NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

Student's full-name: Spanos Nikolaos
Academic Number: 7115112100023
Teacher: Koumparakis Manolis

1st academic homework exercise of course Artificial Intelligence II
Academic year: 2022-2023
Master of Science in Computer Science

Athens, November 2022

Contents

Sentiment analysis on imdb movie reviews

The code is thoroughly developed in the Google Colab notebook (Project I.ipynb) included in the project deliverables. This report will provide a thorough description of the results and the steps followed to develop and evaluate different model classifiers for the sentiment analysis on imdb movie reviews.

1. Importing the dataset & Data exploration

The dataset contains three columns,

- URL of the user comment (hyperlink)
- Rating (positive number from 1.0 to 10.0)
- Review (text describing the opinion of the user for a specific movie)

Firstly, I checked the frequency of the ratings among the dataset. This helped me to understand the dispersion of the movies across the ratings. A frequency table is presented below,

Rating	Frequency
1.0	20.79%
10.0	19.51%
8.0	11.67%
4.0	10.24%
3.0	9.72%
7.0	9.56%
9.0	9.27%
2.0	9.23%

Table 1 - Frequency of review ratings in sample

The most frequent reviews are either very negative, with a rating of 1.0, or very positive, with a rating of 10.0. The rest ~45% of the dataset have ratings in the intermediate scale.

Next, as per the project's description, I have grouped the numerical rating column into a binary group of two values [0, 1]. More specifically,

- 0.0 - 4.0: Negative
- 7.0 - 10.0: Positive

The new frequency table looks like:

Rating	Frequency
1.0	50.01%
0.0	49.99%

Table 2-Frequency of reviews in a binary group

The dataset is almost balanced among the two target classes positive and negative respectively. With the positive reviews to slightly dominate. To resolve this dominance, I have selected the top 20,000 reviews per sentiment group. The main reason to resolve the dominance of the positive class is because I don't want any model classifier to be biased over the positive class.

2. Preparing data samples

Every dataset should be prepared before starting any machine learning experiment. The transformation/preparation methods described in this part, are applicable to mostly any dataset existing out there. The are principles describing how to prepare an unbiased data sample for any machine learning algorithm.

Remove duplicate rows

The first method checks for identical rows, across all data features, and removes the identical rows by keeping the first occurrence. Having the same review twice could result in one of the following scenarios:

a. Same review fed to the model classifier. Feeding the model classifier many times (more than 2 times) with the same review could result to overfit and bias over specific words.
b. Higher volume of words/tokens and trainable features that could cost more resources and higher training times.

Since in the experiment I deal with movie reviews, I have dropped rows with the same review more than two times. I kept the first occurrence of the review.

Totally 72 reviews have been spotted more than two times.

Resampling dataset rows

In the end of the previous part, I referred to a resampling method to eliminate the dominance of the positive sentiment reviews and balance the dataset towards the negative and positive labels. It's of outmost importance when dealing with classification problems to have the same balance per class. If the latter cannot be achieved then the imbalance should be kept across all sample splits (train-test split, cross-validation, etc).

After the resampling the final dataset has 40,000 reviews.

- 20,000 positive reviews.
- 20,000 negative reviews.

Rating	Frequency
1.0	50.00%
0.0	50.00%

Table 3-Frequency of labels after resampling

Shuffling

To perform the second method, *resampling of dataset,* I had to take the first 20,000 positive reviews and then append them to 20,000 negative reviews. Resulting to a strict ordered sample. To avoid the scenario of feeding a machine learning algorithm with only positive reviews and test its prediction ability on only negative reviews we should re-shuffle the 40,000 sample reviews. The shuffle is random.

3. Preprocessing text samples

Having the dataset ready for machine learning experiments, in this section I will focus on preprocessing the trainable features for Natural Language Processing text classification.

The transformation methods applied on the imdb reviews:

- Remove emojis.
- Remove special characters such as punctuation, html tags, extra spaces.
- Lowercase review comments.
- Transform abbreviations. For example, I'm -> I am, It's -> It is.
- Remove stop words (most frequent words such as part of speech, i.e., a, the, it, she, an, etc.).

- Remove numerical characters from text (i.e., 1, 2, 100, 124, etc.).
- Text normalization technique | Lemmatization or Stemming.

Lemmatization was applied instead of stemming. The basic difference of those two methods is that Lemmatization considers the part-of-speech identity of a word before applying the normalization. Whereas Stemming is a more heuristic approach that strips off the suffix from a word without taking into consideration any additional info about the token's placement in the sentence.

A simple example:
cries => cri (stemming)
cries => cry (lemma)

Movie review before text preprocessing

*The movie has started, the wheels spin, your car has entered a race against the Fox.... You're behind you can't get in front, you figure, "if i go out, i'm taking you with me..." You smash into the cars parked on the side of the road, you turn to hit Fox but your aim is bad.... Bang, you've gone up the back of a VW, and smash you've landed in a reservoir. Your car lights on fire. You could get out if you wanted, but the shame of losing has taken you... BANG! Your car blows up, everyone looks on in despair, some crying... The sound of a siren tells you the cops are coming. Everyone gets in their cars and bolts, leaving you to burn. The charred remains of the cars frame sits there, haunting the on lookers... You're dead.

This is one of the scenes; actually its the first scene in the movie. There are many more like it. As you enter the cockpit of the Fox in his pimped out V8 ford, Terry Serio in his crazy GTHO, and many others in this blast from the past.... JOHN CLARKS masterpiece, "RUNNING ON EMPTY" "-He'll win at any cost-"*

Movie review after text preprocessing

movie start wheel spin car enter race fox behind get front figure go take smash car park side road turn hit fox aim bad bang go back vw smash land reservoir car light fire could get wanted shame lose take bang car blow everyone look despair cry sound siren tell cop come everyone get car bolt leave burn char remain car frame sits haunt looker dead one scene actually first scene movie many like enter cockpit fox pimp v ford terry serio crazy gtho many others blast past john clark masterpiece run empty win cost

For the final step of preprocessing, although not affecting the text content of the reviews, I applied a split of the data samples into three sub-samples.

- Training sample: Use to feed the machine learning classifier
- Validation sample: A proportion of the training samples to use for comparing and selecting different classifiers.
- Test sample: The never-seen-before reviews to test predictive power of the selected model classifier.

The initial ratios:

- 80% training samples – 20% test samples [32,000 training – 8,000 testing]
- 80% fitting samples – 20% validation samples [25,600 fitting – 6,400 validation]

Really important to apply stratification. This process will assure that the ratio (50-50) of each sentiment will be preserved among the split samples (train-validation-test).

4. Text Vectorization

Three different techniques have been used for text vectorization. Turning sentences to tokens and then vectors. The techniques are selected based on their level of complexity from the most trivial to the more sophisticated approach.

4.1 CountVectorizer

First, I applied the *CountVectorizer* method to the whole dataset, including unigrams and bigrams, using the default settings of the algorithm:

- lowercase = True,
- analyzer = 'word',
- ngram_range = (1,2),
- min_df = 1,
- max_df = 1.0,
- max_features = None (all the words/tokens of the corpus)

The initial shape of the vocabulary was more than 2,609,487 tokens (including bigrams and unigrams). A large and computationally intensive vocabulary. Sparse vectors with an incredibly high number of integers could possibly add noise to the model classifier, overfit over training samples and decrease the generalization ability of the classifier.
To reduce the size of vectors and remove noise from text I changed the values of the following three arguments of the *CountVectorizer* method.

- min_df = 5 (from default value 1)
- max_df = 0.95 (from default value 1.0)
- max_features = 1500 (vector size)

The vocabulary dropped from 2,609,487 to 1500 words. 1500 of the top importance words from the whole vocabulary. Note that many unique words could be actor names, formal English vocabulary, genres, etc. In general words with no meaning or strictly used on a specific review. The goal is to distinguish between positive and negative reviews. Thus, we need to find words that are not shared among the two sentiments. Since, *CountVectorizer,* was the first vectorizer I tested, I also built a baseline model to compare the rest of the model classifiers trained with different techniques.
For the results of model training with features vectorized by CountVectorizer go to unit Model training | CountVectorizer.

4.2 TF-IDF Vectorizer

Next in the experimentation agent is the Term Frequency - Inverse Document Frequency Vectorizer. Stands for a statistical approach that evaluates the number of times a word is found in a collection of documents plus of how important a word is to the given corpus. Both TF-IDF and CountVectorizer belong to a group of vector techniques related to the traditional approach of Bag-of-Words.
TF-IDF method parameters used:
- ngram_range = (1,2). Retrieve uni-grams and bi-grams (i.e., 'posterior', 'probability', 'posterior probability').
- lowercase = true (default)
- max_df = 0.95 (default)
- min_df = 5 (default)
- max_features = 1500

For the model training results go to unit Model training | TF-IDF Vectorizer.

4.3 Word2Vec Vectorizer

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector of embeddings. The vectors are chosen carefully such that they capture the semantic and syntactic qualities of words. (Source: https://en.wikipedia.org/wiki/Word2vec)

I have performed three different tests with the samples vectorized by the Word2Vec model.

- Skipgram with 300 vector size and minimum token frequency count 10 times.
- Skipegram with 64 vector size and minimum token frequency count 100 times.
- CBoW with with 300 vector size and minimum token frequency count 100 times.

Overall, out of the three word2vec models, the first one dominated the rest of the two, with more accurate predictions and learning curves. For more on the code behind the last two tests, check the units 9.5 and 9.6 from the code included in the deliverables.
For the model's performance on the first option of word2vec vectorizer the reader should check unit 7 *Model training | Word2Vec Vectorizer* .

5. Model training | CountVectorizer

5.1 Baseline Logistic Regression

For the baseline model classifier, I didn't apply either any regularization technique or greatly deviate from the default values of the Logistic Regression classifier.

	precision	recall	f1-score	support
negative	0.88	0.86	0.87	3164
positive	0.86	0.88	0.87	3236
accuracy			0.87	6400
macro avg	0.87	0.87	0.87	6400
weighted avg	0.87	0.87	0.87	6400

Evaluation metrics: Baseline model

precision score: 0.8695
recall_score: 0.8691
f1_score: 0.8691

model bias: 0.1189
model variance: 0.2497

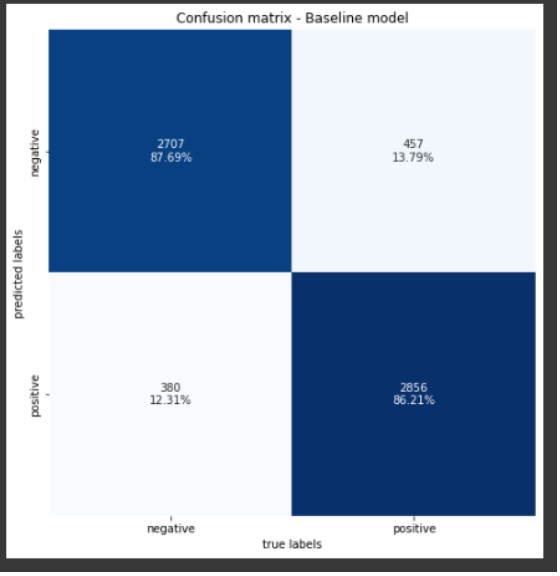

Figure 1-Evaluation metrics of Baseline Logistic Classifier

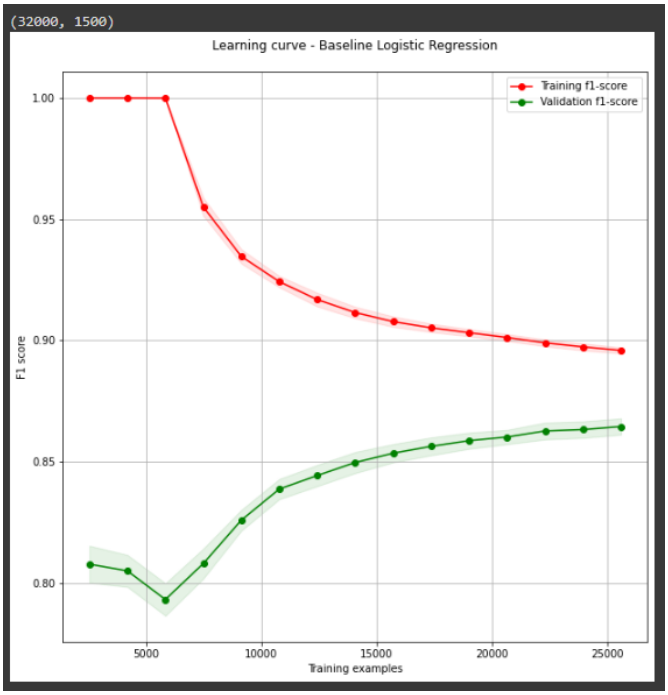

Figure 2- Learning Curve of Baselne Logistic Classifier

Comments

The first try of the CountVectorizer with the default values of the Logistic Regression classifier yielded a quite precise model algorithm with overall F1-score of 0.869. The classifier's bias is 0.1189. This can be also verified by the thickness of the red learning curve. The smaller the bias the thicker the red shadow space around the training learning curve. The bias represents the model's uncertainty between the predictions and the actual values of the review sentiments. A high biased model cannot correctly identify any special pattern or noise in the training data and leads to an oversimplified model estimator. The baseline model classifier did not underfit.

However, another interesting finding for the baseline model classifier is green learning curve, which represents the validation scores. The green learning curve could inform us for the effect of the model's variance. With a value of ~0.2497, the model classifier could slightly overfit over training samples and loose certainty over predicting new samples. That is why the green shadowed space is much more distinct in than the red shadowed space of the red learning curve.

The generalization gap between the red and the green lines is quite narrow however until the two lines come very close to each other, it's not permanent to assume that the model could make accurate predictions on never-seen-before data.

Moving on I have tried three different approaches to minimize the generalization gap between the training and cross-validation learning curves. To do so, I have tested the following methods.

5.1.1 Feed baseline logistic regression with more training samples

By keeping the rest of the arguments same, I only changed the train-test split ratio from 20% to 10% for the testing samples. Raising to 36,000 trainable samples from 32,000.

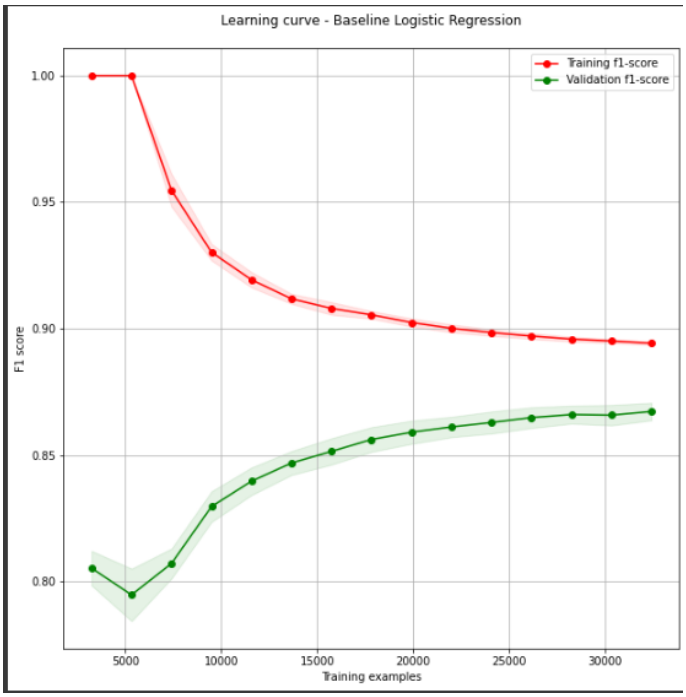

Figure 3-Learning curves after increasing training samples

The generalization gap indeed become narrower. However, the gap is still wide making the predictive power of the model classifier week.

5.2 GridSearchCV – Hyper parameter tuning

Since increasing the training samples didn't have the expected results, then I trained the model by introducing regularization and applying a different set of values on some default parameters used for the baseline model. Based on the outcome of the previous test, I changed the split ratio of the training samples ratio from 80& to 90% training reviews.
More specifically the grid used:

- penalty: ['l1', 'l2']
- solver: ['lbfgs', 'liblinear', 'saga']
- C: [0,5, 07, 0.9]
- max_iterations: [25, 66, 101, 150]
- Fit on f1-score
- Cross-validation with 15 splits and test size 10%

Note that the splits in cross-validation represents the number of splitting iterations the model is re-fit and is then scored on 10% of the samples. The 15 f1-scores are collected and then averaged to give one f1-score representing the model classifier's performance. The classifier with the highest average f1-score is selected and its hyper-parameters are saved.

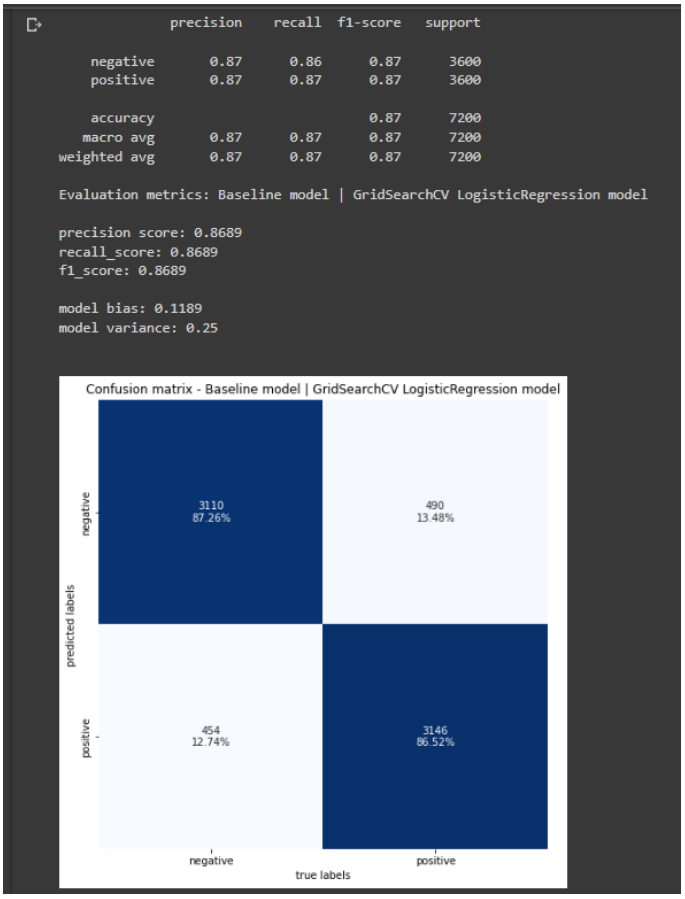

Figure 4-Evaluation Metrics GridSearchCV

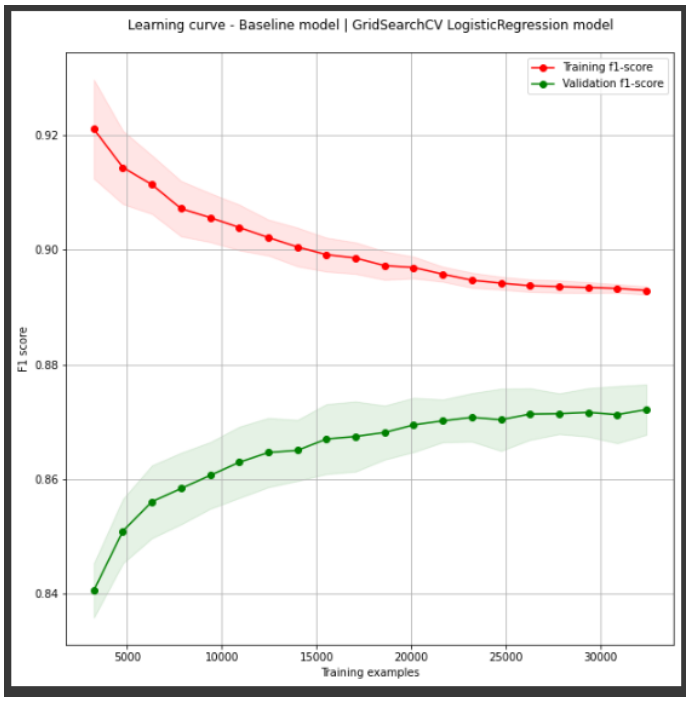

Figure 5-Learning Curves GridSearchCV

Comments

The results after hyper-parameter tuning with cross validation didn't improve compared to baseline model. The f1-score slightly dropped, the bias remained the same and the model variance increased. Even the learning curves are far wider than the baseline model classifier, and there is great dispersion over the cross-validation scores. Overall, hyperparameter tuning didn't make any difference in improving baseline model's prediction power. Still the baseline model dominates.

Best model hyper-parameters:
best parameters: {'C': 0.5, 'max_iter': 66, 'penalty': 'l1', 'solver': 'saga'}

5.3 Testing regularization on baseline model

Moving with the third testing approach, I applied regularization on the baseline model classifier without taking into consideration the best model parameters after hyper-parameter tuning.

Testing L1 regularization

Hyper-parameters of the baseline model:

- multi_class = ovr,
- max_iter = 150

Solver = saga
Had to change the solver from lbfgs to saga, because the former does not support l1 regularization. Also, saga was selected from the hyper-parameter tuning test).
Penalty = L1 regularization (*Lasso regression*).
C = 0.8 (Inverse of regularization strength, smaller values specify stronger regularization).

```
              precision    recall  f1-score   support

    negative       0.87      0.86      0.87      3600
    positive       0.87      0.87      0.87      3600

    accuracy                           0.87      7200
   macro avg       0.87      0.87      0.87      7200
weighted avg       0.87      0.87      0.87      7200

Evaluation metrics: Penalized L1 LogisticRegression model

precision score: 0.8693
recall_score: 0.8693
f1_score: 0.8693

model bias: 0.1193
model variance: 0.25
```

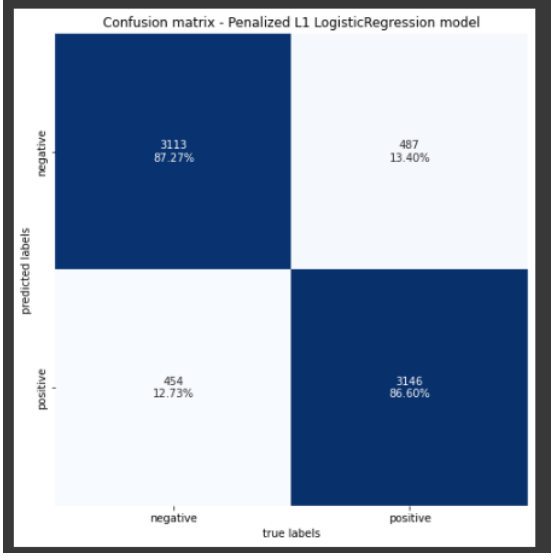

Figure 6-Evaluation metrics L1 regularization

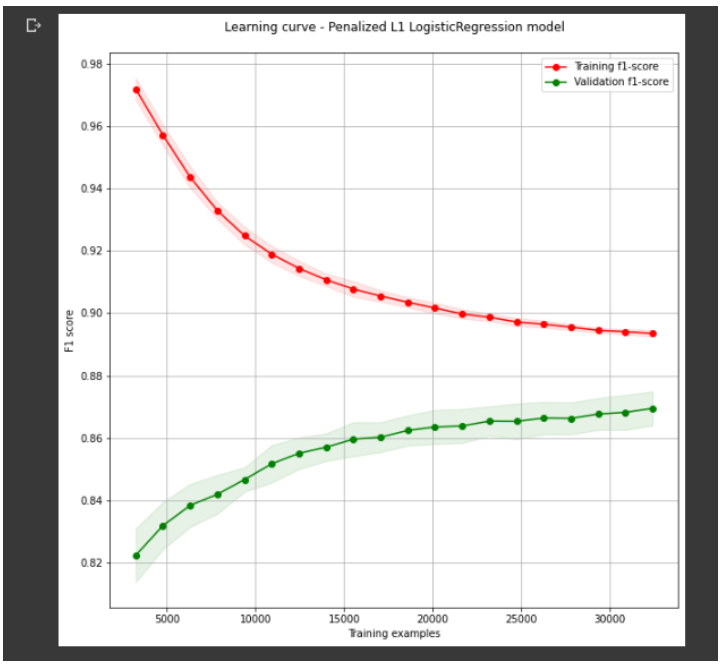

Figure 7- Learning Curves L1 regularization

Comments

Better results than GridSearchCV training but still poorer predictive performance than baseline logistic regression classifier.

Testing L2 regularization

Hyper-parameters of the baseline model:

- multi_class = ovr,
- max_iter = 150

Solver = lbfgs

Had to change the solver from lbfgs to saga, because the former does not support l1 regularization. Also, saga was selected from the hyper-parameter tuning test).

Penalty = L2 regularization (*Ridge regression*).

C = 0.8 (Inverse of regularization strength, smaller values specify stronger regularization).

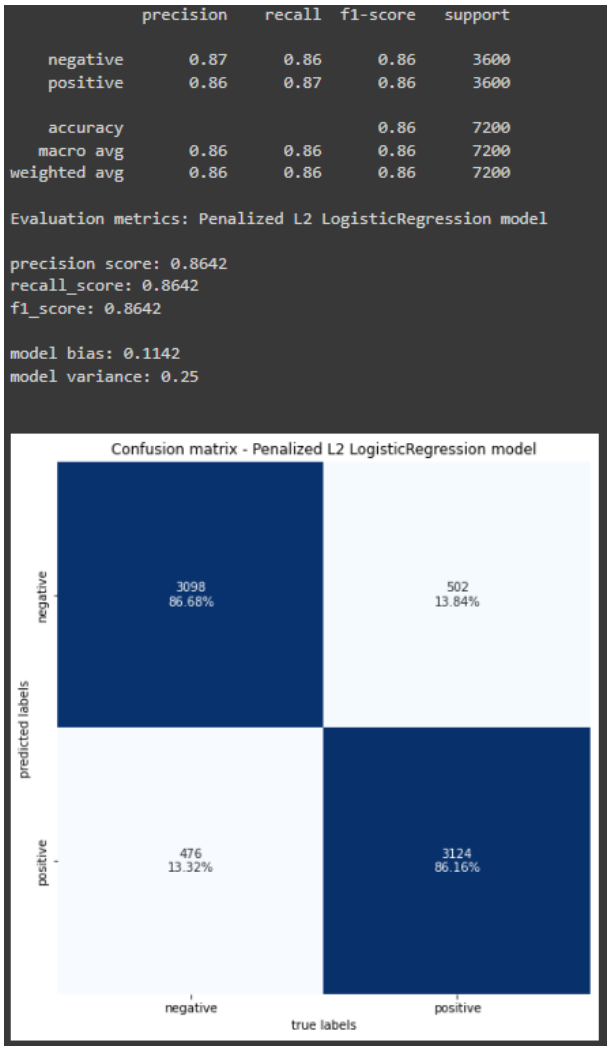

Figure 8- Evaluation metric L2 Regularization

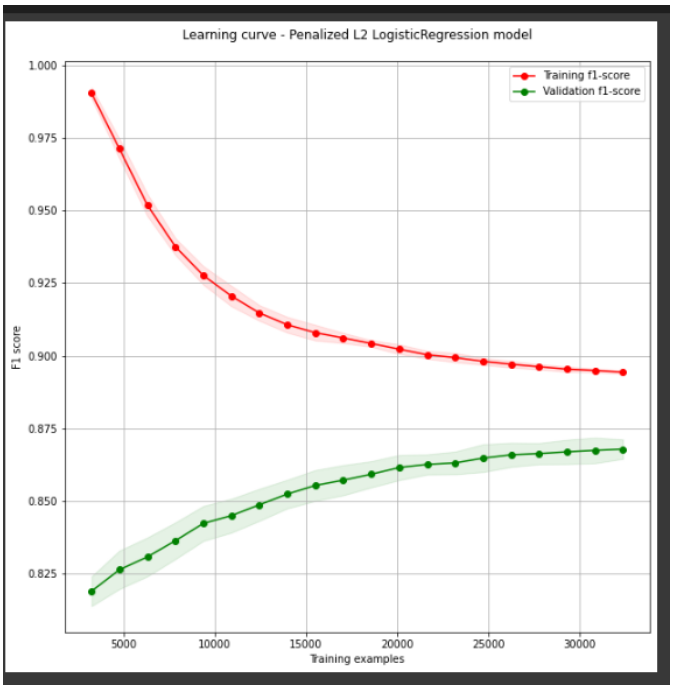

Figure 9-Learning curves L2 regularization

Ridge regression improved the prediction power of the baseline model classifier. Indeed, adding a penalization term during the model training improved f1-score and smoothed the learning curves. L2 regularization would be for sure applied in the next model classifiers trained. However, the generalization gap is still wide, and the model's variance did not drop from 0.25 units.

5.4 Increase/Decrease the vector size of CountVectorizer

For the final test I have trained two model classifiers. The first one, after decreasing the vector size of the trainable features by 1,000 words. Thus, 500 indexes used per vectorized sentence. The second model, trained after increasing the vector size by 1,000. Thus, 2,500 indexes per vectorized sentence.
I won't post any evaluation reports or learning curves because the results didn't present any promising improvement (for more details check the summary table. The grader may execute the code blocks of sections 7.6.1 and 7.6.2 of the notebook included in the deliverables. Although, find below the learning curve of the model classifier with *Ridge regularization,* after training 500 vector size reviews.

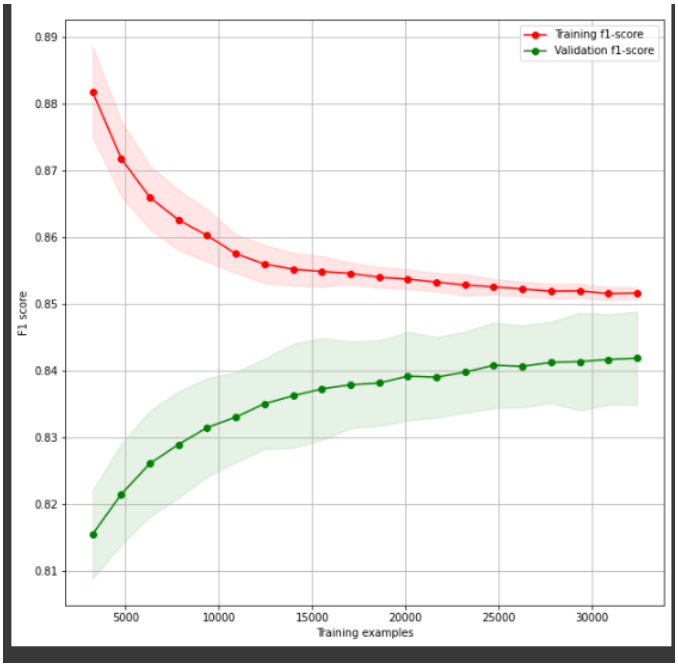

Figure 10-Learning curves 500 vector sized reviews

Summary table from the six (6) tests using *CountVectorizer.*

Test name	F1-score	Precision	Recall	Bias	Variance
Baseline model	0.8632	0.8632	0.8632	0.1132	0.25
GridSearchCV	0.8689	0.8693	0.8693	0.1189	0.25
L1 regularization	0.8693	0.8693	0.8693	0.1193	0.25
L2 regularization	0.8642	0.8642	0.8642	0.1142	0.25
2,500 vector size	0.8646	0.8646	0.8646	0.1146	0.25
500 vector size	0.8443	0.8447	0.8444	0.094	0.2497

Table 4-Summary table CountVectorizer

The baseline model classifier vectorized by *CountVectorizer* dominates over the rest of the tests. A meaningful change could be the addition of L2 regularization.

6. Model training | TF-IDF Vectorizer

6.1 Baseline Logistic Regression

Initially I trained the baseline model classifier as described in unit 5. The baseline model classifier has the same arguments used with *CountVectorizer* application. Carrying the outcomes of the previous unit, the training-test split remains at 90-10% keeping 36,000 reviews for training and 4,000 reviews for testing.

```
              precision    recall  f1-score   support

    negative       0.87      0.86      0.87      3600
    positive       0.86      0.87      0.87      3600

    accuracy                           0.87      7200
   macro avg       0.87      0.87      0.87      7200
weighted avg       0.87      0.87      0.87      7200

Evaluation metrics: TFIDF | Baseline Logistic Regression

precision score: 0.8664
recall_score: 0.8664
f1_score: 0.8664

model bias: 0.1164
model variance: 0.25
```

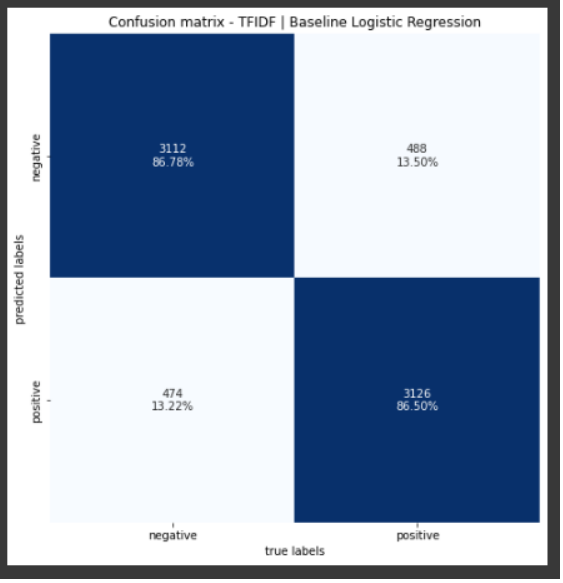

Figure 11-Evaluation metrics TF-IDF Baseline Logistic Regression

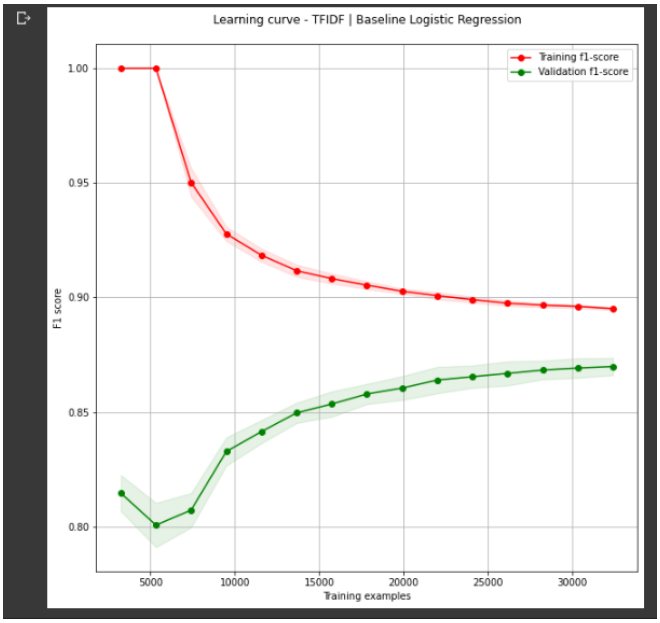

Figure 12-Learning curves TF-IDF Baseline Logistic Regression

Comments

Compared to the baseline model classifier trained with samples vectorized by *CountVectorizer,* with TF-IDF vectorization we achieved better f1-score, precision and recall results. Slightly higher (~+0.002) bias and same variance (~0.25). No regularization was applied for this experiment.

A note on L2 regularization. The experimentation with l2 regularization didn't yield any improved results and won't be presented in this unit. For the results, the grader could check the section 8.2 from the delivered notebook.

6.2 GridSearchCV - Hyper parameter tuning

The GridSearchCV hyper-parameter tuning test used the same hyper-parameter grid and settings with the CountVectorizer experminets in unit 5.2.
The evaluation report had the same poor performance with a promising f1-score of 0.8740 (the highest reached so far). However, bias and variance were higher than the baseline model classifier. The scored metrics can be found in the summary table at the end of the unit.

best parameters: {'C': 0.6499999999999999, 'max_iter': 50, 'penalty': 'l2', 'solver': 'liblinear'}

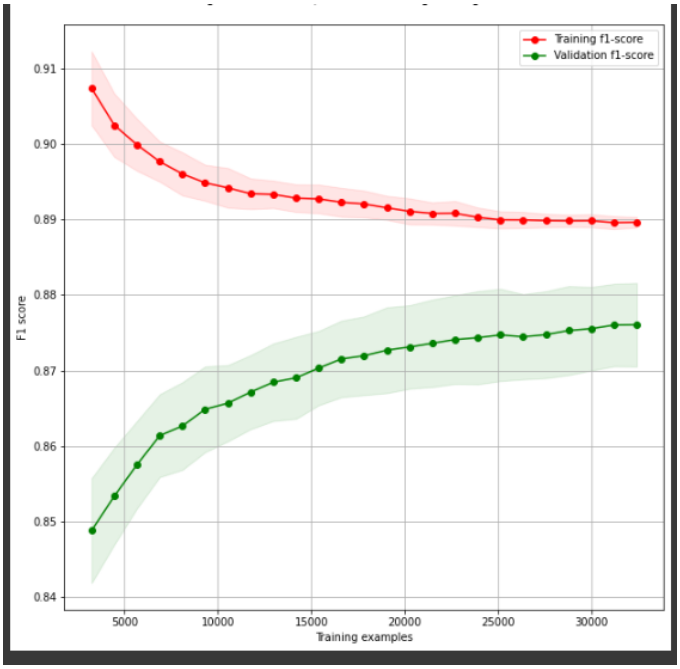

Figure 13-Learning curves TF-IDF GridSearchCV

Comments

The training f1-score yielded a smooth performance at around 0.88 value with very low variance for the training scored values per batch iteration. However, the validation f1-score for never-seen before data had a very poor performance with high variance and disperse among the scored samples.

Summary table from the three (3) tests using *TF-IDF Vectorizer.*

Test name	F1-score	Precision	Recall	Bias	Variance
Baseline model	0.8664	0.8664	0.8664	0.1164	0.25
L2 regularization	0.8710	0.8710	0.8710	0.121	0.25
GridSearchCV	0.8740	0.8743	0.8741	0.1239	0.2499

Table 5-Summary table TF-IDF Vectorizer

7. Model training | Word2Vec Vectorizer

This section tests the model performance of three different model settings:

- Baseline Logistic Regression
- L2 regularization on baseline classifier
- GridSearchCV and hyper-parameter tuning

7.1 Baseline Logistic Regression

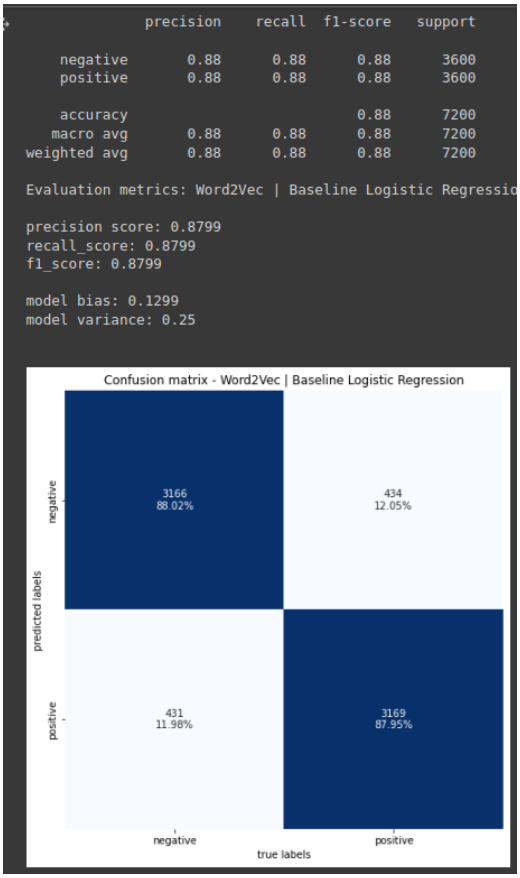

Figure 14-Evaluation metrics Word2Vec Baseline Logistic Regression

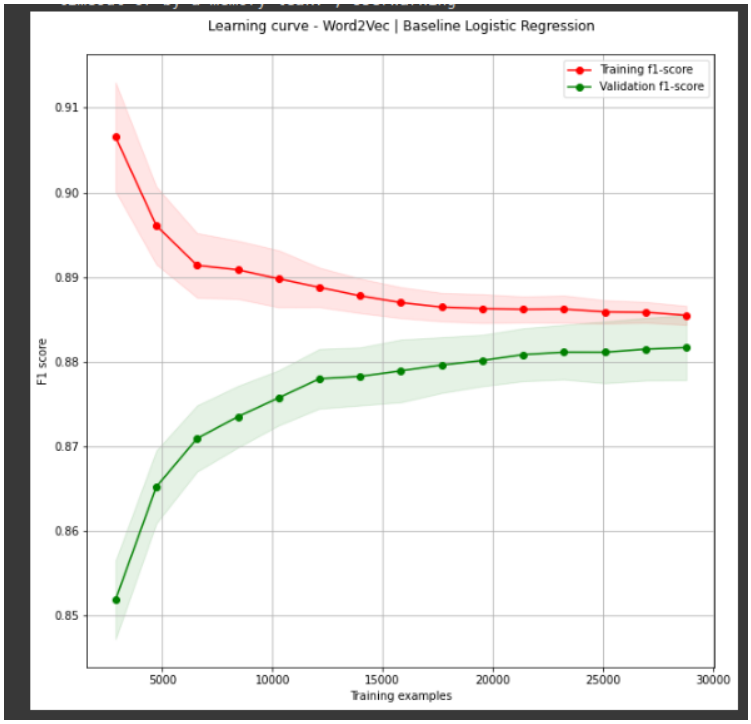

Figure 15-Learning curves Word2Vec Baseline Logistic Regression

Comments

The results are better than any other baseline logistic regression classifier. The f1-score is almost 0.88 units, the total false predictions are at 12% for both labels and model bias is 0.13 with variance still not dropping significantly below 0.25 degrees. The learning curves are much closer to each other, greatly minimizing the generalization gap between training and test scores. However, the 0.25 variance and 0.13 bias creates dispersion among the predictions (red and green shadow space). A strong option to consider as the final model classifier.

7.2 L2 regularization on baseline logistic regression

The evaluation report won't be reported because the baseline classifier didn't improve.
For this experiment's evaluation report, the reader may check the summary table at the end of the unit.

Find below the generated learning curves.

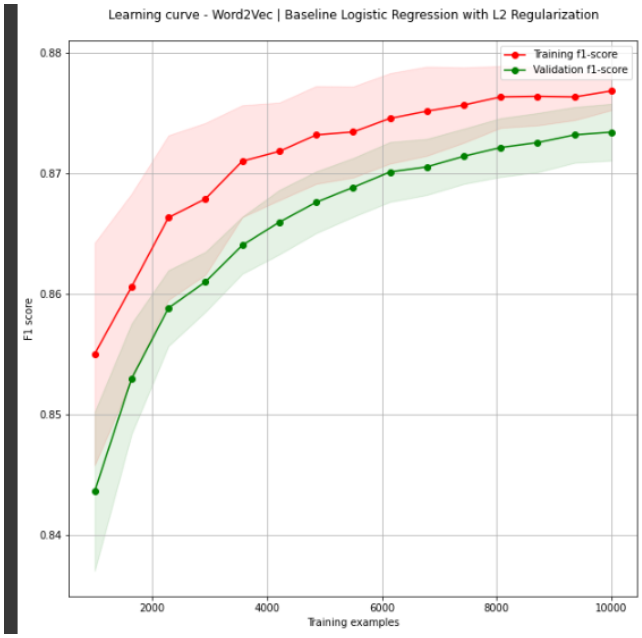

Figure 16-Learning curves L2 Regularization on Word2Vec Baselines Logistic Regression

7.3 GridSearchCV – Hyper parameter tuning

The hyper-parameter tuning method selected the following best parameters:

best parameters: {'C': 0.9, 'max_iter': 50, 'penalty': 'l1', 'solver': 'saga'}

With word2vec vectorizer the tuning selected *lasso regularization* and the *saga* solver compared to the predecessor vectorizer (TF-IDF) were *ridge regularization* was selected. The best parameters selected are closer to the *CountVectorizer* experiment. However, like the L2 regularization experiment, the results didn't improve compared to the baseline regression model fit.
For this experiment's evaluation report, the reader may check the summary table at the end of the unit.

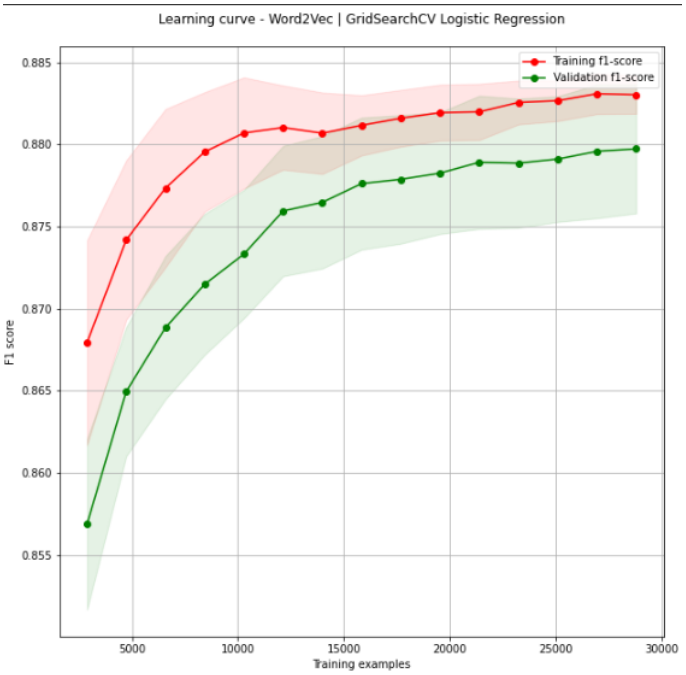

Figure 17-Learning curves GridSearchCV Word2Vec model

Summary table from the three (3) tests using *Word2Vec vectorizer.*

Test name	F1-score	Precision	Recall	Bias	Variance
Baseline model	0.8799	0.8799	0.8799	0.1299	0.25
L2 regularization	0.8920	0.8920	0.8920	0.142	0.25
GridSearchCV	0.8787	0.8788	0.8788	0.1288	0.25

Table 6-Summary table Word2Vec vectorizer

8. Conclusions from the training/validation experiments

- *Word2Vec vectorization* had the best training performance in terms of *f1_score*. However, the learning curves had a poor performance for two of the three tests. The model classifier trained was overfitting. However, for the baseline logistic regression fit had the best performing learning curves.
- *TF-IDF vectorizer* did well but also overfitted over the training samples.
- *CountVectorizer* even though having the lowest *f1-score* performance compared to the other two vectorization algorithms, did well in term of generalization in the learning curves. The model classifier trained with this algorithm had the lowest bias and variance results.
- Applying *L1 regularization* didn't improve the prediction performance of the model classifier.
- Applying *L2 regularization* slightly improved the baseline classifier performance. Ridge regression was selected as a preferrable regularization technique from hyper-parameter tuning and other tests.
- The average bias of logistic regression classifiers was ~0.11-0.12. Close to zero, denoting no sever underfitting of the model classifiers.

- The average variance of logistic regression classifiers was ~0.24-0.25. A slightly high variance value, denoting a possibility of overfitting algorithms. The notable fact is that after applying techniques like *regularization*, *cross-validation,* and under-sampling, I didn't spot any drop in the variance of the logistic regression classifiers.
- Even though non-baseline logistic regression experiments had slightly higher f1-scores, most of the classifiers failed to improve the bias of the test-scores and further generalized the model classifier. What it was also observed, which is quite interesting, was the behavior of regularization, cross-validation and different vectorizers in the classification report of different logistic classifiers. The ratio of true with correct predictions remained the same, while the ratio of wrong observations changed based on regularization or different vectorizers. Basically, the results didn't improve significantly because the classifiers once misclassified more negative reviews or in other situations misclassified more positive reviews. So, the changes applied, regularization, different vectorizers, cross-validation, didn't increase the correct predictions but rather affected the ratio of the incorrect predictions.
- Overall, I will select the *Word2Vec* as the vectorization method for the trainable parameters of the sample. This is because the training samples, vectorized with the *Word2Vec* model, demonstrated the smallest generalization gap in the learning curves for the baseline logistic regression model.
- Overall, the regularization and hyper-parameter tuning on the baseline logistic regression model did not improve the overfitting and generalization of the model classifier. As a result, **the baseline model classifier will be the best choice so far**. With sufficient bias-variance and f1-score metric after applying Word2Vec vectorization.

9. Decreasing the training samples

Before experimenting with more sophisticated algorithms, I run a test on greatly minimizing the training samples and increase the validation & test reviews.
The split ratios applied:

- Training ratio: 50% -> 20,000 samples
- Test ratio: 50% -> 20,000 samples
- Validation ratio: 10% of training samples -> 2,000 samples

Using the baseline classifier vectorized by Word2Vec embeddings the results are reported below.

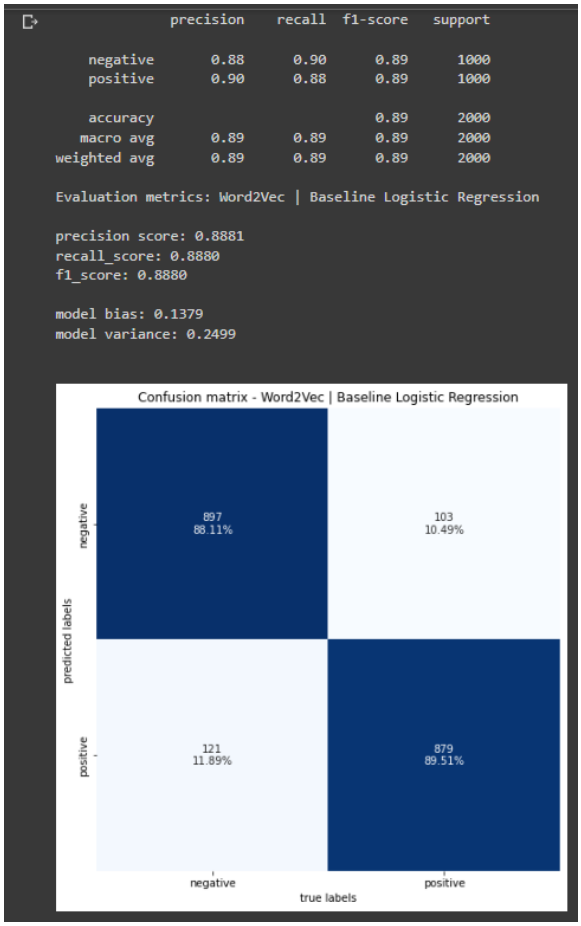

Figure 18-Evaluation metrics 50% training samples

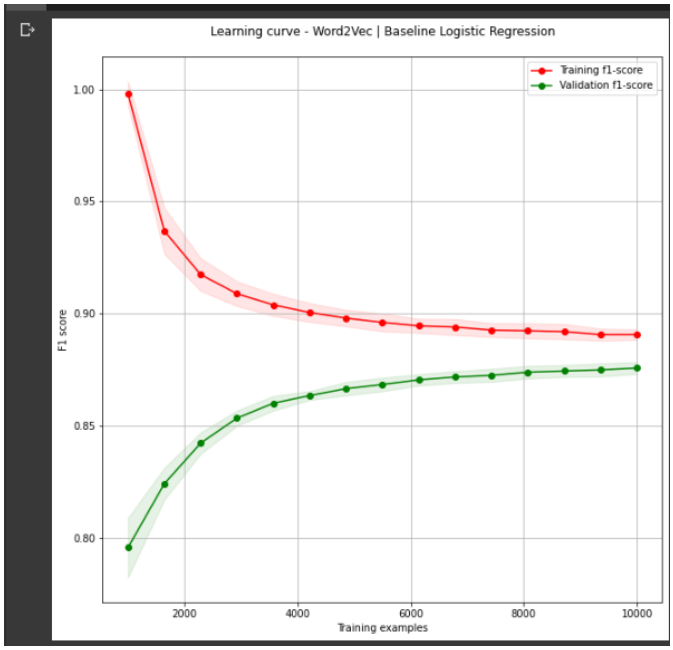

Figure 19-Learning curves 50% training samples

Comments

Almost 0.89 f1-score, precision, and recall. The highest values seen so far.
~10.50 and ~12% of false negative, false positive reviews totaling to 224 false predictions out of 2,000 reviews.
The smoothest learning curves and the narrower generalization gap than any other experiment.
The results could drive into the conclusion that 20,000 reviews could suffice in correctly predicting the review's sentiment.
Clearly the less sophisticated algorithm performed quite well after training in half of the reviews.

10. Random Forest Classifier

For the final two chapters I have experimented with two more sophisticated algorithms. The Random Forest classifier and Gradient Boosting Trees. The results are presented below.

Model classifier parameters:

n_estimators = 150 (default=100)
criterion = gini
oob_score = true
min_samples_split = 15
max_samples = 0.5
class_weight = "balanced"

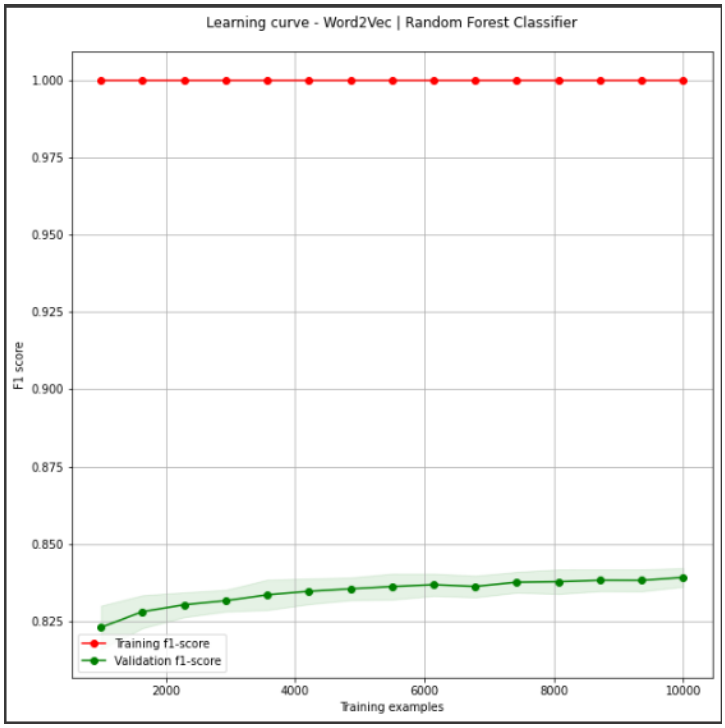

Figure 20-Learning curves Random Forest classifier

Comments

The experiment with the default parameters of the Random Forest had a bad performance on the training samples. The evaluation metric results were the lowest from all other experiments. The training f1-score did not drop at all, while the validation f1-score had a very low and steady performance. Based on the generalization gap between the two learning curves, the model classifier has a poor predictive performance on the test samples. The model classifier clearly underfitted.

Summary table from Random Forest classifier

Test name	F1-score	Precision	Recall	Bias	Variance
Random Forest classifier	0.8615	0.8615	0.8615	0.1115	0.25

11. Gradient Boosting Classifier

Model classifier parameters:

- loss = "exponential",
- learning_rate = 0.01,
- n_estimators = 250,
- subsample = 0.85,
- criterion = "squared_error",
- max_depth = 300,
- max_features = 300,
- n_iter_no_change = 6

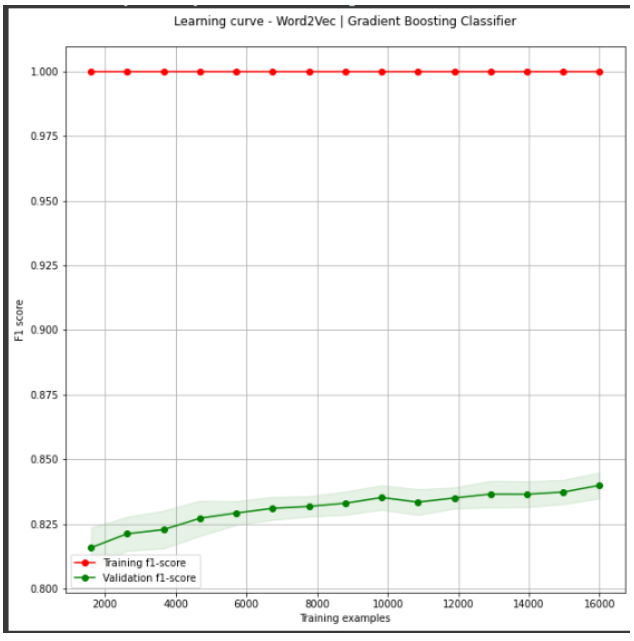

Figure 21-Learning curves Gradient Boosting Trees classification

Comments

Same poor performance with Random Forest Classifier. The Gradient Boosting classifier underfitted.

Summary table from Gradient Boosting Tree classifier

Test name	F1-score	Precision	Recall	Bias	Variance
Gradient Boosting Tree	0.8505	0.8505	0.8505	0.1005	0.25

12. Conclusions

Overall, the selected model will be the baseline logistic regression, without the application of any regularization method.

Key points to consider:

- Train-test split: 50%-50% for the best performance of baseline logistic regression.
- Cross-validation should be used during training and scoring. The learning curves were smoother when a high number of bucker-split was used (>=15).
- Word2Vec embeddings with a vector size 300.
- If to consider any regularization that would be Ridge Regression with a mild penalty coefficient of 0.8 degrees.

13. Next steps / Future suggestions

1. Dimensionality reductions on (PCA) on the word embeddings vector of Word2Vec model for the selected model classifier.
2. Apply stricter and more sophisticated regularization techniques like elasticnet.

References & Resources

1. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
2. https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py
3. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
7. https://en.wikipedia.org/wiki/Word2vec
8. https://towardsdatascience.com/5-simple-ways-to-tokenize-text-in-python-92c6804edfc4
9. https://medium.com/swlh/sentiment-classification-using-word-embeddings-word2vec-aedf28fbb8ca
10. https://www.ahmedbesbes.com/blog/sentiment-analysis-with-keras-and-word-2-vec
11. https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

12. https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c
13. https://towardsdatascience.com/the-word2vec-classifier-5656b04143da