A Needle in a Data Haystack
Introduction to Data Science

Final Project

Movie Review Rating Predictor

Team Members:

- Abdelmoute Ewiwi, abdelmoute.ewiwi@mail.huji.ac.il, abdele
- Laith Abu Omar, laith.abuomar@mail.huji.ac.il, abuhisham

Problem Description:

Our project aims to solve the problem of predicting the possible score (rating) of a review based on the data in it.

We target the reviews of movies written by users at the IMDb website. At this website, users can write reviews on movies they have watched and include a rating from 1 to 10 of the movie.

Our job is to take the review written by a user and predict the value of the rating he gave alongside his review.

In other words, the problem can be thought of as an advanced sentiment analysis system, which identifies whether a given text is positive or negative based on its contents. However, including levels of sentiment (in our case 10 classes) can make the problem much more complicated, as we would see.

After deciding on the data and the goal of the project, we started thinking of the best possible way to achieve the identification of the given review, or in other words, to do the classification process. For this sake, we started

reading a bit about different machine learning that are able to do that job for us. Some of the main names were SVM, Naïve Bayes, and SGD.

We ended up using Sci-Kit learn (sklearn) Python library, and attempting several algorithms and techniques with variety of parameters, and we compared how each one fares.

Data:

The data we used was a collection of reviews of different movies written by different users and spreading over a different range of ratings (1-10), which were all collected from IMDb website.

We collected the data by crawling through the pages of the IMDb website using scrapy (which is the same crawler we used in Ex1).

The data is an extracted text from the reviews alongside the rating for that review.

The data is formed of 29000 entries, divided into 10 classes (the ratings), resulting in 2900 entries for each rating class. However, the data downloaded from the IMDb website using scrapy was much bigger than that (it was 258 MB formed of about 170000 entries). We encountered a problem of data imbalance when attempting to run the learning algorithm on our data, and we found out that there was an enormous difference between the number of entries in each class, so we decided to take the same number of entries from each class as the lowest one of the 10 classes, which turned out to be the class of the rating 2. We will discuss the issue of data imbalance further in the Solution section below.

The process of collecting the data (the crawler) worked as follows:

> There were actually 3 crawlers which collected the reviews from 200 different movies at IMDb.
>
> The first crawler (file: $getMovieList.py$) retrieves the links directing to the 200 movies from the main page of IMDb.

The second crawler (file: $getMovieReviews.py$) is given the 200 link as the start URLs, and it retrieves the links of the reviews page for each one of the 200 movies.

The third and final crawler (file: $getfinalReviews.py$) took as input the list of 200 urls of the reviews link for the movies, and returned as an output a JSON containing all the contents and rating of the reviews.

Once this process was done, we had a JSON file containing all the reviews with their ratings.

Our Solution:

As we mentioned earlier, we thought that the best possible way to tackle the problem is using machine learning algorithms. But we weren't exactly sure which one is better. Therefore, we decided to read on several algorithms, pick the ones which we find useful, and run them all one by one on the data in order to compare the results.

We used the Python library Sci-Kit learn (sklearn) for building the learning algorithms. We used 7 different machine learning algorithms which are:

- Perceptron
- Nearest Neighbors - kNN
- Random Forest
- Support Vector Machines - SVM
- Stochastic Gradient Descent - SGD
- Naïve Bayes

The solution process was as follows:

For easier work with the data we transformed it to $xlsx$ file using the $pandas$ library (file: $makeExcel.py$), then we loaded the data back.

As a first step we grouped the data in classes based on the label (rating). Then we cleaned the text from the excess html tags (like $< p >$) that came with the crawled data. Next, we represented the

data in vector form using TF-IDF vectorizer, which also included cleaning the text from stop words and irrelevant ones. TF-IDF also does a nice job of dividing the text. Instead of dividing it to single tokens it divides it receives as a parameter a range $(1 - n)$ then represents the text as $n - grams$ (unigram, bigram, etc…). And after some experiments we found out the bigrams lead to the best possible results as it helped us to catch some tricky words such as "not good", and "extremely awful" instead of being tackled by good and extreme. Then, we split the data to train (80%), and test (20%) called (trained) the learning algorithm with the given data using several parameters and then tested it on the remaining 20%.

After performing this process for the first time, we found out that the results were too low in terms of accuracy (Precision and Recall). After some searching, we found that we had an imbalance in the data. There were very few entries of one class (rating 2) compared to the other classes. So we decided to take the same number of entries from each class, which is the number of entries present in the lowest class (rating 2). In our case it was 2900 entries, which is where the 29000 entries came from. After doing so, we noticed that the results improved a lot, but the fact that we have 10 classes still affected the accuracy levels, and prevented us from reaching very high levels.

When this process was done, we had a trained algorithm that is able to predict the rating $(1 - 10)$ a given review (text) of a movie review.

Experiments:

We measured the performance of our solution using the results that each algorithm achieved on the test data. In addition, we calculated the precision and recall of each one, and compared the different.

As for success, the higher the accuracy of the algorithm is, the more successful it is. So we were aiming to increase the accuracy, precision and recall as much as possible. However, the fact that we had too many classes to classify affected the levels of accuracy.

Therefore, we attempted to decrease the number of classes in order to find out how it would affect the accuracy. Interestingly, we found out that the lower the number of classes is, the higher the accuracy is.

We tried dividing the data into $5, 4, 3,$ and 2 classes of ratings as follows:

- 5 classes: $\underbrace{1-2}_{class}, \underbrace{3-4}_{class}, 5-6, 7-8, \underbrace{9-10}_{class}$. (i.e. reviews of ratings 1 and 2 were combined in one class, and those of ratings 3 and 4 were combined in another class, and so on)
- 4 classes: $1-3, 4-5, 6-7, 8-10$.
- 3 classes: $\underbrace{1-3}_{Low\ Rating}$, $\underbrace{4-7}_{Medium\ Rating}$, $\underbrace{8-10}_{High\ Rating}$.
- 2 classes: $1-5, 6-10$.

After holding several experiments, the results were as follows:

	Perceptron		**kNN**		**Random Forest**		**SVM**		**SGD**		**NB**	
	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall
2 classes	0.86	0.86	0.79	0.81	0.86	0.85	0.88	0.88	0.89	0.89	0.88	0.87
3 classes	0.76	0.74	0.67	0.68	0.72	0.73	0.76	0.76	0.78	0.78	0.74	0.73
4 classes	0.58	0.61	0.50	0.53	0.62	0.61	0.65	0.65	0.67	0.67	0.63	0.63
5 classes	0.53	0.54	0.45	0.46	0.52	0.54	0.55	0.55	0.56	0.56	0.51	0.54

Important Note:

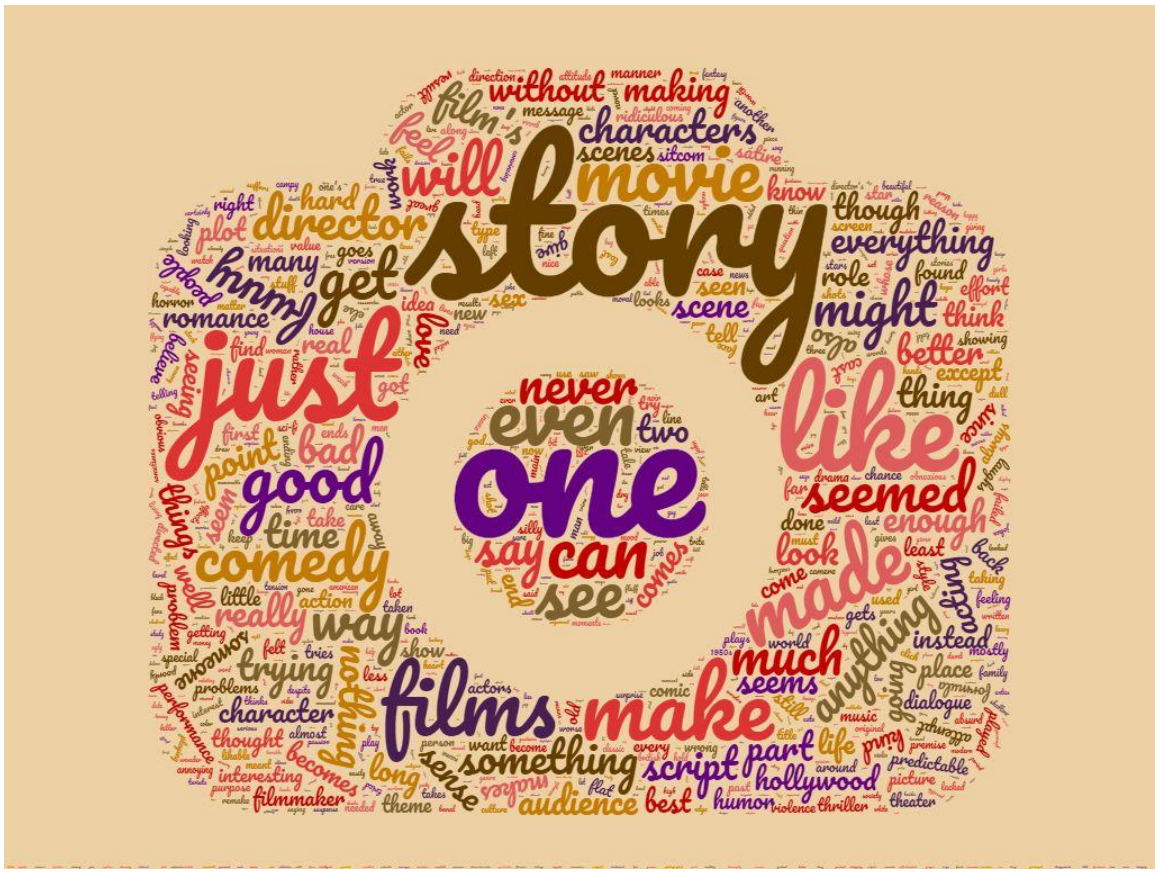
Note that when increasing the number of $n-grams$ used in the TF-IDF, we get higher levels of accuracy. However, we ran the code on unigrams and bigrams and it took a while. We tried running it on trigrams, which we thought would be good to test, but it seemed that it needed a very long time to run. Maybe this is related to our inefficient implementation of the learning algorithms. Anyways, we wanted to note that it is possible to boost the numbers above with more computation time.

Here is a screenshot of some of the results:

```
Training:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=10, p=2,
           weights='uniform')
train time: 0.011s
test time:  3.935s
accuracy:   0.791
             precision    recall  f1-score   support

          0       0.79      0.78      0.79      1140
          1       0.79      0.81      0.80      1180

avg / total       0.79      0.79      0.79      2320

================================================================================
Random forest
________________________________________________________________________________
Training:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=100, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
train time: 70.524s
test time:  0.338s
accuracy:   0.854
             precision    recall  f1-score   support

          0       0.83      0.88      0.86      1140
          1       0.88      0.83      0.85      1180

avg / total       0.86      0.85      0.85      2320

================================================================================
L2 penalty
________________________________________________________________________________
Training:
LinearSVC(C=1.0, class_weight=None, dual=False, fit_intercept=True,
     intercept_scaling=1, loss='l2', max_iter=1000, multi_class='ovr',
     penalty='l2', random_state=None, tol=0.001, verbose=0)
/Users/abed/anaconda/lib/python3.5/site-packages/sklearn/svm/classes.py:199: DeprecationWarning: loss='l2' has been deprecated in favor of loss='squared_hinge' as of 0.16. Backward compatibility for the l
oss='l2' will be removed in 1.0
  DeprecationWarning)
train time: 0.988s
test time:  0.001s
accuracy:   0.887
             precision    recall  f1-score   support

          0       0.88      0.89      0.89      1140
          1       0.89      0.88      0.89      1180

avg / total       0.89      0.89      0.89      2320

________________________________________________________________________________
Training:
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=50, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=None, shuffle=True,
       verbose=0, warm_start=False)
train time: 0.593s
test time:  0.001s
```

Visualization:

Below are some visualization of our data



Future Work:

As a continuation to our project, we think a nice idea might be to group reviews with high level of similarity (intersection) and extract some useful information regarding that class of reviews on a specific movie. For example, discover what the reason behind the dissatisfaction of the viewers was. We might end up getting some weird results such as an agreement that certain actor or scene was responsible for the failure of the movie. Or on the other hand, we might find out that the connection of several actors and actresses seems to be successful.

Going even further, we can build an analysis system that takes all the data from the reviews that it stores, and attempts to suggest which actors to hire for a specific category in order to have good chances of success.

Brief Conclusion:

In conclusion, we saw how important data could be, and found out that there are always endless ways to benefit from it in different forms. We discovered new ways to classify data using new machine learning algorithms. In addition, we saw how data visualization can improve our understanding of the data and discover problems such as imbalance in the corpus. Last but not least, we saw how simple problems such as sentiment analysis can be made much more complex if taken to new levels and defined in new ways, which was clear when we lowered the number of rating classes within our predictor, which lead to a much higher accuracy than the one we got when dealing with a larger number of classes.