SECOND-ORDER HMM FOR TYPOS CORRECTION

INFERENCE ALGORITHM

MEMBERS
VU Trong Bach
NGO HO Anh Khoa
Mahmut CAVDAR

1. **(3pts) Dry run: Train a first-order HMM using the training data. This is basically what we did in lab sessions for POS-tagging. Compute the error rate (at the character level) and compare this results with the dummiest classifier that just do nothing. You can also compute the number of errors your model can correct and the number of errors your model creates.**

2. **(7pts) Second Order HMM: To improve the performance, we can increase the order of the HMM. Implement a second Order model for this task (this means that the probability of the next state depends on the current state and the previous one as well). A convenient way to implement a second order HMM, is to think about it as a variable change.**

3. **(5pts) Critics and evolution: This model is somehow limited. For instance, it only handles substitution errors. Could you describe a way to extend this model to also handle noisy insertion of characters?**

4. **(5pts) Same question for deletion (or omitted characters)?**

5. **(5pts) Unsupervised training: Propose and discuss some ideas to do unsupervised training for this task. For this question you should provide details on: what kind of data, which parameters will be involved, ...**

QUESTION 1 & 2:

Typos correction is statistical solved by HMM with Viterbi Algorithm, which is based correct letters and actual letters. In this case, correct letters are states and actual letters are observations. These elements are used in HMM for providing the states sequence from the input observation sequence.

The data used for training is a text document pre-processed. In fact, the data includes lower case letters and space character and it is stored in the form of array. An entity is an array of two letters being correct and actual letter. The dataset has the training and testing set with 10% error and 20% error.

HMM is an approach of sequential pattern recognition and described as a matrix of states with the probability of state transition and observation. With N distinct states, q_t is the state of the time instance t. In the first order Markov chain, the state transition probability depends only on its previous state. Therefore, $P(q_t = i \mid q_{t-1}, q_{t-2} \dots) = P(q_t \mid q_{t-1})$ and a state transition $a_{i,j} = P(q_t = j \mid q_{t-1} = i)$. The the second order Markov, the state transition probability depends on the two previous states $a_{k,i,j} = P(q_t = j \mid q_{t-1} = i, q_{t-2} = k)$.

These probabilities are represented by a transition probability matrix A and initial state distribution vector π.

$$A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N1} & \dots & a_{NN} \end{vmatrix} \qquad \pi = \begin{vmatrix} \pi_1 \\ \pi_2 \\ \dots \\ \pi_N \end{vmatrix}$$

In addition, HMM has observation probability $b_j(o_t) = P(o_t \mid q_t = j)$. As a result, the model $\lambda = (A, B, \pi)$.

The task is predicting correct letters from actual letters. In this case, with the observation sequence o = ($o_1, o_2, \dots o_T$) and $\lambda = (A, B, \pi)$, the task becomes how to choose the corresponding state sequence q = ($q_1, q_2, \dots q_t$) in an optimal sense.

Based on dynamic programming, Viterbi algorithm provides the result sequence by using the maximization $\delta t(i) = \max P(q_1 q_2 \dots q_{t-1}, q_t = i, o_1 o_2 \dots o_t | \lambda)$.

In the case of typos correction, with training set and test set 10% of error, the number of errors that the model first HMM correct is 0.9443989 and 0.951366.

QUESTION 3 AND 4: HMM EXTENDED FOR INSERTION, DELETION AND SUBSTITUTION [7]

With the above HMM, we can only replace a letter by another letter, but cannot insert or delete a letter. In fact, that really is a problem for correction of a word.
To make more power for HMM, we should extend it by the method following [7].

Instead of using a state space letter by letter, this work supposes a new state space, emission distribution and transition probability distribution for HMM.

1. **State Space**

The state space of HMM in this work, denoted by K, consists of both kmers and (k+1)mers, where a kmers is a substring of a word.
In example, 3mers of the word "information" is {"inf","nfo","for",…"ion"}.

We denote final state space K={3 state spaces $K_1 \cup K_2 \cup K_3$}:

- K_1 as the set of all observed kmers in training set R.
- K_2 as the set of observed (k+1)mers in training set R. Each (k+1)mer, $w \in K2$ is a deletion state, where the last letter w[k+1] is deleted during the word (this space serves to insert a missing letter in a word).
- $K_3 = \{w^{O} : w \in K_1\}$, where w^{O} is a specialized insertion copy of w, (this space serves to delete a superfluous letter in a word).

Let B1= {a-z}(Single bases), B2={aa,bb,…zz} (all pairs of bases) and {O} (the self-transition), then

$$w \gg \beta = \begin{cases} w[(2+\varepsilon(w)):(k+\varepsilon(w))] + \beta, & \beta \in B1 \cup B2 \\ w, & w \in K1, \beta = \mathrm{O} \end{cases}$$

and

$$w\mathrm{O} \gg \beta = \begin{cases} w[2:k] + \beta, & \beta \in B1 \cup B2 \\ w, & \beta = \mathrm{O} \end{cases}$$

Where + is the string concatenation operator and $\varepsilon(w) = 1\{w \in K_2\}$.
To more clearly explain the self-transition, $w^{O} = w[1:k-1]+w[k]+w[k]$.
In example, Inf^{O} = inff, nfo^{O} =nfoo.

2. **Emission Distribution**

If $s_{i,t}$ is the true state (either a kmer or a (k+1)mer) emitting the t-th observed base of the i-th word, $x_{i,t-1}$ is the (t-1)-th observed kmer, and $\beta \in B1$, $y_{i,t}$ is its corresponding word of quality scores that indicate the confidence in the basecall , then for t > k, the emission distribution is

$$f(\mathbf{x}_{i,t}[k] = \beta, \mathbf{y}_{i,t}[k] \mid \mathbf{s}_{i,t}, \mathbf{x}_{i,t-1}) = \underbrace{\varrho(\mathbf{y}_{i,t}[k] \mid \mathbf{s}_{i,t}, \mathbf{x}_{i,t})}_{\text{quality score model}} \underbrace{g(\mathbf{x}_{i,t}[k] = \beta \mid \mathbf{s}_{i,t}, \mathbf{x}_{i,t-1})}_{\text{base emission model}},$$

Quality scores potentially inform on the error state of the current base call.

We can use use Phred [8] quality scores, which consist of about 40 distinct quality scores. We have probability mass functions:

ϱ_1 models quality scores for bases emitted without error

ϱ_2 models quality scores accompanying substitution errors

ϱ_3 models quality scores for bases emitted after a deletion error

ϱ_4 is for quality scores of insertion.

$$\varrho(\mathbf{y}_{i,t}[k] \mid \mathbf{s}_{i,t}, \mathbf{x}_{i,t}) = \begin{cases} \varrho_1(\mathbf{y}_{i,t}[k]) & \text{if } \mathbf{x}_{i,t}[k] = \mathbf{s}_{i,t}[k], \mathbf{s}_{i,t} \in \mathcal{K}_1, \varphi(\mathbf{s}_{i,t}) = 0 \\ \varrho_2(\mathbf{y}_{i,t}[k]) & \text{if } \mathbf{x}_{i,t}[k] \neq \mathbf{s}_{i,t}[k], \mathbf{s}_{i,t} \in \mathcal{K}_1, \varphi(\mathbf{s}_{i,t}) = 0 \\ \varrho_3(\mathbf{y}_{i,t}[k]) & \text{if } \mathbf{s}_{i,t} \in \mathcal{K}_2, \text{ and} \\ \varrho_4(\mathbf{y}_{i,t}[k]) & \text{if } \mathbf{s}_{i,t} \in \mathcal{K}_1, \varphi(\mathbf{s}_{i,t}) = 1, \end{cases}$$

where $\varphi(s_{i,t}) = 1\{s_{i,t} \epsilon K_1^O\}$ is an indicator of insertion copy kmers.

3. Transition Probability Distribution

The transition probability is defined in terms of kmer-to-kmer transition probabilities p($w \gg \beta$ | w) for w ∈ K, $\beta \in B = B1 \cup B2$. Throughout this work the transition from w to β is interchangeably refered to as $w \rightarrow \beta$, $w \rightarrow v$, where $v = w \gg \beta$.

$$\mathrm{p}(w \gg \beta \mid \mathrm{w}) = \begin{cases} (1 - pd - pi * 1\{w \in K1\}) * q(& if\beta \in B1 \\ pd * \mathrm{q}(w \gg \beta[1] \mid \mathrm{w}) * \mathrm{q}((w \gg \beta[1]) \gg \beta[2] \mid \mathrm{w}\ 1{:}\,\mathrm{k}) & if\beta \in B2 \\ pi * 1\{w \in K1\} & if\beta = \emptyset \end{cases}$$

where pd is the deletion error probability and pi is the probability of kmer self transition (insertion error).

4. Modeling

With state space, emission distribution and transition probability distribution, we can compute the HMM with Viterbi algorithm for insertion, deletion and substitution.

5. Conclusion

We’ve proposed a method to extend HMM for resolving the limit of insertion and deletion of HMM.
For the insertion, we can insert a missing letter into a word with help of new spaces kmer and (k+1)mer.
For the deletion, in this work, we can only delete a superfluous letter in a word without a dictionary, if and only if a kmer that consists this letter already exists in the training set and it’s a double letter. I.e. infor**mm**ation => infor**m**ation.

QUESTION 5:

This section describes the application of unsupervised training method in typos correction belonging to non-word error category.

This problem could be solved by using the approach Dictionary Technique mentioned in [Ritika et al, 2003]. The approach Dictionary Lookup Technique, comparing input text words with an external list of correctly spelled words. In fact, the main idea is that a word in the input text is checked against dictionary, its similarities to all words in the dictionary are calculated if this word is not contained in the dictionary, and then the closest words become suggestions of correction. In the automatic correction case, the first word in the result list sorted ascendingly is selected for replacement. As can be seen from this algorithm, the three main matters are dictionary used, word similarity and number of times when word similarity is calculated.

For the choice of dictionary used for this task, [James, 1980] propose an approach of using machine readable form dictionary available. The reasonable size of this dictionary could be 10,000 words for a small community of users. Moreover, using multiple dictionaries, one for each topic area, limits the size of dictionary and user could choose a number of dictionaries pre-determined. A point important is that this dictionary does not use the stemming and lemmatization technique. The technique should be used for storage is Hash Table, which means that a word could be checked by computing its hash address. As a result, the dictionary is a hash table, each entity has two elements namely hash address and list of word having the same hash address.

For calculating word similarity, [Ritika et al, 2003] mention Minimum Edit Distance Technique with Levenshtein, Hamming and Longest Common Subsequence algorithm. Moreover, [Kai et al] uses also an extension of the Levenshtein Damerau-Levenshtein and a n-grams technique on character level. This part describes mainly Levenshtein algorithm whereas others techniques are explained in [Kai et al, 2010] and [Ritika et al, 2003].

Levenshtein-Distance is minimum number of string transformation operations. In [Kai et al], Levenshtein-Distance has three operations being insertion, deletion and permutation. Insertion is adding a character, deletion is omission of a character and permutation is position change between character. In detail of this algorithm, two string X = $x_1, x_2, x_3 \ldots x_m$ and X = $y_1, y_2, y_3 \ldots y_m$ of length m and n respectively are used for procedure of calculating its minimum edit distance. A matrix (n+1) x (m+1) for this recursive distance calculation are efficiently filled by Wagner-Fischer algorithm. Each entry $D_{i,j}$ can be computed in the following way:

$$D_{0,0} = 0$$
$$D_{i,0} = D_{i-1,0} + \text{cost}(x_i, \emptyset), \qquad 1 \le i \le n$$
$$D_{0,j} = D_{0,j-1} + \text{cost}(\emptyset, y_j), \qquad 1 \le j \le m$$
$$D_{i,j} = \min \begin{cases} d_{i-1,j} + \text{cost}(x_i, \emptyset) \\ d_{i,j-1} + \text{cost}(\emptyset, y_j) \\ d_{i-1,j-1} + \text{cost}(x_i, y_j) \end{cases}$$
$$\text{cost}(x_i, y_j) := \begin{cases} 0 \text{ if } x_i = y_j \\ 1 \text{ otherwise} \end{cases}$$

	y_1	y_2	...	y_m
x_1	0	1	...	5
...	...	...	...	...
x_n	5	4	3	2

The limitation of correction system using Levenshtein-Distance is ranking the suggestion word having the same edit distance. Therefore, [Gueddah et al, 2012] propose an adaptation for this distance calculation algorithm with the frequency matrix of the three type errors of the editing operations (Matrix frequency of insertion, deletion and permutation error). In this case, a dataset of error frequency has to be provided for this adaptation. As a result, Each entry $M_{i,j}$ can be computed in the following way:

$$M_{0,0} = 0$$

$$M_{i,0} = M_{i-1,0} + F_{a,j}(x_{i-1}), \quad 1 \le i \le n$$
$$M_{0,j} = M_{0,j-1} + F_{sup}(y_{j-1}), \quad 1 \le j \le m$$
$$M_{i,j} = \min \begin{cases} m_{i-1,j} + \mathrm{cost}(x_i, ø) - F_{a,j}(x_{i-1}) \\ m_{i,j-1} + \mathrm{cost}(ø, y_j) - F_{sup}(y_{j-1}) \\ m_{i-1,j-1} + \mathrm{cost}(x_i, y_j) \end{cases}$$
$$\mathrm{cost}(x_i, y_j) := \begin{cases} 0 \text{ if } x_i = y_j \\ 1 - F_{permut}(x_i/y_j) \text{ otherwise} \end{cases}$$

	y_1	y_2	...	y_m
x_1	0	0.1785	...	0.234
...	...	...	...	...
x_n	0.408	1.234	1.00	0.991

where:

- $F_{a,j}$ (x_i): the error frequency of adding the character x_i in word X
- F_{sup} (y_j): the error frequency of deleting the character y_j in word Y
- F_{permut} (x_i/y_j): the error frequency of permutation the character x_i with y_j.

For reducing the number of times that the similarity between two words is calculated, [Renato et al, 2013] propose using k-medoids clustering algorithm in spell checking with anomalous pattern initialization (API) and partition around medoids (PAM).

PAM is the most common realization of k-medoid clustering using greedy search for minimizing $W(S, M) = \sum_{k=i}^{K} \sum_{i \in S_k} \sum_{v \in V} (y_{iv} - m_{kv})^2$

where:

- S = {s_1, s_2... s_k}: K clusters set
- M = {m_1, m_2... m_k}: Medoid set
- m_k : Medoid represents cluster s_k
- V : Dataset features
- $y_i \in S_k$: Entity with the smalest distance to all others entities of the same cluster.

This algorithm could be explained in three steps below:

1. Select K medoids at random from the dataset, M = {m_1, m_2... m_k}, S <- ø.
2. Update S by assigning each entity in the dataset to the cluster S_k. If S does not change, algorithm breaks and return S and M.
3. Update each m_k to the entity $y_k \in S_k$ having the smallest sum of distances to all other entities in the same cluster and go back to the second step.

Under those circumstances, PAM has two weaknesses: The result depends significantly on the initial random medoids and on the number of clusters. Therefore, the anomalous pattern initialization is proposed by [Renato et al, 2013] for having goods initial medoids. This method of initialization defines m_c (the entity with the smallest sum of distances to all other entities in the dataset) and m_t to the entity farthest away from m_c. This algorithm repeats PAM to find m_t and finally returns initial M for the first step of PAM.

In short, we propose a method based the article of [Renato et al, 2013] in typos correction. Word similarity is computed by the adaptation of Levenshtein-Distance of [Gueddah et al, 2012]. In this case, we use two datasets namely dictionary for clustering method and error dataset for calculating word similarity.

1. Apply the method API and PAM to the dictionary. This step creates a set of medoids M.

2. Use spelling checking for all word in the input by using hash table.
3. For misspelled word w, calculate its distance to each medoids of M and select one medoid m* having the smallest distance.
4. For each entity - word in the cluster of m*, calculate its distance to w and select the word w* having the smallest distance. Break and return word w*.

Another approach, based on non-dictionary technique, comes from one of three important spelling errors, typographical errors on typing, mentioned by [James, 1980]. This problem is related to the position of the keys on the keyboard and finger movement, which leads to errors of large databases. In fact, an error character is probably replaced by characters that its keys are near that error character key. Therefore, its main idea is clustering the characters which could replaced by themselves and then these character groups could be used in Rule Based Technique or HMM with Viterbi Algorithm.

For the task of clustering the characters, a dataset including both the correct words and misspelled words should be created. These words are divided into characters, which shows clearly the relation between replacing or replaced word. Consequently, the character similarity measure $D_{i,j}$ could be computed by the formulas below:

$$D_{i,j} = \frac{C(i \rightarrow j) + C(j \rightarrow i)}{\sum_{k \in A} C(i \rightarrow k) + \sum_{k \in A} C(j \rightarrow k)}$$

Where:

- $D_{i,j}$: Distance – Similarity between two character i and j
- $C(i \rightarrow j)$: Number of times the character j is replaced by i
- $k \in A$: a character of character set on the keyboard without function keys.

Another similarity measure proposed is keyboard distance which defines the distance between two keys in a Standard English Keyboard QWERTY. For example, distance of character “h” to characters “y”, “u”, “j”, “n”, “b”, “g” is 1, to “f” is 2.

Thus, the similarity used in clustering technique is combination of two kind of distance proposed.

For this typos correction method, DBSCAN (Density Based Spatial Clustering of Application with Noise) is proposed. DBSCAN is an algorithm of clustering technique based on density, which determines arbitrary shaped clusters and filters noises. Its main idea is growing a cluster by adding the neighborhood objects which exceed a threshold predefined and by calculating distance between two objects. One positive point of DBSCAN is that it does not require the number of clusters, as opposed to K-Means. In fact, it’s impossible to pre-determine cluster number.

In DBSCAN algorithm, each cluster is expanded with a level of density, which discovers arbitrary shape of the data. In detail, this algorithm needs two threshold parameters: the minimum number of neighborhood object minObjs and the maximum distance between two objects, namely radius epsilon Eps. Therefore, an Eps-Neighborhood is an object within a radius Eps of an object, a Core object contains at least a minimum number of Eps-Neighborhood and a Noise object which is not a Core object are discarded. On the whole, this algorithm could be explained as below:

- An object P which has not been visited is selected and DBSCAN finds all Eps-Neighborhoods.
- If P is a Core object, a cluster including P and its Eps-Neighborhoods is formed and P is marked as visited. DBSCAN repeats this finding Core object step with Eps-Neighborhoods of P.
- If P is a Noise object, DBSCAN visits the next object in the dataset.

In the choice of minObjs, [Kedar et al, 2014] propose an automatic method for calculating this threshold, which means that the result from DBSCAN depends only one parameter Eps. This method uses the average of Eps-Neighborhoods of all objects.

$$\mathrm{minObjs} = \frac{1}{n}\sum_{i=1}^{n} o_i$$

Where:

- o_i: Number of Eps-Neighborhoods of Object i
- n: Number of character in the character set.

After this step with DBSCAN, a list of character groups is provided. For each group, we could use Rule Based Technique for creating rules for the correction operations such as transposition, deletion, insertion and substitution. The use of this technique in spelling correction is mentioned in [Kai et al, 2010], which shows that using more rules will worsen the performance. Therefore, the number of rules and rule definition should be selected by evaluating the probability of the error.

The algorithm could be explained below:

1. Apply DBSCAN with the two distances measures. This step creates a list of character groups.
2. For each word, select groups having its characters.
3. Use the rules in these groups for spelling correction.

For example, a misspelled word “oppulstiion” belongs to this groups {“u”, “i”, “o”, “p”, “l”, “k”, “[“, “0”} having rules such as “op” → “po”, “ii” → “i”. Hence, “oppulstiion” → “populstiion” → “populstion”. With this group {“a”, “s”, “z”, “w”} having rules such as “s” → “a” ..., “populstion” → “population”.

Moreover, for HMM with Viterbi algorithm in practice, this clusters could reduce the number of state characters for each observation character, which improves its performance. Instead of using all of the character states, we use characters being in the same group of the character evaluated.

For example, for a misspelled word “pipulation”, the states for HMM are the characters being in the same cluster of “p”, “i”, “u”, “l”, “a”, “t”, “o” and “n”. The observation “p”, “i”, “u”, “l”, “o” have states of the group {“u”, “i”, “o”, “p”, “l”, “k”, “[“, “0”}. The observation “a” have states of the group {“a”, “s”, “z”, “w”}. The observation “t” have states of the group {“t”, “r”, “y”, “g”}

In brief, this section mentioned two approaches: Dictionary Technique with Levenshtein-Distance - PAM and DBSCAN in typographical errors on typing.

REFERENCES

❖ ***Document***

[1] G. Hicham, Y. Abdallah, B. Mustapha, *Introduction of the weight edition errors in the Levenshtein distance,* International Journal of Advanced Research in Artificial Intelligence (IJARAI)2012.
[2] James L. Peterson, *Computer Programs for Detecting and Correcting Spelling Errors*, Communications of the ACM, Volume 23, Number 12, December 1980.
[3] Kai Niklas, *Unsupervised Post-Correction of OCR Errors,* Diploma Thesis, Hanover, 11th June 2010.
[4] Kedar Sawant, *Adaptive Methods for Determining DBSCAN Parameters,* IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 4, June 2014.
[5] Ritika Mishra, Navjot Kaur, *A Survey of Spelling Error Detection and Correction Techniques,* International Journal of Computer Trends and Technology, volume 4, issue 3, 2013.
[6] Renato Cordeiro de, A. M. Zampieri, *Effective Spell Checking Methods Using Clustering Algorithms,* Proceedings of Recent Advances in Natural Language Processing, pages 172–178, Hissar, Bulgaria, 7-13 September 2013.
[7] Noroozi, V. (2016). Probabilistic insertion, deletion and substitution error correction using Markov inference in next generation sequencing reads.
[8] https://en.wikipedia.org/wiki/Phred_quality_score