Using Bayesian Classifier for E-mail Sorting

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

Machine learning methods help us to draw valid conclusions from big data about user experience, marketing, personal tastes and human behaviour. In this paper, the Naïve Bayesian Classifier technique is used for sorting a variety of emails received by SafeTrust, a NBFC client of Shalom InfoTech, which has many divisions, such as Loan division, Deposits division, Insurance division and Chit Fund division.

Key Terms

Big Data Analytics, Machine Learning, Bayesian Classifier, Clustering

Introduction

Big Organizations are immensely benefitted by the advanced statistical techniques that are related to Big Data Analytics[1]. The successful implementation of the advanced statistical algorithms on the big data is now greatly facilitated by the Hadoop Ecosystem[2] . In this paper, the Naïve Bayesian Classifier technique is used for automatically sorting the variety of e-mails received regarding loans, deposits, insurance policies and chit funds by SafeTrust, a NBFC client of Shalom InfoTech.

E-mail sorting by Bayesian Classifier

The Naïve Bayesian Classifier shall be used to solve the problem of recognizing whether a document belongs to one category or another. For example, one's inbox shall be divided into social e-mails and work-related e-mails, based on the contents of the messages. In this study, we wish to use this technique for automatically sorting the variety of e-mails received by SafeTrust, a NBFC client of Shalom InfoTech. It is a well-known fact that the words EMI, loan amount and repayment period will occur often in the loan-related e-mails. Similarly, the words premium, assured amount and policy period will occur often in the Insurance-related e-mails.

In order to use the naive Bayessian Classifier, we will first determine the probability of an entire document being given a particular classification. In the python program documentclass.py, we create a subclass of classifier called naivebayes and create the documentprob( ) method that extracts the features and multiplies their probabilities together to get an overall probability Pr (Document/ category).

```
import re
import math
def sampletrainer (cla):
    cla.train (‘Please extend the repayment period of my loan’, ‘loan’)
    cla.train (‘I will pay this month's EMI on next monday’, ‘loan’)
    cla.train (‘Inform me the policy period of my Insurance policy’, ‘insurance’)
    cla.train (‘How can I enhance the assured amount of my Insurance policy?’,
            ‘insurance’)
    def getwords (doc):
        splitter = re.compile (‘\\w*’)
        words = [s.lower ( ) for s in splitter.split (doc)
            if len(s) > 2 and len(s) < 20]
        return dict ([w, 1) for w in words])
    class classifier:
    def – init – (self, getfeatures, filename = None):
        self.fc = { }
        self.cc = { }
        self.getfeatures = getfeatures
        def incf (self, f, cat):
            self.fc.setdefault (f, { })
            self.fc [f].setdefault (cat, 0)
            self.fc [f] [cat] += 1
            def incc (self, cat):
            self.cc.setdefault (cat, 0)
            self.cc [cat] += 1
            def fcount (self, f, cat):
                if f in self.fc and cat in self.fc [f]:
                    return float (self.fc [f] [cat])
                return 0.0
            def catecount (self, cat) :
            if cat in self.cc :
                return float (self.cc[cat])
            return 0
        def totalcount(self) :
            return sum(self.cc.values ( ))
        def categories (self) :
            return self.cc.keys( )
        def train (self, item, cat):
            features = self.getfeatures(item)
                for f in features:
                    self.incf (f, cat)
                    self.incc (cat)
          def fproba (self, f, cat):
               if self.catcount(cat) == 0 : return 0
                    return self.fcount (f, cat) / self.catcount (cat)
          class naivebayes (classifier) :
               def documentprob (self, item, cat):
                    features = self.getfeatures (item)
                    p = 1
                    for f in features : p * = self.weightedprob (f, cat, self.fprob)
          return p
```

We calculate now Pr(Category/Document) by using Baye's Theorem as follows:

Pr(Category/Document) =
Pr(Document/Category) × Pr(Category)/Pr(Document)

Here, Pr(Category) is the probability that a randomly selected document will be in this category. So, it is just the number of documents in the category divided by the total number of documents. Here, the calculation of Pr(Document) will be an unnecessary effort, as the results of this calculation will not be used as a real probability. Instead, the probability for each category will be calculated separately, and then all the results will be compared. Since Pr(Document) is the same no matter what category the calculation is being done for, it will scale the results by the exact same amount. So, we can safely ignore that term. We now add the **proba()** method to the **naivebayes** class for calculating Pr(Category/ Document).

```
def proba (self, item, cate):
     catproba = self.catecount (cate) / self.totalcount( )
     documentproba = self.documentprob (item, cate)
     return documentproba * catproba
```

The final step in building the Naive Bayes Classifier is actually deciding which category a new document belongs to. The simplest approach will be to calculate the probability of this item being in each of the different categories and to choose the category with the best probability. The following **classify()** method will calculate the probability for each category and will determine which one is the largest.

```
def classify (self, item, default = None):
     probabs = { }
     maxi = 0.0
     for cate in self.categories:
          probabs[cate] = self.proba(item, cate)
          if probabs[cate] > maxi:
               maxi = probabs[cate]
               best = cate
     return best
```

Now, the entire code that we have developed so far is capable of classifying a new document into the appropriate category. Here is a sample Python session:

```
>>> reload(documentclass)
>>> cla = documentclass.naivebayes (documentclass.getwords)
>>> documentclass.sampletrainer (cla)
>>> cla.classify ('Will you please inform me the policy period of my insurance
        policy? ', default = 'unknown') 'insurance'
>>> cla.classify ('How can I pay my loan's EMI through online?', default =
        'unknown') 'loan'
```

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.