Web Science

Sentiment Analysis and Crowd-sourcing

Bence Olay

March 2021

1 Introduction

Within the Web Science lecture, I had the opportunity to have a deeper understanding how the crowd-sourcing algorithms and recommender systems work. In this report, I will make some conclusions about two data-sets, particularly, the Stanford Sentiment Treebank Data-set (SST), and the Amazon Review Data (2018). The goal is to make a sentimental analysis on the SST, a crowd-sourcing done by the course-members and a recommender for the Amazon data-set.

2 Data-set analysis

2.1 Sentiment Analysis - SST Data-set

2.2 Data-set properties

Train set
Label 0 - Number of reviews: 34702; The average length: 53.96
Label 1 - Number of reviews: 42259; The average length: 50.72

Dev set
Label 0 - Number of reviews: 428; The average length: 101.96
Label 1 - Number of reviews: 444; The average length: 108.43

Test set
Label 0 - Number of reviews: 912; The average length: 101.00
Label 1 - Number of reviews: 909; The average length: 105.59

2.2.1 Data balance

These statistics are describes the Stanford data-set in which we have the following files: *stsa.binary.phrases.train, stsa.binary.dev* and *stsa.binary.test*. Regarding the labels, they encode the negative (0) and positive (1) review of different products. Their distribution in both the dev and test set are almost in perfect balance, but the train set has 1.22 times more of the positive reviews. This

might cause miss-leading assumptions and worst models, but with this ratio, the training set may represents the real-world situation better as people usually more often make sound to their disappointment then happiness.

Also the size of the training data is significantly bigger, compared to the other two, which resulted a reasonable shift in the average length.
As these algorithms rely on having a big data-set, I will continue by talking mainly about the training set.

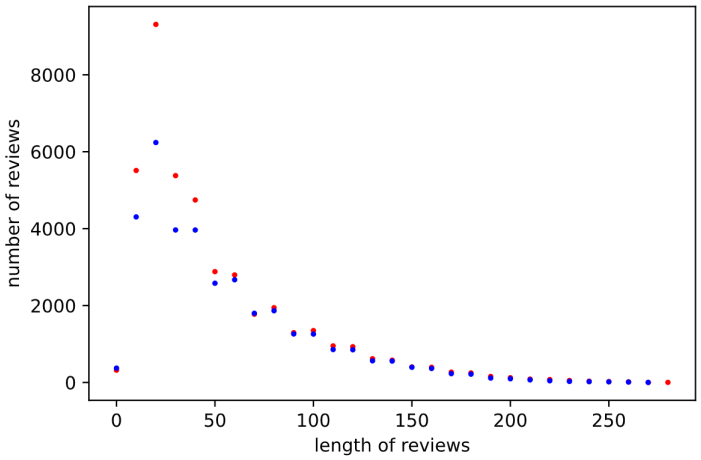

Figure 1: The length distribution of the reviews - Train set

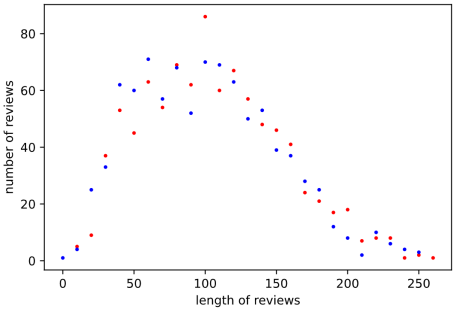

Figure 2: Test set

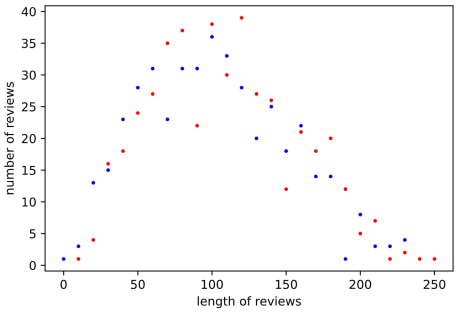

Figure 3: Dev set

2.2.2 Length correlation with labels

Length - Label: -0.037769

The length of the reviews not really changes bye the labels, which is also indicated by the coefficient with the almost 0 value.

2.2.3 Most common n-Grams

As I started the assignment by constructing my own class from the ground, I calculated the occurrence by going throw the review sentences manually, and so I got the following result. This works, but not how we would like it to do so, cause the conjunction words and phrases are the most common not surprisingly.

1-Gram	Label 0	the	,	a	of	and	to
		14357	12716	10715	8955	8783	7424
	Label 1	the	,	a	and	of	.
		16665	16380	13727	13154	11423	7086
	Overall	the	,	a	and	of	to
		31022	29096	24442	21937	20378	14360
2-Gram	Label 0	of the	, and	in the	it 's	, but	the film
		1564	1070	946	931	818	761
	Label 1	of the	, and	in the	it 's	is a	, but
		2232	1565	1052	981	967	926
	Overall	of the	, and	in the	it 's	, but	the film
		3796	2635	1998	1912	1744	1610
3-Gram	Label 0	the film 's	the movie is	it 's a	, it 's	one of the	it 's not
		199	125	124	114	113	102
	Label 1	one of the	it 's a	, but it	the film 's	, and the	the film is
		361	180	168	154	139	137
	Overall	one of the	the film 's	it 's a	, but it	, it 's	, and the
		474	353	304	250	225	224

Figure 4: The most frequently used words of the reviews - Test set

When I started using the

```
Whole database
       tfidf
the  2.210299
and  2.367068
of   2.490469
to   2.807791
is   3.125087
             tfidf
of the    4.053253
in the    4.658701
the film  4.880207
to the    5.155174
to be     5.196617
                tfidf
one of the    6.096209
the film is   6.918348
the kind of   7.009320
the movie is  7.086281
of the year   7.103573
```

```
Label 1 sentences
       tfidf
the  2.228244
and  2.294130
of   2.487614
to   2.936536
is   3.195215
             tfidf
of the    3.997627
in the    4.694099
the film  4.920578
to the    5.166961
and the   5.273170
                tfidf
one of the    5.771063
the film is   6.724343
of the year   6.807409
of the most   6.906664
of the best   6.933097
```

```
Label 0 sentences
       tfidf
the  2.188806
and  2.463642
of   2.493857
to   2.670419
is   3.045642
             tfidf
of the    4.124832
in the    4.616176
the film  4.831845
to be     5.060991
the movie 5.115987
                tfidf
the movie is  6.610394
one of the    6.718383
of the film   6.900705
the kind of   7.097873
of its own    7.137093
```

Figure 5: 1,2,3-Grams for the whole corpus and by each label

3 Crowd-sourcing Exercise

As part of the course, we made a sentiment scoring on a small portion of the SST2 set, where 30 of us rated 101 sentences on a 5 scale rate. As a ground truth we had their actual sentiment, whether the item-review was positive or negative.

3.1 Relabel the SST Data-set with Crowd-sourcing

First of all, the data needs to be formed, as so the rows are the item ratings, and each column is responsible for each person.
At the first time, I approached this problem by implementing the algorithm, where the program calculates the mean of the ratings in each iteration, and then the trust of the individuals. The trust equals to the number of rates on the same half ([0,3[or [3,5]) of the scale as the mean (here we can try if we have a ground truth which might give us better trust scores, but in our case I got a better result by using the means), which were divided by the number of reviews to normalize.
I ended up working on this solution, as around this time I realized it will be much easier to use the packages. Although, I made the crowd rates to converge towards one of the end of the scale and if they reached the upper or lower boundary, they were converged and so got the value of that side. This solution would worked better with a higher *Krippendorff* coefficient, but it's not guaranteed to converge and also not optimized as runs slow loops in Python compared to the packages implemented mostly in C/C++.

3.1.1 Krippendorff coefficient

After that, I made the *CrowdSourcer* class, in which I calculated the *Krippendorff* coefficient that gave 0.005. This value means the that our crowd has many disagreements. On the other hand, as we will reduce the scale to only 2 options, this value is not really represents what our crowd will perform indeed.

3.1.2 Voting Classifier

I used the *SKlearn* package throw-out the whole project.
A *Voting Classifier* is a machine learning model that trains on an ensemble of numerous models and predicts an output based (class) on their highest probability of chosen class as the output. The *VotingClassifier* class supports two types of voting

- **Hard Voting:** In hard voting, the predicted output class is a class with the highest majority of votes.

- **Soft Voting:** In soft voting, the output class is the prediction based on the average of probability given to that class.

This class is using an estimator list to process the training data, for which we need to use logistic regression where we need to set the multi-class property and we can choose which solver it is using, so I tested them.
Of the bat, the *liblinear* doesn't work with multi-class. The *saga* and *sag* was barely converged, and between the *newton-cg* and the *lbfgs* wasn't that much of a difference, so I decided to stick with the default one.
I also tried out each of the voting calculations, where the difference was much visible, as the soft voting technique is much more unstable than the hard one.
Also, in order to have some measurable results I needed to significantly reduce the train-set, which were only 9 instance out of the 101 with 92 test items.

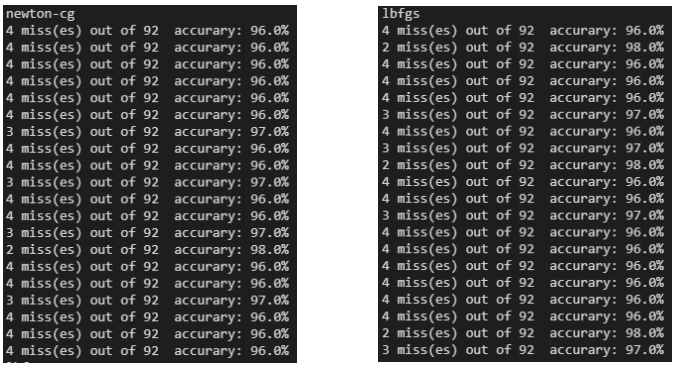

Figure 6: Measurements of the 2 logical regression algorithms

4 Sentiment Analysis

In order to analyze a big data-set consisting of sentences, I first used the *sklearn.feature_extraction.text.CountVectorizer* to create a vocabulary from the data. This contains all the occurred words with which, these sentences can be transformed into an array of integers from the vocabulary by their indexes. For this purpose I used *sklearn.feature_extraction.text.TfidfTransformer* which made a sparse matrix, therefore I could not use the *sklearn.decomposition.PCA* algorithm as it requires a dense matrix.
I found the *sklearn.decomposition.TruncatedSVD* as a suitable solution, and as we have a rather big source we would lose a lot of information with the default 2 component, so I made a 100 dimensional singular value decomposition. I made a training score comparison between the different vocabularies of 1 and 3-grams to see how much better will be a bigger, but therefore, more time-consuming vocabulary. For this, I used the *sklearn.linear_model.SGDClassifier* classifier.

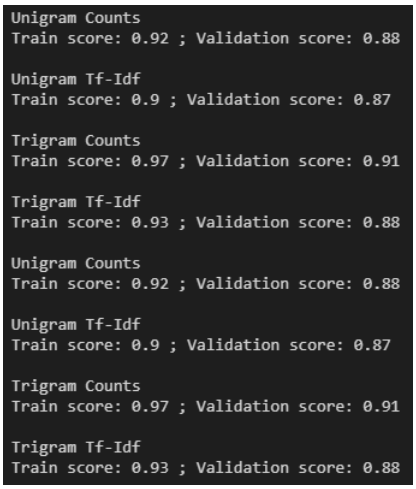

Figure 7: Training with different 1,3-grams

I could not make further progress in this field as I had many tries with the usage of the *sklearn.pipeline.Pipeline* and with the different compositions of the transformation workflows of the data.

I also tried out some of the models from the *Gensim* module, but as it is very time-consuming particularly as my laptop is not built for these purposes and also the lake of time was a big problem at this part of the assignment.

5 Conclusion

In this report I have presented my gained knowledge of the given data-sets and elaborated on the solutions and algorithms that I used. The collaborative work showed me an example of the crowd-sourcing algorithms' strength and the seek of information that is in this subject.