Matching the Users of a Website Using SVM Technique

Dr. C. Muthu
Associate Professor, Department of Statistics
St. Joseph's College (Autonomous), Tiruchirappalli
&
M. C. Prakash
PG Student, Bharathidasan University Institute of Management
Bharathidasan University, Tiruchirappalli

Abstract

The more information the business organizations have on their customers, the better they are able to predict their preferences and customize their offerings, fostering customer trust and leading to a significant improvement in the business. In this paper, the SVM technique is applied to the dataset pertaining to matching people on a website. Given the information about any two users of this website, it is possible to predict whether they will be a good match.

Key Terms

Support-Vector Machines, Machine Learning Algorithms

Introduction

The emerging field of data science is dominated by the application of advanced statistical algorithms.[1] The Hadoop ecosystem is now extensively used for successfully implementing the advanced statistical algorithms on the big data.[2] The dataset we will use in this study is obtained from the portsmouthmatch maker.com site, which is currently enhanced by Shalom InfoTech. This website collects a lot of interesting information about their users, including demographic information, interests, and behaviour. To be more specific, this site collects the following data about its users: age, smoker/non-smoker, want children/don't want children, list of interests, location. This website also collects information about whether two people have made a good match, whether they initially made contact, and if they decided to meet in person. These data are used to create the matchmaker dataset, which contains the two data files namely matchdetails.csv and agedetails.csv. A sample matchdetails.csv file is shown below:

29, yes, no, skiing : knitting : dancing, 220 w 42nd St New York NY
23, no, yes, soccer : reading : scrabble, 824 3rd Ave, New York NY, 0
25, no, no, football : fashion, 102 1st Ave New York NY
30, no, no, snowboarding : knitting : computers : shopping : tv : travel, 151 W 34th St New York NY, 1
32, no, no, fashion : opera : tv : travel, 686 Avenue of the Americas New York NY
29, yes, yes, soccer : fashion : photography : computers : camping : movies : tv, 824 3rd Ave New York NY, 0

Each set of the two rows in the above data set contains information about a man and woman and, in the final column, a 1 or a 0 to indicate whether or not they are considered a good match. Once this website collects a large number of profiles, this information may be used to build a predictive algorithm that assists users in finding other people who are likely to be good matches. It may also indicate particular types of people that the site is lacking, which will be useful in devising strategies for promoting the site to new members. The agedetails.csv file has match information based only on age.

We now create a new python file called classifydata.py and add the **matchrows** class and the loadmatches( ) function to this file.

```
class matchrows:
    def - init -- (self, row, allnum = False):
        if allnum:
            self.data = [float(row[i]) for i in range (len(row) – 1)]
        else:
            self.data = row [0.len(row) – 1]
        self.matches= int (row[len(row) – 1])
    def loadmatches (f, allnum = False):
        rows = [ ]
        for line in file (f):
            rows.append (matchrows (line.split(‘,’), allnum))
        return rows
```

The above loadmatches( ) function creates a list of matchrows classes, each containing the raw data and whether or not there was a match. We shall use this function to load both the agedetails dataset and the matchdetails data set:

```
>>> import classifydata
>>> agedetails = classifydata.loadmatches (‘agedetails.csv’, allnum = True)
>>> matchdetails = classifydata.loadmatches (‘matchdetails.csv’)
```

Two interesting aspects of this dataset are the non-linearity and the interplay of the variables. The matchdetails dataset contains numerical data and categorical data. Some classifiers, like the decision tree, can handle both data types without any preprocessing. But the SVM classifier works only with numerical data. In order to handle this problem, we have to turn data into numbers so that it will be useful to the classifier. We now convert the “yes” into 1 and a “no” into -1. The missing or ambiguous data will be converted into 0. We now add the following yesno( ) function to classifydata.py.

```
def yesno(v) :
    if v == ‘yes’ : return 1
    elseif v == ‘no’ : return –1
    else : return 0
```

In order to treat every possible interest (knitting, dancing, etc.) as a separate numeric variable, we shall assign a 1 if the person has that interest and a 0 if he does not. If we are dealing with individual people, this is the best approach. But, in our study, we have pairs of people. So, a more intuitive approach is to use the number of common interests as a variable. We will now add the following countmatches( ) function to classifydata.py:

```
def countmatches (interest1, interest2) :
    l1 = interest1.split(‘:’)
    l2 = interest2.split(‘:’)
    x = 0
    for v in l1:
        if v in l2 : x += 1
    return x
```

In order to use the python open-source library LIBSVM on the matchdetails dataset, we convert it to the tuple of lists. We do it as follows:

```
>>> answers, inputs = [r.match for r in scaledset], [r.data for v in scaledset]
```

We are using the scaled data here to prevent overweighting of variables, as this improves the performance of the SVM algorithm. We use this new function to generate the new dataset and build a SVM model by using a radial-basis function as the kernel:

```
>>> paramet = svm-parameter (kernel-type = RBF)
>>> proba = svm-problem (answers, inputs)
>>> m = svm-model (proba, paramet)
```

Now, we can make predictions about whether people with a given set of attributes will be a match or not. We have to use the scalef( ) function to scale the data we want predictions for, so that the variables are on the same scale as the ones with which we built the model:

```
>>> newrow = [28.0, –1, –1, 26.0, –1, 1, 2, 0.8]   # Man doesn’t want
                                                   # children, Woman does
>>> m. predict (scalef(newrow))
0.0
>>> newrow = [28.0, –1, 1, 26.0, –1, 1, 2, 0.8] # Both want children
>>> m.predict (scalef(newrow))
1.0
```

Although this appears to give good predictions, it is preferable to test how good they really are so that we can choose the best parameters for our basis function. The cross-validation( ) function of LIBSVM automatically divides the dataset into training sets and test sets. The training sets are used to build the model, and the test sets are used to test the model to see how well it makes predictions.

We can now test the quality of our model by using the cross-validation ( ) function provided by LIBSVM. This function takes a parameter, n, and divides the dataset into n subsets. It then uses each subset as a test set and trains the model with all the other subsets. It returns a list of answers that we can compare to the original list.

```
>>> guesses = cross-validation (proba, paramet, 4)
... ...
... ...
>>> guesses
     [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, ...
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
      1.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
      ... ...
     ]
>>> sum ([abs(answers[i] - guesses[i] for i in range(len(guesses))])
29.0
```

The number of differences between the answers and the guesses is 29. Since there were 500 rows in the original dataset, this means that the algorithm got 471 matches correct.

References

1. Jacques Bughin (2016). Big Data, Big Bang?, *Journal of Big Data*, 3(2): 1-14.
2. Muthu, C. and Prakash, M. C. (2015). Impact of Hadoop Ecosystem on Big Data Analytics, *International Journal of Exclusive Management Research - Special Issue*, pp. 88-90.
3. Wes Mckinney (2012). *Python for Data Analysis*. O'Reilly Press, USA.