Final Project

Part 1: EDA and Features Engineering

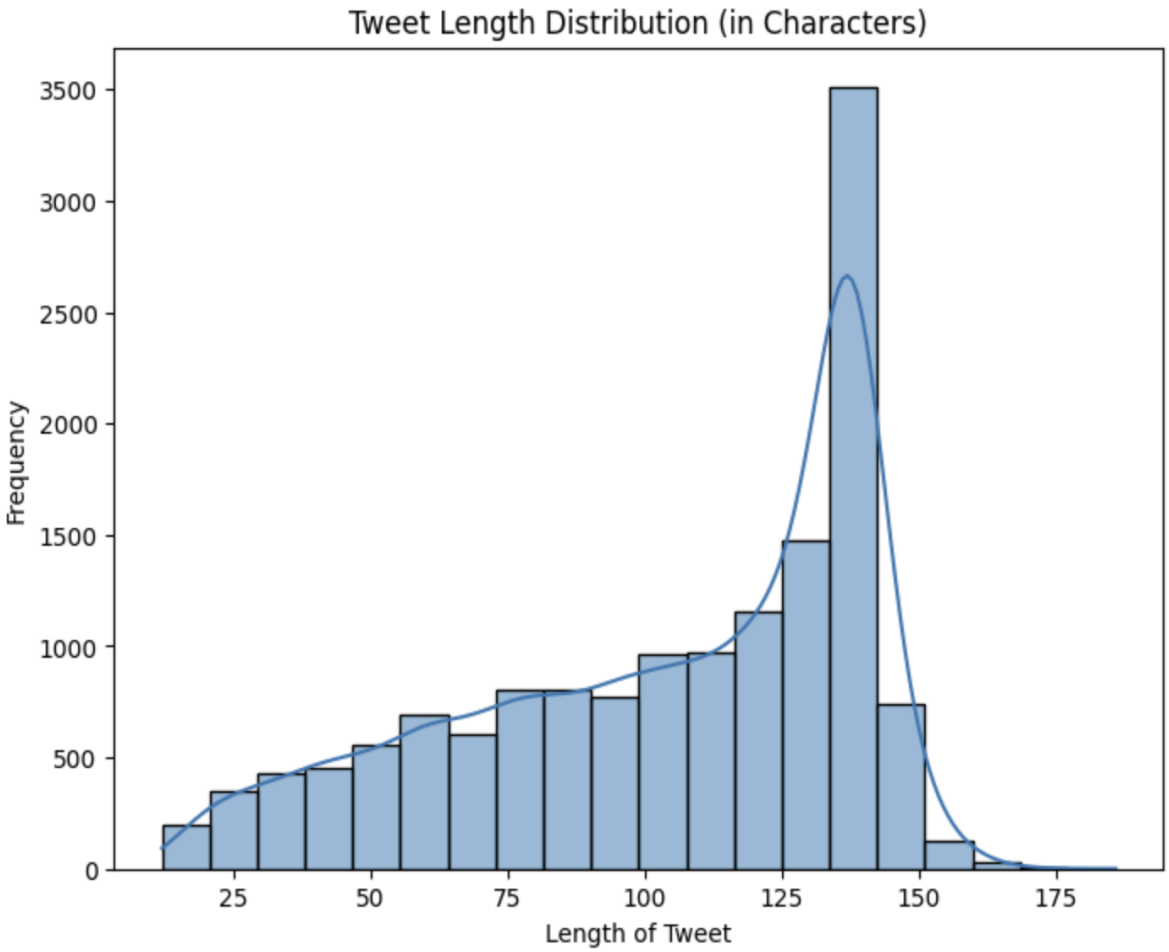
a) EDA:
The dataset csv file was loaded with the help of pandas library. The dataset contained 14640 rows and 15 columns.

The null value counts for each column looked like:

Columns	Null values
tweet_id	0
airline_sentiment	0
airline_sentiment_confidence	0
negativereason	5462
negativereason_confidence	4118
airline	0
airline_sentiment_gold	14600
name	0
negativereason_gold	14608
retweet_count	0
text	0
tweet_coord	13621
tweet_created	0
tweet_location	4733
user_timezone	4820

From the data we can see that the *airline_sentiment_gold, negative_reason_gold, and tweet_coord* can be dropped as almost all the rows on the columns contain null values. Other columns such as *tweet_location, and user_timezone* can also be dropped because it contains null values and is not required for null values. Column '*name*' is not useful for sentiment. Other columns like '*negativereason*', '*negativereason_confidence*' can be used for specific type of analysis on negative classification: reason for the negative review. So, for *negativereason* column I replaced the null with empty string and for *negativereason_confidence,* I replaced the null with 0.

The process above is how I first approached the data analysis. For the shake of the project, the only required feature is the tweet text. So, I will now drop all the other columns and work with only the text column as the feature of our dataset and perform NLP tasks, the target will be '*airline_sentiment*' column.



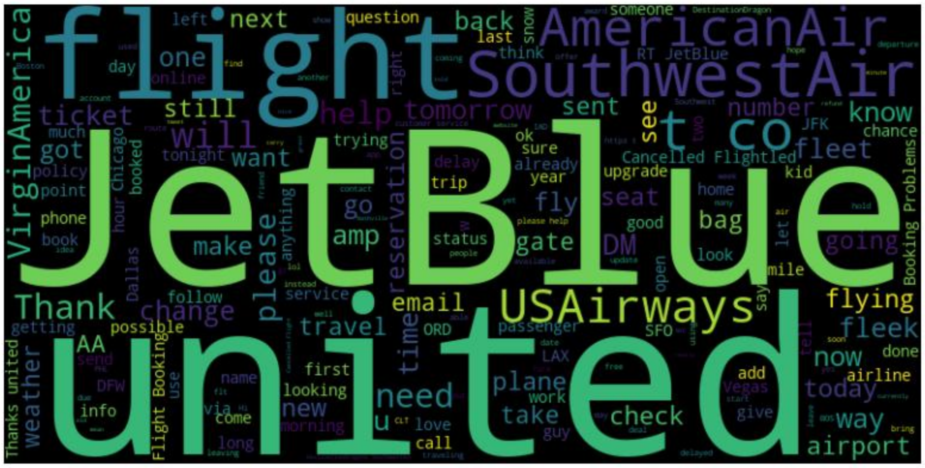
Following is the label proportion for 'airline_sentiment' column:

negative	9178
neutral	3099
positive	2363

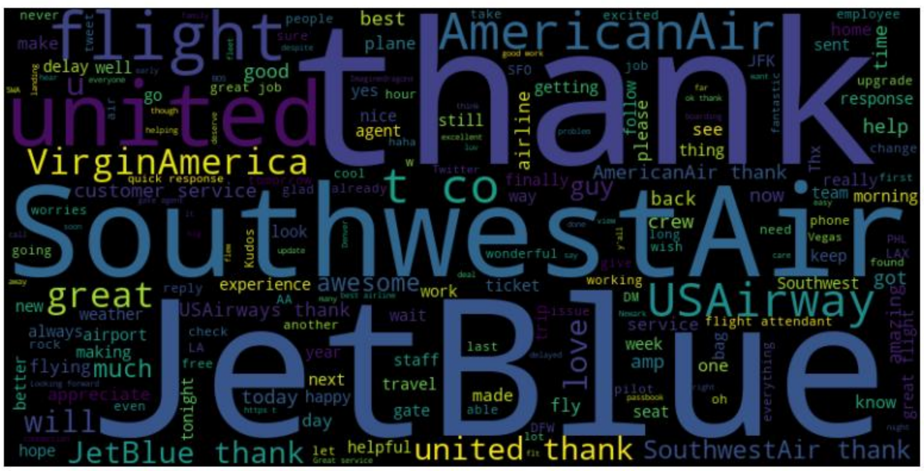
b) Text Visualization:

I used word cloud to visualize words for 3 different categories that were present in the provided data. The word clouds for each class are as follows:

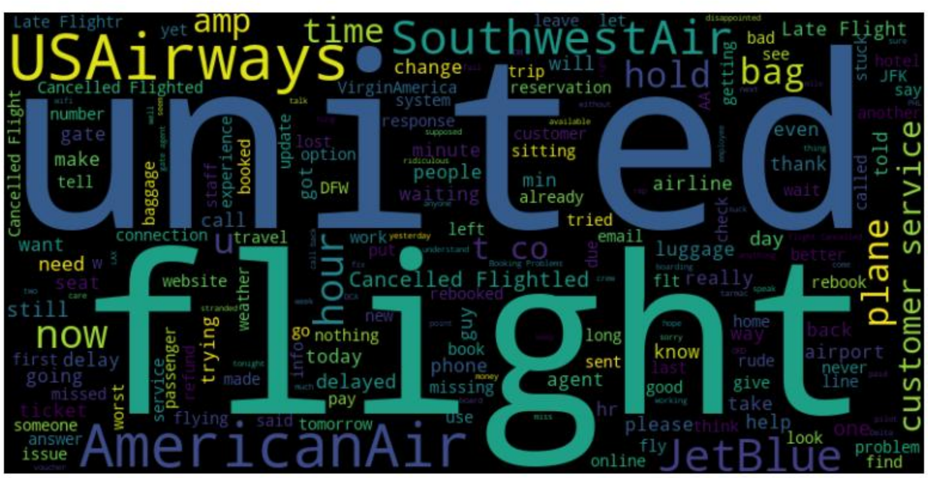
Neutral class:



Positive class:



Negative class:



The top 10 frequent words for each class is shown in the table below:

Rank	Neutral	Frequency	Positive	Frequency	Negative	Frequency
1	to	1666	the	972	to	6048
2	i	1384	to	938	i	4565
3	the	975	you	913	the	4114
4	a	793	i	754	a	3188
5	you	766	for	670	flight	2943
6	jetblue	748	thanks	611	united	2899
7	united	737	jetblue	595	and	2825
8	t	732	southwestair	576	on	2792
9	on	673	a	533	you	2722
10	southwestair	671	united	528	for	2714

p.s. we can obtain top n words from the code by changing the value of n on the code.

Here, we can observe that there are lot of stop words. So, after cleaning the text top 10 (n) words were:

Rank	Neutral	Frequency	Positive	Frequency	Negative	Frequency
1	flight	769	thanks	609	flight	3337
2	united	734	jetblue	589	united	2883
3	jetblue	722	southwestair	573	usairways	2374
4	southwestair	668	united	527	americanair	2104
5	americanair	499	thank	453	southwestair	1212
6	usairways	401	flight	435	jetblue	1050
7	get	241	americanair	354	get	1011
8	need	179	usairways	276	cancelled	920
9	please	179	great	233	customer	773
10	virginamerica	176	service	162	americanair	2104

Here, we can see that the top 10 (n) words are much more meaningful.

I have already placed a screenshot of tweet length histogram as tweet length distribution.

c) Preprocessing and Feature Engineering:

After lowercasing, punctuation removal, stop words removal, and lemmatization, The table for top n words looked like above. Now, we can perform NLP tasks with meaningful words.

The Bag of Words and TF-IDF representation is done on the notebook file.

Part2: Model Building:

Although we were only required to train four models, I was bit curious about what the other models do and how it performs on out given data. I trained 3 explainable models i.e. Logistic Regression, Naive Bayes, Decision Tree, and 3 black-box models i.e. Deep Neural Network, LSTM, and BERT.

a)

I trained all the models using the text column as a feature and 80/20 train/test split and have evaluated the performance using different representations on the notebook file.

Model	Vectorizer	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)	F1-Score (Weighted)
Naive Bayes	Bag of Words	0.77	0.73	0.63	0.67	0.75
Logistic Regression	Bag of Words	0.78	0.72	0.69	0.71	0.77
Decision Tree	Bag of Words	0.69	0.62	0.51	0.52	0.64
Naive Bayes	TF-IDF	0.68	0.77	0.44	0.44	0.60
Logistic Regression	TF-IDF	0.78	0.76	0.66	0.69	0.77
Decision Tree	TF-IDF	0.69	0.61	0.53	0.55	0.66
DNN (Deep Neural Network)	Bag of Words	0.78	0.74	0.70	0.72	0.78
DNN (Deep Neural Network)	TF-IDF	0.77	0.75	0.68	0.71	0.77
LSTM	Word Sequences	0.76	0.72	0.69	0.70	0.76
BERT	Pretrained	0.85	0.82	0.77	0.79	0.84

Confusion Matrices - Bag of Words Representation

Model	**Class**	**Predicted Negative**	**Predicted Neutral**	**Predicted Positive**
Naive Bayes	Negative	1727	76	32
	Neutral	308	253	59
	Positive	160	52	261

Model	Class	Predicted Negative	Predicted Neutral	Predicted Positive
Logistic Regression	Negative	1634	144	57
	Neutral	201	354	65
	Positive	105	75	293
Decision Tree	Negative	1634	144	57
	Neutral	201	354	65
	Positive	105	75	293

Confusion Matrices - TF-IDF Representation

Model	Class	Predicted Negative	Predicted Neutral	Predicted Positive
Naive Bayes	Negative	1727	76	32
	Neutral	308	253	59
	Positive	160	52	261
Logistic Regression	Negative	1634	144	57
	Neutral	201	354	65
	Positive	105	75	293
Decision Tree	Negative	1721	56	58
	Neutral	473	86	61
	Positive	229	26	218

b) Architecture and Implementation Details:

I have used two representations for the implementation of 3 explainable models and Deep neural network. For LSTM and BERT, I used word embedding. To train the LSTM, the text data was encoded in inter and padding sequence and passed through embedding layer to

obtain embedding of the words and then fit to the LSTM model. To implement BERT, I used its own pretrained embedding.

All the observations are provided on the table above. I have used both representations, TF-IDF and Bag of Words for all three explainable models and deep neural networks whereas I used word embeddings for two of the black-box models, LSTM and BERT.

Architecture used for black-box models:
After experimenting with different layers, activation functions, dropout layers, and epochs, I used the values that resulted in best accuracy. Nevertheless, there was only minor improvement with those changed values.

Neural Network:

Model Architecture:

Layer Type	**Description**
Input	Input shape = (X_train.shape[1],)
Dense (1)	128 units, activation = ReLU
Dropout (1)	Rate = 0.4 (for regularization)
Dense (2)	64 units, activation = ReLU
Dropout (2)	Rate = 0.4 (for regularization)
Output	3 units (for 3 sentiment classes), activation = Softmax

Training configuration:

Parameter	**Value**
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Crossentropy
Metrics	Accuracy
Epochs	12
Batch Size	1000
Validation Data	(X_test, y_test)

LSTM:

Model Architecture:

Layer Type	**Description**
Embedding	input_dim = vocab_size, output_dim = 100, input_length = 22
LSTM	128 units, return_sequences = False
Dropout (1)	Rate = 0.4

Dense (1)	64 units, activation = ReLU
Dropout (2)	Rate = 0.2
Output	3 units (for 3 sentiment classes), activation = Softmax

Training Configuration:

Parameter	**Value**
Optimizer	Adam
Learning Rate	0.001
Loss Function	Categorical Cross entropy
Metrics	Accuracy
Epochs	12
Batch Size	1000
Validation Data	(X_test, y_test)

BERT

Model Architecture:

Component	**Description**
Pretrained Model	BERT base uncased (bert-base-uncased)
Classification Head	Dense layer with 3 output units (for 3 sentiment classes)
Output Activation	Implicitly uses logits (from_logits=True)

Training Configuration:

Parameter	**Value**
Optimizer	Adam
Learning Rate	2e-5
Loss Function	Sparse Categorical Crossentropy (from_logits=True)
Metrics	Accuracy
Epochs	3
Batch Size	100
Validation Data	([X_test_ids, test_mask], y_test)

While training with BERT, the validation accuracy was improving with each epoch. I was able to obtain much better accuracy than all the other models, however, due to limited computing resources, I limited the number of epochs to just 3. But with an increased number of epochs, the validation accuracy will improve.

c) Initial Observations:

Here is the performance of all the models with the representations that I used to train the model:

Model	Vectorizer	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)	F1-Score (Weighted)
Naive Bayes	Bag of Words	0.77	0.73	0.63	0.67	0.75
Logistic Regression	Bag of Words	0.78	0.72	0.69	0.71	0.77
Decision Tree	Bag of Words	0.69	0.62	0.51	0.52	0.64
Naive Bayes	TF-IDF	0.68	0.77	0.44	0.44	0.60
Logistic Regression	TF-IDF	0.78	0.76	0.66	0.69	0.77
Decision Tree	TF-IDF	0.69	0.61	0.53	0.55	0.66
DNN (Deep Neural Network)	Bag of Words	0.78	0.74	0.70	0.72	0.78
DNN (Deep Neural Network)	TF-IDF	0.77	0.75	0.68	0.71	0.77
LSTM	Word Sequences	0.76	0.72	0.69	0.70	0.76
BERT	Pretrained	0.85	0.82	0.77	0.79	0.84

Among the explainable models I used, the best performing model was Logistic Regression. The Decision tree's performance was not good with both the presentations. Naïve Bayes performed worst while using TF-IDF. The performance of Logistic Regression was best with Bag of words representation, but it was a tradeoff between precision and recall score.

After implementing black-box models at this point, the performance difference was not that significant as compared to the explainable model. The performance of LSTM and Deep Neural Network was similar. If I must choose between the black-box model and explainable model, I would choose an explainable model, logistic regression, at this point.

After implementing BERT and evaluating the performance the difference was significant. It performed much better than all the other models, explainable and the two black-box models.

Simple models perform poorly because of the imbalance in the dataset. Higher F1 score is obtained for the majority class as compared to the minority class. Specially with Naïve Bayes and Decision tree the precision and recall has high difference suggesting that the model is missing many valid cases.

Part 4: Interpretation, Error Analysis & Trade-Offs

a) Error Analysis:
I pulled 15 misclassified texts for the best performing model, BERT. The pulled texts are on the notebook file.

Some of the misclassifications were because the model didn't train well, and some may have been misclassified because of the tweet itself. For example:

Text: @JetBlue Pittsburgh
True Label: neutral
Predicted Label: positive

This might have happened because of the tweet length. There is not much meaning that can be extracted from the word vector for this tweet.

Text: @AmericanAir We've sent you more info via DM. I truly hope you resolve this very quickly. #media #filmcrew #cnn #nbc
True Label: neutral
Predicted Label: negative

This tweet is good length, but the model misclassifies this tweet. This might be because of the models' training.

Text: @AmericanAir Thanks that was done, I just don't understand why those whose bags couldn't fit aren't notified in air/at landing #technology
True Label: negative
Predicted Label: positive

This is clearly a negative tweet but the model categorized this tweet as positive because of the word 'Thanks'.

Text: @JetBlue I would fly somewhere hotter then here. Puerto Rico here I come. Lol
True Label: neutral
Predicted Label: positive

We can see in this example that the model fails to classify sarcasm as well.

Above were some of the examples but most of the tweets that the model misclassified because of the context, sarcasm and class confusion i.e. the model gave higher priority to the positive word and failed to understand the whole context.

b) Class Sensitivity and Confusion Patterns:

Per class precision, recall, and F1 score:

Model	Class	Precision	Recall	F1-score
Naive Bayes (BoW)	Negative	0.79	0.94	0.86
Naive Bayes (BoW)	Neutral	0.66	0.41	0.51
Naive Bayes (BoW)	Positive	0.74	0.55	0.63
Logistic Regression (BoW)	Negative	0.84	0.89	0.87
Logistic Regression (BoW)	Neutral	0.62	0.57	0.59
Logistic Regression (BoW)	Positive	0.71	0.62	0.66
Decision Tree (BoW)	Negative	0.71	0.94	0.81
Decision Tree (BoW)	Neutral	0.51	0.14	0.22
Decision Tree (BoW)	Positive	0.65	0.46	0.54
Naive Bayes (TF-IDF)	Negative	0.67	1.0	0.8
Naive Bayes (TF-IDF)	Neutral	0.84	0.15	0.25

Naive Bayes (TF-IDF)	Positive	0.79	0.16	0.27
Logistic Regression (TF-IDF)	Negative	0.8	0.95	0.87
Logistic Regression (TF-IDF)	Neutral	0.69	0.47	0.56
Logistic Regression (TF-IDF)	Positive	0.77	0.55	0.64
Decision Tree (TF-IDF)	Negative	0.73	0.9	0.8
Decision Tree (TF-IDF)	Neutral	0.47	0.28	0.35
Decision Tree (TF-IDF)	Positive	0.65	0.4	0.49
DNN (TF-IDF)	Negative	0.84	0.89	0.86
DNN (TF-IDF)	Neutral	0.65	0.52	0.58
DNN (TF-IDF)	Positive	0.76	0.64	0.69
DNN (BoW)	Negative	0.85	0.89	0.87
DNN (BoW)	Neutral	0.64	0.54	0.58
DNN (BoW)	Positive	0.74	0.67	0.7
LSTM	Negative	0.84	0.86	0.85
LSTM	Neutral	0.58	0.58	0.58
LSTM	Positive	0.73	0.62	0.67
BERT	Negative	0.88	0.95	0.91

BERT	Neutral	0.74	0.66	0.7
BERT	Positive	0.85	0.71	0.77

Visualization of Confusion matrices:

Confusion Matrices - Bag of Words Representation

Model	**Class**	**Predicted Negative**	**Predicted Neutral**	**Predicted Positive**
Naive Bayes	Negative	1727	76	32
	Neutral	308	253	59
	Positive	160	52	261
Logistic Regression	Negative	1634	144	57
	Neutral	201	354	65
	Positive	105	75	293
Decision Tree	Negative	1634	144	57
	Neutral	201	354	65
	Positive	105	75	293

Confusion Matrices - TF-IDF Representation

Model	**Class**	**Predicted Negative**	**Predicted Neutral**	**Predicted Positive**
Naive Bayes	Negative	1727	76	32
	Neutral	308	253	59
	Positive	160	52	261
Logistic Regression	Negative	1634	144	57
	Neutral	201	354	65

Model	Class	Predicted Negative	Predicted Neutral	Predicted Positive
Decision Tree	Positive	105	75	293
	Negative	1721	56	58
	Neutral	473	86	61
	Positive	229	26	218

Confusion matrices for Black Box models:

Model: DNN (TF-IDF)

	Negative	Neutral	Positive
True Negative	1667	119	49
True Neutral	239	331	50
True Positive	115	54	304

Model: DNN (BOW)

	Negative	Neutral	Positive
True Negative	1670	103	62
True Neutral	252	314	54
True Positive	104	54	315

Model: LSTM

	Negative	Neutral	Positive
True Negative	1683	106	46
True Neutral	263	315	42
True Positive	119	75	279

Model: BERT

	Negative	Neutral	Positive
True Negative	1667	119	49
True Neutral	239	331	50
True Positive	115	54	304

Per class precision, recall, and f1 score is given in the table above and confusion matrices are also visualized in the table after that.

We can observe that the precision and recall for the neutral and positive class is lower than the negative class because of the imbalance in the data. The neutral class was the hardest

for the model to distinguish. It confuses with positive and negative. This may be because of the negative and positive words in the text that are confused by the model. Positive class has better value for precision than for recall in most of the models. There were more false negatives. This suggests that it's hard to correctly classify positive tweets. F1 scores are better in all the models for negative class. Among these models BERT does the best job while classifying minority class as well as confusing class.

c) As we can see from the performance evaluation, the explainable models perform like the black box model. The performance difference is not that significant with Logistic Regression compared to LSTM or Deep Neural Network. I would say that it is not worth to compromise the explainability. However, BERT performs significantly better than all the other models and it is possible to obtain even higher accuracy and F1 scores for minority classes with a higher number of epochs given we have enough computing resources. Therefore, I would say that given the results, it is worth the loss in interpretability as the model is performing way better than the explainable models.