

1003197 Sakshi Udeshi

Instructor - Lu Wei

Machine Learning

# Machine Learning Assignment 1 Report

## Formulation as a classification problem (Q1)

For the data to be converted into a vector, we need a vocabulary. I built a function to get the corpus and the word(unigram) data from all the texts to construct a vector of each text.

Now, for each text, I check the the frequency of each word in the vocabulary and build a vector.

An example for the texts ["Julie loves me more than Linda loves me", "Jane likes me more than Julie loves me", He likes basketball more than baseball"] is shown below:

The vocabulary vector is [me, basketball, Julie, baseball, likes, loves, Jane, Linda, He, than, more]

The text is "Julie loves me more than Linda loves me"

Vector for Document 1 is [2, 0, 1, 0, 0, 2, 0, 1, 0, 1, 1]

The text is "Jane likes me more than Julie loves me"

Vector for Document 2 is [2, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1]

The text is "He likes basketball more than baseball"

Vector for Document 3 is [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1]

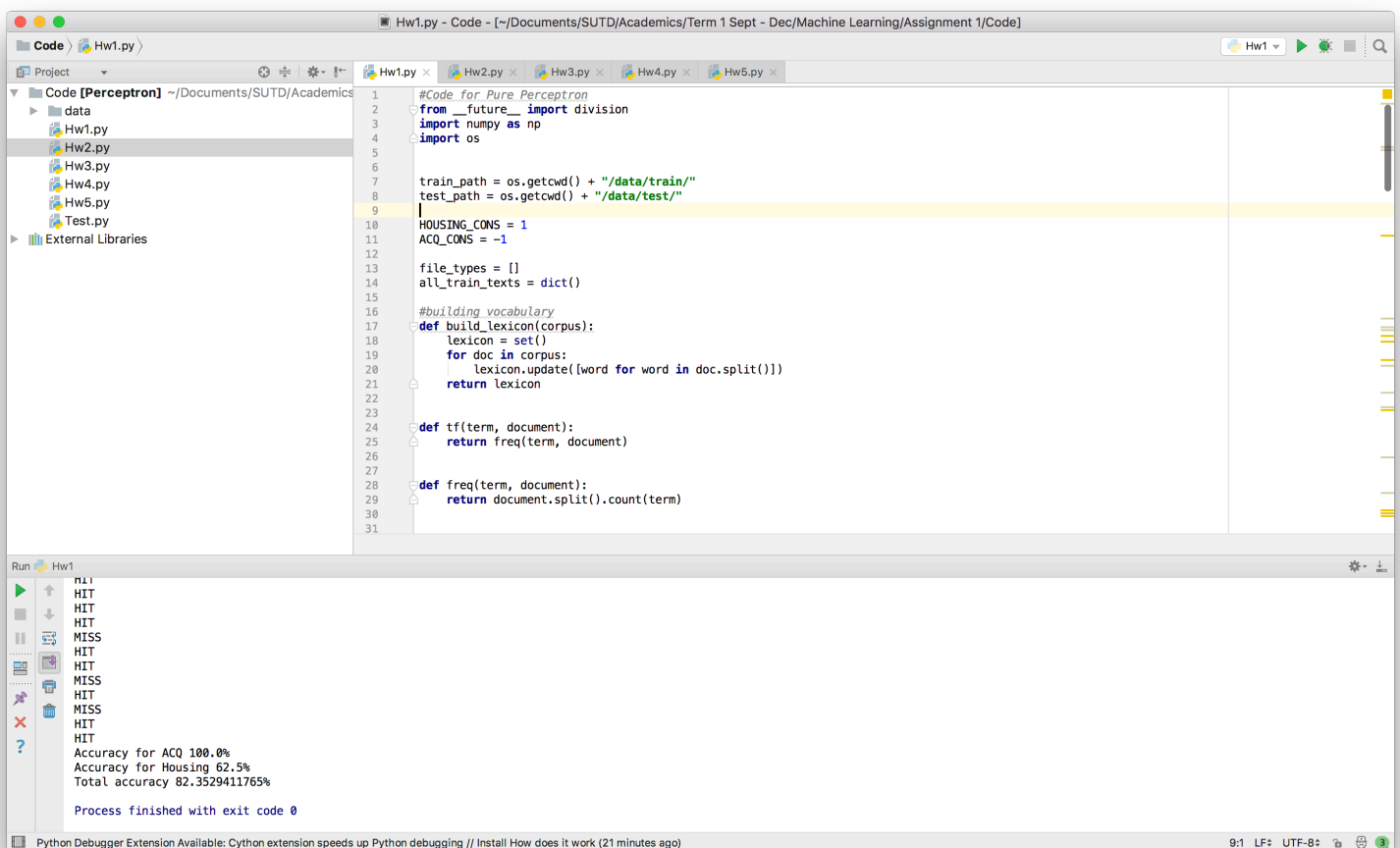
For the particular binary classification problem, I have mapped each of the training vectors to one of two outputs (1 or -1)

## Perceptron Learning Algorithm (Q2)

I initialise the initial  $\bar{\theta}$  vector as a zero vector as discussed in class and run it for all inputs. I put this in a while loop which continues till an instance that none of the training examples change the  $\bar{\theta}$  values (i.e. All the training examples are classified correctly).

I get an overall accuracy of 82.352% on the test data (100 % for acq and 62.5% for housing)

The code can be found [here](#)



The screenshot shows a Python IDE with a project named 'Code [Perceptron]' located at '~/.Documents/SUTD/Academics/Term 1 Sept - Dec/Machine Learning/Assignment 1/Code/'. The code file 'Hw1.py' is open, showing the following code:

```
1 #Code for Pure Perceptron
2 from __future__ import division
3 import numpy as np
4 import os
5
6
7 train_path = os.getcwd() + "/data/train/"
8 test_path = os.getcwd() + "/data/test/"
9
10 HOUSING_CONS = 1
11 ACQ_CONS = -1
12
13 file_types = []
14 all_train_texts = dict()
15
16 #building vocabulary
17 def build_lexicon(corpus):
18     lexicon = set()
19     for doc in corpus:
20         lexicon.update([word for word in doc.split()])
21     return lexicon
22
23
24 def tf(term, document):
25     return freq(term, document)
26
27
28 def freq(term, document):
29     return document.split().count(term)
30
31
```

The Run window shows the following output:

```
Hit
Hit
Hit
Miss
Hit
Hit
Miss
Miss
Hit
Hit
Hit
Hit
Accuracy for ACQ 100.0%
Accuracy for Housing 62.5%
Total accuracy 82.3529411765%
Process finished with exit code 0
```

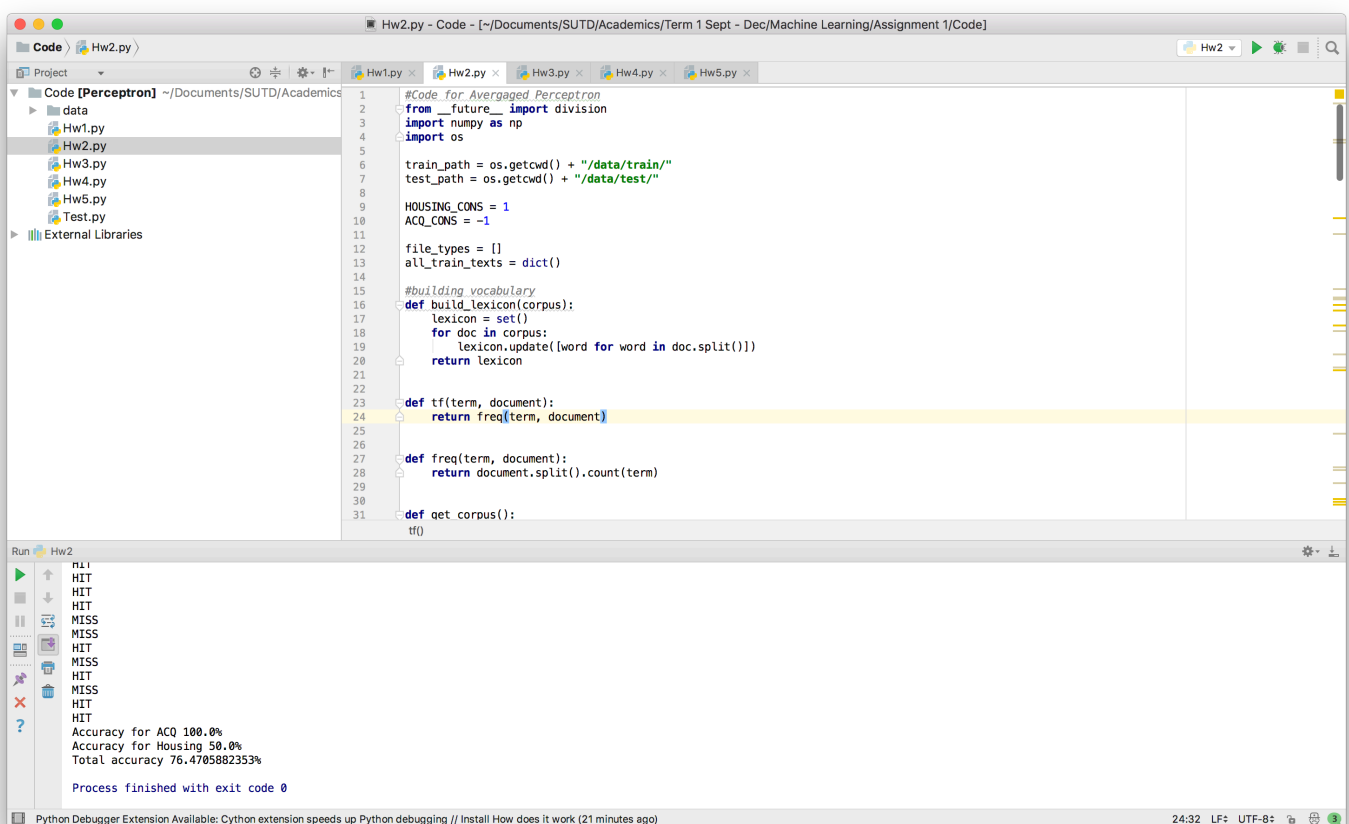
An observation I made while discussing it with a colleague was that, I train the algorithm taking the initial  $\bar{\theta}$  as a zero vector and ordering the input according to type and file name. That is why the code produces a deterministic hyperplane.

## Averaged Perceptron (Q2)

For the averaged perceptron, I have run the normal perceptron algorithm and stored the  $\bar{\theta}$  and values in an array. At the termination I find the average vector and use this to classify the test set. There is a degradation in performance in comparison to the normal perceptron.

I get an overall accuracy of 76.471% on the test data (100 % for acq and 50% for housing)

The code can be found [here](#)



```
1 #Code for Averaged Perceptron
2 from __future__ import division
3 import numpy as np
4 import os
5
6 train_path = os.getcwd() + "/data/train/"
7 test_path = os.getcwd() + "/data/test/"
8
9 HOUSING_CONS = 1
10 ACQ_CONS = -1
11
12 file_types = []
13 all_train_texts = dict()
14
15 #building vocabulary
16 def build_lexicon(corpus):
17     lexicon = set()
18     for doc in corpus:
19         lexicon.update([word for word in doc.split()])
20     return lexicon
21
22
23 def tf(term, document):
24     return freq(term, document)
25
26
27 def freq(term, document):
28     return document.split().count(term)
29
30
31 def get_corpus():
32     tf()
```

Run Hw2

```
111
112 HIT
113 HIT
114 HIT
115 MISS
116 MISS
117 HIT
118 MISS
119 MISS
120 HIT
121 HIT
122 HIT
123 Accuracy for ACQ 100.0%
124 Accuracy for Housing 50.0%
125 Total accuracy 76.4705882353%
126
127 Process finished with exit code 0
```

Python Debugger Extension Available: Cython extension speeds up Python debugging // Install How does it work (21 minutes ago)

24:32 LF: UTF-8: [Icons]

## Stochastic Gradient Descent (Q3)

The objective of stochastic gradient descent is to take steps to minimise the loss function, known as hinge loss.

$$\frac{1}{n} \sum_{t=1}^n \max\{(1 - y^{(t)})(\bar{\theta} \cdot \bar{x}^{(t)} + \theta_0), 0\}$$

With a learning rate of 0.001, I initially tested the function empirically. I stop learning when the loss function goes to zero for the current  $\bar{\theta}$  and  $\theta_0$ . The data seems to have two distinct values for  $\bar{\theta}$  for which the loss function gives goes to zero.

This seems to suggest two minima.

Stochastic Gradient Descent Performance			
	Acq	Housing	Total
Case 1	100%	75%	88.23%
Case 2	100%	87.5%	94.11%

For rates of learning  $< 0.001$  (0.001, 0.00001 etc), the convergence is very slow with no gain in accuracy. For rates of learning  $> 0.001$  (0.01, 0.1, 1 etc), the convergence is faster, but it leads to increased instances of misclassification.

The code can be found [here](#)

Code [Perceptron] ~/Documents/SUTD/Academics

```

109 print len(all_doc_term_matrix_train)
110
111 #train using data
112 train_dict = dict()
113 train_dict[HOUSING_CONS] = housing_doc_term_matrix_train
114 train_dict[ACQ_CONS] = acq_doc_term_matrix_train
115
116 #initializing thetha vector
117 theta_o = 0;
118 theta = []
119 for i in range(0, len(vocabulary)):
120     theta.append(0)
121
122
123
124 for i in range(1, 10000):
125     X = random.choice(all_doc_term_matrix_train)
126     if X in acq_doc_term_matrix_train:
127         Y = ACQ_CONS
128     else:
129         Y = HOUSING_CONS
130     output = result(X)
131     if output*Y <= 1:
132         item = [x*Y*learning_rate for x in X]
133         new_theta = np.add(theta, item)
134
135     theta = new_theta
136     # if (abs(np.mean(np.subtract(theta, new_theta)))):
137
138
139
140
141 for i in range(...) if X in acq_doc...

```

Run Hw3

```

HIT
HIT
HIT
HIT
MISS
HIT
MISS
HIT
HIT
Accuracy for ACQ 100.0%
Accuracy for Housing 75.0%
Total accuracy 88.2352941176%
Process finished with exit code 0

```

Python Debugger Extension Available: Cython extension speeds up Python debugging // Install How does it work (today 8:54 PM)

127:21 LF: UTF-8

Code [Perceptron] ~/Documents/SUTD/Academics

```

123 sum = 0;
124 for X in all_doc_term_matrix_train:
125     if X in acq_doc_term_matrix_train:
126         Y = ACQ_CONS
127     else:
128         Y = HOUSING_CONS
129     output = result(X)
130     sum = sum + max((1 - output*Y), 0)
131 return sum
132
133 while(True):
134     X = random.choice(all_doc_term_matrix_train)
135     if X in acq_doc_term_matrix_train:
136         Y = ACQ_CONS
137     else:
138         Y = HOUSING_CONS
139     output = result(X)
140     if output*Y <= 1:
141         item = [x*Y*learning_rate for x in X]
142         new_theta = np.add(theta, item)
143         theta = new_theta
144
145     if(calc_loss() == 0):
146         break
147
148 all_test_texts = get_files(test_path)
149 acq_hits = test_texts('acq', ACQ_CONS, all_test_texts)
150 housing_hits = test_texts('housing', HOUSING_CONS, all_test_texts)
151
152 housing_tests = len(all_test_texts['housing'])
153

```

Run Hw3

```

HIT
HIT
HIT
HIT
HIT
HIT
HIT
MISS
HIT
HIT
Accuracy for ACQ 100.0%
Accuracy for Housing 87.5%
Total accuracy 94.1176470588%
Process finished with exit code 0

```

Python Debugger Extension Available: Cython extension speeds up Python debugging // Install How does it work (22 minutes ago)

152:2 LF: UTF-8

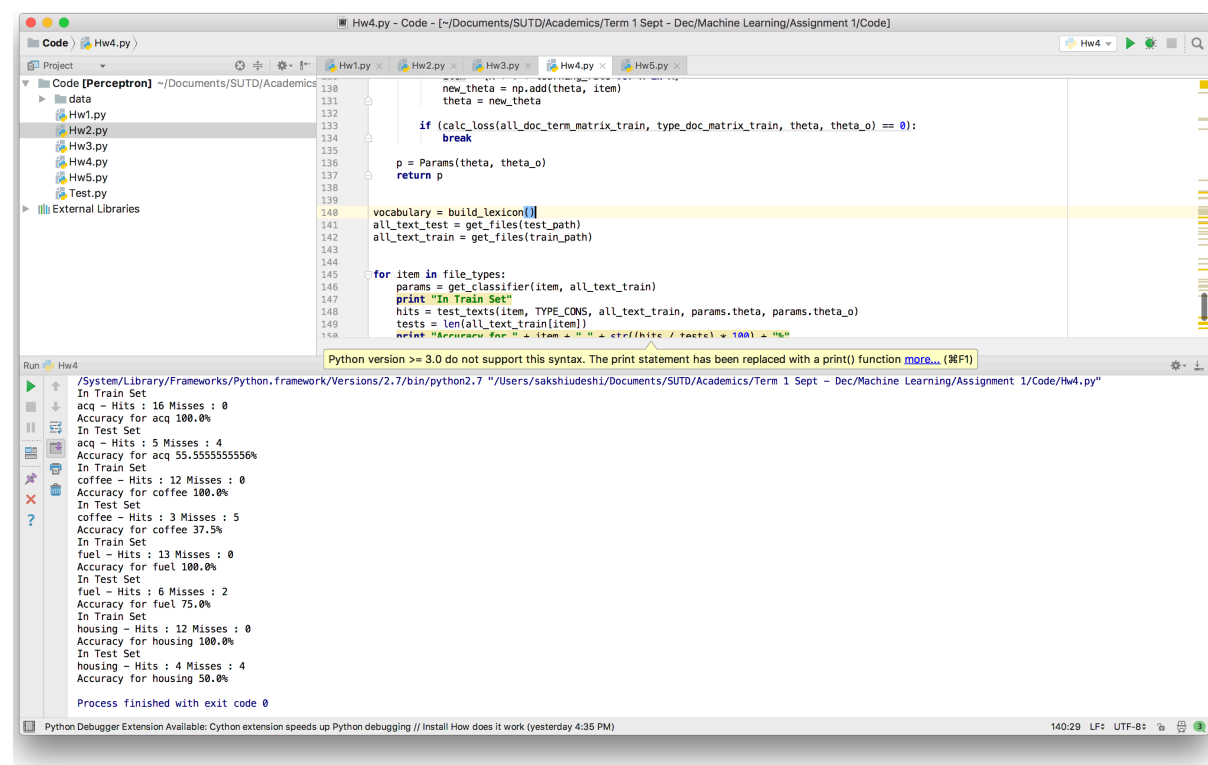
## K-way classification (Q4)

I chose the One Vs Rest approach to transform the problem into a binary classification problem. This involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. As a result we end up with k classifiers for k classes

The results are summarised in the table below

Multiclass Classification				
	Acq	Coffee	Housing	Fuel
Train Set	100%	100%	100%	100%
Test Set	55.55%	37.5%	75%	50%

The performance of the K-way classification turns out to be quite poor as seen.



The screenshot shows a Python IDE with a project named 'Code [Perceptron]' and a file named 'Hw4.py'. The code in 'Hw4.py' is a Python script that implements a One-vs-Rest classifier for four classes: 'Acq', 'Coffee', 'Housing', and 'Fuel'. The script uses the 'Perceptron' class from the 'code' module. It defines a 'vocab' object and a 'params' object. The 'train' function iterates over the training data and updates the weights for each class. The 'test' function iterates over the test data and prints the results for each class. The output of the script is shown in the 'Run' console, displaying the accuracy for each class on both the training and test sets.

```
Python version >= 3.0 do not support this syntax. The print statement has been replaced with a print() function more... (PEP1)
```

```
In Train Set
acq - Hits : 16 Misses : 0
Accuracy for acq 100.0%
In Test Set
acq - Hits : 5 Misses : 4
Accuracy for acq 55.55555556%
In Train Set
coffee - Hits : 12 Misses : 0
Accuracy for coffee 100.0%
In Test Set
coffee - Hits : 3 Misses : 5
Accuracy for coffee 37.5%
In Train Set
fuel - Hits : 13 Misses : 0
Accuracy for fuel 100.0%
In Test Set
fuel - Hits : 6 Misses : 2
Accuracy for fuel 75.0%
In Train Set
housing - Hits : 12 Misses : 0
Accuracy for housing 100.0%
In Test Set
housing - Hits : 4 Misses : 4
Accuracy for housing 50.0%
Process finished with exit code 0
```

The code can be found [here](#).

## Hinge Loss with regularisation (Q5)

The hinge loss now never collapses to 0. Hence now I run the update function a large number of times -  $O(10^3)$  times. I keep track of the  $\bar{\theta}$  and  $\theta^0$  for which the value of the loss function is minimum. As in stochastic gradient descent, I get two cases which I have summarised below.

Hinge Loss with Regularization			
	Acq	Housing	Total
Case 1	100%	75%	88.23%
Case 2	100%	87.5%	94.11%

For values of  $\lambda$  1 and greater, the accuracy suffers. For values of  $\lambda$  lesser and equal to 0.1, the accuracy is the same as the table. For a larger gamma, the initial loss values will be lower, as a result the values of the  $\bar{\theta}$  and  $\theta^0$  that we save do not consider all the training data and consequentially cause test data to be misclassified.

The screenshot shows a Python IDE with a file named 'Hw5.py'. The code defines a function 'min\_theta' that iterates over a range of gamma values (0.001 to 1.0) to find the minimum loss. It uses 'get\_files' to load training and test data. The code calculates accuracy for 'Acq' and 'Housing' classes and prints the results. The output window shows the execution results for gamma = 0.001, 0.01, 0.1, and 1.0, showing that accuracy decreases as gamma increases.

```

162 theta = min_theta
163 theta_o = min_theta_o
164
165 all_test_texts = get_files(test_path)
166
167 print ""
168 print "In Train Set"
169 acq_hits = test_texts('acq', ACQ_CONS, all_train_texts)
170 housing_hits = test_texts('housing', HOUSING_CONS, all_train_texts)
171 acq_tests = len(all_train_texts['acq'])
172 housing_tests = len(all_train_texts['housing'])
173
174 print "Gamma is " + str(gamma)
175 print "Accuracy for ACQ " + str((acq_hits/acq_tests) * 100) + "%"
176 print "Accuracy for Housing " + str((housing_hits/housing_tests) * 100) + "%"
177 tot_acc = ((acq_hits+housing_hits)/(acq_tests+ housing_tests)) * 100
178 print "Total accuracy " + str(tot_acc) + "%"
179
180 print ""
181 print "In Test Set"
182 acq_hits = test_texts('acq', ACQ_CONS, all_test_texts)
183 housing_hits = test_texts('housing', HOUSING_CONS, all_test_texts)
184 acq_tests = len(all_test_texts['acq'])
185 housing_tests = len(all_test_texts['housing'])
186
187 print "Gamma is " + str(gamma)
188 print "Accuracy for ACQ " + str((acq_hits/acq_tests) * 100) + "%"
189 print "Accuracy for Housing " + str((housing_hits/housing_tests) * 100) + "%"

```

Run Hw5

```

Loss is 0.00022215
Loss is 0.00022215

In Train Set
acq - Hits : 16 Misses : 0
housing - Hits : 12 Misses : 0
Gamma is 0.001
Accuracy for ACQ 100.0%
Accuracy for Housing 100.0%
Total accuracy 100.0%

In Test Set
acq - Hits : 9 Misses : 0
housing - Hits : 7 Misses : 1
Gamma is 0.001
Accuracy for ACQ 100.0%
Accuracy for Housing 87.5%
Total accuracy 94.1176470588%

Process finished with exit code 0

```

The code can be found [here](#).

## Vector Representation of Text (Q6)

Bag-of-word model is an orderless document representation—only the counts of words mattered. For instance, in the above example "John likes to watch movies. Mary likes movies too", the bag-of-words representation will not reveal the fact that a person's name is always followed by the verb "likes" in this text. As an alternative, the n-gram model can be used to store this spatial information within the text.

```
[  
  "John likes",  
  "likes to",  
  "to watch",  
  "watch movies",  
  "Mary likes",  
  "likes movies",  
  "movies too",  
]
```

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application.

Interestingly enough, for n-grams upto 8, the performance remains unchanged. This leads me to believe that for this particular case, bag of words is the best option, and other representations do not add much value.

The code for perceptron with n-gram can be found [here](#).



## Other interesting things (Q7)

A interesting thing we could perhaps do with the data is to try and guess the sentiment of the news stories that are provided. We can build a vocabulary of common positive and negative stories. This can be turned into a binary classification problem.

Another thing we can try and do is unsupervised learning on the stories to find clusters of stories and see how well they correlate with the tags. This could help us tune a k-means clustering algorithm we have coded up.

## References

- [https://stanford.edu/~rjweiss/public\\_html/IRiSS2013/text2/notebooks/tfidf.html](https://stanford.edu/~rjweiss/public_html/IRiSS2013/text2/notebooks/tfidf.html)
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Broder, Andrei Z.; Glassman, Steven C.; Manasse, Mark S.; Zweig, Geoffrey (1997). "Syntactic clustering of the web". *Computer Networks and ISDN Systems*. 29 (8): 1157–1166. doi:10.1016/s0169-7552(97)00031-7
- Weinberger, K. Q.; Dasgupta A.; Langford J.; Smola A.; Attenberg, J. (2009). "Feature hashing for large scale multitask learning.". *Proceedings of the 26th Annual International Conference on Machine Learning*: 1113–1120. arXiv:0902.2206
- Youngjoong Ko (2012). "A study of term weighting schemes using class information for text classification". *SIGIR'12*. ACM.