Logistic Regression Sentiment Analysis

CS 4742
Natural Language Processing

Fall 2024

Instructor – Dr. Michail Alexiou

Michael Rizig, Sam Bostian, Colton Baldwin

I. Methodology

The goal of sentiment analysis is to determine if a particular article, comment, review or any group of text conveys a positive or negative tone. One method that can be used to achieve this goal is to use logistic regression. Before a logistic regression model can be used the text that is to be analyzed needs to be preprocessed and converted into a feature vector. The steps to preprocess the text includes tokenizing the text into a bag of words; which represents the text as an unordered group of words. Other features also include accounting for certain punctuation, certain words, certain types of words or the number of words in the text.

A. PREPROCESSING TEXT DATA

For this project, we initially import the testing and training csv files that contain Amazon reviews with labels to classify them with a positive or negative sentiment. To be able to turn the text into a format that a logistic regression can use certain elements need to be removed and important parts of the text need to be converted into numerical features. The features that were chosen to use in this project were the number of positive lexicons, the number of negative lexicons, if a review contains the word no, if a review includes an exclamation (‘!’) point, and the natural log of the total number of words in the review. First, all the letters were converted to lowercase. To do this properly all punctuation needed to be removed except for the ones that were found to be important. The stop words were removed which are commonly used words that have no bearing on the sentiment of the text. Two separate files were obtained that are considered to be positive and negative. The lists were compared to the remaining words that remained and they were tallied in their respective features.

B. FEATURE EXTRACTION

B-1 : HAND PICKED EXTRACTION: Code Documentation

```
def extract_features(dataset, training=True):
    """ Takes in set of tokens and returns feature set
    output X = [x1,x2,x3,x4,x5, c]
    x1 = # of positive lexicons
    x2 = # of negative lexicons
    x3 = # of negations
    x4 = count("?")
    x5 count(keywords)
    """

    count=0
    for sample in dataset:
      # this function is run by multiple threads, so mutex is required.
       lock.acquire()
       if training:
           out = open("dataset/training_features.csv",'a')
       else:
           out = open("dataset/testing_features.csv", 'a')
       x1,x2,x3,x4,x5 = extract1(sample[0])
       out.write(f"{x1},{x2},{x3},{x4},{x5},{sample[1]}\n")
       lock.release()
       count+=1
       if count%(len(dataset)/5)==0:
```

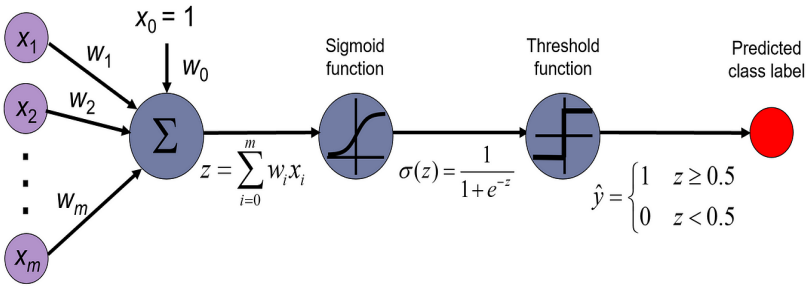
```
            print("Progress on thread ID ", threading.get_ident(), ": ",
100*(count/len(dataset)), "%")
    return
```

```
def extract1(sample):
    # how many tokens appear in positive lexicon dict
    x1 = len([x for x in sample.split(" ") if poswordsdict.get(x,False)==x])
    # how many tokens apppear in negative lexicon doct
    x2 =len([x for x in sample.split(" ") if negwwordsdict.get(x,False)==x])
    # counts negations
    x3 = 0
    # how many '?' tokens exist in sample
    x4 = sample.count("?")
    # how many key positive words appear in sample
    x5 = sample.count("love") + sample.count("amazing")  + sample.count("loved")+  +
sample.count("great")
    # extract pairs of 2 words to count number of negations
    ngrams = extract_ngrams(sample,2)
    # count number of negations
    for n in ngrams:
        if negwwordsdict.get(n[0],False) or n[0] == "not" or n[0] == "dont"  or n[0]
== "don't" or n[0] == "didn't" or n[0] == "didnt"  and poswordsdict.get(n[1],False):
            # negative negation
            x3+=1
    return (x1,x2,x3,x4,x5)
```

B-2 : EMBEDDINGS BASED FEATURES: Code Documentation

```
def extract_features(dataset,):
    weightspre = []
    from sentence_transformers import SentenceTransformer
    model_st = SentenceTransformer('distilroberta-base')
    counter=0
    N = len(dataset)
    for d in dataset:
        counter+=1
        encoded_seq =  model_st.encode(d[0])
        weightspre.append((encoded_seq,d[1]))
        if counter%5000 ==0:
            print((counter/N)*100)
    return weightspre
```

C. LOGISTIC REGRESSION



Logistic regression is a single neuron that uses a linear combination from the feature vector created from the processed data and a group of weights. Logistic regression is commonly used because it is a simple model that uses the sigmoid function to classify the probability of a sample belonging to a certain class. The sigmoid function is used because it binds the probability between 0 to 1. A threshold function is used to classify the probability as one of the two classes available. This type of learning is the basis for many other learning algorithms. While sigmoid is the activation of choice for logistical regression, it runs the risk of creating / exacerbating the vanishing/exploding gradient problem, as numeric instability can lead to very high or low gradients effecting our output.

II. RESULTS

Hand Picked Features Performance			
TARGET / OUTPUT	Positive	Negative	SUM
Positive	10927 36.42%	4041 13.47%	14968 73.00% 27.00%
Negative	3518 11.73%	11514 38.38%	15032 76.60% 23.40%
SUM	14445 75.65% 24.35%	15555 74.02% 25.98%	22441 / 30000 74.80% 25.20%

Figure 1: Confusion matrix for Hand Picked Features

For handpicked features, we had an average accuracy of around 74.80%. Figure 1 shows the Confusion Matrix for this run. We can see that the model can predict negative samples more accurately than positive samples (76.6% vs 73.0%).

Embeddings Feature Performance			
TARGET / OUTPUT	Positive	Negative	SUM
Positive	10955 36.52%	4013 13.38%	14968 73.19% 26.81%
Negative	3063 10.21%	11969 39.90%	15032 79.62% 20.38%
SUM	14018 78.15% 21.85%	15982 74.89% 25.11%	22924 / 30000 76.41% 23.59%

Figure 2: Confusion Matrix for Embeddings Feature

For the embeddings-based features, we had an average accuracy of around 76.41%. Figure 2 shows the Confusion Matrix for this run. We can see that the model also predicts negative samples more accurately than positive samples (76.6 %vs 73.0%).

III. Conclusion

Through the test results shown above a conclusion can be drawn on the effectiveness of the experiments attempt to analysis the sentiment of Amazon product reviews. To break it down further, we tested the text through a binary grouping. This means that either the text was classified as positive or negative sentiment. Then we ran this analysis with the text using two implementations, Hand – Picked Features and Embeddings Feature. Finally, we collected the data of both implementations using a confusion matrix to record the data, as shown above in the results section. Firstly, when looking at the Hand – Picked Features Performance we can use that to get the precession and the recall of the analysis. Through the data in the table, we get the true positives (10927) and divide that by the sum of the true positives and the false positives (10927 + 4041) which yields ~ 0.73 percent. Furthermore, the recall is calculated through dividing the true positives (10927) by the sum of the true positives and the false negatives (10927 + 3518). This yields a recall of ~ 0.76 percent. This means that our analysis through the Hand-Picked Features, has around 73% when correctly identifying a text as either positive or negative. Then our recall shows us that when we select the text as positive, we had a 76% rate of being correct in that analysis. This can be shown in our graphs as our total correct predictions out of all our data values is at 74% which is around both of our values.

Comparatively, our Embeddings Feature has some slightly different calculations. When looking at the precision results in a yield of ~ 0.73 percent. Furthermore, our recall results in a yield of ~ 0.78 percent. This means that both implementations have decently high succession rates but with some room for improvement. Overall, both implementations feature similar rates of both precision and recall; with the Embedding having a higher recall rate and the handpicked having a slightly higher persecion rate. Meaning, Handpicked might have a slightly higher chance of correctly placing a text into either positive or negative, but Embedding will have a lower chance of incorrectly picking a text as positive. Metrics like these

Positive – Negative tables are used to test the effectiveness of your models and to better understand the downfalls of your model.