UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Dimitrios Chrysos*
sdi: 2100275

Course: *Artificial Intelligence II (ΥΣ19)*
Semester: *Spring Semester 2024-2025*

Contents

1. Abstract

The task is to perform sentiment analysis using only Logistic Regression and the TF-IDF method in Python on a given English-language Twitter dataset. Three datasets will be used: `train_dataset`, `val_dataset`, and `test_dataset`, which are used for training, validation, and testing, respectively.

2. Data processing and analysis

2.1. Pre-processing

The data cleaning and pre-processing were done using regular expressions and Python methods and applied to the three provided datasets. The following steps were used:

1. To ensure uniformity, all text is converted to lowercase
2. Common spelling mistakes, contractions, slang words, and informal abbreviations are corrected to improve standardization and prevent misinterpretations during feature extraction.
3. Removal of URLs, Hashtags, Emails, Mentions, Numbers, Emojis, Non-ASCII characters, Single-Letter Words, and Special Characters because they provide close to no value for sentiment analysis.
4. Sequences of three or more identical letters are reduced for standardization.
5. Excess whitespace is replaced with a single space for readability purposes.

By applying these techniques, we ensure that the text data is clean, structured, and suitable for sentiment analysis. This pre-processing step improves model accuracy by eliminating noise and standardizing input text.

2.2. Analysis

- Below are some visualizations of the datasets.
    1. Train data:

Figure 1: WordCloud after pre-processing

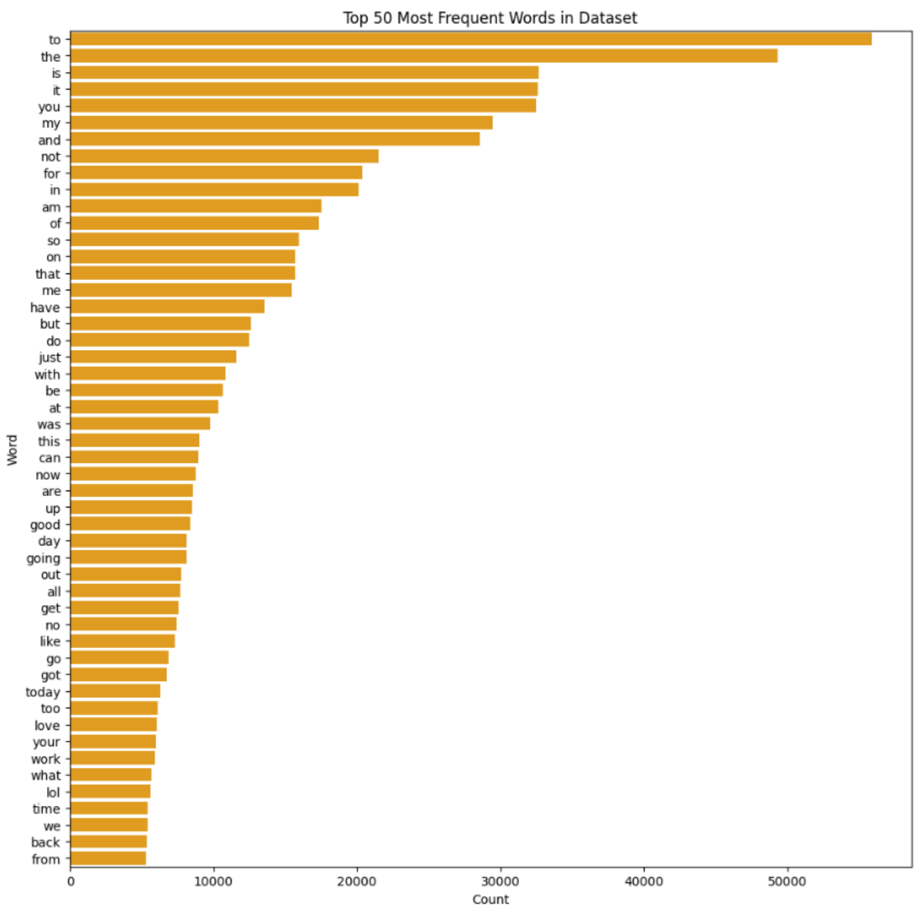

Figure 2: Top 50 Most Frequent Words after pre-processing

Word	Frequency
to	55846
the	49328
is	32668
it	32610
you	32469
my	29437
and	28571
not	21475
for	20377
in	20143
am	17514
of	17349
so	15931
on	15708
that	15695
me	15444
have	13525
but	12623
do	12461
just	11621
with	10869
be	10655
at	10316
was	9775
this	9030
can	8945
now	8754
are	8591
up	8484
good	8391
day	8150
going	8138
out	7746
all	7680
get	7580
no	7428
like	7282
go	6884
got	6741
today	6301
too	6105
love	6054
your	6005
work	5925
what	5688
lol	5631
time	5449
we	5430
back	5363
from	5267

Figure 3: List of the Top 50 Most Frequent Words after pre-processing

2. Validation data:

Figure 4: WordCloud after pre-processing

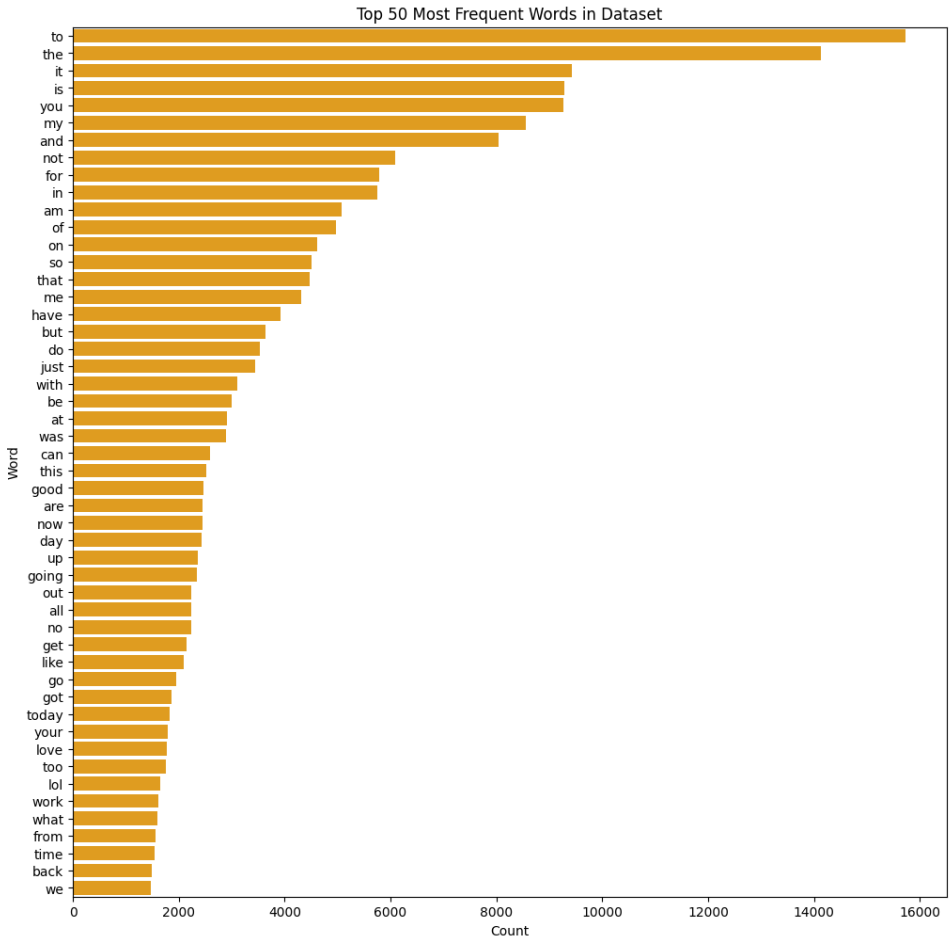

Figure 5: Top 50 Most Frequent Words after pre-processing

	Word	Frequency
32	to	15723
40	the	14124
21	it	9421
138	is	9285
109	you	9262
104	my	8555
18	and	8041
26	not	6093
59	for	5795
54	in	5759
72	am	5078
156	of	4965
77	on	4623
41	so	4512
220	that	4468
225	me	4311
100	have	3916
9	but	3645
25	do	3529
342	just	3442
46	with	3112
83	be	3004
252	at	2910
13	was	2890
123	can	2586
95	this	2514
98	good	2459
309	are	2451
323	now	2447
49	day	2425
74	up	2356
168	going	2342
87	out	2235
193	all	2235
65	no	2228
3	get	2142
70	like	2087
45	go	1953
306	got	1868
263	today	1832
16	your	1788
343	love	1780
459	too	1749
146	lol	1644
12	work	1616
167	what	1603
78	from	1562
383	time	1547
440	back	1483
142	we	1471

Figure 6: List of the Top 50 Most Frequent Words after pre-processing

3. Test data:

Figure 7: WordCloud after pre-processing

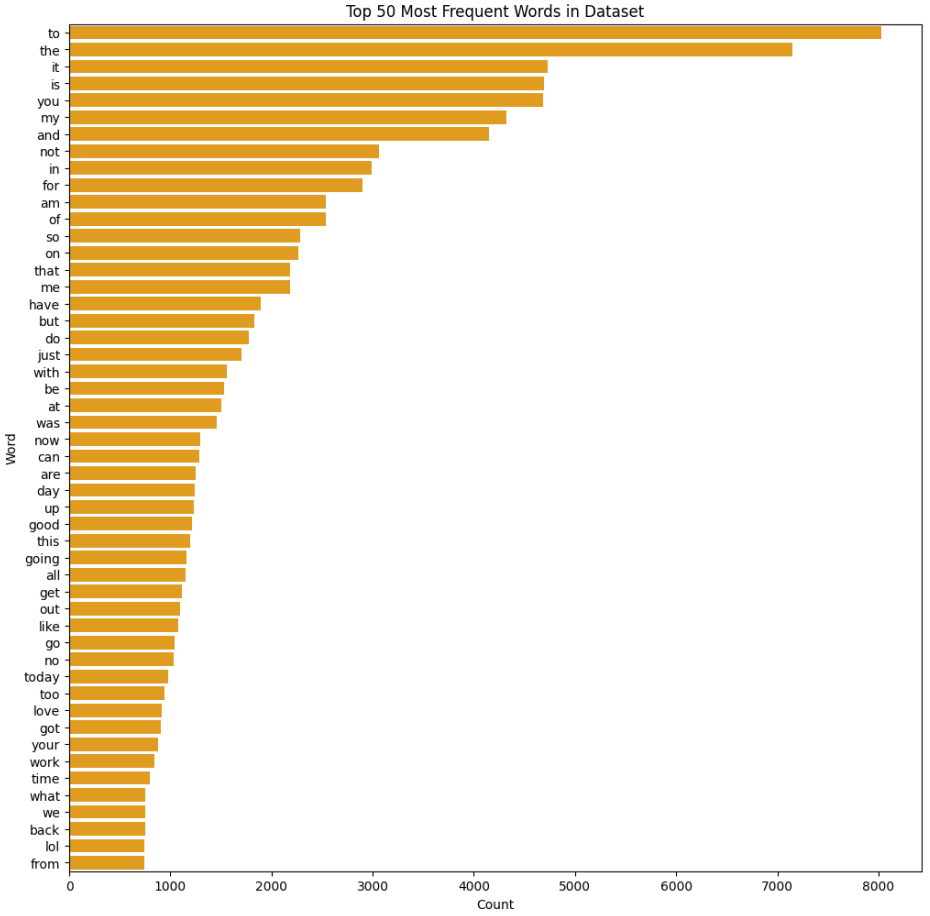

Figure 8: Top 50 Most Frequent Words after pre-processing

	Word	Frequency
8	to	8026
0	the	7147
13	it	4729
14	is	4690
54	you	4684
141	my	4318
70	and	4147
86	not	3064
119	in	2993
62	for	2903
36	am	2538
48	of	2533
64	so	2280
26	on	2267
138	that	2186
171	me	2180
160	have	1893
190	but	1833
383	do	1772
278	just	1705
124	with	1562
195	be	1533
317	at	1502
169	was	1460
289	now	1300
66	can	1283
117	are	1248
132	day	1237
218	up	1232
33	good	1218
17	this	1193
193	going	1161
23	all	1149
293	get	1116
43	out	1095
110	like	1081
335	go	1039
30	no	1036
273	today	977
68	too	943
83	love	913
41	got	904
79	your	875
102	work	842
38	time	797
106	what	754
19	we	750
145	back	748
143	lol	744
481	from	742

Figure 9: List of the Top 50 Most Frequent Words after pre-processing

- For exploratory data analysis, I examined the train, validation, and test datasets to identify common patterns in the text. Word clouds and word frequency graphs revealed that all three datasets share similar top words, indicating consistency in their distributions. This suggests that the datasets are well-aligned, reducing the risk of data shifts that could affect model performance.

2.3. Data partitioning for train, test and validation

- The data was already portioned.

2.4. Vectorization

- The vectorization technique used is Term Frequency-Inverse Document Frequency

(`TF-IDF`), which converts text into numerical features by weighting words based on their frequency in a document and across all documents.

```
vectorizer = TfidfVectorizer(ngram_range=(1,2))
```

- The TfidfVectorizer with `ngram_range=(1,2)` captures both single words and two-word phrases. The method helps highlight important terms while reducing the influence of common words.

3. Algorithms and Experiments

3.1. Experiments

1. First Experiment
   - I began my experiments with a very simple strategy and used the following:
     - `TfidfVectorizer()`
     - `LogisticRegression(max_iter=500, random_state=random_seed)`
   - This strategy produced the following evaluation and plot:

```
Accuracy: 0.79
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79     21197
           1       0.79      0.80      0.79     21199

    accuracy                           0.79     42396
   macro avg       0.79      0.79      0.79     42396
weighted avg       0.79      0.79      0.79     42396
```

Figure 10: Experiment 1 - Evaluation

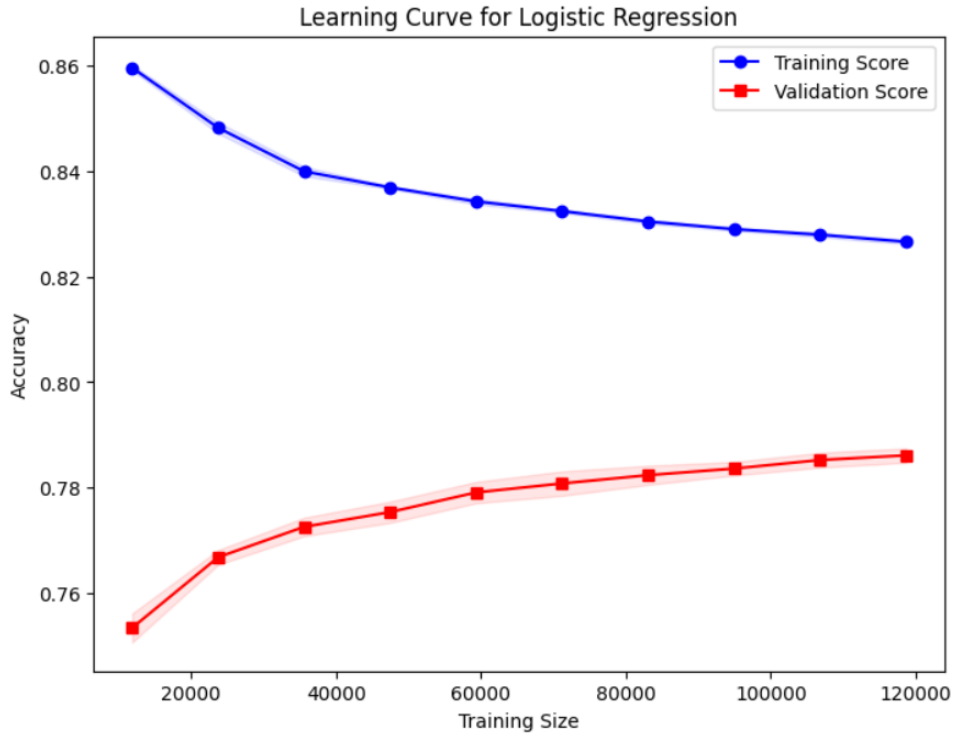

Figure 11: Experiment 1 - Plot

2. Second Experiment

   - For my second experiment I only changed my vectorization approach and kept everything else the same.
     - `vectorizer = TfidfVectorizer(ngram_range=(1,2))`
     - `LogisticRegression(max_iter=500, random_state=random_seed)`
   - This approach allowed the model to consider both single words (unigrams) and two-word sequences (bigrams) when extracting features potentially improving the models ability to capture context, meaning, and relationships between words and potentially leading to better accuracy but with a greater risk of overfitting the training data.
   - **Note:**

     By only having the parameters `max_iter` and `random_state` at `LogisticRegression()` default values are assigned to the other parameters (used at the next experiments) as follows:
     - `C = 1.0`
     - `solver = 'lbfgs'`
   - Those were the results:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.81      0.80     21197
           1       0.81      0.80      0.80     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 12: Experiment 2 - Evaluation

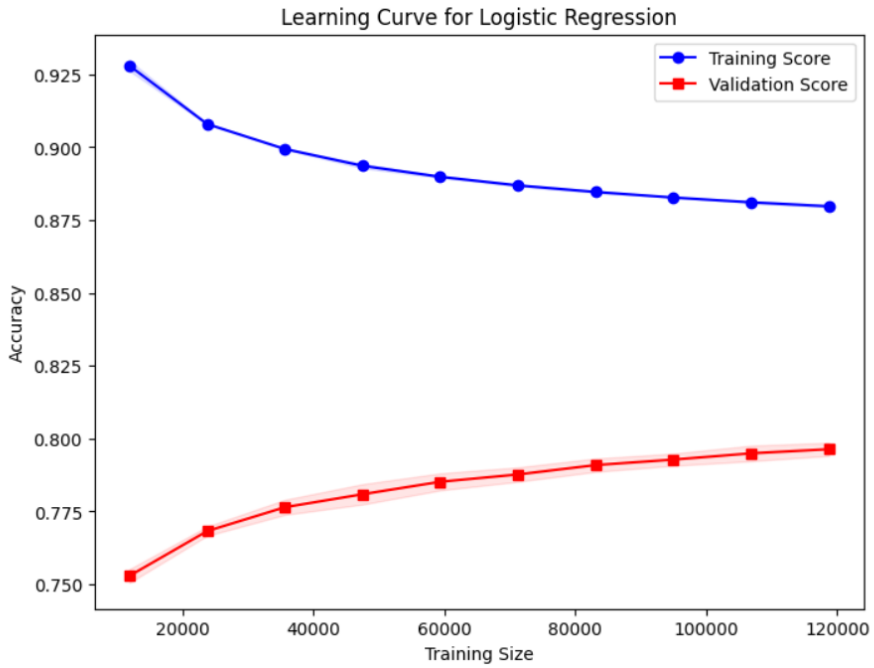

Figure 13: Experiment 2 - Plot

- The results showed a `0.1` improvement in accuracy, for this reason the decision was made to keep this vectorization approach.

3. Third Experiment

- My third experiment I again changed my vectorization approach and kept everything else the same.
  - `vectorizer = TfidfVectorizer(ngram_range=(1,3))`
  - `LogisticRegression(max_iter=500, random_state=random_seed)`
- This approach allowed the model to consider both single words (unigrams), two-word sequences (bigrams) and three-word sequences (trigrams) when extracting features improving the models ability to capture context, meaning, and relationships between words and leading to better accuracy.

- Those were the results:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.82      0.80     21197
           1       0.81      0.78      0.79     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 14: Experiment 3 - Evaluation

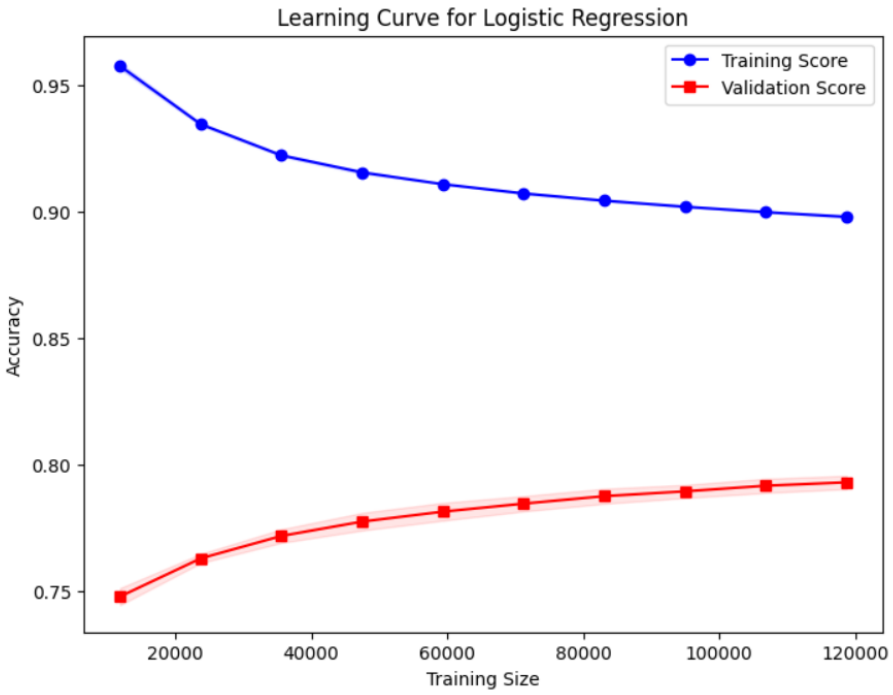

Figure 15: Experiment 3 - Plot

- The results are similar having the same accuracy, but with some important differences.
  (a) `Experiment 2`, had better balance between **precision** and **recall** meaning potential more stable results for **False Positives** and **False Negatives** respectively.
  (b) `Experiment 3`, also had better **f1-score** meaning all around fewer **False Positives** and **False Negatives**.
  (c) From the plot above some amount of overfitting is also noticeable.

For the reasons mentioned above, the following experiments and my final strategy use the `vectorizer = TfidfVectorizer(ngram_range=(1,2))` approach for vectorization.

4. Fourth Experiment

   - My fourth experiment used Grid Search and the vectorization found in `section 2.4` to find the best custom parameters from the following configuration:

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'saga', 'sag'],
    'max_iter': [100, 500, 1000]
}

grid_search = GridSearchCV(
    LogisticRegression(random_state=random_seed),
    param_grid,
    cv=2,
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)
```

   - Those were the best parameters it choose:
     `Best Parameters: 'C': 10, 'max_iter': 100, 'solver': 'saga'`
   - This strategy produced the following evaluation and plot:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.81      0.80     21197
           1       0.80      0.80      0.80     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 16: Experiment 4 - Evaluation

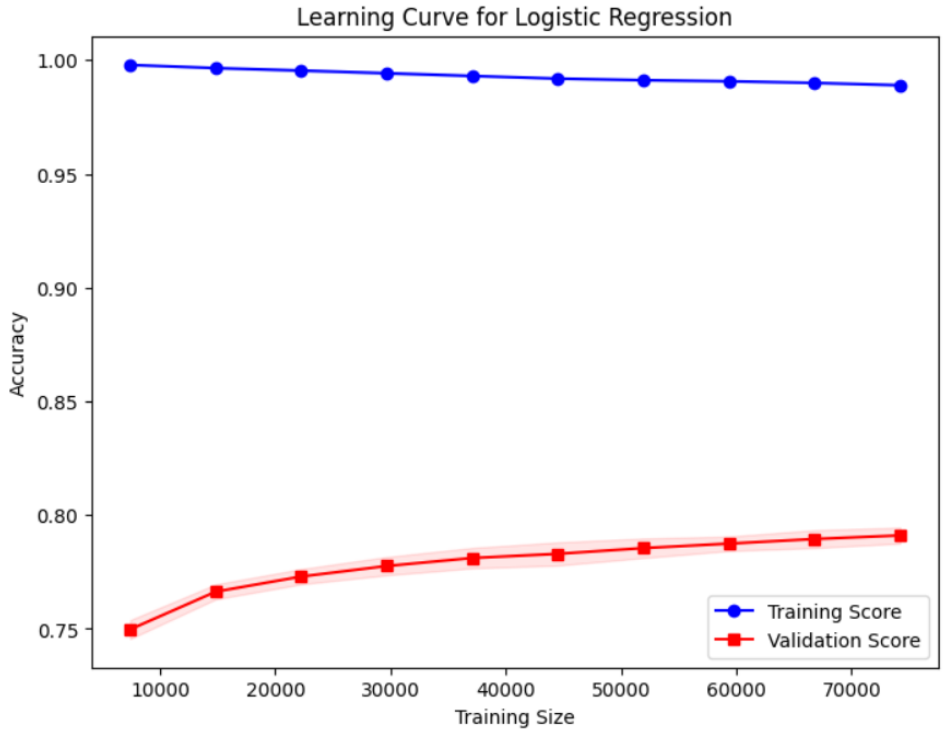

Figure 17: Experiment 4 - Plot

- There are two issues with those results:
  (a) The evaluation results are almost the same with `Experiment 2`, only difference being `Experiment 2`, having a slightly higher precision (`0.01 difference`).
  (b) The plot curves of this experiment show significant overfitting.
- For the above reasons we consider `Experiment 2` as having the best results so far.

5. Fifth Experiment

- The fifth experiment only changes with the fourth one at `cv`, specifically instead of `cv=2` now it uses `cv=3`
- Those were the best parameters it choose:
  `Best Parameters: 'C': 10, 'max_iter': 100, 'solver': 'saga'`
- Both the evaluation statistics and the plot are the same with `Experiment 4`, due to this, the same reasons for rejecting it apply.
- cv greater than 2 seems to not affect our results.
- `Experiment 2`, is still considered the best performing.

6. Sixth Experiment

- This experiment changes from `Experiment 5` only on the fact that `'lbfgs'` is added to the `solver` parameter list.
- Those were the best parameters it choose:
  `Best Parameters: 'C': 10, 'max_iter': 500, 'solver': 'lbfgs'`

- This model produced the following evaluation and plot:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80     21197
           1       0.80      0.80      0.80     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 18: Experiment 6 - Evaluation

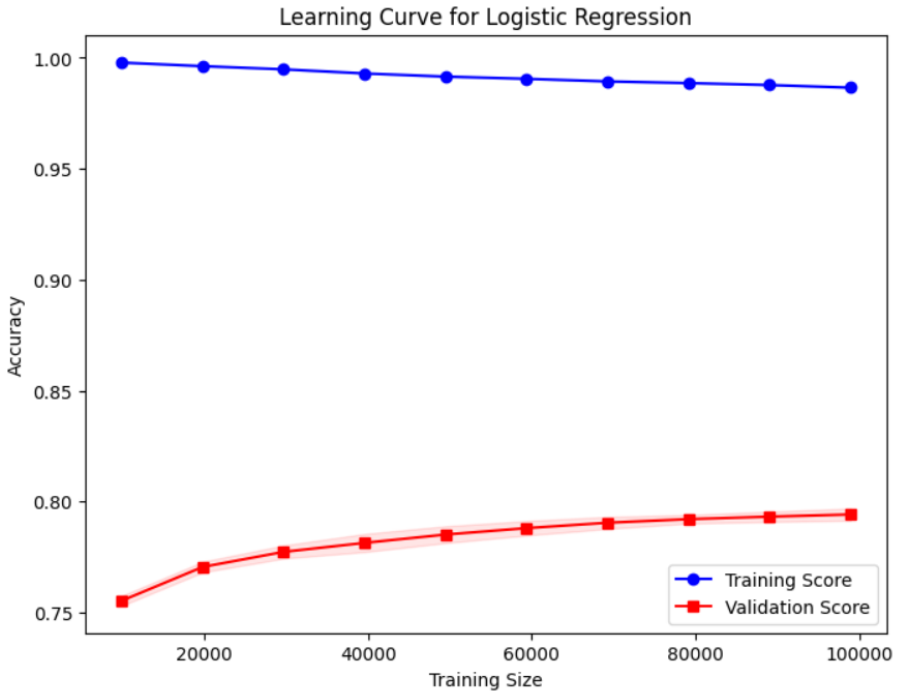

Figure 19: Experiment 6 - Plot

- Interestingly by adding the `lbfgs` solver, grid search choose different parameters this time.
  - It actually choose almost the default parameters as `Experiment 2` only difference being the `C` parameter.
  - `Experiment 2 -> C=1`
  - `Experiment 6 -> C=10`
- The model also got a big worse on precision and recall metrics than `Experiment 2`.
- The plot also shows significant overfitting.

- Since only difference between `Experiment 2` and `Experiment 6` is the `C` parameter as described above we can understand that overfitting is caused because of it. This make sense considering that smaller values of this parameter specify stronger regularization meaning simpler model and potentially less overfitting.
- `Experiment 2`, is still considered the best performing.

7. Seventh Experiment

- This experiment tries a new approach on the preprocessing step. Specifically it removes the stop-words from the initial data using the following:

```
vectorizer = TfidfVectorizer(ngram_range=(1,2), stop_words='english')
```

- The parameters used here are the ones from `Experiment 2`.
- This model produced the following evaluation and plot:

```
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.76      0.77     21197
           1       0.76      0.78      0.77     21199

    accuracy                           0.77     42396
   macro avg       0.77      0.77      0.77     42396
weighted avg       0.77      0.77      0.77     42396
```

Figure 20: Experiment 7 - Evaluation

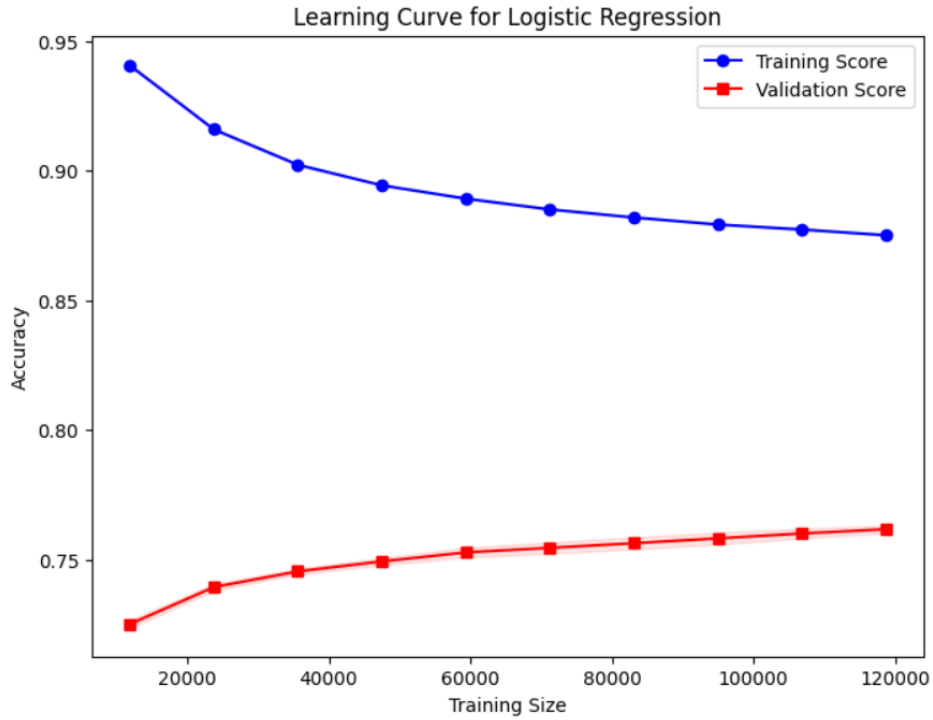

Figure 21: Experiment 7 - Plot

- The accuracy drops by `0.03` points, so we reject this approach.
- `Experiment 2`, is still considered the best performing.

Trial	Accuracy Score
1	0.79
2	0.80
3	0.80
4	0.80
5	0.80
6	0.80
7	0.77

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning

1. Final Results:
    - The final results are the ones of `Experiment 2`:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.81      0.80     21197
           1       0.81      0.80      0.80     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 22: Final - Evaluation

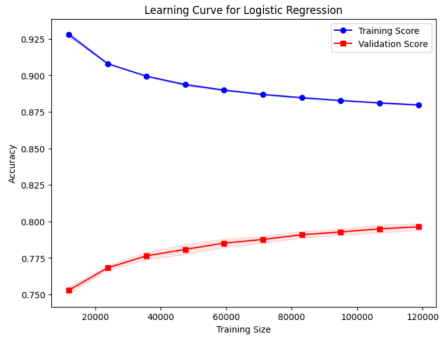

Figure 23: Final - Plot

- The result seem to get around the same value of 0.80 for all metrics, this demonstrates a balanced performance and indicates an effective classification without significant bias.

2. Model Configuration:
   - `vectorizer = TfidfVectorizer(ngram_range=(1,2))`
   - `model = LogisticRegression(max_iter=500, random_state=random_seed, C=1.0, solver='lbfgs')`
3. Under-Fitting & Over-Fitting:
   - No under-fitting was observed in all of the experiments.
   - While the model generalizes reasonably well considering the scores of the metrics mentioned at the `Final Results`, the gap between the training and validation curve suggests a potential overfitting.
   - The continued improvement in validation score with larger datasets also suggests that gathering more data could further enhance the model's overall performance.

3.3. Optimization techniques

1. I used GridSearchCV, an optimization framework that systematically searches for the best hyperparameter values.
2. For regularization, I experimented with a variety of C values, which control the strength of regularization of the model.

3. Additionally, I tried different n-gram vectorization techniques to find the best trade-off between capturing more contextual information and managing the dimensionality of the feature space.

4. I did a preprocessing step by removing stop words, lower casing text, and many other fixes to standardize the vocabulary. This helped clean the data, reduce noise, and improve model accuracy.

3.4. Evaluation

- I evaluated the predictions using accuracy, precision, recall, and F1-score.
- Accuracy reached 80%, showing the general accuracy of the model. Precision and recall were around 80% for each experiment, leading to an F1 score of also around 80%, indicating a good balance between false positives and false negatives.
- A learning curve was plotted to analyze model performance across different training sizes, helping to detect overfitting or underfitting trends.
- The following table shows the average metrics between the two classes of each experiment.

Experiment	Accuracy	Precision	Recall	F1-Score
1	0.79	0.79	0.79	0.79
2	0.80	0.80	0.80	0.80
3	0.80	0.80	0.80	0.80
4	0.80	0.80	0.80	0.80
5	0.80	0.80	0.80	0.80
6	0.80	0.80	0.80	0.80
7	0.77	0.77	0.77	0.77

Table 2: Experiments

3.4.1. Learning Curve.

- The model seems to effectively learn from increasing data, as shown by the rising validation accuracy. While a gap between training and validation scores indicates some potential overfitting, the continuous improvement suggests that this would be decreased with a larger number of data.

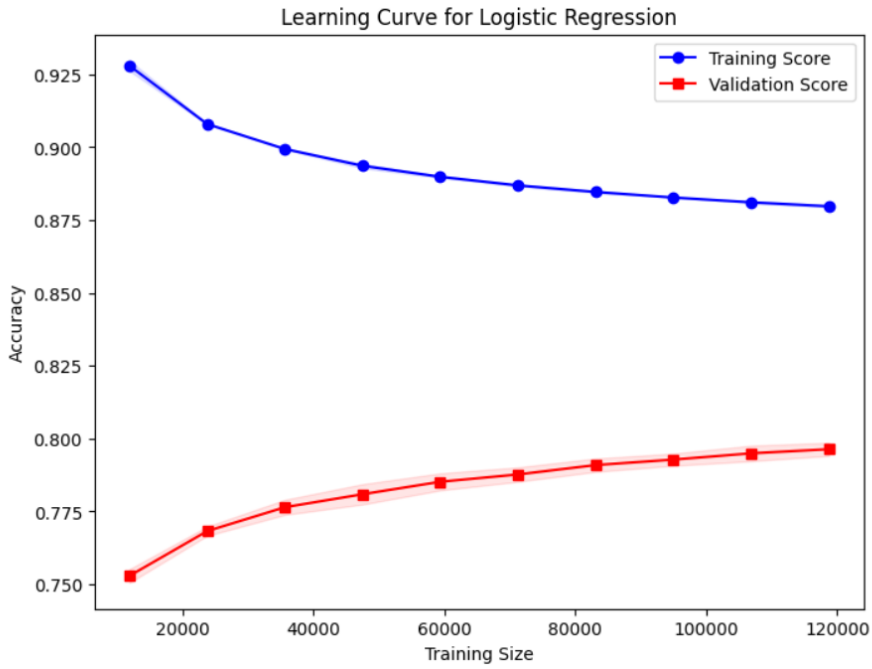

Figure 24: Learning Curve

3.4.2. Confusion matrix.

- True Negatives (TN) = 17,132
- False Positives (FP) = 4,065
- False Negatives (FN) = 4,295
- True Positives (TP) = 16,904

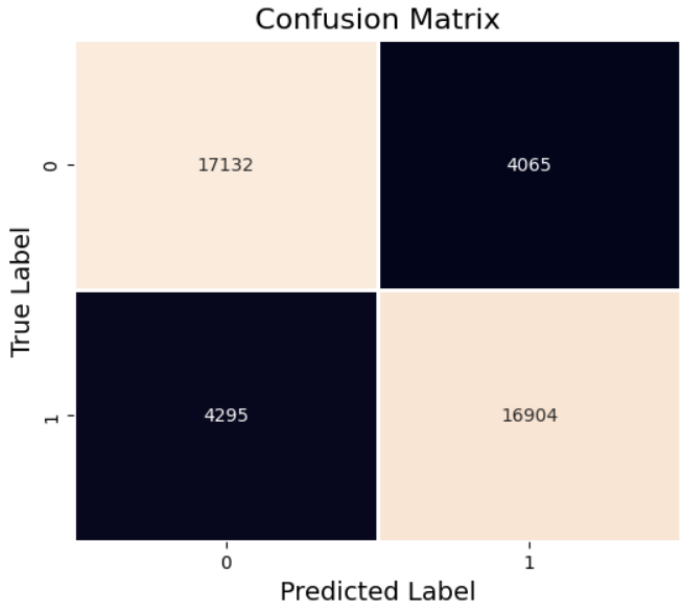

Figure 25: Confusion Matrix

- The model performs well overall, as the numbers for correct predictions (TP & TN) are significantly higher than incorrect ones.

4. Results and Overall Analysis

4.1. Results Analysis

- My final results show an accuracy of 80%, meaning that the model makes a correct sentiment prediction 80% of times, metrics such as precision, recall and f1-score also show that the model is balanced.
- If I could run more experiments, I would train the model on a larger dataset since the learning curve shows signs of overfitting, though it improves as the training data increases.

4.1.1. Best trial.

- Evaluation & Plot:

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.81      0.80     21197
           1       0.81      0.80      0.80     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

Figure 26: Experiment 2 - Evaluation

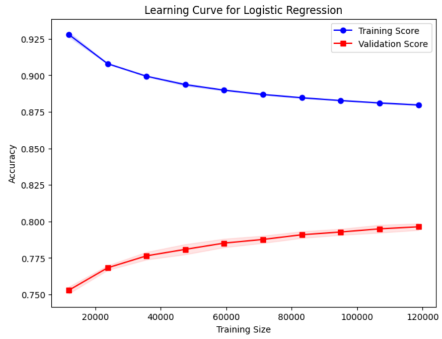

Figure 27: Experiment 2 - Plot

- Look at experiment 2 for more information.

5. Bibliography

References

[1] Confusion Matrix. Accessed: 2024-03. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[2] Grid Search. Accessed: 2024-03. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[3] Learning Curve. Accessed: 2024-03. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html.

[4] Logistic Regression. Accessed: 2024-03. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[5] Tfidf Vectorizer. Accessed: 2024-03. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.