Name: Yan Zhao
Andrew ID: yanzhao2
Nickname: Yan

Machine Learning for Text Mining

Homework 5

1. Statement of Assurance

I did not receive help of any kind from anyone in developing your software for this assignment.

I did not give help of any kind to anyone in developing their software for this assignment.

I am the author of every line of source code submitted for this assignment.

I am the author of every word of your report.

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	742871	No. 2	place	716170	No. 3	food	691624
No. 4	great	585193	No. 5	like	546186	No. 6	just	521376
No. 7	time	442505	No. 8	service	431406	No. 9	really	387359

(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%	36.689%

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

The dataset is imbalanced. Top 9 tokens cover large percentage of total term frequency, and the rank 1 token has frequency nearly twice as many as rank 9 token. Also, for rating, 65% users give star 4 or star 5, while only 9% user give star 2. This reach my expectation, according to Zipf's law, term frequency of each term in collections are quite skewed, which is the nature of languages. As for rating, it is highly possible that users tend to give relatively higher stars for restaurants. In real application, what we need to do is to take advantage and handle imbalanced dataset.

[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?

For the imbalance in term frequency, it is the nature of language, and there are many techniques like tf-idf weighting to handle this bias.

For the imbalance in rating, we could adjust weights across difference classes in loss function. Abundant class has less weight and rare class has more weight. The idea is to avoid bias towards large class by balancing influence of each class to the loss function.

We can also randomly discard some instances in abundant data. This might be dangerous since we may lose some informative, but it may be safe when we have lots of 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., $\frac{\partial l(\boldsymbol{W})}{\partial \boldsymbol{w}_c}$) 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 $\boldsymbol{w}_c$ is itself a vector, whose i-th element is defined as $\frac{\partial l(\boldsymbol{W})}{\partial w_{ci}}$, where w_{ci} is the i-th element of vector $\boldsymbol{w}_c$.

$$l(W)=\sum_{i=1}^{n}logP(y_i|x_i,W)-\frac{\lambda}{2}\sum_{i=1}^{C}\left||W_i|\right|^2$$

$$=\sum_{i=1}^{n}\sum_{c''=1}^{C}\log\left(\frac{e^{W_{c''}^Tx_i}}{\sum_{c'=1}^{C}e^{W_{c'}^Tx_i}}\right)^{y_{c''i}}-\frac{\lambda}{2}\sum_{i=1}^{C}\left||W_i|\right|^2$$

$$=\sum_{i=1}^{n}\sum_{c''=1}^{C}y_{c''i}\cdot\left(W_{c''}^Tx_i-\log\sum_{c'=1}^{C}e^{W_{c'}^Tx_i}\right)-\frac{\lambda}{2}\sum_{i=1}^{C}\left||W_i|\right|^2$$

$$\frac{\partial l(W)}{\partial W_c}=\frac{\partial\sum_{i=1}^{n}\sum_{c''=1}^{C}y_{c''i}\cdot\left(W_{c''}^Tx_i-\log\sum_{c'=1}^{C}e^{W_{c'}^Tx_i}\right)-\frac{\lambda}{2}\sum_{i=1}^{C}\left||W_i|\right|^2}{\partial W_c}$$

The derivative of $\sum_{c''=1}^{C}y_{c''i}\cdot W_{c''}^Tx_i$ is zero when $W_c\neq W_{c''}$

And since $y_{c''i}$ is probability of label c'', which sums to 1, so we have

$$\sum_{c''=1}^{C} y_{c''i} \cdot \log \sum_{c'=1}^{C} e^{W_{c'}^T x_i} \equiv \log \sum_{c'=1}^{C} e^{W_{c'}^T x_i}$$

Based on this, we can simplify and calculate the partial derivative of loss function as

$$\frac{\partial l(W)}{\partial W_c} = \sum_{i=1}^{n} (y_c x_i - \frac{e^{W_c^T x_i} \cdot x_i}{\sum_{c'=1}^{C} e^{W_{c'}^T x_i}}) - \lambda \cdot W_c$$

$$= \sum_{i=1}^{n} (y_c - \frac{e^{W_c^T x_i}}{\sum_{c'=1}^{C} e^{W_{c'}^T x_i}}) \cdot x_i - \lambda \cdot W_c$$

(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.

I choose Batched-SGD for implementation, with each batch contains about 500 instances.

I initialize weight matrix W by picking a random 5 by 2000 feature matrix filled a random number in (0,0.05).

I use learning rate α and momentum in each iteration update, in order to get converge faster. In specific, I calculate $Grad = momentum * prev_{Grad} + \alpha * dW$, where dW is calculated by equation given in section (1). Then I use W = W+Grad to update weight matrix W. Parameter settings are given in the table below.

As for stop criterion, I repeat until cost difference *loss* between current iteration and previous iteration is smaller than 1e-6 (here one iteration means all data has been passed once)

I found that it is very important to improve training process, since it will effect final performance vey much. So I made a lot of experiments to tune parameters and introduce some tricks in training, as follows.

1) Training Parameter Tuning

After finishing implementation, I tune training parameters including number of iterations, stop criterion, regularization parameter, momentum parameter and learning rate. Since training data we use is quite large, a small change in learning rate may cause overshooting optimal in loss function.

One trick here is I use a damp factor equals to 0.99 to let learning rate damp at each iteration, in this way, with more iterations, I can get more close to optimal. Another trick is only increasing number of iterations can get higher accuracy on training data, but will get lower accuracy on development data, caused by overfitting. Due to this reason, for stop criterion, besides number of iterations, I also monitor training error changes at each iteration, if changes are smaller than 1e-6, I will stop training. Using these techniques, my performance on training data improve from (accuracy: 0.5418, rmse: 0.9640) to (accuracy: 0.5836, rmse: 0.8019).

2) Shuffle data at each Training Iteration

Since I use batch-SGD, shuffle data before each training iteration can help me get to optimal more quickly in gradient ascent. Using this technique, my performance on training data improve from (accuracy: 0.5836, rmse: 0.8019) to (accuracy: 0.5998, rmse: 0.7973), on development data improve from (accuracy: 0.5795, rmse: 0.8034) to (accuracy: 0.5960, rmse: 0.7985).

(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	0.5998	0.5960	0.5991	0.5959
RMSE	0.7973	0.7985	0.7978	0.7994
Parameters Setting	Learning Rate alpha=0.0001 Regularization Parameter lambda=0.001 100 iterations used Momentum = 0.4			

[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	0.5851	0.5839	0.5854	0.5848
Parameters Setting	L2-regularized L1-loss support vector classification (dual) C = 0.1 Termination criterion e = 1e-6 For training accuracy, I use 5-folder cross validation Train: $ ./train –s 3 –c 0.1 –e 1e-6 –v 5 svm_train model.model Predict: $ ./predict svm_dev model.model svm_dev_result			

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).

I tried with different kinds of ways to generate dictionary, like CTF, DF or mixed of both. Results shows no significant improvement by different dictionaries. So in the experiments, I use CTF to generate dictionaries, and have following design to improve my performance.

1) TF-iDF Term Weighting

My intuition is that there are three aspects that could have influence on representing a document. The first one is obvious term frequency, which has already been represented in feature vectors of documents in baseline. The second one is document frequency of each term. The reason is one term with high document frequency should be considered less useful than a term with low document frequency. The third one is document length. The reason is that for a long document, the average term frequency would be of course larger than that of a short document, so a term appearing k times in a short document should be more representative than a term appearing same times in a long document.

Based on the intuition above, for each term frequency, I give a penalty based on its document frequency and document length. The formula is shown below:

$$\text{Term Weight} = \text{tf} * \log\left(\frac{N-df+0.5}{df+0.5}\right) * \frac{1}{k*\frac{doclen}{avg\_len}}$$

Here in my formula, *Term Weight* is my modified feature value; *tf* represents for term weight; *N* is the total number of documents in corpus; *df* is the document frequency for each term; *doclen* is the length of document; *avg_len* is the average document length for whole corpus; ***k*** is parameter I set myself, which I adjusted by different experiments and set it equals to 1.2 finally.

As is seen from the formula, I design the feature for each term in a document from three aspects, as stated above. The first part is term frequency. The second part is my *idf* calculation, here I use RSJ weight intuited by BM25 score in Search Engine, it is a smoothing version of *log(N/df)*. The third part is my penalty on document length. After using this modification, my accuracy on training data improve from (accuracy: 0.5998, rmse: 0.7973) to (accuracy: 0.6144, rmse: 0.7379), on development data improve from (accuracy: 0.5960, rmse: 0.7958) to (accuracy: 0.6044, rmse: 0.7446).

2) Number of Features

According to Zipf's law, $Rank_t * freq_t = A * N$, where A is constant and N is total number of words in vocabulary. The percentage of terms with frequency less than 100 equals to $(Rank_{Max} - Rank_{100})/Rank_{Max} = 99\%$. In corpus exploration, I found that N equals to 645856, so the total number of words with CTF larger than 100 is approximately 6000.

Based on TF-iDF weight, I increase number of features from 2000 to 6000, covering all terms with CTF larger than 100. My accuracy on training data improve from (accuracy: 0.6144, rmse: 0.7379) to (accuracy: 0.6198, rmse: 0.7287, on development data improve from (accuracy: 0.6044, rmse: 0.7446) to (accuracy: 0.6078 rmse: 0.7589), which is tiny bit improvement.

3) Label Prior

Since the label distribution is not balanced. I adjust the gradient of different label(stars) according to following equation, grad = grad * 0.2 / beta. (beta is percentage of that label in all training data). The intuition is to balance influence across different classes to model update. However, results show no significant improvement, with performance on development data about (accuracy: 0.6060 rmse: 0.7523).

5. One sentence of your feeling for this homework

Is that good or not? Why?

I feel quite good. This assignment gives me experience on how to solve real problem, yelp review, with large training dataset and noises. And it gives me intuition on doing text mining analysis, like feature engineering and model tuning. I hope we can get some feedback on how to further improve performance with feature selection and parameter tuning.