Student: Elissaios Baxevanis
Student ID: 2588922B
Student email: 2588922b@student.gla.ac.uk
Date: 04-06-2021

Coursework Text-as-Data Report

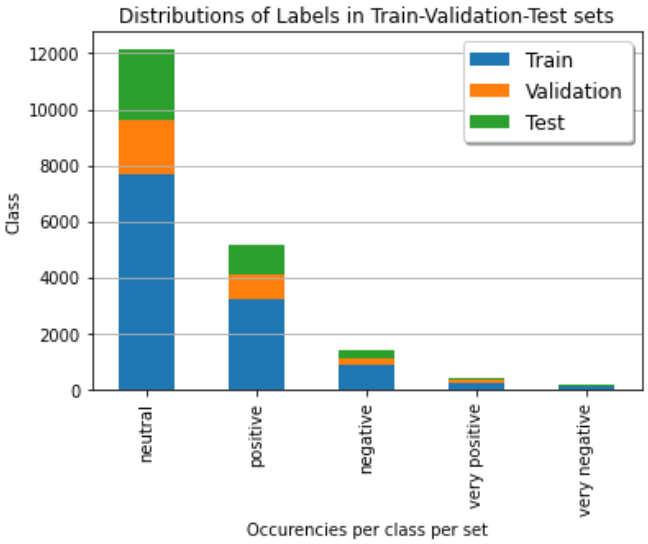
Introduction

Subject of this project is to perform multi-class classification on Reddit datasets. More precisely, the goal is the sentiment classification of Subreddit Posts based on the polarity of each sentiment, by creating a sentiment analysis model that combines Machine Learning and Natural Language Processing strategies. Sentiment polarity refers to the "direction" of each sentiment to express something positive-negative-neutral, in this case sentiments have been categorized into five bins, "very negative", "negative", "neutral", "positive", "very positive". The project is structured in three parts:

I. Implement and test different classifies, measure their performance on test set and perform analysis for them,
II. Hyperparameter optimisation of a specific classifier used in the first part and perform error analysis,
III. Investigation of different features in order to find, two, that enhance the performance of the optimised classifier obtained in the second part.

Question 1

Dataset consists of twelve (12) features (columns), and it is splitted into three subsets, train-validation-test, each one containing a different number of posts (rows). Initially a sanity check for missing labels was done, but none was missing. As it can easily be observed in the following plot, there is a clear imbalance among the five classes, moreover the imbalance seems to be "symmetric" to all subsets. The table below makes clear that neutral class is the dominant class across all subsets, while very-negative has an extreme degree of imbalance, very-positive has a moderate degree of imbalance, and positive & negative classes have a mild degree of imbalance.



	Train Class %	Validation Class %	Test Class %
neutral	63.264	63.075	62.600
positive	26.619	27.179	27.440
negative	7.233	6.915	7.022
very positive	2.084	2.348	2.141
very negative	0.799	0.482	0.797

In this project we are mainly interested in two features of the data set, "body" which are the training data and "sentiment.polarity" which are the labels.

```
Text:  Size reference?
Polarity:  neutral
```

As labels were in the ordinal format described earlier, we used ordinal encoder to transform them into numerical in order to be able to perform the analysis.

Also, tokenization & normalization were applied the same way we did during labs, more specifically normalization includes: transformation to lowercase representation, lemmatization (the embedded model in NLP model), removal of whitespace characters and digits, as shown in the image.

```
It's a sad realization, isn't it?
['it', 'a', 'sad', 'realization', 'is', 'it']
```

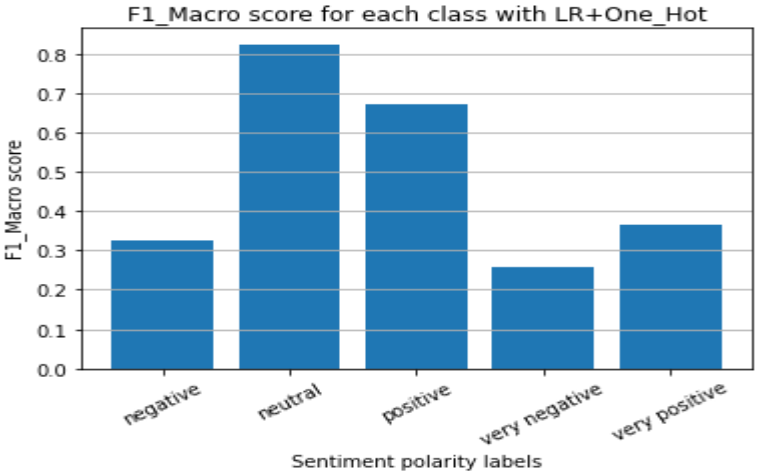
As the train data is also in text format, we had to transform them into numerical representation, techniques that we used were One-Hot vectorisation and Tf-Idf vectorisation.

The classifier of our choice was Multinomial Naive Bayes (MNB), which is a suitable classifier for this task, as it is a probabilistic classifier and in case of absence of clear evidence of a clear classification it chooses the class with the higher probability. As other experiments included Dummy Classifier, Logistic Regression and SVM classifiers, our intention was to explore a different classifier instead of one among the already used ones. Moreover, it allows Laplace smoothing tuning in hyperparameters ("alpha") and allows me to explore this technique in the given datasets. Lastly, we used the same tokenization & normalization techniques with the other models, in order to evaluate the performance of our classifier without any other "variables changed".

Also, the metric that we were taking mostly into account was F1 Score, as there is heavy class imbalance, thus accuracy is not a realistic measure.

Dummy Classifier with the “most_frequent” strategy always predicts the dominant class, as described earlier there is severe class imbalance, thus its accuracy is high but the actual measure unit is the F1-Macro, which is poor. Dummy Classifier with “stratified” strategy is also affected heavily by the class imbalance, as its predictions follow the data distribution in the dataset, and results similar to “most_frequent” strategy. The combination of Logistic Regression (LR) with One-Hot vectorisation outperforms the other combinations, both in train and test sets. On the other hand, LR with TF-IDF vectorisation is not achieving such high scores as the relevance of each token is taken into account, and class imbalance contributes to that, as the distribution of features differs highly. Next combination that was used was SVM with One-Hot encoding, which is a geometric classifier, although its performance on the train set seems to be descent, it drops significantly on the test set. Moreover, it's the slowest among all combinations, as it has to evaluate distances among all data points, thus it's time consuming and in our point of view not appropriate for this task. Last, our choice of MNB with One-Hot vectorisation, even though in text classification literature MNB is a common, starting, option though it’s naive assumption of independence is proved (logically) to be problematic, as words have meaning based on the context. All the classifiers seem to overfit as their performance decreases significantly from train to test set, although all of them have achieved the goal of learning as they outperform the basis, Dummy Classifier, their performance in train & test sets are shown in the tables below, with the max score of F1-Macro highlighted.

	Precision_macro	Recall_macro	F1_macro	Precision_weighted	Recall_weighted	F1_weighted	Accuracy
DummyClassifier_MostFrequent_Train	0.200000	0.127000	0.155000	1.000000	0.633000	0.775000	0.633000
DummyClassifier_Stratified_Train	0.199000	0.199000	0.199000	0.473000	0.475000	0.474000	0.475000
LogisticRegression_OneHot_Train	0.845000	0.982000	0.901000	0.967000	0.963000	0.964000	0.963000
LogisticRegression_TF-IDF_Train	0.418000	0.909000	0.473000	0.901000	0.817000	0.844000	0.817000
SVC_OneHot_Train	0.423000	0.750000	0.454000	0.931000	0.862000	0.887000	0.862000
MultinomialNB_OneHot_Train	0.384000	0.730000	0.407000	0.893000	0.818000	0.847000	0.818000
	Precision_macro	Recall_macro	F1_macro	Precision_weighted	Recall_weighted	F1_weighted	Accuracy
DummyClassifier_MostFrequent_Test	0.200000	0.125000	0.154000	1.000000	0.626000	0.770000	0.626000
DummyClassifier_Stratified_Test	0.191000	0.191000	0.191000	0.465000	0.466000	0.466000	0.466000
LogisticRegression_OneHot_Test	0.441000	0.635000	0.489000	0.786000	0.748000	0.762000	0.748000
LogisticRegression_TF-IDF_Test	0.330000	0.574000	0.356000	0.852000	0.740000	0.779000	0.740000
SVC_OneHot_Test	0.288000	0.458000	0.287000	0.875000	0.729000	0.782000	0.729000
MultinomialNB_OneHot_Test	0.264000	0.337000	0.259000	0.830000	0.684000	0.738000	0.684000



The best performing classifier is Logistic Regression with One-Hot vectorisation, the following bar chart shows its F1-Macro scores for each of the classes.

Question 2

In this part, the goal was to perform hyperparameter optimization for the combination Logistic Regression with TF-IDF vectorisation, explored earlier. More precisely, goal is to find the optimal values for the following parameters:

- LR Classifier
  - C (regularisation)
  - max_iter (number of iterations to convergence): Parameter of our choice
- TF-IDF Vectorizer
  - sublinear_tf (replace tf with 1+log(tf))
  - max_features (the number of features that participate in vocabulary)

In order to implement the optimization, GridSearchCV (GS) was used as, we did in labs. Moreover, we joined the train and validation sets, to create a bigger sample and performed GS on it, with a custom CV (fit on train-validate on validation). The process was sequential, as we examined each parameter, received an optimisation proposal, then moved to the next parameter, where we applied the previous obtained optimisation proposals and so on, until we tuned all four parameters. The F1-Macro scores alongside with the proposed optimized value at each step can be seen in the table on the right.

	Tuned Values	F1_score
LogReg-C	(10000.0)	0.509566
LogReg-max_iter & (C tuned)	(100)	0.509566
Tfidf-sublinear_tf & (max_iter & C tuned)	(True)	0.527930
Tfidf-max_features & (max_iter & C & sublinear_tf tuned)	(1000)	0.539068

	Precision(macro)	Recall(macro)	Accuracy	F1_score(macro)
Baseline	0.330	0.574	0.740	0.356
Optimized	0.485	0.550	0.759	0.512

After the optimisation process, we evaluated the model on the test set and compared with the baseline model used in Q1, and a significant improvement in F1-Macro score was observed, as shown in the table on the left.

The parameter that improved the score significantly during tuning was regularisation term C in LR, sublinear_tf & max_features in Vectorizer did also improve the performance but it was not that huge an improvement. Finally, the parameter of our choice max_iter in LR did not offer any improvement, as the optimisation process concluded that the optimal value was the default one (already used). Moreover, we attempted to optimise different parameters such as “solver” in LR, which determines the algorithm of optimisation, tried ‘newton-cg’, ‘saga’ and default (‘lbfgs’) which are the suggested ones for multi-class classification tasks but also the default option (‘lbfgs’) was the optimisation proposal. The last parameter that we tried to optimise was “ngram_range” in TFIDF-Vectorizer, tried unigram - bigram - trigram - fourgram, however the default option ‘unigram’ was the proposed value. Thus, we concluded that given the nature of the dataset, we should stick to the ‘max_iter’ option as the dataset can become much larger by adding posts thus the convergence time would be more important, while the features will be the same thus the same optimisation algorithm & ngram_range will still apply. One should take into account the severe class imbalance in the dataset, thus the improvements cannot be “amazingly significant”, as no over/under sampling techniques applied, which would have led to much better results.

	negative	neutral	positive	very negative	very positive
negative	102	166	7	4	3
neutral	103	2181	205	8	17
positive	7	345	729	1	20
very negative	9	17	0	6	0
very positive	0	18	38	0	30

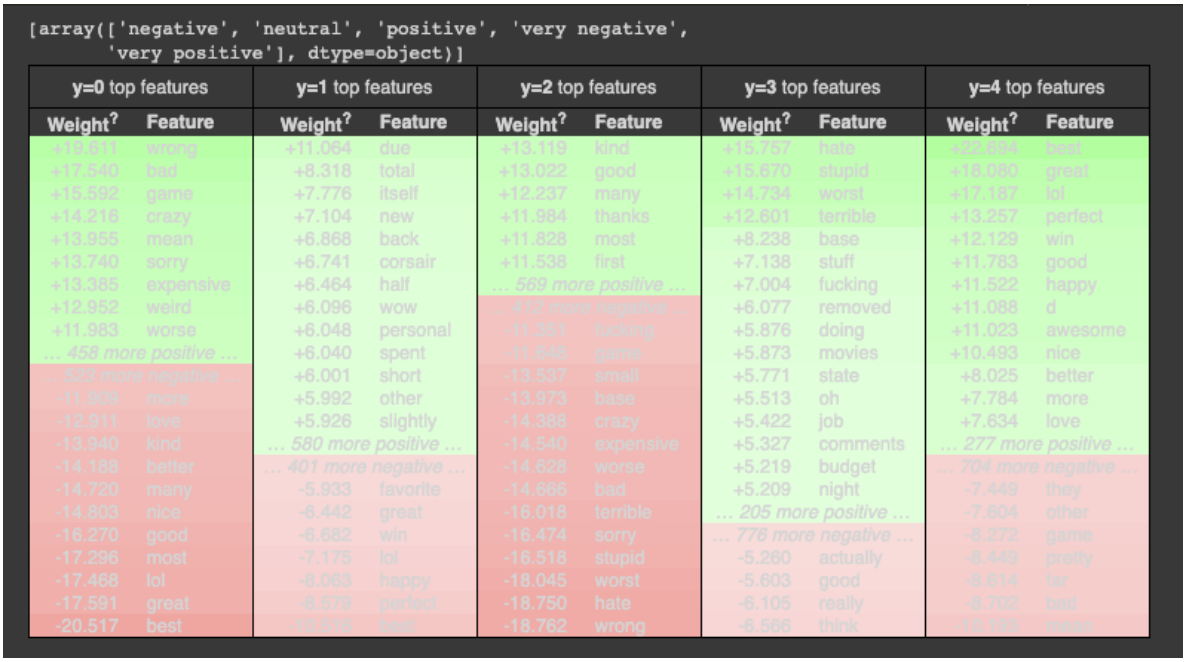
Confusion Matrix of the optimised model evaluated at the test set is shown on the left-hand side, and it's clear that the model faces a significant issue with the neutral class, which in our point of view is logical due to the class imbalance.

More precisely, it can easily be observed that the most obvious confusions that happen are happening to all classes with neutral class due to the class imbalance, also one misclassification example is provided to the major confusions (there are more examples of each confusion in the notebook).

Raw Text (test_data[‘body’] column)	**True Label**	**Prediction**
They only get an initial 5-10 point boost at the start, irrelevant as the game goes on.	Negative	Neutral
Here's what your build should look like. It includes one of the top performing CPUs out there, a very fast SSD, and two of what's currently the second best single GPU card out there. This is just an example.	Neutral	Positive
A face mage with ice block?	Neutral	Negative
Nothing beats it on a hot day working on the farm. Sweet or	Positive	Neutral

`unsweet even.`		
`Chef was great. Heartfelt without being too sacharine. Good choice.`	Positive	Very Positive
`Chaox :(`	Very Negative	Neutral
`that's the worst answer I've ever heard.`	Very Negative	Negative
`Yes, if 6mg doesn't suffice, up your nic amount.`	Very Positive	Positive
`Are legendary also put into bundles?`	Very Positive	Neutral

Another issue that we noticed after doing this error analysis is that in the given datasets, there are sentiments that hold the wrong label; thus, this also affects the performance of the model.



```
[array(['negative', 'neutral', 'positive', 'very negative',
       'very positive'], dtype=object)]
```

y=0 top features		y=1 top features		y=2 top features		y=3 top features		y=4 top features	
Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature
+19.611	wrong	+11.064	due	+13.119	kind	+15.757	hate	+22.694	best
+17.540	bad	+8.318	total	+13.022	good	+15.670	stupid	+18.080	great
+15.592	game	+7.776	itself	+12.237	many	+14.734	worst	+17.187	lol
+14.216	crazy	+7.104	new	+11.984	thanks	+12.601	terrible	+13.257	perfect
+13.955	mean	+6.868	back	+11.828	most	+8.238	base	+12.129	win
+13.740	sorry	+6.741	corsair	+11.538	first	+7.138	stuff	+11.783	good
+13.385	expensive	+6.464	half	… 569 more positive …		+7.004	fucking	+11.522	happy
+12.952	weird	+6.096	wow	472 more negative …		+6.077	removed	+11.088	d
+11.983	worse	+6.048	personal	-11.331	fucking	+5.876	doing	+11.023	awesome
… 458 more positive …		+6.040	spent	-11.648	game	+5.873	movies	+10.493	nice
… 523 more negative …		+6.001	short	-13.537	small	+5.771	state	+8.025	better
-11.909	more	+5.992	other	-13.973	base	+5.513	oh	+7.784	more
-12.811	love	+5.926	slightly	-14.388	crazy	+5.422	job	+7.634	love
-13.940	kind	… 580 more positive …		-14.540	expensive	+5.327	comments	… 277 more positive …	
-14.188	better	… 401 more negative …		-14.628	worse	+5.219	budget	… 704 more negative …	
-14.720	many	-5.933	favorite	-14.666	bad	+5.209	night	-7.449	they
-14.803	nice	-6.442	great	-16.018	terrible	… 205 more positive …		-7.604	other
-16.270	good	-6.682	win	-16.474	sorry	… 776 more negative …		-8.272	game
-17.296	most	-7.175	lol	-16.518	stupid	-5.260	actually	-8.449	pretty
-17.468	lol	-8.063	happy	-18.045	worst	-5.603	good	-8.614	far
-17.591	great	-8.579	perfect	-18.750	hate	-6.105	really	-8.702	bad
-20.517	best	-10.516	best	-18.762	wrong	-6.566	think	-10.193	mean

Then, we made use of the Eli5 in order to get a better insight of the optimised model weights associated with each class, more precisely we used the explain weights function to get the top positive and negative features per class, as shown on the left.

Moreover, as Eli5 offers inspection of a single misclassification, we used the function show_predictions, in order to get a better intuition of the mistakes and why they are happening, an example can be seen on the right (showing only prediction-misclassification columns, the whole table is available in the notebook).

y=1 (probability 0.380, score 3.691) top features			y=2 (probability 0.615, score 4.171) top features		
Contribution?	Feature	Value	Contribution?	Feature	Value
+3.944	<BIAS>	1.000	+2.045	first	0.177
+1.010	last	0.207	+1.879	great	0.191
+0.858	first	0.177	+1.183	much	0.157
+0.446	game	0.172	+1.149	<BIAS>	1.000
+0.403	much	0.157	+0.540	shows	0.265
+0.283	shows	0.265	+0.392	mana	0.243
+0.228	lost	0.256	+0.262	anyone	0.210
+0.103	like	0.126	+0.188	a	0.132
+0.092	else	0.209	+0.163	else	0.209
+0.088	from	0.141	+0.159	has	0.150
+0.084	the	0.149	+0.129	the	0.149
+0.063	in	0.098	+0.123	years	0.191
+0.058	a	0.132	+0.090	last	0.207
+0.044	has	0.150	+0.061	in	0.098
+0.031	vs	0.256	+0.053	like	0.126
-0.028	it	0.083	+0.031	from	0.141
-0.030	close	0.228	+0.010	this	0.116
-0.073	this	0.116	-0.010	it	0.083
-0.085	years	0.191	-0.062	is	0.092
-0.092	how	0.152	-0.094	vs	0.256
-0.136	anyone	0.210	-0.115	was	0.212
-0.150	is	0.092	-0.116	close	0.228
-0.162	that	0.092	-0.131	stuff	0.217
-0.278	was	0.212	-0.136	watching	0.252
-0.300	ground	0.274	-0.185	that	0.092
-0.331	mana	0.243	-0.193	how	0.152
-0.478	stuff	0.217	-0.395	ground	0.274
-0.670	watching	0.252	-0.842	lost	0.256
-1.231	great	0.191	-2.007	game	0.172

```
Body:
Was watching a VOD from last years DreamHack Winter.The first map in the finals (Naama vs Mana) was Lost Temple.. spawning close ground positions..It's a great sign that shows how much the game is evolving imo. Has anyone else noticed stuff like this?

True Label: positive (y=1)
Prediction:neutral (y=2.
```

It is worth mentioning that the top features in the neutral class are three words that we would

expect, to be in favour of the positive class, which are “first” and “great” and “much”, as they imply positive attitude. Also, as we can compare with the above table of top features per class, all the others that are conformed with each class are low weighted, thus the model has uncertainty at its prediction and the risk of misclassification is high, as happened.

Question 3

In this part we were asked to add two new features to enhance the model acquired in the previous part. Models were trained on the training set, evaluated on the validation set and compared to the baseline model of Q2, finally the best performing model with the new features was tested on the test set. As “n_gram range” was investigated in Q2 during the optimisation process, we decided not to investigate any further due to pressure of time.

The first step was to investigate different options of vectorizers, in order to choose the optimal one. We tried CountVectorizer and Tfidf vectorizers used already in the project, with various combinations (with and without stop_words and One-Hot Encoding) alongside with the optimised Classifier obtained in previous part. Results of their performance can be seen in the next table. The best option was CountVectorizer, <u>without</u> OH & stop_words.

DIFFERENT VECTORISATION APPROACHES

	Precision(macro)	Recall(macro)	Accuracy	F1(macro)
CV-OH	0.555000	0.558000	0.732000	0.550000
CV	0.541000	0.592000	0.749000	0.562000
CV-OH-StopWords	0.536000	0.519000	0.709000	0.517000
CV-StopWords	0.540000	0.520000	0.708000	0.522000
TFIDF	0.493000	0.552000	0.736000	0.512000
TFIDF-OH	0.524000	0.568000	0.743000	0.529000
TFIDF-StopWords	0.506000	0.520000	0.719000	0.506000
TFIDF-OH-StopWords	0.495000	0.518000	0.712000	0.497000

In order to be sure that it performs better than the optimised model acquired in the previous part, we evaluated both of them on the validation sets, and results confirmed the findings, as we can see in the right image.

```
'Q2 Baseline on Validation:' has Acc=0.757 P=0.507 R=0.584 F1=0.539
'Vectorizer Optimised on Validation:' has Acc=0.744 P=0.540 R=0.589 F1=0.561
```

Next step was to combine the basic feature, “body”, alongside with one more feature, based on our understanding of the dataset and what would be more beneficial. We combined “body” with the following, already existing, features: “subreddit”, “title” and “majority_type”, “title”, “author”. Moreover we created a new feature associated with the main feature “body”, which is named “body_length” and it counts the length of each post, then by making use of its statistics, we splitted it into 4 subsets, with associated feature labels 0-1-2-3, each one consisting of 25% of the total data. Last additional feature was created, by making use of the VADER sentiment library, which creates new features based on their polarity scores. Although VADER provides features categorised only with negative-neutral-positive labels, their range is [-1,1], thus we splitted this range into 5 equal parts (0.4 each part) in order to adapt it to our task, which requires 5 labels. The compound score was used to assign the labels, the same way we did in the lab.
In order to be able to use these two new features, as they hold numerical values, we created a new ItemSelector named NumericalItemSelector, based on lab 2. In order to combine multiple features, we made use of FeatureUnion.

COMBINATION OF BODY WITH ONE EXTRA FEATURE , AND NEW VECTORIZER

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline: body	0.540	0.589	0.561	0.744
Add: subreddit	0.506	0.552	0.524	0.739
Add: majority_type	0.522	0.559	0.537	0.746
Add: title	0.429	0.494	0.454	0.731
Add:body_length	0.521	0.563	0.539	0.758
Add:vader	0.542	0.613	0.572	0.751
Add: is_first_post	0.537	0.588	0.557	0.747
Add: author	0.516	0.516	0.542	0.752

Table on the left shows the metrics of the various combinations of a single feature alongside with “body” and the new vectorisation. As the addition of the feature “title” produced bad performance, it led me to drop it from further investigation.

The next step was to combine body with the best performing single feature “body length” and one more feature. Surprisingly, the combination of and extra feature with the basic one and “body length”, increased performance only at a single case (“body”, “body length” & “is_first_post”), but only by a negligible % in comparison to the simpler model of “body” & “body_length”, thus the other features are probably not that important to the task. Table on the left summarizes the performance of each combination in the validation set.

COMBINATION OF BODY WITH Body_length & 1 MORE FEATURE & NEW VECTORIZER

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline: body	0.540	0.589	0.561	0.744
Add: body_length & subreddit	0.532	0.567	0.547	0.744
Add: body_length & majority_type	0.526	0.572	0.546	0.746
Add:vader & body_length	0.514	0.546	0.528	0.760
Add: body_length & is_first_post	0.552	0.607	0.576	0.752
Add:body_length & author	0.546	0.577	0.559	0.750

There were two models that outperformed baseline model during the evaluation on validation set, these involved a)”body”, “body_length” & vectorizer and b)”body”, “body_length”, “is_first_post” & new vectorizer, though the difference in performance was negligible if we take into account the increment in model complexity.

All in all, we consider the best model the simpler one (a) (Occam’s razor “rule”) among the ones that performed better than the baseline in validation set. By this said, the best model involves the addition of a new vectorizer (CountVectorizer) and one extra feature, the length of the main feature “body” (“body_length”).

TEST SET COMPARISON

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline-Optimised on Q2	0.485000	0.550000	0.512000	0.759000
Best Model	0.532000	0.571000	0.549000	0.757000

The performance of the model was evaluated on the test set, which is presented at the table on the left.

Furthermore, precision, recall and F1 metrics increase and accuracy drops slightly, which can be considered, negligible (0.1%).
That can be explained from the fact that the model predicts better the minority classes and there is a trade-off with the predictions of the major class (neutral), as shown in the following confusion matrix.

	negative	neutral	positive	very negative	very positive
negative	88	172	14	6	2
neutral	102	2116	270	9	17
positive	6	282	791	1	22
very negative	6	12	1	13	0
very positive	0	11	42	0	33

It is noticeable that the proposed model generalizes better than the optimised one from Q2, by comparing their confusion matrices.