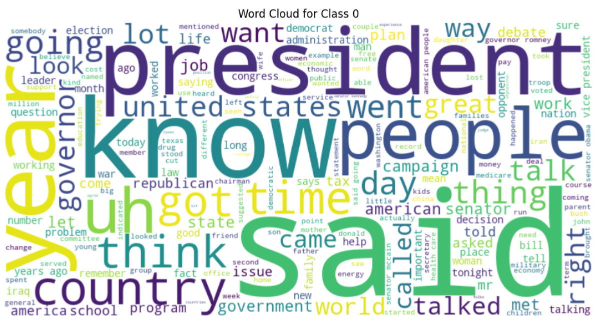
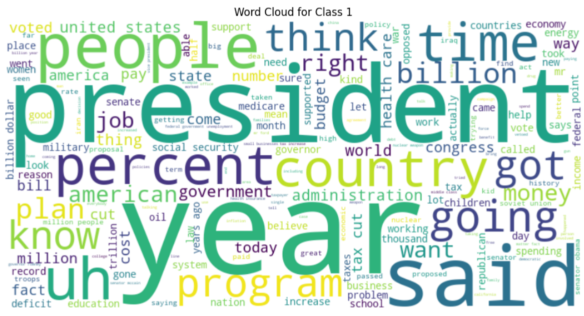
Model

Based on the models and features that I have tried, the best solution for this is to use TF-IDF to create vectors of the sentences, calculate syntactic features and use LightGBM classifier to classify the sentences.

Based on the data, each sentence was similar to each other in terms of the domains. I plotted a word cloud for each class, and found that all 3 classes in the training data had similar words (president, think, said, year, know, people). Hence, it would make sense to calculate the inverse frequencies of the words to better separate the classes. Using rare words (or phrases), this method to generate features gave the best results.

Word Cloud for Class -1
people
think
going
believe
president
country
know
want
thing
america
world
need
year
way



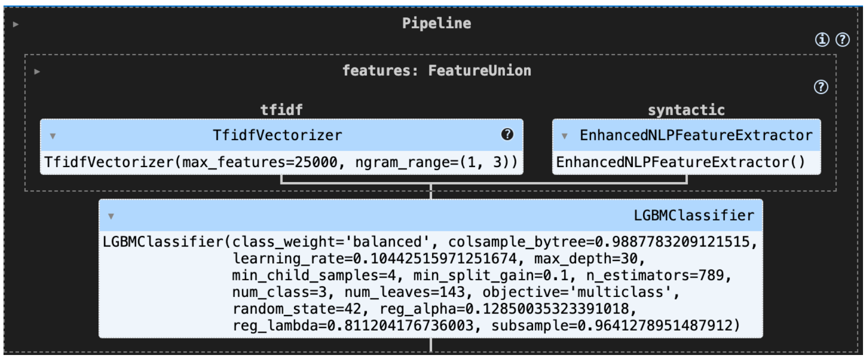


Some syntactic features calculated were sentence length, punctuation ratio, ratio of nouns, verbs, adjectives, POS tags, dependency features, entity ratios and semantic similarity. These were done using spacy's model en_core_web_md.

I also noticed the imbalanced dataset, where there were more samples belonging to the -1 class label and least samples belonging to the 0 class label.

```
Verdict
-1    14685
 1     5413
 0     2403
```

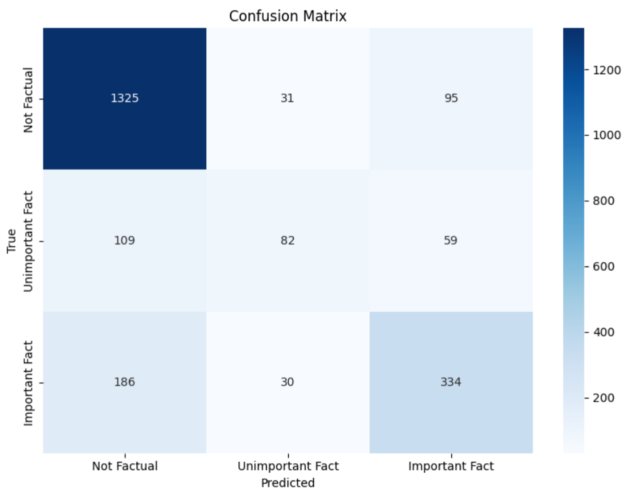
To prevent the model from always predicting -1 due to the majority class, I used oversampling to balance the samples for each class. This was done using SMOTE. I also used Logistic Regression model and set weights to "balanced".
I also tested with Naïve Bayes, SVM, Logistic Regression and Neural Network. More will be explained below in the ablation study.
This was the pipeline for the model:



```
Pipeline
features: FeatureUnion
tfidf                                             syntactic
TfidfVectorizer                                   EnhancedNLPFeatureExtractor
TfidfVectorizer(max_features=25000, ngram_range=(1, 3))   EnhancedNLPFeatureExtractor()

LGBMClassifier
LGBMClassifier(class_weight='balanced', colsample_bytree=0.9887783209121515,
               learning_rate=0.10442515971251674, max_depth=30,
               min_child_samples=4, min_split_gain=0.1, n_estimators=789,
               num_class=3, num_leaves=143, objective='multiclass',
               random_state=42, reg_alpha=0.12850035323391018,
               reg_lambda=0.811204176736003, subsample=0.9641278951487912)
```

This was the confusion matrix for the model:



Pre-processing
After experimenting with multiple text preprocessing techniques, I ultimately decided to use raw text without additional preprocessing, as I found that preprocessing negatively impacted model performance.

I initially tried several standard preprocessing steps, including:

1. **Stopword removal**: Removing common words like "I think," "I believe," and "I feel". However, these words were crucial in distinguishing factual statements from opinions, and their removal led to poorer classification performance.
2. **Stemming and lemmatization**: Reducing words to their root forms resulted in a loss of context, which may have affected the model's ability to differentiate between subtle linguistic patterns.
3. **Punctuation removal**: Eliminating punctuation stripped the text of important cues, such as pauses and hesitations, which could indicate uncertainty in speech.

Since the dataset consists of spoken sentences, these linguistic elements likely contained meaningful signals. Removing them may have caused the model to lose important contextual information, leading to a drop in F1 score and reduced generalization to the test set.

Based on these findings, I chose to retain the raw text and extract features directly from it, which yielded better performance.

Feature Engineering
I experimented with multiple feature engineering techniques to enhance the model's performance. Below are the approaches I tested, including those that did not work, along with their effects:

1. TF-IDF with different configurations:
    a. Tested various vocabulary sizes: 5k, 10k, 15k, 20k, 25k, 30k max features.
    b. Tried different n-gram ranges: (1,2), (1,3), (1,4).
    c. Effect: TF-IDF helped highlight rare words that might carry important meaning in distinguishing between factual and unfactual statements. However, increasing the vocabulary size beyond 20k did not significantly improve performance, likely due to data sparsity.
2. TF-IDF + Linguistic Features (POS tags, Dependency Parsing, Named Entity Recognition - NER):
    a. Effect: These features aimed to capture grammatical structure and named entities that could help differentiate between facts and opinions. However, they did not significantly improve the model's performance, possibly because the dataset was too small to effectively utilize syntactic patterns.
3. TF-IDF + Syntactic Features:
    a. Included features such as sentence length, punctuation ratio, ratio of nouns, verbs, adjectives, POS tag distributions, dependency features, entity ratios, and semantic similarity measures.
    b. Effect: Some of these features, such as punctuation ratio and sentence length, provided minor improvements, likely because factual sentences tend to have different structural properties than opinions.
4. TF-IDF + Word2Vec trained on the dataset:
    a. Trained a Word2Vec model on the dataset to capture contextual relationships between words.
    b. Effect: Performance was similar to TF-IDF alone, likely because the dataset was too small to train meaningful word embeddings.
5. Pretrained Word2Vec:
    a. Used pretrained Word2Vec embeddings to incorporate external semantic knowledge.
    b. Effect: Only useful for improving the performance of Neural Networks but not for traditional models.

Handling Class Imbalance
Since the dataset had imbalanced classes, I experimented with two methods:

1. Class Weighting: Assigned weights as the inverse of class frequencies to balance the influence of different classes.
2. SMOTE (Synthetic Minority Over-sampling Technique): Generated synthetic examples to equalize class distribution.
    a. Effect: Class weighting provided better results compared to SMOTE, likely because the generated synthetic samples did not fully capture the complexity of real data.

Final Choice
Based on my experiments, I chose to use TF-IDF (with an optimal vocabulary size and n-gram range) combined with selected syntactic features. This combination yielded the best performance, as TF-IDF captured important words while syntactic features added useful structural information.

Ablation study

Features	Classification model	Feature hyper parameters	Model Hyper parameters	Macro f1 score	Macro f1 score (-1)	Macro f1 score (0)	Macro f1 score (1)
Tf-idf	SVM	Max features = 5000 n-gram range = (1, 2)		0.6418	0.8587	0.4000	0.6667
Tf-idf + SMOTE		Max features = 1000 n-gram range = (1, 2)		0.5811	0.8145	0.3482	0.5806
Tf-idf + (POS tag counts, dependency counts, NER counts)	SVM	Max features = 5000 n-gram range = (1, 2)		0.5358	0.7584	0.3044	0.5445
	Simple Neural Network	Max features = 3000 n-gram range = (1, 2)	Dropout = 0.3	0.4602	0.5985	0.2869	0.4953
			Dropout = 0.5	0.5984	0.8077	0.3560	0.6316
Tf-idf + Word2Vec (self-trained)	Logistic regression	Tf-idf max features = 5000 n-gram range = (1, 3)	Max_iter = 1000 Class_weight = balanced	0.6154	0.8084	0.4003	0.6374
Tf-idf (unbalanced)	Naïve bayes	Tf-idf max features = 5000 n-gram range = (1, 3)	Alpha = 0.1	0.5815	0.8571	0.2772	0.6102
Tf-idf + syntactic features (unbalanced)				0.5819	0.8483	0.2872	0.6104
Tf-idf + syntactic features (SMOTE oversampled)				0.5957	0.8083	0.3707	0.6079
Tf-idf + syntactic features (SMOTE oversampled)	Logistic Regression	Tf-idf max features = 5000 n-gram range = (1, 3)	Max_iter = 1000 Class_weight = balanced	0.6342	0.8389	0.4107	0.6531
		Tf-idf max features = 10000 n-gram range = (1, 3)		0.6495	0.8477	0.4324	0.6684
		Tf-idf max features = 15000 n-gram range = (1, 3)	Max_iter = 1500 Class_weight = balanced	0.6501	0.8516	0.4311	0.6676
		Tf-idf max features = 20000 n-gram range = (1, 4)		0.6521	0.8523	0.4330	0.6709
		Tf-idf max features = 25000 n-gram range = (1, 3)	Max_iter = 4000 Class_weight = balanced	0.6613	0.8591	0.4456	0.6792
		Tf-idf max features = 30000 n-gram range = (1, 4)		0.6607	0.8588	0.4452	0.6780
	SVD + 3 layer NN (256, 128, 64) Dropout = 0.5	Tf-idf max features = 15000 n-gram range = (1, 3)	3000 components, Adam optimizer, relu activation fn, learning rate = 0.001	0.5709	0.8264	0.3114	0.5751

	LightGBM classifier with hyperparameters tuned	Tf-idf max features = 25000 n-gram range = (1, 3)	Used RandomizedSearchCV to get best hyperparams	0.6413	0.8629	0.4173	0.6435
Pretrained word2vec (SMOTE oversampled)	NN 128 * 64 * 3 Dropout = 0.5	GoogleNews-vectors-negative300GoogleNews-vectors-negative300	Cross entropy loss, Adam optimizer, relu activation fn, learning rate = 0.0001 Weight_decay = 0.001	0.6155	0.8249	0.3909	0.6307
Pretrained word2vec * tf-idf weights + syntactic features (SMOTE oversampled)				0.6118	0.8338	0.3540	0.6477

Base Feature Representation
TF-IDF vectors consistently outperformed word embeddings across model types. This suggested that lexical features were more discriminative than semantic representations for this specific classification task. This also corroborates my initial Exploratory Data Analysis, where similar words were present among the classes and the domain of the sentences is the same, i.e. US politics

Comparing Neural Networks vs traditional machine learning models
I can safely conclude that Neural Networks do not generalize well for this dataset due to two key reasons:

1. TF-IDF features prove suboptimal for neural networks due to their inherent sparsity. These sparse vectors contain mostly zero values, which complicates the weight update process that neural networks rely on. Additionally, the high dimensionality of these sparse representations creates challenges for neural networks to fit properly, especially given the limited size of the dataset. This combination of factors led to rapid overfitting in my neural network models, which struggled to exceed an F1-score of 0.6 despite various optimization attempts.
2. TF-IDF vectors captured more discriminative information than sentence embeddings for this particular classification task. This is likely because the lexical differences between classes (especially between "unimportant facts" and "important facts") are more significant than the semantic differences. While these classes share similar sentence structures and vocabulary, TF-IDF can better capture the subtle differences in word frequency patterns that distinguish between classes.

Neural networks using pretrained Word2Vec embeddings did perform better than those using TF-IDF vectors, confirming that dense representations generally work better for neural architectures. However, even these improved neural models couldn't match the performance of traditional machine learning approaches like logistic regression for this specific task.

Compare Logistic Regression vs Naïve Bayes
I then decided to compare a few traditional machine learning models. In comparing Naïve Bayes and Logistic Regression, I observed that Naïve Bayes handled the class imbalance less effectively. With identical features (TF-IDF with max_features=5000 and n_gram_range=(1,3)) and SMOTE oversampling, Logistic Regression achieved an F1 score of 0.6342 compared to 0.5957 for Naïve Bayes. This performance gap widened as feature dimensionality increased, with Logistic Regression's performance peaking at 25,000 features with n_gram_range=(1,3). A key advantage of Logistic Regression is its ability to incorporate class_weights to accommodate imbalanced data distributions—a capability that Naïve Bayes lacks.

Compare SVM vs Logistic Regression
Support Vector Machines (SVM) also demonstrated strong performance. Using TF-IDF vectors with 5,000 features yielded comparable results to Logistic Regression, achieving an F1 score of 0.64 on the validation set and a Kaggle score of 0.81. This similarity in performance likely stems from both being linear models that effectively leverage high-dimensional TF-IDF representations to find decision boundaries between classes.

Investigating feature set size
I then decided to compare the effect of varying features for the model training. As seen from the table, I kept the model constant, using a Logistic Regression classifier, and incrementally changed the number of features generated by TF-IDF vectorizer, testing 5k, 10k, 15k, 20k, 25k, 30k features. I also experimented with changing the n_gram range from 1 to 3 to 1 to 4. It seemed that the peak performance of the model based on macro F1 score was when I used 25k features and an n_gram range of 1 to 3. This process increased the F1 score from 0.63 to 0.66, corresponding to a minor improvement in Kaggle score from 0.81 to 0.84. This showed me that there is a maximum limit to how well the model can perform based on the number of features, and more features doesn't necessarily mean better model performance.

Gradient Boosting vs Logistic Regression
I then decided to try some gradient boosting models, as they were known to be more robust for text classification. The most significant performance leap came from switching to LightGBM, which immediately improved results despite showing a slightly lower F1 score during cross-validation compared to Logistic Regression. After implementing randomized search with cross-validation to optimize

LightGBM's hyperparameters while maintaining the same feature set, the Kaggle score increased substantially from 0.84 to 0.87. Despite showing a slightly lower cross-validation F1 score during development, LightGBM performed better on the test set, indicating:

1. Enhanced generalization: LightGBM's ensemble nature provides better robustness against overfitting.
2. Non-linear pattern recognition: The gradient boosting approach captured complex relationships in the data that linear models missed.
3. Feature interaction modeling: LightGBM automatically detected and leveraged important feature interactions.

Conclusion of Ablation Study

This systematic analysis revealed that:

1. For this specific classification task, traditional ML models outperformed neural networks.
2. TF-IDF representations with high dimensionality and n-gram ranges captured the most discriminative information.
3. Class imbalance handling through SMOTE and class weighting provided consistent improvements.
4. The final LightGBM model offered the best balance between complexity and performance, effectively leveraging the engineered features while avoiding overfitting.

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 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.
1B. Exception to the Class Policy. I did not follow the CS4248 Class Policy in doing this assignment. This text explains why and how I believe I should be assessed for this assignment given the circumstances explained.

Signed, A0233272M