Running the Code

The code runs on Kaggle, but I am not sure what versions Kaggle is using. Packages used: Numpy, Pandas, spaCy, PyTorch. Please also download the Stanford gloVe Wikipedia + Gigaword 6B token (300d) file as it is used for the word embeddings.

Model

The model chosen is a neural network. It is implemented simply, consisting of 3 linear layers, with ReLU activation layers. It was trained with 7000 epochs at a 0.001 learning rate. These hyperparameters were chosen such that the model could train sufficiently without overfitting.

In order to compare Naive Bayes, Logistic Regression, and Neural Network, different combinations of features and models were tested.

Features	Naive Bayes	Logistic Regression	Neural Network
unigram, tf-idf	0.645	0.646	N/A, see (b)
(1,2)-gram, tf-idf	0.636	**0.654**	N/A, see (b)
(1,3)-gram, tf-idf	0.637	**0.650**	N/A, see (b)
word embeddings (50 dim)	N/A, see (a)	0.628	0.632
word embeddings (300 dim)	N/A, see (a)	0.638	0.642
word embeddings (300 dim) + keyword counts	N/A, see (a)	**0.651**	**0.654**

Some fields are not applicable because:

a. For Naive Bayes model, it does not make sense to use a word embedding as an input because the model is based on word counts and word probabilities and the assumption of conditional independence between words, not vector values.
b. For Neural Network, using unigram or higher order grams will result in a high dimensionality, very sparse input. It is not practical to train a neural network this way because it will be very difficult for it to converge.

As shown by the bolded values, both the logistic regression and neural network perform decently well. Naive bayes performed worse because it works on the assumption of conditional independence of words, whereas the words are likely not independent of one another. Also, it would probably fare better on a larger dataset where the bigrams and trigrams are more frequent. Logistic regression performed well when fed bigrams and trigrams, likely because it was able to detect some contextual cues from the text for better prediction. However, ultimately I decided on the neural network for its flexibility and ability to learn nonlinear information from the data, which gives it a small edge over logistic regression.

Preprocessing

Before features were extracted, these preprocessing steps were performed on the data:

- Data Cleaning
- Balancing
- Tokenization

Data Cleaning involves removing missing and duplicate entries. There were no missing entries in the data, but a few duplicate entries were found with the same “Text” but different “Verdict”. Since it is not clear what the actual correct “Verdict” is, these entries were removed.

Balancing involves balancing the data classes to obtain an equal amount of each class before training a model on the data. This reduces bias towards any one class. The data was found to have a very strong bias towards the -1 class (14k rows), and a modest 5k rows for the 1 class and 2k rows for the 0 class. To balance the data, downsampling was conducted. 2388 rows were obtained for all 3 classes.

Tokenization involves splitting the “Text” column into tokens which features can then be extracted from. SpaCy’s tokenizer was used to extract lower case forms. Since spaCy is able to determine the word class of each token, I attempted removing all punctuation, determiners, numbers, symbols, and interjections in this step. I also attempted removing stopwords and lemmatization.

The following results were obtained for the neural network model.
Only tokenization was varied, keeping all other parameters the same.

Lemmatiza -tion	Stopword Removal	Removal of determiners, numbers, etc.	Lowercase	Model Performance (Macro F1 Score)
No	No	Yes	Yes	0.634
No	Yes	Yes	Yes	0.613
Yes	No	Yes	Yes	0.595
No	**No**	**No**	**Yes**	**0.648**
No	No	Yes	No	0.616
No	No	No	No	0.626

From the data, it seems that the stopwords and other words that don’t contribute much to a sentence’s meaning carried some signal to the prediction. Hence, only case folding was done.

Feature Engineering

Word embeddings were used to derive features from the text, summing over the individual word embeddings in the whole sentence. This is supposed to represent the meaning of the sentence. However, there are limitations to this approach. By summing over a whole sentence, the final word embeddings are likely very noisy.

Perhaps, tf-idf unigram/bigrams would have worked better to capture the meaning of a sentence by capturing the context of each word, but would make addition of other features difficult. This is because of its high dimensionality (up to 13k columns) so for instance, adding 9 columns of keyword counts would not affect the result very much. Hence, word embeddings, despite their limitations, were chosen for their dense representation.

Keyword counts were used to encode other data not well represented semantically in text. I counted keywords which I deemed to represent concepts that were likely present in a lie or truth. This is partly based on my own judgement and partly based on categories derived by other researchers [1]. These categories are:

- Reference other people (you, people, others, etc.)
- Reference self, and those close to self (i, self, me, friends, etc.)
- Causal (cause, effect, hence, etc.)
- Negation (no, not, neither, etc.)
- Promises (‘ll, will, believe, upcoming, etc.)
- All or nothing (everything, nothing, all, no, never, etc.)
- Prosocial (care, help, please, thank, etc.)
- Togetherness (we, us, together, etc.)
- Accusation (lie, steal, betray, harm, ruin etc.)

This worked well, though had limited effect because it is impossible to enumerate all keywords representing a certain concept. The keyword list is also prone to bias because it is based on my judgement.

Word Class of each word was counted and vectorized with tf-idf to attempt to represent sentence structure. However, this worsened the model’s performance. This is perhaps because the structure of the sentence does not have much bearing on whether it is a truthful statement.

Representing nouns, verbs, and the rest in 3 separate embedding vectors was attempted, but worsened the model’s performance. The hypothesis was that the model would perform better if it understood the meaning of the nouns and verbs in the sentence. However, perhaps the truthfulness of a statement does not have a strong relation to the meaning of the nouns and verbs, so this feature simply added more noise than signal as it increased dimensionality significantly.

Ultimately, only **word embeddings** and **keyword counts** were chosen to be used as features for the model. These contribute to the performance of the model whereas other features were found to not be helpful to the model’s learning.

Analysis

I conduct an ablation study here to see how each part of the workflow contributes to the performance of the model. I will start with a baseline neural network that is trained on 50-dimensional word embeddings, without any data cleaning, balancing, and only very basic tokenization (no case folding). Then, I will progressively add on to the workflow to see how the model performance improves. The data shown here is an average Macro F1 score taken over a few training attempts, though variance is controlled for by setting sampling algorithm seeds.

Data Processing and Model Architecture	Model Performance (Macro F1 Score)	Description, Explanation
Baseline Model: Only 50-dim word embeddings	0.549	Extreme bias towards the -1 class achieves ~0.9 recall score on the -1 class, negatively affecting macro F1 greatly.
+ data cleaned (removes ~150 rows)	0.573	Not much improvement because data cleaning only removes missing and duplicate values. These comprise only a very small amount of data points. The training data is still very biased.
+ classes balanced	0.642	Very large improvement because this resolves the strong bias towards the -1 class present in the training set.
+ case folding	0.648	Small improvement, mainly from removal of capitalisation at the start of a sentence.
+ 300-dim word embeddings instead	0.667	300 dimensional word vectors encode more information, helping the model better understand the sentence's meaning.
+ keyword counts	0.675	Keyword counts encode more information for the model to learn intangible concepts.

References

1. Brzic, B., Boticki, I., & Bagic Babac, M. (2023). Detecting Deception Using Natural Language Processing and Machine Learning in Datasets on COVID-19 and Climate Change. Algorithms, 16(5), 221. https://doi.org/10.3390/a16050221
   → For better understanding the features that a false statement can contain

Declaration of Original Work

By entering my Student ID, I certify that I completed my assignment independently of all others. In particular, I did not discuss the problems and my solutions in this assignment with any student, alumnus, or AI tool. The work I submit is solely my own work.
Signed, A0252082M