Faculty of mathematics physics and informatics

Seminar thesis for Machine learning

Movie genre classification

Student:	Martin Slavkovský
Studies:	Applied informatics
Semester:	1.
E-Mail:	martinslavkovsky7@gmail.com

14.12.2015 Bratislava

1 Introduction

Goal of the project was to find suitable method for genre classification of the movies according to the synopsis of the movie. As synopses of particular genre contain similar words (for example action movies contain a lot of “kill” words) this can be in fact used as an attribute for a classification algorithm.

My objective was to create simple application which would be able to parse raw data from IMDB database, select random movies and create training and testing data which can be later used by classifier. Later some preprocessing would be done on generated data and finally one of the machine learning methods which we went trough during semester would be used. As one movie can be classified by multiple genres (Terminator is action, sci-fi)it was necessary to find algorithm which is able to do multi-label classification. It was also necessary to do multi-class classification as there are more than two genres in the dataset.

I personally think project was successful as accuracy and precision rates were in 65%-95% range which is probably better than untrained human would do. After some preprocessing SVM was used as classification method. I have also used two methods for dealing with multi-class problem, they are described later in the paper.

2 Dataset

As IMDb is currently largest and most popular movie database on the internet selecting this for my project was a logical choice. IMDb portal provides [1] alternative interface for their data and movie synopsis and genres which can be downloaded from ftp server as simple text files. However these must be parsed as they are not in a common format or some kind of database file.

Plain data files contained more than 100 000 movies which is sufficient for the project but also unnecessary large. In experiments I have worked with datasets with sizes 5000 and 20000 and in fact there were only minor changes in accuracy of algorithm.
As plain english text is not really good for classification one must find a way to transform synopsis of the movie to a vector. As generated plain text of 20 000 synopses might contains up to 100 000 unique words classification with vector of this size would be painfully slow and also ineffective. When we realise that many “unique” words are the same just in the different form we can find a way to reduce size of this

vector. This is called ***stemming*** and by experimenting I found out that size of the original dictionary was reduced approximately to one third. As example movie dataset containing 20 000 synopses would generate dictionary of 154226 unique word, but after stemming size was only 45898 . I have tried several stemmers and I got best result with [2] [3] ***Porter Stemmer Algorithm(1980).*** Second choice would be *Lovins Stemmer*, but it seemed that was it cutting a lot more from word than Porter's algorithm which could effect accuracy in a bad way. Before stemming some stop words (a, an, which, what, or, every ..) are also removed from the movie plots as they would be most common words in almost every genre and would not provide very good genre related information. From dictionary reduced by stemming ***Bag of Words*** representation is applied to store movie plots into vector and only words which are present in the whole dataset more than ten times are used. Final vector used in the classification would have then around 10 000 attributes which may seem still too large but it's sufficient for the good classification by SVM or other methods. Just for the information dataset of 5000 words generated vectors with circa 4000 attributes so it's obvious that reducing number of movies will not reduce vector size linearly.

Even after removing *stop words* there were be words in dictionary which would occur more often than the others. That's why we need to apply some kind of term weighting to generated vector. Method used in project is called [4] ***term frequency-inverse document frequency*** or short ***tf-idf***. *T*f-idf will is computed by following formula

$$tf.idf(i,j) = \frac{n_{i,j}}{\sum_k n_{k,j}} log \frac{|D|}{|\{j : t_i \epsilon d_j\}|}$$

This basically means -> *tf-idf(word, synopsis) = (word count in movie synopses / plot.length) * log * (synopses count / synopses which contain word)*

Finally weighted vector is normalized in order to minimize impact of variation of lengths of synopses. I will get to number of positive/negative samples in next chapter as it's closely connected to the multi-label classification problem. Filtered dataset was slitted into two subsets. Training data would contain 80% of sample white testing subset would contain 20%.

3 Classification methods

As mentioned before SVM was used for the classification. Main reason for choosing SVM was interesting [5] work about classification of wikipedia articles in which author was solving very similar problem to mine. Genre as well as wikipedia article classification need to solve multi-class, multi-label classification. They claimed 95-99% accuracy of the algorithm which they trained using SVM. As classification between two classes {-1,1} is very simple the real issue here was to solve multi-label problem for which I have used two methods. As mentioned before also two ways of generating training, testing data were used.

3.0.1 Label combination

First method I have tried to experiment with was *label combination*. In this case we only use one set of training/testing data (let's say 20 000) where data are selected completely by random choice and ratio between genres is ignored . That means dataset can contain for example 50% of action movies and less than 10% of other genres. After generating synapses vectors arrays of classes/genres $\vec{\mathbf{c}}$ (movie can be [action, drama, thriller]) are transform to single class $\boldsymbol{s}$. Original training set is than transformed to set where $\boldsymbol{s} \in \mathbb{R}1$. After training and prediction is made predicted data are transformed back to $\vec{\mathbf{c}}$. Accuracy of classifying data perfectly, meaning when all genres were predicted correctly and we didn't miss any, was around 15-20% which is pretty bad. Changing kernel from linear to other types or playing with other parameters didn't really help. However if say we ignore that some of actual genres are missing in predicted $\vec{\mathbf{c}}$ or even if we allow misclassifying some of them, results are much better. With threshold set to 1.0 (non of predicted genres were misclassified but some of them could be missed) accuracy was around 50% , 65% with threshold set to 0.5 which still is not perfect but if we realise that we have algorithm that is able to guess at least half of the genres correctly it's still probably better than untrained human which is almost everyone. I fact it would interesting to see how would average person perform in this task. However these “mixed” results forced me to try *binary approach.*

3.0.2 Binary approach

In this scenario dataset is generated for every genre separately. During classification we ask if movie belongs to certain genre or not. For example if we have movie M1[action, genre,drama] and M2[history], M1 will be classified as positive sample , M2 as negative. Reason why I don't use the same dataset for every genre is weight balancing as in absolutely random dataset less common genres would be represented only by 5% of data and result would be pretty bad. I also have tried to experiment with -w option of libsvm but it didn't really help. In training set there are approximately half of positive samples and half of negative ones. Disadvantage of the method is that we have to do a lot more file parsing also we have at least 20 trained svm models but result in accuracy and precision are much better. (from 65-95% depending on genre). Enlarging training set didn't really make much difference in terms of accuracy or recall.

4 Implementation

Resulting application is written in Java programming language . I know Java pretty well , it's easy to use and it is pretty well suited for text parsing in contrast to Mathlab or Octave. Also there is Porter stemming algorithm available on Mr. Porter's web written in Java. For SVM I use **[6] *LIBSVM*** which is also open source and it's free to use or modify. It also supports one-vs-all algorithm for dealing with multi-class problem. I actually had to modify source code of ***LIBSVM*** little bit as it was lacking support for multi-core computing for which I have used intel ***libiomp5*** library. This makes training a lot faster especially in *Label combination* approach.

As it's not necessary to do everything at the everytime we would like to run application as it is slitted into four phases which can be run separately. In first phase we parse data files downloaded from IMDb database and we select specified number of movies to save them into $GENRE_plain.txt file. Movie synopses are stemmed and stop words are filtered in second phase. Result is saved to $GENRE_stemmed.txt. In next phase BoW dictionary is generated from which we compute tf-idf vectors. Computing tf-idf is in fact CPU consuming as we need to do |BoW|*|Movies| log computations so if we use large set of movies we don't want to repeat this phase again. Results are saved into $GENRE_train.svm, $GENRE_test.svm which are input files for libsvm binaries **svm-train, svm-predict.** Final phase uses **svm-train** to create model for

training data and finally prediction is made. Application has also option to compute precision,recall and F-mesaure as **svm-predict** is only able to compute accuracy. Different approaches have been used for computing accuracy of *Binary approach* than in Label combination approach.

5 Experiment

As mentioned before I have used two methods for dealing with multi-class problem so I also had to use two ways of compute results.

5.0.1 Label combination

Using *Label combination* I was able predict 67.7% of movies with threshold set to 0.5 (half of the movies were guessed correctly). I didn't try to research more what was problem with kind a low accuracy and I rather focused on binary solution. Switching between kernel also didn't make almost any difference in final accuracy.

5.0.2 Binary approach

Following tables show results for binary method with datasets of 5000 movies:

	Action	Adult	adventure	animation	biography	comedy	crime
Accuracy	77.70%	85.25%	73.20%	80.50%	76.70%	71.40%	82.00%
Precession	74.40%	88.32%	76.23%	83.23%	74.51%	74.24%	83.51%
Recall	78.37%	82.89%	69.40%	77.17%	78.38%	67.25%	79.84%
F-Mesaure	76.40%	85.52%	72.65%	80.08%	76.39%	70.58%	81.63%
Genres pre movie	3.22	1.74	3.3868	3.46	3.3308	2.4872	3.22
Averag. plot length	55	46.16	54.2158	48.38	51.3544	50.0794	54.5054

	documentary	drama	family	fantasy	Film-noir	Game-show	history
Accuracy	84.70%	68.50%	71.00%	77.30%	85.19%	92.86%	76.80%
Precession	83.40%	69.96%	71.61%	76.84%	82.91%	99.12%	77.34%
Recall	85.95%	65.94%	68.42%	76.69%	88.99%	86.26%	75.15%
F-Mesaure	84.65%	67.89%	69.98%	76.77%	85.84%	92.24%	76.23%
Genres pre movie	2.16	2.46	3.36	3.55	3.62	1.7366818874	3.3684
Averag. plot length	50.71	50.88	52.43	51.68	65.88	59.331050228	53.1444

	horror	music	musical	mystery	news	Reality-tv	romance
Accuracy	84.20%	83.40%	76.80%	74.30%	85.10%	87.05%	73.40%
Precession	88.15%	89.64%	77.99%	79.01%	83.91%	88.86%	70.58%
Recall	79.88%	75.15%	74.55%	68.09%	86.77%	84.72%	75.05%
F-Mesaure	83.81%	81.76%	76.23%	73.15%	85.32%	86.74%	72.75%
Genres pre movie	2.62	2.83	3.08	3.43	3.29	1.44	3.06
Averag. plot length	51.28	50.63	54.83	53.28	50.49	50.03	53.63

	Sci-fi	short	sport	Talk-show	war	western	thriller
Accuracy	83.60%	72.10%	89.30%	92.96%	85.50%	91.80%	73.30%
Precession	85.09%	74.70%	94.26%	97.44%	90.16%	96.80%	73.35%
Recall	80.17%	70.86%	84.06%	87.02%	78.89%	87.14%	72.01%
F-Mesaure	82.55%	72.73%	88.87%	91.94%	84.15%	91.72%	72.67%
Genres pre movie	3.20	2.63	2.93	1.56	3.27	2.41	3.06
Averag. plot length	52.06	49.79	52.25	48.96	53.22	57.27	53.83

When using *Binary approach* I have tried to make training set larger (I've tried 20 000) but it didn't boost accuracy much. For example with action movies accuracy was 81%.

At first when I had some issues with very low recall. As mentioned before data were unbalanced (for example I had only 5% of positive samples) and I have tried to play with -w option but I still had issues with the very low *recall* even when accuracy was at 95%. All test data were in this case predicted as negative (none of test data were classified as action movie). Solution was to generate balanced data when parsing IMDb database.

I got best result in terms of accuracy and variations between accuracy, recall and precision when using Radial basis function kernel (-t 2 option in libsvm).

	action without using cost	action with C = 2, gamma = 2	linear kernel C = 2, gamma = 2
Accuracy	51.20%	77.70%	75.50%
Precession	51.20%	78.96%	75.92%
Recall	100.00%	76.95%	76.37%
F-Mesaure	67.72%	77.94%	76.14%

To boost accuracy I've tried to constrain number of movies per genre. Result's were following:

	action multiple genres	Action 1 genre	Adventure multiple genres	Adventure 1 genre
Accuracy	76.90%	83.68%	72.00%	77.06%
Precession	78.40%	82.06%	72.94%	80.00%
Recall	75.15%	84.74%	73.22%	74.24%
F-Mesaure	76.74%	83.38%	73.08%	77.01%
Genres pre movie	3.28	1.00	3.40	1.00
Average plot size	55.71	52.19	54.65	48.98
Most common genres	drama(952),adventure(643),thriller(569),short(513)		action(877),drama(657),short(531),comedy(508)	
Most common word	find(901),kill(690),man(684),life(676),world(638),year(547),live(494),time(479),fight(478),forc(473)	kill(696),find(632),life(516),man(501),polic(490),fight(431),live(391),world(371),year(356),forc(354)	find(969),world(787),year(566),life(565),time(562),man(520),friend(509),young(498),film(497),stori(494)	find(355),stori(306),gag(294),world(286),adventur(262),love(239),beauti(223),bondag(213),life(207),year(196)

I have intentionally picked here action and adventure as they seem to be very similar and many of movies are action as well as adventure movies. (can be seen in most common genres row) They share common words and this is probably the reason why I accuracy was higher for some genres than for the others. Less common genres and with more unique words showed better results. For example these were most common words for war movies : *war, world, soldier, film, story, life, german, american, army, year.* Words like war or soldier couldn't be found in list of most common words in other genres.

6 Conclusion

I think project was successful even if I wasn't able to gain more than 90% accuracy for every genre. In the case I would get IMDB date but without genres it would be probably good way to tag them. I would do project again I would probably try to use different classification methods or boost accuracy a little bit. I would also research more *Label combination approach* to see if I would be to get better result using some kind of dataset weighting.

In general approach using stemming, tf-idf, svm could and from what I have read is commonly used for text classification. (wikipedia article classification [6])

7 Bibliography

[1] http://www.imdb.com/interfaces
[2] http://tartarus.org/martin/PorterStemmer/
[3] http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
[4] http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html
[5] http://konceptgeek.github.io/assets/files/multi_label_svm.pdf
[6] http://www.csie.ntu.edu.tw/~cjlin/libsvm/