Technische Universität Berlin

AIM3 project

IMDb Movie Classification

Author:
Florian Gössler
Xi Yang

Supervisor:
Christoph Boden
Sebastian Schelter
Juan Soto
Prof. Volker Markl

July 15, 2015

1. INTRODUCTION

The Internet Movie Database (IMDb) [1] with its over 3 million listed titles provides a great resources for information about all sorts of movies. Maintaining this amount of data is quite challenging as new movies need to be added all the time.

Although the data is in a multi GB space one would not necessarily consider it to be 'Big Data'. Especially in comparison to things like Twitter's continuous stream of data or large scale sensor networks the IMDb is quite small. Any analysis could probably be done with conventional systems. But for this academic project we wanted to try to build a scaleable solution, which could be easily adopted for other, larger data sources.

For this project we decided to try a classification task on the available movie metadata to predict the genres of the movie. This could be helpful for people who enter information about new movies or also to complete the IMDb since there are a lot of movies without genre information. Part of this analysis is text classification which is a problem that occurs in many domains.

For the genre classification it is also interesting whether it is possible to predict a genre in a reliable way. Often this task is challenging even for humans cause a genre itself doesn't have a formal specification and can be a bit fuzzy. There are also mixups of genres, like a tragic comedy, which might be hard to classify since they contain features of multiple genres. Evaluating whether a machine can learn sort of a formal definition in form of some model is interesting since a further analysis of the model could lead to a formal, human understandable definition of a genre. It could solve the question of "What makes up a genre?".

2. PROBLEM STATEMENT

2.1 Background

Movie genres are manually created categorical labels to characterise different movies. These genre labels can be used to describe the movie, contribute in movie recommendations based on domain specified interests and some other fields. Currently the movie genres annotation are determined by human beings. The complexity in automatic genre classification is caused by the fact that usually the genres are not well defined. Normally there are no strict definitions or boundaries between 2 different genres. And of course they can share certain overlapping characteristics.

Facing the fact that the number of movies is growing at a fast pace, automatic movie genre classification can assist or replace the human work in this process and provide important components for the complete movie information. Thus, we begin to think about: what kind of information is sufficient to automatically classify the movie and how to approach that?

Thus, our project is aimed to build a scalable movie genre classification system using the Internet Movie Database(IMDb) and compare the results using different features like plot description, actors and key words.

2.2 Dataset

IMDb is an online dataset which provides all kinds of information related to movies. The information includes actors, production crew, fictional characters featured in these and so on. There are some publicly available datasets containing a subset of the online database, which can be downloaded and processed.

The dataset is organised into a set of compressed text files, where each file contains information on a specific aspect. The 4 main files we use are genres.list, plot.list, actors.list and keywords.list. The total size of the uncompressed files is about 1.7 GB. Although this is not big enough to be considered as 'Big Data', our approach is easy to adopted for larger data.

2.3 Tools & Languages

For our project the tools should meet the following requirements:

- Easy and convinient way to preprocess the raw data to get formatted data
- Scalable data processing framwork with suited machine learning algorithms

By comparing the advantages and disadvantages, we decided to use these tools & languages:

Python: Python is a fast and friendly language. It's easy to learn, can work everywhere and it takes much less time to develop with it. Python is an excellent tool for scanning and manipulating textual data. The raw data from IMDb cannot be used directly, so we choose Python to do the preprocessing work and generate formatted data for the next steps.

Apache Spark [2]: Spark is an open source Apache project built around scaleability, speed and ease of use. It provides a fast and generic data processing platform. Spark enables us to quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators. Other than Spark Core API, there are additional libraries that are part of the Spark ecosystem and provide advanced capabilities for big data analytics and machine learning. The scalable machine learning library (MLlib) consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimisation primitives.

Scala: Scala is a programming language for general software applications based on the JVM. It has a built-in type inference mechanism which allows the programmer to omit certain type annotations by still maintaining the benefits of a strongly typed language. For instance, it is not necessary to specify the type of a variable, since the compiler can deduce the type from the initialisation expression of the variable. The conciseness of Scala has a productivity benefit and can increase the readability of the code. Additionally, it is also a full-blown functional language. Except for the first-class functions, a library with efficient immutable data structures and a general preference of immutability over mutation are included in Scala. Most Scala data frameworks have the notion of some abstract data type, which is extremely consistent with Scala's collection API. Similarly, the concept of RDDs in Spark leverages the same functional and immutable style with function like map, flatMap, filter, reduce, fold and groupBy. Knowing these higher order functions it's easy to use Scala as well as those dataflow frameworks like Spark.

3. METHODOLOGY

3.1 Approach

For this data analysis task we decided to follow a multi step approach and build a machine learning pipeline using Apache Spark.

We started our work with some manual exploration of the data. After that we had to do some preprocessing to get the raw data into a format that could be used by Spark.

Based on this preprocessed data and the insight we got about the data we started to construct the pipeline which consists of the following steps:

- Load the raw data.
- Split textual documents into words.
- Convert those word vectors into a numerical feature vectors.
- Learn a prediction model using the feature vectors and labels.
- Use the learned model to classify unseen data points.
- Evaluate those classifications.

Of course there are a lot of tuning possibilities and variations in the generation of features which we will discuss in the experiments section.

3.2 Exploration

To get started, we had a look at the IMDb website itself. That way we could look at some random data points in a nicely formatted way and could evaluate which features we could use for our analysis and also how those features are structured.

After downloading the raw data we had a look at its format and could evaluate the size of the data.

We used Tableau to get further insight of the dataset and see what genres we are dealing with. Figure 1 shows the distribution of all movie genres. We can easily see that Drama, Comedy and Documentary are the largest genres. This means for those 3 genres, there is much more input data to train the corresponding models. This might increase the classification accuracy. On the other hand, the large quantity means that the scope for those 3 genres is much larger, which means there are not so many common similarities inside each scope, leading to a decrease in accuracy.

3.3 Preprocessing

After downloading the data file from `ftp.fu-berlin.de`, we found that these files are formatted just to be difficult to use. There are no csv files and all the textual files are formatted in a general way:

- CRC Line with file name and date
- Header data, sometimes instructions
- Header line (MOVIES LIST, etc.)
- Separator line, usually dashes
- Optionally more data and instructions

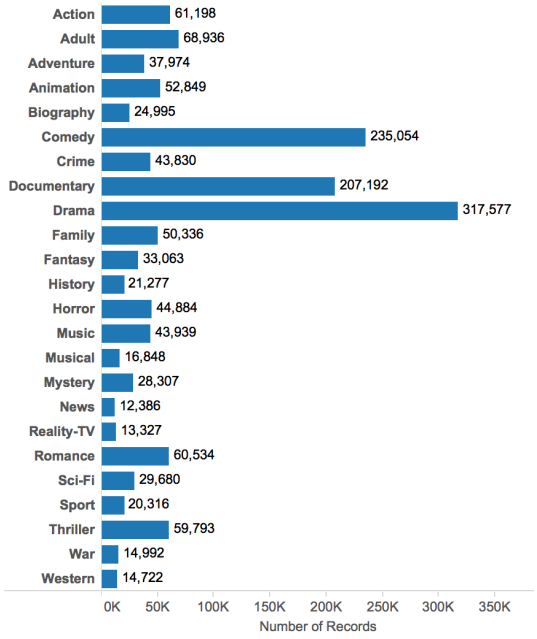

Figure 1: size of the evaluated genres

The real data is between the head line and the separator line and still not good formatted. For example, in plot.list, the plot description is not a single line as we need it for the input of Spark pipeline, but split into several lines and begin with the notation "PL". So we need to scan and preprocess the raw data with small Python scripts and simple Scala programs. Although one could write custom input format handlers for Spark, it is more convenient to have each datapoint in one line, cause then Spark can automatically distribute the file to multiple computing nodes from start.

The formatted data after the preprocessing is:

Plot:

```
MovieName|PlotDescription
```

Actors:

```
ActorName|MovieName
```

Key words:

```
MovieName|KeyWords
```

3.4 Apache Spark Pipeline

Spark ML[3] represents a workflow as a Pipeline, which consists of a sequence of PipelineStages (each stage is either a Transformer or an Estimator) to be run in a specific order. For Transformer stages, the transform() method is called on the dataset and the input dataset is modified as it passes through each stage. For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer's transform() method is then called on the dataset.

The top part of Figure 2 shows the 3-stage pipeline we used in this project. These 3 stages are all transformers and

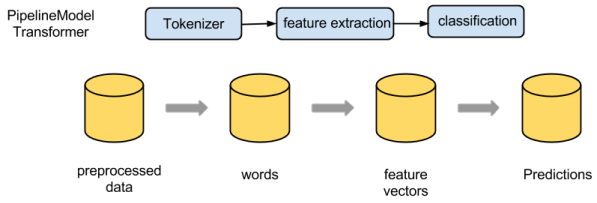

Figure 2: movie genres classification pipeline

generate a model at the end. Each stage's transform method updates the dataset and passes it to the next stage. The bottom row represents data flowing through the pipeline, where cylinders indicate DataFrames. The Tokenizer.transform() method splits the raw text documents into words, converting the dataset of texts to a dataset of vectors of words. The feature extraction stage converts the word vectors into feature vectors. In our experiments we split the data into a training and testing dataset to be able to validate our model on a separate dataset which wasn't used for training. The train method of the learning algorithm is called with the training data set to produce a model. After that this model can be used to predict a classification for items of the testing dataset via the predict method.

In the feature extraction stage, we use different feature extraction methods for the 3 different input data categories (plot, actors and key words). For plot input, HashingTF() and IDF() is applied to generate the feature vectors. For the other 2 kinds of input data, since there are no duplicated words, we simply use binary n-dim-vectors to represent whether a actor(or key word) appears or not.

3.5 Feature Extraction

For the plot description based classification, we used TF-IDF[4] to vectorise the feature.

Term frequency-inverse document frequency (TF-IDF) is a feature vectorisation method that is intended to reflect how important a word is to a document in a collection of the corpus. The TF-IDF value increases probability to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general but provide few information.

Denote a term by t, a document by d, and the corpus by D. Term frequency $TF(t,d)$ is the number of times term t appears in document d, while the document frequency $DF(t,D)$ is the number of documents that contain term t in corpus D. However, if we only use term frequency, for some common words, like "the", it will tend to incorrectly emphasise the documents which happen to use "the" more frequently. But these kind of words can't provide enough information to distinguish relevant and non-relevant documents and terms. If a term appears very often across the corpus, it means it doesn't carry special information about a particular document. Thus, we use inverse document frequency, a numerical measure to show how much information a term carries:

$$IDF(t,D) = \log \frac{|D|+1}{DF(t,D)+1},$$

where $|D|$ is the number of documents in the corpus. Since a logarithm is used the IDF value of a term, which appears in all documents, becomes 0. Note that a smoothing term is applied to avoid dividing by zero for terms outside of the corpus. The TF-IDF measure is simply the product of TF and IDF:

$$TFIDF(t,d,D) = TF(t,d)IDF(t,D).$$

In the implementation of TF-IDF in the Apache Spark machine learning library, a raw feature is mapped into an index(term) by using a hashing function. Then term frequency is calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1048576$

By using TF-IDF, we generate a vector for every plot description.

3.6 Classification

In machine learning, support vector machines(SVMs)[4] are supervised learning models used for binary classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

An SVM model constructs a hyperplane in high(or infinite)dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalisation error of the classifier.

Given some training dataset $\mathcal{D}$ containing n points

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)|\mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1,1\}\}_{i=1}^n$$

where y_i is either 1 or -1, indicating the class to which $\mathbf{x}_i$ belongs. Each $\mathbf{x}_i$ is a p dimensional vector. If the data is linearly separable, we can find a hyperplane:

$$\mathbf{w} \cdot \mathbf{x} - b = 0$$

to separate the dataset. By determining the distance of a data point to the hyperplane $|\mathbf{w} \cdot \mathbf{x} + b|$, which is either positive or negative, we can see which class the point belongs to.

To find the maximum margin to divide the dataset into 2 classes($y = 1$ or $y = -1$), we can select two hyperplanes in a way that they separate the data and there are no points between them, and then try to maximise their distance. The region bounded by them is called "the margin". These hyperplanes can be described by the equations

$$\mathbf{w} \cdot \mathbf{x} - b = 1$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = -1$$

By using geometry, we find the distance between these two hyperplanes is $\frac{2}{||w||}$, so we want to minimise $||w||$. As

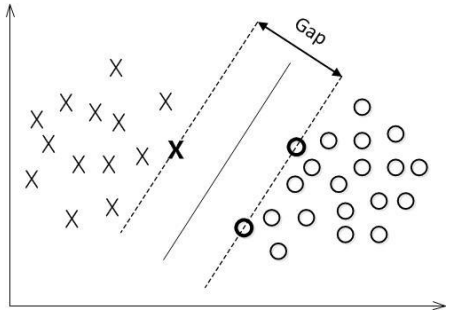

Figure 3: margins for an SVM trained with samples from two classes

we also have to prevent data points from falling into the margin, we add the following constraint:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x} - b \geq 1, y_i = 1 \\ \mathbf{w} \cdot \mathbf{x} - b \leq -1, y_i = -1 \end{cases}$$

This can be written as:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, 1 \leq i \leq n$$

More clearly, the constraint problem can be written as:

$$\arg\min \frac{1}{2}||\mathbf{w}||^2$$

subject to (for any $i = 1, ..., n$)

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1$$

After getting the $\mathbf{w}$, given a new data point, denoted by $\mathbf{x}$, the model makes predictions based on the value of $\mathbf{wx}$. By the default, if $\mathbf{wx} \geq 0$, the outcome is positive, otherwise negative.

Since SVMs only perform a binary classification but we have the requirement to classify movies into multiple genres we trained one SVM per genre and feature. Our internal model of a movie therefore consists of 3 parts: its title, a 24 dimensional binary vector with the genre information from the IMDb and array of n-dimensional feature vectors - one vector for each feature. Thus we trained 24 SVMs per feature vector. If we use more than one feature at a time we needed to train even more SVMs.

In the testing phase the SVMs were used to generate a predicted 24 dimensional genre vector, which then could be compared to the original genre vector derived from the explicit labelling in the IMDb data. Only if both vectors are identical a movie is considered to be completely correctly classified.

4. EXPERIMENT

We started our experiments by using the plot descriptions and performed multiple runs with different number of learning iterations for the SVMs. We started with 5 training iterations and increased that number by 5 until we reached a maximum of 100 training iterations. For every training and testing run 70 % of the data was used for training and the remaining 30 % were used for testing to see how good the learned models classify unseen data. After the plot descriptions we used other features of the movie to see whether we could do better. We also combined several features.

The first approach was to use the plot description and transform it to a TF-IDF vector for each movie. For this approach we could use 249761 movies, which had a plot description as well as a genre label. We also tried to tune the IDF calculation by setting a min-doc-frequency. With this we could exclude words which do not appear in a minimum number of documents, but this seemed to not affect our results significantly so we didn't perform a detailed analysis of the effects of this parameter.

The second approach was to use keywords of the movies, which were explicitly assigned by the maintainers of the IMDb. This provides us a similar textual representation of the content of the movie but with the advantage of being already filtered by a human and also containing words that do not appear in a typical plot description but might describe the type of the movie quite well. We didn't perform a TF-IDF transformation here and just constructed a vector per movie containing a 1 if it has a certain keyword and 0 if not. For this approach we could use 395647 movies, which provided a set of keywords and a genre label.

The third approach was to create a connection between actors and the genre of movies they usually appear in. This feature isn't content based but could be of significance cause we assume that most actors stick to one kind of genre during their carer. Similar to the keywords approach we didn't use a TF-IDF transformation and just binary n-dimensional vectors to represent the fact whether an actor appeared in a movie or not. Here a dataset of 693520 movies, which provided a list of actors as well as a genre label, was available. We only tested it with the actors data and not on with the actress data as well, cause our results with this feature weren't that satisfying and we decided to spend the time on other research directions.

As another experiment we combined all 3 features. Therefore we constructed a set of 3 feature vectors - one for plot, one for keywords and one for actors - per movie. Now we trained 3 SVMs per genre - one for each feature vector. In the testing phase we feed the feature vectors to the SVMs and build a simple average over the results of all 3 SVMs per genre. Cause we could only use movies which provide valid feature vectors for all 3 features, we could only use a dataset of 103536 movies.

This experiment raised the question whether a combination of features really leads to an improved classification or if this is only an effect of higher quality data caused by a smaller subset of the data. To validate this we ran the single-feature approaches and two-feature combination approaches on the reduced subset of the data that provides all 3 features for every movie. We trained the SVMs over 50 iterations for this experiment.

See Figure 4 for a comparison of the dataset sizes we were able to use for the different features.

For all experiments we tried to classify the movie in 24 genres - see Figure 1. We didn't consider the genre Short in our computation and analysis since it's a very large and fuzzy genre which is basically only based on the length of a movie but the goal was to create a content related classification. Also the genres Lifestyle, Erotica, Experimental, Commercial & Film-Noir weren't used cause they only contained a very small number of movies - sometimes only one.

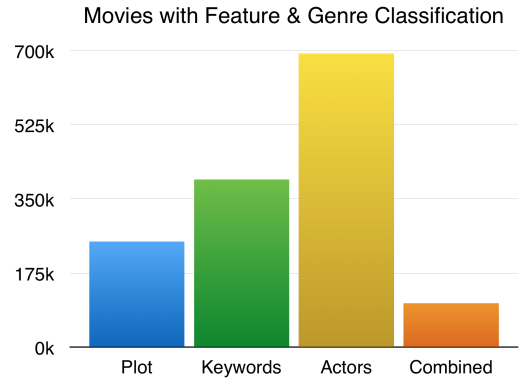

Figure 4: number of movies with a certain feature and a genre labeling

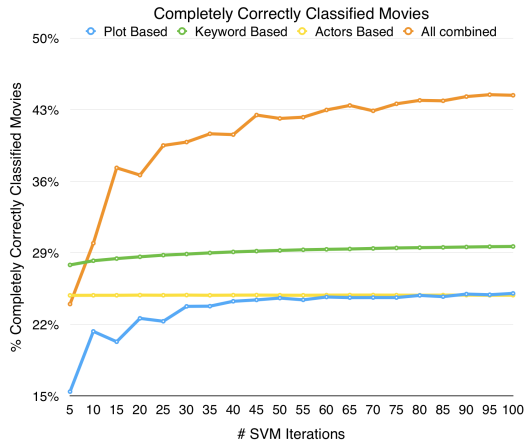

Figure 5: completely correctly classified movies based on different features and different number of SVM traning iterations

The genres Game-Show and Talk-Show weren't used cause they are not classical movies in the sense of having a plot, professional actors etc..

5. RESULTS

5.1 Plot Description Based

Using the plot description (see Figure 6 for error rate graphs) as the only feature wasn't really successful. Although the portion of erroneous single classification operations was at around 6 %, there were many movies which weren't classified correctly. This is caused by the fact that we use 24 SVMs - one SVM for each genre. Most of these SVMs answer with a correct prediction - in most cases with false, cause a movie usually only belongs to a small number of genres. But it is enough that one of those SVMs answers with a wrong value and the movie ends up misclassified.

Another interesting observation is that the amount of errors for the genres Drama, Comedy and Documentary is significantly higher than for other genres. Those 3 genres are the most common ones. One could expect that the SVMs are trained more precisely cause of a larger amount of positive training examples and a therefore more even distribution of negative and positive data points. But it seems like that the larger amount of movies in those genres leads to more diverse plot descriptions and a more fuzzy definition of the genre.

When looking at the development of false-positive and false-negative classifications over increasing number of training iterations, one can see a strong oscillation for Drama. With 5 iterations the false-positive rate is low and the false-negative rate is high, for 10 iterations the false-positive rate is high and the false-negative rate is low. The oscillation and error rate reduces with increasing number of training iterations but continues to be visible even at around 100 iterations. A similar pattern can be seen for the Comedy genre, although the oscillation isn't that distinctive.

For most of the other genres false-positives aren't an issue until 10 to 15 or even until 35 iterations. This is most likely due to the small amount of positive training examples which leads to a strong tendency of always answering with no.

The false-negative rates are quite stable with a slight tendency to a lower error rate in most cases. But there are also some genres like Romance which have a constant increase in the error rate with increasing number of iterations. The most significant improvements can be seen between 0 and 25 iterations.

If we look at the overall error rate there's usually an improvement with increasing number of training iterations. It seems to be stable after around 50 iterations. An reduction of false-negatives is often consumed by an increase in false-positives, leading to nearly horizontal lines for some genres. The alternating false-negatives and false-positives for genres like Drama or Comedy even themselves out and lead to a continuous curve in the total error rate.

Figure 5 shows the number of completely correctly classified movies. It increased nearly continuously with increasing number of iterations - as one would expect it to be after seeing decreasing error rates. Interestingly there are small drops in the curve at 15 and 25 iterations. Those are the same spots where the Drama false-negative rates are high but the Drama false-positive rates are low. The curve increases significantly until 30 iterations with 23.7 % accuracy and only slowly develops to 25 % accuracy at 100 iterations.

Overall the approach wasn't satisfying with only $\sim$ 25 % completely correctly classified movies and in general the following rules hold:

- The more SVMs iterations the more precise is the classification.
- The more movies belong to a genre the more errors in classification will happen.
- For reasonable results one can stop training after around 50 iterations.

5.2 Keyword Based

In the next experiment (see Figure 7) we used the keywords as the only feature. The overall result was slightly

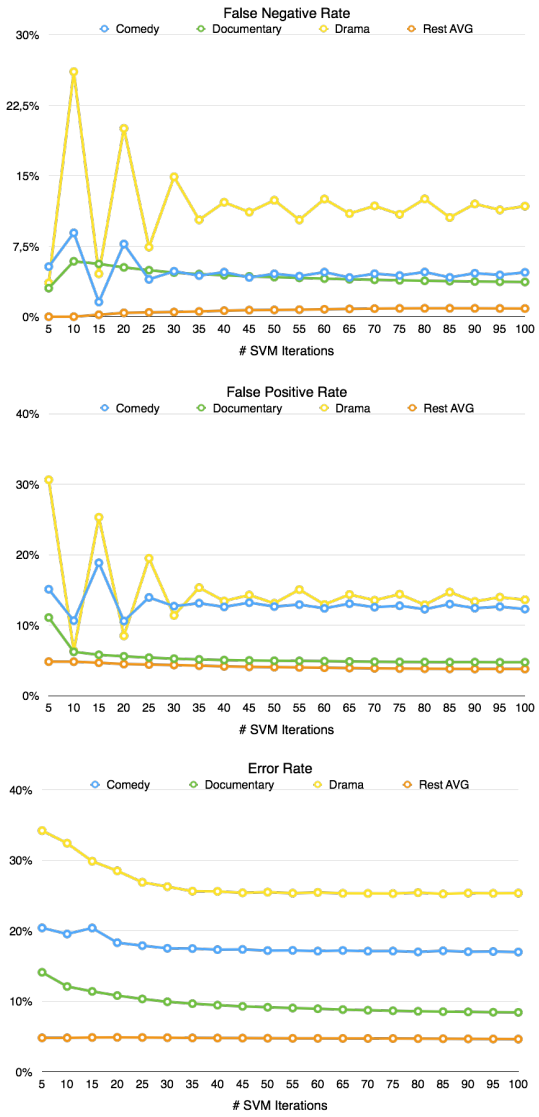

Figure 6: error rates for selected genres using plot descriptions as the only feature

better with 27.8 % to 29.6 % completely correctly classified movies. As for the plot based classification the genres Drama, Comedy and Documentary were the most difficult ones to classify. With false-positive rates of 13 % to 19 %. All other genres had false-positive rates below 7 %. The false-positive rate for Comedy was higher than for all other genres. Interestingly the false negative rates were all below 2 %, except for Drama which was between 10 % and 11 %. Nevertheless those rates were much better than for the plot based classification approach. For a higher number of iterations the same issue of increasing false-negatives rates were observable. The phenomena of alternating high and low false-negative and false-positive rates couldn't be observed in this experiment.

In the overall error rates Drama was the most difficult to classify ($\sim$ 24 % error) followed by Comedy ($\sim$ 19 %) and Documentary($\sim$ 12 %). All other genres where quite close and below 7 % error rate. The error rate and also the rate of completely correctly classified movies didn't improve significantly with increasing number of iterations. Again most improvements in false-positives rates were neutralised by increasing false-negative rates. Only a small improvement between 5 and 20 iterations was notable. The most significant improvements were observable for the 3 difficult genres Drama, Comedy & Documentary.

5.3 Actors Based

The third experiment (see Figure 8) was to only use the actors data as the feature vector. This experiment didn't perform much better than the plot based classification and lead to nearly the same rate of completely correctly classified movies of about 24.8 %.

The error rates stayed nearly constant and also the rate of completely correctly classified movies didn't change with an increasing number of training iterations. Cause of that this approach delivered better results for a low number of iterations and was faster in the training phase cause of a lower dimensionality of the feature vectors. As in the previous two experiments the genres Drama, Comedy and Documentary were the ones with the highest error rates.

5.4 Combined Features

As the next experiment (see Figure 9) we combined all 3 features and averaged the results to obtain a classification per genre. This lead to significantly better result of up to 43.9 % completely correctly classified movies. The disadvantage of this approach is the long training phase. For every feature and every genre the systems trains a different SVM, which leads to 72 SVMs that need to be trained.

The rate of completely correctly classified movies increased significantly with the number of training iterations. The largest increase is from 5 to 15 iterations - going up from 23.9 % to 37.3 % - and after about 45 iterations it only increases a little. Similar to the plot based approach there are some small throwbacks at 20, after 45 and at 70 iterations but overall it still increases.

The error rates are a lot better than for the single feature approaches, but still the genres Drama, Comedy and Documentary are outliers compared to the other genres. In this combined approach Drama isn't the worst genre anymore and provides a quite stable overall error rate. Now Comedy is the most difficult genre. The error rate of documentary improved so far that it is now below the average

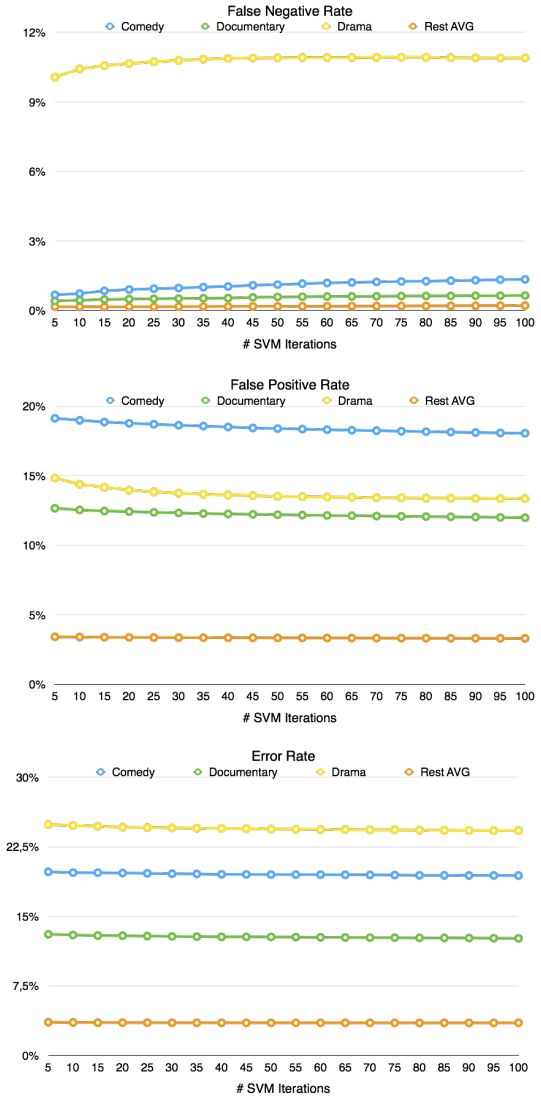

Figure 7: error rates for selected genres using keywords data as the only feature

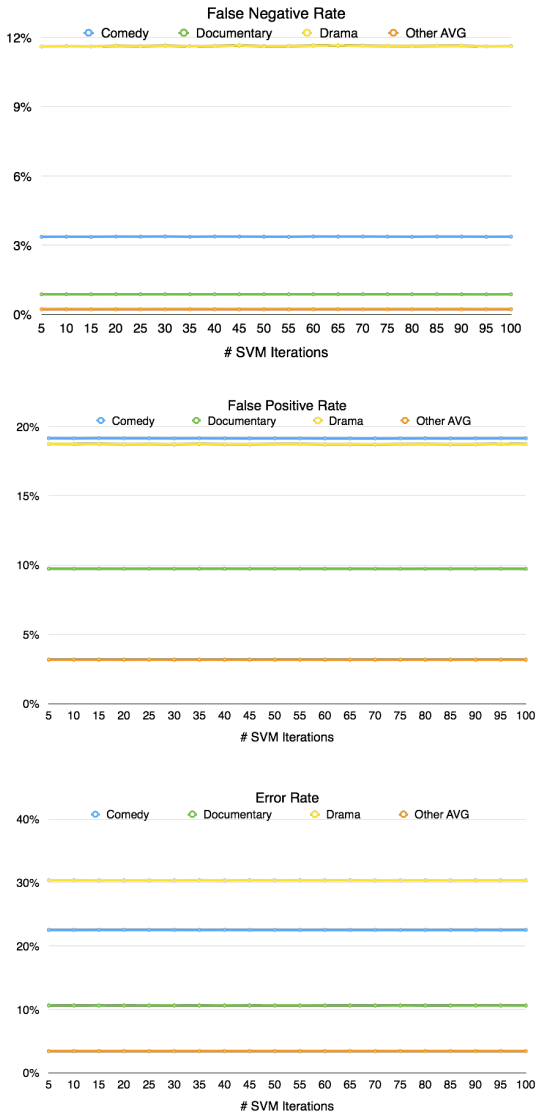

Figure 8: error rates for selected genres using actor data as the only feature

of the other genres. As expected the largest improvement happens within the first 20 iterations. Especially Comedy improves it's error rate from above 20 % to below 14 %.

Looking at the false-positive and false-negative rates in detail shows some interesting phenomena. The false-negative rates for all genres except Drama (1.5 % to 5 %) are really low (below 0.5 %) and increase only slightly with the increasing number of iterations. The false-positive rates for Drama and Documentary are quite similar or sometimes even below the average of the other genres. But Drama and Comedy have an alternating increase and decrease in the error rates similar to the curve of Drama for the plot based approach. For Drama the alternating pattern smoothes the overall error rate but for Comedy the false-positive rate is orders of magnitude larger than the false-negative rate so that this zick-zack pattern is still visible in the overall error rate.

The reason for the zick-zack pattern isn't obvious and would need some more investigation. The results are much better than using just one feature at a time. The question arises whether this is due to an actual improved algorithm or just a better quality of the input data. Since we're only using movies that provide all 3 features we do not have so many training and testing samples, but they might have a higher quality since they have more features.

Running the single feature algorithms on the subset of the data that provides all 3 features for each movie we could validate the hypothesis that a combination of features leads to better results and that it's not just about a difference in the quality of single features. Figure 10 shows the percentage of completely correctly classified movies when using the reduced dataset of about 100000 movies and training the SVMs with different feature combinations with 50 iterations.

Surprisingly the single feature approaches perform worse on this subset than on the other datasets. Combining one other feature with the plot description feature already achieves a precision of over 36 %. Using all 3 features it was possible to classify about 42 % of the movies correctly.

With this experiment we proved that having more features for a movie doesn't mean that those features have a higher quality in terms of providing a better SVM based classification. But combining multiple features increases the accuracy significantly.

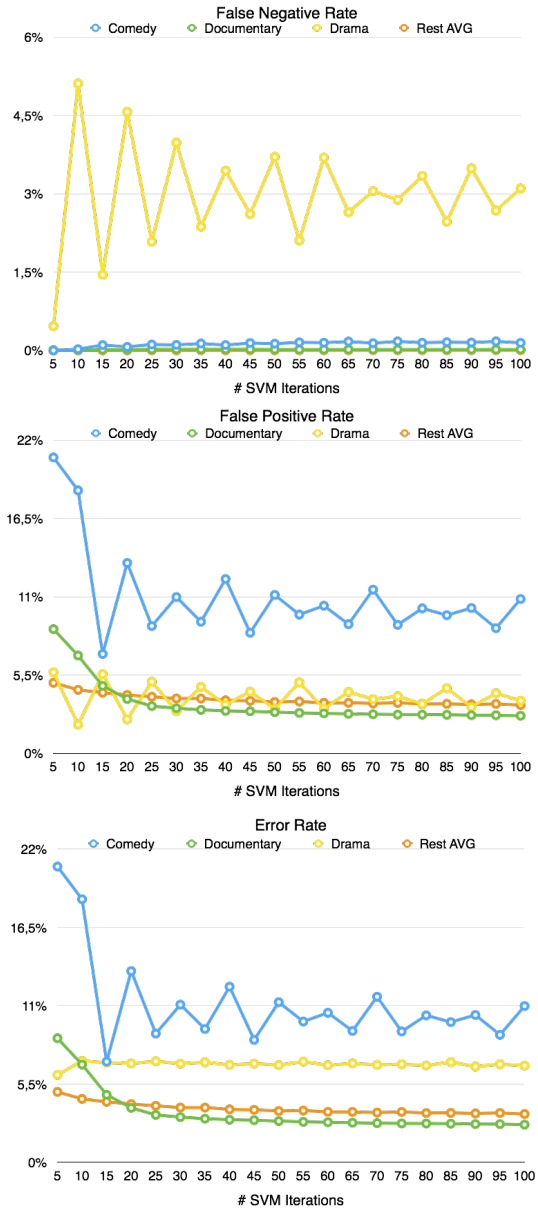

Figure 9: error rates when using all 3 features at the same time for classification

6. CONCLUSION & FUTURE WORK

Overall our implementation couldn't fulfil our goal of a reliable classification system for movie genres. We have seen some improvements by trying different features and especially using feature combinations worked better.

The quality of the data is an important factor for machine learning algorithms since bad, insufficient or even misclassified data can ruin the learned model. Having data with more features improves the quality of the prediction. We could prove that this isn't caused by just having a subset of the data with single higher quality features. Using only single features for classification on the same subset of the data performed worse and therefore the combination of features was the key element for the improvement. Looking at the rate of correctly classified movies one should note that we cannot assume for sure that every movie was labeled with every genre it might belong to. Therefor the false-positives could also be interpreted as suggestions to add missing labels.

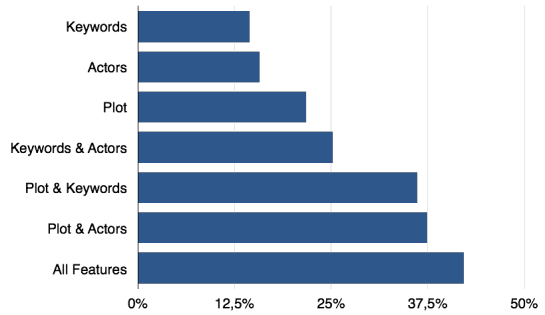

Figure 10: completely correctlt classified movies using different feature combinations on the subset of the movies that provide all 3 features - trained with 50 training iterations

Seeing large error rates for, from a content perspective, very diverse and fuzzy genres like Drama, Comedy & Documentary makes sense since they contain a large number of movies and even for humans it might be difficult by just looking at the provided features to decide for a genre classification. For other more specific genres like Western or Fantasy the success rates were over 90 % and based on that one could try to analyse the generated models to derive an answer to the question "What makes up a genre?".

Working with SVMs as machine learning component worked out quite well and was easier to set up than Neural Networks, which we used in other projects. One should note that increasing the number of training iterations usually increases the precision but that doesn't always hold. After all one has to choose between the training time & cost and the gain in precision. As we've seen with some of the zick-zack curves it is possible to end at a very suboptimal point on the curve.

Those zick-zack curves are also a mysterious outcome of our experiments and it would be interesting if this is due to properties of the SVM or of our features and how it behaves in the gaps between our 5 step iteration jumps. Unfortunately we didn't had time to investigate this.

Another interesting behaviour of the SVMs is the increasing false-negative and decreasing false-positive rate for most genres and features. Based on this observation one could choose whether to have more errors of the one class than of the other. This can be relevant for some domains where one class of errors is considered to be worse than the other - e.g. a not detected attack vs. a false alert in a security domain.

Using Apache Spark for our experiments was interesting and it's APIs proved their usefulness. The MLlib provided everything we needed and was easy to use. We didn't encounter any bugs. Using Spark as someone who is familiar with Apache Flink[5] everything was quite self explaining and similar, although they use quite different mechanisms under the hood. We also considered Apache Flink for this project but cause of it's unstable machine learning API, which was also missing an IDF transformer at the beginning of this project, we preferred Spark, which is more battle proven and therefore more useful for a project under heavy time pressure like this.

The only thing we couldn't do in neither Spark nor Flink, was the preprocessing of our data. Since it doesn't come in a simple format like CSV and also includes some headers we needed to perform some preprocessing via Python scripts and small Scala programs.

Our dataset wasn't real 'Big Data', although we crossed the giga-byte limit all calculations could be performed on a single consumer machine. Training and testing with all 3 features on the whole dataset over 100 iterations took less than an hour on a recent quad-core machine with 16 GB of ram. But from an academic point of view it was very interesting to work with those 'Big Data' systems and get some hands-on experience with real data and build a scaleable solution.

7. REFERENCES

[1] *Internet Movie Database*, http://www.imdb.com
[2] *Apache Spark 1.4.0*, http://spark.apache.org/docs/1.4.0/
[3] *Apache Spark MLlib*, http://spark.apache.org/docs/1.4.0/mllib-guide.html
[4] Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.
[5] *Apache Flink*, http://flink.apache.org

The code for this project as well as the raw result data can be found on https://github.com/FGoessler/AIM3Project.