Your Name **Chao Zhang**

Your Andrew ID **chaozhan**

Homework 5

0. Statement of Assurance

You must certify that all of the material that you submit is original work that was done only by you. If your report does not have this statement, it will not be graded.

I certify that all of the material that I submit is original work that was done only by myself.

1. Performance evaluation (60%)

1.1 P@10, NDCG@10 and MAP (10%)

Write down the formula of each metric, and give one example for each of them to show that single metric is not enough to guarantee the ranking quality.

$P@N = \frac{R}{N}$, where R is the number of relevant documents in top N results.

$MAP = \frac{\sum_{q \in Q} AP_q}{|Q|} = \frac{\sum_{q \in Q} \frac{1}{r_q} \sum_{i=1}^{r_q} P_q @ n_i}{|Q|}$, where Q is a set of queries, r_q is the number of relevant documents for query q, n_i is the position for i[th] relevant document in the result list, $P_q @ n_i$ is the precision at cut-off at n_i.

$DCG(L_q) = \sum_{i=1}^{|L_q|} \frac{2^{R(d_i,q)} - 1}{\log_2(1+i)}, NDCG(L_q) = \frac{DCG(L_q)}{DCG(L_q^*)}$, where L_q is the ranked list given query q, d_i is the document with rank i, $R(d_i, q)$ is graded relevance and L_q^* is the ideal ranked list given query q.

For example, if the returned list is [I, R, I, I, R, I, I, R] where I indicates an irrelevant document and R indicates an relevant document, and the graded relevance for all relevant documents are all 3. Then P@2 = 0.5, P@5 = 0.4, P@8 = 0.375, MAP = (0.5 + 0.4 + 0.375) / 3 = 0.425, DCG = 9.33, Ideal DCG = 14.92, NDCG = 0.625

If the returned list is [I, I, R, R, R, I, I, I] Then P@3 = 1/3, P@4 = 0.5, P@5 = 0.6, MAP = (0.33 + 0.5 + 0.6) / 3= 0.478, DCG = 9.223, Ideal DCG = 14.92, NDCG = 0.618.

As a conclusion, a single metric is not enough to guarantee the ranking quality. In the examples above, MAP of the second result is higher while NDCG of the first result is higher. And there are good reasons for both of them to be better than the other.

1.2 Performance and time cost table (20%)

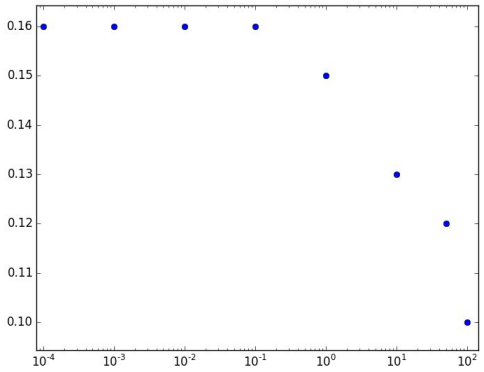
Fill in the table with your experiment results.

C	Logistic Regression			SVM		
	P@10	NDCG@10	MAP	P@10	NDCG@10	MAP
0.0001	0.16	0.28493	0.21903	0.09	0.12107	0.06547
0.001	0.16	0.28493	0.21903	0.07	0.11740	0.08386
0.01	0.16	0.28493	0.21904	0.11	0.19305	0.16331
0.1	0.16	0.28429	0.21825	0.14	0.24868	0.18974
1	0.15	0.27034	0.20963	0.16	0.28452	0.21766
10	0.13	0.22812	0.18404	0.18	0.33832	0.26437
50	0.12	0.19422	0.16587	0.15	0.31449	0.25966
100	0.1	0.18208	0.15831	0.15	0.31312	0.26007

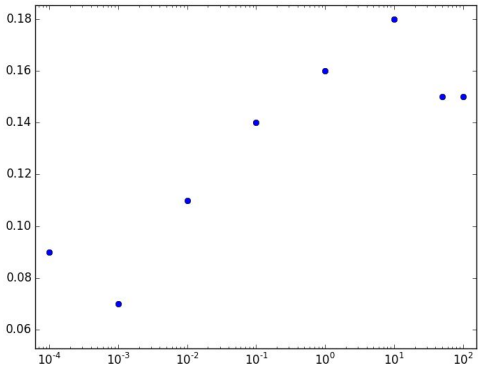
C	Time	
	Logistic Regression (s)	SVM (s)
0.0001	318.008955002	738.445586642
0.001	308.217638969	642.603888713
0.01	322.239006996	445.309245349
0.1	286.843077898	228.616918354
1	176.333677769	53.788590546
10	56.8179368973	30.562038442
50	18.7701699734	34.281250796
100	11.196876049	35.709726224

1.3 Plots (10%)

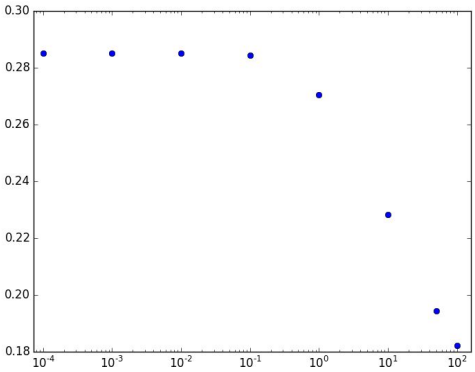
Plots graphs for each metric of Logistic Regression and SVM. You should have six graphs in total.



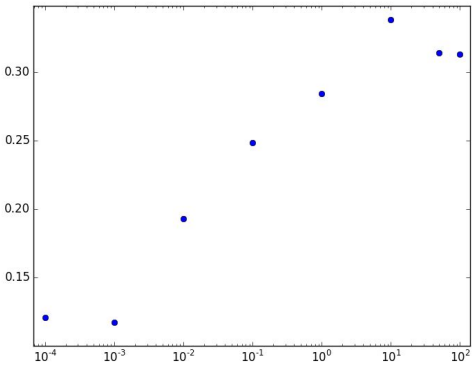
LR P@10



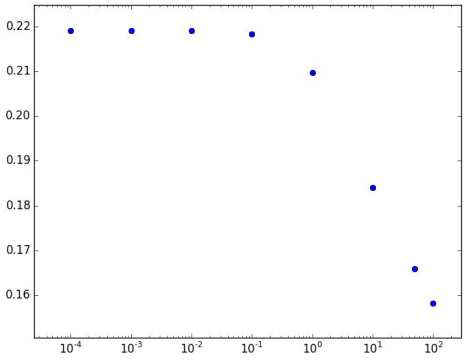
SVM P@10



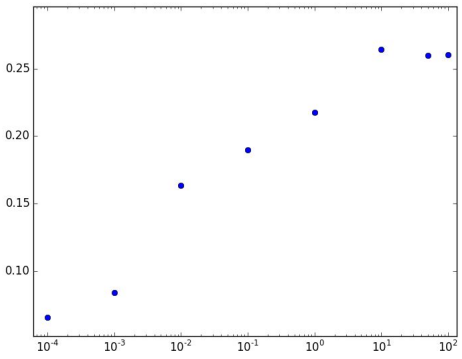
LR NDCG@10



SVM NDCG@10



LR MAP



SVM MAP

1.4 Analysis (20%)

Analysis based on your experiment results (performance and time), and some conclusions (What did you learn from the results? What are the advantages and disadvantages of using Logistic Regression and SVM? etc)

SVMlight's implementation gives less weight on regularization as C increases. While my implementation of Logistic Regression gives more weight on regularization as C increases. For LR, when C goes over 1, all metrics drops significantly. For SVM, when C goes below 1, all metrics drops significantly. LR with C = 0.01 achieves the highest values for all metrics, while SVM with C = 10 achieves the highest values for all metrics. So regularization plays a positive role in preventing the model from overfitting the training data.

SVM with C = 10 performs the best among all settings. It seems that SVM is better in classifying in a relatively small feature space(44 features in HW5, while there are over 10,000 features in HW4) than LR.

The running time for LR decreases as C goes higher. The stopping criteria of my implementation is described in the last part. Basically, if $\|\Delta w\| \leq threshold$,the iteration stops. So when the regularization becomes larger, $\|\Delta w\|$ becomes smaller, resulting in quicker termination of the iteration. The conclusion is that for LR, the running time is positively related to its performance.

It is a different story when analyzing the running time for SVM. The running time is negatively related to its performance. When the performance is worse, the running time increases. I infer that SVMlight's implementation can dynamically adjusts its learning rate. If it arrives at an unsatisfying local optimum, it may restart or jump further instead of being trapped in the local optimum.

Generally speaking, SVM performs significantly better in a small feature space than LR, and it does not perform significantly better in a large feature space than LR(HW4). The running time for SVM is still significantly less than that for LR.

2. Feature Design (25%)

Add three new features that you think might work, and test it. Give some analysis and conclusions based on the experiment results.

I have considered the following new features:

dlsum = dl of body + dl of anchor + dl of URL + dl or body

idfsum = idf of body + idf of anchor + idf of title + idf of URL

tfsum = tf of body + tf of anchor + tf of title + tf of URL

tfidfsum = tfidf of body + tfidf of anchor + tfidf of title + tfidf of URL

bm25sum = bm25 of body + bm25 of anchor + bm25 of extracted title + bm25 of title

tspr2= topic sentitive pagerank * topic sentitive pagerank

I used SVM with C = 10 to see whether there is any improvement. And I used P@10, MAP, NDCG@10 as the metrics. Below are my results:

Features	SVM C = 10		
	P@10	NDCG@10	MAP
44 features	0.18	0.33832	0.26437
44 features + bm25sum	0.18	0.34654	0.27648
44 features + tfidfsum	0.18	0.33832	0.26437
44 features + dlsum	0.18	0.33370	0.26206
44 features + idfsum	0.18	0.33832	0.26437
44 features + tfsum	0.18	0.33997	0.26888
44 features + tspr2	0.18	0.33832	0.26437
44 features + tfsum + bm25sum	0.18	0.34688	0.27713

I added tfsum and bm25sum respectively to the original 44 features, the performances were much better. While other features did not improve the performance a lot, dlsum even lowered the performance. I then tried to added both tfsum and bm25sum to the original 44 features, the performance was further improved.

The initiative for those *sum features was that the title, body, anchor and URL are closely related to each other so it is better to have a standalone feature to represent their sum. That is a possible explanation of why tfsum and bm25sum improve the performance.

I also inferred that dl, idf, tfidf and tspr are not given much weights in the SVM classifier, so the sum of these features did not improve the performance.

The scripts for my experiment are in code/SVM/run2.sh and code/SVM/Implementation/svm_proc2.py

3. The software implementation (5%)

Description of your code including any data structures used, design considerations etc. In addition, please describe any changes you made to your system after HW4.

Programming Language: Python 2

Libraries: Numpy, SciPy, Sckit-learn(Only one preprocessing function is used)

I use ndarray(numpy). So there are no self-implemented data structures.

Description of code

logit.py # logistic regression:

def read_config():

read and parse configuration from 'DATA.TXT'

def L2norm(vector):

 # return a L2-normalized vector

def read_train_data(filename):

 # read training data, returns pairwise data matrix X and class value y

def read_test_data(filename):

 # read testing data, returns data matrix X

def train(X, y, eps=None, threshold=0.01, c=1)

training process, return trained weights

def predict(X, W)

 # predict process, return weights dot data

def main()

#main function

svm_proc.py #preprocessing for SVM

def L2norm(vector):

 # return a L2-normalized vector

def transform_train(infile, outfile):

 # transform train data into pairwise data matrix

def transform_test(infile, outfile):

 # transform train data into L2-normalized data

def main():

#main function

Design Decisions

In my implementation, I decided to choose the learning rate ε as the **following value**

$\varepsilon = 0.00005$

The learning rate should be smaller than that in HW4 because there are fewer features. If ε is set to a larger value, LR classifier will be trapped in an extremely unsatisfying local optimum.

I also set **threshold=0.002** so that the training time is reasonable on Andrew UNIX machine. Setting threshold large may improve the performance a little but at the cost of much more training time.

Because the feature space is relatively small, I switched from sparse matrix into dense matrix. This could reduce the execution time.

All my changes I made after HW4 are labeled **bold**.