Sentiment Analysis with Python (Part 1)

Classifying IMDb Movie Reviews

Aaron Kub Follow
Jul 31, 2018 · 6 min read

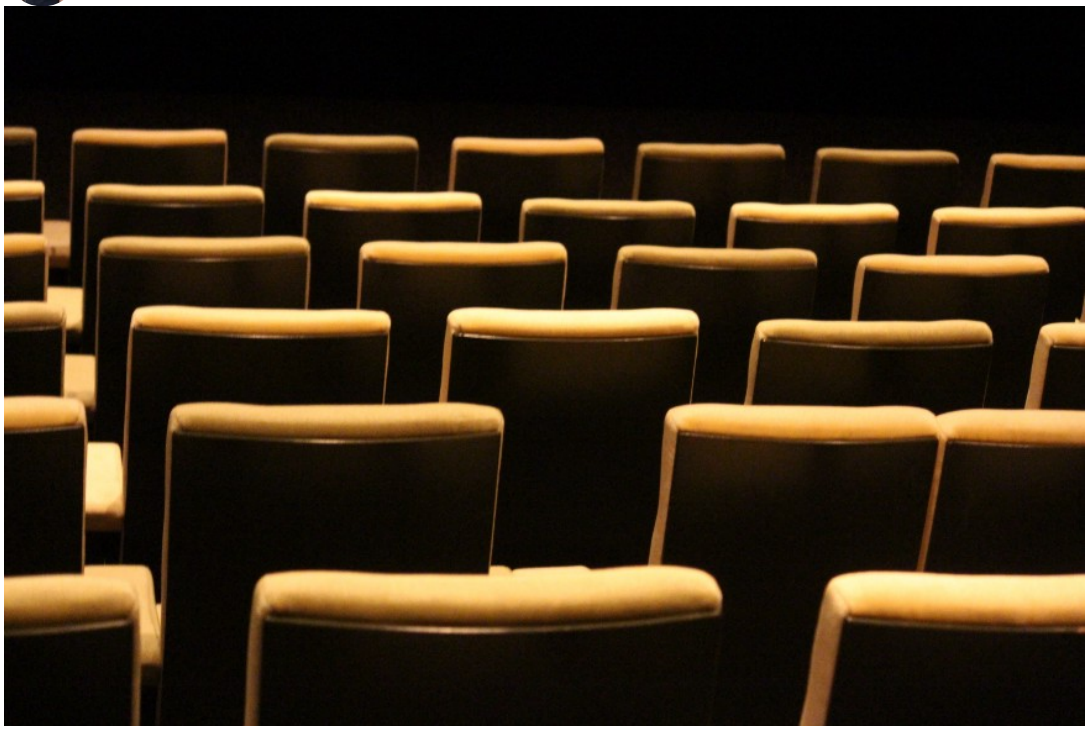

Photo by Denise Jans on Unsplash

Sentiment Analysis is a common NLP task that Data Scientists need to perform. This is a straightforward guide to creating a barebones movie review classifier in Python. Future parts of this series will focus on improving the classifier.

Data Overview

For this analysis we'll be using a dataset of 50,000 movie reviews taken from IMDb. The data was compiled by Andrew Maas and can be found here: IMDb Reviews.
The data is split evenly with 25k reviews intended for training and 25k for testing your classifier. Moreover, each set has 12.5k positive and 12.5k negative reviews.
IMDb lets users rate movies on a scale from 1 to 10. To label these reviews the curator of the data labeled anything with ≤ 4 stars as negative and anything with ≥ 7 stars as positive. Reviews with 5 or 6 stars were left out.

. . .

Step 1: Download and Combine Movie Reviews

If you haven't yet, go to IMDb Reviews and click on "Large Movie Review Dataset v1.0". Once that is complete you'll have a file called `aclImdb_v1.tar.gz` in your downloads folder.
Shortcut: If you want to get straight to the data analysis and/or aren't super comfortable with the terminal, I've put a tar file of the final directory that this step creates here: Merged Movie Data. Double clicking this file should be sufficient to unpack it (at least on a Mac), otherwise `gunzip -c movie_data.tar.gz | tar xopf –` in a terminal will do it.

Unpacking and Merging

Follow these steps or run the shell script here: Preprocessing Script

1. Move the tar file to the directory where you want this data to be stored.
2. Open a terminal window and `cd` to the directory that you put `aclImdb_v1.tar.gz` in.
3. `gunzip -c aclImdb_v1.tar.gz | tar xopf -`
4. `cd aclImdb && mkdir movie_data`
5. `for split in train test; do for sentiment in pos neg; do for file in $split/$sentiment/*; do cat $file >> movie_data/full_${split}.txt; echo >> movie_data/full_${split}.txt; done; done; done;`

. . .

Step 2: Read into Python

For most of what we want to do in this walkthrough we'll only need our reviews to be in a Python `list`. Make sure to point `open` to the directory where you put the movie data.

```
1  reviews_train = []
2  for line in open('../data/movie_data/full_train.txt', 'r'):
3      reviews_train.append(line.strip())
4
```

. . .

Step 3: Clean and Preprocess

The raw text is pretty messy for these reviews so before we can do any analytics we need to clean things up. Here's one example:

```
"This isn't the comedic Robin Williams, nor is it the quirky/insane
Robin Williams of recent thriller fame. This is a hybrid of the
classic drama without over-dramatization, mixed with Robin's new love
of the thriller. But this isn't a thriller, per se. This is more a
mystery/suspense vehicle through which Williams attempts to locate a
sick boy and his keeper.<br /><br />Also starring Sandra Oh and Rory
Culkin, this Suspense Drama plays pretty much like a news report,
until William's character gets close to achieving his goal.<br /><br
/>I must say that I was highly entertained, though this movie fails
to teach, guide, inspect, or amuse. It felt more like I was watching
a guy (Williams), as he was actually performing the actions, from a
third person perspective. In other words, it felt real, and I was
able to subscribe to the premise of the story.<br /><br />All in all,
it's worth a watch, though it's definitely not Friday/Saturday night
fare.<br /><br />It rates a 7.7/10 from...<br /><br />the Fiend :."
```

Note: Understanding and being able to use regular expressions is a prerequisite for doing any Natural Language Processing task. If you're unfamiliar with them perhaps start here: Regex Tutorial

```python
import re

REPLACE_NO_SPACE = re.compile("[.;:!\'?,\"()\[\]]")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")

def preprocess_reviews(reviews):
    reviews = [REPLACE_NO_SPACE.sub("", line.lower()) for line in reviews]
    reviews = [REPLACE_WITH_SPACE.sub(" ", line) for line in reviews]

    return reviews

reviews_train_clean = preprocess_reviews(reviews_train)
reviews_test_clean = preprocess_reviews(reviews_test)
```

clean_movie_reviews.py hosted with ♡ by GitHub view raw

```
sick boy and his keeper also starring sandra oh and rory culkin this
suspense drama plays pretty much like a news report until williams
character gets close to achieving his goal i must say that i was
highly entertained though this movie fails to teach guide inspect or
amuse it felt more like i was watching a guy williams as he was
actually performing the actions from a third person perspective in
other words it felt real and i was able to subscribe to the premise
of the story all in all its worth a watch though its definitely not
friday saturday night fare it rates a   from the fiend"
```

Note: There are a lot of different and more sophisticated ways to clean text data that would likely produce better results than what I've done here. I wanted part 1 of this tutorial to be as simple as possible. Also, I generally think it's best to get baseline predictions with the simplest possible solution before spending time doing potentially unnecessary transformations.

Vectorization

In order for this data to make sense to our machine learning algorithm we'll need to convert each review to a numeric representation, which we call *vectorization*.
The simplest form of this is to create one very large matrix with one column for every unique word in your corpus (where the corpus is all 50k reviews in our case). Then we transform each review into one row containing 0s and 1s, where 1 means that the word in the *corpus* corresponding to that column appears in that *review*. That being said, each row of the matrix will be very sparse (mostly zeros). This process is also known as *one hot encoding*.

```
1  from sklearn.feature_extraction.text import CountVectorizer
2
3  cv = CountVectorizer(binary=True)
4  cv.fit(reviews_train_clean)
5  X = cv.transform(reviews_train_clean)
6  X_test = cv.transform(reviews_test_clean)
```

vectorize_reviews.py hosted with ♡ by GitHub view raw

. . .

Step 4: Build Classifier

Now that we've transformed our dataset into a format suitable for modeling we can start building a classifier. Logistic Regression is a good baseline model for us to use for several reasons: (1) They're

structured the same, where the first 12.5k are positive and the last 12.5k are negative.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

target = [1 if i < 12500 else 0 for i in range(25000)]

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75
)

for c in [0.01, 0.05, 0.25, 0.5, 1]:

    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))

#     Accuracy for C=0.01: 0.87472
#     Accuracy for C=0.05: 0.88368
#     Accuracy for C=0.25: 0.88016
#     Accuracy for C=0.5: 0.87808
#     Accuracy for C=1: 0.87648
```

LR_review_classifier_part1.py hosted with ♡ by GitHub view raw

It looks like the value of C that gives us the highest accuracy is `0.05`.

. . .

Train Final Model

Now that we've found the optimal value for C, we should train a model using the entire training set and evaluate our accuracy on the 25k test reviews.

```
final_model = LogisticRegression(C=0.05)
final_model.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final_model.predict(X_test)))
# Final Accuracy: 0.88128
```

```
        cv.get_feature_names(), final_model.coef_[0]
    )
}
for best_positive in sorted(
    feature_to_coef.items(),
    key=lambda x: x[1],
    reverse=True)[:5]:
    print (best_positive)

#     ('excellent', 0.9288812418118644)
#     ('perfect', 0.7934641227980576)
#     ('great', 0.675040909917553)
#     ('amazing', 0.6160398142631545)
#     ('superb', 0.6063967799425831)

for best_negative in sorted(
    feature_to_coef.items(),
    key=lambda x: x[1])[:5]:
    print (best_negative)

#     ('worst', -1.367978497228895)
#     ('waste', -1.1684451288279047)
#     ('awful', -1.0277001734353677)
#     ('poorly', -0.8748317895742782)
#     ('boring', -0.8587249740682945)
```

best_predictors_lr_part1.py hosted with ♡ by GitHub view raw

And there it is. A very simple classifier with pretty decent accuracy out of the box.

. . .

Next Time

In the next parts of this series we'll look at more sophisticated ways to get better performance out of our classifier.

- **Text Processing**: Stemming/Lemmatizing to convert different forms of each word into one.
- **n-grams**: Instead of just single-word tokens (1-gram/unigram) we can also include word pairs.
- **Representations**: Instead of simple, binary vectors we can use word counts or *TF-IDF* to transform those counts.

Improving a Movie Review Sentiment Classifier

towardsdatascience.com

. . .

Thanks for reading! This is my first post on Medium so please comment with any questions or suggestions you have.

Machine Learning Sentiment Analysis Data Science Python NLP

1.7K claps

WRITTEN BY

Aaron Kub

Follow

Data Scientist at IBM Watson Health. Opinions are my own.

Towards Data Science

Following

Sharing concepts, ideas, and codes.

See responses (21)

More from Towards Data Science

12 Things I Learned During My First Year as a Machine Learning Engineer

Daniel Bourke in Towar...
Jul 6 · 11 min read ★

5.4K

More from Towards Data Science

Hands On Bayesian Statistics with Python, PyMC3 & ArviZ

Susan Li in Towards Dat...
Jul 17 · 10 min read

586

More from Towards Data Science

P-values Explained By Data Scientist

Admond Lee in Towards...
Jul 13 · 11 min read ★

542