Report for Sentimental classification

Cindy Tao

Table of contents

4.1. Bag-of-Words Design Decision Description

- How did you "clean" and "standardize" the data?

Before officially starting data preprocessing, I had a look at how the raw dataset looked like in csv files and identified which data shall be fed into training. In this case, they are column 2 of x_train.csv and column 1 of y_train.csv. Column 2 in x_train.csv is the actual review string, which contains uppercase, special characters and non-stemmed word format. These irrelevant data can increase outliners and complexities when model learning them. Reducing these data can maximize the review dataset efficiency. While lemmatization normalizes the wods, some information can be missed such as the difference given by present and past tense.

Understanding the above gave me a clear mind on what training information I'd like to get out of a particular dataset. That formed my decisions on what parts of data should be cleaned and normalized. So for clearer demonstration on assignment1, I removed punctuations and capitalizations before tokenizing, using string.punctuation() and string.lower(). Column heading was given during data loading, so that I can easily specify which column needs data cleaning and normalization.

- How did you determine the final vocabulary set?
- What was your final vocabulary size (or ballpark size(s), if size varies across folds because it depends on the training set)?

Two reasons why I use TFIDFVectorizer over TFIDFTransformer for bag-of-words feature representation(BoW representation). In the case of TFIDFTrnasformer, I need a Countvectorizer to compute word counts, on top of the TdidfTransformer that computes the inverse document frequency and term frequency / inverse document frequency(TF-IDF) score. TFIDFVectorizer did these three things in one step using fewer lines of codes. Secondly, TFIDFVectorizer is friendly to TF-IDF score generation on documents within one training dataset, whereas TFIDFTransformer fits TF-IDF for different tasks.

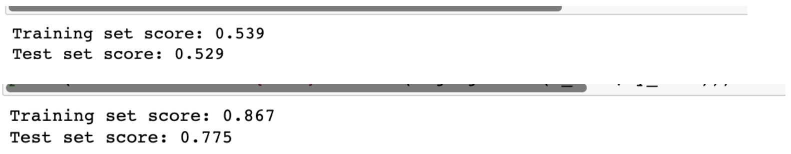
Final vocabulary size is 552 terms. These terms are turned into an array of numbers, so that the training model can accept and learn them. Vocabulary sizes in each training group varied by the number of K-folds as experimented.

- Did you exclude words, and if so how?
- Did you use unigram or bigrams?
- Did you use counts or binary values or something else?
- How did you handle out-of-vocabulary words in the test set?

Following design decisions are taken for better determining final vocabulary set in BoW pipeline:

- Excluded rare words showing in less than 10 documents by setting min_df=5 in TfidfVectorizer. Parameter max_df=0.75 kept out common words appearing in more than 50% of documents.
- Common words like stopwords are excluded through stop_words='english'.
- Setted ngram_range(1, 2) to consider both unigram and bigram as bag-of-words vocabulary. This improved training accuracy improves from 0.539 to 0.867(fig1).

Because the model treats influential information in both single term and two terms as features to classify reviews.



```
Training set score: 0.539
Test set score: 0.529
```

```
Training set score: 0.867
Test set score: 0.775
```

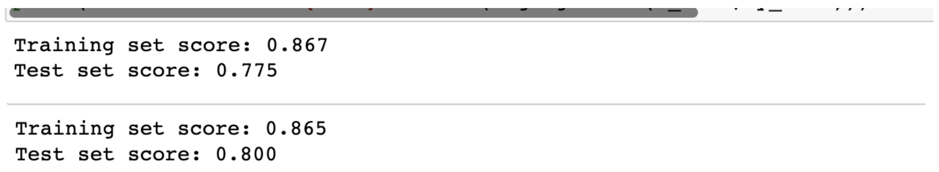
Figure 1. *Training accuracy when only considered unigrams VS after considered bigrams*

- By using TfidfVectorizer(), term count values are automatically recorded rather than purely binary values(0/1). The parameter binary=False furtherly confirmed this design decision.
- Smart term reweighting was done by picking the most influential terms in TfidfVectorizer(). This reveals the features for each class classification, and thereby avoids the training model taking the bias of the vectorized model. Parameter use_idf=True enables inverse-document-frequency reweighting.
- Smoothing technique is implemented by smooth_idf=True to handle out-of-vocabulary words in the test set. Smooth_idf=True adds a constant term everytime calculating term probability in each document, so that the probability will never be 0. This prevents the model from ignoring the unseen terms in test data.

4.2. Cross Validation and Hyperparameter Selection Design Description

- What is the source of your holdout data for performance estimates?

I splitted 80% of the review column of x_train.csv and sentiment column of y_train.csv for x and y training data. The rest of 20% are used as validation data of x and y. This reduces more data overfitting than 90%10% data splitting, according to my experimentation(fig2). Gaps between training accuracy and validation accuracy reduced to 0.6, while training accuracy remained well.



```
Training set score: 0.867
Test set score: 0.775
```

```
Training set score: 0.865
Test set score: 0.800
```

Figure 2. *Validation accuracy after 90%10% data splitting VS after 80%20% data splitting*

Such data splitting partitioned the holdout data(featured and labeled) and shuffled them into training validation samples for the cross validation usage. Here I used 10-folds cross validation(CV) grouping these training samples into 10 experiments. Each experiment has 1 validation sample. Taking advantage of its randomly organized samples, each experiment is built intentionally.

I predicted the holdout data performance in the training model, by putting x and y validation data into a LogisticRegression algorithm with 10-folds CV. 1 experiment evaluated model prediction performance, while the rest of 9 experiments learnt from samples. In this way, 10-folds gives sufficient datasets to robustly evaluate the prediction performance.

- What performance metric will your search try to optimize on holdout data?

Performance matrix mean and standard deviation explain the model accuracy with holdout data. Standard deviation dropped to 0.025, meaning the data were clustered near the mean and performed more reliably in model. Observing mean and standard deviation after hyperparameter search also helped me to determine the best approach to optimize holdout data.

- What hyperparameter search strategy will you use?

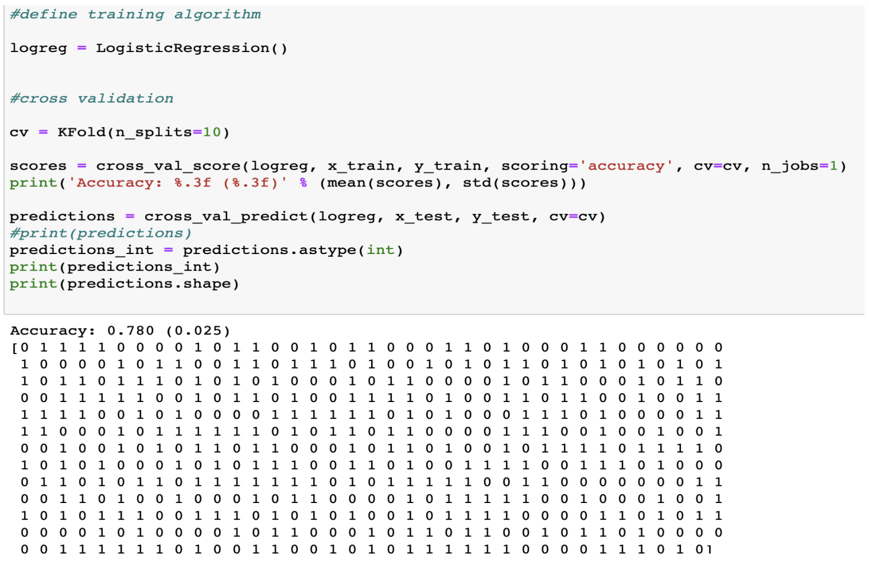
In comparison to MultinomialNB and RandomForestClassifier, LogisticRegression has better performance. Even though training accuracy can vary by its hyperparameter. That's why I used GridSearchCV to search hyperparameters for the LogisticRegression model. First, I specified the LogisticRegression algorithm and its hyperparameter options. Then I defined what the grid should search for. Lastly, I analyzed results by giving means and standard deviation to the outcome of each hyperparameter configuration. I ended it up by using the hyperparameter configurations that gave the highest mean value.

- Given a selected hyperparameter configuration created using CV by training models across several folds, how will you then build a "final" model to apply on the test set?

Given the configurations selected in the final model, I trained it by fitting x and y train data into the model and achieved 0.86 training accuracy. Given the requirement of binary classification, class_weight='balanced' implicitly duplicates smaller classes until they have the equivalent scales as the larger class.

4.3. Hyperparameter Selection Figure for Classifier

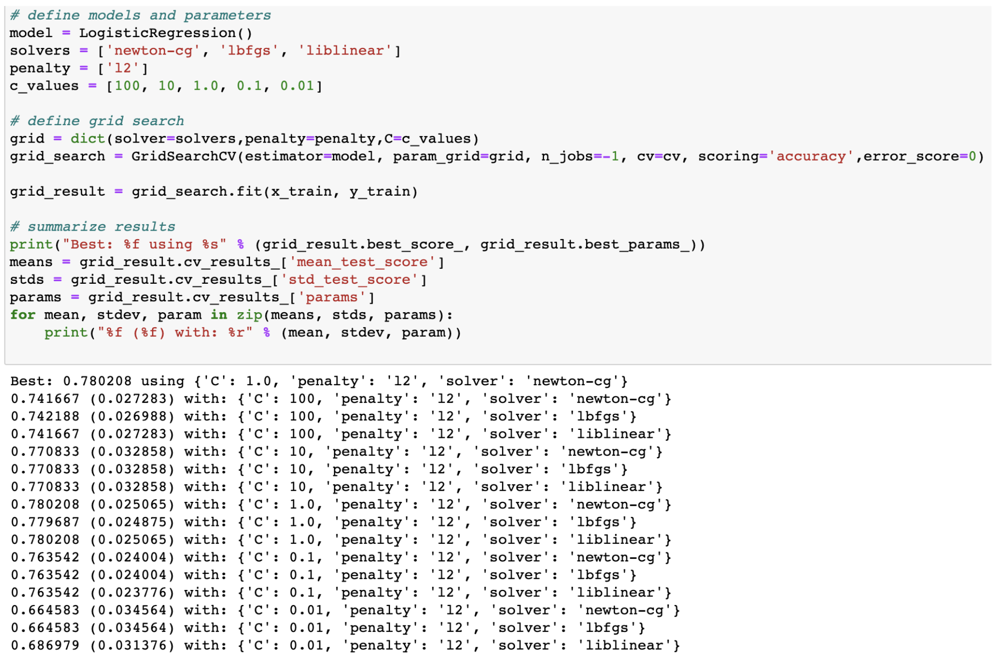
To summarize, the GridSearchCV selected the hyperparameter best fits a specified classifier. This classifier was constructed by a LogisticRegression algorithm, which was training 10-fold CV on BoW preprocessed datasets as elaborated before. Detailed implementation of CV is shown below (fig3). Parameter CV=CV in GridSearchCV() executed the search based on 10-folds specified earlier(fig4).



```
#define training algorithm

logreg = LogisticRegression()


#cross validation

cv = KFold(n_splits=10)

scores = cross_val_score(logreg, x_train, y_train, scoring='accuracy', cv=cv, n_jobs=1)
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

predictions = cross_val_predict(logreg, x_test, y_test, cv=cv)
#print(predictions)
predictions_int = predictions.astype(int)
print(predictions_int)
print(predictions.shape)
```

```
Accuracy: 0.780 (0.025)
[0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0
 1 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1
 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0
 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1
 1 1 1 1 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1
 1 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1
 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1 0
 1 0 1 0 1 0 0 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0
 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1
 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1
 1 0 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1
 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0
 0 0 1 1 1 1 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 01
```

Figure 3. *Codes to undertake 10-folds validation after data splitting & Results of cross validation*



```
# define models and parameters
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)

grid_result = grid_search.fit(x_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.780208 using {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.741667 (0.027283) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.742188 (0.026988) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.741667 (0.027283) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.770833 (0.032858) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.770833 (0.032858) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.770833 (0.032858) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.780208 (0.025065) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.779687 (0.024875) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.780208 (0.025065) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.763542 (0.024004) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.763542 (0.024004) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.763542 (0.023776) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.664583 (0.034564) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.664583 (0.034564) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.686979 (0.031376) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Figure 4. Codes of GridSearchCV() method to search for hyperparameter based on 10-folds CV

4.4 Analysis of Predictions for the Classifier

Table 1. *representative examples of FPs and FN & Mistakes model made*

	Does it do better on longer sentences or shorter sentences?	**Does it do better on a particular kind of review?**	**Does it do better on sentences without negation words?**
Confusion matrix - FP	Less FP in both longer and shorter sentences *It's pretty surprising that this wonderful film was made in 1949, as Hollywood generally had its collective heads in the sand concerning black and white issues at that time.* *imdb*	Model didn't create bias by categories. All categories have FP and FN. *It's the one movie that never ceases to interest me, simply because it keeps me alert, as I try to attempt to decipher it's meanings.* *imdb*	The model got confused when mixing two words that contain 'no' with a positive single word. *I purchased this and within 2 days it was no longer working!!!!!!!!!* *amazon*
Confusion matrix -	More FN in longer	Model didn't create	The model interprets

FN	sentences. *My boyfriend and i sat at the bar and had a completely delightful experience.* *yelp*	bias by categories. All categories have FP and FN.	well on sentences purely contain negative words "not", "did not", "should not", "didn't" and "shouldn't"
Accuracy, Precision, Recall, F1-score	Both short and long sentences have high accuracy and precision. Slightly F1 score on short sentences because of its high recall score.	Short sentences in both imdb and yelp has higher accuracy overall	The model gave higher accuracy and precision on sentences including two positive words, such as 'pretty surprise', 'pretty well'.

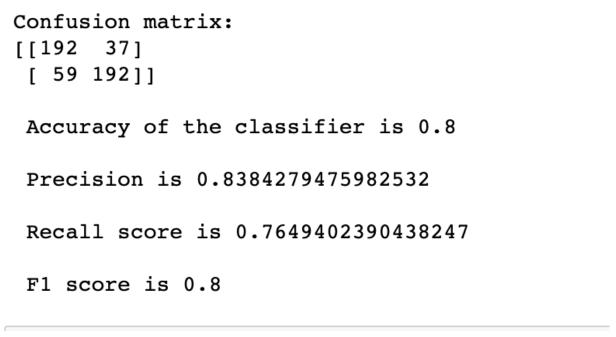
4.5. Performance on Test Set

- A summary paragraph stating your ultimate test set performance
- Compare it to your previous estimates of holdout performance from cross-validation reflect on any differences and discuss the potential reasons

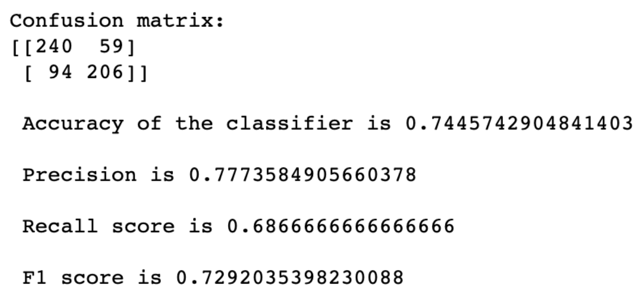
I tested the model using the review integer of x_test.csv and sentiment integer y_test.csv. Accuracy score 74% provided total correct predictions on test data. It didn't differ much from 80% accuracy for holdout performance from CV(fig5). This has benefited from the robust validation harness and appropriate model fitting through 10-folds CV and 80%20% data splitting.

Due to the possibilities of uneven datasets, I took care of other matrices for neutral evaluation. As can be told by the confusion matrix evaluating test dataset performance(fig5), the model predicted 240 true positive sentiment(1) and 206 true negative sentiment(0). 59 false positive sentiment and 94 false negative sentiment. Benefiting from the low false positive rate, precision reached 77% among all reviews that were labeled as positive, which is a good signal. For those reviews truly represents positive sentiment, the model labeled 68% of them as positive. This recall score measures model sensitivity to good review. Lower sensitivity here is caused by a small scale of overfitting. In comparison to the model performance on test data, holdout data had precision 83% and recall 76%. More BoW and smaller c value for the model could improve recall score. However, In the case of sentimental analysis, high precision values more than high recall score. Because high precision means more good reviews are detected.

The F1 score for test data is 72% versus 80% for holdout data. This means the average of recall and precision score. For this testing, the amount of false positives and false negatives still has 35 differences. Hence, individual scores for recall and precision is a stronger approval on the model performance.



```
Confusion matrix:
[[192  37]
 [ 59 192]]

 Accuracy of the classifier is 0.8

 Precision is 0.8384279475982532

 Recall score is 0.7649402390438247

 F1 score is 0.8
```



```
Confusion matrix:
[[240  59]
 [ 94 206]]

 Accuracy of the classifier is 0.7445742904841403

 Precision is 0.7773584905660378

 Recall score is 0.6866666666666666

 F1 score is 0.7292035398230088
```

Figure 5. *Confusion matrix, classification accuracy, precision, recall and F1 on validation set* VS on test set

References

"Don't Sweat the Solver Stuff", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451. [Accessed: 17- Apr- 2022].

J. Brownlee, "Tune Hyperparameters for Classification Machine Learning Algorithms", *Machine Learning Mastery*, 2022. [Online]. Available: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/. [Accessed: 17- Apr- 2022].

"Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures - Exsilio Blog", *Exsilio Blog*, 2022. [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. [Accessed: 17- Apr- 2022].