My models for competition

WorkFlow 1.

This workflow was focused on more simple preprocessing and more conventional models.

1. Preprocessing

1.1. Type 1.

After loading the data, I noticed that twitter texts are stored in json format, which was not very convenient to operate in a python environment. Later, I had converted .json to dataframe. Although it became easier to understand the file, the tweet_df['_source'] variable was still messy.

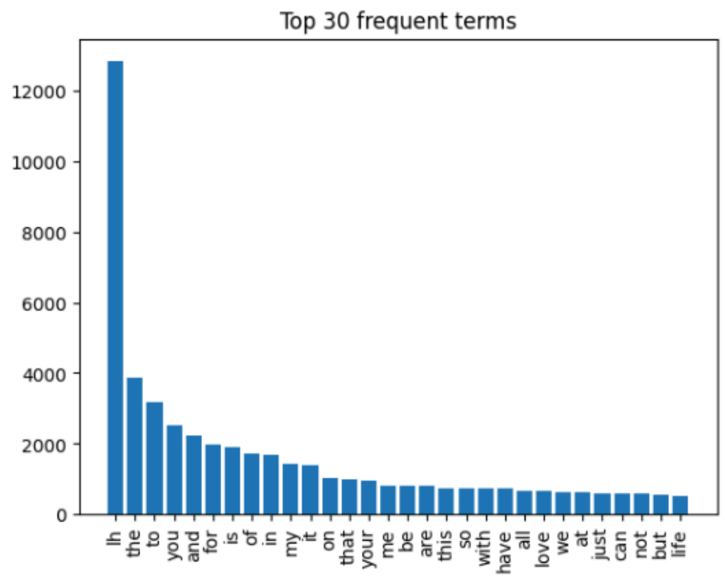
To resolve it, I had regrouped it by lambda into several variables that it included such as tweet, tweet_id, text and hashtags. After these transformations, I noticed that some of the variables as "_index", "_source", "_type", "tweet" are not very informative, so I decided to drop them. I kept the rest of the variables, in case I might need them later.

After that, I merged the dataset with identification to train or test, splitted them into train and test sets and assigned them to the train emotions labels. Finally, I used a label encoder to transform labels into categorical variables. The training dataset was of size 1455563 rows × 6 columns.

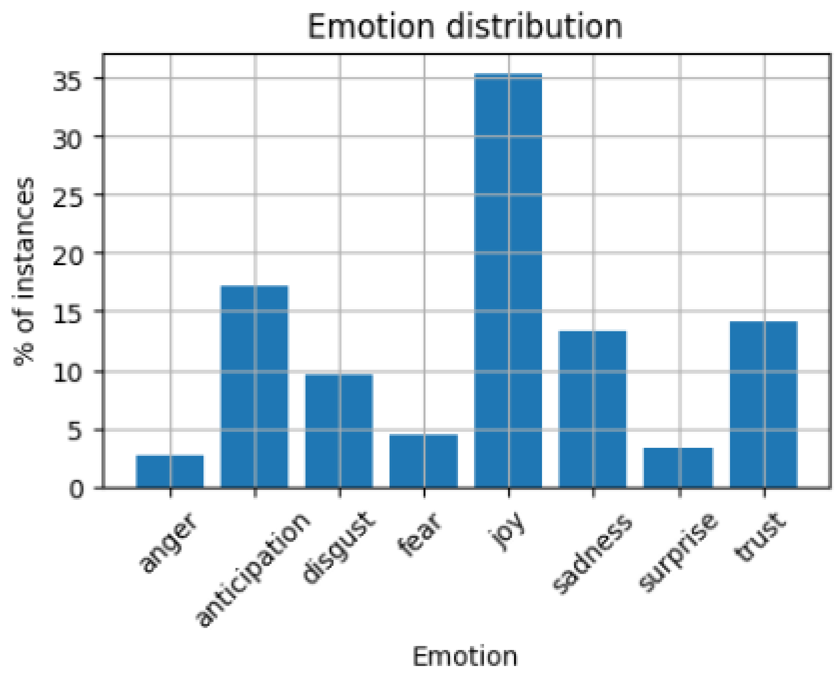
2. Simple EDA

I did a simple EDA to gain some hints and insights about the data.

Firstly, I examined the top 30 most frequent words at a baseline. For it, CountVectorizer was applied to the random sample of 5000 words from the dataset.



To understand the target variable class distribution, I get the count by each group of the emotion.



There are 8 emotions in total that we need to predict. It is noticeable that joy was the most prevalent emotion in the target variable, while anger was the least prevalent one.

3. Feature Extraction

3.1. TFIDF (n=500)

Firstly, I was interested to see how extracting 500 TFIDF features will allow me to predict the emotions.

4. Modeling

4.1. XGBoost

As the first try for modelling, I tried the XGBoost classifier. XGBoost is famous for its use in kaggle competitions, and this was my rationale to give it firstly a go.

Firstly, I trained my model on a baseline settings, such as

```
model = XGBClassifier(
    max_depth=5,
    n_estimators=100,
    learning_rate=0.1
)
```

Because XGBoost might highly benefit from hyperparameter tuning, I tuned it before submitting for the predictions.

For hyperparameter tuning, I utilized a randomized search approach with number of iterations=5 and cv=5. As the dataset is huge, I performed the search of hyperparameters on a random sample, which makes 0.2 fraction of the all dataset.

As a candidates, I have set these hyperparameters:

```
hyperparameters = {
    'max_depth': [2,3,4],
    'n_estimators': [100, 150, 200],
    'learning_rate': [0.01,0.05, 0.2],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.6, 0.8],
    'colsample_bytree': [0.6, 0.8, 1.0]
}
```

As a result, the Best Parameters and Score were as on the table below.

Best parameters	Best F1-score
'subsample': 0.6, 'n_estimators': 200, 'min_child_weight': 5, 'max_depth': 2, 'learning_rate': 0.2, 'colsample_bytree': 1.0	0.27778712481205925

After submission to the kaggle, tuned XGBoost Classifier model had achieved F1-score = 0.29797.

4.2. Random Forest Classifier

As the classes were highly imbalanced in the target variable, I was curious to apply Random Forest Classifier to the same set of TFIDF (n=500) extracted features.

Firstly, I fitted the baseline model. Then, I performed the hyperparameter tuning on the random sample with fraction=0.2 of the all dataset.

Below are the candidates that I used to find the best model by Randomized Search:

```
hyperparameters = {
    'n_estimators': [100, 200, 300],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [2, 3, 5],
    'max_depth': [None, 2, 3],
    'bootstrap': [True, False]
}
```

The results demonstrated that the best candidates were with the best score as below.

Best Hyperparameters	Best F1-score
'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_depth': None, 'bootstrap': False	0.3088039831945576

WorkFlow 2.

This workflow was focused on more simple preprocessing and more conventional models.

1. Preprocessing

1.1. Type 2.

As I wanted to check if simple preprocessing (type 1) will give us good results on the baseline, I realized that this text was having some problems.

For this type of preprocessing, I first will follow all of the steps from Type 1 preprocessing.

After loading the first 5 records, the following problems were encountered:

The text is filled with:

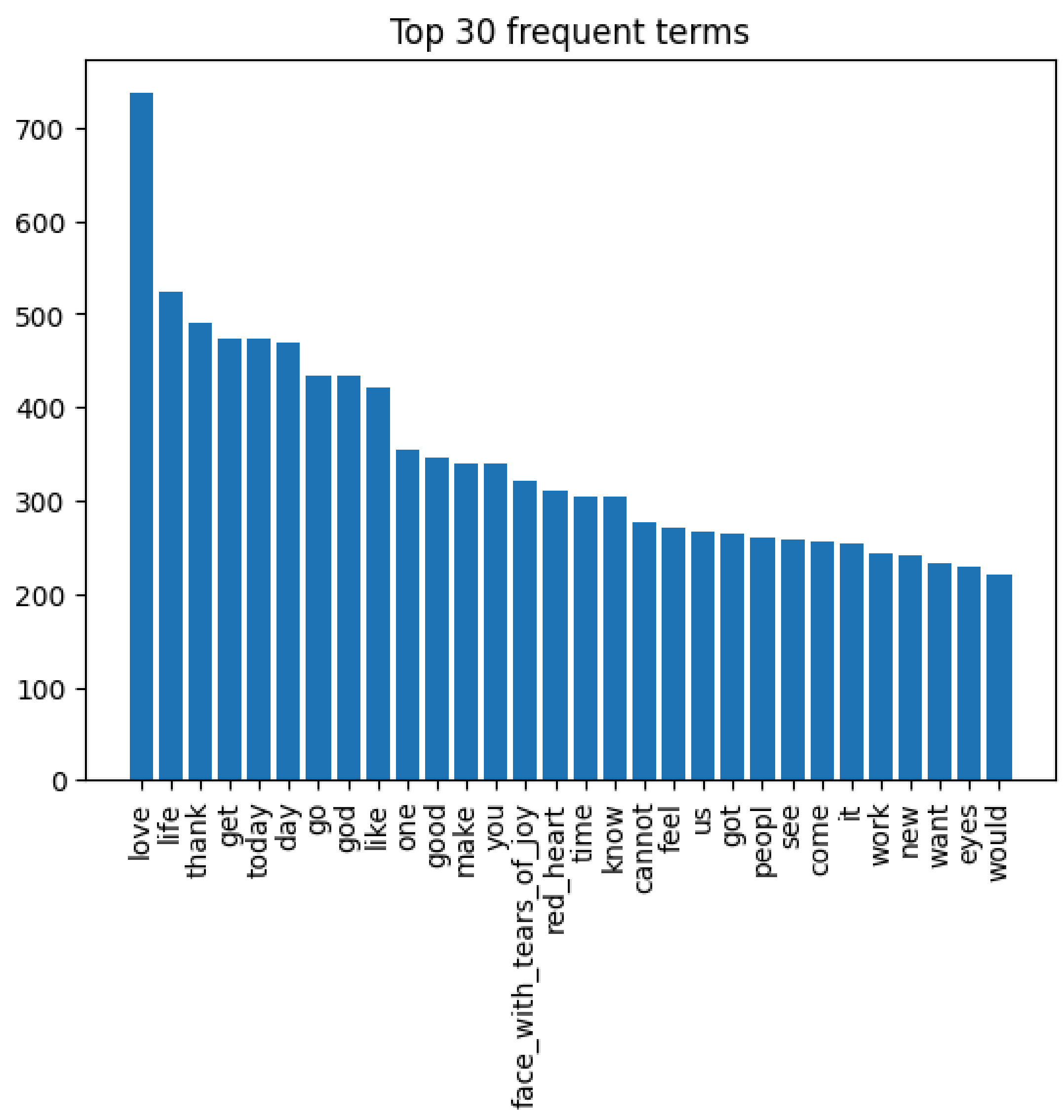
a. LH, I am not sure what it is, but seems redundant
b. Emojis
c. there are urls and mentioning of other people
d. some word shortenings like aren't or doesn'
e. upper letters might affect the word count

Therefore by using str.replace methods, list comprehension or specialized packages for addressing these issues, such as emoji or contractions, I addressed these issues successfully.

I also kept in mind the class example of many different word forms in Spanish for running. I considered that a good idea to apply PorterStemmer and remove stopwords at the same time to the text corpus.

2. Simple EDA

After applying Type 2 pre processing and visualising top 30 words, we have got a different word distribution



3. Feature extraction

3.1. TFIDF (n=1000)

Before we extracted only 500 features this way. Now, I was interested to see how extracting 1000 TFIDF features will allow me to predict the emotions.

3.2. Bag of Words (n=500)

Additionally, I was curious to extract the features by using the Bag of Words approach.

3.3. Tokenization and Word Embeddings

This is another way, how I extracted the features for training the RNN model. This seems to be the most effective way to do so, as a token I indicated oov_token="<OOV>". Our vocabulary will comprise 20000 words because the number of tweets exceeds one million, I suppose this is a reasonable number. Considering the situation, where a tweet is too long, I set max_len = 100 and each word will be mapped to a vector of 50 continuous values.

4. Modeling

4.1. Random Forest Classifier

Surprisingly, when I fitted Random Forest Classifier to TFIDF features (n=1000) extracted on a Type 2 preprocessed data, the accuracy dropped down significantly after submitting to the kaggle competition to F1-score = 0.22684.

This happened likely due to the fact that my preprocessing removed something meaningful that the model was able to learn before or due to the high sparsity of the features matrix, where random forest algorithms may not excel.

When I tried to feed BoW features (n=500) to this classifier, the accuracy had reached only 0.32, which was not bad, but also not too good.

4.2. Light GBM Classifier

I had a new idea, what if I can combine these features (TFIDF, n=1000 + BoW, n=500) for predicting emotions? But I might need to change the model type because right now my data is super sparse. Light GBM seems to possess the ability to handle sparse datasets. Therefore, I chose it as my next prediction model.

Firstly, I set the baseline hyperparameters for model training as follows.

```
lgb = LGBMClassifier(
    boosting_type='gbdt',
    num_leaves=31,
    learning_rate=0.1,
    n_estimators=300,
    class_weight='balanced',
    random_state=42
)
```

The result surprised me because it gave me the best F1-score so far = 0.35710.

However, after hyperparameter tuning and testing on validation test sets, I did not observe huge improvement, hence I decided to try deep learning algorithms.

4.3. Recurrent Neural Networks (RNN)

RNN have demonstrated their potential in text-related prediction tasks, such as sentiment analysis, language modeling and even translations. Therefore, I decided to try this model for emotion prediction from our twitter data.

4.3.1. RNN with 2 Bidirectional LSTM Layers.

This was a baseline model, where I decided to experiment with 2 bidirectional LSTM layers.

The architecture of the model was as follows:

```
Embedding(vocabulary_size, embedding_dim),
Bidirectional(LSTM(128, return_sequences=True)),
Bidirectional(LSTM(64, return_sequences=False)),
Dropout(0.5),
Dense(128, activation='relu'),
Dense(8, activation='softmax')
```

Why was it like this? Because I thought that if we added more LSTM layers, it would be for the better. I set dropout - 0.5 to prevent overfitting and the last dense layer has 8 units because we have 8 emotions to predict.

By fitting the model within 10 epochs, it already improved my F1-score and gave me hope. Submission in a Kaggle platform resulted in 0.36753.

4.3.2. RNN with 2 Bidirectional LSTM Layers and tuning to address the class imbalance

I also tried to address the class imbalance by applying to RNN model Focal loss function, which is particularly designed to address the inherited class imbalance. However, that does not improve the score.

4.4. RNN with LSTM Layer + attention layer

Is attention really what we need? Seems so, as this approach was giving me yet the best scores.

Firstly, I use the same architecture as above but only with one Bidirectional(LSTM(128, return_sequences=True)). Then, I added the attention layer to that.

However, with 10 epochs of training, it did not improve the performance dramatically.

4.5. RNN with 2 Bidirectional LSTM Layers + attention layer

This was the best model so far. The final architecture comprise of two Bidirectional LSTM layers and one attention layer as below:

```
x = Embedding(vocabulary_size, embedding_dim)(model_input)
x = Bidirectional(LSTM(128, activation='tanh', return_sequences=True))(x)
x = Bidirectional(LSTM(64, activation='tanh', return_sequences=True))(x)
attention = Attention()([x, x])
```

```
x = GlobalAveragePooling1D()(attention)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu')(x)
x = Dense(8, activation='softmax')(x)
```

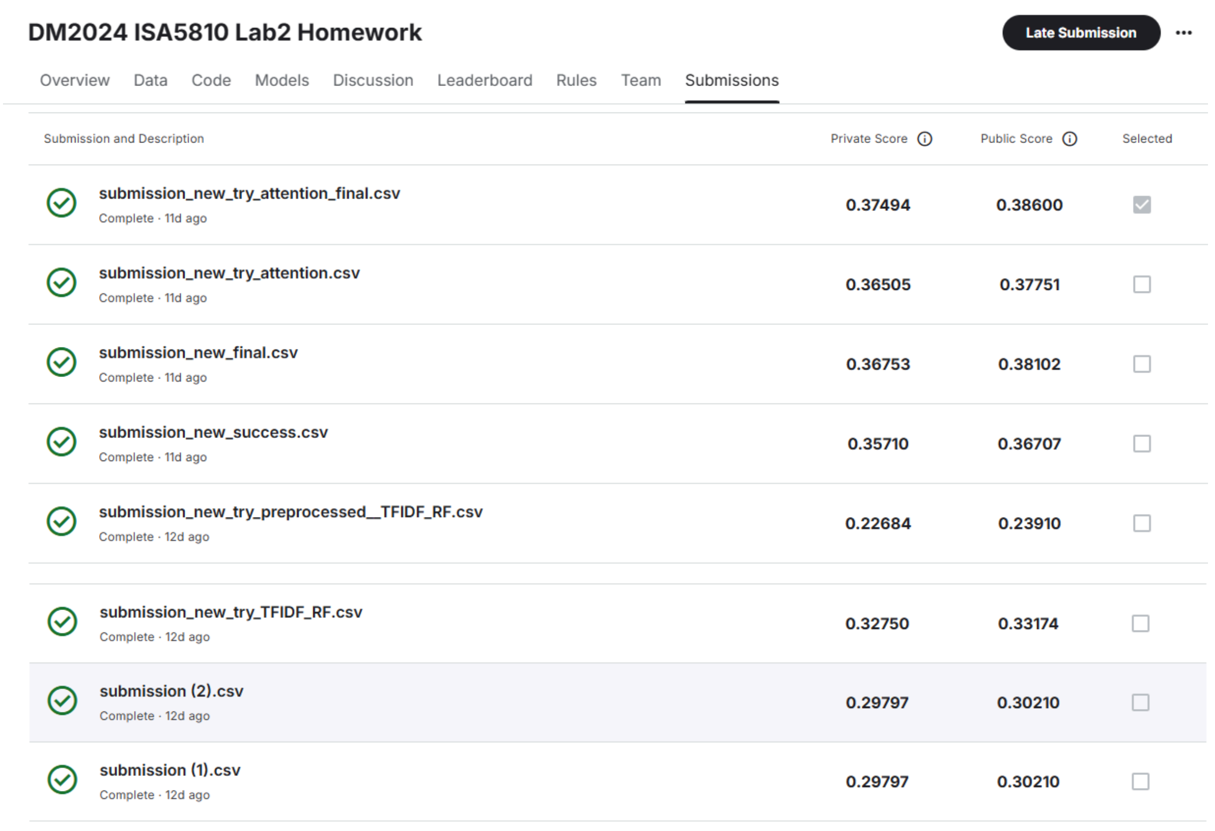
However, as I came up with this model almost close to the deadline, I only was able to run over 3 epochs. Yet, the best F1-score was achieved as 0.37494.

The model summary for this competition:

Try #N	Model	Hyperparameters	Features	Preprocessing type	F1-score
1	XGBoost Classifier	'n_estimators': 200, 'min_child_weight': 5, 'max_depth': 2, 'learning_rate': 0.2, 'colsample_bytree': 1.0 objective='multi:softmax', num_class=8, use_label_encoder=False	TFIDF (n=500)	Type 1	0.29797
2	Random Forest Classifier	'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_depth': None, 'bootstrap': False	TFIDF (n=500)	Type 1	0.32750
3	Random Forest Classifier	'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_depth': None, 'bootstrap': False	TFIDF (n=1000)	Type 2	0.22684
4	LGBM Classifier	boosting_type='gbdt', num_leaves=31, learning_rate=0.1, n_estimators=300, class_weight='balanced', random_state=42	TFIDF (n=1000) + BoW (n=500)	Type 2	0.35710
5	RNN (LSTM+LSTM)	Embedding (vocabulary_size, embedding_dim), Bidirectional(LSTM(128, return_sequences=True)), Bidirectional(LSTM(64, return_sequences=False)), Dropout(0.5), Dense(128, activation='relu'), Dense(8, activation='softmax')	Tokenization and Word Embeddings	Type 2	0.36753
6	RNN (LSTM+atte	Same as above but only one LSTM layer and added attention	Tokenization and Word	Type 2	0.36505

	ntion)		Embeddings		
7	RNN(LSTM +LSTM+atte ntion)	x = Embedding(vocabulary_size, embedding_dim)(model_input) x = Bidirectional(LSTM(128, activation='tanh', return_sequences=True))(x) x = Bidirectional(LSTM(64, activation='tanh', return_sequences=True))(x) attention = Attention()([x, x]) x = GlobalAveragePooling1D()(attention) x = Dropout(0.5)(x) x = Dense(64, activation='relu')(x) x = Dense(8, activation='softmax')(x)	Tokenization and Word Embeddings, but here reduced the vocabulary_si ze to 10000	Type 2	0.37494

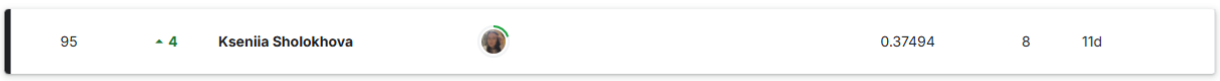
Submission to kaggle screenshots:



DM2024 ISA5810 Lab2 Homework

Submission and Description	Private Score	Public Score	Selected
submission_new_try_attention_final.csv Complete · 11d ago	0.37494	0.38600	✓
submission_new_try_attention.csv Complete · 11d ago	0.36505	0.37751	☐
submission_new_final.csv Complete · 11d ago	0.36753	0.38102	☐
submission_new_success.csv Complete · 11d ago	0.35710	0.36707	☐
submission_new_try_preprocessed__TFIDF_RF.csv Complete · 12d ago	0.22684	0.23910	☐
submission_new_try_TFIDF_RF.csv Complete · 12d ago	0.32750	0.33174	☐
submission (2).csv Complete · 12d ago	0.29797	0.30210	☐
submission (1).csv Complete · 12d ago	0.29797	0.30210	☐

The ranked score:



95	▲ 4	Kseniia Sholokhova	0.37494	8	11d

Insights

1. Emotions are hard to predict for machines, but how about humans? The first tweet is already left me wonder:

-People who post \"add me on #Snapchat\" must be dehydrated. Cuz man.... that's <LH - labeled as anticipation.

Why is that anticipation? I might say it is a joy or anger. What is there to anticipate by this tweet? I can not understand.

Therefore, it is no surprise to me why machines fail to get these classes correctly. It made me wonder, how did we originally come up with these labels? Do we have a label noise?

2. Some emotions can be described in a positive or negative way at the same time. Surprise can be a good emotion or a bad emotion depending on the context. It also applies if a person tweets something in a sarcastic way, then it produces more false positives to the joy class. This is a complex human nature.

3. RNN with an attention layer seems to give the best performance. However, when I came up with this model I was close to the end of competition and could only run it within three epochs. If I had run it over more epochs, I believe I would have gotten better results. I really believe in that.

4. I am a slow learner and I always try to get the perfect code and perfect submission. Therefore, although I finished this assignment and polished the code lines on time, I did not notice that I needed to create a pdf file describing my report. I think I learned a lot from this experience, my fixation on getting things perfect is a greedy algorithm, but it does not guarantee the global optimum. Just better to get things done, even if they are not perfect but on time.

5. I wish I could try more models, like transformers and generating LLM embeddings to try to predict the emotions. They might have had a better performance.

6. Some of the methods I tried are computationally expensive, generating TFIDF =1000 features and using these features for Random Forest Classifier to predict emotions takes a long time for running. I apologize for that.

7. Most importantly, I learned a lot from this experience. Thank you, the teacher and TAs.