Name: YongKyung Oh
Andrew ID: yongkyuo
Nickname: YK

Machine Learning for Text Mining

Homework 5

1. Statement of Assurance

You must certify that all of the material that you submit is original work that was done only by you. If your report does not have this statement, it will not be graded.

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	742812	No. 2	place	716149	No. 3	food	691505
No. 4	great	585143	No. 5	like	546150	No. 6	just	521349
No. 7	time	442453	No. 8	service	431306	No. 9	really	387324

(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	10.199	8.965	14.196	29.750	36.889

Do you find something unexpected from the distribution (e.g., whether the dataset is balanced)? 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?

ANS: Data is not balanced. Most of training data belong to 4 stars and 5 stars. In this case, sampling technique (e.g. SMOTE) can be used to deal with imbalance issue. Also, in the classifier, we can set a classification weight to consider all class as same weight (e.g. panelize some classes). In this case, we have enough data to train each class. So, we can extract enough insights from the data.

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.

ANS:

$$P(y_i|x_i,W)=\prod_{c=1}^{C}\left(\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}\right)^{y_{ic}}$$

$$l(W)=\sum_{i=1}^{n}\log P(y_i|x_i,W)-\frac{\lambda}{2}\sum_{i=1}^{C}||w_i||^2$$

$$=\sum_{i=1}^{n}\log\left[\prod_{c=1}^{C}\left(\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}\right)^{y_{ic}}\right]-\frac{\lambda}{2}\sum_{i=1}^{C}||w_i||^2$$

Gradient of $l(W)$ with respect to vector w_c

$$\frac{\partial l(W)}{\partial w_c}=\frac{\partial}{\partial w_c}\left[\sum_{i=1}^{n}\log\left[\prod_{c=1}^{C}\left(\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}\right)^{y_{ic}}\right]-\frac{\lambda}{2}\sum_{i=1}^{C}||w_i||^2\right]$$

$$=\sum_{i=1}^{n}\left(y_{ic}\frac{\partial}{\partial w_c}\log e^{w_c^Tx_i}-\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}x_i\right)-\lambda w_c$$

$$=\sum_{i=1}^{n}\left(y_{ic}x_i-\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}x_i\right)-\lambda w_c$$

$$=\sum_{i=1}^{n}\left(y_{ic}-\frac{e^{w_c^Tx_i}}{\sum_{c=1}^{C}e^{w_c^Tx_i}}\right)x_i-\lambda w_c$$

(2) **[5 pts]** Let the learning rate be α, outline the algorithm (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.

ANS: I used the mini-batch SGD for RMLR. I set lambda 0.01, mini-batch size 256, learning rate 0.0025 and apply decay every iteration t. First of all, I select the random mini-batch by shuffling the training data. It can help to manage the biased result. Second, I set threshold 1e-8, which is used to check the convergence during the iteration. Difference ratio between old loss and new loss is less

than threshold, training is stopped. Random validation set, which is 30% randomly selected from training set, is used to evaluate the loss and validate model.

$$\boldsymbol{W} = \gamma_t \sum_{i=1}^{n} \left(y_{ic} - \frac{e^{w_c^T}}{\sum_{c=1}^{C} e^{w_c^T}} \right) x_i - \lambda \boldsymbol{w_c}$$

$$\gamma_t = \frac{\alpha}{1 + \beta t}$$

$$Diff = \frac{abs(old\_loss - new\_loss)}{old\_loss}$$

(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	CTF		DF	
Dataset	Training	Development	Training	Development
Accuracy	0.58548	0.58315	0.59156	0.58899
RMSE	0.80245	0.80471	0.79725	0.79902
Parameters Setting	Learning Rate alpha=0.0025 Regularization Parameter lambda=0.01 How many iterations used? ~5,000 with Batch size=256			

[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	CTF		DF	
Dataset	Training	Development	Training	Development
Accuracy	0.58567	0.58463	0.58489	0.58433
Parameters Setting	All parameters you used to run SVM. If you run SVM in terminal, include your command line here. Using sklearn.svm.LinearSVC LinearSVC(penalty='l2', loss='squared_hinge', dual=False, C=1.0, class_weight='balanced')			

4. **Feature Engineering**

[10 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).

ANS: To improve the overall performance, I selected 1000 top tokens for each class and filtered out the unique tokens (2720 out of 5000). As we check the first part, the class is imbalanced and

CTF and DF are highly depending on the majority. To balance the features for the minority, I manually consider it. If I consider SMOTE or balanced model, performance may be improved. Here is the result comparison.

	CTF		DF		BAL	
LR	**Train**	**Develop**	**Train**	**Develop**	**Train**	**Develop**
Accuracy	0.58548	0.58315	0.59156	0.58899	0.59930	0.59636
RMSE	0.80245	0.80471	0.79725	0.79902	0.77841	0.78041
SVM	**Train**	**Develop**	**Train**	**Develop**	**Train**	**Develop**
Accuracy	0.58567	0.58463	0.58489	0.58433	0.59476	0.59465

Leaderboard Score for Dev set:

Accuracy: 0.593650085982

RMSE: 0.780213480216

5. One sentence of your feeling for this homework

Is that good or not? Why?

In the perspective of feature engineering, word embedding may improve the performance. However it is hard to use w2v model due to the data dimension. Compare to the count of the token, word embedding use the embedding vectors. Therefore it can be used in the DNN and DNN may works better.