Sentiment analysis on yelp dataset

Naga Venkata Sai Indubhaskar Jupudi NJUPUDI@UCSC.EDU
Trivikram Bollempalli TBOLLEMP@UCSC.EDU
Chandrahas Jagadish Ramalad CJAGADIS@UCSC.EDU
Karthikeyan Subramanian KSUBRAM1@UCSC.EDU
GitHub Link : https://github.com/jupudibhaskar967/CMPS-242-Project.git

1. Problem statement

In this project, we aim to perform sentiment analysis on the yelp dataset. Using reviews and rating data for training, given a new review we classify it as a positive/negative review (binary) and also predict the rating (multiclass). A review is considered positive if it has received a rating of 4 or 5 and negative if it's rating is 1 or 2. The classification problem can be solved by a set of algorithms. Every algorithm has its own advantages and disadvantages in terms of accuracy and model complexity.

For example, naive Bayes classifier is faster to compute than logistic regression classifier for huge datasets. But the disadvantage with the former is that it assumes the features are independent where as the latter has no such assumptions which can lead to better precdiction. Our work mainly concentrates on implementing these two classifiers and techniques to improve their performance. We have adopted multi-processing for feature extraction to make it considerably faster. We have implemented stochastic gradient descent as an optimizer for logistic regression. We have also implemented Naive Bayes classifier as a baseline to compare our results from logistic regression. Finally, we contrast these algorithms based on time taken for execution and performance metrics like accuracy, precision and recall.

2. Feature Extraction

We have extracted the features for respective algorithms using our own constraints instead of using countvectorizer(). This is because countvectorizer() is throwing a run-time exception for a large dataset (>300k reviews).

2.1. Binary classification

For logistic regression, we need to form a count vector from bag of words. To generate this, first we extracted 4/5 th of our total dataset (which is our training data) and extracted the words and their respective counts into a dictionary. We make sure none of the stop words are added to this dictionary. Further, we removed all the punctuation and numerical characters from the words before adding them to the dictionary.

We then apply three constraints on this dictionary to form a bag of words. One, if a word occurs for 'x' times in all the positive reviews then we considered that word into bag of words only if the same word occurs for less than 4/10 th times in the negative review and vice-versa. Second, total word count for a specific word should be more than 14. Third, we check if length of the word is more than 2 characters thereby restricting the number of features to 8895. The exact numbers we have used is just an arbitrary choice and can vary to adjust the accuracy. The intuition behind this is, a word which have equal count over both postive and negative reviews and if a word occurs over all the reviews for very less frequent times (<10 or so which might be a typing mistake) will be having a very less impact over the decision rule and single character words may not effect the meaning of the review when considered as a whole.

After obtaining bag of words, we formed data vector 'x', using multi-processing (as it is both compute and memory intensive operation) to make it more faster as each review can be formed into a count vector indepedently. For this we have used multiprocessing.Pool library in python. We used pool.map_async() method to make all the individual vectors into a single data vector.

For Naive Bayes algorithm, we directly used all the words in the initial dictionary without using any constraints as there is no requirement for us to form the data vector as we did in logistic regression and the computation is way faster because of the assumption that features are independent of each other.

2.2. Multiclass classification

For logistic regression, we did not use the same constriants as we did earlier because there are multiple classes instead of 2. Alternatively, we have selected most frequent 10k words from the dictionary after extracting the reviews. Later, we have formed the data vector 'x' in the same way

as above using multi-processing.

We did the same thing for Naive Bayes algorithm, as we did in the binary classification.

2.3. Most distinctive features

The following tables shows a set of disctinctive features which shows a glimpse of our dataset.

Table 1. Distinctive words of the dataset.

CLASS	DISCTINCTIVE WORDS
POSITIVE	SATISFYING, HEAVENLY, TREATS, SWEETNESS, HEAVEN, INCREDIBLE, GEM, ADDICTED, COMPLEMENT, GREAT, CHEERS, AWESOME, CREAM, CANDIES, CONCOCTION
NEGATIVE	UNINTERESTED, ROACHES, BLATANT, DISPUTE, KILLS, ANNOYED, UNETHICAL, PLAGUE, TRASH,CREEPY, FLAVORLESS, DISRESPECT, UNSALTED, MANAGER, INSULTED

3. Model Formulation

From Feature Engineering, we get input data and its labels. we will use this to train our logistic regression algorithm. We have implemented both logistic regression for two classes and also multi-class logistic regression.

3.1. Binary Logistic Regression

$$P(C_1|X) = \frac{p(X|C_1).p(C_1)}{p(X|C_1).p(C_1) + p(X|C_2).p(C_2)}$$

The probability for the first class is given by

$$P(C_1|X) = y(x) = \sigma(W^{\mathrm{T}}X + b)$$

The likelihood is given as

$$P(t|W) = \prod_{n=1}^{N} (y_n)^{t_n}(1-y_n)^{1-t_n}$$

The cost function is

$$E(w) = -lnp(t|W) = -\sum_{n=1}^{N} t_n ln(y_n) + (1-t_n) ln(1-y_n)$$

Gradient is calculated as follows

$$\nabla E(w) = \sum_{n=1}^{N} (y_n - t_n)\phi_n$$

3.2. Multiclass Logistic regression

The probability of a class in logistic regression with multiple classes is given by:

$$P(C_k|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|C_k).p(C_k)}{\sum\limits_{j} p(\boldsymbol{X}|C_j).p(C_j)}$$

that is,,

$$P(C_k|\boldsymbol{X}) = y_k(\phi) = \frac{exp(a_k)}{\sum\limits_{j} exp(a_j)}$$

where

$$a_k = lnp(\boldsymbol{X}|C_k) + lnp(C_k)$$

We estimated the parameters using Maximum Log Likelihood estimation. For N data points, the likelihood function is

$$p(T|\boldsymbol{w_1}, .., \boldsymbol{w_k}) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

where,

$$y_{nk} = y_k(\phi_n)$$

Taking log on both sides and negation of it, we get the below negative log likelihood function:

$$E(\boldsymbol{w_1}, ..., \boldsymbol{w_k}) = -lnp(T|\boldsymbol{w_1}, ..., \boldsymbol{w_k}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln y_{nk}$$

E(w) is our cost function which we want to minimize to estimate W parameters. The gradient for the cost function is given by

$$\nabla E(\boldsymbol{w_1}, ..., \boldsymbol{w_k}) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi_n$$

3.3. Optimization techniques

We used fmin_l_bfgs_b and stochastic gradient descent to minimize the cost function and obtain W. fmin_l_bfgs_b is from scipy.optimize module in python. It uses the gradient that we calculated above in minimizing the cost function.

We also implemented Stochastic Gradient Descent(SGD). SGD is similar to gradient descent algorithm. In gradient descent, we find gradient for the entire training dataset and then update W using W = W - alpha*gradient(w,x,y). Here alpha is the learning rate, W is the weight vector, x,y are training data. At each step, gradient descent update W, hoping that it minimizes the cost function. This is true as long as we choose proper alpha and gradient descent converges close to global minimum since the cost function is convex.

In SGD, w is updated by finding gradient over each sample. We can also find gradient over a batch of training data which is a compromise between the two extremes. Finding gradient over a batch increases performance compared to single example because python can use the underlying vectorization libraries. We have implemented SGD in this way where you can specify batch size.

4. Evaluations

We performed all our experiments on a server that has 24 physical cores (with hyperthreading 2) and 128GB of DRAM. We are testing our binary and multinomial logistic regression algorithms on yelp dataset. We are using three metrics viz., Accuracy, Precision, Recall to evaluate these algorithms. These three metrics are calculated as follows.

Accuracy = (number of correctly classified test cases) / (total number of test cases)

Precision = (true positives) / (true positives + false positives)

Recall = (true positives) / (true positives + false negatives)

We also compare the performance of our logistic regression algorithm with naive bayes approach, which we implemented from scratch and discuss tradeoffs. We compare the performance of logistic regression when fmin_l_bfgs_b and Stochastic Gradient Descent are used as optimizers.

5. Results

5.1. Effect of parallelism

We implemented multi-processing to compute the feature vector in order to make it faster as it is a compute intensive operation. The following table is drawn to illustrate the time taken for extracting the features from data of size 100k and 50k reviews. With parallelism, we are able to achieve it in approximately 1/10 th of the total time without parallelism.

Table 2. Execution time for extraction of features in Logistic regression classifier.

PARALLELISM	SIZE	FEATURES	TIME
NO	100K	9049	65M36.271S
NO	50K	5323	18M32.441S
YES	100K	9049	7M8.291S
YES	50K	5323	2M36.947S

As explained earlier, we have designed our feature extraction algorithm because the built-in function countvectorizer() is throwing a run-time exception for a huge dataset which is beyond 300k reviews.

Table 3. Parallelism vs countvectorizer()

METHOD	FEATURES	TIME
PARALLEL	17083	42M27.394S
COUNTVECTORIZER	10K	RUN-TIME ERROR

5.2. Binary classification

We have extracted 50k reviews out of which we use 40K reviews for training and 10k reviews for testing. The total number of features are 5437. Figure 1 illustrates the performance statistics of logistic regression classifier with fmin_l_bfgs_b vs stochastic gradient descent optimization. As it is evident from the figure, the former optimiztion is 16 times slower than the latter and also good accuracy can be obtained by carefully varying the alpha parameter in SGD.

Figure 1. fmin_l_bfgs_b vs stochastic gradient descent

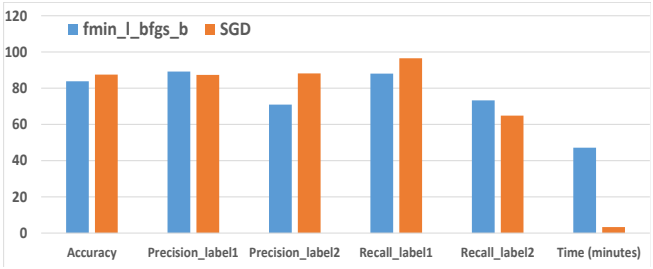
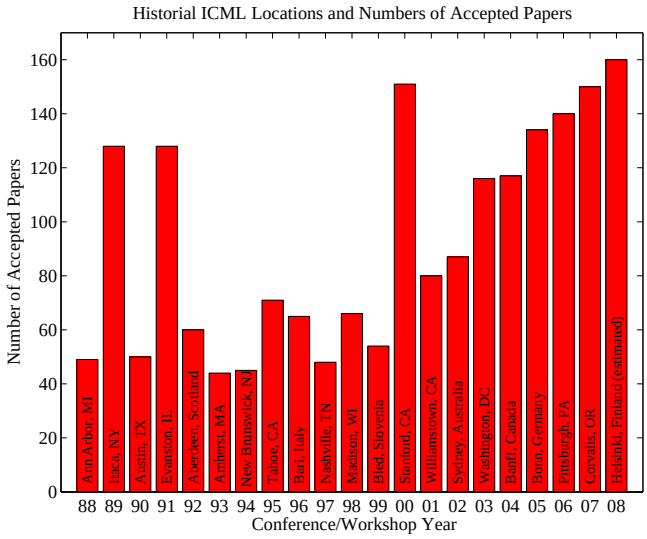
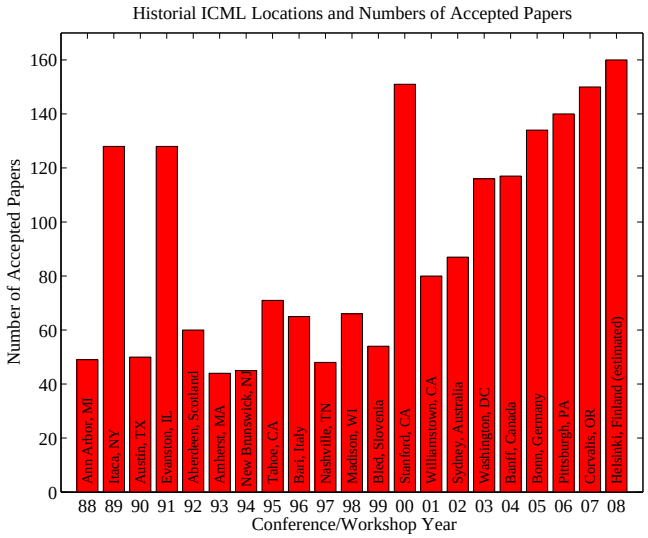


Figure 2. stochastic gradient descent vs Naive Bayes classifier



5.3. Multiclass classification

Figure 3. stochastic gradient descent vs Naive Bayes classifier



6. Conclusion

References

Langley, P. Crafting papers on machine learning. In Langley, Pat (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.