Report of TC3

Movie Genre Classification with Plot Summary

YANG Mo, ZHANG Xudong

21 février 2018

1 Individual Contributions to the Group

1.1 Mo Yang

I do part of data prepossessing, part of algorithms implementation, evaluation and continuous amelioration.

1.2 Xudong Zhang

This project is completed with close corporation by Mo Yang and me. Close discussion and corporation exist through whole parts of the project.

2 Introduction

In this project, we try to realize the a system which can classify the film into different genres according to their plot summary. The idea is to use movie genres as labels and we try to extract features for labels with plot summary. When treating problems with texts, the problem is that there are too many different words, which means there will be too many potential features. However, too many features will lead to over-fitting. The solution to this problem is to find out words which contain most information. In this project, we try to select features with the idea of Chi-Square and Information Gain. Then we compare the performance of these two feature selection strategies with different algorithm of machine learning. The details of this project is explained in next several sections.

3 prepossessing

In this project, we use the CMU Movie Summary Corpus for training and testing. Particularly, we use two files in this data set : $movie.metadata.tsv$ and $plot_summaries.txt$. The first file contains the general information of movies like Wikipedia movie ID, Movie name, Movie genres ect. The second file contains the Wikipedia movie ID and the corresponding Plot summary. The necessity of prepossessing is that we can not use directly these two file for feature extraction. Information in this dataset is obtain from website and it is not clean. For example, the number of different movie genres is around 300. There are different expressions for one movie genre and some genres contain a extremely small scope of film and of course it can be classified into one more general genre. Also for the plot summary, we can not use directly use it for classification.

Although it may be a little fussy, prepossessing is really important for classification for texts. In the next several parts, we state the details of prepossessing for different parts.

3.1 Preparation

In this project, we realize all the function under *Python*3. The first step is to read in the data. Here we read the data into DataFrame as it is extremely convenient to use. The function we use for reading in data is shown below.

```
variable = pandas.read_csv(address_of_file,sep='\t',
header=None,names=name_of_column)
```

After reading in the data, we can find that in movie.metadata not all movies have genres. So we need to delete the movie which does not have explicit genre. After that, we can find that not all movies with genres have plots. Luckily, the two variables which contains movie metadata and movie plot separately have one common column–Wikipedia movie ID. We can use the function below to merge the two DataFrame into a big one, which will only keep movies existing in the two DataFrame.

```
variable=pandas.merge(movie_metadata,movie_plot,on='Wikipedia movie
    ID')
```

Now we can get everything we want from this big DataFrame.

3.2 Normalization of Plot

Normalization of string is consisted of four steps. First, we put the letters of string into lower form. Then we tokenize the string into words. After that we use lemmatization to put the word into its original form. Finally, we use stemmer to get the stem of words.

Here we explain the reason for normalization of plot. Now our purpose is to extract features from plot, i.e. words which can best represent its genres. So we do not care whether the word is in upper form or lower form. Also we do not care whether the word is in plural form or single form, etc. So we put all the words into lower form, then transform them into the original form. In this way, we can decrease the number of different words, so as the number of features to extract. Moreover, this treatment is reasonable. For example we do not need to differ between the word lie and the word lying.

3.3 Clustering of Movie Genres

In this part, we try to group the movie genres. The problem in the original file is that the number of different genres can be as many as around 300 and of course it is not reasonable to classify films into 300 genres. Moreover, if we dig into the data, here we count the number of films under each genre. We can find that for some of the genres, the number of films under them is less than 10. So this also demonstrate that we do not need to consider those genres which contain a extremely scope of films. Also it is not hard to re-classify those genres into a more general genre.

Here in this part, our method to cluster the films genres is somehow empirical. Another method which is more objective will be presented afterwards.

In the beginning, we count the number of films under each genre and sort the list according in descending order. Among the first several genres, we choose 18 of them according to our experience for movie genres and we also refer to the films genre in IMBDA. Then we cluster the film genres to these more general genres. After some tedious data processing like deleting films without genres, we finally get a relatively clean DataFrame.

3.4 Build Multilabels of instances

So far we have obtained clean data, in the next step we are going to transform these data into binary vectors. In sklearn, the way how computer learn multiple labels of each instance is to have a n binary indicators vector to indicate the presense of a label. Here is the code to build the vectors.

```
# get genres information from dataframe
genres=[]
for index,row in table.iterrows():
    genres.append(row['Movie genres'])
# build multilabels format
from sklearn.preprocessing import MultiLabelBinarizer
mlb=MultiLabelBinarizer()
Y=mlb.fit_transform(genres)
```

Listing 1 – build multilabel of instances

4 Feature extraction

The objective of this part is to extract features for learning. In our case, we want to extract features from a movie text plot which help us to classify which genre it belongs to. That is what we do at first. We extract the content of summary plot for each movie and restore them in a list for further treatment :

```
import re
content=[]
for index,row in table.iterrows():
    content.append(' '.join(row['Plot summary']))
```

Listing 2 – extract text content from table

So we get the texts, but we need to work with numbers. What we need is a matrix which we have seen in tp3 : *Bag of words*. The basic idea of it is that we get every individual word in our dataset, so that we can represent each text by a vector with binary values to represent weather the text contain a given word or not. For example, we have n individual word in a dataset, then we can represent each text as a $n * 1$ vector, where each position in the vector corresponds to the presence or absence of a particular word.

we use the function in sklearn *CountVectorizer()* to do trick. Also, we set the parameter *max_ df* to 0.95 and *min_ df* to 0.005 to get rid of words occur either too many or too few in the text. Cause the words only occur in one document

might be names or other particular rare terms. Words occur in most documents are probably stop words, those aren't good descriptors which contribute to the task of genres classification.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorize=CountVectorizer(max_df=0.95, min_df=0.005)
X=vectorize.fit_transform(content)
```

Listing 3 – vectorization of texts

If we take a look at the structure of the vector :

```
print (X.shape)
>>> (39445, 3906)
```

Listing 4 – Shape of vector

We will see that we have 39945 samples and each movie's overview gets represented by a 1x3965 dimensional vector.

We are almost ready for learning If we want the presents of these words to be features to fit learning algorithms. But we want to further more reduce the dimension of features to avoid 'Curse of dimensions'. What we want is to select only the most distinctive words for a certain genre. The first method come into our mind is to use *TF-IDF*, while *TF-IDF* basically measure the importance of the word for the document it belongs to (according to the definition of *TF-IDF* which we have seen in tp3). After some research we have found 2 indicators to select proper features : *chi-square* and *information gain*.

The Chi-square test is used in statistics to test independence between events. More specifically in feature selection we use it to test the dependence between the occurence of a certain word and the genre the text belongs to. We can evaluate each term and reorder them to select the most contributing words to genres classification.

Information gain is yet another technique to help extracting features for text classification. This measures how much information the presence or absence of a particular term contributes to making the correct classification decision.

In this project we use *chi-square test* to select 1000 best words as feature vector :

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
Selec = SelectKBest(chi2, k=1000)
Selec.fit(X,Y)
X_new = Selec.transform(X)
```

Listing 5 – Select 1000 best words according to chi-square score

So far we have finished the extraction of features.

5 Implementation of different machine learning models

In former work we have already done collecting dataset for model training (including features and labels). the object of this part is to process model training with several representative algorithms and evaluate their results. Before implementation we choose to divide the instances at hand into 2 parts : training_set and test_set.

```
msk = np.random.rand(X_tfidf.shape[0]) < 0.8
X_train=X_new[msk].toarray()
X_test=X_new[~msk].toarray()
Y_train=Y[msk]
Y_test=Y[~msk]
```

the two parts will be used seperately for model training and model evalutation.

There are lots of models suitable for multilabel-classification. For example, there are :

1. Generalized Linear Models
2. SVM
3. Neural Network
4. Decision Tree
5. Random Forest

Or some approches bayesian :

1. Naive Bayes
2. Linear or Quadratic Discriminant Analysis
3. Bayesian Hierarchical models

The list is endless. What we do is that we try to implement some representative approaches in machine learning and evaluate their performance in our case. We start from the most basic algorithm bayesian : Naive Bayes. We use directly the function in Sklearn (so as for all algorithm implementations in this project). Sklearn offer a report about performance of the classifier :

	precision	recall	f1-score	support
Action	0.45	0.68	0.55	1369
Adventure	0.42	0.69	0.52	1180
Animation	0.46	0.69	0.55	513
Comedy	0.55	0.60	0.57	2553
Crime	0.45	0.67	0.54	965
Documentary	0.19	0.81	0.31	203
Drama	0.65	0.79	0.71	3911
Family	0.43	0.58	0.49	659
Fantasy	0.27	0.61	0.37	404
Horror	0.58	0.72	0.64	810
Music	0.20	0.61	0.31	554
Mystery	0.24	0.58	0.34	417
Romance	0.40	0.69	0.51	1497
Science Fiction	0.34	0.75	0.47	455
Sports	0.36	0.77	0.49	151
Thriller	0.46	0.61	0.52	1288
War	0.31	0.82	0.45	306
Western	0.36	0.81	0.50	203
avg / total	0.48	0.69	0.56	17438

FIGURE 1 – report for Naive Bayes

Similarly, Desision Tree, Random Forest, SVM, Shallow Neural Network are implemented and tested. We resume the result of each model in the table below :

Different algorithms and their results

Score indicator / Algorithm	Precision	recall	f1-score	Executing time
Naive Bayes	0.48	0.69	0.56	0 :00 :03.652172
Decision Tree	0.51	0.31	0.37	0 :00 :21.477360
Random Forest	0.66	0.20	0.23	0 :00 :06.045194
Multilayer Neural Network	0.62	0.42	0.49	0 :01 :29.351357
SVM	Too slow, can not be completed			

From the result we can see that the result of Naive Bayes is significantly outstanding at least time cost. Multilayer Neural Network works also pretty well. Decision Tree and Random Forest get pretty low score and SVM nearly doesn't work at all. We have found out Naive Bayes is the optimal algorithnm for extremely outstanding in term of fast and accurency.

6 Continious Improvement

So far we are not yet happy with the result we've got. When implementing different algorithms and optimizing parameters can no longer augmenting the

performance of classification, we start to think of the classification task itself, how reasonable it is when we define different genres of movies ? What we want is the movie summary plots can be classified into different genres which are most likely *'Independent'*. Note that a movie can be associated with more than one genres, We found that some genres occurs more often together than others. For example, it is intuitive for us that 'Adventure' and 'Action' is often overlapping as labels. A movie can be both 'Drama' and 'Comedy' and actually that is quite common. Conversely, you can hardly imagine a movie can be both 'Thriller' and 'Sport'. It is important to know that when we do machine learning, especially for those probability based algorithms, we have assumed that all variables are independent as well as classes. We believe that such inherent biases will block us to improve the precision of prediction further more. So it makes sense to find out which genres occurs more likely together. That is exactely what we do next.

6.1 Pairwise analysis of Movie Genres

The main purpose of this step is to find out which genres occurs most often together in the same movie. First we are dedicated to find out all pairs of genres in each instance. Note that we should also take care of single items and duplicate them as pairs for comparison.

```
#now start processing the plot(now the plot is fairly clean)
#first step: we want to find the co-relation between the genre of
    film from (haverd deep learning)
import itertools
def list2pairs(l):
    # itertools.combinations(l,2) makes all pairs of length 2 from
    list l.
    pairs = list(itertools.combinations(l, 2))
    # then the one item pairs, as duplicate pairs aren't accounted
    for by itertools
    for i in l:
        pairs.append([i,i])
    return pairs
allPairs = []
for index,row in table.iterrows():
    allPairs.extend(list2pairs(row['Movie genres']))
list_of_genre = ['Drama','Comedy','Romance','Action','Horror', '
    Thriller','Crime','Documentary','Adventure','Family','Music','
    Animation','Mystery','Science Fiction','Fantasy','War','Western
    ','Sports']
```

Listing 6 – Finding all pairs

Then we visualize the result in heat map :

```
visGrid = np.zeros((len(list_of_genre), len(list_of_genre)))
for p in allPairs:
    visGrid[list_of_genre.index(p[0]), list_of_genre.index(p[1])
    ]+=1
    if p[1] != p[0]:
        visGrid[list_of_genre.index(p[1]), list_of_genre.index(p
    [0])]+=1

```

```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.heatmap(visGrid, xticklabels=list_of_genre, yticklabels=
    list_of_genre)
plt.show()
```

Listing 7 – Finding all pairs

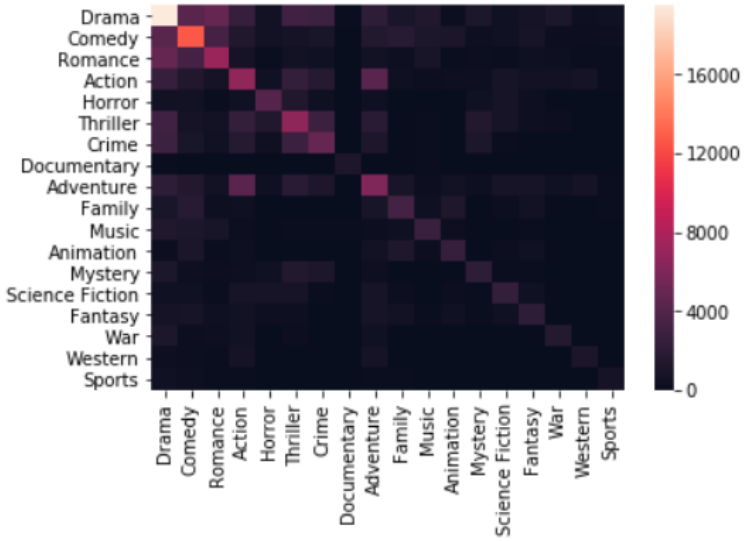

FIGURE 2 – genres co-occurance heat map

As we can see from the plot above, the elements in diagnol line represent the occurance times of each genres(as they were counted as duplicated pairs). The rest blocks indicate the occurance times of each pair of genres. We found that there are a lot of 'Drama', which means that 'Drama' can be quite a common label. Conversely, there are nearly no Sports and Western. We can find out many strong co-occurance like 'Comedy-Romance' there are also examples like documentary, which seems like quite distinct. But still, il is not yet clear which genres occurs most often together in the same movie.

To have a better view of that, we use a techinique 'biclustering'. It allows to make cluster of those who are most related to each other so it will show as evident clusters in heat map. After treatment the heat map looks like this :

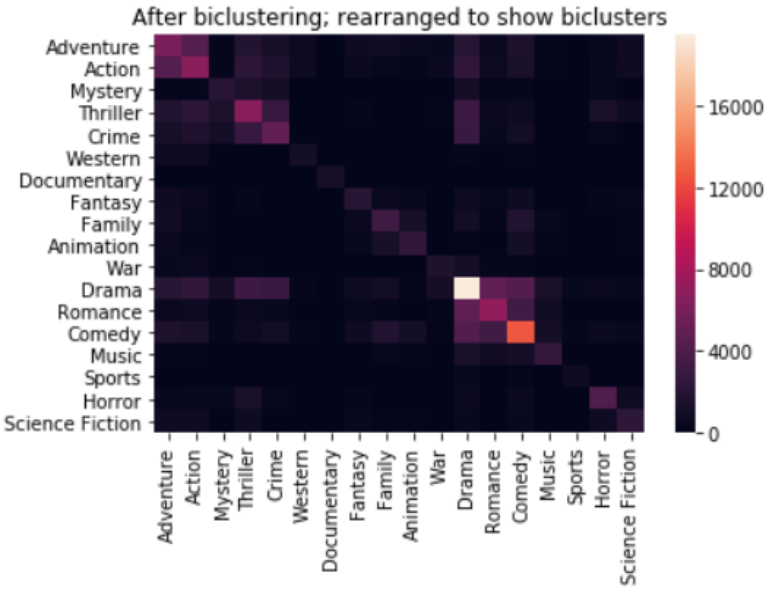

FIGURE 3 – biclustering_heatmap

From this plot we can see that 'Drama', 'Romance', 'Comedy' makes a very evident blocks, which indicate that the three labels occurs most frequently. Additionaly, 'Adventure' and 'Action' , 'Thriller' and 'Crime' also popular pairs. Intuitively that makes sense. So next we are going to group those genres and consider them as a independent unbiased label. Finally we will process the same set of implementation to see whether the results will be ameliorated.

After treatment the results are resumed as the table below :

Different algorithms and their results

Score indicator / Algorithm	Precision	recall	f1-score	Executing time
Naive Bayes	0.61	0.73	0.65	0 :00 :03.333933
Decision Tree	0.52	0.52	0.52	0 :00 :50.384274
Random Forest	0.73	0.46	0.47	0 :00 :08.143200
Multilayer Neural Network	0.73	0.61	0.64	0 :01 :16.138791
SVM	Too slow, can not be completed			

As shown above the result has been significantly ameliorated. It worth a discussion about what is a good classification. So far we have demonstrated that the dependency between labels will hurt the performance of ML classifier, so from the point of view of amelioration we need to design balanced and unbiased da-

taset. But in the real world, we really need those genres like 'Drama','Comedy', 'Romance' even though they are not good labels in supervised classification mathematically. Yet another compromise can be adding sub classification tasks to classify those strongly dependent genres after the first classification.

7 Conclusion

In this project we have developed a complete task of text classification from problem identification to all steps dedicated to solutions, including dataset selection, data cleaning, pre-processing, feature extraction, learning, evaluation, and continious amelioration. We have deeply studied the context, learned how to analyze and process natural language learning step by step. We have also evaluated learning results and think of its potential defects. We have then used some statistic analyst techinques to verify the defect and proposed the solution. Finally we have evaluated the results and the improvement is obvious. This project is not only a great practice on everything we have learned in class, but also gives us an idea about how to analyse a real world problem and to extend the frame for amelioration. There is no doubt that it is always possible to extend the frame we have. For example, we can imagine to implement deep learning, we can have multiple layers of machine learning..... There is no final solutions to problematic we have in the real world, what's important is to get a solution and get a better one if needed.