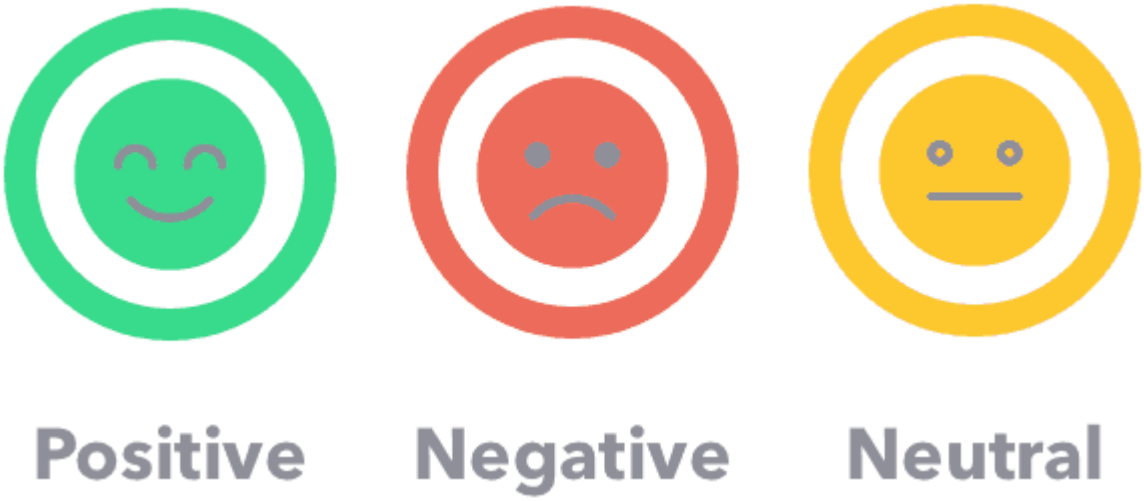
GROUP 17

EURECOM
Sophia Antipolis

AML Challenge 3 :

Text-based Sentiment Analysis

Vincent Labarre, Ankita Sahoo, Yassine Elkheir, Lamia Salhi



From: https://thevaultznews.com/economics/securities-markets/investor-sentiment-to-boost-following-inflation-rate-declines/

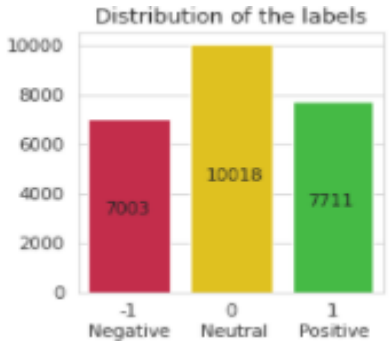
I. Introduction :

During this challenge, we tried to determine whether a tweet is a positive, negative or neutral message. For this purpose, we implemented different models. We tried a few basic models as well as a few advanced ones. After implementation, we compared their results and selected the best performing one.

II. Data Pre-processing :

a. Data exploration

The training dataset is composed of 24732 tweets (with 0 Nan or missing values), which is relatively important. Moreover, from the shown label distribution on the right, we can say that this distribution is skewed between neutral and positive/negative. Positive and Negative classes are well balanced. But in fact, we will see after, this does not entail a predominance prediction of neutral tweets over the positive/negative tweets. Thus we kept this configuration for the rest (we did not do data augmentation here).



b. Basic techniques for preprocessing

A preprocessing was already done on the “text” column to obtain the “selected_text” column. Although this pre-processing had majorly cleaned up the tweets, after inspection, it seemed to us that it could be greatly improved if we do it on “selected_text”. Therefore, we did the basic preprocessing steps for NLP on the “selected_text” column. Here is a table resuming all the step we made:

text	selected_text	Tweet_punct	Tweet_tokenized	Tweet_nonstop	Tweet_stemmed	Tweet_lemmatized
Happy Mother`s Day hahaha	Happy Mother`s Day	Happy Mothers Day	[happy, mothers, day]	[happy, mothers, day]	[happi, mother, day]	happy mother day
Sorry for the triple twitter post, was having ...	Sorry for the triple twitter post, was having ...	Sorry for the triple twitter post was having t...	[sorry, for, the, triple, twitter, post, was, ...	[sorry, triple, twitter, post, trouble, wstock...	[sorri, tripl, twitter, post, troubl, wstocktw...	sorry triple twitter post trouble wstocktwits ...
thats much better than the flu syndrome!	thats much better	thats much better	[, much, better]	[much, better]	[much, better]	much better
Aww I have a tummy ache	tummy ache	tummy ache	[tummy, ache]	[tummy, ache]	[tummi, ach]	tummy ache

Table showing each step of the preprocessing

First, we suppressed the punctuation, **tokenized** the text and then, we put all the letters in lowercase. The tokenization is a way of separating a piece of text (a tweet here) into smaller units (words here) called tokens. Technically, it transforms a string of text into a list of strings, each string representing a word from the text.

Then, we did some **word replacements**. Indeed, they are many piece of texts that are not useful for the prediction, like the @user_name or url (http…). We removed them as well as the stop words (the commonly used words in english) but not all of them. For instance, after a few experiments, it appeared that it was better to keep words like “not, no” to detect more negative sentences. Some other tweets contained typos or “stylistic” manner to write it (e.g. “ahmazing”). We corrected them.
To notice what words must be changed, we looked at N-gram. We did so for N=1,2 and 3. For example, we saw that there were many occurrences of the unigram “inlove” and of the 2-gram “in love”, so we splitted the first unigram to obtain the second unigram, which permits better performance (this is detailed in the part IV.a.i) of the report).

Finally, we had to choose if we **stemmed or lemmatized** the words. **Stemming** is the process of reducing inflection in words (e.g. “troubles”, “troubled”) to their canonical form (e.g. “troubl”) and **lemmatizing** actually transforms words to the actual root (e.g. “trouble”). Stemming is typically easier to implement and run faster than lemmatization, but it operates on a single word without knowledge of the context. Therefore it cannot discriminate between words which have different meanings depending on part of text, unlike the lemmatization. Thus, lemmatization is more efficient but takes much longer to run. But here we can apply it and it actually gives slightly better results than the stemming. For example, for the transformer-BERT model we will discuss later, we obtained these results on the validation set:

Lemmatized

	precision	recall	f1-score	support	pred
-1 (negative)	0.892523	0.835277	0.862952	686	642
0 (neutral)	0.856016	0.875000	0.865404	992	1014
1 (positive)	0.892421	0.917085	0.904585	796	818
avg / total	0.877852	0.877526	0.877330	2474	2474

(val_loss = 0.45562342231949937, val_accuracy = 87.86057692307692)

Stemmed

	precision	recall	f1-score	support	pred
-1 (negative)	0.864907	0.811953	0.837594	686	644
0 (neutral)	0.841223	0.859879	0.850449	992	1014
1 (positive)	0.886029	0.908291	0.897022	796	816
avg / total	0.862206	0.862167	0.861869	2474	2474

(val_loss = 0.45601214685787755, val_accuracy = 86.33814102564102)

Transformer-BERT results depending on the lemmatization or stemming (all other variables fixed

c. Features Extraction

To analyse preprocessed data, we need to convert them into features. Depending upon the usage, text features can be constructed using many techniques. In this challenge, we have tried different techniques to extract the desirable features: Bag of Words, Word2Vec and BERT.

i. Bag-Of-Words :

A bag-of-words model is a way of extracting features from text to be used in modeling, such as with machine learning algorithms. The approach is very simple and flexible, and can be used in a myriad of ways of extracting features from documents. A bag-of-word is a presentation of text that describes the occurrence of words within a document. It involves two things: 1. A vocabulary of known words. 2. A measure of the presence of known words. It is the simplest way to vectorize a record, but the main challenge in this technique is that all the words (among the ones you chose) are weighted uniformly which is not true in all scenarios as the importance of the word differs with respect to context.

ii. Word2Vec :

Word2Vec is a Word embedding technique, which is the modern way of representing words as vectors. The objective of word embeddings is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. The word2vec model is more advanced than bag-of-words. It is not a single algorithm but a combination of two techniques – **CBOW (Continuous bag of words)** and **Skip-gram** model. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations. CBOW tends to predict the probability of a word given a context. A context may be a single adjacent word or a group of surrounding words. The Skip-gram model works in the reverse manner, it tries to predict the context for a given word.

iii. Bert

BERT stands for Bidirectional Encoder Representations from Transformers. It is a pre-trained model which is trained bidirectionally from left to right and vice versa. It is basically used to extract high quality language features from the text data. Bert Tokenzier which is used is trained with two training tasks:

a) Classification Task: to determine which category the input sentence should fall into
b) Next Sentence Prediction Task: to determine if the second sentence naturally follows the first sentence.

We have chosen the Bert Tokenizer as a pre-trained model as it is trained on a large corpus of unlabelled text including the entire Wikipedia (that is 2,500 million words!) and Book Corpus (800 million words) which allows us to get appropriate pre-trained weights. Using these pre-trained weights is easier to fine tune the model.

III. Models :

In this part we introduce the different models by explaining the reason we tried them, the metrics we chose and the way we tune them.

a. LSTM/K-NN/Random Forest/MLP

First, we used the preprocessing introduced in part one and we compared the word2vec and the Bag-of-words embedding techniques on four models: LSTM, K-NN, Random Forest and MLP.

The **LSTM** is a recurrent neural network used in deep learning. The power of LSTM is that it has feedback connections which allows it to treat sequences of data such as speech or video. This is why we used it as we have sequences of words. To implement it we used keras library. Our neural network includes: an embedding layer, an LSTM layer with dropout, a dense layer and a final layer to output 3 dimensional vectors using softmax. As a consequence, we used the categorical_entropy loss function. Finally, we used cross-validation with the **auc** metric to tune all the hyperparameters (dropout, number of neurons, units of LSTM, max words for the embedding, optimizers).

The **kNN** is considered a non-parametric method given that it makes few assumptions about the form of the data distribution. This approach is memory-based as it requires no model to be fit. Nearest-neighbor methods use the observations from the training set closest in input. We have chosen k=4 to escape from overfitting and to generalise the model well.

The **Random Forest** is an ensemble method. We adjust parameters such as max depth, min samples split, n_estimators, and random state to achieve the best performance and to escape from overfitting, we have set max_depth = 40 and n_estimators = 300 (n_estimators is the number of decision trees in the random forest and max_depth is the maximum depth of a decision tree)

The **MLP** is a feed forward artificial neural network that trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. We have set as optimizer Adam, learning rate equals to 1e-05 and three hiddens layers of 300 neurons.

We compared Bag of words and word2vec techniques on those models and here is a table resuming the results :

	BAG OF WORDS				WORD2VEC			
Metrics	**Accuracy**		**AUC (Area under the ROC Curve)**		**Accuracy**		**AUC (Area under the ROC Curve)**	
Models/Split	**Training Set**	**Validation Set**	**Training Set**	**Validation Set**	**Training Set**	**Validation Set**	**Training Set**	**Validation Set**
k-Nearest Neighbors	**0.7497**	**0.6673**	**0.9533**	**0.8536**	**0.8175**	**0.7263**	**0.9499**	**0.8677**
Multi Layer Perceptron	**0.8803**	**0.8168**	**0.9676**	**0.9361**	**0.7802**	**0.7744**	**0.9147**	**0.9094**
Random Forest	**0.7425**	**0.7029**	**0.9524**	**0.8957**	**0.9699**	**0.8043**	**0.9953**	**0.9260**
LSTM	**0.5688**	**0.5821**	**0.79**	**0.76**	**0.4006**	**0.4010**	**0.5**	**0.5**

Comparing Bag-of-words and word2vec using K-NN, MLP, Random forest, LSTM based on auc and accuracy metrics

Using the **Bag of words** model, we obtained good accuracies and AUC scores from a simple feature extraction model. The increase in training accuracies for the models using **Word2Vec** used previously was foreseen, since it does not equalize all the inputs (as in the bag of words method), but it is based on a continuous bag of words and ng-grams. But the abrupt increase in the training accuracy has induced a big difference with the validation accuracy for most of the models, which is interpreted as overfitting. Thus, in our case Bag of words is the chosen technique.

b. Transformer(+Bert)

A transformer is a deep learning model that adopts the mechanism of attention, weighing the influence of different parts of the input data. It provides general-purpose architectures like BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet for Sentiment analysis with over 32+ pretrained models in 100+ languages. For our scenario we have used the Bert Classifier model with a pretrained Bert Tokenizer. We choose Bert models to pre-train the model as well as the classifier because it is a simple and empirically powerful model. Meanwhile, it is a “deep bidirectional” which means it learns information from both the left and the right side of a token’s context during the training phase which is more efficient. Using the Bert Classifier from Transformers we got the below accuracy for the given scenario.

Models/Split	Accuracy Training Set	Accuracy Validation Set	AUC Training Set	AUC Validation Set
Transformer + Bert	**0.9116**	**0.8794**	**0.98**	**0.96**

Transformer results (accuracy and auc)

c. Model selected

Considering the different models, the KNN classifier and the Random Forest models are trained well but while it predicts, it overfits. This may be because of the hyperparameters which could have been better tuned. On the other hand, the basic MLP performs better because of its adaptive learning which allows the model to learn how to do tasks based on the data given for training.Then we tried more advanced models like LSTM and Bert Classifier from Transformers. The LSTM did not perform well. We were expecting it to perform better but we think that because of our preprocessing there is not much necessity to remember things and then classify. If the scenario would have been remembering the sentences without the preprocessing we used, then maybe the LSTM would have performed better. The Bert Classifier from the transformer gave us the best results as it learns bidirectionally which was indeed efficient.

Finally, considering the accuracy and the auc of the different models, we selected the Bert Classifier from Transformers.

.

IV. Model Selected and Performance :

a. Improvement methods

i. Orthographic improvement

As mentioned in the preprocessing part, one major improvement we can do, particularly for the transformer, is to correct or replace some words. For example, by correcting the abbreviation “tmr”, “tmw”, “tmrw” by “tomorrow”, we can avoid the attribution of weight for the former words to attribute them to the later word. This allows the improvement of the attention mechanism, which for each word, weighs the relevance of the other words and draws information from them and produces the output accordingly (here the type of sentiment of the tweet). Of course, there is a risk of misinterpretation of some “tomorrow” abbreviation (for instance the “tmw” abbreviation can stand for “the mortgage works” (tmw)). But here we have to think in terms of occurrence: there are much more occurrences of the abbreviation “tmw”, “tmr”,”tmrw” for the word tomorrow than for other words or acronyms. However, the drawback is that it is really time consuming to do it: we have to scrutinize the datasets to observe what we can replace (after applying some methods to suppress or replace the most common abbreviation / stopwords already recapped in some python dictionaries). In fact, doing too many corrections leads to an oversimplification of the text that prevents the model from generalizing well.

ii. Dropout

Like all neural network models, the transformer is prone to overfitting. One technique to avoid this is to use the dropout method. Indeed, during the training some number of layer outputs are randomly ignored or “*dropped out”.* This also permits learning more robust representations of data. We did some experiments and recapped them on the table at right. We thus conclude that the dropout technique allows us to improve the validation accuracy and reduce the overfitting.

Transformer/BERT, epoch=3, Adamw (lr=5e-5, eps=1e-8)				
Dropout	Train Acc	Val Acc	Train AUC	Val AUC
None	0.9112	0.8754	0.9786	0.9564
0.3	0.9095	0.8802	0.9527	0.9554
0.5	0.9527	0.8774	0.9812	0.9568
0.7	0.9451	0.8786	0.9789	0.9599

b. Further evaluation comments

To understand the extent of our model performance we used other evaluation metrics. The precision metric is used to know how well a positive value is determined, i.e. it displays the proportion of true positive value out of the predicted positive values. The recall metric is used to know how much positive values are determined, i.e. it displays the proportion of true positive value out of the actual positive values. The f1-score metric permits synthesizing both previous metrics.

For the training:

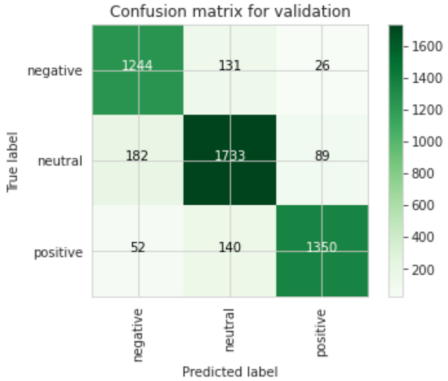
	precision	recall	f1-score	support	pred
-1 (negative)	0.922193	0.930918	0.926535	5602	5655
0 (neutral)	0.924917	0.936112	0.930481	8014	8111
1 (positive)	0.960625	0.937267	0.948802	6169	6019
avg / total	0.935279	0.935001	0.935076	19785	19785

For the validation:

	precision	recall	f1-score	support	pred
-1 (negative)	0.841678	0.887937	0.864189	1401	1478
0 (neutral)	0.864770	0.864770	0.864770	2004	2004
1 (positive)	0.921502	0.875486	0.897905	1542	1465
avg / total	0.875914	0.874672	0.874934	4947	4947

We see here that the f1 score is the best for positive labels, which means that we detect the positive sentences more easily in both training and validation, because they have more obvious key words to characterize them. Moreover, we notice that the f1 score is greater for the training than for the validation, which indicates that there is a little overfitting, even with the use of dropout.

We also looked at the confusion matrices. On the right we can see the confusion matrix on the validation set. Most of the values located on the diagonal show us that the true positive values are largely predominant and that this model is a good classifier.



V. Conclusion :

To conclude, through this challenge we discovered different kinds of embedding techniques such as word2vec, bag-of-words and bert. We learned what kind of basic preprocessing one can do on typescripts data. And by comparing different models based on the accuracy and the auc we chose the transformer with the bert model. For our submissions we submit the outputs of the transformer with bert. Natural language is a large domain and we saw how difficult it was to tune models and preprocess the data in order to reach good auc and accuracy. For future work we could enhance our model to detect the words based on which, it classifies the sentiment as positive, negative or neutral.