Sentiment analysis on Amazon user reviews

Federico Pappani - 298223

August 2021

Project for Course of Big Data and Business Intelligence

Bachelor's Degree in Computer, Electronic and Communications Engineering
University of Parma
github.com/pappani/AmazonSentimentAnalysis

Project Goal

Analysis of Amazon user reviews through different machine learning algorithms, to then create a model for sentiment analysis prediction to be tested on arbitrary strings.

State-of-the-art

Currently the best implementations of natural language processing and sentiment analysis have achieved an accuracy of 95% or more. Research scientist Sebastian Ruder has made a great repository to track the progress of NLP; his work can be found here.
There are also commercial and open source voice recognition systems, (like Amazon Alexa, Siri or Mycroft) with a high degree of accuracy, which probably integrate sentiment analysis systems to some extent.

Tools and Dataset used

Python was used to carry out the analysis, together with the Jupyter Notebook software, and the Scikit Learn library for the Machine Learning part.
A dataset consisting of reviews of electronic devices was used for the analysis, the dataset is made available by Julian McAuley of the University of California San Diego, and can be found here.

Dataset processing

Reviews inside the dataset are saved in JSON format:

```
{
"reviewerID": "A11AA01YRZT8DP",
"asin": "B004LGNB0A",
"reviewerName": "NP",
"helpful": [0, 0],
"reviewText": "Does the job well.",
"overall": 5.0,
"summary": "Five Stars",
"unixReviewTime": 1404345600,
"reviewTime": "07 3, 2014"
}
```

where:
”reviewerID” is the ID of the reviewer,
”asin” is the ID of the product,
”reviewerName” is the name of the reviewer,
”helpful” is the helpfulness rating of the review,
”reviewText” is the text of the review,
”overall” is the overall rating of the product,
”summary” is the summary title of the review,
”unixReviewTime” is the UNIX time of the review,
and lastly, ”reviewTime” is the time of the review.

The original dataset contains more than one and a half million reviews; through a simple Python script a limited number of reviews were extracted, to make the analysis feasible on an home computer.
Removing outliers was not necessary as the data was already ”clean” from the source, but the only data used in processing is the ”reviewText” and ”overall” fields, the other fields are ignored.

Training

A random sample of 75,000 reviews was taken from the original dataset. The sample was then split into two parts, one for training and one for testing, respectively 67% and 33% of the initial sample.
To exclude any bias related to dataset asymmetries, the data was processed with a special function, to obtain an equal number of negative and positive reviews.
The text was then vectorized using the scikit-learn CountVectorizer function, obtaining the so called bags-of-words.
The vectors obtained were then classified using 4 different classification algorithms: Linear SVM Classifier, Decision Tree Classifier, Gaussian Naïve Bayes Classifier, and Logistic Regression Classifier.

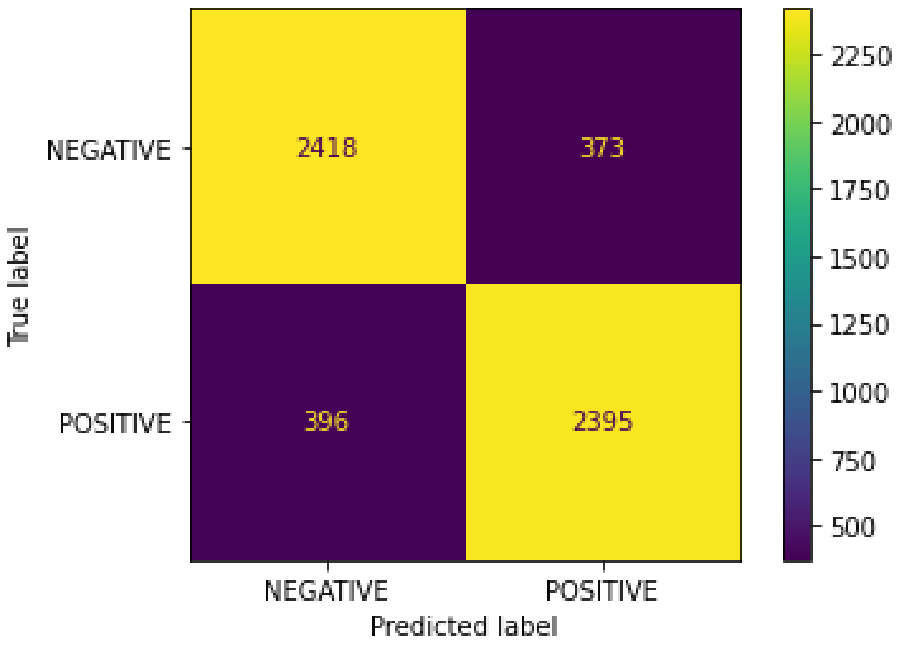
Results

The testing of the 4 obtained models gave the following accuracy and F_1 score results:

- Linear SVM: 86.2%, (86.1% positive, 86.2% negative)
- Decision Tree: 68.8%, (69.2% positive, 68.4% negative)
- Gaussian Naïve Bayes: 69.6%, (69.9% positive, 69.3% negative)
- Logistic Regression: 86.0%, (86.0% positive, 86.1% negative)

The best performing classifier was the Linear SVM, which was then used for the subsequent analysis and predictions.

Confusion matrix obtained from the Linear SVM classification:



The model was then tested with some arbitrary text strings:

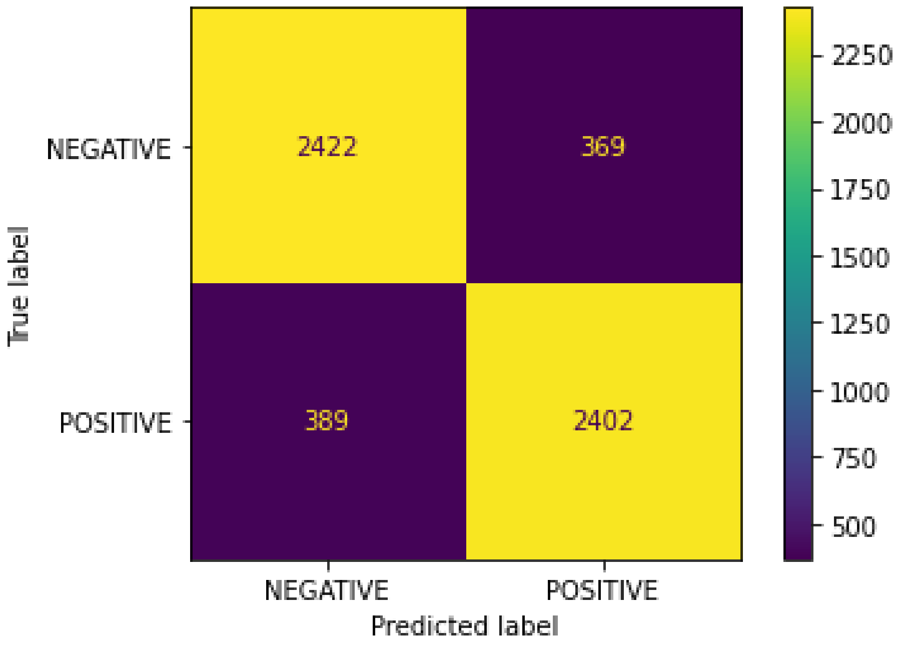
String	Expected Result	Model Result
”nice product!”	Positive	Positive
”incredibly good device”	Positive	Positive
”not good at all”	Negative	Negative
”it broke after 3 days”	Negative	Negative
”it doesn’t work properly”	Negative	Negative
”it’s amazing”	Positive	Positive
”I had to return it”	Negative	Negative
”it feels old”	Negative	Negative
”it fell apart quickly”	Negative	Negative
”I don’t know why Amazon still sells this”	Negative	Negative
”it went straight to the trash”	Negative	Negative
”it’s garbage”	Negative	Negative
”it does what it’s meant to do”	Positive	Positive
”it works fine”	Positive	Positive
”recommended”	Positive	Positive
”you should buy it now”	Positive	Negative
”it looks good but it does not work as intended”	Negative	Negative
”today is a sunny day”	Positive	Negative
”good morning my dear”	Positive	Positive
”traffic around here has been quite noisy in the past few days”	Negative	Negative
”I crashed my car into a tree”	Negative	Negative
”today is a rainy day”	Negative	Negative

The results are positive, even if two strings were incorrectly classified by the model.

Model Tuning

An attempt at model tuning using grid search was tried.
Model accuracy increased to 86.4%, but testing with arbitrary strings produced the same results.

New confusion matrix obtained after model tuning:



Model Saving

The obtained model was then saved through the Pickle function, in order to use it in the future without having to re-train.

Source Code

Dataset reduction:

```
import json
import random
data = []
file_name = 'Electronics_5'
with open(f'./{file_name}.json') as f:
        for line in f:
                review = json.loads(line)
                data.append(review)

final_data = random.sample(data, 75000)   # num. of entries
print(len(final_data))
with open(f'./{file_name}_small.json', 'w') as f:
        for review in final_data:
                f.write(json.dumps(review)+'\n')
```

Data classes:

```
import random

class Sentiment:
    NEGATIVE = "NEGATIVE"
    POSITIVE = "POSITIVE"

class Review:
    def __init__(self, text, score):
        self.text = text
        self.score = score
        self.sentiment = self.get_sentiment()

    def get_sentiment(self):
        if self.score <= 2:
            return Sentiment.NEGATIVE
        elif self.score >= 4:
            return Sentiment.POSITIVE
        # let's ignore the reviews with 3 stars

class ReviewContainer:
    def __init__(self, reviews):
        self.reviews = reviews

    def get_text(self):
        return [x.text for x in self.reviews]
    def get_sentiment(self):
        return [x.sentiment for x in self.reviews]

    def evenly_distribute(self):
        negative = list(filter(lambda x: x.sentiment == Sentiment.NEGATIVE, self.reviews))
        positive = list(filter(lambda x: x.sentiment == Sentiment.POSITIVE, self.reviews))
        positive_shrunk = positive[:len(negative)]
        self.reviews = negative + positive_shrunk
        random.shuffle(self.reviews)
```

Data loading:

```
import json
file_name = 'C:/Electronics_5_small.json'    # 75k entries
reviews = []
with open(file_name) as f:
    for line in f:
        review = json.loads(line)
        reviews.append(Review(review['reviewText'], review['overall']))

reviews[0].text
```

Data preparation/splitting:

```
from sklearn.model_selection import train_test_split
training, test = train_test_split(reviews, test_size=0.33, random_state=42)
# let's use 42 since it is the answer to everything
train_container = ReviewContainer(training)
test_container = ReviewContainer(test)
train_container.evenly_distribute()
train_x = train_container.get_text()
train_y = train_container.get_sentiment()
test_container.evenly_distribute()
test_x = test_container.get_text()
test_y = test_container.get_sentiment()
print(train_y.count(Sentiment.POSITIVE))
print(train_y.count(Sentiment.NEGATIVE))
```

Vectorization:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
vectorizer = TfidfVectorizer()
train_x_vectors = vectorizer.fit_transform(train_x)
test_x_vectors = vectorizer.transform(test_x)
```

```python
print(train_x[0])
print(train_x_vectors[0].toarray())
```

Classification:

Linear SVM

```python
from sklearn import svm
clf_svm = svm.SVC(kernel='linear')
clf_svm.fit(train_x_vectors, train_y)
test_x[0]
clf_svm.predict(test_x_vectors[0])
```

Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
clf_dec = DecisionTreeClassifier()
clf_dec.fit(train_x_vectors, train_y)
clf_dec.predict(test_x_vectors[0])
```

Gaussian Naïve Bayes

```python
from sklearn.naive_bayes import GaussianNB
clf_gnb = DecisionTreeClassifier()
clf_gnb.fit(train_x_vectors, train_y)
clf_gnb.predict(test_x_vectors[0])
```

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
clf_log = LogisticRegression()
clf_log.fit(train_x_vectors, train_y)
clf_log.predict(test_x_vectors[0])
```

Results Evaluation:

```python
# Mean Accuracy
print(clf_svm.score(test_x_vectors, test_y))
print(clf_dec.score(test_x_vectors, test_y))
print(clf_gnb.score(test_x_vectors, test_y))
print(clf_log.score(test_x_vectors, test_y))
# F1 Scores
from sklearn.metrics import f1_score
f1_score(test_y, clf_svm.predict(test_x_vectors), average=None,
labels=[Sentiment.POSITIVE, Sentiment.NEGATIVE])
f1_score(test_y, clf_dec.predict(test_x_vectors), average=None,
labels=[Sentiment.POSITIVE, Sentiment.NEGATIVE])
```

```
f1_score(test_y, clf_gnb.predict(test_x_vectors), average=None, l
abels=[Sentiment.POSITIVE, Sentiment.NEGATIVE])
f1_score(test_y, clf_log.predict(test_x_vectors), average=None,
labels=[Sentiment.POSITIVE, Sentiment.NEGATIVE])
# Arbitrary strings
test_set = ["nice product!", "incredibly good device", "not good at all",
"it broke after 3 days", "it doesn't work properly", "it's amazing", "I had to return it",
"it feels old", "it fell apart quickly", "I don't know why Amazon still sells this",
"it went straight to the trash", "it's garbage",
"it does what it's meant to do", "it works fine", "recommended",
"you should buy it now", "it looks good but it does not work as intended",
"today is a sunny day", "good morning my dear",
"traffic around here has been quite noisy in the past few days",
"I crashed my car into a tree", "today is a rainy day"]
new_test = vectorizer.transform(test_set)
clf_svm.predict(new_test)
# Confusion matrix
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
plot_confusion_matrix(clf_svm, test_x_vectors, test_y)
plt.show()
# Model tuning
from sklearn.model_selection import GridSearchCV
parameters = {'kernel': ('linear', 'rbf'), 'C': (1,4,8,16,32)}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters, cv=5)
clf.fit(train_x_vectors, train_y)
print(clf.score(test_x_vectors, test_y))
```

Model saving:

```
import pickle
with open('./models/sentiment_classifier.pkl', 'wb') as f:
    pickle.dump(clf_svm, f)
```

Conclusion

The model performed quite well, achieving 86% accuracy.
The analysis could be improved by using a larger dataset or the entire original dataset, but the compute resources available in this specific case were not sufficient for that.

Notes and Bibliography

Analysis performed on a computer equipped with an Intel Core i5 6400 CPU and 16 GB of RAM, a discrete graphics card was also available but was not used for ”number crunching”.

- Python3
- scikit-learn
- SVM
- Decision Tree
- Gaussian Naïve Bayes
- Logistic Regression
- Amazon Reviews Dataset