Project Report for a Self-implemented Log Linear Model

Shudi Hou
the School of EECS
housd@pku.edu.cn

1 Introduction

This is the report for the self-implemented log linear model. In this report, I will briefly introduce the task and the structure of the log linear model. After that, I will describe the features I extracted and how I extract them from given text. Then, some implement details will be explained, including text preprocessing, lemmatization and feature building. In the last section I will analyze the results of log linear models, based on Acc, Macro F1 and the features that models believes important.

2 Log Linear Model

2.1 Problem Definition

Let $\mathcal{X} = \{x_1, ...x_n\}$ be a input text set, and $\mathcal{Y} = \{0, 1, .., C\}$ be the label space. For each instance in X, there is a corresponding ground truth label $y_i \in \mathcal{Y}$. We consider a training dataset $\mathcal{D} = \{(x_i, y_i)|x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$. Taking the restriction of log linear models in consideration, we perform a feature function $f : \mathcal{X} \times \mathcal{Y} \to R^m$ to map the (x, y) pairs in training dataset to m dimension vectors $X_i \in R^m$ in input feature space $\mathcal{X}'$. This projection could be written as

$$X_i = f(x_i, y) \ \ \forall x_i \in \mathcal{X}, \forall y \in \mathcal{Y}$$

And $X_{ij} = f_j(x_i, y)$ denotes the jth element of the feature vector of ith instance in the dataset. Our goal is to find the corresponding label for input texts using log-linear models, given the feature vector of the texts.

2.2 Log Linear Model

For each $x \in \mathcal{X}$, $y \in \mathcal{Y}$, the *conditional probability* of y is calculated as:

$$p(y|x; \boldsymbol{\lambda}) = \frac{exp(\boldsymbol{\lambda} \cdot \boldsymbol{f}(x, y))}{\sum_{y'} exp(\boldsymbol{\lambda} \cdot \boldsymbol{f}(x, y'))}$$

Here exp(x) = e^x , and $\boldsymbol{\lambda}$ is the parameter vector that needs to optimize. $\boldsymbol{\lambda} \cdot \boldsymbol{f}(x, y) = \sum_k \lambda_k \cdot f_k(x, y)$ is the inner product between $\boldsymbol{\lambda}$ and $\boldsymbol{f}(x, y)$.

Intrinsically, we could see that the bigger $p(y_i|x_i; \lambda)$ get, the better the model is. Moreover, in order to achieve a better overall performance, $\prod_i p(y_i|x_i; \lambda)$ needs to be maximized. We could formalize our optimize target as:

$$\boldsymbol{\lambda}_{goal} = arg \max_{\boldsymbol{\lambda}} \prod_i p(y_i|x_i; \boldsymbol{\lambda})$$

For calculation simplicity, we choose to maximize $\log \prod_i p(y_i|x_i; \boldsymbol{\lambda})$

3 Features

There are two kinds of features: word based features and non-word based features. There are 3 kinds of word based features: unigram features, bigram features and trigram features. For example, for sentence "I love this movie.", it has unigram feature (love), bigram feature (love this) and trigram feature (love this movie). The times that punctuation ! occurs text is a non-word based feature. Words(bigrams, and trigrams) which occurs more than α times and less than γ times in the train dataset are treated like features. α and γ are hyper parameters.

For efficiency, features are calculated without considering labels. It is because if a word is in the vectors, then no matter what label is assigned to it, the corresponding element is always 1.

3.1 Unigram Features

Unigram features are simple yet effective because different word occurrence implicitly shows attitudes. To be more specifc, if a passage is full of words like "bad", "awkward", then it is most likely to have opposite emotions with a passage with "good", "comforting" and "beautiful". "unigram features" are are written as "UF" for simplicity.

3.2 Bigram Feautures

To extract bigram features, texts are reformulated as <START> text <END>. Bigram features are effective, for they extract more comprehensive information compares to unigram features. To be more specific, bigram features can extract information like "not good". What's more, we can expect that bigram features works well on ratings. For example, "2 stars" and "5 stars" have different labels, but splitting them into "2", "3", "stars" makes it hard for the model to find the inner correlation. "bigram features" is written as "BF" in the following section for simplicity..

3.3 Trigram Features

To extract trigram features, the given text is reformed as <START><START>text<END><END>. Trigram features are considered useful, for it can capture a simple SVO sentence's structure. However, empirical results showed that trigram features are not so important as bigram and unigram features in SST dataset, and for long text sentiment classification, models utilizing trigram features outperformed other models. "triigram features" is written as "TF" in the following section for simplicity..

3.4 Non-word Features

Beside features based on words, I also utilize punctuation and total passage length to form features.
To be more specific, if the passage is longer than a given number of words (30 for SST and 200 for Yelp), the corresponding feature is set to 1. Also, if the number of '!', '?' and '.' is bigger than usual passage, their corresponding feature is set to 1, too. "non-word features" is written as "NW" in the following section for simplicity..

4 Implement Details

4.1 Text Preprocessing

Various ways of text processing were experimented and some of them are useful, Table 1 for an example of text preprossessing.

Text Cleaning In the first step, all the upper case characters are replaced by lower case characters, then abbreviates are replaced by full names. After that,special tokens are removed. Only characters, numbers and selected punctuation(.,:()?!) are kept. Finally, I used blank space to separate words and punctuation. Empirical result show that filtering stop words like "I", "and" degrades the performance of log linear models, so I choose to keep them in processed texts. "stopword filtering" is written as "SW" in the following section for simplicity.

Step	Text
0	The Rock is destined to be the 21st Century 's new " Conan " and that he 's going to make a splash.:)@#!
1	**the rock** is destined to be the 21st **century is** new **" conan "** and that **he is** going to make a splash.:)@#!
2	the rock is destined to be the 21st century is new **conan** and that he is going to make a splash**.:)!**
3	the rock is destined to be the 21st century is new **conan** and that he is going to make a splash **. :) !**
lemma	the rock is **destine** to be the 21st century is new conan and that he is **go** to make a splash . :) !
stop-word	the rock destine 21st century new conan he go make splash . :) !

Table 1: Example of text preprocessing.

Lemmatization Utilizing lemmatization dictionary on github[1], words are translated to their lemmas. This technique significantly cut down the size of vocabulary, and improved the performance.

Feature Building Sort the extracted tokens by their frequency and select the features based on rules mentioned previously.

4.2 Gradient Ascend

The gradient is calculated as:

$$\frac{\partial LL(\boldsymbol{\lambda})}{\partial \lambda_i} = \sum_k f_i(x_k, y_k) + \sum_k \sum_{y'} f_i(x_k, y') p(y'|x; \boldsymbol{\lambda})$$

For efficiency, features are first calculated without taking labels into consideration. Then, when calculating gradients, features are expanded based on their labels. Please see the code for details.
The goal of our training is to find the suitable λ for the dataset.After calculating the gradient, we use:

$$\lambda^{t+1} = \lambda^t + \beta \times \lambda^t$$

to update lambda.

[1] https://github.com/michmech/lemmatization-lists/blob/master/lemmatization-en.txt

Preprocess	Word
Full	ultimately, well, powerful, (, not), enjoy, entertain, year, performance, bad
w/o SW	well, like, year, (be not), entertain, (, but), bad, performance, too
w/o SW & NW	(doe not), well, like, year, (be not), entertain, (, but), performance, bad, too
UF & BF & L	plot, well, (the well), (be not), entertain, year, (, but), performance, bad, too
UF & L	move, beautifully, powerful, doe, well, entertain, performance, year, bad, too
UF	?, does, beautifully, powerful, entertaining, performances, moving, bad, too, best

Table 2: Words with highest lambdas in SST dataset

4.3 Optimization

Stochastic Gradient Ascend In order to perform the optimization on my PC, I choose to use SGD. SGD actively update $lambda$ for each instances. It saves time and memory, but it may cause unstable result because it may be easily mislead-ed by noisy data.

4.4 Hyperparameters

β is fixed on 0.005, for SST, the upper bound of feature frequency α is 1000, and the lower bound γ is 10. For Yelp, α is 3000, γ is 2000.

5 Result Analysis

The result of SST data varies between **38-42%** and the result of Yelp data varies between **44-49%**. Due to the time limitation, I will mainly analyze the result of SST datasets, and compare it with Yelp dataset. The result of these 2 datasets is in Table 4 5. Table 2 3 shows the features with top-10 biggest λ. To some extend, it shows how important this feature is to models.

Surprisingly, we could find out from the Figure 3 2 1,the trigram features, the filtering of stopwords and the non-word features are useless

Preprocess	Word
Full	(one wrong), (. horrible), (never come back), (2 star .), (waste money), (never return), poison, (. never go), (3 star .), (three star)
w/o NW	(. horrible), (never come back), (recommend !), (waste money), (2 star .), (never return), poison, (. never go), (3 star .), (three star)
w/o SW & NW	(. i highly), (of the wrong), (never return), (<START> wrong), (<START> <START> wrong), (avoid this), poison, (<START> love this), (3 star .), (three star)
UF & BF & L	(be disgust), (would .), (. wrong), (never return), (. highly), poison, (avoid this), (<START> horrible), (three star), (<START> wrong)
UF & L	contract, downhill, yuck, phenomenal, garbage, insult, unacceptable, inedible, filthy, poison
UF	yuck, downhill, superb, unacceptable, inedible, phenomenal, cancel, garbage, rip, filthy

Table 3: Words with highest lambdas in Yelp dataset

Preprocess	LL	F1	Acc
Full	-2537.2	0.351	0.390
w/o NW	-2539.9	0.352	0.390
w/o SW & NW	**-2413.4**	0.381	0.411
UF & BF & L	-2421.6	**0.382**	**0.414**
UF & L	-2486.8	0.354	0.399
UF	-2469.5	0.361	0.399

Table 4: Results of SST dataset

Preprocess	LL	F1	Acc
Full	-162799.9	0.442	0.446
w/o NW	-163425.6	0.442	0.437
w/o SW & NW	**-150188.3**	**0.491**	**0.495**
UF & BF & L	-164715.5	0.441	0.437
UF & L	-197003.0	0.269	0.289
UF	-194597.7	0.290	0.305

Table 5: Results of Yelp dataset

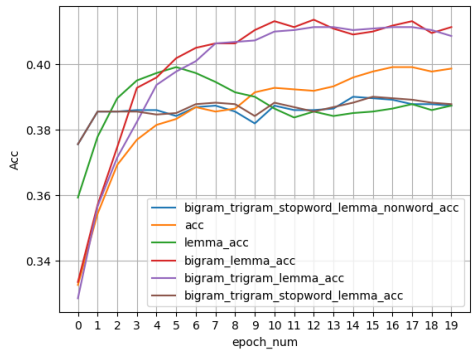

Figure 1: the Acc of SST dataset

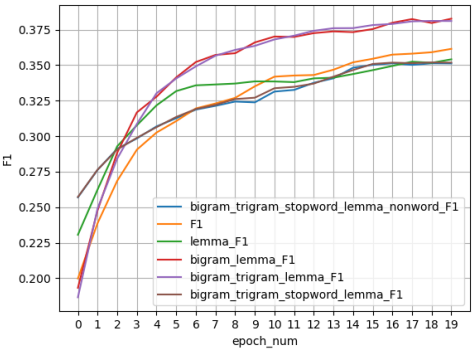

Figure 2: the F1 of SST dataset

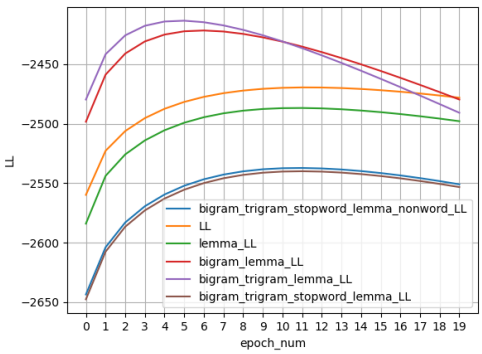

Figure 3: the LL of SST dataset

in SST datasets, and a higher LL does not always ensures a higher accuracy or macro F1 score. Results on Yelp dataset proved that, too.

Also, although lemmatization greatly improved the log linear model's performance when used with bigram features and trigram features, It did not work well on models that use unigram features as their only features. From Table 2 we could see, the word "best" is an important feature in unigram models. However, the important word "best" when using only unigram feature will be changed to "well" according to the lemmatization rules. Intrinsically, "This is well." and "This is the best." shows different attitudes, and if there are no other information(in bigrams and trigrams, words' neighbors to offer some extra information.), it may be hard for log linear models to distinguish the sentiment in texts.

Filtering stopwords are hurting models' performance. One important reason is there are "stopwords" that mdoels believe it is important(See Table 2). Also, filtering stopwords may hurting the fluency of the original text, making it less similiar to natural language, and even hurt its logic.

Using non-word features is not helping the models. Maybe it is because the use of punctuation and the length of texts reflect users personal preference, so that theres no clear difference between different kinds of sentiments. Also, there are only 4 non-word features, the impact of them is relatively small comparing to word-based features. We can also infer from the table that important features are always words, but not non-word features.