titolo

autore

September 3, 2008

Abstract

abstract

1 Introduction

text typing prediction assistance via text prediction

1) Mettiamo in evidenza l'idea che l'uso dei q-grammi per la predizione, con q arbitrario, possibile grazie agli indici compressi. L'approccio dunque senza usi di DB-linguistici, ma sfrutta solo l'evidenza statistica.

in linea di principio usare anche le informazioni linguistiche puo' essere piu' efficace, ma sottolineavo che il ritmo di crescita di capacita' di memoria e di calcolo delle macchine ha un'ordine di grandezza diverso dal ritmo di crescita della dimensione dei corpus linguistici disponibili, quindi da un certo punto in poi (e gia da ora) l'approccio statistico su corpus non taggati sempre piu' grandi sara' piu' efficace degli approcci su corpus taggati.

2 The Problem of statistical prediction enhanced text typing

We consider the text typing process as a sequential human-machine interaction process, where, in response of one or more keystrokes, a text fragment is appended to the already typed text.

In the case of natural language, the user usually doesn't really need to type a particular text but simply wants to communicate a message that can be expressed through several alternative texts (not only the case of alternative phrase construction or alternative synonyms, but punctuation and typesetting choices too): we relax the problem definition of text statistical prediction enhanced text typing assuming that the user needs to type exactly a particular target text $S \in \Sigma^*$ of length $t = |S|$.

In the case of the standard typing environment we assume that the effect of every keystroke is the appending of a single character to the already typed text and that there is a distinct key for every possible character in Σ; therefore, $|S|$ keystrokes are needed in order to produce the target string S.

We want to decrease the number of keystrokes needed in order to produce S, making use of a text statistical prediction system.

2.1 p-keys extended interface

In order to take advantage of a text statistical prediction system, we extend the standard input interface with p additinal keys, associated to variable character sequences.

In every human-machine interaction step, our text statistical prediction system will associate some character sequences to every one of the p additinal keys; if the user finds that one or more of the proposals are exact prefixes of the suffix of the text he need to type, then the additional key associated to the longest correct proposal p_i is pressed and the sequence p_i is appended to the alredy typed text; otherwise, if none of the proposals turns out to be correct, the user press one of the standard interface keys, appending only a single character.

2.2 Linguistic domain

We can fix the linguistic domain of the prediction system defining the perfect corpus C_p as the set of all the possible texts of finite length that the user will eventually write, associated to their relative frequencies.

The perfect corpus assumption is useful for a theoretical formulation of the text statistical prediction problem and for an exact solution analysis; the case of a real corpus and all the related problems that arise will be discussed where some heuristical solutions to the real case will be exposed (section 3.3).

2.3 From the text typing to the path selection paradigm

We can build a trie data structure T from the string elements of the perfect corpus C_p. We append a special string terminator character to every string $s \in C_p$, in order to obtain a distinct leaf for every s.

For every node i of T we define:

- the associated string s_i, as the concatenation of the character labels of the edges that form the path from the root node to the node i.
- the associated relative frequency f_i, as, in the case of leaf nodes, the relative frequency of s_i (without the terminator character) from the perfect corpus C_p, in the case of internal nodes, the sum of the relative frequencies associated to the leaves of the sub-trie rooted by the node i.

Considering the trie T, in the standard typing environment, the text typing process of a target string S can be viewed as a path selection process, starting from the root node, with every keystroke corresponding to the label of the next edge to sequentially append to the path, until the node i with the associated string $s_i = S$ is reached.

Given T, the text statistical prediction enhanced text typing with a p-keys extended interface problem can be formulated as the problem of adding to evey internal node i of T, p additional edges pointing to internal nodes of the sub-trie rooted by i, with associated labels corresponding to the concatenation of the single character labels of the standard edges that form the path from node i to the pointed node.

We will refer to the trie T with all the additional edges as the prediction graph G. We will continue to refer to the root node in T as the root of G, and all the edges are directed toward the root-to-leaves direction.

Thanks to the additional edges, in G there can be several different paths from the root to a node i in addition to the only one in T, and the length of every additional path is less than $|s_i|$.

Following the same paradigm of viewing the text typing as a path selection, all the paths from the root node to node i in G represents all the possible keystroke sequences in the p-keys extended interface that produce s_i, but, given that the user doesn't have the possibility of doing any look-ahead in the prediction graph, the path followed is the greedy one, not necessarily the shortest one.

2.4 Prediction graph optimality conditions

Obviously, some additional edge choices can be better than others, but every definition for the optimality can focus on different properties of the prediction graph. We propose two different optimality conditions:

Total keystrokes saving : minimize the number of keystrokes needed in order to produce the whole corpus, taking into account the corpus text frequencies.

We fulfill this condition minimizing the sum, for all the nodes i such that $s_i \in C_p$, of the product of f_i and the length of the greedy path from the root to the node i.

Mean keystrokes saving ratio : minimize the frequencies weighted mean ratio keystrokes over produced text length for the strings in the corpus.

We fulfill this condition minimizing the sum, for all the nodes i such that $s_i \in C_p$, of the product of f_i and the ratio of the length of the greedy path from the root to the node i over $|s_i|$.

Over all the possible additional edges choices, the optimum choices that maximize the *total keystroke saving* can be different from the choices that maximize the *mean keystrokes saving ratio*. Figure 1 is a simple trie exemple where the optimal choices with $p = 1$ for the two optimality conditions are different: for the case of *total keystroke saving*, the sum to minimize for the left additional edges choice turns out to be 2 and 2.05 for the right choice, for the case of *mean keystrokes saving ratio*, the sum to minimize in the left turns out 0.8 and in the right 0.75.

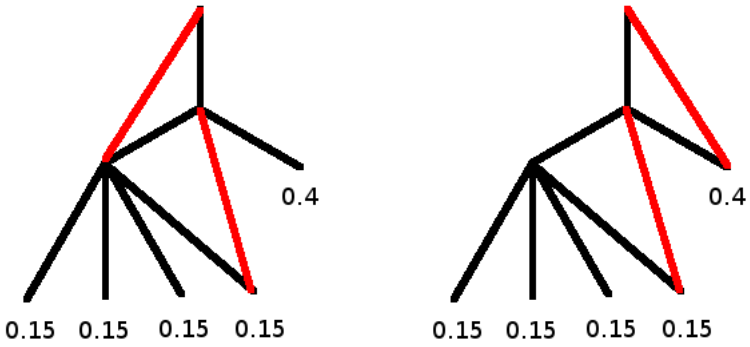

Figure 1: Two different additional edges (the red ones) choices with $p = 1$, the left choice is optimal for the case of *total keystroke saving* optimality condition, the right choice is optimal for the case of *mean keystrokes saving ratio* optimality condition.

Even if the two optimality conditions lead to chose different prediction graphs from the same trie, each of them deserve consideration. The *mean keystrokes saving ratio* condition is in line with the quite common practice to test the performance of predictors with a mean keystroke saving ratio over a set of test texts; adopting this condition will likely make the prediction system score better performance in this typology of tests than the other condition. On the other hand, we consider the *total keystroke saving* a better condition for the abitual user point of view: of course if we assume the use case of a single text typing the *mean keystrokes saving ratio* condition will ensure an higher mean time saving (short time better performance), but in the use case of an abitual user that will adopt the predictor enhanced typing environment for a big number of texts, at the end the *total keystroke saving* condition will give an higher keystroke saving (long time better performance).

Therefore we chose to focus our work on the first condition and we will propose solutions for the problem of the *total keystroke saving* maximization.

2.5 Space Complexity Constrains

The focus of our work is to develop a text statistical prediction system with the lowest possible space complexity, in order to take advantage of huge untagged corpus.

In the section 4 we will describe how we can obtain the trie structure T and the frequencies numbers f_i of the nodes, making use of a compressed index data structure that take only a space that is essentially bounded by the k^{th} order empirical entropy of the corpus. Anyway, we are unable to precompute and store the additional edges in the same space complexity, therefore our system needs to have an algorithm that compute the edges on the fly.

2.6 Problem formulation

Taking into account the space complexity constrains, our target problem became the fast computation of the additional edges of a particular node $a \in T$, such that, executing the computation for every node in T, the resulting prediction graph G satisfy the *total keystroke saving* condition.

3 Problem Solution

The problem of computing the additinal edges choice for a particular node a turns out to be a not trivial task.

From the analysis of the problem we are able to make two observations:

Observation 1 *The optimum additional edges choice for the node a depends on the additional edges choices of all the nodes in the greedy path from the root to a.*

> As previously stated, even if in the prediction graph there can be several alternative paths from the root to node a, the user unability of making any look-ahead in the path selection process will cause that the only possible path to a is the greedy one.
>
> Then, if a node along the root to a greedy path has an additional edge pointing to the node b, and b is in the a-rooted sub-trie, we know that any path going from the root to node a and then to any node in the b-rooted sub-trie isn't a greedy path.
>
> Therefore, all the additional edge choices done for the nodes along the root to a greedy path needs to be considered in order to prune the a-rooted sub-trie when computing the additional edges for node a.
>
> For example, consider the simple case of Figure 2: if we doesn't consider the additional edge of the root, the greedy choice to add a link to the node with 0.6 frequency turns out to be suboptimal.

Observation 2 *Even if we impose a maximum length for the additional edges, in order to chose the optimal additional edges of the node a, in the general case, the structure of the sub-trie rooted by a needs to be analyzed up to the leaves.*

> The fact that the additional edges of the nodes on the greedy path from the root to node a can alter the additional edges choice for a (*observation* 1), imply that the choice done in a will have repercussions on the a-rooted sub-trie.
>
> For example in Figure 3 it's shown how, analyzing the sub-trie up to several incremental depths, different choiches for the root node additional edge are computed as optimal: the example construction can be extended to any depth, therefore the sub-trie needs to be analyzed up to the leaves.

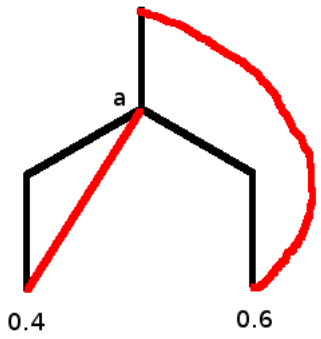

Figure 2: The additional edges choice with $p = 1$ of node a depends on the previous additinal edges.

3.1 Exact Solution

From the previously described observations it turns out that a local alteration of the additional edges of a single node can generate a global repercussion on the whole prediction graph G. Therefore we are not able to find any problem deconstruction trivial approach, needed in order to develop a dynamic programming exact solution algorithm.

Without problem deconstruction, in order to compute the additional edge choice for a particular node, we have to read a big portion of the trie (from the *observation* 2).

Given our needs of computing the choice on the fly in a text typing environment using huge corpus (and therefore an huge trie), we consider the exact solution approach too much computational time consuming.

Therefore, until a problem decomposition is found, we will drop the exact solution approach: in this work we propose a local heuristical algorithm that try to approximate the global *total keystroke saving* condition.

3.2 Heuristical Naive Solution

We define:

$$depth(i) : node \rightarrow \mathbb{N}^+$$

as the depth in the trie T of the node i.

The more naive heuristical solution to the a node additional p edges choice problem is to chose the p distinct edges pointing to nodes $b_1, b_2, \ldots b_p$ of the a-rooted sub-trie, such that the sum:

$$\sum_{i=1}^{p} (depth(b_i) - depth(a) - 1) \cdot f_{b_i}$$

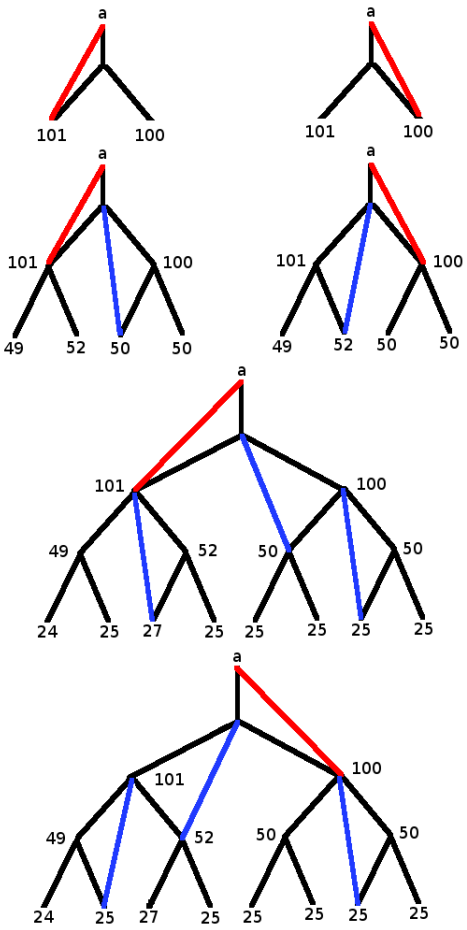

Figure 3: In this figure the additional edges choice with $p = 1$ of node a has the additional constrains to admits only additional edges of length ≤ 2. Even with this constrains we have the necessity of inspecting the sub-trie up to the leaves: in the first two cases we inspect the trie only for 2 depth levels and the better additional edge choice turns out to be a connection to the left node (keystroke savings 101 for the left case, 100 for the right), inspecting up to depth level 3 the better choice turn out to be a connection to the right node (keystroke savings 151 for the left case, 152 for the right), inspecting up to depth level 4 it turns out to be the left again (keystroke savings 203 for the left case, 202 for the right).

is maximized (there is a -1 because we want to express the keystroke saving obtained thanks to the additional edges in respect to an interface with the standard single character keys).

But we have to consider the fact that, even if we remain in the naive condition of not taking into account the additional edges of the nodes of T (with the only exception of node a), the p edges of node a will interact with the same mechanism described in *observation* 1.

Consider for example Figure 4: the more keystroke saving additional edge is the one present only in the left, with a saving of 2, but if we consider that we have to choose another edge, then the optimal choice is the one in the right (total keystroke saving of 2.5 for the left choice, 3 for the right one).

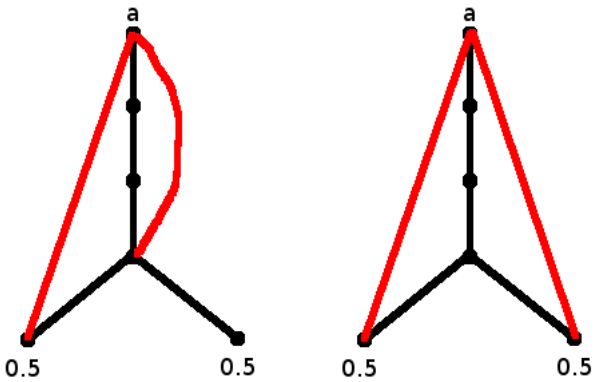

Figure 4: The additional edges choice with $p = 2$ of node a, without considering edges interaction on the left, considering edges interaction on the right.

In order to select the optimal choice we have to take into account quite complex mechanisms, but in this heuristical naive solution we doesn't want to develop a too much complex algorithm. The trade-off we will adopt is the following:

ST = a-rooted sub-trie of T;

additionaledges(ST) {

$ST_1 = ST$;

for $i = 1 \rightarrow p$ {

select b_i in ST_i such that $((depth(b_i) - depth(a) - 1) * (f_{b_i}/f_a))$ is maximized;

$ST_{i+1} = ST_i$ where {

for every nodes c along the path from a to b_i : $f_c = f_c - f_{b_i}$;

the b_i-rooted sub-trie is disconnected from b_i's father and connected to node a;

for every nodes d in the b_i-rooted sub-trie : $depth(d) = depth(a) +$
$(depth(d) - depth(b_i)) + 1$;
}
}
}

3.3 Real corpus

In the section 2 we defined the perfect corpus C_p as the set of all the possible texts of finite length that the user will eventually write, associated to their relative frequencies.

In the real case, expecially for the linguistic domain of natural languages, we doesn't have the perfect corpus, but a real corpus C_r composed by the concatenation of various texts, more or less statistically representative of the linguistic domain.

Adopting a single string C_r of concatenated texts, T is the trie of the suffixes of C_r, with all the frequencies of the suffixes set to $1/|C_r|$.

3.4 Statistical significance: $supMIN$ parameter

We say that a real corpus C_r is statistically representative of a linguistic domain if, assuming a prefix string P, the relative frequencies for all the alphabet characters to follow an occurrences of P in C_r is statistically similar to the same frequencies we can calculate from the perfect corpus C_p (weighing every occurrence in a string of C_p with the string's associated relative frequency).

Intuitively, for the case of a natural language very statistically representative corpus composed by real world texts, the previously described statistical evaluation of the character that will follow a prefix string P will become less similar to the correct linguistic domain statistics when the number of occurrences of P in C_r decrease.

The $supMIN$ parameter denote the minimum number of occurrences of a text prefix we consider acceptable in order to assure a statistical significance.

In the text typing process of path selection in T, every time we are in a node a whose associated string s_a has a frequency of $f_a \leq supMIN/|C_r|$, then we have to reduce the considered prefix s_a, recursively removing the first characters, until the node associated to the reduced prefix has a frequency that can assure the statistical significance.

3.5 Locality of the heuristic: $dMAX$ parameter

In order to develop a fast on-line algorithm, we doesn't want to analyze all the a-rooted sub-trie, but only the first $dMAX$ depth levels of the sub-trie.

The effect on the user side of the $dMAX$ parameter is that the length of the p proposed strings will be limited to $dMAX$ characters. The prediction

performance effect of this parameter will became negligible when the linguistic domain doesn't have many frequent occourring strings of length greater than $dMAX$ (in the case of formal languages probably there will be the need of a bigger $dMAX$ than in the case of natural languages).

3.6 Fast sub-trie search: $pruningFraction$ parameter

Our heuristical method, as previously stated, needs to select a node b in the a-rooted sub-trie that maximize the function:

$$(depth(b) - depth(a) - 1) \cdot (f_b / f_a)$$

Unluckily, descending through a path in the trie, the depth values are strictly increasing but the frequency values are non-increasing, therefore the target function is non-monotonic.

This means that on the general case we will needs to scan all the $dMAX$ levels of the sub-trie in order to find the node that maximize the function, an exponential Σ^{dMAX} nodes in the worst case.

In order to decrease the number of the nodes we need to analize, we propose the $pruningFraction$ parameter: the search for the node that maximize the function will be a depth-first sub-trie traversal, but restricted only to the nodes i with $f_i \geq pruningFraction \cdot f_a$.

This heuristical search method will analyze not more than $dMAX/pruningFraction$ nodes, and, if the node that maximize the function in the sub-trie has a value $\geq (dMAX - 1) \cdot pruningFraction$ then the heuristical search will surely find it.

3.7 Heuristical Look-back Solution

An heuristical algorithm less naive than the one exposed in section 3.2 will take into account the *observation* 1 about the additional edges already chosen along the greedy path from root to node a.

Every time in the path selection process a proposal isn't selected by the user, it's added to the set of discarded suffixes D. When a fragment of text S is appended to the already written prefix of the target text, for every discarded suffixes $d \in D$, if d is prefixed by S and $|d| > |S|$ then it's replaced by its suffix of length $|d| - |S|$, else it is removed from the set of discarded suffixes.

The algorithm is essentially the same alredy described for the naive solution, but the starting a-rooted sub-trie is pruned of all the i-rooted sub-tries such that $s_i \in D$.

3.8 Heuristical Look-ahead Solution

From *observation* 2 we know that a sub-optimal local solution can be globally optimal if we consider the effect of the a node additional edges constrain upon the additional edges choices for all the nodes in the sub-trie.

In order to find the optimal additional edges choice we need to analyze the whole a-rooted sub-trie structure, but in an heuristical solution we can limit the depth of the node we will analyze.

Currently we don't have developed any fast heuristical look-ahead algorithm, therefore all the experiments will be done with the naive and with the look-back solutions only.

4 Compressed Index

5 Experiments

In all the following experiments we will adopt tha same protocol; given the untagged textual corpus C, the prediction algorithm P, and the number of additional keys p:

- C is fragmented in 5 different texts, of size $|C|/5$, referred as $C_1, C_2 \ldots C_5$.
- there will be 5 different prediction experiments: in the i-th experiment the *test corpus* is the fragment C_i and the *training corpus* is the ordered concatenation of the other 4 remaining fragments.
- for every one of the 5 experiments, the typing of the *test corpus* is executed by an automated writer: the writer simulate the use of the p-keys extended interface. After every simulated keystroke, the additional p keys associated strings are reassigned with the strings that P will propose, considering only the *training corpus* and the keystrokes history.
- the numbers of keystrokes needed by the 5 experiments to simulate the typing of the *test corpus* are summed up; the parameter *prod* that we adopt to indicate the performance of P is the ratio $|C|$ over the keystrokes sum.

Intuitively, the performance parameter *prod* indicate the mean number of character producted by each keystroke.

We will refer to the heuristical naive solution algorithm described in section 3.2 as *naive*, and to the heuristical look-ahead solution algorithm described in section 3.8 as *lookAhead*.

5.1 Parameter analysis

Both *lookAhead* and *naive* algorithms use the same parameters:

supMIN : the minimum number of occurrences of a text prefix we consider acceptable in order to assure a statistical significance. If, when computing the additional edges for a node a, $s_a \leq supMIN$, then

(described in section 3.4)

5.2 English natural language

http://pizzachili.di.unipi.it/texts/nlang

http://www.gutenberg.org

This file is the concatenation of English text files selected from etext02 to etext05 collections of Gutenberg Project. We deleted the headers related to the project so as to leave just the real text. Downloaded on May 4, 2005.

5.3 English Wikipedia

http://download.wikimedia.org/enwiki/20080524/enwiki-20080524-pages-articles.xml.bz2

5.4 Italian Wikipedia

lingua inflectional

5.5 Dutch Wikipedia

lingua agglutinante

5.6 Multilingual News

http://www.di.unipi.it/ gulli/newsspace200.xml.bz (cos vediamo nel caso multi-language)

5.7 Corpus tecnico da decidere

4.4) Consideriamo dataset specializzati (uno informatico, uno economico, uno biologico???)

5.8 Corpus Size

4.1) Stimiamo l'impatto della dimensione di Wikipedia indicizzata sulla bont della predizione.

5.9 Additional keys number

4.1) Stimiamo l'impatto della dimensione di Wikipedia indicizzata sulla bont della predizione.

5.10 Comparation with other softwares

spiega perche' e' difficile

decidi protocollo

4.0) Occorre scaricare i dataset che sono indicati nella survey che ci ha passato il Mancarella, per dimostrare che il nostro approccio funziona in diversi contesti. Il dataset di Mancarella non ci porta all'accettazione dell'articolo;-)

L'idea di usare le prestazioni citate in quell'articolo per confrontarle con quelle ottenute dal nostro metodo, a parit di dataset, ovviamente.

6 Conclusions