EECS 349 Machine Learning

Homework 2

Ding Xiang

Sep. 30 2017

Problem 1

A) We can define a feature set with two simple features, gender and age. The value of gender is chosen from {0, 1, 2}, where 0 means unknown, 1 means female and 2 means male. The value of age is chosen from all non-negative integers. The feature set can capture the users basic information about their gender and ages. So for example's if someone wants to meet a friend on Facebook with the same gender or similar generation, then the user can use this feature set to find the possible group.

B) Denote the vector above as x=(x_1, x_2), where x_1 means the gender, x_2 means the age, $x_1 \in \{1, 2, 3\}$ and integer $x_2 \geq 0$. Define the metric as $d(x, y) = |x_1 - y_1| + |x_2 - y_2|/100$.

Proof: (1) Reflexivity: $d(x, y) = 0$ only when the two abstract value equals to 0, i.e. $x_1 = y_1$ and $x_2 = y_2$.
(2) Non-negativity: $d(x, y) \geq 0$ is obvious since both of the abstract values are non-negative.
(3) Symmetry: $d(x, y) = d(y, x)$ because changing the

order of terms inside the abstract operation doesn't affect the value.

(4) Triangle inequality: $d(x,y)+d(y,z) \geq d(x,z)$, because

$$\begin{aligned} d(x,y)+d(y,z)-d(x,z) &= (|x_1-y_1|+|y_1-z_1|-|x_1-z_1|) \\ &+ (|x_2-y_2|+|y_2-z_2|-|x_2-z_2|)/100 \geq 0 \end{aligned}$$

Here we use the triangle inequality of real numbers, which says for any two real numbers a, b, we have

$$|a+b| \leq |a|+|b|$$

If we substitute $a = x_i - y_i$, $b = y_i - z_i$, then $a+b = x_i - z_i$, where $i = 1, 2$.

C) Height in feet: (0, 9)

Weight in kilograms: (0, 200)

Number of hairs: (0, 1 million)

No, because if we treat them equally, then the number of hairs will dominate the distance, and the effect of the other two factors will become very small. To cluster people appropriately with these 3 elements, we could use weighted norms. For example, define $d(x,y) = |x_1-y_1|/9 + |x_2-y_2|/200 + |x_3-y_3|/10^6$. By doing this, the distance number will be affected by three elements in a comparable level.

D) If we use "A, G, C, T" to denote adenine, guanine, cytosine, and thymine respectively, then we can represent each strand of DNA by a long string consists of "A, G, C, T". Then by using the metric defined in "the

String to String Correction Problem", the distance between two strands of DNA is defined by the edit distance between these two stings. It needs to be adapted. This edit distance obviously ignores the meaning and importance of other factors behind biology. For example, in different parts of a DNA strings, the effect of changing the bases may have different importance. So simply defining the distance by "edit distance" between two strings can not reflect such difference of importance in biological meaning.

Problem 2

A) and B) Please see file 'spellcheck.py'.
C) Please see file 'measureError.py'

Problem 3

A) It's quite long to test the exact total time of spell checking for all the words in the list. Here we choose some sample words and try to find the average checking time and estimate the total checking time. The file of sample words named wikicompact.txt, which includes 10 words and totally 77 letters. The total checking time is 75.01 seconds. So in average, each word takes 0.974 second. The file wikipediatypo.txt contains totally 37543 letters. So the estimated time of spell checking is about 36568 seconds, i.e. 10.2 hours.

By trying 64 combination of parameters, we get the following results for a case with 2 words and 19 letters:

Cost Combinations	Simulation Time (s)	Estimated Total Time (h)
0 0 0	19.105396986	10.21915284
0 0 1	18.2344150543	9.753279374
0 0 2	18.6562139988	9.978892476
0 0 4	19.1421341896	10.23880295
0 1 0	18.7056679726	10.00534457
0 1 1	18.4412839413	9.863930033
0 1 2	17.8440411091	9.544474971
0 1 4	18.8136970997	10.06312752
0 2 0	19.2421860695	10.292319
0 2 1	19.0063450336	10.16617163
0 2 2	19.7832138538	10.58170559
0 2 4	20.5839488506	11.01000516
0 4 0	19.3312690258	10.33996796
0 4 1	18.5028231144	9.896846293
0 4 2	18.4190788269	9.852052894
0 4 4	18.6309711933	9.965390527
1 0 0	18.7250640392	10.0157192
1 0 1	18.2082669735	9.739293209
1 0 2	18.3104760647	9.793963118
1 0 4	18.4840750694	9.886818282
1 1 0	18.7727608681	10.04123142
1 1 1	18.5564141273	9.925511217
1 1 2	18.3570878506	9.818894972
1 1 4	20.1138250828	10.75854393
1 2 0	22.3969349861	11.97974069
1 2 1	18.695152998	9.999720286
1 2 2	20.3174989223	10.86748561
1 2 4	19.2399098873	10.29110151

1 4 0	19.1588208675	10.24772837
1 4 1	18.7992420197	10.05539574
1 4 2	18.5712530613	9.933448311
1 4 4	18.5815989971	9.938982177
2 0 0	19.6178040504	10.49323069
2 0 1	19.7755520344	10.57760741
2 0 2	19.5157999992	10.43867045
2 0 4	19.2752652168	10.31001247
2 1 0	18.4189510345	9.85198454
2 1 1	18.5135309696	9.902573743
2 1 2	19.6809492111	10.52700596
2 1 4	21.4363620281	11.46594651
2 2 0	20.3338499069	10.87623147
2 2 1	19.1014339924	10.2170331
2 2 2	18.897452116	10.10792665
2 2 4	18.8658680916	10.09103289
2 4 0	18.424489975	9.854947226
2 4 1	18.3447101116	9.81227433
2 4 2	19.2232608795	10.28219624
2 4 4	18.8596940041	10.08773048
4 0 0	18.6741230488	9.988471723
4 0 1	18.7303590775	10.01855142
4 0 2	19.8847708702	10.63602671
4 0 4	20.8415389061	11.14778571
4 1 0	20.487169981	10.95823978
4 1 1	18.5561671257	9.9253791
4 1 2	18.4582588673	9.873009633
4 1 4	20.2862341404	10.85076261
4 2 0	19.8944549561	10.64120656

4 2 1	18.7182149887	10.01205575
4 2 2	18.3969330788	9.840207509
4 2 4	18.7607150078	10.03478829
4 4 0	20.6274790764	11.03328874
4 4 1	19.9289519787	10.65965844
4 4 2	19.8857469559	10.6365488
4 4 4	20.3343880177	10.8765193

In the table above, we highlight three different cases:
Yellow Case: It has the minimal estimated total time, where the cost combination is (0, 1, 2) and the estimated total time is about 9.54 hours.
Green Case: It has the second minimal estimated total time, where the cost combination is (1, 1, 2) and the estimated total time is about 9.82 hours.
Red Case: It has the largest estimated total time, where the cost combination is (1, 2, 0) and the estimated total time is about 11.98 hours.

The difference between yellow case and green case is that yellow case regards insertion and deletion cost as 0, whereas green case regards all costs are positive. So considering the meaning of the word distance, we think it is necessary and reasonable to set all costs with positive value. So we chose the cost combination (1, 1, 2) as the best combination for problem A, the corresponding total spell check time is about 9.82 hours.

B) If we need to make the running time of our system under 1 hour, then we may think of more improvements in the design of this experiment.

Improvement 1: Data subsample. Randomly choose samples from the file “*wikipediatypoclean.txt”* which is a portion of typo list from *“wikipediatypo.txt”*. The words only start with letters “a”, “b” and “c”.

Improvement 2: Dictionary subsample. To decrease the running time we could get rid of redundant searching work. So for dictionary we could pick all the words starting with letters “a”, “b” and “c” to improve the searching efficiency.

Improvement 3: Algorithm improvement. Because we said it’s more reasonable to set all cost with positive values than to set cost as 0, we will take advantage of this property to improve our algorithm. Note that most of time is consumed in calculating Levenshtein distance from every dictionary word to a given typo word. But in fact, there are some cases in which the calculating is not necessary, in another word, in certain cases it’s very obvious that some words in dictionary is not possible be the closest word and we don’t need to calculate Levenshtein distance. For example, we try to find the closest word x_i in dictionary to a given word e with typo. If the updated lowest Levenshtein distance is d_i, then for all the following words x_k in the dictionary list (where $k > i$) if the difference between the total number of letters in word x_k and the total number of letters in word e is more than d_i, then there’s no need to calculate Levenshtein distance, because the distance must be larger than d_i. This would significantly reduce the running time.

So finally, the experiment code is in file "*experimentDesign.py*".

By doing the above improvement, for a randomly picking 10 words case, the running time of spell checking in the improved system is 12 times faster than the original system in problem A. The different cost combination and corresponding running time for such "random 10 words case" and estimated total running time for "the whole words with typo case" are listed below.

Cost Combinations	**Simulation Time (s)**	**Estimated Total Time (h)**
1 1 1	7.47356700897	0.903678811
1 1 2	8.64105701447	1.044847811
1 1 4	8.68103599548	1.049681936
1 2 1	8.32121396065	1.006173455
1 2 2	9.75876712799	1.179997592
1 2 4	10.2836899757	1.243469513
1 4 1	8.96733903885	1.084300745
1 4 2	11.1259450912	1.345312194
1 4 4	11.5757129192	1.39969662
2 1 1	10.0566248894	1.21601356
2 1 2	11.6539101601	1.40915197
2 1 4	12.1809849739	1.4728841
2 2 1	10.7406251431	1.29872059
2 2 2	12.0903818607	1.461928673
2 2 4	12.9173071384	1.561917721
2 4 1	11.1968140602	1.353881433

2 4 2	12.9289958477	1.563331081
2 4 4	13.9075558186	1.681655291
4 1 1	10.4037959576	1.257992328
4 1 2	13.2758870125	1.605276005
4 1 4	14.3702111244	1.737598028
4 2 1	11.2559480667	1.36103172
4 2 2	14.185806036	1.71530038
4 2 4	15.4735147953	1.871005831
4 4 1	11.8780348301	1.436252378
4 4 2	15.1264050007	1.829034471
4 4 4	15.9396321774	1.927367191

From the table above, it's quite clear that the best cost combination is (1, 1, 1) which is highlighted by green bar. It takes about 0.9 hour to finish all the spell checking work. The worst combination is (4, 4, 4) with red bar, which takes almost 2 times of running time than that of (1, 1, 1).

C) The graphs are shown below, the best combination of insertion, deletion and substitution costs is (1, 1, 1).

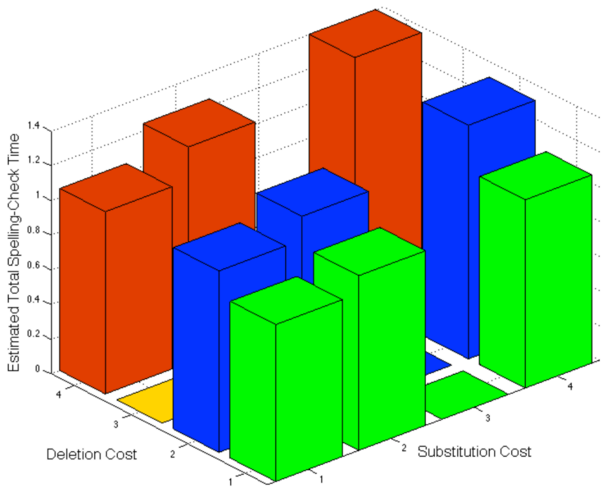

Figure 1. Experiment results when insertion cost is 1

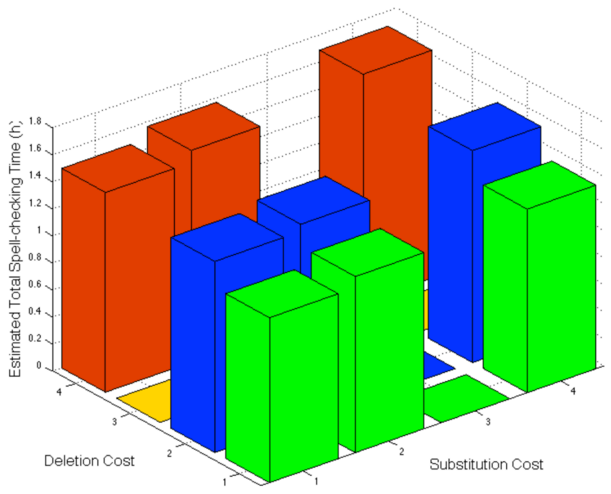

Figure 2. Experiment results when insertion cost is 2

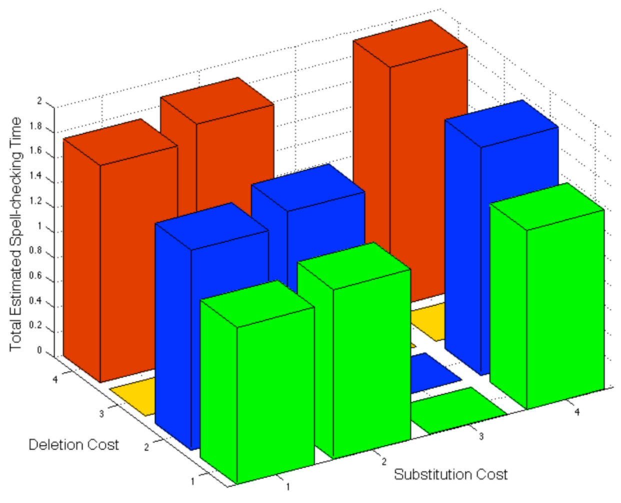

Figure 3. Experiment results when insertion cost is 3

So the best combination happens in the first figure when insertion cost is 1, the deletion cost and substitution cost are 1 and 1 respectively.

Problem 4

A) Please see code '*betterDistance4A.py*'. The basic principle of calculating Manhattan is to locate the position of two letters that will be compared. Here I create 2 matrices for lower case and upper case respectively. Exactly correspond to the position of QWERTY keyboard buttons. So the first row has "123456789", the second row has "qwertyuiop", the third has "asdfghjkl" and the last row has "zxcvbnm". Similarly build upper case matrix.

B) I already wrote it into the code of problem 4A, i.e. file '*betterDistance4A.py'.* The dictionary word list is the still

‘*3esl.txt*’ with first letter ‘a’, ‘b’ and ‘c’. The typo words waiting for spell check are randomly chosen 10 from file ‘*wikipediatypoclean.txt*’. The total performance is shown in the table and graph below.

Cost Combinations (Deletion cost, Insertion cost)	**Simulation Time (s)**	**Estimated Total Time (h)**	**Error Rate**
1 1	23.6130669117	2. 769943932	0.2
1 2	26.6272439957	3. 12352365	0.6
1 4	28.420042038	3. 33382882	0.8
2 1	28.5701169968	3. 351433447	0.1
2 2	34.414000988	4. 036953505	0.2
2 4	39.547755003	4. 639171372	0.5
4 1	36.5105359554	4. 282888704	0.4
4 2	41.9361422062	4. 919342459	0.4
4 4	45.0082089901	5. 27971296	0.5

From the above table, we can see the cost combinations with shortest running time is (1, 1) in green, and the cost combinations with lowest error rate is (2, 1) in yellow. So there’s a small trade-off between running time and error rate. The results are also shown in the following graph.

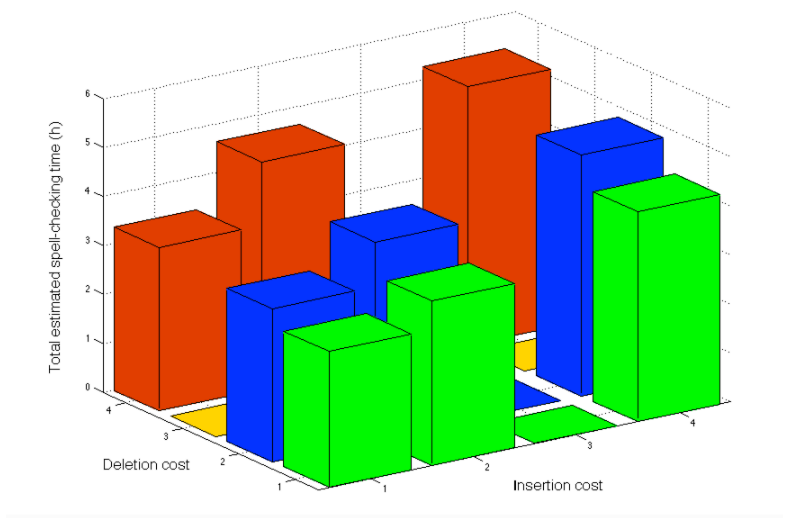

Figure 4. Total estimated spell-checking time under different combinations of deletion and insertion cost

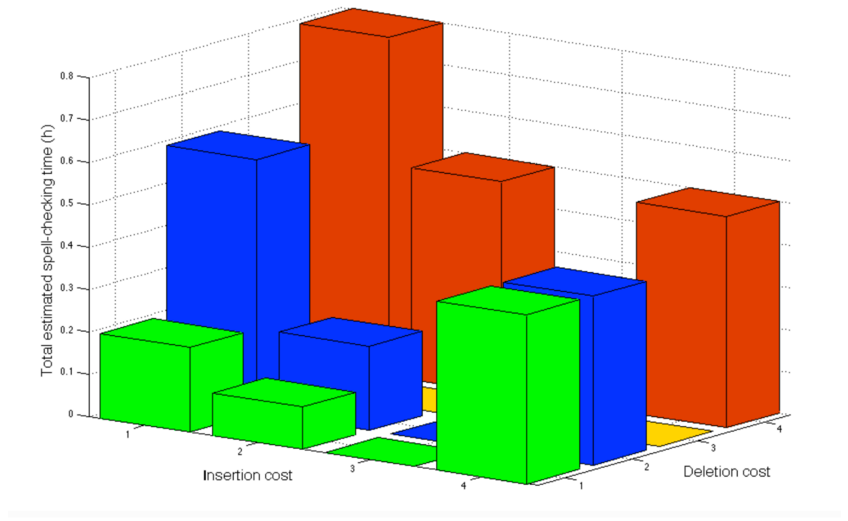

Figure 5. Error rate of spell checking under different combinations of deletion and insertion cost

Considering the trade-off between running time and error rate, we prefer to lower the running time by about 1 hour, and a little bit increase the error rate by 0.1. So the best combination of insertion, deletion cost, in our opinion, is still (1, 1) under the QWERTY distance case. Comparing QWERTY Levenshtein distance and Levenshtein distance, we find that **in terms of total running time spell-checking, Levenshtein distance is faster than QWERTY distance**; however **in terms of error rate it becomes vague**, because whether error rates for QWERTY distance is lower than Levenshtein distance totally depends on the type of errors. For example, if the errors are mis-typing letters and both insertion and deletion cost are 1, then QWERTY may not regard a true word as a closet word than another one in dictionary just by deleting and inserting some spaces from the mis-typed word.