, and online reviews play a crucial role in influencing a customer's purchasing decisions. Deceptive opinion spam, also known as fake reviews, can be frequently encountered on such online platforms. These fraudulent reviews are typically created by means of fake accounts, bots, or paid individuals. The dishonest and misleading information presented in such reviews can misguide customers. This project focuses on developing a system to detect fake reviews mainly developed by bots that are prevalent on online platforms. Fake reviews can lead to inappropriate purchasing decisions for customers, which highlights the importance of developing an automated system to identify and flag these reviews. To achieve this goal, we have used machine learning techniques and natural language processing to analyze reviews and identify any suspicious activities that could indicate a fake review. The system architecture includes a variety of supervised fake review detection techniques, including traditional feature-based methods and modern machine learning-based approaches, such as deep neural networks. In our project, we have used 12 algorithms for training and testing on the datasets, including Logistic Regression [3], K-nearest neighbors [4], Support Vector Classifier [5], Naive Bayes, Decision Tree [6], Random Forests [7], AdaBoost [8], XGBoost [9], Bidirectional Encoder Representations from Transformers (BERT) [10], Convolutional Neural Network (CNN) [11], CNN + Attention, and Bidirectional Long Short-Term Memory (BiLSTM) [12].

Additionally, we have used two types of datasets – one generated using GPT-2 (referred to as old data) and another generated using back-translation (referred to as new data). The datasets include features such as text, category, rating, sentiment, and word count categories. The evaluation of the project is based on metrics such as accuracy, recall, and speed in detecting fake reviews.

The development of our system can contribute to the online platform's integrity and provide reliable reviews for customers to make informed decisions. The user interface that accepts input and displays the model prediction can be an added advantage for users to verify the authenticity of the reviews.

Overall, our project report aims to compare and analyze various supervised fake review detection techniques and evaluate their strengths and limitations in detecting fake reviews. The project's findings can contribute to developing effective solutions to detect fake reviews and ensure customer trust and satisfaction.

II. Problem Statement

In today's world, online reviews have become a crucial factor in a customer's decision-making process. However, the rise in the influence of user reviews has led to an increase in the prevalence of fake reviews, also known as deceptive opinion spam. These fake reviews are commonly found on various online platforms, such as e-commerce websites, travel booking sites, and social media platforms, and are generated by fake accounts, bots, or even paid individuals. The dishonest and misleading information provided in these fake reviews can result in inappropriate purchasing decisions for customers. Therefore, the development of an automated system that can accurately identify and flag fake reviews is crucial. Traditional methods of detecting fake reviews, such as manual screening or crowdsourcing, can be time-consuming and inefficient. In contrast, an automated system can analyze a large volume of reviews in a short amount of time and flag any suspicious activity. This project aims to achieve this goal by using machine learning techniques and Natural Language Processing (NLP) to analyze reviews and identify any suspicious activities that could indicate a fake review. By combining effective features with the best-performing machine learning algorithm, the proposed system will aim to achieve high accuracy, recall, and speed in detecting fake reviews. The evaluation of the project will be based on these metrics to ensure its effectiveness in detecting fake reviews.

III. Related Works

To get an overview of the task at hand, Mohawesh et al. [13] have performed an exhaustive survey for all steps from start to finish. This research paper aims to understand the problem in depth by analyzing the task of fake review detection, including existing datasets and their collection methods, feature extraction techniques, and traditional statistical machine learning and deep learning methods. Additionally, the study benchmarks the performance of different neural network models and transformers for fake review detection. The paper also identifies gaps in current research and suggests possible future directions for this field.

Detecting fake reviews automatically has proven difficult, and Salminen et al. [2] explore both the creation and detection of fake reviews. In this paper, fake reviews are generated using two language models, Universal Language Model Fine-Tuning (ULMFiT) [14] and Generative Pre-training Transformer-2 (GPT-2) [1], based on an Amazon e-commerce dataset. The study demonstrates that machine classifiers can effectively detect fake reviews with near-perfect accuracy, whereas human raters exhibit significantly lower accuracy and agreement in detecting fake reviews. This approach is also effective in detecting human-generated fake reviews. The fine-tuned Robustly Optimized Bidirectional Encoder Representations from Transformers Approach (RoBERTa) [15] built by the authors boasts an impressive precision and recall of 0.97. The findings have important implications for consumer protection, defense against unfair competition, and the responsibility of review platforms.

When it comes to building fake review detection systems, there are market leaders like Yelp who have built such exemplary systems. Mukherjee et al. [16] have tried to understand the backend systems guiding Yelp's fake spam detection. In this research, they analyzed Yelp's filtered reviews to understand how they might be detecting fake reviews. The focus is on a supervised learning approach, using Yelp's filtered reviews for training. Existing approaches based on supervised learning are based on pseudo-fake reviews rather than real fake reviews filtered by a commercial website. The performance evaluation is done based on existing research methods on real-life Yelp data and finds that behavioral features perform well while linguistic features are less effective. A novel information theoretic analysis to understand the precise psycholinguistic difference between crowdsourced fake reviews (generated using Amazon Mechanical Turk) and commercial fake reviews (filtered by Yelp) is proposed. After performing an experimental study, the authors achieved the highest accuracy of 86.1% in classifying Restaurant reviews and 84.8% in Hotel reviews. The analysis and experimental results suggest that Yelp's filtering is reasonable and its algorithm is correlated with abnormal spamming behaviors. These findings can be useful to other such systems in their filtering efforts.

Sentiment analysis is a powerful tool for learning the polarity of a user's opinion. For example, text-based tweets can be categorized into positive, negative, and neutral. Dang et al. [17] studied various methods researchers and industry experts used for this task. Since the two most popular methods of statistically representing words are Term Frequency-Inverse Document Frequency (TF-IDF) and word embedding, this paper attempts to study the variation of classification metrics (accuracy, recall, precision, F1-score, and Area Under Curve (AUC)) for deep learning methods for both these representation techniques. The study was conducted for eight different datasets, and it was observed that Recurrent Neural Network (RNN) [18] models with word embedding performed best for a majority of these datasets. The paper attributes this to the ability of word embeddings to capture some level of context, while traditional approaches like TF-IDF are incapable of the same.

RNNs are popular deep-learning methods for modeling sequential data. However, vanilla RNN suffers from the problem of vanishing gradients, where the gradients (used for parameter update) become smaller and smaller, leading to insignificant parameter updates, and, thus, no real learning is done. Long short-term memory (LSTM) [19], introduced by Hochreiter and Schmidhuber in 1997, deals with the vanishing gradient problem by introducing additional gates. The input, output, and forget gates allow the gradients to flow through the network without vanishing as quickly.

BERT [10], or Bidirectional Encoder Representations from Transformers, is another popular algorithm for natural language processing applications. BERT improves on standard transformers by pre-training deep bidirectional representations from unlabeled text by jointly conditioning on both the left and right context in all layers. As a result, the pre-trained model can be fine-tuned for each downstream task without substantial task-specific architecture modifications.

Convolutional neural networks (CNNs) [11] have traditionally been used for image processing and image classification tasks. However, recent research has also shown its usefulness in the NLP domain. In image processing, convolutional filters capture different features of an image; a deeper filter is more likely to learn more complex features. For NLP tasks, such as sentence classification [20], the text is represented as an array of vectors (each word is mapped to a specific vector depending on the choice of parameterization technique used). However, in this case, one-dimensional convolutions are calculated. This allows us to pick patterns in a text sequence in a way similar to feature extraction in image processing.

IV. System Architecture and Algorithms

We have used a total of 12 algorithms for training and testing on our datasets. Each algorithm has been trained and tested on 6 combinations of data:

1) Old data (fake using GPT-2)
   a) Features - Text, Category, Rating, Word count categories
   b) Features - Text, Category, Sentiment, Word count categories
   c) Features - Text
2) New data (fake using back-translation)
   a) Features - Text, Category, Rating, Word count categories
   b) Features - Text, Category, Sentiment, Word count categories
   c) Features - Text

The algorithms which were used in this project include:

- Traditional algorithms - Logistic Regression, K-nearest neighbors, Support Vector Classifier, Naive Bayes, Decision Tree, Random Forests, AdaBoost, and XGBoost
- Neural networks - Bidirectional Encoder Representations from Transformers (BERT), Convolutional Neural Network (CNN), CNN + Attention, Bidirectional long short-term memory (BiLSTM)

The code for each of these algorithms is present on the GitHub repo of the project.[1]. The traditional algorithms were implemented using the scikit-learn library [21] while the neural network algorithms were implemented using TensorFlow [22] and Keras [23].

For each of the traditional algorithms, the best hyperparameters are found using the *RandomizedSearchCV* function in the scikit-learn library. Given below are the details of the architectures of all the algorithms:

A. Logistic Regression

Logistic regression is a statistical algorithm used for binary classification problems, where the goal is to predict the probability of an event occurring (e.g., whether a customer will churn or not, whether an email is spam or not, etc.). The output of logistic regression is a probability value between 0 and 1, which can be converted to a binary prediction by applying a threshold.

The logistic regression model is trained using a supervised learning approach called maximum likelihood estimation. The goal of this approach is to find the set of weights that maximizes the likelihood of the observed data given the model. This is typically achieved using iterative optimization algorithms such as gradient descent.

There were 3 hyperparameters which were needed to be set for this algorithm:

- C: This hyperparameter controls the regularization strength of the model. The regularization term is added to the cost function of the logistic regression model in order to prevent overfitting by penalizing large weights.
- Penalty: This hyperparameter specifies the type of regularization to be used in the logistic regression algorithm. Regularization is a technique used to prevent overfitting in machine learning models by adding a penalty term to the objective function that the model tries to minimize during training. The penalty hyperparameter can take on two values: 'l1' and 'l2'. If penalty='l1', the logistic regression algorithm uses L1 regularization, also known as Lasso regularization, which adds a penalty term proportional to the absolute value of the weights. If penalty='l2', the logistic regression algorithm uses L2 regularization, also known as Ridge regularization, which adds a penalty term proportional to the square of the weights.
- Solver: This hyperparameter specifies the optimization algorithm to be used to solve the logistic regression problem. The choice of optimization algorithm can affect the convergence speed, scalability, and accuracy of the model.

B. K-nearest neighbors

k-Nearest Neighbors (k-NN) is a non-parametric machine learning algorithm used for both classification and regression problems. It is a lazy learning algorithm, meaning that it does not make any assumptions about the underlying data distribution but instead learns the model from the training data at the time of prediction.

The k-NN algorithm works by finding the k nearest neighbors to a given test point in the training data, based on a distance metric such as Euclidean distance or Manhattan distance. The value of k is a hyperparameter that must be specified in advance. For a given test point, the k-NN algorithm assigns the class label (in the case of classification) or the output value (in the case of regression) that is most common among its k nearest neighbors.

There were 2 hyperparameters which were needed to be set for this algorithm:

1) n_neighbors: This hyperparameter specifies the number of nearest neighbors to consider when making a prediction for a new data point. It is a positive integer that needs to be set by the user. It should be chosen based on the complexity of the problem and the size of the dataset.

[1] https://github.com/hastinmodi/SWM_Project_Fake_Review_Detection

2) algorithm: This hyperparameter specifies the algorithm used to compute the nearest neighbors. The algorithm hyperparameter can take on three values: 'brute', 'kd_tree', or 'ball_tree'. The default value is 'auto', which means that the model automatically chooses the most appropriate algorithm based on the input data.
    a) The 'brute' algorithm is the simplest and most straightforward approach, which computes the distances between all pairs of points in the dataset.
    b) The 'kd_tree' algorithm uses a tree data structure to store the training data and quickly compute the nearest neighbors for a new data point.
    c) The 'ball_tree' algorithm also uses a tree data structure, but with a different approach to partitioning the data.

C. Support Vector Classifier

Support Vector Classifier (SVC) is a type of supervised learning algorithm in machine learning that can be used for classification tasks. It is based on the concept of Support Vector Machines (SVMs), which are a family of algorithms that seek to find the optimal hyperplane that separates two classes in a high-dimensional space.

The idea behind SVC is to find the hyperplane that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest points of each class.

There were 3 hyperparameters which were needed to be set for this algorithm:

- kernel: This hyperparameter specifies the kernel function used for transforming the input data to a higher-dimensional space. The kernel function is used to find the decision boundary (hyperplane) between the classes.
- gamma: This hyperparameter controls the shape of the decision boundary and the influence of each training example. It determines the reach of the individual training examples, with low values meaning a larger reach and high values meaning a smaller reach.
- C: This hyperparameter controls the trade-off between achieving a low training error and a low testing error by adjusting the penalty for misclassifying training examples. It determines the level of regularization in the model, with smaller values of C causing more regularization and larger values of C causing less regularization.

D. Naive Bayes

Naive Bayes is a family of simple but powerful probabilistic classifiers based on the Bayes' theorem. It is called "naive" because it assumes that the features in the dataset are conditionally independent of each other given the class label.

In Naive Bayes, each feature is modeled using a probability distribution, such as a Gaussian, multinomial, or Bernoulli distribution, depending on the type of data being modeled.

There are three main types of Naive Bayes classifiers: Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes, each with their own assumptions about the distribution of the features.

There was 1 hyperparameter which was needed to be set for this algorithm:

- alpha: This hyperparameter is a smoothing parameter that controls the balance between the estimated probabilities based on the training data and the prior probabilities of the classes. When calculating the conditional probability of a feature given a class, the Multinomial Naive Bayes algorithm uses the count of the feature in the training set, and adds alpha to it. This helps avoid zero probabilities when a feature is not present in the training set for a particular class.

E. Decision Tree

Decision tree classifier is a popular and powerful supervised machine learning algorithm used for classification tasks. It is constructed by partitioning the data into subsets based on the values of the features. At each node in the tree, a decision is made based on the value of a feature, and the data is split into two or more subsets based on that decision. This process is repeated recursively until a stopping criterion is met, such as a minimum number of samples per leaf node, or a maximum depth of the tree.

There were 6 hyperparameters which were needed to be set for this algorithm:

1) criterion: This hyperparameter specifies the function to measure the quality of a split. The two available options for the criterion hyperparameter in Scikit-learn are gini and entropy.
    a) gini: This criterion measures the impurity of a node by computing the Gini impurity, which is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.
    b) entropy: This criterion measures the impurity of a node by computing the entropy of the labels, which is a measure of the disorder or uncertainty in the subset.
2) max_depth: This hyperparameter specifies the maximum depth of the decision tree. The depth of a decision tree is defined as the number of nodes from the root to the farthest leaf node.
3) max_features: This hyperparameter specifies the maximum number of features to consider when looking for the best split. When constructing a decision tree, the algorithm evaluates each feature to determine which one best separates the data into classes.
4) min_samples_leaf: This hyperparameter specifies the minimum number of samples required to be at a leaf node. The decision tree algorithm continues to split the data at each node until all leaves contain a minimum number of samples.
5) min_samples_split: This hyperparameter specifies the minimum number of samples required to split an internal

node. The decision tree algorithm works by recursively splitting the data into subsets based on the values of different features until it creates a tree structure that can accurately predict the target variable.

6) splitter: This hyperparameter specifies the strategy used to choose the feature to split on at each node.There are two possible values for splitter:
    a) best: This strategy chooses the best feature to split on based on a criterion such as entropy or Gini impurity.
    b) random: This strategy chooses a random feature to split on at each node.

F. Random Forests

Random Forest Classifier is a type of ensemble learning method in which multiple decision trees are trained on different subsets of the training data and the final prediction is made by aggregating the predictions of individual trees.

Random Forest Classifier works by creating a set of decision trees, each of which is trained on a randomly sampled subset of the training data and a randomly selected subset of the features.

There were 6 hyperparameters which were needed to be set for this algorithm:

- criterion: Same as Decision Tree
- max_depth: Same as Decision Tree
- max_features: Same as Decision Tree
- min_samples_leaf: Same as Decision Tree
- min_samples_split: Same as Decision Tree
- n_estimators: This hyperparameter specifies the number of decision trees to include in the forest. Increasing the value of n_estimators can lead to a better performance of the model, up to a certain point, as it will allow the forest to capture more complex relationships in the data and reduce the effects of randomness.

G. AdaBoost

AdaBoost (Adaptive Boosting) is a machine learning algorithm that is used for classification problems. The basic idea behind AdaBoost is to combine multiple weak classifiers to create a strong classifier. A weak classifier is a classifier that performs only slightly better than random guessing, for example, a decision tree with only a few nodes. AdaBoost sequentially trains a series of weak classifiers on the training data, with each subsequent classifier giving more weight to the misclassified samples from the previous classifier.

There were 2 hyperparameters which were needed to be set for this algorithm.

- learning_rate: This hyperparameter controls the contribution of each weak classifier to the final ensemble.
- n_estimators: Same as Random Forests

H. XGBoost

XGBoost (Extreme Gradient Boosting) is a popular open-source library for gradient boosting, a technique for building ensemble models that combines multiple weak learners to create a stronger predictor. XGBoost is designed to be highly scalable, efficient, and flexible. In XGBoost, each weak learner is a decision tree, and the goal is to iteratively add trees to the ensemble to minimize a loss function. The loss function measures the error between the predicted and actual values of the target variable and guides the learning process.

There were 5 hyperparameters which were needed to be set for this algorithm:

- colsample_bytree: This hyperparameter in XGBoost that controls the fraction of features (columns) to randomly sample at each tree node when building each decision tree in the ensemble.
- learning_rate: This hyperparameter in XGBoost controls the step size used during the boosting process. It is a scalar value between 0 and 1 that determines how aggressively the boosting algorithm tries to correct the errors of the previous model in the ensemble.
- max_depth: This hyperparameter in XGBoost specifies the maximum depth of a tree in the ensemble. It is an integer value that controls the complexity of the trees and determines how deeply each tree is allowed to grow during the boosting process.
- n_estimators: This hyperparameter in XGBoost specifies the number of trees in the ensemble. It is an integer value that controls the number of boosting rounds or iterations, which in turn determines the number of trees to be built.
- subsample: This hyperparameter in XGBoost controls the fraction of the training data that is randomly sampled to train each tree in the ensemble. It is a float value that ranges between 0 and 1. A value of 1.0 means that all the training data is used to train each tree, while a value less than 1.0 means that only a fraction of the training data is used.

I. BERT

BERT is a powerful pre-trained language model developed by Google. It is capable of generating contextualized word embeddings. A BERT-based classifier is a machine learning model that uses a pre-trained BERT model as a feature extractor and then trains a classifier on top of the extracted features to perform a specific task. In BERT-based classification, the input text is first tokenized into subwords, which are then fed into the pre-trained BERT model to generate contextualized word embeddings. These embeddings are then used as features to train a classifier on a specific downstream task.

For all our 6 datasets, we have used a BERT layer (with 12 encoder layers) to extract the word embeddings and then passed it through 2 dense layers containing 256 and 128 neurons respectively, after which we finally receive our output.

Binary crossentropy loss has been used for the loss function along with the Adam optimizer [24] with a learning rate of 0.0001. The batch size is kept as 64 and the dropout rate is 0.2.

J. CNN

A CNN classifier is a type of deep learning model commonly used for image classification tasks. The basic building block of a CNN is the convolutional layer, which applies a set of filters to the input image to extract features that are relevant to the classification task.

In addition to convolutional layers, a CNN classifier typically includes pooling layers, which reduce the dimensionality of the feature maps produced by the convolutional layers, and fully connected layers, which use the features extracted by the convolutional layers to make the final classification decision.

For all our 6 datasets, we have used a RoBERTa large [25] model layer to extract the word embeddings and then passed it through 3 convolutional layers, each having 64 filters and a kernel size of 2, 3, and 4 respectively. The output from the 3rd CNN layer is pooled after which we finally receive the output using the sigmoid activation.

Binary crossentropy loss has been used for the loss function along with the Adam optimizer with a learning rate of 0.001. The batch size is kept as 128 and the dropout rate is 0.2.

K. CNN + Attention

The CNN + Attention model is similar to the CNN model with the only difference being the addition of an Attention layer after the convolutional layers. The Attention [26] layer is a type of layer used in deep learning models that enables the model to focus on the most important parts of the input data when making predictions.

The Attention layer works by computing a set of attention weights for each element in the input sequence, which indicate how important each element is for predicting the output. The attention weights are typically computed based on a combination of the current state of the model and the input sequence. The attention weights are then used to compute a weighted sum of the input sequence, where the elements with higher attention weights contribute more to the final output. This weighted sum is used as input to the next layer in the model. Overall, the Attention layer enables the model to dynamically focus on different parts of the input sequence based on the current state of the model, making it a powerful tool.

For all our 6 datasets, we have used a BERT layer (with 12 encoder layers) to extract the word embeddings and then passed it through 2 convolutional layers, each having 64 filters and a kernel size of 2 and 3 respectively. The output from the 2nd CNN layer is pooled after which we pass it through an Attention layer and then receive the final output using the sigmoid activation.

Binary crossentropy loss has been used for the loss function along with the Adam optimizer with a learning rate of 0.001. The batch size is kept as 64 and the dropout rate is 0.2

L. BiLSTM

The BiLSTM is a type of recurrent neural network (RNN) architecture used in deep learning for sequential data modeling, such as text classification, sentiment analysis, and speech recognition. The BiLSTM combines the forward and backward information of a sequence to capture the context of the data better.

In a standard LSTM, the input sequence is processed in a forward direction from the first element to the last element, and the hidden state of the final element is fed to a dense layer for classification. In contrast, a BiLSTM processes the sequence in both forward and backward directions by introducing a second hidden state that processes the sequence from the last element to the first element. The output of both forward and backward states is then concatenated and fed to a dense layer for classification. The BiLSTM architecture can capture the dependencies between the past and future elements of a sequence, making it more effective for tasks that require context information.

Based on the dataset, the BiLSTM model concatenates different input branches to use in the fully connected layer. The review branch (that contains the text of the review) uses a Bidirectional layer with 64 units and a dense layer with 64 units. All the other input branches have a dense layer with 8 units, a dropout layer, and a dense layer with 16 units. The output of these branches is concatenated and passed through the dense layer with 8 units and used for final prediction.

V. Datasets

In this project, we focus primarily on machine-generated fake reviews. Therefore, we have used two different datasets – one generated using GPT-2 and another generated using back-translation, with equal split between real and fake. Each dataset includes features such as text, category, rating, sentiment, and word count categories. Table I shows the total number of samples and features of each dataset. The original attributes are review text, product category, and rating. For this classification task, we also derive two new attributes – sentiment and word count categories.

Total samples	40433
Original features	3
Derived attributes	2

TABLE I
Dataset description

The following steps were taken to obtain the final dataset for training, testing, and validation of the machine-learning models:

- preprocessing
  - removing extra spaces on the data
  - lemmatization on the data
- deriving new attributes
  - finding word count category based on the ranges observed in the text data histogram (Fig. 1 and Fig. 2). Categories - 0-10, 11-20, 21-50, 51-100, 101-max. length of review in dataset
  - sentiment based on the rating (1 if rating $>$ 3, else 0)

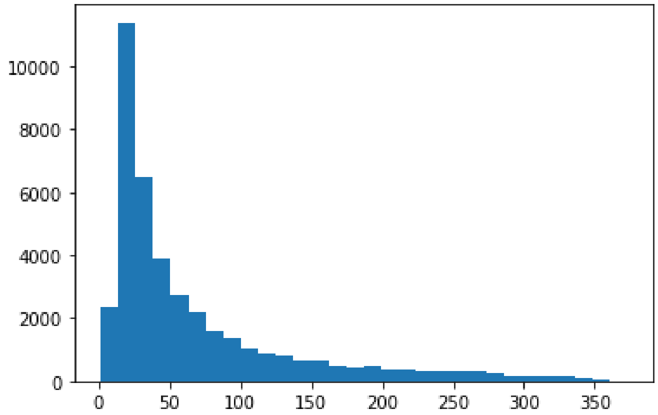


Fig. 1. Word count for original data (GPT-2)

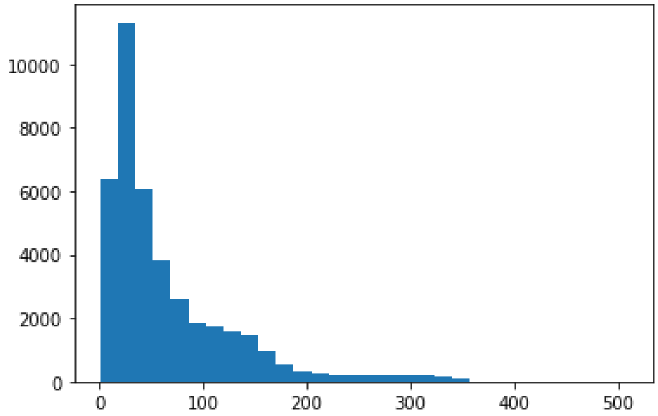


Fig. 2. Word count for new data (back-translation)

- split the data into training (80%), validation (10%), and testing (10%) sets

For advanced machine learning models, we also perform tokenization and padding, and created word embeddings using GloVe/BERT. [27]

Fig (1, 2) visualize the distribution of reviews according to the number of words. Fig (3, 4) are donut plots showing the top 20 frequently-occurring unique words in CG and OR reviews for the new data generated using back translation. One observation that we can make from here is that the CG reviews tend to use many jargons like ”tarpaulin”, ”chronometer”, and ”baguette”, whereas OR reviews written by humans tend to be more simplistic and use words spoken in everyday life.

VI. Evaluations

Given below are the results for all the algorithms implemented by us, based on the best hyperparameters. The best models for the UI were selected based on accuracy and recall since they are the most important metrics when trying to identiy fake reviews. Accuracy is important as our dataset is balanced and models with higher accuracies would tend to classify both real and fake reviews correctly. Recall is important as real detected as fake is acceptable upto some

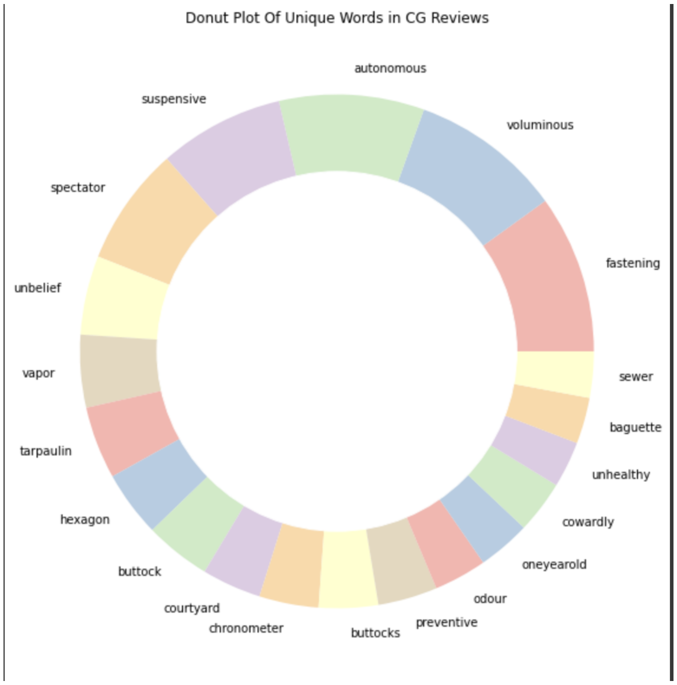


Fig. 3. Donut Plot of Unique Words in CG Reviews

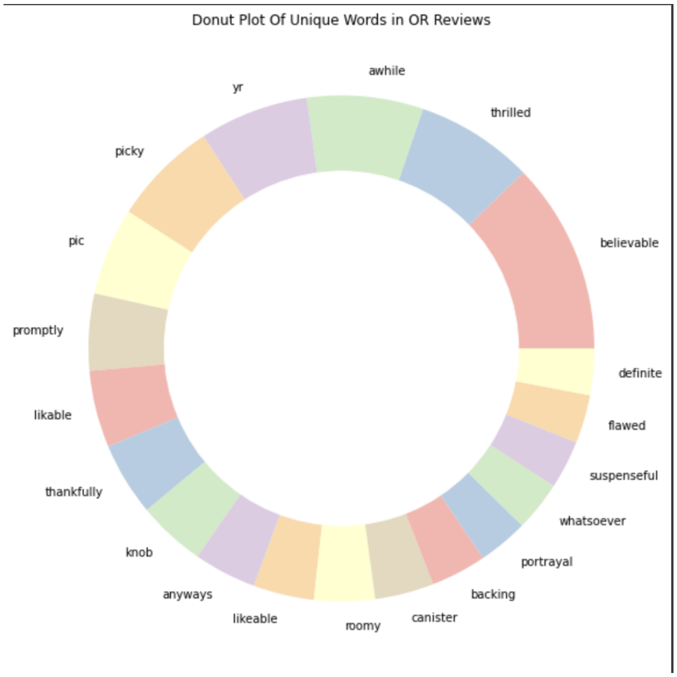


Fig. 4. Donut Plot of Unique Words in OR Reviews

extent, but the other way around is not. Here, we present the results with accuracy and recall for all the algorithms on all the datasets.

Dataset	New data - Rating		New data - Sentiment	
Algorithm	**Accuracy**	**Recall**	**Accuracy**	**Recall**
Logistic Regression	**77.0**	80.7	**77.1**	80.9
K-nearest neighbors	51.9	46.3	54.4	49.6
Support Vector Classifier	71.0	76.0	71.2	76.4
Naive Bayes	73.0	76.3	73.1	76.3
Decision Tree	63.3	70.8	63.8	80.1
Random Forests	67.2	**83.2**	67.8	75.0
AdaBoost	71.7	81.0	71.7	81.0
XGBoost	73.8	79.2	74.2	80.0
CNN + Attention	75.7	73.5	75.9	**83.6**
BERT	65.8	63.2	65.8	67.4
CNN	67.6	68.0	66.0	66.1
BiLSTM	73.1	81.2	74.0	76.7

TABLE II
ACCURACY & RECALL FOR NEW DATA

The 'New data - Rating' in Table 1 means the combination of features like 'Text, Category, Rating, Word count categories' were used for training the model, and 'New data - Sentiment' here means the combination of features like 'Text, Category, Sentiment, Word count categories' were used for training the model. As can be inferred from the results, accuracy and recall aren't affected much whether we use 'rating' or 'sentiment'. Logistic Regression provides the best results among the traditional algorithms and CNN + Attention provides the best results among the neural network algorithms. Another observation is that overall, Logistic Regression gives the best results across the features.

Dataset	Old data - Rating		Old data - Sentiment	
Algorithm	**Accuracy**	**Recall**	**Accuracy**	**Recall**
Logistic Regression	92.1	92.5	92.1	92.5
K-nearest neighbors	76.5	87.7	77.6	90.0
Support Vector Classifier	88.6	89.7	88.7	89.8
Naive Bayes	86.8	87.0	87.1	87.4
Decision Tree	80.1	74.6	80.8	74.7
Random Forests	86.4	85.7	86.4	85.6
AdaBoost	85.9	84.8	85.9	84.8
XGBoost	91.1	90.6	91.2	90.2
CNN + Attention	**95.4**	**94.9**	**94.7**	**95.5**
BERT	89.8	87.8	89.4	91.8
CNN	93.2	93.2	93.2	93.2
BiLSTM	93.3	93.1	93.3	93.0

TABLE III
ACCURACY & RECALL FOR OLD DATA

The 'Old data - Rating' in Table 2 means the combination of features like 'Text, Category, Rating, Word count categories' were used for training the model, and 'Old data - Sentiment' here means the combination of features like 'Text, Category, Sentiment, Word count categories' were used for training the model. As can be inferred from the results, accuracy and recall aren't affected much whether we use 'rating' or 'sentiment'. Here also, Logistic Regression provides the best results among the traditional algorithms and CNN + Attention provides the best results among the neural network algorithms. Another observation is that overall, CNN + Attention gives the best results across the features.

Dataset	New data - Text		Old data - Text	
Algorithm	**Accuracy**	**Recall**	**Accuracy**	**Recall**
Logistic Regression	**76.4**	80.6	91.6	91.8
K-nearest neighbors	52.0	74.5	72.8	54.4
Support Vector Classifier	71.2	76.8	87.9	88.2
Naive Bayes	74.1	76.2	86.9	86.2
Decision Tree	63.0	83.6	77.9	69.6
Random Forests	67.3	**85.1**	86.0	84.1
AdaBoost	71.7	81.0	85.6	83.7
XGBoost	73.4	79.0	90.0	89.5
CNN + Attention	51.2	0.1	94.5	92.4
BERT	66.0	68.4	88.4	84.7
CNN	71.4	63.7	**95.3**	**95.4**
BiLSTM	73.1	80.3	93.4	92.8

TABLE IV
ACCURACY & RECALL FOR ONLY TEXT DATA

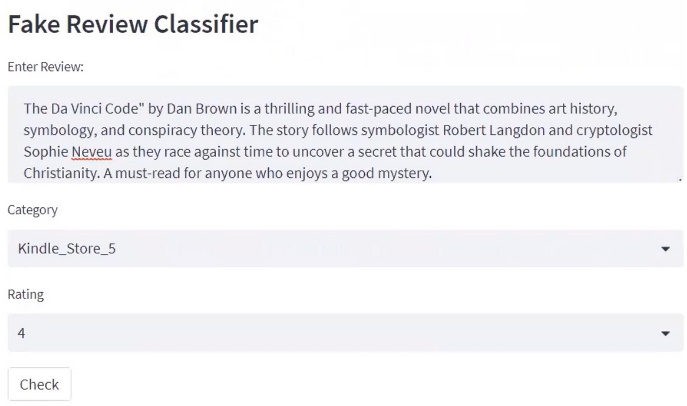
The 'New data - Text ' in Table 3 means only 'text' was used for training the model, and 'Old data - Text' here means only 'text' was used for training the model. Here also, Logistic Regression provides the best results among the traditional algorithms and CNN provides the best results among the neural network algorithms.

Based on the results provided in the above 3 tables, we can infer that Logistic Regression performs best for the new data while CNN-based architectures provide the best results for the old data. One more thing to note is that in general, the algorithms found it difficult to classify the new data compared to the old data and one potential reason could be that in back translation, the output is semi-restricted to produce words similar to the input words whereas when using GPT-2, the model can use its creativity to generate words and sentence structure different than the input sentence. The major reasons for the high accuracies of the models could be the presence of few features and high-dimensionality data for Logistic Regression and the use of Attention layer which has been proven to be successful for text data for CNN + Attention.

Moreover, there isn't much difference in the accuracy and recall of the best-performing models when using all the features compared to only using text, implying that text is the single most important feature for classification. This is also shown by the feature importance calculated while training the traditional models, where we see that all features apart from the text have a minimal effect on the accuracy and recall of most of the models. with regards to the inference speed, Logistic Regression performs much better than CNN as Logistic (about 15-20 secs) takes 3-4 times less time than the CNN architectures (about 60 secs) to provide inference along with the explanations for any input.

VII. UI / Visualization Interface Designs

We built our UI using Streamlit [28] + Python. Streamlit is a Python framework that allows developers to create interactive web applications with ease. Using Python libraries like Pandas [29], NumPy [30], and Matplotlib [31], developers can build data-centric applications quickly with Streamlit.



Fake Review Classifier

Enter Review:

The Da Vinci Code" by Dan Brown is a thrilling and fast-paced novel that combines art history, symbology, and conspiracy theory. The story follows symbologist Robert Langdon and cryptologist Sophie Neveu as they race against time to uncover a secret that could shake the foundations of Christianity. A must-read for anyone who enjoys a good mystery.

Category

Kindle_Store_5

Rating

4

Fig. 5. UI Visualization - I

2 models have been used for generating the explanations, one with the Logistic regression trained on the new text data and another being the CNN + Attention model trained on the old text data since they are the best-performing models on each of the datasets respectively. The confidence being provided is the average of the output of both the models.

The following are the main features of our UI (Fig 5, Fig 6):-

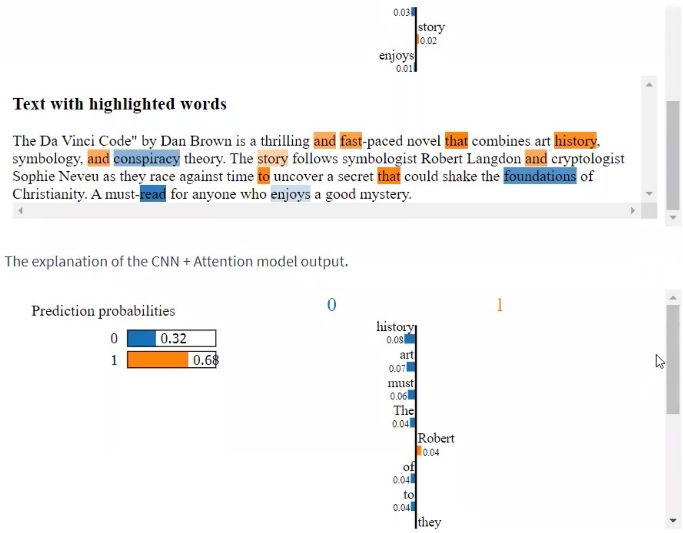
- Project Description: Related links to the dataset, project repo and classifiers used is provided
- User Input: Input box for the user to enter the review to check, drop-down lists to select review item category and rating
- Prediction and Confidence: The prediction (Spam / Not Spam) is shown along with the % confidence model has in the prediction
- Model Explanation: The % confidence, contribution chart highlighting the contribution of each token to the prediction, and text with words highlighted based on their contribution. The numbers next to each word denote the amount of accuracy which could be affected if this word was not present in the input. the model explanations are generated using the LIME [32] library.

VIII. Division of Work and Team Members' Contributions

The division of work has been presented in Table V. All the members contributed equally to the presentation and report preparation. The individual contributions to the other tasks are detailed below:

Hastin Modi: Created the new dataset using back-translation and contributed to feature pre-processing. Also trained all the traditional models on 4 datasets (except those containing only text) and the CNN + Attention architecture on 2 datasets (text ones). He also created the web UI interface using Streamlit for the demo.

Shivam Bodiwala: Researched for the literature review. Trained the traditional models on 2 datasets (text ones) and the BERT architecture on 4 datasets (except those containing only text). Also found the feature importance for all the traditional algorithms. Finally, he worked on exploring future avenues for the project.



Text with highlighted words

The Da Vinci Code" by Dan Brown is a thrilling and fast-paced novel that combines art history, symbology, and conspiracy theory. The story follows symbologist Robert Langdon and cryptologist Sophie Neveu as they race against time to uncover a secret that could shake the foundations of Christianity. A must-read for anyone who enjoys a good mystery.

The explanation of the CNN + Attention model output.

Fig. 6. UI Visualization - II

Visaj Nirav Shah: He researched papers describing existing popular fake review detection systems. For feature extraction, he contributed to data preprocessing and finalizing which features to train on. He also created data visualization charts for this part. He performed all experiments for the CNN architecture on all 6 datasets. He was part of results aggregation, and deciding which models performed the best and which to deploy in the UI.

Rahil Ashish Shah: Trained the BERT architecture on 2 datasets (text ones) and contributed to hyperparameter optimizations for the deep learning models. Conducted error analysis to identify and address any issues in the models' performance.

Mohil Devan Khimani: He was part of the Data Exploration team that worked on the back-translated dataset. He also worked on the feature extraction process from the datasets. Along with that, he trained the CNN + Attention architecture on 4 datasets (except those containing text). Moreover, he was responsible for doing the comparative analysis of the models.

Shivani Nandani: Researched various papers to find relevant papers for the project. Contributed to the process of finding and extracting the features for the training process. Implemented the BiLSTM model for all 6 datasets. Along with that, contributed to development and optimization. Worked on error analysis for implemented models.

IX. Conclusions

This project involved using traditional and modern machine learning algorithms such as neural networks to classify reviews as fake or real. We used a dataset that included fake reviews generated from GPT-2 and real reviews from Amazon. Using the real reviews, another dataset was generated where the fake

Task	Deadline	Ownership
Literature Review and Research Plan	03/01/23	Visaj, Shivani, Shivam
Data Exploration	03/01/23	Hastin, Mohil, Rahil
Feature Extraction	03/15/23	Rahil, Mohil, Visaj, Shivani
Model Creation and Implementation	03/31/23	Visaj, Shivani, Hastin, Shivam, Rahil, Mohil
Optimizations for Model Training	04/07/23	Hastin, Rahil, Shivani, Mohil
Comparative Analysis of Models	04/12/23	Visaj, Mohil
Error Analysis	04/15/23	Shivani, Rahil
Future Scope	04/15/23	Shivam, Hastin
Presentation/Demo	04/18/23	Visaj, Hastin, Mohil, Rahil, Shivam, Shivani
Final Report	04/30/23	Visaj, Hastin, Mohil, Rahil, Shivam, Shivani

TABLE V
MEMBERS' CONTRIBUTIONS

reviews were generated using back-translation. New features like sentiment and word count categories were generated to try and improve the models' accuracy. Overall, we found that Logistic Regression performs best for the new data while CNN-based architectures provide the best results for the old data. We implemented the traditional algorithms using the scikit-learn library and the neural networks using TensorFlow and Keras. Future work could include analyzing in more depth the main difference in the datasets of back-translation and GPT-2 so that the models could be trained to focus on the main features affecting the models' accuracies on the back-translation dataset. One potential area of research could be looking at words where there is no direct translation between the original language, English, and the language it is being translated into, French, this could show instances where the model might be struggling to classify fake reviews properly and could offer more insight into why the accuracies are significantly different between both the datasets.

REFERENCES

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[2] J. Salminen, C. Kandpal, A. M. Kamel, S. gyo Jung, and B. J. Jansen, "Creating and detecting fake reviews of online products," *Journal of Retailing and Consumer Services*, vol. 64, p. 102771, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0969698921003374

[3] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–242, 1958.

[4] T. Cover and P. Hart, "Nearest neighbor pattern classification," in *Information Theory, IEEE Transactions on*, vol. 13, no. 1. IEEE, 1967, pp. 21–27.

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[7] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] A. Graves, S. Fernandez, and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[13] R. Mohawesh, S. Xu, S. N. Tran, R. Ollington, M. Springer, Y. Jararweh, and S. Maqsood, "Fake reviews detection: A survey," *IEEE Access*, vol. 9, pp. 65 771–65 802, 2021.

[14] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018. [Online]. Available: https://arxiv.org/abs/1801.06146

[15] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[16] A. Mukherjee, V. Venkataraman, B. Liu, and N. Glance, "What yelp fake review filter might be doing?" *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 7, no. 1, pp. 409–418, Aug. 2021. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14389

[17] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics*, vol. 9, no. 3, p. 483, 2020.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] Y. Chen, "Convolutional neural network for sentence classification," Master's thesis, University of Waterloo, 2015.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," 2016.

[23] F. Chollet, "Keras: Deep learning library for theano and tensorflow," *GitHub*, vol. 87, p. 2, 2015.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017.

[27] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[28] A. Aneja, A. Ramaiah, A. Patnaik *et al.*, "Streamlit: The fastest way to build custom ml tools," https://www.streamlit.io/, 2021.

[29] W. McKinney, "Data structures for statistical computing in python," *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, 2010.

[30] T. E. Oliphant, "A guide to numpy," *USA: Trelgol Publishing*, vol. 1, pp. 1–283, 2006.

[31] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[32] M. T. Ribeiro, S. Singh, and C. Guestrin, "“why should i trust you?”: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144.