Tim auf der Landwehr
t.auf.der.landwehr@student.rug.nl
S2548682
Last revision: January 20, 2014

Learning from Data - Final Project

Predicting opening-weekend revenue for movies from critic reviews

---

Summary

This is the documentation of the final project in the course *Learning from Data* at the University of Groningen.

Predicting opening-weekend revenue for movies from critic reviews

1. Problem description

This project is about the prediction of opening weekend revenue for movies from critic reviews. It is based on the dataset published[1] by some researchers at the *Carnagie Mellon University*, Pittburgh, USA.

This project is based on the paper [JDGS10] they published about predicting the overall opening-weekend revenue and the per-screen revenue per movie. In this projekt I will focus only on the first part of predicting the overall opening-weekend revenue and first try to reproduce their results before I try to improve them by using different features.

The data that is provided contains in addition to the reviews from seven internet pages review snippets from many different sources, that mostly consist of only one ore two sentences and include the source URL and the name of the author. Furthermore a lot of meta data is provided as shown in table 2. The value *weekend gross* is the one to predict and therefore the value *US gross* cannot be considered, because it is not known at the real time of prediction.

The data is furthermore divided in the sets *train*, *dev* and *test*. In the following all results show a model trained on the set *train* and tested on *dev*. The *test* set was not touched during this work.

Table 1: The seven review sources

www.austinchronicle.com
www.boston.com
www.calendarlive.com
www.ew.com
www.nytimes.com
www.variety.com
www.villagevoice.com

Table 2: The available metadata

company	number of screens
genres	summer release
authors	christmas release
directors	memorial release
actors	independence release
origins	labor release
release date	no of highest grossing actors
running time	production budget
rating	*weekend gross*
US gross	

2. Metadata features

The first step is to attempt to reproduce the results which are presented in the paper. Their metadata model contains the features shown in table 3. According to that, the features 1 to 6 are implemented as suggested. Feature 7 is implemented in that way, that all five different release types are added as single features. Feature 8 is skipped, because this would require some more logic in such a way, that only those Oscar-winners may be considered who won an oscar before the release date of the certain movie. Ignoring this would be cheating, because it would make use of information that was not available on the release date of the movie. Feature 9 is

[1] http://www.ark.cs.cmu.edu/movie$-data/

Table 3: Metadata features used in the paper

1. whether the film is of U.S. origin
2. running time (in minutes)
3. the logarithm of its budget
4. number of opening screens
5. genre (e.g., Action, Comedy)
6. MPAA rating (e.g., G, PG, PG-13)
7. whether the movie opened on a holiday weekend or in summer months
8. total count as well as of presence of individual Oscar-winning actors and directors
9. total count as well as of presence of high-grossing actors

limited to the count of high-grossing actors, the presence was not added as a seperate feature. The described set of features leads to the following peformance on the development set.

```
=== Error on test data ===
Correlation coefficient              0.7803
Mean absolute error            6384990.8
```

This result is quite surprising, since the original paper states a correlation coefficient of $r = 0.722$ and mean absolute error $MAE = 5983000$. This means my implementation of the features correlates better with the weekend-gross but has also a higher mean absolute error. It can be reproduced using the command `movie_pirate.py --fs 1` (for further information see appendix).

Motivated from these results I tried some modifications. Instead of using the binary feature if a movie was produced in the United States, I tried to add all origins that occur at least five times in the training set as single features. Unfortunately this decreased the results slightly (`movie_pirate.py --fs 2`).

```
=== Error on test data ===
Correlation coefficient              0.7796
Mean absolute error            6506275.919
```

Wondering why the logarithm of the production budget was used instead of the plain budget, I simply tried that variant. It increased the results enourmously (`movie_pirate.py --fs 3`).

```
=== Error on test data ===
Correlation coefficient              0.8547
Mean absolute error            5338370.8613
```

In addition to that I tried several different combinations of adding features based on directors, actors, authors and the company. Many of these decrease the results quite much, but adding only the authors who occur at least five times in the training set and at the same time removing the genre results in an even better correlation and a lower mean error (`movie_pirate.py --fs 4`).

```
=== Error on test data ===
Correlation coefficient              0.8607
Mean absolute error            5199621.8426
```

Finally after many different versions I added the most frequent snippet writers to the feature set and could once more improve the result slightly (`movie_pirate.py --fs 11`).

```
=== Error on test data ===
Correlation coefficient              0.8656
Mean absolute error            5148645.6489
```

This is the best result I was able to reach by only using metadata features. For some reason, this result is much higher than the metadata approach used in the paper, and it even beats the best results that were reached by combining the metadata features with the review text features.

Table 4: Review based text features used in the paper

word features:	1. word unigrams
	2. word bigrams
	3. word trigrams
pos features:	4. pos unigrams
	5. pos bigrams
	6. pos trigrams
dependency features:	7. dependency triples <relation, head word, modifier word>

3. Review features

The main part of the given paper covers the features that can be extracted from the reviews of a movie. In the paper the features shown in table 4 are used. Unfortunately the explanations are quite rare, so they do not state how many features there were in the end and which words are on the stop list. I implemented the word features unigrams, bigrams and trigrams that way, that I first ignored those that consist only of words which are under the 100 most frequent english words[2]. Later I figured that this is not the best solution, because the word 'good' for example was also under these words. Therefore I replaced it with an actual stoplist wof words[3] In addition to that I limited the features to those that occured at least 30 times for unigrams and bigrams and 20 times for trigrams. This lead to a total of 4190 features which were extracted from the training set. Unfortunately after running `movie\_pirate.py` for about 20 minutes to extract these features, weka did not even give an answer within 2 hours. Therefore I tried to reduce the number of features used strongly. This lead to recognizing unigrams only with at least 100, bigrams with at least 60 and trigrams with at least 40 occurences, what reduced the total number of features to 664 (`movie_pirate.py --fs 5`). With this set I ran *WEKA* again and got the result:

```
=== Error on test data ===
Correlation coefficient              0.5832
Mean absolute error            9926281.4848
```

[2] https://en.wikipedia.org/wiki/Most_common_words_in_English
[3] http://www.ranks.nl/resources/stopwords.html

This was not as good as I expected, so I set the limit further upwards to at least 200, 100 and 50 uni-, bi- and trigrams. This lead to the following results (`movie_pirate.py --fs 6`).

```
=== Error on test data ===
Correlation coefficient              0.6372
Mean absolute error            9026632.8168
```

Since that was a huge improvement I went further in that direction, but decreased the trigrams limit, because there appeared to be no more left in the feature set. At least 250, 120 and 40 uni-, bi- and trigrams produced the following outcome (`movie_pirate.py --fs 7`).

```
=== Error on test data ===
Correlation coefficient              0.5878
Mean absolute error            9261026.9581
```

That is again worse. I cosidered to use *SimpleLinearRegression* instead of *LinearRegression* in *WEKA*, I ran this method on the last two feature sets to compare these two regression methods. Unfortunately the results were much worse and additionally I figured out, that *SimpleLinear-Regression* only extracts one best fitting feature from the feature set. For this reason it is no alternative. Theredore I changed the way of way of feature selection to be able to select the x most frequent uni-, bi- and trigrams. The first attempt improved both the correlation and the error again, but did not beat the results of feature set 6. I considered here the 100 most frequent of each of the uni-, bi- and trigrams (`movie_pirate.py --fs 8`).

```
=== Error on test data ===
Correlation coefficient              0.6261
Mean absolute error            8530556.4866
```

Leaving out the trigrams and using the each 150 most frequent unigrams and bigrams improved the result slightly (`movie_pirate.py --fs 9`).

```
=== Error on test data ===
Correlation coefficient              0.6485
Mean absolute error            8767310.8511
```

Adding to this configuration again the 50 most frequent trigrams improved the correlation slightly (`movie_pirate.py --fs 10`).

```
=== Error on test data ===
Correlation coefficient              0.6531
Mean absolute error            8692146.9429
```

Further experiments increasing and/or decreasing the number of most frequent uni-,bi- and trigrams were not successful, they decreased the correlation. Changing the length of the stop-list from 100 to 50 or 25 words does not harm the outcome of the tests, removing the stop list completely however resulted in pretty low results.

4. Feature Combination

The most obvious combination of the feature sets is the combination of the best working meta-data features with the best working review text features. This combination can be run adding multiple parameters (`movie_pirate.py --fs 11 --fs 10`) and produces the following output:

```
=== Error on test data ===
Correlation coefficient              0.8362
Mean absolute error            6277648.3008
```

5. Conclusion and Remarks

The results of the metadata approach are stunning and are difficult to explain from my point of view. I do not see why they beat the results in the paper by far. In contradiction to that, the results of the review part are not satisfactory at all. I was not able to reproduce the results in the paper. This is partially caused by the simple fact that the running time of these features is really high, which makes it difficult to simply try many different configurations of the word n-gram features. Furthermore I did not implement the pos-n-grams and dependency relations, which however do not significantly increase the results in the paper. The combination of meta data features and review text features performs worse than the plain meta data model, but still better than the one in the paper.

Since the best performing feature set was the one using only metadata, I would like to let this model perform on the unknown test-data. Besides that it would still be interesting to compare the performance of the combination of the metadata and the n-grams (featuresets 11 and 10) on the testset.

The appendix contains all necessary information to use `movie_pirate.py` and *WEKA* in order to reproduce the provided results. In addition to that it contains information to run weka on the testset.

References

[JDGS10] Mahesh Joshi, Dipanjan Das, Kevin Gimpel, and Noah A Smith. Movie reviews and revenues: An experiment in text regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 293–296. Association for Computational Linguistics, 2010.

Appendix

A. movie_pirate.py

A.1 Requirements

The file `movie_pirate.py` is written in `python3` and requires the following imports:

```
import sys
import getopt
import xml.etree.ElementTree as ET
from collections import OrderedDict
import string
import os
import json
from math import log
import re
import operator
```

In addition to that it requires the following files and folders to lie in the same directory.

```
POS_DIR = 'pos/'
PARSED_DIR = 'parsed/'
TEXT_DIR = 'text/'
META_DIR = 'metacritic+starpower+holiday+revenue+screens+reviews/'
WEKA_JAR_PATH = './weka.jar'
STOPWORDS_FILE = 'STOPWORDS'
```

`STOPWORDS` contains the list of stopwords as explained in section 1. If the according features to `OSCAR_ACTORS` and `OSCAR_DIRECTORS` are used, the files having the same names are required as well. By default in no feature set these features are used, because they can be considered as cheating as explained in section 2.

Furthermore the movie-data must be available in the same directory. The folders `pos`, `parsed` and `text` contain each one file for each movie with the pos-tagged, the parsed and the plain text version of the reviews. The folder `metacritic+starpower+holiday+revenue+screens+reviews/` contains one xml file for each movie with meta data and review snippets.

Last but not least the input files must be present. As input files the files `train` and `dev` (and `test`) from the folder `traindevtest_splits` are used. They should be available in the same folder as the `movie_pirate.py`

A.2 Usage

The usage of `movie\_pirate.py` is very simple and basically explained by the help output:

```
Usage: movie\_pirate.py
  > Add at least a single feature or a feature set

--fs [1..11]
 Add a set of features
--f [feature]
 Add a single feature

--train <file>
```

```
 Use <file> as training file, default: <train>
--test <file>
 Use <file> as training file, default: <dev>
--run-weka
 Run WEKA on the generated output files,
  > requires [weka.jar] in CLASSPATH or ./weka.jar
```

The parameter `--fs X` is required, while `X` defines the feature set to be used. In this report each time the correspondent feature set for reproduction is mentioned. Alternatively the parameter `--f [feature]` can be used as well, but this parameter is intended for debugging purposes and is used to add single features and certain named feature sets to the model. Running `movie_pirate.py --f BEST` will trigger the best known featureset which is the same as `movie_pirate.py --fs 4`.

Furthermore the training and test data can be selected by using the parameters `--train <file>` and `--test <file>`. By default they point at the training and development set.

Using the parameter `--run-weka` will immediately after the generation of the arff files run *WEKA's* LinearRegression on these. That parameter requires the weka.jar file lying in the same directory or being in the java CLASSPATH.

B. Usage of WEKA

To run weka, always the following command was used:

```
java weka.classifiers.functions.LinearRegression -t train.arff -T dev.arff
```

This actually skips the step of writing the model to a certain file. In order to dump the model for reusage (which especially for a high number of features saves computing time), the following two commands can be used:

```
java weka.classifiers.functions.LinearRegression -t train.arff -d model.weka
java weka.classifiers.functions.LinearRegression -l model.weka -T dev.arff
```

If the parameter `--run-weka` is set, the movie pirate will run the first command and pass the appropriate arff files to it.

C. How to run on the testset

To run the testset is as simple as the following command:

```
./movie_pirate --fs 11 --test=test
```

or equally `./movie_pirate --f BEST --test=test` If weka is in the classpath or the weka.jar is in the same folder it is also possible to run it immediately after the generation by adding the parameter `--run-weka`.

```
./movie_pirate --fs 11 --test=test --run-weka
```

In addition to that the perfomance of the best review text feature set (`--fs 10`) and the best combination (`--fs 11 --fs 10`) could also be interesting, but will probably be below the performance of the meta data model.