CS135 Intro to ML

Project 2 Pengcheng Xu

Part One: Classifying Review Sentiment with Bag-of-Words Features

1 (10 pts.) In your report, include a paragraph or two that explain the "pipeline" for generating your BoW features. This should include a clear description of any pre-processing you did on the basic text, along with the sorts of decisions you made in generating your final feature- vectors. You should present this in complete enough form that someone else (another student say) could produce a model identical to yours if they wished, based upon reading your report. As we have said before, keep code samples to a minimum; ideally, you should be able to explain what you did in plain language. Your paragraph should also contain some justification for why you made the decisions you did.

>> First, we take a glimpse of the data, and we find that the raw data having **4510** (as shown below) tokens without any pre-processing.

```
Number of features:  4510
['00' '10' '100' ... 'zero' 'zillion' 'zombie']
```

	00	10	100	11	12	13	15	15g	15pm	17	...	youtube	yucky	yukon	yum	yummy	yun	z500a	zero	zillion	zombie
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2395	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2396	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2397	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2398	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2399	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2400 rows × 4510 columns

Then we did the following pre-processing pocedures:

- **Convert words' common contraction form to their complete form**
  e.g. since "It's" and "It is" are actually the same semantically, this will help us reduce the total number of tokens in the following steps

- **Filtering out some common stop words using customized dict**
  Stop words are frequently occurred words but have little meaning, like "a", "the", "this", "that" … , so we need to filter them out. However, commonly used stop word lists (e.g. sklearn built-in "english ", or "english" from nltk) often include "no", "not" in the list, which are very important in our context (e.g. "This is good" has totally opposite meaning against "This is not good"). So we need to create our own customized stop word list (i.e. keep those negation words).
  And also, we rule out many adverbs, since in our context, adjectives are way more important, adverbs usually don't change the sentiment of the sentence (e.g. "This is good" and "This is very good" both convey positive sentiments)
- **Covert words to their stem form**
  Many words, although in different forms, are actually origin from the same stem, so we could treat them as a same token (e.g. "loves", "loved" will only merge to a single token "love"). This will help us greatly reduce the total number of tokens.
- **Term frequency (TF) and inverse document frequency**
  We also need to normalized our tokens and taken into consideration that the mount of information a token provides about the documents overall. Thus, we include sklearn "TfidfVectorizer" to help us do this featuring enginerring.

The following is a screen shot of the customized stop word list we're using:

```
# custome stopword list
custom_stopword_list = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
                        "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
                        'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their
                        'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'th
                        'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',
                        'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
                        'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'b
                        'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under
                        'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both'
                        'few', 'more', 'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
                        'too', 'very', 's', 't', 'can', 'will', 'just', 'should', "should've", 'now', 'd', 'll',
                        'm', 'o', 're', 've', 'y', "would", "could", "might", "shall", 'ma', "must", "some",
"somebody","somehow","someone","somethan","something","sometime","sometimes","somewhat","somewhere", "near","nearly","n
"extremely", "incredibly", "remarkably", "very", "totally", "completely", "utterly", "absolutely", "entirely"
                       ]
```

After these pre-processing steps, we reduced dimension of features to **3391**.

```
Number of features:  3391
['abandon' 'abhor' 'abil' ... 'zero' 'zillion' 'zombi']
```

2. Generate a logistic regression model for your feature-data and use it to classify the training data. In your report:

- **Give a few sentences describing the model you built, and any decision made about how you set is parameters, trained it, etc.**

>> The model we chose here is the "LogsiticRegression" model from "sklearn". Hyperparameters, we selected are penalty params (i.e. "l1", "l2", where "l1" could drive some weights to 0, while "l2" doesn't), and C (i.e. the inverse of regularization strength. The smaller value, the strong we put on the regularization).

- **Choose at least two hyperparameters that control model complexity and/or its tendency to overfit. Vary those hyperparameters in a systematic way, testing it using a cross-validation methodology (you can use libraries that search through and cross-validate different hyperparameters here if you like). Explain the hyperparameters you chose, the range of values you explored (and why), and describe the cross-validation testing in a clear enough manner that the reader could reproduce its basic form, if desired.**

>> The two Hyperparameters we chose are penalty params (i.e. 'l1' or 'l2') and C (i.e. the inverse of regularization strength). We want to see how quickly (i.e. determined by C) and whether forcing some features to zero (e.g. 'l1' will do this) will go us a better result.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import seaborn as sns

logReg = LogisticRegression(solver='saga')

penalty = ['l1', 'l2']
c = np.logspace(-4, 4, 20)
hyperParams = {
    "C": c,
    "penalty": penalty
}
clf = GridSearchCV(logReg, hyperParams, cv = 5, n_jobs= -1, return_train_score = True)
```

- **Produce at least one figure that shows, for at least two tested hyperparameters, performance for at least 5 distinct values—this performance should be plotted in terms of average error for both training and validation data across the multiple folds, for each of the values of the hyperparameter. Include information, either in the figure, or along with it in the report, on the uncertainty in these results.**

  >> Here, we show our best 5 hyper-params classifiers as our first graph below. The second graph shows we have more detailed info (in terms of each fold), but because of the width of the report, we only show the most important summary as in the first graph.

  (Top 5 classifiers)

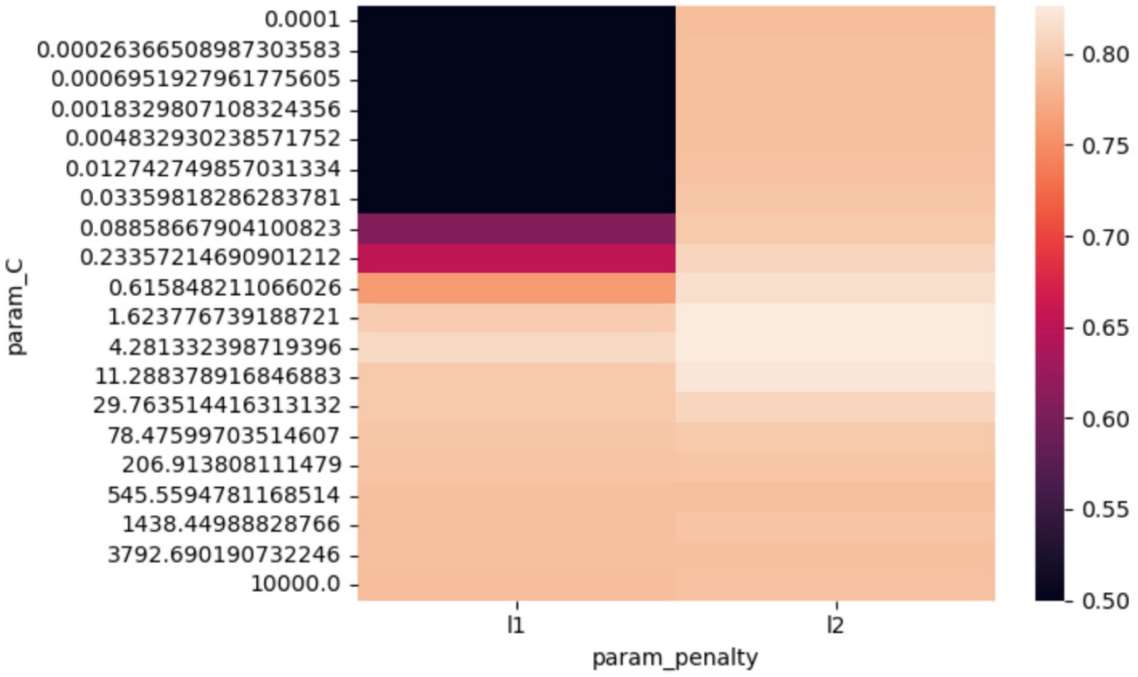
rank_test_score	params	mean_test_score	std_test_score	mean_train_score	std_train_score
1	{'C': 1.623776739188721, 'penalty': 'l2'}	0.826250	0.019030	0.958437	0.001524
1	{'C': 4.281332398719396, 'penalty': 'l2'}	0.826250	0.021024	0.973750	0.001259
3	{'C': 11.288378916846883, 'penalty': 'l2'}	0.821667	0.020353	0.988438	0.001206
4	{'C': 0.615848211066026, 'penalty': 'l2'}	0.817083	0.025766	0.938125	0.002517
5	{'C': 4.281332398719396, 'penalty': 'l1'}	0.810417	0.030533	0.971250	0.001816

  (full classifiers table)

params	split0_test_score	split1_test_score	split2_test_score	...	mean_test_score	std_test_score	rank_test_score	split0_train_score	split1_train_score	split2_
{'C': 0.0001, 'penalty': 'l1'}	0.500000	0.500000	0.500000	...	0.500000	0.000000	34	0.500000	0.500000	
{'C': 0.0001, 'penalty': 'l2'}	0.795833	0.762500	0.752083	...	0.784167	0.027087	30	0.867708	0.876042	
{'C': 0.00026366508987303583, 'penalty': 'l1'}	0.500000	0.500000	0.500000	...	0.500000	0.000000	34	0.500000	0.500000	
{'C': 0.00026366508987303583, 'penalty': 'l2'}	0.800000	0.762500	0.762500	...	0.787917	0.026628	27	0.866667	0.879167	
{'C': 0.0006951927961775605, 'penalty': 'l1'}	0.500000	0.500000	0.500000	...	0.500000	0.000000	34	0.500000	0.500000	
{'C': 0.0006951927961775605, 'penalty': 'l2'}	0.800000	0.768750	0.762500	...	0.789583	0.026253	25	0.867188	0.881771	

- **Give a few sentences analyzing these results. Are there hyperparameter settings for which the classifier clearly does better (or worse)? Is there evidence of over-fitting at some settings?**

>> The visualized result could be seen as the heat map below. From the table above, we could see that the best classifier doesn't performed best on our training data (i.e. the mean train score is only 0.958), while the 3rd and the 5th classifiers performed better on training data (i.e. mean train scores are 0.988 and 0.971, respectively). Which somehow shows these two are maybe a little bit overfit.



3 (15 pts.) Generate a neural network (or MLP) model for you feature-data. Produce the same sort of description and analysis for it as you did for the previous model, including variation of two or more hyperparameters, cross-validation testing, and at least one figure for each hyperparameter (minimum two) that shows how performance on training and validation data is affected as the hyperparameters change.

>> In this section, we choose neural network as our model.

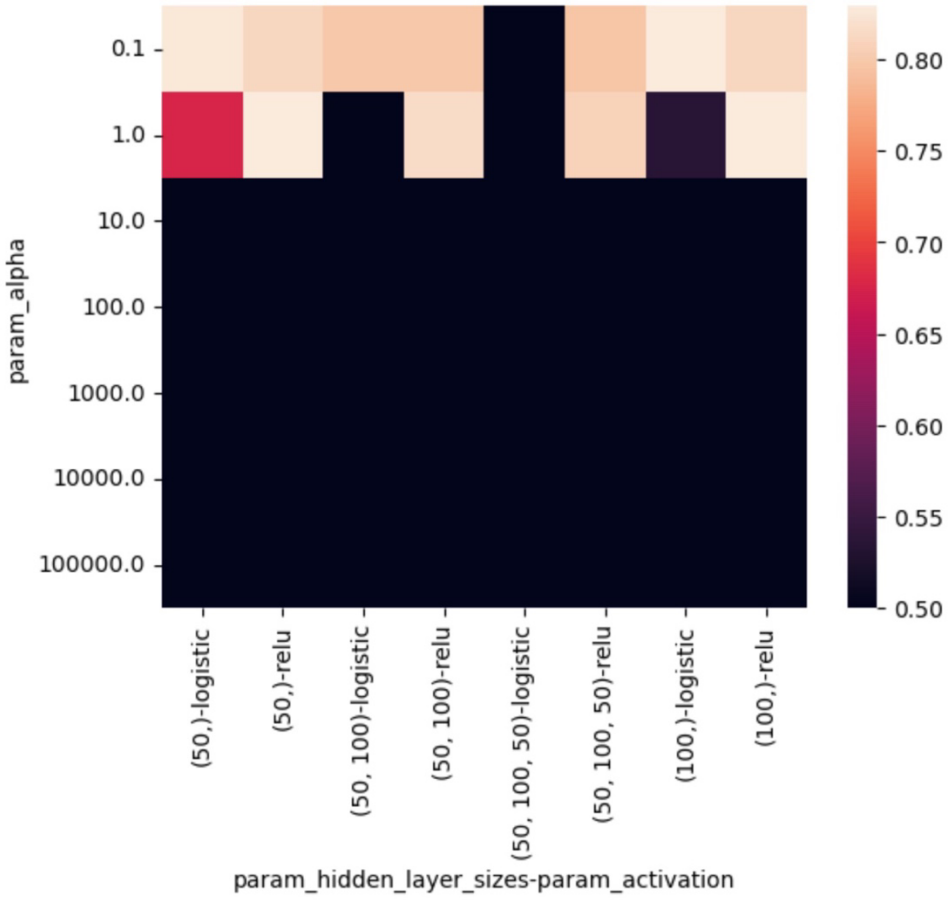
For hyper parameters, we select three parameters: “activation function”, “hidden_layer_sizes”, and “alpha” (i.e. strength of ‘l2’ regularization term). We choose ‘relu’ and ‘logistic’ as our activation candidates since these two are commonly used and they represent linear and sigmoid categories, respectively. I also searched on the Internet, many resource say it’s commonly used 1 to 3 hidden layers, with 50 to 100 nodes each layer in practice. Finally, we want to also see which value could perform best to prevent overfitting (i.e. alpha).

```
try_params = {
    'activation': ['logistic', 'relu'],
    'hidden_layer_sizes': [(50,), (100,), (50, 100), (50, 100, 50)],
    'alpha': 10.0 ** -np.arange(-5, 2)
}
mlp = MLPClassifier(max_iter=400)
```

Here, we also show the top 5 hyper-parameter stats, but the story is a bit different here. As we could see from the table below, neural network tend to overfit the training data than logistic model. Since all of the combs have the accurate score above 0.970 for training data, which is pretty amazing. But for testing set, all top 5 performed pretty much the same as logistic model.

rank_test_score	params	mean_test_score	std_test_score	mean_train_score	std_train_score
1	{'activation': 'relu', 'alpha': 1.0, 'hidden_l...	0.829167	0.020999	0.970729	0.001451
2	{'activation': 'relu', 'alpha': 1.0, 'hidden_l...	0.828750	0.020514	0.970104	0.001600
3	{'activation': 'logistic', 'alpha': 0.1, 'hidd...	0.827917	0.019167	0.978333	0.000966
4	{'activation': 'logistic', 'alpha': 0.1, 'hidd...	0.826250	0.020983	0.977604	0.001398
5	{'activation': 'relu', 'alpha': 1.0, 'hidden_l...	0.815833	0.011071	0.996979	0.000510

Again, from the heatmap, we could clearly see that the top most give us the best classifier hyperparameters (i.e. using 'logistic' activation function, only 1 hidden layer with 50 nodes and alpha around 0.1).



4 (15 pts.) Generate a third model, of whatever type you choose; you could use, for in- stance, SVM classifiers, or try ones that we have not yet explored directly (sklearn has its own decision-tree and decision-forest classifiers, for example). Whatever you choose, pro- duce the same analysis as for the prior models, including a description of what you did, how hyperparameter variation affected results, and so forth. Figures are expected showing training/validation performance relative to hyperparameter variation; additional figures are allowed, of course.

>> In this section, we choose K Nearest Neighbor Model, since usually same sentiment reviews would share very similar patterns (e.g. positive reviews often have optimistic words like “good”, “amazing”, etc), we want to explore whether this will give us a better performance.

Hyper-parameters we selected are “n_neighbors” (i.e. neither too few nor too many neighbors will give us good predictions), and “weights” (i.e. we want to explore whether if closer a neighbor is to our target node, the more influence it will impose on).

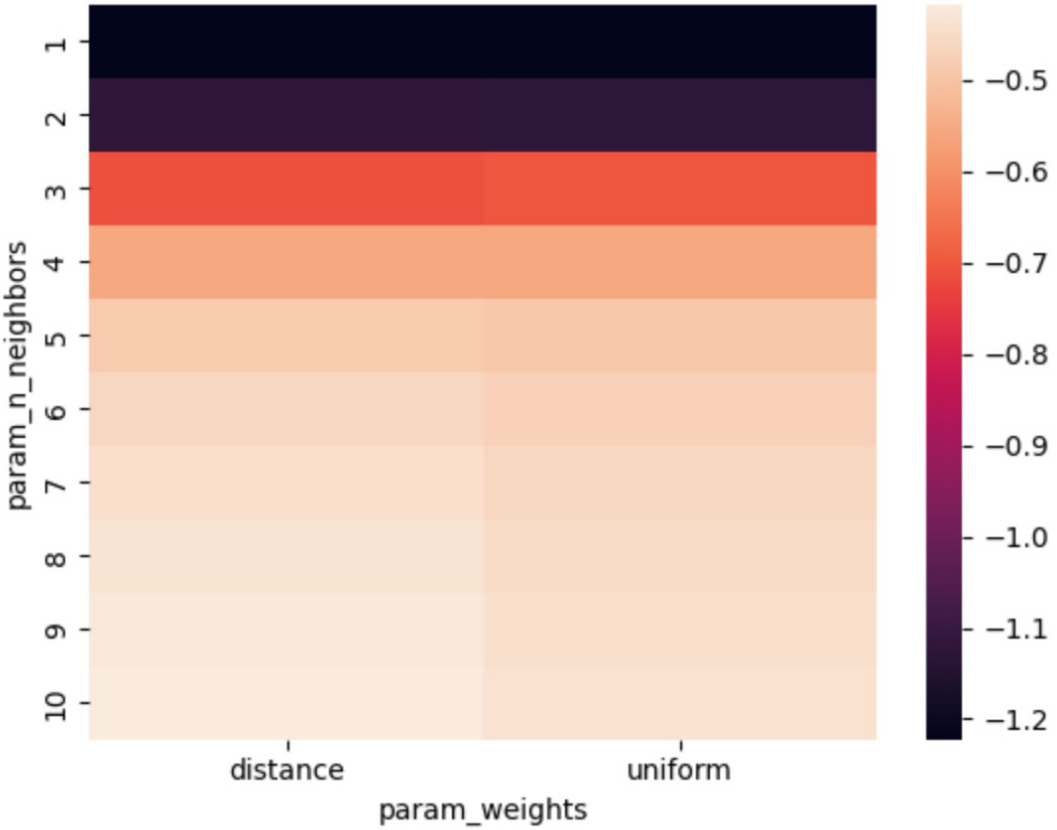
```
knn_params = {
    'n_neighbors' : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'weights' : ['uniform', 'distance']
    }
```

```
knn_clf = KNeighborsRegressor()
knn_search = GridSearchCV(knn_clf, knn_params, cv = 5, n_jobs=-1, return_train_score = True )
```

As before, we got top 5 hyper params combinations. But the story is quite weird, though. It’s obvious that for KNN (K nearest neighbor) mode, it tends to overfit the data since all top 3 combs give a 1.00 mean training score, while all combs give negative testing scores, which usually indicate overfitings.

rank_test_score	params	mean_test_score	std_test_score	mean_train_score	std_train_score
1	{'n_neighbors': 10, 'weights': 'distance'}	-0.417689	0.817595	1.000000	1.986027e-16
2	{'n_neighbors': 9, 'weights': 'distance'}	-0.422333	0.841401	1.000000	1.986027e-16
3	{'n_neighbors': 8, 'weights': 'distance'}	-0.433879	0.861479	1.000000	1.719950e-16
4	{'n_neighbors': 10, 'weights': 'uniform'}	-0.437100	0.797391	0.461162	3.738608e-02
5	{'n_neighbors': 9, 'weights': 'uniform'}	-0.440576	0.819505	0.465318	4.414504e-02

From the heatmap, we could see that the distance weight (i.e. the closer the , more influence) and 10 nearest neighbors give us the best classifier. But this is fishy, since many resources show that 4 to 5 neighbors usually perform better in practice.



5 (10 pts.) Summarize which classifier of the three you built performs best overall on your labeled data, and give some reasons why this may be so. Does it have more flexibility? Is it better at avoiding overfitting on this data? In addition, look at the performance of your best classifier and try to characterize the mistakes that it makes. Are there common features to the sentences that it gets wrong (e.g., are they mostly from one of the three source websites)? Are there other features that you can identify? Can you hypothesize why you see the results you do?

>> Among all three models, I would say the neural network model performs the best overall on the training set, since first it gives the highest avg mean train score, and avg std is also very low (i.e. which indicates the result is stable and reliable). And actually, it also gives me the best result on the leaderboard.

I think this model is more flexible, since for logistic regression, it only performs well if the underling function is in sigmoid shape, while Neural Network doesn't have this limit, the underlying function could be any shape, as long as we have enough hidden layers and sufficient time, we could train the model pretty well.

A common mistake I think is that since our training set has more features than testing set after (i.e. tokenizing, counting, normalizing, etc.). The trained model actually has many features that would be useful only in predicting training set (i.e. since testing set doesn't have those features/tokens), which result in those features' weights become useless in testing set.

From the result, we could see that natural language processing is not a easy thing. Since our model could achieve accurate rate around 0.97 on training set (e.g. Neural Network) will only get 0.82 or so on testing set.

6 (5 pts.) Apply your best classifier from the previous steps to the text data in x_test.csv file, storing the outcomes as a probabilistic prediction and then submitting them to the leaderboard, as described below. In your report, describe the performance that you see there. How does that match up to the performance you saw during training and cross-validation? If it is as expected, what does that tell us, do you think? If it is not as expected, what does that tell us?

>>

Among three models, Logistic model and Neural Network model perform at very similar level, although Neural Network did a slightly better job. The K Nearest Neighbor performs the worst.

This result is pretty much as expected. Since in our training and cross-validation steps, NN (i.e. Neural Network) performs at the similar level as Logistic, but more stable and reliable (i.e. std is less). While KNN (i.e. K Nearest Neighbor) tend to overfit the data (i.e. it has training score 1.00 while testing score negative).

This tell us prevent overfitting and get a stable model is very important.

(Neural Network leaderboard)

3	ML4Fun_nn	0.16333	0.90632

(K Nearest Neighbor Leaderboard)

25	ML4Fun_kn	0.25167	0.86932