UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

FACOLTÀ DI INGEGNERIA

Ottimizzazione per Big Data

A.A. 2015/2016

Exam Report

Sentiment Analysis and Event Suggestion

PROFESSORE

Ing. Veronica Piccialli

STUDENTE

Giovanni Balestrieri

Contents

Abstract The growth of content on the web has been followed by increasing interest in opinion mining. Software companies typically register high volumes of digital customer activities every day. Several small businesses rely on the extraction of relevant features from these flaws in order to improve the provided services and the user experience. In order to extract implicit feedback from user's activities, a preliminary filtering has to be applied. This paper presents a method to estimate preferences and draw users' profile based on Data Mining software and python libraries. Furthermore, a voting system will be proposed to identify new events with the highest probability to be appreciated by a considered user.

0 Introduction

0.1 The problem

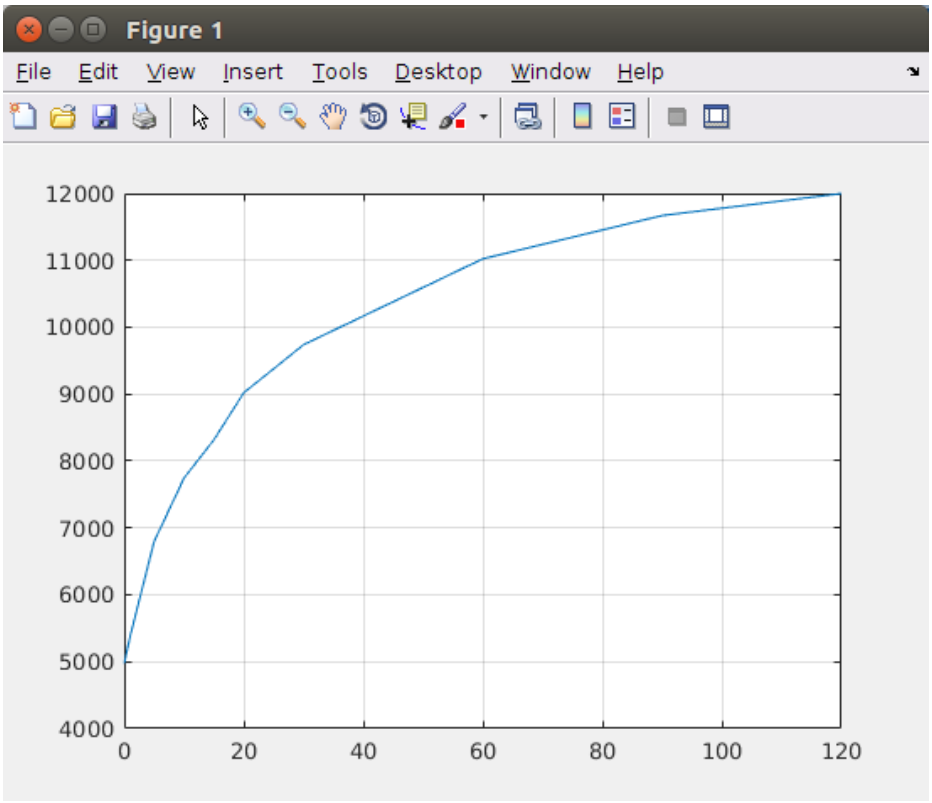
0.2 The ListItem

0.3 Technologies

Data

Our data consists of 255 260 actions performed on the mobile applications. We have identified the 5 most active profiles of 2016 and collected an average of 13300 requested events per user. We excluded incomplete action records

Each record consists in links between the user's id, the action performed in the mobile apps, the time spend on the page and the identification code of the event. There are 9000 titles visualizations for which the time spend on the page is less that 20 seconds.



For each record a query is performed to retrieve additional details of the event such as category, genre, location id, actors, directors and artists.

It is desirable to provide an automatic classification of the upcoming events based on the performed actions in the database so it is necessary to determine whether the record is related to an event that has been appreciated or not.

Training and Test sets

In order to split the data set in Training and Test sets, a date is chosen and the events are stored in different tables. Two scenarios have been considered. In the scenario B, every action performed before march 25^{th} is used to train the classifiers and the rest is used to test the predictions.

Scenario	Date	Training Set	Data Set
A	01/03/2016	8591	4708
B	25/03/2016	10563	2730

In the above table are shown the number of events considered for the training and test sets for three different attempts.

In each scenario, both sets have to be labelled in oder to train and test the accuracy of the models. A set is divided in two classes 'pos' and 'neg' depending on the time spend on page. Recalling that the objective of this work is to filter a set of upcoming events in order to suggest the most interesting titles to the user, three time threshold have been considered. For the β case every action in which the user spent more that 20 seconds is labelled as $'pos'$ and associated to a negative instance when the time on page is less that the considered threshold θ. In the following table are reported the thresholds used in this work.

- α: $\theta = 5$ sec
- β: $\theta = 20$ sec
- γ: $\theta = 40$ sec

The training and test set can be divided in positive and negative instances. Table A shows the number of events for each data set for scenario A and Table B.

θ	Train Pos	Train Neg	Test Pos	Test Neg
α	4962	4827	2740	1968
β	2519	6072	1760	2948
γ	1825	6766	1173	3535

Table A: 01/03/2016

In order to increase the accuracy of the classifier, the data set had been inspected. All the events in the Negative Training set that appear in the positive set have been removed. A similar process has been applied to the Test set. The numbers of instances in the filtered data sets are shown in Table Astart.

θ	Train Pos	Train Neg	Test Pos	Test Neg
α	3764	966	1268	174
β	2519	1785	764	281
γ	1825	2539	493	368

Table Astart: 01/03/2016 (filtered)

θ	Train Pos	Train Neg	Test Pos	Test Neg
α	4962	5607	1542	1188
β	3387	7182	1760	1838
γ	2407	8162	1173	2139

Table B: 25/03/2016

θ	Train Pos	Train Neg	Test Pos	Test Neg
α	4962	1042	1264	92
β	3387	1939	754	187
γ	2407	8162	1173	2139

Table B: 25/03/2016 (Filtered)

All the duplicated in the test set have been removed.

Text Classification

Classification is a supervised data mining technique that involves assigning a label to a set of unlabeled input objects. In this work, we have implemented a binary classification that classifies input objects into one of the two classes.

In text classification, algorithms are applied to text documents. The goal is to assign a document in one class based on its content. In order to train a recommendation system, the input documents have been divided into train and testing data. The training set contains all the labeled documents. Testing data sets are those where the documents are unlabeled. The system learns the model from the labeled data and predicts the class label on the testing documents. This type of learning is called supervised learning because a supervisor defines the classes and labels training documents. The trained recommendation system applies a classification function f that maps the document D to class C.
The dimension of the training and testing data set is very important. If the classifier is fed with a small number of documents to train from, it may not acquire substantial knowledge to classify the test data correctly. If the training data is too large compared to the test data, it leads to overfitting. Even the ratio of positive to negative instances plays a crucial role during the training step.

Text Representation

In order to train the model and compute the classification function, the input set has to be studied and translated in feature sets. Hence, there is a preliminary work to perform on the data set to represent text in a structured manner.
The most common technique to represent text is the Bag of Words model. Note that in this model the order of words is ignored, bigrams or n-grams could be considered to keep track of the spacial property of the words in a sentence.
From the id of the events clicked, a MySQL query retrieves the details of the object. The title, the category and the genre have been added and linked to the action. Other experiments have been carried out considering additional details such as the description, the director, actors, and the location.

Each records has been converted into a string containing a combination of the title, the category and the genre. Then, the class label is appended to the string.

Feature Vectors

The preprocess step brakes the record down into tokens or individual words. This process is also referred as Tokenization and each word represents a feature. For each document a vector of features is determined.
Italian and english stopwords have been removed from the temporary feature sets and features have been grouped by their roots (stemming).
Then all words have been weighted using the their Term Frequency in the training set i.e. the total number of occurrences of a particular word in a collection of documents.

Finally, the feature vector is filled with the N[1] most common words in the training set.

Feature reduction

Feature reduction was an important part of the optimization of the classifiers' performance. Reducing the feature vector to a size that does not exceed the number of training cases was a starting point.

Since the dimension of the feature vector increases with length of the document. Reducing the number of features can be done in several ways:

- reduction to the top n features
- reduction by elimination of sets of features
- reduction by combining the above methods

The proposed approach is to eliminate a subset of features and then take the top ranking n features from the remaining set.
Techniques like stop word removal and stemming are commonly applied. Stop word removal involves the elimination of words which add no significant value to the document. Stemming is the process of reducing derived words to their word stem, base or root form. All the above have been applied using the StringToWordVector function in Weka and implemented in Python.

[1]Several percentages of features to consider in the final vector have been tested. The threshold depends on the size of the training set. In our case the size of the feature vector is 1140 words.

Experimental Results

In this chapter, we present the experimental results for several algorithms. The data sets used are the one described in the Training and Test set section.

Six classifier have been tested and compared.

- 1) Naive Bayes
- 2) SVC with linear kernel
- 3) Multinomial Classifier
- 4) Logistic Regression
- 5) Stochastic Gradient Descent
- 6) Linear SVC
- 7) Voted Classifier

Note that the Linear Support Vector Classifier is similar to SVC with linear kernel but it is implemented in terms of liblinear rather than libsvm. The implementation of a support vector machine using libsvm does not allow to scale to large number of samples.

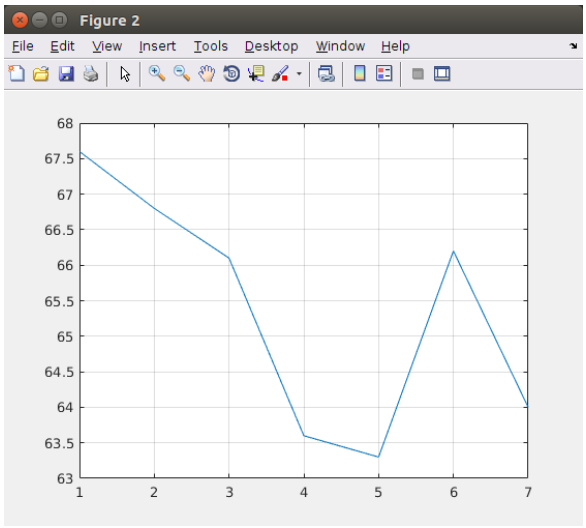


Figure 1: Scenario A threshold alpha

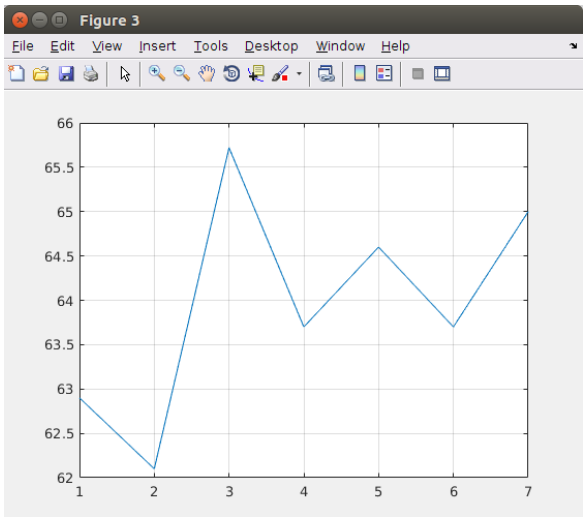


Figure 2: Scenario A threshold beta

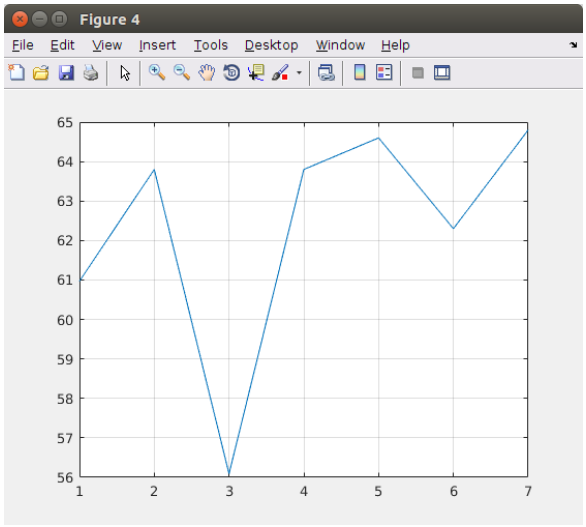


Figure 3: Scenario A threshold gamma

Voting system A voting system has been implemented in order to take into account all the predictions at the same time. All votes are counted and aggregated to yield a final result and a confidence value.

```
1  class VoteClassifier(ClassifierI):
2          def __init__(self,*classifiers):
3                  self._classifiers = classifiers
4
5          def classify(self,features):
6                  votes=[]
7                  for c in self._classifiers:
```

```
8                       v = c.classify(features)
9                       votes.append(v)
10
11              modeVotes = max(set(votes), key=votes.count)
12              return modeVotes
13
14      def classifyString(self,string):
15              features = find_feature(string)
16              votes=[]
17              for c in self._classifiers:
18                      v = c.classify(features)
19                      votes.append(v)
20
21              modeVotes = max(set(votes), key=votes.count)
22              return modeVotes
23
24      def confidenceFromString(self,string):
25              features = find_feature(string)
26              votes = []
27              for c in self._classifiers:
28                      v = c.classify(features)
29                      #print "\n",v,"\nnext:"
30                      votes.append(v)
31
32              modeVotes = max(set(votes), key=votes.count)
33
34              choice_votes = (float) (votes.count(modeVotes))
35              tot = len(votes)
36              conf = (float) (choice_votes/tot)
37              #print(votes)
38              #print(len(votes))
39              #print("Confidence: ", conf)
40              return conf
41
42
43      def confidence(self,features):
44              votes = []
45              for c in self._classifiers:
46                      v = c.classify(features)
47                      votes.append(v)
48
49              modeVotes = max(set(votes), key=votes.count)
50
51              choice_votes = (float) (votes.count(modeVotes))
52              tot = len(votes)
53              conf = (float) (choice_votes/tot)
54              #print(votes)
55              #print(len(votes))
56              #print("Confidence: ", conf)
57              return conf
```

The implemented class can be used to predict the label of a string.

Algorithm Comparison Below is reported a comparison of the implemented classifiers as a function of the size of the feature vector.

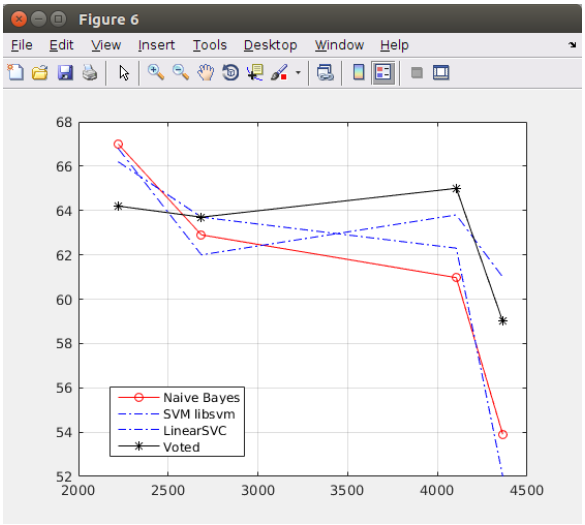


Figure 4: Naive Bayes, SVM with linear kernel, LinSVC, Voted classifier

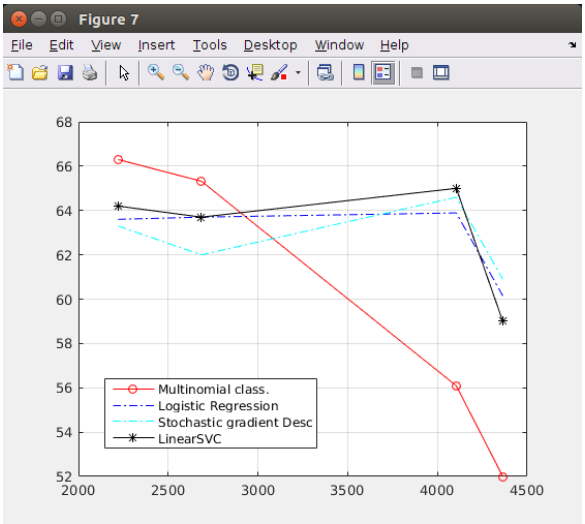


Figure 5: Multinomial Classifier, Logistic Regression, Stochastic Gradient Descent and voted classifier

References

[1] G. Balestrieri, *"Project Overview"*, Whazzo s.r.l., 2014.