# Web Science cs532-s16

## Assignment 9 Report

Dr. Michael L. Nelson

By: Huan Huang
04/22/2016

# Problem 1

```
Choose a blog or a newsfeed (or something similar with an Atom
or RSS feed).  Every student should do a unique feed, so please
"claim" the feed on the class email list (first come, first served).
It should be on a topic or topics of which you are qualified to
provide classification training data.  Find something with at least
100 entries (or items if RSS).

Create between four and eight different categories for the entries
in the feed:

examples:

work, class, family, news, deals

liberal, conservative, moderate, libertarian

sports, local, financial, national, international, entertainment

metal, electronic, ambient, folk, hip-hop, pop

Download and process the pages of the feed as per the week 12
class slides.
```

## Answer

I took me a long time to find a web page with over 100 RSS feeds. I eventually used PlayStation's RSS feeds for movies, which has over 10 thousand feeds. I downloaded and save the feeds with the curl -o command and saved it as psMovies.xml. Next, I picked out the categories for classifying the entries, which are:

action - a film genre in which the characters are thrust into a series of challenges that involve physical feats, extended fight scenes, violence, and frantic chases.

comedy - a genre of film in which the main emphasis is on humour. These films are designed to make the audience laugh through amusement and most often work by exaggerating characteristics for humorous effect. Films in this style traditionally have a happy ending (black comedy being an exception).

scifi - a film genre that uses science fiction: speculative, fictional science-based depictions of phenomena that are not fully accepted by mainstream science, such as extraterrestrial life forms, alien worlds, extrasensory perception and time travel, along with futuristic el-

ements such as spacecraft, robots, cyborgs, interstellar space travel or other technologies. Science fiction films have often been used to focus on political or social issues, and to explore philosophical issues like the human condition.

horror - unsettling films designed to frighten and panic, cause dread and alarm, and to invoke our hidden worst fears, often in a terrifying, shocking finale, while captivating and entertaining us at the same time in a cathartic experience.

romance - romantic love stories recorded in visual media for broadcast in theaters and on television that focus on passion, emotion, and the affectionate romantic involvement of the main characters and the journey that their genuinely strong, true and pure romantic love takes them through dating, courtship or marrige. Romance films make the romantic love story or the search for strong and pure love and romance the main plot focus.

other - other movie genres that are not included above.
(I copied the movie genre definitions from WIKIPEKIA)

# Problem 2

```
Manually classify the first 50 entries, and then classify (using
the fisher classifier) the remaining 50 entries.

Create a table with the title, predicted category, actual category,
and cprob() and fisherprob() for the actual category.
```

## Answer

I copy and modified the files docclass.py and feedfilter.py to solve this problem. The program feedfilter.py reads and parse the entries from psMovies.xml. And for each of the top 50 entries, I enter an actual classification which is use to call the classifier.train() function in docclass.py. This is used to train the classifier and enables it to make predictions of the category of the movies based on my input. I also get the fisher's conditional probability by calling the classifier.cprob() and classifier.fisherprob() functions, the actual category is to those functions to calculate the probabilities .

```
------------------------------ Manually classify first 50 entries ------------------------------
Title                                              Prediction  Actual   cprob   fisherprob
```

| Title | Prediction | Actual | cprob | fisherprob |
|---|---|---|---|---|
| Holy Ghost People | None | other | 1.0 | 0.75 |
| Bad Ass 2: Bad Asses | other | action | 0.0 | 0.8861 |
| Dom Hemingway - Theatrical Tralier | other | comedy | 0.0 | 0.75 |
| Nymphomaniac: Volume 2 | other | other | 1.0 | 0.75 |
| In The Blood | action | action | 1.0 | 0.75 |
| 10 Rules for Sleeping Around | action | romance | 1.0 | 0.75 |
| The Unknown Known | action | other | 1.0 | 0.75 |
| Afflicted | action | horror | 1.0 | 0.75 |
| Alien Abduction | horror | horror | 0.0 | 0.8861 |
| Alan Partridge | action | comedy | 1.0 | 0.75 |
| The Occupants | action | horror | 1.0 | 0.75 |
| The Little Rascals Save the Day | action | comedy | 1.0 | 0.75 |
| Seal Team Eight: Behind Enemy Lines | horror | action | 1.0 | 0.75 |
| The Pirate Fairy | horror | other | 1.0 | 0.75 |
| The Pirate Fairy (3D) | other | other | 1.0 | 0.8333 |
| Ages and Stages: The Story of the Meligrove Band | other | other | 1.0 | 0.875 |
| Blumenthal | action | comedy | 1.0 | 0.75 |
| Let Timmy Smoke | other | other | 0.0 | 0.8861 |
| I Am Divine | action | other | 1.0 | 0.75 |
| Four Seasons | horror | romance | 1.0 | 0.75 |
| About A Zombie | action | horror | 0.0 | 0.7428 |
| The Living Matrix | other | other | 1.0 | 0.8333 |
| Herbert Nitsch: Back from the Abyss | comedy | other | 1.0 | 0.75 |
| HazMat | horror | horror | 1.0 | 0.75 |
| Battle of the Empires | action | action | 1.0 | 0.75 |
| Volcom: True to This | horror | other | 0.0 | 0.8861 |
| E.T.X.R. | horror | scifi | 0.0 | 0.8738 |
| Hide Your Smiling Faces | horror | other | 1.0 | 0.75 |
| Locker 13 | other | other | 1.0 | 0.75 |
| Airplane vs Volcano | horror | scifi | 1.0 | 0.75 |
| Mistaken For Strangers | other | comedy | 1.0 | 0.75 |
| Red Bull X Fighters 2013 | horror | other | 0.2632 | 0.3421 |
| The Appearing | horror | horror | 0.0 | 0.8861 |
| SSI: Sex Squad Investigation | other | comedy | 1.0 | 0.75 |
| Black Roots | horror | other | 0.0 | 0.8861 |
| Happy Camp | horror | horror | 1.0 | 0.75 |
| Odd Thomas | horror | comedy | 0.0 | 0.8861 |
| Countdown | horror | horror | 1.0 | 0.75 |
| Awake Zion | horror | other | 1.0 | 0.75 |
| Beast of the Bering Sea | other | scifi | 1.0 | 0.75 |
| Wendy Liebman: Taller on TV | horror | comedy | 0.0 | 0.9188 |
| How to Follow Strangers | comedy | romance | 1.0 | 0.75 |
| In Heaven There is No Beer | horror | other | 0.4848 | 0.4886 |
| Come Back, Africa | horror | other | 0.3077 | 0.3718 |
| Home | horror | other | 1.0 | 0.75 |
| Scooby-Doo! Wrestlemania Mystery | comedy | comedy | 0.0 | 0.8861 |
| Avengers Confidential: Black Widow & Punisher | comedy | scifi | 1.0 | 0.75 |
| Bad Words - Theatrical Tralier | comedy | comedy | 0.0 | 0.6207 |
| Cheap Thrills | horror | horror | 1.0 | 0.75 |
| The French Minister | comedy | comedy | 0.0 | 0.8861 |

```
--------------------------- Other 50 automaticly classified entries ----------------------------
Title                                                  Prediction  Actual    cprob    fisherprob
```

| Title | Prediction | Actual | cprob | fisherprob |
|---|---|---|---|---|
| Mardock Scramble: Third Exhaust | other | action | 0.0 | 0.5 |
| Blood Ties (2014) | horror | other | 0.0 | 0.25 |
| Maladies | comedy | other | 0.0 | 0.5 |
| Divergent – Theatrical Trailer | comedy | scifi | 0.0 | 0.5 |
| McCanick | horror | action | 0.0 | 0.5 |
| Nymphomaniac: Volume 1 | other | other | 0.0 | 0.7428 |
| Battle of the Undead | horror | horror | 0.0 | 0.5 |
| American Virgins | horror | comedy | 0.0 | 0.0714 |
| Almost Sharkproof | horror | comedy | 0.0 | 0.5 |
| Sparks | horror | action | 0.0 | 0.5 |
| The Wrath of Vajra | action | action | 0.0 | 0.5399 |
| The Laughing Matter | horror | comedy | 1.0 | 0.8333 |
| Tom Hollands Twisted Tales | horror | horror | 0.0 | 0.5 |
| Jeff Dunhams Achmed Saves America: The Animated Movie | horror | comedy | 0.0 | 0.4752 |
| Waking | horror | romance | 0.0 | 0.5 |
| Romeo, Romeo | horror | romance | 0.0 | 0.5 |
| CyberGeddon | horror | other | 0.0 | 0.5 |
| Signature Sounds: Music of WWE | horror | other | 0.0 | 0.25 |
| Maidentrip | romance | other | 0.0 | 0.5 |
| Foxfire | romance | other | 0.0 | 0.1 |
| Danny MacAskills Imaginate Documentary | comedy | other | 0.0 | 0.5 |
| Buck Wild | comedy | comedy | 0.0 | 0.5 |
| 5th Street | horror | action | 0.0 | 0.5 |
| How to Be a Man | other | comedy | 0.0 | 0.3997 |
| UFC Event Replays \| UFC 171: Hendricks vs. Lawler | horror | other | 0.0 | 0.5 |
| Patrick: Evil Awakens | horror | horror | 0.0 | 0.5 |
| Better Living Through Chemistry | horror | comedy | 0.0 | 0.2358 |
| The Den | horror | horror | 0.0 | 0.5 |
| The Art of the Steal | horror | comedy | 0.0 | 0.5 |
| uwantme2killhim? | horror | horror | 0.0 | 0.5 |
| The Right Kind of Wrong | other | romance | 0.0 | 0.5 |
| Need for Speed – Theatrical Trailer | comedy | action | 0.0 | 0.5 |
| Veronica Mars | horror | comedy | 0.0 | 0.5 |
| The Last Days | scifi | scifi | 0.0 | 0.5 |
| WWE WCWs Greatest PPV Matches Vol. 1 | other | other | 0.0 | 0.5966 |
| Bubble Guppies: Animals Everywhere! | other | other | 0.0 | 0.3964 |
| End of the World | comedy | scifi | 0.0 | 0.5 |
| A Cross to Bear | horror | other | 0.0 | 0.5966 |
| Shadowboxing | horror | comedy | 0.0 | 0.5 |
| Dark House | horror | horror | 0.0 | 0.5 |
| The Hungover Games | other | comedy | 0.0 | 0.5 |
| The Hungover Games (Unrated) | other | comedy | 0.0 | 0.5 |
| A Dance for Bethany | horror | romance | 0.0 | 0.5 |
| WWE WCWs Greatest PPV Matches Vol. 3 | other | other | 0.0 | 0.5966 |
| WWE WCWs Greatest PPV Matches Vol. 2 | other | other | 0.0 | 0.5966 |
| Soap Life | other | other | 0.0 | 0.5966 |
| Hated | other | other | 0.6346 | 0.6077 |
| In the Name of the King 3: The Last Mission | other | action | 0.0 | 0.5 |
| Celebrating the Music of "Inside Llewyn Davis" | other | other | 0.3426 | 0.374 |
| All Things to All Men | horror | other | 0.0 | 0.1667 |

```python
import feedparser
import re
import math
import docclass

savefile = open("data.txt", "a")

# Takes a filename or URL of a blog feed and classifies the entries
def read(feed, classifier):
```

```
10
11   splitRegexp = re.compile( r"<[^>]+>" )
12
13   num=0
14   # Get feed entries and loop over them
15   f=feedparser.parse(feed)
16
17   string1 = '—————————————————————— Manually classify first 50
       entries ——————————————————————\n'
18   dividline = "_" * 100 + "\n"
19   print string1
20   savefile.write(string1)
21   string2 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format("Title", "Prediction", "
       Actual", 'cprob', 'fisherprob',"\n")
22   print string2
23   savefile.write(string2)
24   savefile.write(dividline)
25
26   for entry in f['entries'][0:50]:
27     num=num +1
28     # Print the contents of the entry
29     title=entry['title'].encode('utf-8').replace("'","")
30     print 'Title:      '+ title
31
32     summary = splitRegexp.sub( "", entry[ "summary" ] )
33
34     print summary #entry['summary'].encode('utf-8')
35
36     # Combine all the text to create one item for the classifier
37     #fulltext='%s\n%s\n%s' % (entry['title'],entry['publisher'],entry['summary
       '])
38     fulltext='%s\n%s' % (entry['title'],entry['summary'])
39     # Remove apostrophes
40     fulltext=fulltext.replace("'","")
41     # Print the best guess at the current category
42     predicted=str(classifier.classify(fulltext))
43     print 'Predicted category: ', predicted
44
45     # Ask the user to specify the correct category and train on that
46     actual=raw_input('Actual category: ')
47     classifier.train(fulltext, actual)
48
49     feature=raw_input('Enter string classifier: ')
50
51     #classifier.train(entry,cl)
52     # probability the item should be in this category
53     cp=round(classifier.cprob(feature,predicted),4)
54     fcp=round(classifier.fisherprob(feature,predicted),4)
55     print 'cprob: ', str(cp)
56     print 'fisherprob: ', str(fcp)
57     string3 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format(title, predicted,
       actual, str(cp), str(fcp), "\n")
58     savefile.write(string3)
59
```

```python
60     # Save the manual classifications
61     # num, entry, feature, predicted, actual, cprob=None
62      classifier.manualClassdb(num, title, feature, predicted, actual, cp, fcp)
63    savefile.write(dividline)
64 #def autoClassify(feed,classifier):
65    num=50
66     string4 = '\n——————————————————— Other 50 automaticly classified
       entries —————————————————————\n'
67    print string4
68    savefile.write(string4)
69    savefile.write(string2)
70    savefile.write(dividline)
71   # Get feed entries and loop over them
72   #f=feedparser.parse(feed)
73   for entry in f['entries'][50:100]:
74     num=num+1
75     # Print the contents of the entry
76      title=entry['title'].encode('utf-8').replace("'","")
77      print 'Title:       '+ title
78     summary = splitRegexp.sub( "", entry[ "summary" ] )
79
80      print summary #entry['summary'].encode('utf-8')
81
82     # Combine all the text to create one item for the classifier
83     #fulltext='%s\n%s\n%s' % (entry['title'],entry['publisher'],entry['summary
       '])
84      fulltext='%s\n%s' % (entry['title'],entry['summary'])
85      fulltext=fulltext.replace("'","")
86     # Print the best guess at the current category
87      predicted=str(classifier.classify(fulltext))
88      print 'Predicted: ', predicted
89
90     # Ask the user to specify the correct category
91      actual=raw_input('Actual category: ')
92      feature=raw_input('Enter string classifier: ')
93
94
95     # probability the item should be in this category
96      cp=round(classifier.cprob(feature,predicted),4)
97      fcp=round(classifier.fisherprob(feature,predicted),4)
98      print 'cprob: ', str(cp)
99      print 'fisherprob: ', str(fcp)
100
101      string6 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format(title, predicted,
       actual, str(cp), str(fcp), "\n")
102      savefile.write(string6)
103     # Save the trained classifications
104     # num, entry, feature, predicted, cprob(feature, predicted)
105      classifier.autoClassdb(num, title, feature, predicted, actual, cp, fcp)
106   #return classifier
107    savefile.write(dividline)
108
109 def entryfeatures(entry):
110    splitter=re.compile('\\W*')
```

```
111    f={}
112
113    # Extract the title words and annotate
114    titlewords=[s.lower() for s in splitter.split(entry['title'])
115            if len(s)>2 and len(s)<20]
116    for w in titlewords: f['Title:'+w]=1
117
118    # Extract the summary words
119    summarywords=[s.lower() for s in splitter.split(entry['summary'])
120            if len(s)>2 and len(s)<20]
121
122    # Count uppercase words
123    uc=0
124    for i in range(len(summarywords)):
125      w=summarywords[i]
126      f[w]=1
127      if w.isupper(): uc+=1
128
129      # Get word pairs in summary as features
130      if i<len(summarywords)-1:
131        twowords=' '.join(summarywords[i:i+1])
132        f[twowords]=1
133
134    # Removed: Keep creator and publisher whole
135    #f['Publisher:'+entry['publisher']]=1
136
137    # UPPERCASE is a virtual word flagging too much shouting
138    if float(uc)/len(summarywords)>0.3: f['UPPERCASE']=1
139
140    return f
141
142
143 cl=docclass.fisherclassifier(docclass.getwords)
144 cl.setdb('psMovies.db')
145 read('psMovies.xml',cl)
```

```
1  #from pysqlite2 import dbapi2 as sqlite
2  from sqlite3 import dbapi2 as sqlite
3  import re
4  import math
5
6  def getwords(doc):
7    splitter=re.compile('\\W*')
8    #print doc
9    ## Remove all the HTML tags
10   doc=re.compile(r'<[^>]+>').sub('',doc)
11   # Split the words by non-alpha characters
12   words=[s.lower() for s in splitter.split(doc)
13           if len(s)>2 and len(s)<20]
14
15   # Return the unique set of words only
16   return dict([(w,1) for w in words])
17
18 class classifier:
```

```
19    def __init__(self, getfeatures, filename=None):
20      # Counts of feature/category combinations
21      self.fc={}
22      # Counts of documents in each category
23      self.cc={}
24      ## extract features for classification
25      self.getfeatures=getfeatures
26
27    def setdb(self, dbfile):
28      self.con=sqlite.connect(dbfile)
29      self.con.execute('create table if not exists rss(num, entry, feature,
      predicted, actual, cprob)')
30      self.con.execute('create table if not exists fc(feature,category,count)')
31      self.con.execute('create table if not exists cc(category,count)')
32      # remove old data from previous sessions
33      self.con.execute('delete from rss')
34      self.con.execute('delete from fc')
35      self.con.execute('delete from cc')
36
37    def manualClassdb(self, num, entry, feature, predicted, actual, cp, fcp):
38      self.con.execute("insert into rss values ('%s', '%s', '%s', '%s', '%s', '%
      s')"
39                          % (num, entry, feature, predicted, actual, fcp))
40      self.con.commit()
41
42    def autoClassdb(self, num, entry, feature, predicted, actual, cp, fcp):
43      self.con.execute("insert into rss values ('%s', '%s', '%s', '%s', '%s', '%
      s')"
44                          % (num, entry, feature, predicted, actual, fcp))
45      self.con.commit()
46    ## Increase the count of a feature/category pair
47    def incf(self,f,cat):
48      count=self.fcount(f,cat)
49      if count==0:
50        self.con.execute("insert into fc values ('%s','%s',1)"
51                            % (f,cat.lower()))
52      else:
53        self.con.execute(
54          "update fc set count=%d where feature='%s' and category='%s'"
55          % (count+1,f,cat.lower()))
56
57    ## The number of times a feature has appeared in a category
58    def fcount(self,f,cat):
59      res=self.con.execute(
60        'select count from fc where feature="%s" and category="%s"'
61        %(f,cat)).fetchone()
62      if res==None: return 0
63      else: return float(res[0])
64
65    ## Increase the count of a category
66    def incc(self,cat):
67      count=self.catcount(cat)
68      if count==0:
69        self.con.execute("insert into cc values ('%s',1)" % (cat.lower()))
```

```
70        else :
71          self.con.execute("update cc set count=%d where category='%s'"
72                           % (count+1,cat))
73
74    ## The number of items in a category
75     def catcount(self,cat):
76       res=self.con.execute('select count from cc where category="%s"'
77                             %(cat)).fetchone()
78       if res==None: return 0
79       else: return float(res[0])
80
81    ## The list of all categories
82     def categories(self):
83       cur=self.con.execute('select category from cc');
84       return [d[0] for d in cur]
85
86    ## The total number of items
87     def totalcount(self):
88       res=self.con.execute('select sum(count) from cc').fetchone();
89       if res==None: return 0
90       return res[0]
91
92
93    ## The train method takes an item(document) and a classification.
94    ## It uses the getfeatures function to the break the item into its
95    ## separate features. It then calls incf to increase the counts for
96    ## this classification for every feature. Finally, it increases
97    ## the total count for this classification.
98     def train(self,item,cat):
99        features=self.getfeatures(item)
100      # Increment the count for every feature with this category
101       for f in features:
102         self.incf(f,cat)
103
104      # Increment the count for this category
105       self.incc(cat)
106       self.con.commit()
107
108    ## Probability is a number between 0 and 1, indicating
109    ## the likelihood of an event. You calculate the probability of
110    ## a word in a particular category by dividing the number of
111    ## times the word appears in a document in that category
112    ## by the total number of documents in the category.
113     def fprob(self,f,cat):
114       if self.catcount(cat)==0: return 0
115
116      # The total number of times this feature appeared in this
117      # category divided by the total number of items in this category
118       return self.fcount(f,cat)/self.catcount(cat)
119
120     def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
121       # Calculate current probability
122       basicprob=prf(f,cat)
123
```

```
124      # Count the number of times this feature has appeared in
125      # all categories
126      totals=sum([self.fcount(f,c) for c in self.categories()])
127
128      # Calculate the weighted average
129      bp=((weight*ap)+(totals*basicprob))/(weight+totals)
130      return bp
131
132
133
134
135 class naivebayes(classifier):
136
137    def __init__(self,getfeatures):
138       classifier.__init__(self,getfeatures)
139       self.thresholds={}
140
141    def docprob(self,item,cat):
142       features=self.getfeatures(item)
143
144       # Multiply the probabilities of all the features together
145       p=1
146       for f in features: p*=self.weightedprob(f,cat,self.fprob)
147       return p
148
149    def prob(self,item,cat):
150       catprob=self.catcount(cat)/self.totalcount()
151       docprob=self.docprob(item,cat)
152       return docprob*catprob
153
154    def setthreshold(self,cat,t):
155       self.thresholds[cat]=t
156
157    def getthreshold(self,cat):
158       if cat not in self.thresholds: return 1.0
159       return self.thresholds[cat]
160
161    def classify(self,item,default=None):
162       probs={}
163       # Find the category with the highest probability
164       max=0.0
165       for cat in self.categories():
166          probs[cat]=self.prob(item,cat)
167          if probs[cat]>max:
168             max=probs[cat]
169             best=cat
170
171       # Make sure the probability exceeds threshold*next best
172       for cat in probs:
173          if cat==best: continue
174          if probs[cat]*self.getthreshold(best)>probs[best]: return default
175       return best
176
177 ## This function will return the probability that an item with the
```

```python
178 ## specified feature belongs in the specified category, assuming there
179 ## will be an equal number of items in each category.
180 class fisherclassifier(classifier):
181   def cprob(self,f,cat):
182     # The frequency of this feature in this category
183     clf=self.fprob(f,cat)
184     if clf==0: return 0
185
186     # The frequency of this feature in all the categories
187     freqsum=sum([self.fprob(f,c) for c in self.categories()])
188
189     # The probability is the frequency in this category divided by
190     # the overall frequency
191     p=clf/(freqsum)
192
193     return p
194
195
196   def fisherprob(self,item,cat):
197     # Multiply all the probabilities together
198     p=1
199     features=self.getfeatures(item)
200     for f in features:
201       p*=(self.weightedprob(f,cat,self.cprob))
202
203     # Take the natural log and multiply by −2
204     fscore=−2*math.log(p)
205
206     # Use the inverse chi2 function to get a probability
207     return self.invchi2(fscore,len(features)*2)
208
209   ## Inverse chi−squared function
210   def invchi2(self,chi, df):
211     m = chi / 2.0
212     sum = term = math.exp(−m)
213     for i in range(1, df//2):
214         term *= m / i
215         sum += term
216     return min(sum, 1.0)
217
218   def __init__(self,getfeatures):
219     classifier.__init__(self,getfeatures)
220     self.minimums={}
221
222   def setminimum(self,cat,min):
223     self.minimums[cat]=min
224
225   def getminimum(self,cat):
226     if cat not in self.minimums: return 0
227     return self.minimums[cat]
228
229   def classify(self,item,default=None):
230     # Loop through looking for the best result
231     best=default
```

```
232      max=0.0
233      for c in self.categories():
234        p=self.fisherprob(item,c)
235        # Make sure it exceeds its minimum
236        if p>self.getminimum(c) and p>max:
237          best=c
238          max=p
239      return best
```

# Problem 3

Assess the performance of your classifier in each of your categories by computing precision, recall, and F-measure.

## Answer

In order to calculate the precision, recall, and F-measures. I first have to get the false positive, false negative, and true positive of the predictions. Where the prediction matched my actual input for each category is the true positive for each category. The false positive and false negative is obtained by subtracting the TP from predicted and actual counts of each category. Then use the formulas from lecture slides to calculated the precision, recall, and F-measures.

Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F-Measure = 2 * P*R / (P+R)

| Categories | FP | FN | TP | Precision | Recall | F-measure |
|------------|----|----|----|-----------|--------|-----------|
| Action | 10 | 7 | 3 | 0.23 | 0.3 | 0.26 |
| Comedy | 8 | 19 | 4 | 0.33 | 0.17 | 0.22 |
| Horror | 35 | 3 | 12 | 0.26 | 0.8 | 0.39 |
| Other | 10 | 24 | 14 | 0.58 | 0.37 | 0.45 |
| Romance | 2 | 7 | 0 | 0 | 0 | 0 |
| Scifi | 0 | 6 | 1 | 1 | 0.14 | 0.25 |

Based on the result, the classifier did not perform well at all, but it was expected, since I only trained it with 50 entries.