Name: Liruoyang Yu
Andrew ID: liruoyay
Nickname: 66666

Machine Learning for Text Mining
Homework 5

1. Statement of Assurance

I certify that all of the material that I submit is original work that was done only by me.

2. Data Preprocessing

(1) **[5 pts]** After your finishing the data preprocessing, report the top 9 frequent tokens and corresponding counts in the report.

Rank	Token	Count	Rank	Token	Count	Rank	Token	Count
No. 1	good	742854	No. 2	place	716169	No. 3	food	691603
No. 4	great	585177	No. 5	like	546183	No. 6	just	521355
No. 7	time	442484	No. 8	service	431385	No. 9	really	387356

(2) **[5 pts]** Before continuing to the next step, another interesting problem is to check the star distribution of training samples. Report the count of training samples for each star (i.e., 1 to 5).

Star	1	2	3	4	5
# of training data	128038	112547	178215	373469	463084
Percentage	10.2%	9.0%	14.2%	29.7%	36.9%

Do you find something unexpected from the distribution (e.g., whether the dataset is balanced)?
Yes, the dataset is heavily biased. 4 and 5 take up almost 67% of all ratings.

[5 bonus pts] Will this be a problem in training the model? If so, could you give some idea about how to address it and explain why your idea should work?

Yes. Because if we consider in the way that P(Y|X) ~ P(X|Y)P(Y), then in this case the priors P(Y=4) and P(Y=5) are much larger than others. So, the resulting classifier is much more likely to predict 4 and 5.

One idea to address this is throwing away some data from classes 4 and 5. It works because the priors will have less effects on the predictions.

3. Model Design

(1) **[5 pts]** Show that the gradient of regularized conditional log-likelihood function with respect to the weight vector of class c (i.e., $\frac{\partial l(W)}{\partial w_c}$) is equal to

$$\sum_{i=1}^{n}\left(y_{ic}-\frac{e^{\mathbf{w}_c^T\mathbf{x}_i}}{\sum_{c'=1}^{C}e^{\mathbf{w}_{c'}^T\mathbf{x}_i}}\right)\cdot\mathbf{x}_i-\lambda\mathbf{w_c}$$

Notice that the gradient of log-likelihood function with respect to a vector w_c is itself a vector, whose i-th element is defined as $\frac{\partial l(W)}{\partial w_{ci}}$, where w_{ci} is the i-th element of vector w_c.

$$l(W) = \sum_{i=1}^{n}\sum_{c=1}^{C}(y_{ic}w_c^T x_i - y_{ic} ln \sum_{c'=1}^{C} e^{w_{c'}^T x_i}) - \frac{\lambda}{2}\sum_{c=1}^{C}||w_c||_2^2$$

$$= \sum_{i=1}^{n}(\sum_{c=1}^{C} y_{ic}w_c^T x_i - ln \sum_{c'=1}^{C} e^{w_{c'}^T x_i}) - \frac{\lambda}{2}\sum_{c=1}^{C}||w_c||_2^2$$

$$\frac{\partial l(W)}{\partial w_c} = \sum_{i=1}^{n}(y_{ic}x_i - \frac{e^{w_c^T x_i}}{\sum_{c'=1}^{C} e^{w_{c'}^T x_i}} x_i) - \lambda w_c$$

$$= \sum_{i=1}^{n}(y_{ic} - \frac{e^{w_c^T x_i}}{\sum_{c'=1}^{C} e^{w_{c'}^T x_i}}) * x_i - \lambda w_c$$

(2) **[5 pts + 5 bonus pts]** Let the learning rate be α, outline the algorithm you choose (SGD or Batched-SGD) for implementation. You should cover how would you like to update the weights in each iteration, how to check the convergence and stop the algorithm and so on.

```
n = 0
β = 1.2
last_log_likelihood = -inf
stop_criterion = 0.00001
converged = false
while not converged:
        log-likelihood = 0
        for i in 1 to data size / batch size:
                log-likelihood += log-likelihood for current batch
                α = (100 + n)^-β
                for c in 1 to 5:
                        w_c = w_c + α sum(gradient regarding to x_i 's in the batch)
                n++
        if (log-likelihood - last_log_likelihood) / abs(last_log_likelihood) < stop_criterion:
                converged = true
        last_log_likelihood = log-likelihood
```

I use batch SGD. Every time I process one batch, I calculate the log-likelihood of this batch according the current parameters and add it to the sum of log-likelihood of the whole epoch. Then I compute the gradient of this batch and update the parameters.

The way I calculate the learning rate can guarantee that the SGD converge fast enough and won't escape from the local minimum.

After each epoch, I check if the change in log-likelihood is small enough, i.e. it changed no more than 0.001%, so that the program can stop.

(3) **[10 pts]** After implementing your model, please use these two types of prediction to calculate and report the Accuracy and RMSE (See definition in Evaluation part) on the entire training set with the two features designed in Task 2.

Feature	Baseline Model		Feature Hashing Model	
Dataset	Training	Development	Training	Development
Accuracy	0.5825	0.5785	0.5387	0.5364
RMSE	0.8454	0.8448	0.9549	0.9568
Parameters Setting	Learning Rate alpha= $(100+n)^{-1.2}$, Regularization Parameter lambda=0.03, Batch size=900, Iterations used: 39			

[10 pts] Multi-class Support Vector Machine

After you figure them out, report only the accuracy on the training and development set using the two features designed in Task 2.

Feature	Baseline Model		Feature Hashing Model	
Dataset	Training	Development	Training	Development
Accuracy	0.5760	0.5735	0.5424	0.5392
Parameters Setting	../liblinear-2.1/train -s 2 datasets/yelp_reviews_train_baselinesvm.txt baseline_svm.model			

The results from RMLR and Linear SVM look similar. I think the reason is that they used the same datasets with the same features, which were not quite linearly separable, whereas they are both linear classifiers. I think if Kernel SVM was used here, the results would be different.

4. Feature Engineering (Continued)

[10 pts + 10 bonus pts] Describe in details your most satisfying design and the corresponding considerations, use formula to illustrate your idea if necessary. Besides, report the evaluation results on training and development set here (The reported result here should match the record on the leaderboard).

For this part, I clustered the top 10000 words into 2000 clusters. I used 5 features for each word when doing clustering, namely how many time the words appeared in reviews rated from 1 to 5 respectively. Then I used the 2000 clusters as features to train the RMLR classifier.

The reason I did this is that I wanted to use as much information as I can to train the classifier, while keeping the number of features manageable. I think this is the same consideration as hashing, but instead of relying on a non-semantic strategy like hashing to reduce the dimensionality, clustering take the similarity of words into consideration, which makes more sense, hence better performance.

Evaluation results

Training set: Accuracy **0.6110**; RMSE **0.7721**

Dev set: Accuracy **0.6051**; RMSE **0.7741**

5. One sentence of your feeling for this homework

It was fun. During the experimentations, I learned how subtle changes in the hyperparameters can affect the results. And I learned that words can be very indicative for predictions.