CS4248 Assignment 2
A0236430N

Observation / Measurement Data:
https://docs.google.com/spreadsheets/d/19-RVrBroG2b0y0my1UT0PGsxieykSEIZ0aCPL45P6Cl/edit?usp=sharing

Instructions to Run
Make sure Python 3.10 is installed with numpy, pandas, nltk, sklearn libraries, Glove 6B, nltk('punkt') and nltk('wordnet') datasets downloaded.
To run, use `python3 assignment2.py`.

Summary
I explored 3 models (NB, LR, MLP), 3 feature extraction (CountVectorizer, tf-idf, Glove embeddings), various data preprocessing (upsampling and lemmatization), and perform ablation study on whether including or removing one component changes model performance. All measurement data from the "best" performance model in each combination is recorded in the link above. Here, "performance" is analogous to F1 score, not "speed" performance.

Models
The models I tried were MultinomialNB (NB), LogisticRegression (LR), MLPClassifier (MLP), GradientBoostingClassifier (GBC), and ComplementNB (CNB). However, we will only discuss NB, LR, and MLP, and discard the rest. After 7 runs on CNB, I decided to discard it due to a much lower F1 score on validation and test (~0.65) compared to NB (~0.78) for all feature extraction. Possible explanation is that CNB uses complement statistics to compute the model weights and our dataset may not be suitable for that weight calculation. Both NB and CNB assume feature independence given the class label, but this assumption might not hold for our dataset causing NB to outperform CNB. I planned to try HistGradientBoostingClassifier instead of GBC due to faster estimator calculation compared to GBC for larger datasets, but it throws sklearn Not Fitted Error on my machine. Thus, I proceeded with GBC, and it takes too much time to train, more than 10 times longer than MLP. Due to time constraint and subpar performance (~0.69), I discard GBC.

I chose NB because it is the default model against which we are benchmarking, so I wanted to explore the benchmark performance to beat. I chose MultinomialNB among other sklearn Naive Bayes models because it learns from discrete features and is suitable for text classification including tf-idf features. Using GridSearchCV, I varied 'alpha' / smoothing parameter from [0.01, 0.1, 1, 10] and whether to learn class prior probabilities. The most performant model (~0.78) is when using alpha = 0.1 or 0.01 and fit_prior = False.

I chose LR because it works well for text classification due to its simplicity, efficiency, and interpretability. In text classification where the goal is to predict a categorical verdict, logistic regression can effectively model the relationship between the input text features and the target labels and build hyperplanes to separate 3 classes. It handles high-dimensional sparse data

common in text analytics efficiently and doesn't require extensive computational resources. In the end, this is my best model with an F1 score of 0.85214 on testing with ‘C’ / L2 regularization strength = 1 and max iteration of 300. Possible explanation why LR achieves the highest F1 score is that it best discriminates the 3 classes among other models. NB assumes that features are conditionally independent given the class label, which might not hold true for text data where word occurrences are correlated. LR doesn't make such strict assumptions about feature independence. Moreover, MLPs are prone to overfitting, especially when dealing with small datasets or when the model architecture is too complex relative to the amount of available data. LR, being a simpler model, is less susceptible to overfitting. In this task, especially with high-dimensional sparse feature representations like bag-of-words or TF-IDF, logistic regression can perform well because it handles sparse data efficiently. MLPs might require more data or regularization techniques to effectively handle sparsity. One reservation is that it has a low validation score (~0.67), so perhaps the testing performance is just “lucky”.

I chose MLP due to their ability to capture complex nonlinear relationships within the data. Text data is inherently high-dimensional and nonlinear, with intricate patterns that may not be easily captured by simpler models. By processing text data through hidden layers, MLPs can extract hierarchical features, learning to represent the underlying pattern of the text. In fact, I was most confident with my MLP performance because its training score is ~0.99 and validation score is ~0.93. However, its testing score is ~0.80. This may be a sign of overfitting, which is why I employed several techniques to reduce MLP overfitting, such as early stopping, smaller hidden layers ((100,) down to (25,)) to reduce model complexity, weight regularization, and cross validation. In the end, testing score increases and validation score decreases, which are signs of overfitting reduced. However, the testing performance is still below LR, although I am more confident in MLP performance because it has a high validation score (LR can just be “lucky”).

Preprocessing

In my text preprocessing function, I tried removing stopwords, lemmatization using WordNetLemmatizer, nltk word_tokenize, and cleaning to only include alphabetical words. In theory, these preprocessing can help the model to learn the “important” underlying patterns of the language. But after experimenting and exploring combinations of these preprocessing methods, the one that produces the highest results for NB, LR, and MLP is when using **only** nltk word_tokenize (up by ~0.05 points). Including any other preprocessing lowers F1 validation and testing score. Possible explanation is that to differentiate important and non-factual statements, the models can better discriminate when stopwords, non-lemmatized words, and numerical tokens are included. It makes sense that usually non-important and non-factual statements include too many punctuations (lost by removing stopwords), informal words (lost by lemmatization), and numerical data (lost by alphabetical cleaning). When we preserve this information by not removing them through preprocessing, the model can learn which feature accounts for factual and which accounts for unimportant statements. Additional preprocessing may cause loss of information that may be beneficial for the model to discriminate between different classes.

Other data preprocessing that I used is data upsample. I noticed that the composition of classes in our dataset is imbalanced (5213, 2403, 14685). So I performed upsampling for Verdict 1 and 0 because models trained on balanced datasets tend to perform better in terms of overall performance and generalization to unseen data. Imbalanced datasets can lead to models biased towards the majority class, resulting in poor performance on minority classes. Balanced datasets help reduce bias towards the majority class, ensuring that the model learns to distinguish between different classes more effectively. Models trained on balanced datasets are often more robust, fair, and reliable, as they are less likely to be influenced by the skewed distribution of classes. Indeed, for all models and feature engineering, those with upsampled data perform better than otherwise. The only reason I did not upsample is due to my machine's memory constraint. For LR using CountVectorizer ngram_range = (1,2) and max features more than 20000, my operating system throws an exception of insufficient memory, so not doing upsampling enables my machine to process more features and get more comprehensive experiment results.

Feature Engineering

I experimented using CountVectorizer (CV), TfidfVectorizer (tfidf), and Glove 6B 300D vector embeddings as my feature extraction methods. For CV, I varied the params between ngram (1,) and accept all features, and ngram (1,2) and set max_features between 8000 to 20000. More than 20000 for ngram (1,2) lead to insufficient memory on my machine. I found that using ngram (1,2), i.e., including bigrams, increases performance (by ~0.05 points). Possible explanation is that including bigrams allows the model to capture more contextual information compared to unigrams alone, so that models can learn more about given text hence better classification performance. After I noticed that LR with CV is the best model on testing, I tuned the max_features of CV and found that max_features of 9000 gets me the highest F1 score. This makes sense because if the number of features is too low, the model may not get enough data to learn text pattern optimally, and if it's too high, overfitting may occur which does not scale well to test data.

For tfidf, I use ngram_range of (1,2) which leads to better performance compared to ngram (1,) due to similar reasons as above. I found that MLP performs better using tfidf than CV, and LR better using CV than tfidf. Possible explanation is that tfidf weighs terms based on their importance in the document and across the corpus. This weighting scheme captures not only the frequency of terms in individual documents (as in CV) but also their discriminative power across the entire dataset. MLP can effectively leverage these weighted features to learn more meaningful representations of the text data, potentially leading to better classification performance. Furthermore, tfidf implicitly performs dimensionality reduction by downweighting terms that occur frequently across documents, resulting in a sparser feature space compared to CV. This can help reduce the computational complexity of MLP training, making it more efficient and less prone to overfitting. On the other hand, LR is a linear model that assumes a linear relationship between the features and the log-odds of the target variable. CV produces raw counts of term occurrences, which aligns well with the linear assumptions of logistic regression. In contrast, tfidf applies a transformation that can introduce nonlinearity in the feature space, potentially making it less suitable for LR. CV preserves the raw frequency information of terms in

the document, which can be valuable for LR since it makes predictions based on the weighted sum of input features, and raw term frequencies provide direct information about the importance of each term in the document. Tfidf, on the other hand, scales down the importance of terms that are frequent across the entire corpus, potentially losing some valuable frequency information that LR can benefit from.

After experimenting with Glove embeddings, I did not get any increase in performance compared to CV and tfidf for NB, LR, and MLP models (reduction of ~0.05 points). I store all embeddings in a dictionary with the format {word: vector embedding}. For each sentence / text, I build a vector that represents that text by summing the vectors of each word in the sentence. Then, I tried 2 methods to get to the final vector: take the average (word count does not matter) or take the weighted average (word count matters). Both methods lower the performance of all models compared to CV and tfidf regardless of preprocessing. Possible explanation is that Glove embeddings are trained on large corpora and capture general semantic relationships between words. However, in this case, the specific context and text structure encoded in tfidf or CV vectors might be more relevant for classification. In addition, in the transformation of finding the mean of vectors, information is lost so the models may not capture the sentence pattern to differentiate factual and unimportant statements.

Further Research and Conclusion
Ablation study and detailed explanation is included above on why I made certain decisions and why some choices perform better than the other in each section.

Overall, my best submission on Kaggle is LR with CV with only word_tokenize as preprocessing. However, I still think MLP with tfidf and word_tokenize should be the “best” model of predicting the overall test data due to its high validation score. LR may just get “lucky” on the 50% of test data. Alternatively, MLP might overfit with training data and LR is actually better. Additional experiments need to be done to determine the best model overall that can better predict test data. Possible further research may include better models such as LSTM or transformers, data augmentation (adding more data), and more advanced feature extraction.

1A. Declaration of Original Work. *By entering my Student ID below, I certify that I completed my assignment independently of all others (except where sanctioned during in-class sessions), obeying the class policy outlined in the introductory lecture. In particular, I am allowed to discuss the problems and solutions in this assignment, but have waited at least 30 minutes by doing other activities unrelated to class before attempting to complete or modify my answers as per the Pokémon Go rule.*

Signed, A0236430N

2. References. *I acknowledge that I used the following websites or contacts to complete this assignment (but please note that many uses of Web search and detailed discussion are not allowed:*

- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://nlp.stanford.edu/projects/glove/
- https://spotintelligence.com/2023/11/27/glove-embedding/