Assignment 4

CS834 Introduction to Information Retrieval

Fall 2017

Polykarpos Thomadakis

December 11, 2017

Question 10.3

Compute five iterations of HITS (see Algorithm 3) and PageRank (see Figure 4.11) on the graph in Figure 10.3. Discuss how the PageRank scores compare to the hub and authority scores produced by HITS.

Answer

For the purposes of this assignment I wrote a script that computes the values desired based on the algorithms provided in the book. First I create 2 classes that represent a node in the graph and the graph itself. Then I use them to run the HITS and PageRank algorithms on the graph. The output is a set of files that are then used to create png pictures for the graphs, containing the appropriate information using the graphviz tool. The values of PageRank, hubs and authorities per iteration are presented in figures 2 to 6. The initial graph is shown in 1.

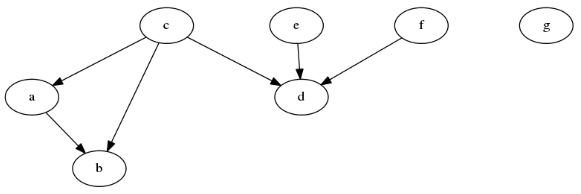
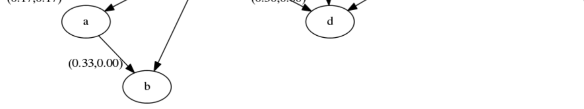
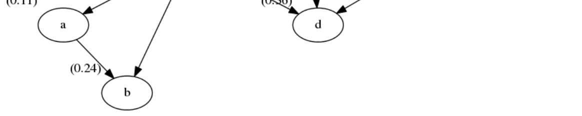


Figure 1: The initial graph from the book

We can see that the final ranking for both algorithms is the same as shown in table 1. The scores in HITS are higher for the nodes that have at least one incoming link, however, the nodes that do not have any, are set to 0. And since PageRank sets a non-zero value to all nodes, including the ones without inlinks and the fact that the total score in both algorithms is 1, it makes sense that

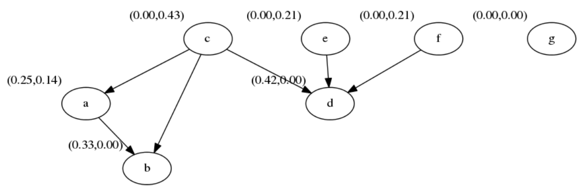


(a) HITS iteration 1

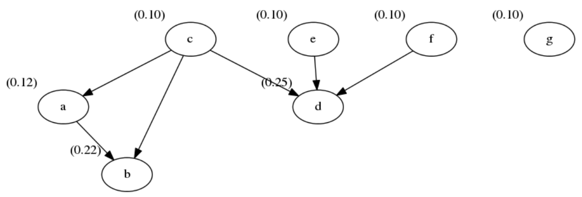


(b) PageRank iteration 1

Figure 2: Iteration 1 of the two algorithms

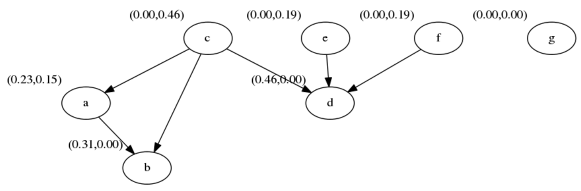


(a) HITS iteration 2

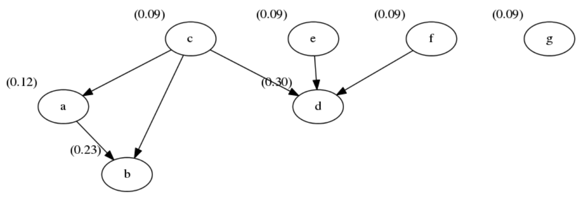


(b) PageRank iteration 2

Figure 3: Iteration 2 of the two algorithms

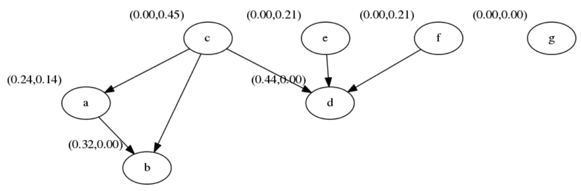


(a) HITS iteration 3

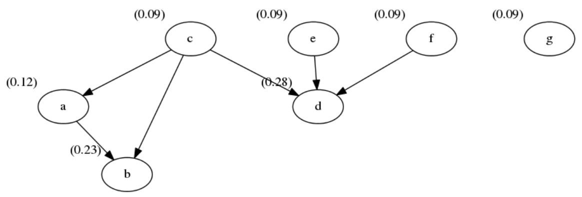


(b) PageRank iteration 3

Figure 4: Iteration 3 of the two algorithms

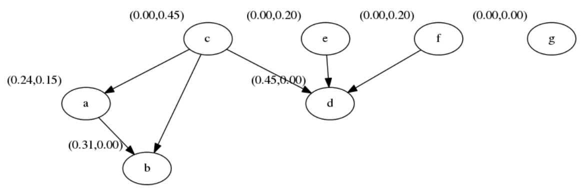


(a) HITS iteration 4

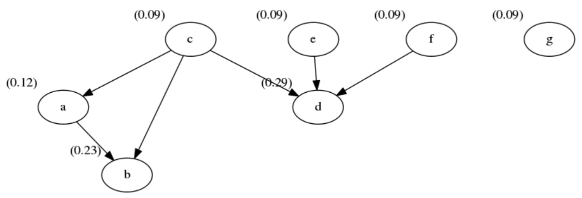


(b) PageRank iteration 4

Figure 5: Iteration 4 of the two algorithms



(a) HITS iteration 5



(b) PageRank iteration 5

Figure 6: Iteration 5 of the two algorithms

Table 1: Node ranking based on PageRank and HITS algorithms

Node	PageRank	HITS
a	3	3
b	2	2
c	4	4
d	1	1
e	4	4
f	4	4
g	4	4

HITS scores are higher for the nodes with inlinks. The code for this question can be seen in listing 1 and the script to produce the pictures in listing 2.

```
def converged(I,R):
  threshold = 0.01
  for p in I:
    if I[p]-R[p]>threshold or I[p]-R[p]<-threshold :
      return False
  return True


def PageRank(G):
  l = 0.12
  nodes_size = len(G.get_nodes())
  I = {}
  R = {}
  for p in G.get_nodes():
    I[p] = float(1)/nodes_size
    R[p] = 0
  iters = 0
  while(1):
    iters +=1
    for r in R:
      R[r] = l/nodes_size
    for p in sorted(G.get_nodes()):
      Q = G.get_adj_nodes(p)
      if len(Q)>0:
        for q in Q:
          R[q] = R[q] + (1-l)*float(I[p])/len(Q)
      else :
        for q in G.get_nodes():
          R[q] = R[q] + (1-l)*float(I[p])/nodes_size

    if(converged(I,R)):
      break
    for p in R:
      I[p]=R[p]
    with open('PR_graph_iter_'+str(iters)+'.dot','w') as f:
      f.write("digraph g {\n")
      f.write("graph [nodesep=1];\n")
      for v in R:
        f.write(v+" [xlabel=\"("+str.format('{:.2f}',R[v])+")\"]\n")
      f.write(str(g))
      f.write("}\n")

  with open('PR_graph_iter_'+str(iters)+'.dot','w') as f:
      f.write("digraph g {\n")
      f.write("graph [nodesep=1];\n")
      for v in R:
        f.write(v+" [xlabel=\"("+str.format('{:.2f}',R[v])+")\"]\n")
      f.write(str(g))
      f.write("}\n")
  print str(iters)
  return R

def HITS(G,K):
  A = []
  H = []
  A.append(dict())
  H.append(dict())
  for p in G.get_nodes():
    A[0][p] = 1
    H[0][p] = 1
  for i in range(1,K+1):
    A.append(dict())
    H.append(dict())
    for p in G.get_nodes():
      A[i][p] = 0
      H[i][p] = 0
    Z_A = 0
    Z_H = 0
    for p in G.get_nodes():
      for q in G.get_nodes():
        if G.edge_exists(p,q):
          H[i][p]+= A[i-1][q]
          Z_H = Z_H + A[i-1][q]
        if G.edge_exists(q,p):
          A[i][p] += H[i-1][q]
          Z_A = Z_A + H[i-1][q]
    for p in G.get_nodes():
      A[i][p] = float(A[i][p])/Z_A
      H[i][p] = float(H[i][p])/Z_H
  return A[K],H[K]

class node:

  def __init__(self,name):
    self.adjacent_nodes = []
    self.name = name

  def add_adj_node(self,n):
    if n not in self.adjacent_nodes and n != self:
      self.adjacent_nodes.append(n)

  def __str__(self):
    return self.name

  def get_adj_nodes(self):
    return self.adjacent_nodes

  def is_connected(self,n):
    if n in self.adjacent_nodes:
      return True
    else:
      return False

  def __eq__(self,other):
    if isinstance(other,node):
      return self.name == other.name
    return False

  def __ne__(self,other):
    if isinstance(other,node):
      return self.name != other.name
    return True

class Graph:
  def __init__(self):
    self.g_nodes = {}

  def add_node(self,n):
    if str(n) not in self.g_nodes:
      self.g_nodes[str(n)] = n

  def get_nodes(self):
    return self.g_nodes

  def get_adj_nodes(self,n):
    adj = self.g_nodes[n].get_adj_nodes()
    names = []
    for a in adj:
      names.append(str(a))
    return names

  def add_edge(self,v1,v2):
    if v1 not in self.g_nodes or v2 not in self.g_nodes:
      return
    self.g_nodes[v1].add_adj_node(self.g_nodes[v2])

  def edge_exists(self,v1,v2):
    if self.g_nodes[v1].is_connected(self.g_nodes[v2]):
      return True
    else:
      return False

  def __str__(self):
    graph_str =""
    for n in sorted(self.g_nodes):
      for adj in self.g_nodes[n].get_adj_nodes():
        graph_str += n +"-> "+str(adj)+";\n"
      if len(self.g_nodes[n].get_adj_nodes()) ==0:
        graph_str += n +";\n"

    return graph_str

g = Graph()
g.add_node(node('a'))
g.add_node(node('b'))
g.add_node(node('c'))
g.add_node(node('d'))
g.add_node(node('e'))
g.add_node(node('f'))
g.add_node(node('g'))
g.add_edge('a','b')
g.add_edge('c','a')
g.add_edge('c','b')
g.add_edge('c','d')
g.add_edge('e','d')
g.add_edge('f','d')

with open('initial_graph.dot','w') as f:
  f.write("digraph g {\n")
  f.write("graph [nodesep=1];\n")
  f.write(str(g))
  f.write("}\n")


for k in range(1,6):
  A,H = HITS(g,k)
  with open('HITS_graph_iter_'+str(k)+'.dot','w') as f:
    f.write("digraph g {\n")
    f.write("graph [nodesep=1];\n")
```

```python

    for v in A:
      f.write(v+" [xlabel=\"("+str.format('{:.2f}',A[v])+","+str.format('{:.2f}',H[v
    ])+")\"]\n")
    f.write(str(g))
    f.write("}\n")

PageRank(g)
```

Listing 1: Script to generate the PageRank and HITS values for the given graph

```bash
#!/bin/bash

dot initial_graph.dot -Tpng > init.png
for i in {1..5}
do
dot HITS_graph_iter_$i.dot -Tpng > HITS_it$i.png

done

PR=$(ls PR_graph_iter*)
for g in $PR
do
dot $g -Tpng >${g//dot/png}
done
```

Listing 2: Bash script to generate the graphs with PageRank and HITS annotations

Question 10.5

Find a community-based question answering site on the Web and ask two questions, one that is low-quality and one that is high-quality. Describe the answer quality of each question.

Answer

Since there are already so many bad and good questions on the internet, I decided not to post some new ones but instead just use some of the existing ones. Those had more time to collect answers and thus, I believe I will have a more reliable conclusion based on them. The low-quality question that I use for this assignment is "Y r u living?" from Yahoo! answers. This looks quite bad to me. First of all, it doesn't even use full words to form the sentence, just some letters that sound the same as the words that should be there. Second, it's so general and does not really make any sense to ask this question. A sample of the answers to this question are:

"You must be a thicko to type on here in text language.
You must work in WalMart where you do not need an education, you sounds ridiculous."
"Bcuz Itz betr thn the alternative"
"for some reason i can't explain"

We see that the quality of the answers is low as well. I don't think that someone asking like that could expect a better answer than those. The answers mainly make fun of the question and one also uses a same way of excluding letters from the words. For the high-quality question I chose the question:
"Do lambda expressions have any use other than saving lines of code? Are there any special features provided by lambdas which solved problems which weren't easy to solve? " from stackoverflow. I think that this question is understandable and to the point with correct spelling and grammar. That makes it a good question in my opinion. There were several long and good answers but I chose a

smaller one that resembles them well.

"Programming languages are not for machines to execute. They are for programmers to think in.

Languages are a conversation with a compiler to turn our thoughts into something a machine can execute. One of the chief complaints about Java from people who come to it from other languages (or leave it for other languages) used to be that it forces a certain mental model on the programmer (i.e. everything is a class).

I'm not going to weigh in on whether that's good or bad: everything is trade-offs. But Java 8 lambdas allow programmers to think in terms of functions, which is something you previously could not do in Java.

It's the same thing as a procedural programmer learning to think in terms of classes when they come to Java: you see them gradually move from classes that are glorified structs and have 'helper' classes with a bunch of static methods and move on to something that more closely resembles a rational OO design (mea culpa).

If you just think of them as a shorter way to express anonymous inner classes then you are probably not going to find them very impressive in the same way that the procedural programmer above probably didn't think classes were any great improvement." The answer is precise, answers well the question asked and is very nicely constructed. We can see that the person that wrote this answer spend some time to think and form this answer in order to best convey to the other user his/her knowledge.

Question 10.6

Find two examples of document filtering systems on the Web. How do they build a profile for your information need? Is the system static or adaptive?

Answer

Nowadays, most of the leading websites use some kind of profiling system that is able to generate user profiles that are used to provide user-relevant content. For my two examples I used YouTube and Amazon. YouTube is the most popular video hosting website featuring millions of user uploaded videos.
YouTube tracks the videos watched by each user and is then able to make recommendations on the main page, set the next to play video or recommend channels from creators that are supposed to create videos relevant to the user. One can clearly see the difference of the next video to play by accessing it through a freshly installed browser (or by deleting cookies) and not signing in. In this case, the next video is found looking at videos relevant to the previous one only.
Amazon is another top-traffic website. It consists an Internet marketplace where almost every product can be purchased. The filtering works by tracking the products a user visits and buys. It realizes correlations by the purchases that come together by users. When a user buys a set of products at the same transaction, it is quite propable that another user will also be interested in the other products of the set when checking one of them.
Both websites use adaptive filtering, since the preferences of their users may change from time to time, or expand to different areas.

Question SVMLight1

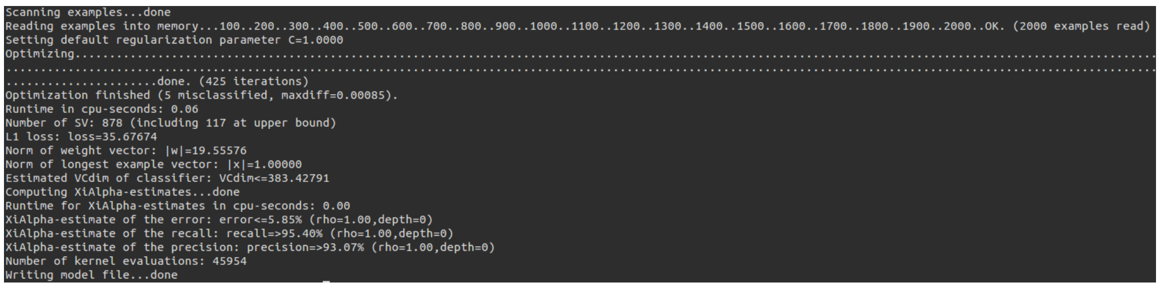
Work through the "Inductive SVM"example, discuss in detail the steps and resulting output.

Answer

First I installed SVMLight from `http://www.cs.cornell.edu/people/tj/svm_light/`. Then I downloaded the example folder located at `http://download.joachims.org/svm_light/examples/example1.tar.gz`. The content of the examples is the training set examples. The first lines may contain comments and are ignored if they start with #. Each of the following lin es represents one training example and is of the following format:

```
<line> .=. <target> <feature>:<value> <feature>:<value> ... <feature>:<value> # <info>
<target> .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value> .=. <float>
<info> .=. <string>
```

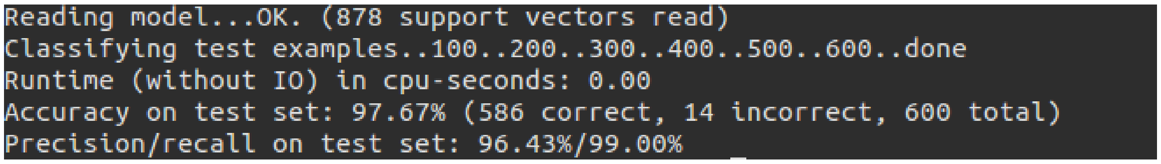
The target value and each of the feature/value pairs are separated by a space character. Features with value zero can be skipped. The string info can be used to pass additional information to the kernel (e.g. non feature vector data). In classification mode, the target value denotes the class of the example. +1 as the target value marks a positive example, -1 a negative example respectively. Running the svm_learn program as denoted on the Inductive example produced the model file used for the classification. The output of the svm_learn is shown in figure 7. Once the model is generated,



```
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..900..1000..1100..1200..1300..1400..1500..1600..1700..1800..1900..2000..OK. (2000 examples read)
Setting default regularization parameter C=1.0000
Optimizing.........................................................................................................................................
...............................................................................................................................................
.....................done. (425 iterations)
Optimization finished (5 misclassified, maxdiff=0.00085).
Runtime in cpu-seconds: 0.06
Number of SV: 878 (including 117 at upper bound)
L1 loss: loss=35.67674
Norm of weight vector: |w|=19.55576
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=383.42791
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=5.85% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>95.40% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>93.07% (rho=1.00,depth=0)
Number of kernel evaluations: 45954
Writing model file...done
```

Figure 7: Output of the svm_learn program

svm_classify is run to perform the classification on the test set which consists of 600 examples. As seen in figure 8, the accuracy of the classifier was 97.67 having 14 incorrect results out of 600 tests. The precision on the set test was 96.43% and the recall was 99%



```
Reading model...OK. (878 support vectors read)
Classifying test examples..100..200..300..400..500..600..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 97.67% (586 correct, 14 incorrect, 600 total)
Precision/recall on test set: 96.43%/99.00%
```

Figure 8: Output of the svm_learn program

Question SVMLight2

Create your own example modeled after the "Inductive SVM" example

1. pick a topic (e.g., "Australia") and provide 100 positive and 100 negative examples for training data:

    - using the Reuters-21578 collection (linked from the SVMlight page)

- or, create your own collection with crawled web pages

2. pick 30 documents not in the training set for your test data
3. stem the words in the collection, using TFIDF as the features (compute for the 230 documents)
4. train, classify, and discuss the results

Answer

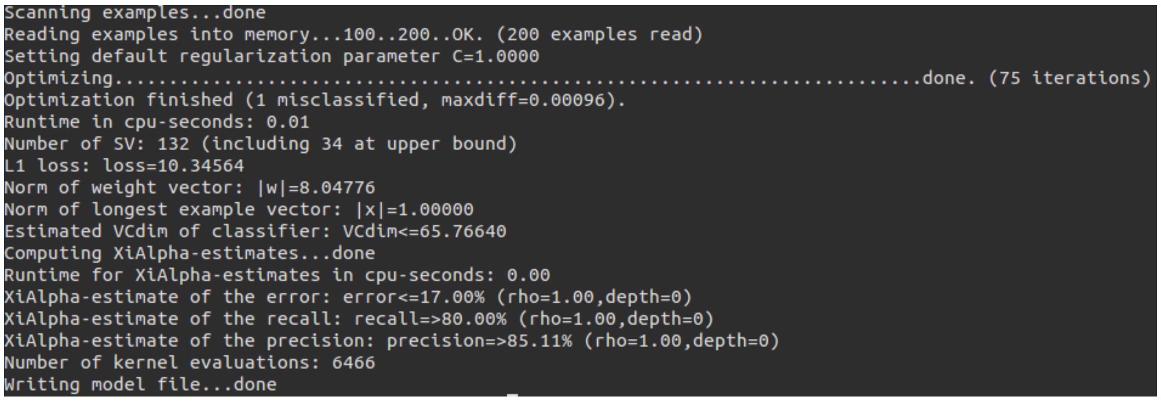
To answer this question, I used nltk.corpus library and its Reuters-21578 collection.This library provides an API that can be used to extract the words, topics, raw text and other information about each of the documents. I picked as the word “Oil”. To extract the 100 positive examples for training, I used the function that returns all the words in a document for each document in the collection and then checked if the word “Oil” is part of them. For the negative examples, I used the documents that did not include this word at all.
I kept the first 100 of the results for both cases as training set and another 15 from each one for the testing set. The TDIDF values of the features were then computed using the sklearn library and the tokenize function found in 3. Once those values are computed for each document, the results are written in a file ”train_data.dat” for the training data and ”test_data.dat” for the testing data in the format SVMlight requires.
We can now use SVMLight to perform the training and the classifying using the files generated by the Python script. The output of the programs svm_learn and svm_classify can be seen in Figures 9 and 10, respectively.

```python
from nltk.corpus import reuters
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

cachedStopWords = stopwords.words("english")

def tf_idf(docs):
    tfidf = TfidfVectorizer(tokenizer=tokenize, min_df=1, vocabulary=total_words,
                            use_idf=True, sublinear_tf=True, max_df=0.98,
                            norm='l2')
    return tfidf.fit(docs)

def feature_values(doc, representer):
    doc_representation = representer.transform([doc])
    features = representer.get_feature_names()
    return [(features[index], doc_representation[0, index])
                 for index in doc_representation.nonzero()[1]]

def tokenize(text):
    min_length = 3
    words = map(lambda word: word.lower(), word_tokenize(text));
    words = [word for word in words
                  if word not in cachedStopWords]
    tokens =(list(map(lambda token: PorterStemmer().stem(token),
                  words)));
    p = re.compile('[a-zA-Z]+');
    filtered_tokens =list(filter(lambda token:p.match(token) and len(token)>=
    min_length,
         tokens));
    return filtered_tokens

documentIDs= reuters.fileids()

print documentIDs[0]
```

```
word = 'Oil'
pos_list = []
neg_list = []
total_words = set()
for doc in documentIDs:
  words = reuters.words(doc)
  total_words = total_words.union(set(words))
  if word in words:
    pos_list.append(doc)
  else:
    neg_list.append(doc)
word_map = {}
idx = 1
for word in total_words:
  word_map[word] = idx
  idx +=1

# Write the positive train cases
features = []
for doc in pos_list[:100]:
  features.append(feature_values(reuters.raw(doc),tf_idf(reuters.words(doc))))

with open('train_data.dat','w+') as f:
  for feature in features:
    f.write("+1 ")
    idx_to_value = {}
    for term,value in feature:
      idx_to_value[word_map[term]] = value
    for idx in sorted(idx_to_value):
      f.write(str(idx)+":"+str(idx_to_value[idx])+" ")
    f.write('\n')

# Write the negative train cases
features  = []
for doc in neg_list[:100]:
  features.append(feature_values(reuters.raw(doc),tf_idf(reuters.words(doc))))

with open('train_data.dat','a') as f:
  for feature in features:
    f.write("-1 ")
    idx_to_value = {}
    for term,value in feature:
      idx_to_value[word_map[term]] = value
    for idx in sorted(idx_to_value):
      f.write(str(idx)+":"+str(idx_to_value[idx])+" ")
    f.write('\n')

# Write the positive test cases
features  = []
for doc in pos_list[101:116]:
  features.append(feature_values(reuters.raw(doc),tf_idf(reuters.words(doc))))

with open('test_data.dat','w+') as f:
  for feature in features:
    f.write("+1 ")
    idx_to_value = {}
    for term,value in feature:
      idx_to_value[word_map[term]] = value
    for idx in sorted(idx_to_value):
      f.write(str(idx)+":"+str(idx_to_value[idx])+" ")
    f.write('\n')

# Write the negative test cases
features  = []
for doc in neg_list[101:116]:
  features.append(feature_values(reuters.raw(doc),tf_idf(reuters.words(doc))))
```

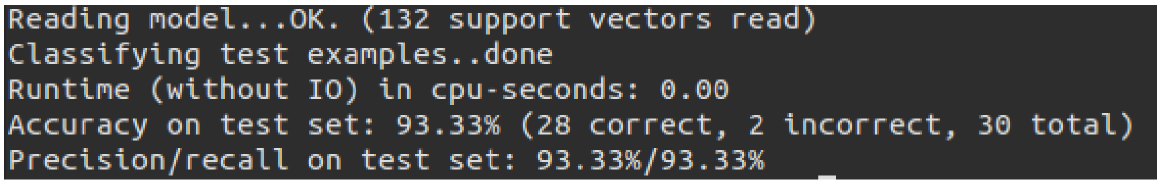
```
with open('test_data.dat','a') as f:
  for feature in features:
    f.write("-1 ")
    idx_to_value = {}
    for term,value in feature:
      idx_to_value[word_map[term]] = value
    for idx in sorted(idx_to_value):
      f.write(str(idx)+":"+str(idx_to_value[idx])+" ")
    f.write('\n')
```

Listing 3: Script to generate the training and test data to feed SVMLight with



```
Scanning examples...done
Reading examples into memory...100..200..OK. (200 examples read)
Setting default regularization parameter C=1.0000
Optimizing..........................................................................done. (75 iterations)
Optimization finished (1 misclassified, maxdiff=0.00096).
Runtime in cpu-seconds: 0.01
Number of SV: 132 (including 34 at upper bound)
L1 loss: loss=10.34564
Norm of weight vector: |w|=8.04776
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=65.76640
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=17.00% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>80.00% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>85.11% (rho=1.00,depth=0)
Number of kernel evaluations: 6466
Writing model file...done
```

Figure 9: Output of the svm_learn for the "Oil" topic



```
Reading model...OK. (132 support vectors read)
Classifying test examples..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 93.33% (28 correct, 2 incorrect, 30 total)
Precision/recall on test set: 93.33%/93.33%
```

Figure 10: Output of the svm_classify for the "Oil" topic

The precision estimation we get according to fig. 9 is 85%. The actual accuracy on the test data however, is 93%, scoring 28 correct classifications out of 30 documents (fig. 10).