Project B Report

Problem 1

1A:

I set the lower case parameter of to True so all letters will be transformed to lower case, so we can extract words that is all lower case.

I determine the final vocabulary set by choosing the top hundreds of words ordered by term frequency, and I will decide the number in hyperparameter selection.

The final vocabulary size is 500, this is gained by grid search.

I use bigrams, because some words will be meaningful in pair, like the word ‘really’ does not indicate anything, but ‘really bad’ and ‘really good’ are very good features, so bigrams will be better.

I use TfidfVectorizer to extract text features, the vector will be Tfidf, the term-frequency times inverse document-frequency, so the words that appear too frequently will have less meaning , like if the word ‘a’ ‘the’ ‘and’.

For out-of-vocabulary word in test set, they will be ignored, because we can not decide whether they are positive words or negative words without any context.

1B:

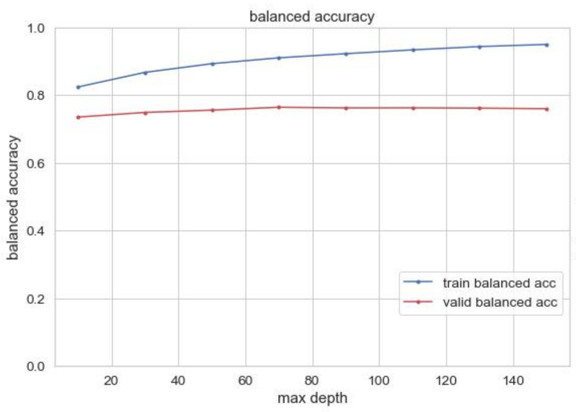
I use balanced accuracy as performance metric, it is calculated as the average of the proportion corrects of each class, it is suitable for this classification problem.

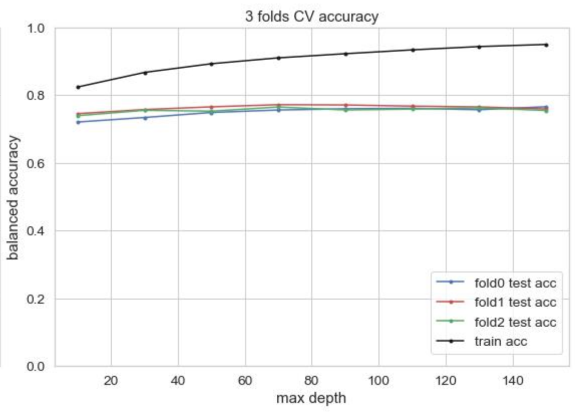
I use gridSearchCV for random forest, and grid search for logistic regression, because the random forest has more hyperparameter to configure.

I use cross validation with 3 folds, each fold size is 800, I shuffle the data and then divide it into folds with equal size in sequence.

Given a select hyperparameter configuration, I will train the model with all training data with the configuration and use this final model on test set.

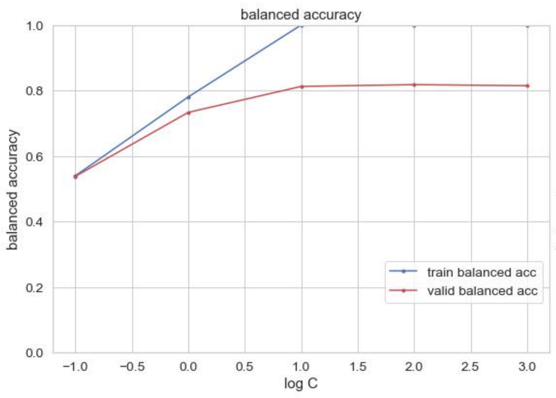
1C:

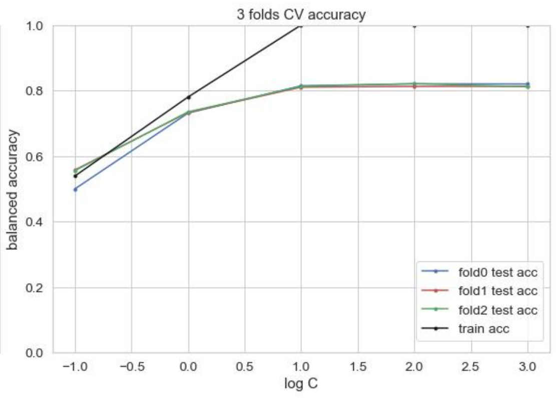




I use random forest for the this part of the question. Random forest is very suitable with bag-of-word, we have many features and it will take too long to train each tree on all the features, so we use random forest which provide faster training and also have good performance. In training progress, I check on max depth for under and over fitting, and I also tune the max feature number see which number is the most suitable. For random forest, I explore max depth and max feature, they control the tree depth and random feature number of each split respectively, the grids are reasonable because I run several times to make sure the model have underfitting and overfitting spot. The hyperparameter max_features= 10 and max_depth = 70 is preferred. The max_features= 10 is certain because in several runs, it is consistently 10, and max_depth may vary between 70 and 110.

1D:





I use logistic regression for the this part of the question. Because we are making a binary classification, so logistic regression can perform well on the task and it is easy and fast to train, with logistic regression we are assigning weights to our vocabulary, so positive words are likely to have positive weight and same for negative words. In training progress, we need to check for under and over fitting, and we also need to tune the iteration number to make sure the model get to a ideal accuracy. For logistic regression, I explore max iteration and C value, they control the iteration times and regularization strength respectively, the grids are reasonable because I run several times to make sure it range from under fitting spot to overfitting spot. The hyperparameter max_iteration = 10 and C = 100 is preferred. The evidence is decisive because the gird search a larger range and this pair of parameter give out the best performance, and the result is same for several run of search.

1E:

For 1C random forest, the best heldout accuracy is 0.778, for 1D logistic regression the best heldout accuracy is 0.817. The logistic regression performs better, it is better suited to features and avoid overfitting with l1 penalty, and it is also fast to train. The logistic regression is stable than the random forest, the random forest is not very stable and the accuracy varies with several runs.

1F:

False negative

```
amazon ['It does everything the description said it would.']
amazon ["Better than you'd expect."]
amazon ['It has kept up very well.']
amazon ['It worked very well.']
amazon ['This case seems well made.']
amazon ['Just what I wanted.']
amazon ["I'm still infatuated with this phone."]
amazon ['Would recommend this item.']
amazon ['Also makes it easier to hold on to.']
imdb ['GO AND SEE IT!  ']
imdb ['Predictable, but not a bad watch.  ']
imdb ['Kieslowski never ceases to amaze me.  ']
imdb ['I advise you to look out for it.  ']
imdb ['Go rent it.  ']
imdb ['Give this one a look.  ']
imdb ["I don't think you will be disappointed.  "]
yelp ['This place has it!']
yelp ['Will be back again!']
yelp ['After one bite, I was hooked.']
yelp ['Eclectic selection.']
yelp ['Food was so gooodd.']
yelp ['I could eat their bruschetta all day it is devine.']
```

False positive

```
amazon ['I advise EVERYONE DO NOT BE FOOLED!']
amazon ['The loudspeaker option is great, the bumpers with the lights is very ... appealing.']
amazon ['Not good enough for the price.']
amazon ['I was not happy with this item.']
imdb ['Graphics is far from the best part of the game.  ']
imdb ['It is not good.  ']
imdb ['There was NOTHING believable about it at all.  ']
imdb ['Nothing new there.  ']
imdb ['All in all, a great disappointment.  ']
imdb ['Why was this film made?  ']
imdb ['Nothing at all to recommend.  ']
imdb ['Not easy to watch.  ']
imdb ["But it's just not funny.  "]
yelp ['Not good for the money.']
yelp ['I give it 2 thumbs down']
yelp ['The selection of food was not the best.']
yelp ['It was not good.']
yelp ['not even a "hello, we will be right with you."']
yelp ['Would not recommend to others.']
```

First look at false negative example, ‘I dont think you will be disappointed’ is a double negation sentence so it actually mean positive, but with my vectorization it is hard to recognize double negation sentence. In ‘Food was so gooodd’ the good is not in a formal style so it is not recognized. Some other sentences may not have very strong or obvious words that can express positive feeling.

And it seem to do worse on short sentence as we can observe that most of the wrong cases are short sentences, and it do better on sentence without negation words like ‘not’, ‘wouldn’t’, as we can see some wrong cases contains negation words.

So to improve performance, I should try to make it recognize negation sentence better, and also try to make it perform better on short sentence.

1G:

The ultimate accuracy is 0.83 , it is consistent with the estimate of previous heldout accuracy 0.817.

Problem 2

2A:

I convert all sentence to lower cases, and I split some words like ‘wasn’ to ‘was not’, ‘couldn’ to ‘could not’. So they can be recognized by the word2vec converter.

I use max_features parameter of vectorizor to control the vocabulary size, I exclude stop words ['and', 'for', 'in', 'is', 'it', 'of', 'the', 'this', 'to', 'was'], I choose these stops words by frequency and I exclude ‘not’ from them

The size of final vocabulary is 500.

Each word is passed to the word2vec converter to get a vector.

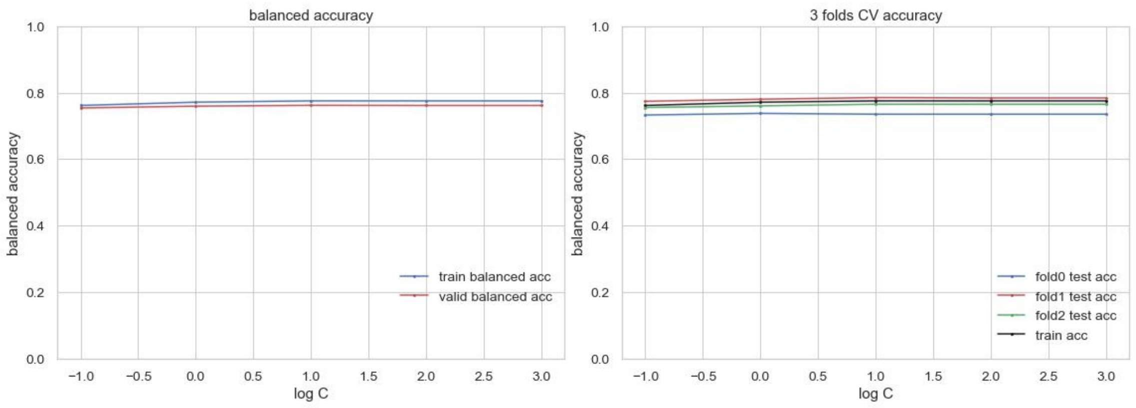
I sum all the embedded vector to produce a vector representing a sentence, each sentence’s feature vector is of size 50, as the size of a vector. Because I add up the vector.

For out of vocabulary words, they are ignored since we can not find a embedded vector for them.

2B:

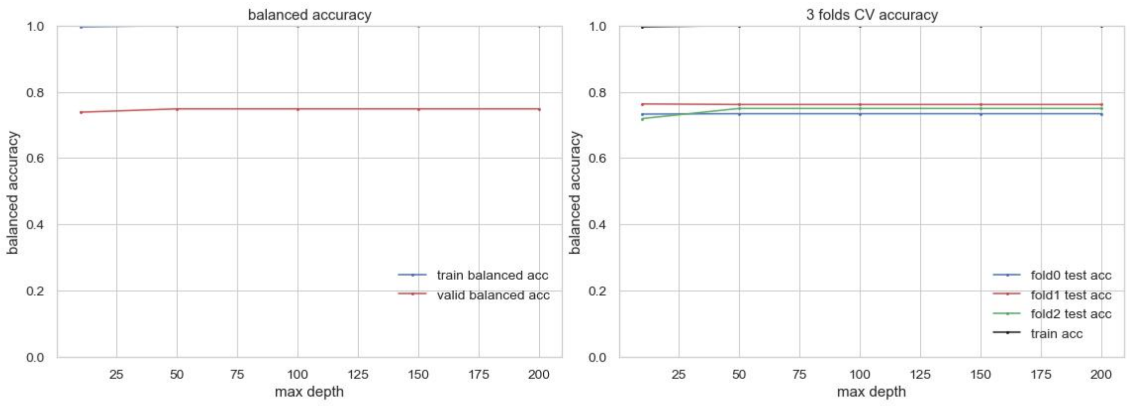
Nothing changed, it is same as 1B.

2C:



I use logistic regression for the this part of the question. Because we are making a binary classification, so logistic regression can perform well on the task and it is easy and fast to train, with logistic regression we are assigning weights to our vocabulary, so positive words are likely to have positive weight and same for negative words. In training progress, we need to check for under and over fitting, and we also need to tune the iteration number to make sure the model get to a ideal accuracy. For logistic regression, I explore max iteration and C value, they control the iteration times and regularization strength respectively, the grids are reasonable because I run several times to make sure it range from under fitting spot to overfitting spot. The hyperparameter max_iteration = 100 and C = 10 is preferred. The evidence is decisive because the overall performance is very consistent.

2D:



The training curve is close to 1 all the time.

I use random forest for the this part of the question. The input training is vector and we are making binary classification so random forest can be applied to make fast and stable training. The result is very consistent and stable. In training progress, it seemed to be overfitting all the time, and I tune hyperparameters several times, it is still the same. For random forest, I explore max depth and max feature, they control the tree depth and random feature number of each split respectively. The hyperparameter max_features= 10 and max_depth = 50 is preferred. The evidence is not quite decisive because all hyperparameters are very close.

2E:

For 2D random forest, the best heldout accuracy is 0.717, for 2C logistic regression the best heldout accuracy is 0.768. The logistic regression performs better, it is better suited to features and avoid overfitting with l1 penalty, and it is also fast to train.

Both model give consistent performance. I think this is because we are using word embedding and the vectors that it produces are very deterministic, so the performance will be very consistent and stable, so the model does not vary a lot, it is the transformation from words to vector that matters.

2F:

False negative	amazon ['Five star Plus, plus.']
	amazon ['WORTHWHILE.']
	amazon ['Magical Help.']
	amazon ['Now I know that I made a wise decision.']
	amazon ['I did not have any problem with this item and would ord
	er it again if needed.']
	imdb ['Predictable, but not a bad watch. ']
	imdb ['I struggle to find anything bad to say about it. ']
	imdb ['How can anyone in their right mind ask for anything more
	from a movie than this? ']
	imdb ['Great movie! ']
	imdb ['The last 15 minutes of movie are also not bad as well.
	']
	imdb ['Waste your money on this game. ']
	imdb ['Not much dialogue, not much music, the whole film was sho
	t as elaborately and aesthetically like a sculpture. ']
	imdb ['You wont regret it! ']
	imdb ['Anne Heche was utterly convincing. ']
	yelp ['No complaints!']
	yelp ['It was just not a fun experience.']
	yelp ['I was seated immediately.']
False positive	amazon ["Unreliable - I'm giving up."]
	amazon ["Couldn't figure it out"]
	amazon ['Phone falls out easily.']
	amazon ["I wouldn't recommend buying this product."]
	imdb ['Highly unrecommended. ']
	imdb ['The basic premise is wasted since it is sidelined by the
	inexplicable focus on the documentary crew. ']
	imdb ['The death row scenes were entirely unmoving. ']
	imdb ['And the accents are absolutely abysmal! ']
	imdb ['It was so BORING! ']
	imdb ['This scene is very strong and unpleasant. ']
	imdb ['The results, well, are a shame. ']
	yelp ['The kids play area is NASTY!']
	yelp ['It was attached to a gas station, and that is rarely a go
	od sign.']
	yelp ["I'm super pissd."]

It do better on longer sentences as we can see there are many shorter sentences that are mis-classified. And sentences containing negation are also likely to be mis-classified like the second and 4th ones of false positive cases, there are words like ‘couldn’t’ and ‘wouldn’t’.

It perform better on yelp because there are fewer wrong cases from yelp.

To improve performance, I should try to make it recognize negation and short sentences better.

2G:

The ultimate accuracy is 0.783 , it is consistent with the estimate of previous heldout accuracy 0.768.

Problem 3

3A:

To obtain features, I use TfidfVectorizer to get more balanced weight, I convert all sentence to lower cases and I set the stop words to remove common words that is meaning less, then I set the max_features to 500, and this time I only use unigram to extract features, because MLP will establish relationship between unigrams, so simple unigram will be faster for training.

I use MLP as classifier, it can establish relationship between words to make better prediction, and I train it with the extracted text feature and label with cross validation.

Hyperparameter are layer size, batch size and learning rate, they control complexity, learning speed. I use gridSearcher to do the search. With layer size [25,50,100], and batch size[20,50,100], learning rate[0.3,0.5,1.0].

3B:

The ultimate accuracy is 0.823, it is better than previous problems, because I improve the features extraction process on the basis of previous problems, and that is the most important factor, more accurate text features have better upper bound of performance since it can better represent sentences.

And the MLP model perform well on the unigram, it can make connection between words to give better prediction.