Machine Learning Technologies Coursework

jh11g20@soton.ac.uk — John Holman

26th January 2024

This document details the planning, pre-processing, implementation, and evaluation of a machine learning model that was to be trained to identify mis-informative tweets with a given dataset.

1 Introduction and Data Analysis

From importing the data and reading the spec, it was immediately clear that the task was a supervised classification problem. The dataset was provided as a tab delimited .txt file which looking through, meant there were some parsing issues and things that would need to be sorted out or removed from the set.

Exploring the dataset revealed that despite being binary classification, the labels included three options and were in fact categorical. This then presented a choice as to how to proceed, either by attempting to train a model with the 3rd "humor" label option, removing entries with the humor label, or by turning all humor labels into fake labels.

Checking the types of the features showed that almost all of them were string-based and would require formatting and vectorization. The only two that were not strings were the tweetID and the userID.

I noticed that the imageId(s) were all of a uniform format and this lead me to believe they were names given by the researchers who created the dataset. It would need to at the very least be modified as the word 'real' and 'fake' was contained within and would poison the dataset if left there.

In total, the raw dataset contained 14,277 entries and had 7 features (including the target feature)

Of the labels, 6742 were fake, 4921 were real, and 2614 were humor.

2 Pipeline Design

I decided it was preferable to rename 'humor' as fake given that the humor options were still recognised as fake when the data was collected, but it was identified that the tweets were not trying to pass off the false information as fact[1]. Beyond this, the positive value for the binary classifier was 'fake' so therefore having more data available to train the model on identifying fakes was preferable in my view.

I also kept the idea of potentially removing instances of data with the humor label to see whether it affects the model as it might have been useful if the model did not train effectively.

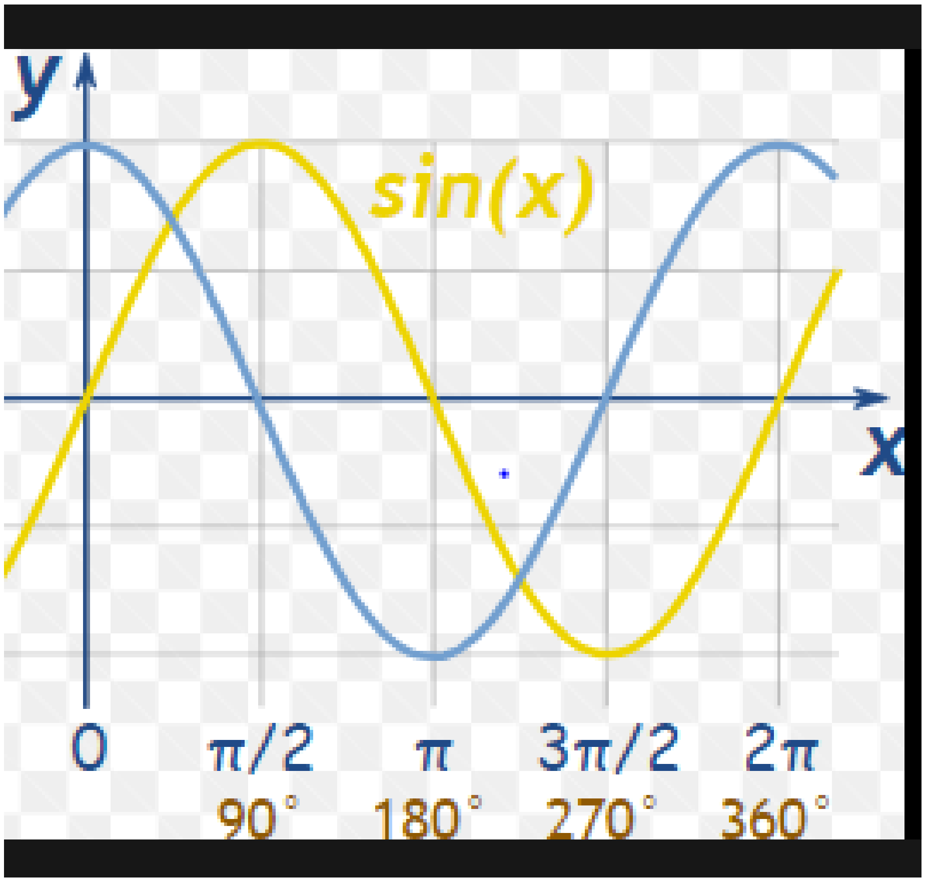
2.1 Time

The first step to turning the text data to numerical data was to look at the timestamp. It's in a unique text format and so I formatted the feature to turn them into pandas dateTime values. There was a parsing error however, due to an entry having a space character between hours, minutes, and seconds. I then rectified this and it worked.

The next thing was to turn the dateTime value into something that a machine learning algorith could use. Something numerical.

I looked at the spread of the dates and saw the vast majority of entries were in 2012, with a few in 2013 and 2014, so I felt that year wouldn't be useful especially given that future data could have a different year. However, I believed that the data could have a periodicity on a time of day, and time of week basis.

I converted the dateTime object into seconds, and then made features for a sine and cosine value for the time of week and the time of day.



This was better than simply using a number like the hour, as it would mean that the algorithm would be able to tell that 23:55 is very near to 00:05, and it was better than simply using only one wave, as then the algorithm essentially wouldn't be able to tell AM from PM or the start of the week from the end of it, by combining the two you end up with a very solid value for time in a given periodicity that is repeated enough that there should be an adequately sizeable volume of data concerning it. I originally did this for time of year also, but noticed there was not enough data to cover a periodicity there so left it with time of week and time of day in the hopes that trends would be useful such as perhaps bot accounts being more active at night.

2.2 IDs

I originally wanted to try to use the twitter API to gather extra data on the users and tweets such as likes, retweets, interactions, followers, average interactions for the user, etc. However the spec specified that task relevant data not be retrieved. Given this, I decided to remove the tweetId and userId as they didn't have any more information to really use.

The next thing to focus on was language detection and translation as the data included many languages.

2.3 Language detection

I chose lingua as a language detector because of how accurate it is (at the cost of speed, but that didn't matter as it would only need to be run once) and its ability to give confidence scores for detection. I reasoned that confidence scores would naturally go down with badly worded messages or ones with poor spelling which could be useful.

Before translating though, the text needed to be cleaned, and so I cleaned the tweet texxt bodies to remove @ mentions, hashtags, hyperlinks, newlines, and email addresses. This took a few iterations as I dealt with the issues caused by typos and poor formatting.

2.4 Extracting web-relevant parts from text

Once that was done, all the text was analysed and language detected. In addition to confidence scores, I added the character length of the tweet and a value of the score divided by the length as an extra variable because confidence scores went down a lot with shorter text.

Then I extracted the mentions, as a feature, the hashtags, and the links, of each entry.

To simplify things, I changed the mention feature to simply be a binary option between 'did mention someone' or 'didn't mention anyone'. And for hashtags I simply put down how many hashtags were included.

2.5 Text translation

Next was translating the text. I decided to translate the text, as removing non-english posts would have removed over 30 percent of the entries, and the tokenising tools readily available (nltk, and keras) did not have full support for all languages.

I used the python deep translate library to access the google translate API and translated all of the text.

I also then changed the categorical/text based value for predicted country into a binary option for wether the text was natively english or translated. This was mainly for simplicity and because there was not enough data points for most of the languages to warrant specifying.

Google translate didn't support all languages and so I removed all entries that had a recognised language that was not in google's list of supported languages. (and for chinese, changed it to specify simplified as I believe that is more common, particularly on the internet.)

I left in punctuation for now because translation apis, particularly lingua, were known for struggling to understand text without punctuation. Though I would eventaully remove it.

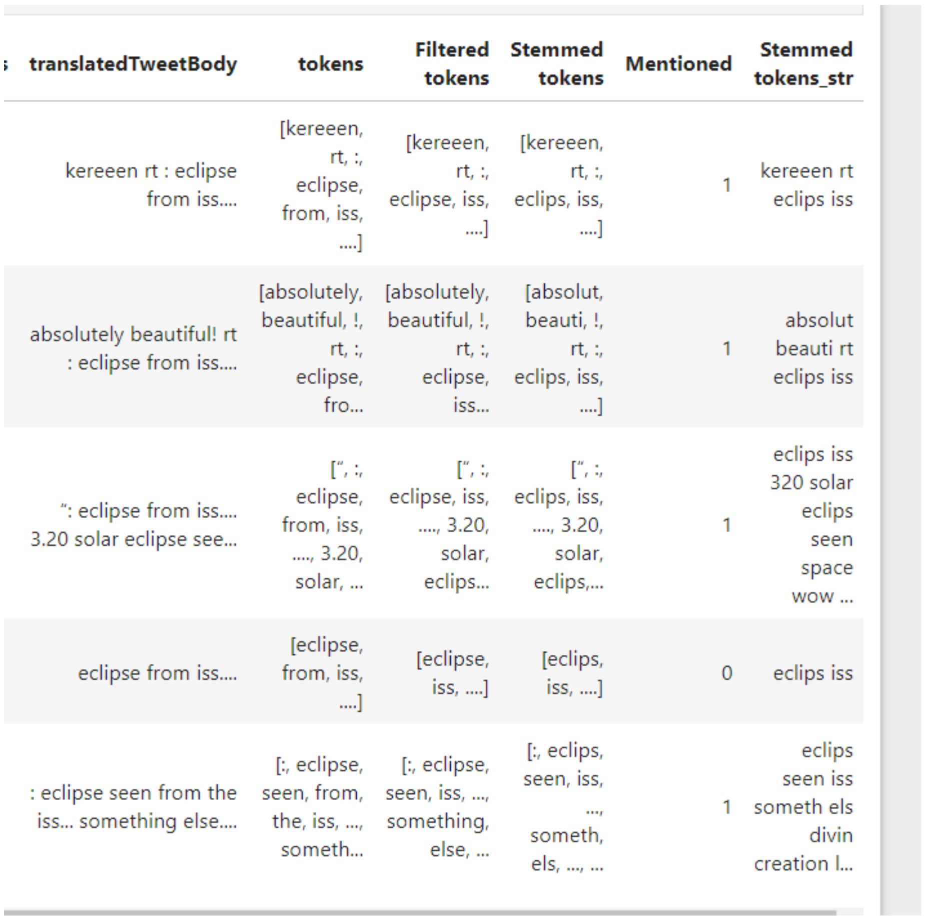
2.6 Natural Language Processing

Next, was time to process the text properly. I moved everything to lowercase as I didn't believe much information would be lost given how on the internet people often don't adhere to rules like that.

then, using NLTK I tokenised the text, removed stopwords from the stopwords provided by nltk. and then stemmed the tokens as I wanted to reduce how many potential tokens might be used, and for short-form internet text, keywords matter more than the sentence as a whole.

I then did some dimensional reduction to reduce the number of features, I didn't see a viable way to process the username in time so it went, along with any older features left from older formatting. I also removed the links as there was no way to retrieve much meaningful data without going outside the spec and looking at the images.

I then took the tokens that had been produced, and turned them back into a string, removing all non-alphanumeric characters. This was because the vectorisation function I wanted to use only took text, however I still wanted to use nltk for the tokenisation, stopwords, and stemming so that I could control the process more.



translatedTweetBody	tokens	Filtered tokens	Stemmed tokens	Mentioned	Stemmed tokens_str
kereeen rt : eclipse from iss....	[kereeen, rt, :, eclipse, from, iss, ...]	[kereeen, rt, :, eclipse, iss, ...]	[kereeen, rt, :, eclips, iss, ...]	1	kereeen rt eclips iss
absolutely beautiful! rt : eclipse from iss....	[absolutely, beautiful, !, rt, :, eclipse, fro...	[absolutely, beautiful, !, rt, :, eclipse, iss...	[absolut, beauti, !, rt, :, eclips, iss, ...]	1	absolut beauti rt eclips iss
“: eclipse from iss.... 3.20 solar eclipse see...	[“, :, eclipse, from, iss, ..., 3.20, solar, ...	[“, :, eclipse, iss, ..., 3.20, solar, eclips...	[“, :, eclips, iss, ..., 3.20, solar, eclips,...	1	eclips iss 320 solar eclips seen space wow ...
eclipse from iss....	[eclipse, from, iss, ...]	[eclipse, iss, ...]	[eclips, iss, ...]	0	eclips iss
: eclipse seen from the iss... something else....	[:, eclipse, seen, from, the, iss, ..., someth...	[:, eclipse, seen, iss, ..., something, else, ...	[:, eclips, seen, iss, ..., someth, els, ..., ...	1	eclips seen iss someth els divin creation l...

2.7 Vectorisation

For vectorisation, I trained a TF-IDF vectoriser. It is a form of bag of words vectorisation, however instead of counting occurences, it gives a float value for how unique the word is in the text, 1 being unique, 0 being the most common. Given the 280 character limit of tweets, as well as the general style of communication on the internet, it made sense to use a bag of words method as there often isn't enough data there to get much meaningful semantics from fully processing the data as a sentence with greater context given to the words.

Initially I had been trying to use NLTK to do the vectorisation however it did not have any of the tools that I required and so it made sense to use the TF-IDF.

This vectoriser was able to identify about 6500 tokens. This was far too many for the given dataset and would have given the model the curse of dimensionality. So I sought to do some feature selection to identify which tokens were the most important.

Before the selection, I went and standardise the features that needed it, namely 'Confidence/Length', and 'HashCount'. The rest of the values would have lost their meaning through standardisation as they were either binary 1 or 0 values or the sin and cosine waves which already ran from a range between -1 and 1.

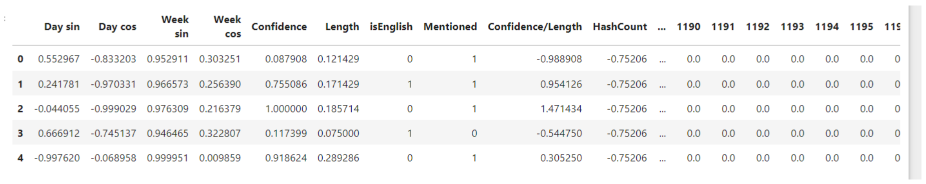
I also separated the labels to a different dataframe so that it wouldn't be used in the training accidentally.

2.8 Feature Selection for Vectors

Initially, I attempted to use the SelectKBest model, set to classification on the whole dataset. And it interestingly enough found only some of the manual features like 'Day sin', 'week sin', 'week cos' 'translation confidence'. 'length' and 'is english' as some of the best, along with a load of the tokens from the vector.

Instead, I decided to keep all the features that I had manually done as there weren't many anyway. And to do the feature selection on the token vectors, to get the best 500 of them. 500 seemed appropriate given the dataset length of 14000. General advice seems to say that a rule of thumb is there should be a minimum of 10 entries per feature, so this meant ideally I shouldn't have been going above 1400 features.

The final format of the dataset ended up being this:



	Day sin	Day cos	Week sin	Week cos	Confidence	Length	isEnglish	Mentioned	Confidence/Length	HashCount	...	1190	1191	1192	1193	1194	1195	119
0	0.552967	-0.833203	0.952911	0.303251	0.087908	0.121429	0	1	-0.988908	-0.75206	...	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.241781	-0.970331	0.966573	0.256390	0.755086	0.171429	1	1	0.954126	-0.75206	...	0.0	0.0	0.0	0.0	0.0	0.0	0
2	-0.044055	-0.999029	0.976309	0.216379	1.000000	0.185714	0	1	1.471434	-0.75206	...	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.666912	-0.745137	0.946465	0.322807	0.117399	0.075000	1	0	-0.544750	-0.75206	...	0.0	0.0	0.0	0.0	0.0	0.0	0
4	-0.997620	-0.068958	0.999951	0.009859	0.918624	0.289286	0	1	0.305250	-0.75206	...	0.0	0.0	0.0	0.0	0.0	0.0	0

Next involved selecting a machine learning algorithm. Firstly, I wanted to train a neural network because of their high level of ability to notice subtleties in the dataset, their potential for fine-tuning, and their ability to use validation to improve the results.

The next option for me was most likely between SVM and logistic regression as they are natively designed for binary classification.

I ended up going with SVM. This was due to it generally handling unstructured and semi-structured data a bit better, and given that the dataset was almost exclusive text, the SVM was able to play to its strengths. Additionally, it is less prone to overfitting [2] It was set up as standard. With both an accuracy score and an F1 score.

The neural network was initially set up with one hidden layer. Since there were 510 features, the first layer had 510 neurons, it used relu activation. It had one hidden layer of about 250 neurons, relu again. then finally an output layer of 1 neuron and a sigmoid function.

3 Evaluation

3.1 Adding more features

The first thing I realise that might have been an issue was not enough features. 510 seemed somewhat low and the initial score gotten back were not great, so I redid the dataset using 1200 feature selected vectors making the feature count 1210. This seemed to improve things, particularly with the SVM.

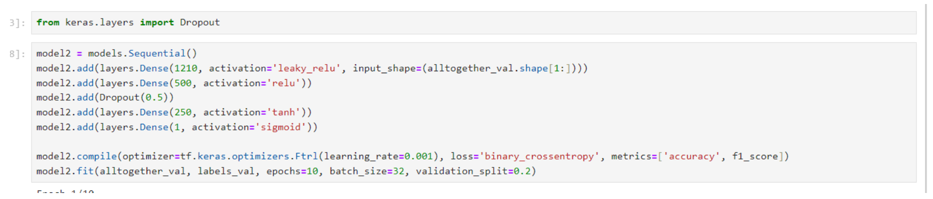
3.2 Initial comparisons

As I tinkered with the models I found something interesting. Which was that the neural network ended up performing a fair bit amount worse than the SVM. The SVM was able to achieve an F1 Score of 0.768125132107377 on the test dataset. A score that was higher than its accuracy.

Howeve the neural network had some issues, I tried many many many different configurations, different activation functions, layers, regularization, dropout, changing the threshold for fake or real. Changing the output function, changing the learning rate. All of it still had the f1 score be low, around 0.5 both with test and training. Despite this, it had good accuracy, which I can summarise is likely due to the data being skewed towards fakes given the fakes were the largest volume of data even without including humor. And beyond that it seemed like the network was overfitting. Despite trying regularisation and dropout, the final f1 score would never go much above 0.5.

3.3 Neural network solution

Then, the one thing that worked was changing the optimization function to Ftrl.

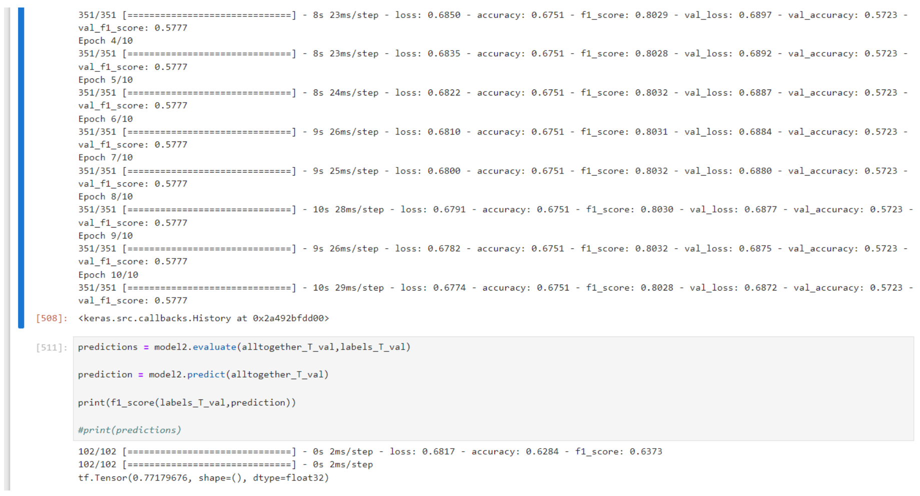


```
3]: from keras.layers import Dropout

8]: model2 = models.Sequential()
    model2.add(layers.Dense(1210, activation='leaky_relu', input_shape=(alltogether_val.shape[1:])))
    model2.add(layers.Dense(500, activation='relu'))
    model2.add(Dropout(0.5))
    model2.add(layers.Dense(250, activation='tanh'))
    model2.add(layers.Dense(1, activation='sigmoid'))

    model2.compile(optimizer=tf.keras.optimizers.Ftrl(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy', f1_score])
    model2.fit(alltogether_val, labels_val, epochs=10, batch_size=32, validation_split=0.2)
```

This did not help the accuracy much but the f1 score for the training was good at about 0.8. The validation f1 score was 0.577, however, when testing it on the test set, the f1 score was at 0.77179676 which was by far the highest it had gotten.



```
351/351 [==============================] - 8s 23ms/step - loss: 0.6850 - accuracy: 0.6751 - f1_score: 0.8029 - val_loss: 0.6897 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 4/10
351/351 [==============================] - 8s 23ms/step - loss: 0.6835 - accuracy: 0.6751 - f1_score: 0.8028 - val_loss: 0.6892 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 5/10
351/351 [==============================] - 8s 24ms/step - loss: 0.6822 - accuracy: 0.6751 - f1_score: 0.8032 - val_loss: 0.6887 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 6/10
351/351 [==============================] - 9s 26ms/step - loss: 0.6810 - accuracy: 0.6751 - f1_score: 0.8031 - val_loss: 0.6884 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 7/10
351/351 [==============================] - 9s 25ms/step - loss: 0.6800 - accuracy: 0.6751 - f1_score: 0.8032 - val_loss: 0.6880 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 8/10
351/351 [==============================] - 10s 28ms/step - loss: 0.6791 - accuracy: 0.6751 - f1_score: 0.8030 - val_loss: 0.6877 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 9/10
351/351 [==============================] - 9s 26ms/step - loss: 0.6782 - accuracy: 0.6751 - f1_score: 0.8032 - val_loss: 0.6875 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
Epoch 10/10
351/351 [==============================] - 10s 29ms/step - loss: 0.6774 - accuracy: 0.6751 - f1_score: 0.8028 - val_loss: 0.6872 - val_accuracy: 0.5723 -
val_f1_score: 0.5777
[508]: <keras.src.callbacks.History at 0x2a492bfdd00>
```

```
[511]: predictions = model2.evaluate(alltogether_T_val,labels_T_val)

       prediction = model2.predict(alltogether_T_val)

       print(f1_score(labels_T_val,prediction))

       #print(predictions)

102/102 [==============================] - 0s 2ms/step - loss: 0.6817 - accuracy: 0.6284 - f1_score: 0.6373
102/102 [==============================] - 0s 2ms/step
tf.Tensor(0.77179676, shape=(), dtype=float32)
```

I believe the imbalance of the dataset may have influenced the neural network and caused it to lean more towards predicting fake than real, giving it high recall but low precision, which pushed the f1 scored down.

The SVM seems to have done alright, it didn't have quite as much tweaking potential but it remained solid and with a somewhat reliable 0.77 F1 score.

4 Conclusion

Based on the observations from the behaviour of the models, particularly the neural network. It is clear that the dataset is imbalanced. In the future, I would prioritise trimming the data down to get an even split.

Whilst restricting to simply the text and metadata was interesting, I would be very curious to see where this could have gone if the specification allowed for task-relevant data to be collected, particularly things from the Twitter API and more contextual to relational information.

I believe a key thing that might have improved the end result would have been further natural language processing for the text and possibly some more advanced feature selection. Having extra relational processing for the hashtags and mentions could also have improved things a lot given the type of hashtag did not make it into the final processed data set, only how many hashtags the tweet contained. Cross-referencing data with external things like common names, as well as dates of natural disasters, could have potentially helped a massive amount too.

Overall I believe mid-high 0.7s is still a respectable F1 score, and I feel satisfied with it. Working to improve the model has let me acquire a lot of firsthand experience about how the dataset can influence the model more than the algorithm itself.

References

[1] https://ceur-ws.org/Vol-1436/Paper4.pdf.

[2] Patricia Bassey. https://medium.com/axum-labs/logistic-regression-vs-support-vector-machines-svm-c335610a3d16.