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]

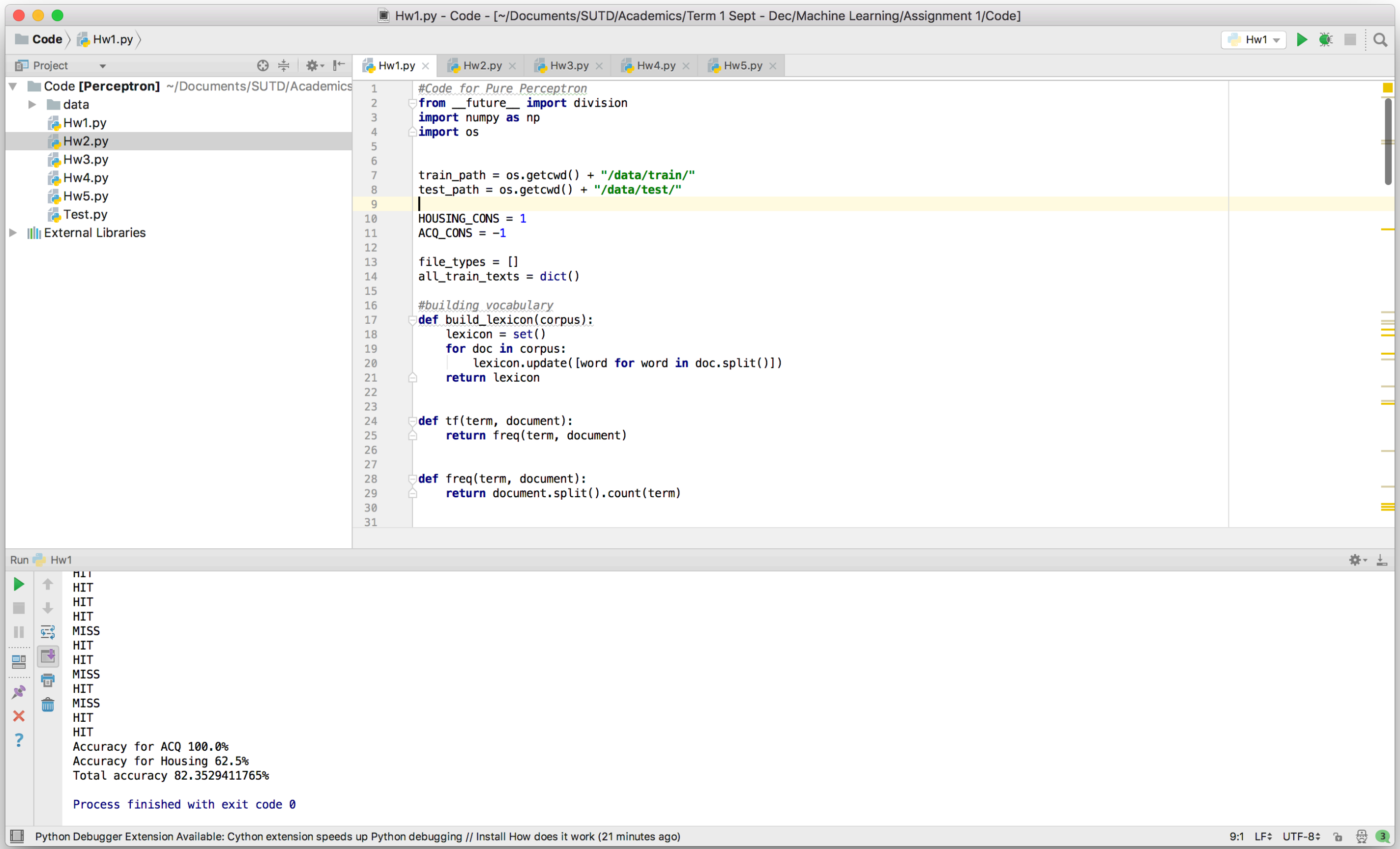
For the particular binary classification problem, I have mapped each of the training vectors to one of two outputs (1 or -1)

Perceptron Learning Algorithm (Q2)

I initialise the initial $\overline{\theta}$ 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 $\overline{\theta}$ 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



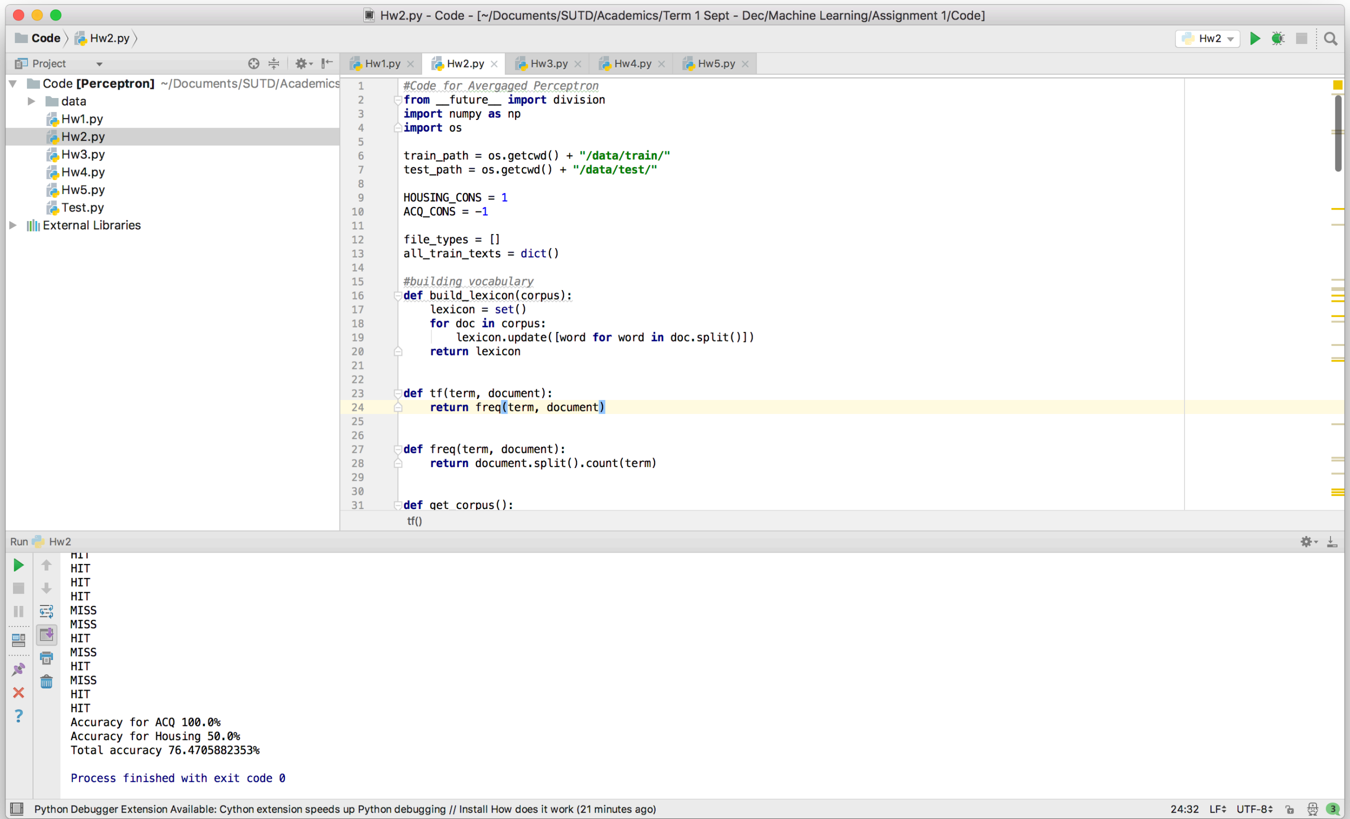
An observation I made while discussing it with a colleague was that, I train the algorithm taking the initial $\overline{\theta}$ as a zero vector and ordering the input according to type and file name. That is why the code produces a deterministic hyperplane.

Averaged Perceptron (Q2)

For the averaged perceptron, I have run the normal perceptron algorithm and stored the $\overline{\theta}$ 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



Stochastic Gradient Descent (Q3)

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

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

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 θ_0. The data seems to have two distinct values for $\bar{\theta}$ for which the loss function gives goes to zero.

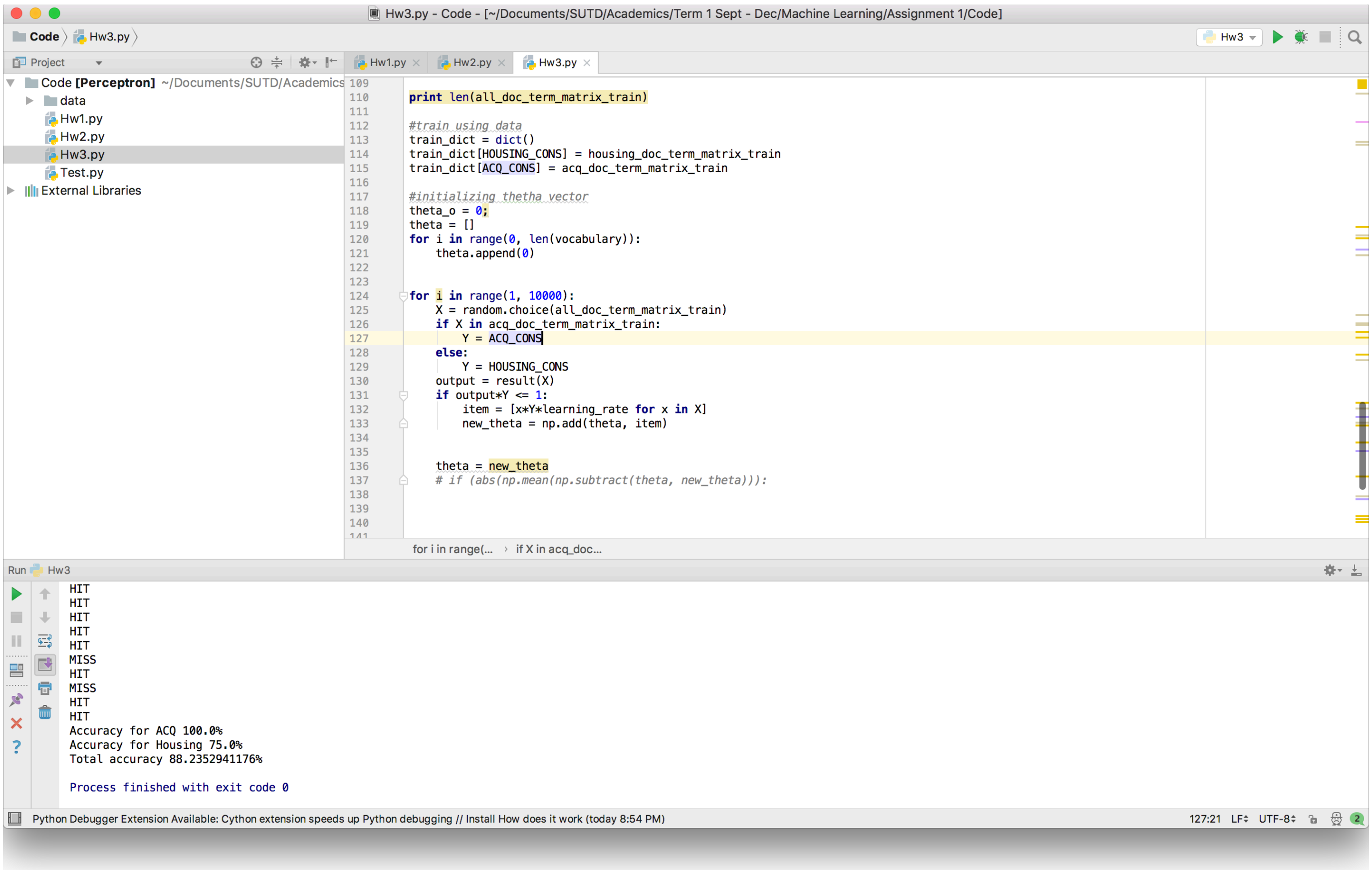
This seems to suggest two minima.

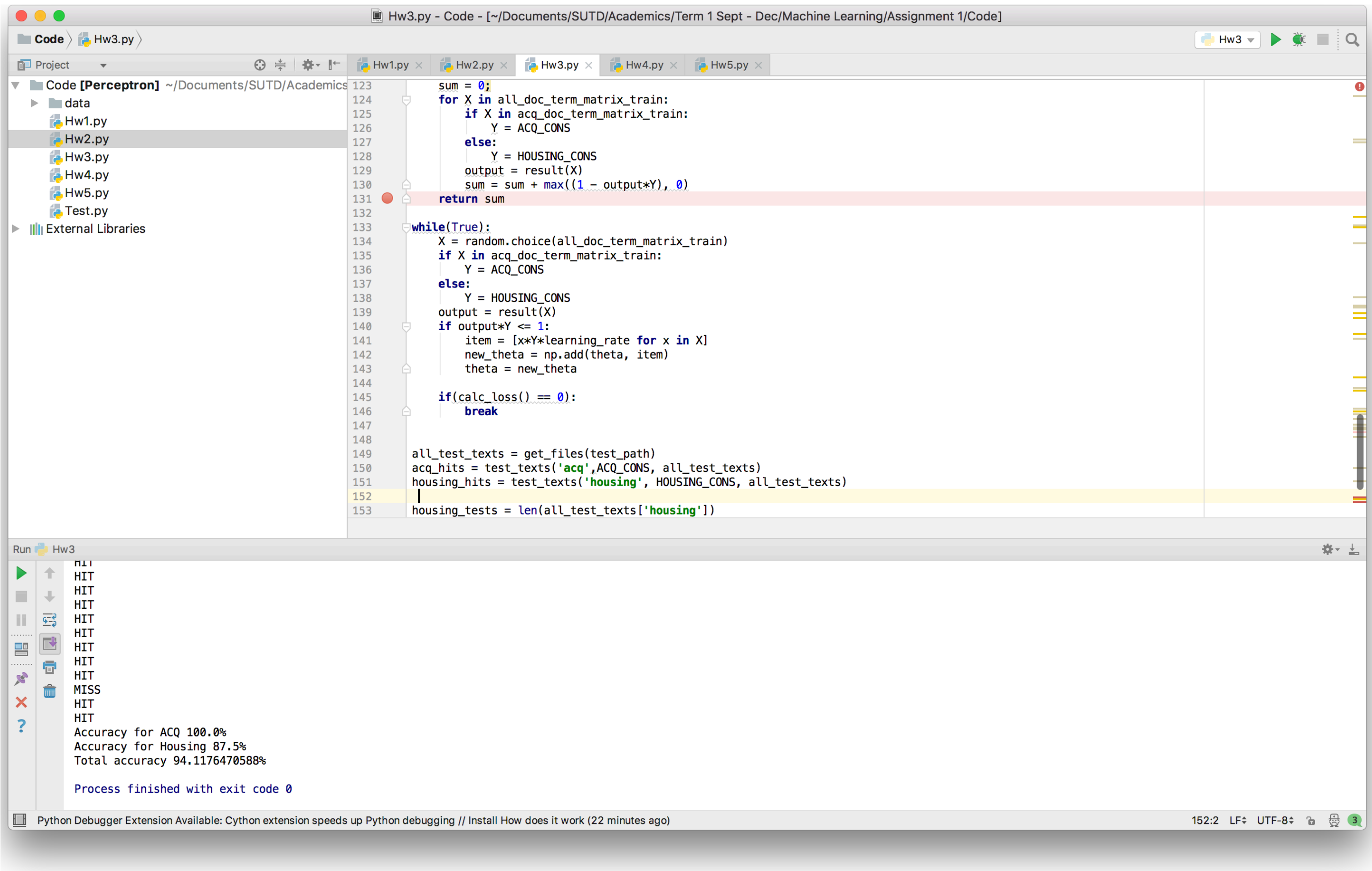
Stochastic Gradient Descent Performance

	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


Hw3.py - Code - [~/Documents/SUTD/Academics/Term 1 Sept - Dec/Machine Learning/Assignment 1/Code]
Code [Perceptron] ~/Documents/SUTD/Academics
data
Hw1.py
Hw2.py
Hw3.py
Test.py
External Libraries
print len(all_doc_term_matrix_train)
#train using data
train_dict = dict()
train_dict[HOUSING_CONS] = housing_doc_term_matrix_train
train_dict[ACQ_CONS] = acq_doc_term_matrix_train
#initializing thetha vector
theta_o = 0;
theta = []
for i in range(0, len(vocabulary)):
theta.append(0)
for i in range(1, 10000):
X = random.choice(all_doc_term_matrix_train)
if X in acq_doc_term_matrix_train:
Y = ACQ_CONS
else:
Y = HOUSING_CONS
output = result(X)
if output*Y <= 1:
item = [x*Y*learning_rate for x in X]
new_theta = np.add(theta, item)
theta = new_theta
# if (abs(np.mean(np.subtract(theta, new_theta))):
HIT
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)


Hw3.py - Code - [~/Documents/SUTD/Academics/Term 1 Sept - Dec/Machine Learning/Assignment 1/Code]
Code [Perceptron] ~/Documents/SUTD/Academics
data
Hw1.py
Hw2.py
Hw3.py
Hw4.py
Hw5.py
Test.py
External Libraries
sum = 0;
for X in all_doc_term_matrix_train:
if X in acq_doc_term_matrix_train:
Y = ACQ_CONS
else:
Y = HOUSING_CONS
output = result(X)
sum = sum + max((1 - output*Y), 0)
return sum
while(True):
X = random.choice(all_doc_term_matrix_train)
if X in acq_doc_term_matrix_train:
Y = ACQ_CONS
else:
Y = HOUSING_CONS
output = result(X)
if output*Y <= 1:
item = [x*Y*learning_rate for x in X]
new_theta = np.add(theta, item)
theta = new_theta
if(calc_loss() == 0):
break
all_test_texts = get_files(test_path)
acq_hits = test_texts('acq',ACQ_CONS, all_test_texts)
housing_hits = test_texts('housing', HOUSING_CONS, all_test_texts)
housing_tests = len(all_test_texts['housing'])
HIT
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)

K-way classification (Q4)

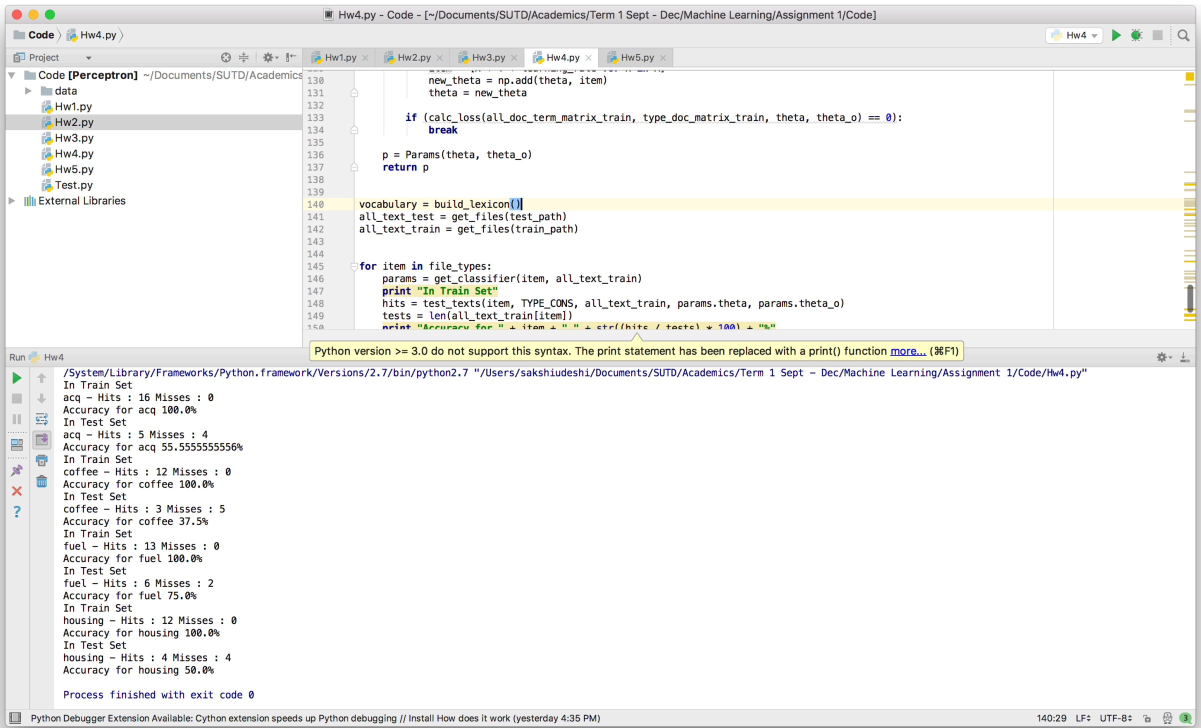
I chose the One Vs Rest approach to transform the problem into a binary classification problem. This involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. As a result we end up with k classifiers for k classes

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.



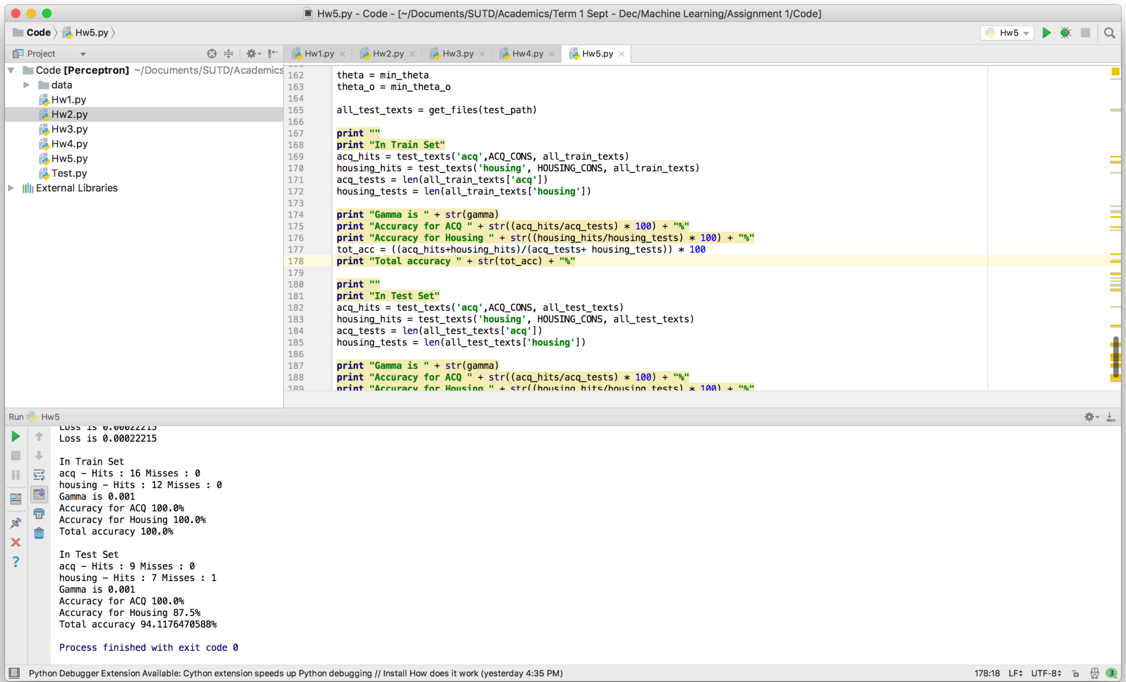
The code can be found here.

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 θ^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 λ 1 and greater, the accuracy suffers. For values of λ 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 θ^0 that we save do not consider all the training data and consequentially cause test data to be misclassified.



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*
- *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.*