11-741/641/441 Machine Learning for Text Mining
Homework 5: Rating Prediction from Review Text Alone

TA in charge: Guoqing Zheng

Due: Mar. 31 2016, 11:59 PM

1. Introduction

In this assignment, you will work on a real world dataset from Yelp. The ultimate goal of your job is to predict the customer rating for businesses like restaurants based on the customer review text alone. Compared with previous assignments where the features have been extracted, you need to do all these things by yourself. Nevertheless, this write-up will guide you to build a text mining based machine learning system step by step. Hopefully, after this homework you can know the general approach to solve a text related machine learning problem and understand what kind of challenges you may meet when working on the real world datasets.

This assignment consists of the following major tasks:

a. Data preprocessing
b. Feature Engineering
c. Problem Modeling and Implementation
d. Feature Engineering (Continued)
e. Evaluation

Notice: You will handle a large dataset for this homework, so please start early to download the dataset and get a sense of what kind of problem you are facing.

2. Dataset

The dataset contains 4 files:

Description	Filename	Size
Training Set	yelp_reviews_train.json	**1255353**
Development Set	yelp_reviews_dev.json	157010
Test Set	yelp_reviews_test.json	156901
Stop word list	stopword.list	319

The data are stored in JSON format. Each line of the file is a JSON object. The content of JSON is shown as follows:

```
{
  'type': 'review',
  'review_id': (encrypted review id),
  'business_id': (encrypted business id),
```

'user_id': (encrypted user id),
'stars': (star rating), **# Notice: This field is only available to training set.**
'text': (review text),
'date': (date, formatted like '2012-03-14'),
'votes': {(vote type): (count)},
}

In this homework, we will focus on using text field (the review content) only to predict the rating stars (the stars field). The review_id field will be used to submit your prediction for the development and test set.

3. Task 1 – Data Preprocessing

In the first task, we prepare the training set with the following steps:

a) Extract the useful fields from the training set
b) Tokenize the text field and remove the stop words (see stopword.list)
c) Calculate some statistics to verify your implementation.

Here are the requirements for tokenization:

1) Turn all the tokens into lower case.
2) Remove all the punctuation in the tokens (e.g., "it's" will become "its", "?" or "!" will become "").
3) Remove all the tokens that contain numbers.

[5 pts] After the above step of data preprocessing, report the top 9 most frequent tokens and corresponding counts in the report.

Rank	Token	Count	Rank	Token	Count	Rank	Token	Count
No. 1			No. 2			No. 3		
No. 4			No. 5			No. 6		
No. 7			No. 8			No. 9		

[5 pts] Report distribution of user assigned stars, i.e., the number of training instances with 1 star, 2 stars, 3 stars, 4 stars and 5 stars respectively.

Star	1	2	3	4	5
# of training data					
Percentage (%)					

[5 bonus pts] Do you find something unexpected from the distribution (e.g., whether the dataset is balanced)? Will this be problematic in training the model? If so, could you give some idea about how to address it and explain why your idea should work?

4. Task 2 – Feature Design

In this part, we will design and construct the feature space for the machine learning models (Logistic Regression and SVM) in the next task. Since we are only using review text for prediction, the generated tokens for each training instance will be the source for feature extraction.

Feature Selection based on Collection Term Frequency (CTF)

As a simple baseline approach, let us use a bag-of-words to represent each document, i.e., using a bag of tokens and the corresponding counts in each review, and ignore the order of tokens. For this approach, we need to first define a dictionary. In this baseline model, take the top-**2000** most frequent tokens from the entire training collection as dictionary and map all the training instances into sparse feature vectors.

For example, if your dictionary is [a, b, c, d, e, f, g], and a training instance contains tokens [a:2, b:2, d:1, g:1], your feature vector will be [2, 2, 0 ,1 ,0, 0, 1]. Note that the order of token in dictionary really doesn't matter, but you should make sure the order is the same for all the instances.

Feature Selection based on Document Frequency (DF)

As another simple baseline approach, let us use a bag-of-words representation for each document again, but with the top-**2000** tokens with the highest document frequencies. For example, if token 'apple' appears in document 1, 5, 8, the document frequency for 'apple' is 3 (Though in each of the three documents, 'apple' may appear more than just once). After determining the dictionary, use the same way as CTF to convert each document to a feature vector.

5. Task 3 –Model Design

Our interest here is to predict the ratings (from 1 star to 5 stars) over items (restaurants, shops, pharmacies, etc.) based on the review (text document) by each user on that item. We can consider this as a multi-class classification problem[1] where the system takes the feature vector of each review (which we constructed in Task 2) as the input, and predicts the rating (among 1 to 5) as the class label for the item being reviewed. Specifically, we want you to firstly implement, train and evaluate a regularized multi-class logistic regression (RMLR) method on the training set and development set we provided. Secondly, we want you to train a Support Vector Machine (SVM) model using liblinear, and compare the result of SVM on the same data set with that of the RMLR.

Multi-class Logistic Regression

Similar to we have learned in class about binary logistic regression, for the multi-class version of logistic regression we need to 1) formulate the objective function, and 2) derive the update rule for

[1] One may argue that whether or not we should consider ordinal regression (which takes the order of output labels into account, i.e., 5 > 4 > 3 > 2 > 1) as an alternative. We think that both approaches are fine, but we just want to focus on multi-label classification model in this assignment.

iterative improvement of model parameters (regression coefficients) assuming that we are using gradient ascent (or descent). Letting $c \in \{1,2,\dots,C\}$ be the class-label indicator, we estimate the probability of the class as:

$$P(y = c \mid \boldsymbol{x}, \mathrm{W}) = \frac{e^{\boldsymbol{w}_c^\top \boldsymbol{x}}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^\top \boldsymbol{x}}}$$

where $\boldsymbol{x}$ is the feature vector of a review, $y \in \{1,2,\dots,C\}$ is a class label, and W is the model-parameter matrix whose columns are w_c for $c \in \{1,2,\dots,C\}$. Given a training set of labeled pairs $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ we optimize the model parameters as:

$$\mathrm{W}^* = \underset{\mathrm{W}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i \mid \boldsymbol{x}_i, \mathrm{W})$$

Notice that we assume all the training data are independent and identically distributed (i.i.d.), so we use a product of the probabilities over individual training pairs.

When we actually implement the algorithm, we often choose to maximize the log of the conditional likelihood. This is because computing the product of probabilities in log space will be more stable and the calculations are also easier. Also, we usually add a regularization term (we use L2 regularization in this assignment) to prevent overfitting. As a result, we can write our regularized objective function as:

$$l(\mathbf{W}) = \sum_{i=1}^{n} \log P(y_i \mid \boldsymbol{x}_i, \mathbf{W}) - \frac{\lambda}{2} \sum_{i=1}^{C} \|\mathbf{w}_i\|^2$$

where λ is a pre-specified parameter that controls the weight of the regularization term.

(1) **[5 pts]** In order to derive the gradient of our objective function (the regularized conditional log-likelihood), it is convenient to redefine the output label of i-th input (i.e., y_i) using a indicator vector $\boldsymbol{y_i} = (y_{i1}, y_{i2}, \dots, y_{iC})^T$, whose elements are defined as

$$y_{ic} = \begin{cases} 1, & \text{if } y_i = c \\ 0, & \text{otherwise} \end{cases}$$

Then the conditional probability of y_i given $\boldsymbol{x}_i$ and W can be written as:

$$P(y_i|\mathbf{x}_i, \mathbf{W}) = \prod_{c=1}^{C} \left(\frac{e^{\mathbf{w}_c^T \mathbf{x}_i}}{\sum_{c'=1}^{C} e^{\mathbf{w}_{c'}^T \mathbf{x}_i}} \right)^{y_{ic}}$$

Now we want you to prove that the gradient of $l(\boldsymbol{W})$ with respect to vector $\boldsymbol{w}_c$ (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 $\boldsymbol{w}_c$.

(2) **[5 pts]** A standard approach to the training of logistic regression is to use Gradient Descent (notice that the goal of our problem is to maximize the regularized log-likelihood, so we will actually use gradient ascent instead of gradient descent). Notice that when the training set is very large, it would be computational expensive to compute the gradient on the entire training set for each iteration. A remedy is to use Stochastic Gradient Descent (SGD) instead, which means to update the model parameters using the gradient on only one training instance per iteration.

[5 bonus pts] Generally speaking, the SGD will converge much slower than GD. A midpoint alternative is to use Batched SGD, which means to divide the training set into equally sized subsets (e.g., 100 instances per subset) and to compute the gradient on each subset per iteration. Clearly, there is a trade-off between the number of iterations and the cost of computing the gradient per iteration.

Let the learning rate be α. Outline the algorithm you choose (SGD or Batched-SGD) for implementation. You should specify the update rule for the model parameters ($\boldsymbol{W}$ or $\boldsymbol{w}_c$) in each iteration, how to check the convergence of the solution, the stop criterion and so on.

(3) **[Graded by source code]** Implement the Regularized Multi-class Logistic Regression (RMLR) using the optimization algorithm you chose from (2). You can use any 3rd party package to improve the computational efficiency in subroutines, but you must write the RMLR by yourself.

Suggestions:

a) To check the correctness of your algorithm during the developing phase, one may often check the changes of log-likelihood function. However, when training set is very large, evaluating the log-likelihood function is very expensive. So one trick here is to hold some (e.g., ~3%-5%) of your training data as a small evaluation dataset, and monitor the performance on the dataset during each iteration.

b) To check the performance of your algorithm, it is very useful to perform a k-fold cross validation or random cross validation.

(4) **[10 pts]** Since the RMLR can provide the conditional probability for each class, then we can make two types of prediction. The first one is a hard prediction, which class that maximizes the conditional probability, e.g.,

$$\hat{y}_{hard} = argmax_y \, P(y|\boldsymbol{x}, W)$$

So your prediction can only be the integer among 1 to 5 in the rating prediction problem.

The second one is a soft prediction, which is the expectation of prediction over the conditional distribution, e.g.,

$$\hat{y}_{soft} = E[y] = \sum_{i=1}^{C} i \cdot P(y = i|\boldsymbol{x}, W)$$

In this case, your predicted rating can be any floating number between 1 and 5.
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				
RMSE				
Parameters Setting	Learning Rate alpha=? Regularization Parameter lambda=? How many iterations used?			

[10 pts] Multi-class Support Vector Machine

In this part, we will use the LIBLINEAR (Version 2.1) for multi-class classification. You can download the package from its official website. You will need to read through the documentation to see how to use that package and then adapt the features you generated for RMLR to SVM input format. For the training, you should use **L2-regularized L2-loss support vector classification**. 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				
Parameters Setting	All parameters you used to run SVM. If you run SVM in terminal, include your command line here.			

Compare the results from RMLR with SVM. Do the results look similar or does one of them look better? Could you try to give some explanation?

6. Task 4 – Feature Engineering (Continued)

Now let's focus on your RMLR model. Since not all the tokens used in feature vectors are equally important, then in this part you are required to come up with some creative idea to re-design the feature. For re-designing feature vector, you can increase the size of the dictionary, or you can find other ways to select tokens as dictionary, or you can different weights for each feature (other than just term frequency in the two baseline feature models), but the only resource you are allowed to use is the review text.

Notice that this is an open-ended problem. The ultimate goal is to improve the performance of your RMLR model for both **hard** and **soft** metrics (see evaluation part for more details). Your approach should at least beat the CTF and DF feature model. Since nearly half of the points for

this assignment will be assigned based on the online evaluation performance on the development and test sets (See grading part for more details), you may want to iterate on this part to improve your performance.

[10 pts + 10 bonus pts] Describe in detail your most satisfying design and the corresponding considerations, using formulas to illustrate your idea if necessary. Additionally, report the evaluation results on the training and development sets here (The reported results here should match the record on the leaderboard).

7. Evaluation and Online Testing

We will use two metrics (hard one and soft one) to evaluate your rating prediction results.

Hard Metric

The hard metric is the overall Accuracy (under zero/one loss) for your multi-class classifier, which is defined as the number of instances that you correctly predict its star divided by the size of the testing set. Note that your prediction used for evaluation must be an integer between 1 and 5.

Soft Metric

The soft metric is the Root Mean Square Error (RMSE) for your prediction results, which is defined as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{r}_i - r_i)^2}$$

where $\hat{r}_i$ is your predicted rating for i-th instance, r_i is the corresponding true rating and n is the size of dataset used for testing. Note that the prediction used for evaluation in this part can be any floating number between 1 and 5 (e.g., 4.2, 3.55).

Online Testing Service (website) Submission

Your prediction results for development and test sets will be evaluated online since both of them will be graded (See Grading part for details). As usual, the feedback (hard and soft metrics) will be only provided for development set.

For submission, the prediction results for development set should contain 157010 lines while the prediction results for test set should contain 156901 lines. In this prediction file, **the order of the predictions should be exactly the same as the testing data**. Each line should contain two numbers separated by blank space, one for hard prediction and the other for soft prediction. For example:

```
<hard prediction><blank space><soft prediction>
3 2.8
4 4.1234
…
```

```
5 4.95
```

Notice: Even if you only want to test for hard metric (e.g., for the SVM task), you should still put some number in the soft prediction part (e.g., you can just duplicate your hard prediction) to make sure the testing script can recognize your result.

8. Grading and Submission

Grading [100 pts + 30 bonus pts]

(1) [50 pts + 20 bonus pts] Written report that contains all the problems in this homework. A template is provided to help you get started.

(2) [40 pts + 5 bonus pts] Online Testing Results (Rank)

- [10 pts] Performance on the development set. Yes, that's result you see on the leader board.
- [30 pts] Performance on the test set. Will be partially released after the deadline of the homework.
- [5 bonus pts] For top 10 ranking on the development set.

(3) [10 pts + 5 bonus pts] Source Code

- Bonus points will be assigned to TA friendly code (e.g., good coding style, proper optimization and reasonable documentations). This is a Computer Science course – the instructors will actually care about the quality of your source code.

Submission

For submission, please compress all the following files into a .zip file and use HW5-YourAndrewID.zip as the filename.

a. All Source Code in **src** folder
b. Report HW5-YourAndrewID.pdf