Github: https://github.com/MSIA/qyt9304_msia_text_analytics_2021/tree/homework3

Dataset facts

I'm using the first 500K entries of the Yelp review dataset for this HW. The dataset consists of reviews given by Yelp users as well as their star ratings (integers from 1 to 5).

For text classification, I will use the ratings as the resopnse variable.

Below are the dataset stats:

	value
number_of_documents	500000.000000
number_of_labels	5.000000
label_mean	3.768648
label_median	4.000000
label_min	1.000000
label_max	5.000000
label_std	1.385620
label_25_quantile	3.000000
label_75_quantile	5.000000

Approach

First, we transform 500K rows of labelled Yelp reviews into datasets with 1-gram or 2-gram features. Next, for both processed datasets, we randomly divide the 500K rows into train and test sets (80% for training, 20% for testing). For each model (i.e. logistic regression and SVM) and each dataset, we perform hyperparameter tuning via grid search and 3-fold CV on the training set. Then, we fit the best model on the hold-out test set for model/CBOW comparisons. Finally, we use the best model-CBOW-hyperparameter combination to predict on data that the model has not seen before.

The metric used for model evaluation and model selection is micro-averaged f1 score.

Disclaimer

Note that the entire pipeline is vetted for reproducible results (random seeds are set for every step in the modeling process).

Instead of sklearn's GridSearchCV method, I used the RandomSearchCV method, based on prior experience, yields better results. In order not to kill my computer, I've also capped max iterations for cross validation, model fitting, and random search.

To better organize my python scripts, I added 'grid_search' and 'evaluate' functions to a standalone module named 'train_test.py'.

I also pushed the 500K rows of Yelp reviews onto github.

Choosing logistic regression hyperparameters

I played with the amount of regularization applied to the model as well as well as the solver. For regularization, I tested C = 0.8, 0.9, and 1.0. For the solver, I tested newton-cg, saga, and lbfgs, which are the only solvers compatible with l2-regularization and multi-class predictions.

Choosing SVM hyperparameters

I played with the shape of the kernel as well as the amount of regularization. For regularization, I tested C = 0.8, 0.9, and 1.0. For the kernel, I tested polynomial, rgb, and sigmoid shapes. Based on prior experience, kernels with more complex shapes and stronger regularization tend to yield better performances.

Grid search and test set results

In the table printout below, each row represents a different set of hyperparameter combinations.

Result of hyperparameter tuning for logistic regression with 1-gram features:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_solver \
0	33.767880	2.279096	0.183503	0.004163	newton-cg
1	36.636091	1.964463	0.181716	0.003534	newton-cg
2	23.200813	0.563632	0.179724	0.002164	saga
3	6.557003	0.173391	0.180550	0.002028	lbfgs
4	23.335785	0.594142	0.180612	0.003373	saga

	param_C	params	split0_test_f1_micro \
0	1.0	{'solver': 'newton-cg', 'C': 1.0}	0.527255
1	0.8	{'solver': 'newton-cg', 'C': 0.8}	0.527240
2	0.9	{'solver': 'saga', 'C': 0.9}	0.527240
3	0.8	{'solver': 'lbfgs', 'C': 0.8}	0.527315
4	1.0	{'solver': 'saga', 'C': 1.0}	0.527247

	split1_test_f1_micro	split2_test_f1_micro	mean_test_f1_micro \
0	0.525549	0.528376	0.527060
1	0.525571	0.528376	0.527062
2	0.525549	0.528376	0.527055
3	0.525811	0.528669	0.527265
4	0.525549	0.528376	0.527057

	std_test_f1_micro	rank_test_f1_micro	n_gram
0	0.001163	3	1
1	0.001152	2	1
2	0.001162	5	1
3	0.001167	1	1
4	0.001162	4	1

f1 result on test set: 0.53009

Result of hyperparameter tuning for logistic regression with 2-gram features:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_solver
0	15.399896	0.480275	0.177984	0.003492	newton-cg
1	16.257997	0.272905	0.177221	0.002420	newton-cg
2	6.891226	0.329609	0.177662	0.003287	saga
3	6.931776	0.143437	0.175347	0.002784	lbfgs
4	6.873224	0.372286	0.175553	0.002971	saga

	param_C	params	split0_test_f1_micro
0	1.0	{'solver': 'newton-cg', 'C': 1.0}	0.470345
1	0.8	{'solver': 'newton-cg', 'C': 0.8}	0.470353
2	0.9	{'solver': 'saga', 'C': 0.9}	0.470345
3	0.8	{'solver': 'lbfgs', 'C': 0.8}	0.470338
4	1.0	{'solver': 'saga', 'C': 1.0}	0.470345

	split1_test_f1_micro	split2_test_f1_micro	mean_test_f1_micro
0	0.469929	0.469891	0.470055
1	0.469929	0.469884	0.470055
2	0.469936	0.469891	0.470057
3	0.469906	0.469891	0.470045
4	0.469951	0.469884	0.470060

	std_test_f1_micro	rank_test_f1_micro	n_gram
0	0.000206	3	2
1	0.000211	4	2
2	0.000204	2	2
3	0.000207	5	2
4	0.000204	1	2

f1 result on test set: 0.47171

Result of hyperparameter tuning for SVM with 1-gram features:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_kernel
0	22.698955	0.085245	18.551620	0.498985	rbf
1	22.825945	0.080709	19.149259	0.789464	rbf
2	24.413917	0.223839	9.230359	0.120152	sigmoid
3	21.413352	0.310265	7.817502	0.405617	poly
4	24.161114	0.137829	9.211541	0.079287	sigmoid

	param_C	params	split0_test_f1_micro
0	1.0	{'kernel': 'rbf', 'C': 1.0}	0.318036
1	0.8	{'kernel': 'rbf', 'C': 0.8}	0.317766
2	0.9	{'kernel': 'sigmoid', 'C': 0.9}	0.374788
3	0.8	{'kernel': 'poly', 'C': 0.8}	0.422930
4	1.0	{'kernel': 'sigmoid', 'C': 1.0}	0.374788

	split1_test_f1_micro	split2_test_f1_micro	mean_test_f1_micro
0	0.309496	0.317873	0.315135
1	0.309496	0.317971	0.315077
2	0.376283	0.372976	0.374682
3	0.423076	0.422679	0.422895
4	0.376283	0.372976	0.374682

	std_test_f1_micro	rank_test_f1_micro	n_gram
0	0.003988	4	1
1	0.003948	5	1
2	0.001352	2	1
3	0.000164	1	1
4	0.001352	2	1

f1 result on test set: 0.42202

Result of hyperparameter tuning for SVM with 2-gram features:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_kernel \
0	22.931757	0.113773	20.884288	2.313909	rbf
1	23.205891	0.178477	20.912485	2.343333	rbf
2	24.012467	0.169316	13.043323	0.573042	sigmoid
3	21.898145	0.175219	9.473777	0.198051	poly
4	23.973082	0.059758	13.012695	0.505945	sigmoid

	param_C	params	split0_test_f1_micro \
0	1.0	{'kernel': 'rbf', 'C': 1.0}	0.310513
1	0.8	{'kernel': 'rbf', 'C': 0.8}	0.310513
2	0.9	{'kernel': 'sigmoid', 'C': 0.9}	0.209856
3	0.8	{'kernel': 'poly', 'C': 0.8}	0.422608
4	1.0	{'kernel': 'sigmoid', 'C': 1.0}	0.209856

	split1_test_f1_micro	split2_test_f1_micro	mean_test_f1_micro \
0	0.329723	0.333421	0.324553
1	0.323881	0.333421	0.322605
2	0.143483	0.131580	0.161640
3	0.422379	0.422716	0.422567
4	0.143483	0.131580	0.161640

	std_test_f1_micro	rank_test_f1_micro	n_gram
0	0.010041	2	2
1	0.009395	3	2
2	0.034439	4	2
3	0.000141	1	2
4	0.034439	4	2

f1 result on test set: 0.42137

Final predictions

Based on the above iterations of hyperparameter tuning and CV evaluation, the best model is logistic regression with solver = 'lbfgs' solver C = 0.8.

We fed the following three pieces of text into the model:

```
"This is so highly rated for a reason. \
If you're looking for the best in Boulder for Italian, \
go here! The food is absolutely delicious. The smell from \
when you walk in is intoxicating. It's worth the price. \
```

The portions are not huge for the price but they're the right \
size in my opinion. The butternut squash ravioli with sage are \
a must-try. You can tell they take a ton of pride in their food. \
Its authentic, delicious, beautiful. It does not have a fine \
dining atmosphere, in fact the building has a very humble \
hole-in-the-wall feel to it, nothing like most of the other \
Italian places in Boulder. But I actually prefer that type of \
atmosphere anyway. Highly recommend!!",
"Best pizza in the neighborhood!!! \
Love the this crust, moderate amount of sauce and cheese which we like!",
"Bagels are decent enough, but the coffee is horrible! \
It's the worst my wife and I have ever tasted.\n\n\
This was the Downtown Winter Garden location today."

The predicted labels as well as the probabilities of each ratings are as follows:

{"label": 5.0,
"rating1_probability": 0.0008438856930757459,
"rating2_probability": 0.003225488860759244,
"rating3_probability": 0.03459860520743991,
"rating4_probability": 0.3094928706721507,
"rating5_probability": 0.6518391495665744}

{"label": 5.0,
"rating1_probability": 0.017885057347125574,
"rating2_probability": 0.02108608265665447,
"rating3_probability": 0.03843370869126584,
"rating4_probability": 0.1720670834850842,
"rating5_probability": 0.7505280678198699}

{"label": 5.0,
"rating1_probability": 0.14248450738406138,
"rating2_probability": 0.09519321888022958,
"rating3_probability": 0.12356681057079022,
"rating4_probability": 0.21904388414905504,
"rating5_probability": 0.4197115790158638}

Note that the third piece of text is predicted with rating 5 while the review obviously corresponds to a low rating. On the other hand, compared to the predictions for the first two pieces of text, the probabilities for ratings 4 & 5 in this case are much lower, meaning that the model is less confident about its high-rating predictions.