The past fifteen years have seen extensive investments in business infrastructure, which have improved the ability to collect data throughout the enterprise. Virtually every aspect of business is now open to data collection and often even instrumented for data collection: operations, manufacturing, supply-chain management, customer behavior, marketing campaign performance, workflow procedures, and so on. This broad availability of data has led to increasing interest in methods for extracting useful information and knowledge from data—the realm of data science.

This realm is huge. To understand the whole picture, we have to go through the individual building blocks; specific algorithms being used for specific tasks. Let’s consider the case of retail industry. Some of the use cases exploiting Data Science tools are:

* Recommendation engines: compute a similarity index in the customers’ preferences and offer the goods or services accordingly.
* Fraud detection: continuous monitoring of the activity and ensure the detection of the fraudulent activity.
* Customer sentiment analysis: analysts can perform the brand-customer sentiment analysis by data received from social networks and online services feedbacks.

There are hundreds of other examples. Consider the fellows at Gradient, who showcased at CES 2019, a technology allows companies to offer an extra level of security in online processes.

Let us take a small piece of the pie and understand it on a deeper level: consider a case study on Sentiment Analysis.

## Data

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](http://ai.stanford.edu/~amaas/data/sentiment/).

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 us 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.

First import the essentials:

*from sklearn.feature\_extraction.text import TfidfVectorizer*

*from sklearn.metrics import accuracy\_score*

*import numpy as np*

*import pandas as pd*

*import os*

*import re*

Then, read the training and validation files:

*reviews\_train = []*

*for line in open('C:/Users/SOUMYA/Downloads/movie\_data.tar/movie\_data/movie\_data/full\_train.txt', encoding="utf8"):*

*reviews\_train.append(line.strip())*

*reviews\_test = []*

*for line in open('C:/Users/SOUMYA/Downloads/movie\_data.tar/movie\_data/movie\_data/full\_test.txt', encoding="utf8"):*

*reviews\_test.append(line.strip())*

*reviews\_train[5]*

"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 :."

Ok. So we have successfully imported the required files. Now, as you can imagine, special characters (like ‘,’ ‘-‘ ‘,’ ‘/’ etc) and numbers are not essential for a sentiment analysis. Hence, we write a function to eliminate these characters using regular expression (re) :

*REPLACE\_NO\_SPACE = re.compile("(\.)|(\;)|(\:)|(\!)|(\')|(\?)|(\,)|(\")|(\()|(\))|(\[)|(\])|(\d+)")*

*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)*

*reviews\_train\_clean[5]*

'this isnt 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 robins new love of the thriller but this isnt a thriller per se this is more a mystery suspense vehicle through which williams attempts to locate a 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 '

*Food for thought: Why did we use two variables REPLACE\_NO\_SPACE and REPLACE\_WITH\_SPACE?*

## 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 way to implement this is by *one hot encoding:*

*from sklearn.feature\_extraction.text import CountVectorizer*

*cv = CountVectorizer(binary=True)*

*cv.fit(reviews\_train\_clean)*

*X = cv.transform(reviews\_train\_clean)*

*X\_test = cv.transform(reviews\_test\_clean)*

## Building Classifier

Now, we build the **classifier**. We can choose from numerous algorithms but I’m going to go with **Logistic Regression** as our baseline model because they’re easy to interpret, fast and its performance on sparse datasets like ours.

**Note**: The targets/labels we use will be the same for training and testing because both datasets are structured the same, where the first 12.5k are positive and the last 12.5k are negative.

Also I should point out that there are several hyper-parameters associated with all regression algorithms but I’m only going to concentrate on *c* (which adjusts regularization) to keep things simple.

*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: 0.87744

Accuracy for C=0.05: 0.88256

Accuracy for C=0.25: 0.88144

Accuracy for C=0.5: 0.87984

Accuracy for C=1: 0.87792

We get highest accuracy with *c=0.05*. If we were to stop here, we would select our final model with *c=0.05*.

## Removing Stop Words

To go one step further, we will try and **remove any stop words** from our input text.

Stop words are the very common words like ‘if’, ‘but’, ‘we’, ‘he’, ‘she’, and ‘they’. We can usually remove these words without changing the semantics of a text and doing so often (but not always) improves the performance of a model. Removing these stop words becomes a lot more useful when we start using longer word sequences as model features (see n-grams below).

This can be done by a couple of ways:

1. using stop words from nltk and making an udf to remove words
2. using manual stop words and making an udf to remove words (we will see this in our final model)
3. using countVectorizer

*from nltk.corpus import stopwords*

*english\_stop\_words = stopwords.words('english')*

*def remove\_stop\_words(corpus):*

*removed\_stop\_words = []*

*for review in corpus:*

*removed\_stop\_words.append(*

*' '.join([word for word in review.split()*

*if word not in english\_stop\_words])*

*)*

*return removed\_stop\_words*

*no\_stop\_words = remove\_stop\_words(reviews\_train\_clean)*

Before:

"bromwell high is a cartoon comedy it ran at the same time as some other programs about school life such as teachers my years in the teaching profession lead me to believe that bromwell high’s satire is much closer to reality than is teachers the scramble to survive financially the insightful students who can see right through their pathetic teachers’ pomp the pettiness of the whole situation all remind me of the schools i knew and their students when i saw the episode in which a student repeatedly tried to burn down the school i immediately recalled at high a classic line inspector i’m here to sack one of your teachers student welcome to bromwell high i expect that many adults of my age think that bromwell high is far fetched what a pity that it isn’t"

After:

"bromwell high cartoon comedy ran time programs school life teachers years teaching profession lead believe bromwell high's satire much closer reality teachers scramble survive financially insightful students see right pathetic teachers' pomp pettiness whole situation remind schools knew students saw episode student repeatedly tried burn school immediately recalled high classic line inspector i'm sack one teachers student welcome bromwell high expect many adults age think bromwell high far fetched pity"

## Normalization

Next up is text **Normalization** in which we try to convert all of the different forms of a given word into one. Most common 2 methods are *Stemming* and *Lemmatization*.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

However, the two words differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

For our tutorial here, I’m going with *Lemmatization.*

*def get\_lemmatized\_text(corpus):*

*from nltk.stem import WordNetLemmatizer*

*lemmatizer = WordNetLemmatizer()*

*return [' '.join([lemmatizer.lemmatize(word) for word in review.split()]) for review in corpus]*

*lemmatized\_reviews\_train = get\_lemmatized\_text(reviews\_train\_clean)*

*lemmatized\_reviews\_test = get\_lemmatized\_text(reviews\_test\_clean)*

*print('before lemmatization\n------------------------\n',reviews\_train\_clean[5],'\n\nafter lemmatization\n------------------------\n',lemmatized\_reviews\_train[5])*

before lemmatization

------------------------

this isnt 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 robins new love of the thriller but this isnt a thriller per se this is more a mystery suspense vehicle through which williams attempts to locate a 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

after lemmatization

------------------------

this isnt 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 new love of the thriller but this isnt a thriller per se this is more a mystery suspense vehicle through which williams attempt to locate a sick boy and his keeper also starring sandra oh and rory culkin this suspense drama play pretty much like a news report until williams character get close to achieving his goal i must say that i wa highly entertained though this movie fails to teach guide inspect or amuse it felt more like i wa watching a guy williams a he wa actually performing the action from a third person perspective in other word it felt real and i wa able to subscribe to the premise of the story all in all it worth a watch though it definitely not friday saturday night fare it rate a from the fiend

## n-grams

Consider **n-grams**. Up until now, we used only single word features in our model, which we call 1-grams or unigrams. We can potentially add more predictive power to our model by adding two or three word sequences (**bigrams** or **trigrams**) as well. For example, if a review had the three word sequence “didn’t love movie” we would only consider these words individually with a unigram-only model and probably not capture that this is actually a *negative* sentiment because the word ‘love’ by itself is going to be highly correlated with a positive review.

*cv = CountVectorizer(binary=True, ngram\_range=(1, 2))*

*cv.fit(lemmatized\_reviews\_train)*

*X = cv.transform(lemmatized\_reviews\_train)*

*X\_test = cv.transform(lemmatized\_reviews\_test)*

*X\_train, X\_val, y\_train, y\_val = train\_test\_split(*

*X, target, train\_size = 0.75*

*)*

*for c in [0.01, 0.05,0.06,0.07,0.08, 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: 0.88384

Accuracy for C=0.05: 0.89184

Accuracy for C=0.06: 0.89248

Accuracy for C=0.07: 0.89296

Accuracy for C=0.08: 0.8928

Accuracy for C=0.25: 0.8944

Accuracy for C=0.5: 0.89424

Accuracy for C=1: 0.89376

**Note**: There’s technically no limit on the size that n can be for your model, but there are several things to consider. First, increasing the number of grams will not necessarily give you better performance. Second, the size of your matrix grows exponentially as you increment n, so if you have a large corpus that is comprised of large documents your model may take a very long time to train.

## Algorithms

As I said earlier, there are a variety of algorithms to choose from. So far we’ve chosen to represent each review as a very sparse vector (lots of zeros!) with a slot for every unique n-gram in the corpus (minus n-grams that appear too often or not often enough). Linear classifiers typically perform better than other algorithms on data that is represented in this way. So far we’ve chosen to represent each review as a very sparse vector (lots of zeros!) with a slot for every unique n-gram in the corpus (minus n-grams that appear too often or not often enough). Linear classifiers typically perform better than other algorithms on data that is represented in this way.

Another algorithm that can produce great results with a quick training time are Support Vector Machines with a linear kernel. Here is an example:

*from sklearn.svm import LinearSVC*

*ngram\_vectorizer = CountVectorizer(binary=True, ngram\_range=(1, 2))*

*ngram\_vectorizer.fit(reviews\_train\_clean)*

*X = ngram\_vectorizer.transform(reviews\_train\_clean)*

*X\_test = ngram\_vectorizer.transform(reviews\_test\_clean)*

*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]:*

*svm = LinearSVC(C=c)*

*svm.fit(X\_train, y\_train)*

*print ("Accuracy for C=%s: %s"*

*% (c, accuracy\_score(y\_val, svm.predict(X\_val))))*

Accuracy for C=0.01: 0.89136

Accuracy for C=0.05: 0.89104

Accuracy for C=0.25: 0.88816

Accuracy for C=0.5: 0.8872

Accuracy for C=1: 0.88704

## Final Model

Finally, let’s make our **final model**:

*from sklearn.feature\_extraction.text import CountVectorizer*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import accuracy\_score*

*from sklearn.svm import LinearSVC*

*stop\_words = ['in', 'of', 'at', 'a', 'the']*

*ngram\_vectorizer = CountVectorizer(binary=True, ngram\_range=(1, 3), stop\_words=stop\_words)*

*ngram\_vectorizer.fit(reviews\_train\_clean)*

*X = ngram\_vectorizer.transform(reviews\_train\_clean)*

*X\_test = ngram\_vectorizer.transform(reviews\_test\_clean)*

*X\_train, X\_val, y\_train, y\_val = train\_test\_split(*

*X, target, train\_size = 0.75*

*)*

*for c in [0.001, 0.005, 0.01, 0.05, 0.1]:*

*svm = LinearSVC(C=c)*

*svm.fit(X\_train, y\_train)*

*print ("Accuracy for C=%s: %s"*

*% (c, accuracy\_score(y\_val, svm.predict(X\_val))))*

Accuracy for C=0.001: 0.8808

Accuracy for C=0.005: 0.88528

Accuracy for C=0.01: 0.88704

Accuracy for C=0.05: 0.88592

Accuracy for C=0.1: 0.88608

*final = LinearSVC(C=0.1)*

*final.fit(X, target)*

*print ("Final Accuracy: %s"*

*% accuracy\_score(target, final.predict(X\_test)))*

Final Accuracy: 0.89944

Pretty close to 90!

## Visualization

Visualizing the most defining words/phrases is a good way to check your model’s sanity. One way of doing this is by using WordCloud:

*feature\_to\_coef = {*

*word: coef for word, coef in zip(*

*ngram\_vectorizer.get\_feature\_names(), final.coef\_[0]*

*)*

*}*

*positive\_words=[]*

*for best\_positive in sorted(*

*feature\_to\_coef.items(),*

*key=lambda x: x[1],*

*reverse=True)[:30]:*

*positive\_words.append(best\_positive[0])*

*best\_positive\_words = ' '.join(positive\_words)*

*from wordcloud import WordCloud*

*import matplotlib.pyplot as plt*

*wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(best\_positive\_words)*

*plt.figure(figsize=(10, 7))*

*plt.imshow(wordcloud, interpolation="bilinear")*

*plt.axis('off')*

*plt.show()*



*negative\_words=[]*

*for best\_negative in sorted(*

*feature\_to\_coef.items(),*

*key=lambda x: x[1],*

*reverse=False)[:30]:*

*negative\_words.append(best\_negative[0])*

*best\_neg\_words = ' '.join(negative\_words)*

*wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(best\_neg\_words)*

*plt.figure(figsize=(10, 7))*

*plt.imshow(wordcloud, interpolation="bilinear")*

*plt.axis('off')*

*plt.show()*



Clearly these results make sense.

#### Summary

#### We’ve gone over several options for transforming text that can improve the accuracy of an NLP model. Which combination of these techniques will yield the best results will depend on the task, data representation, and algorithms you choose. It’s always a good idea to try out many different combinations to see what works.