英文单词拼写错误自动检查系统

包利强，雷雪林

2017 年 6 月 23 日

目录

1 课题简介

在英语写作中，单词的拼写错误是一种较为常见的错误，因此英文单词自动纠错的研究成为了自然语言处理技术的重要子课题。课题11要求我们实现一个英文文本中单词拼写错误自动检查系统，如果在检查的同时还要求进行纠错错，那么这个系统将包括两个基本的部分，即检错（detecting）和纠错（correcting）。在现代自然语言处理的研究中，英文单词拼写纠错系统又可以按照不同的情境分别进行研究，其一是上下文无关单词的检错和改错（*context-free word error detection and correction or isolated-word error detection and correction*），在该情境下，每个单词的检查都是与它的上下文环境相独立的，为了检测该单词是否正确，很自然的一个做法是在词典中搜索该单词，如果搜索成功则拼写正确，否则错误，在这种方法中，算法对词典非常敏感。然而这个方法的一个很大的缺点是无法对那些拼写正确但是不符合上下文的单词进行检错，因此依赖上下文的检错和改错（*context-dependent error detection and correction*）也是一个非常重要的研究课题，但由于这个课题需要语法相关的分析，因此比较复杂[1]。

2 项目目标

目前在自然语言处理研究中，拼写检查和纠错方面的研究已经进行了很长一段时间，有了很多成果和基础，也有了不少开源的拼写检查与纠错的成熟系统，主要以GNU Aspell和Hunspell为代表。我们小组在阅读相关文献和实践的基础上，将动手实现一个简单的类GNU Aspell系统，由于在1中所解释的原因，我们将只实现上个一下文无关的单词检错和改错系统。拼写错误改错意味着要对文本中出现的错误单词提供可能的正确单词建议，一般是一个概率递减的建议列表，但在自动改错时，我们只选择概率最大的一个单词替换错拼单词即可。

3 国内外相关工作

3.0.1 拼写纠错相关数据结构

如1中所说，检测一个单词是否拼写错误看起来非常简单，我们只需要在一个词典中对该单词进行查询即可，但这个方法同时也有很大的问题，首先一个包含所有正确单词的词典文件可能会非常大，查询会造成非常糟糕的空间和时间复杂度；其次，错误的拼写在很多时候会产生可以在词典中查询到的正确单词，这种情况叫做**真词错误**；最后，过大的词典文件会包含很多怪异的单词，这会增加产生真词错误的可能。

20世纪80年代末到90年代初，有研究者发现，对于不同的语言，所需要的适当的词典大小是不同的，对于屈折度不高的语言如英语来说，包含50000-200000单词的词典的效果是较好的[2, 3, 4]。而对于其他一些屈折度较高的语言，所需要词典的大小可能会非常大。

在经典的数据结构中，可以满足词典的快速查询的一个实现是哈希表[5]。但传统的哈希表具有的一个缺点是哈希函数的选择和合适的表规模设置，否则会产生严重的碰撞问题。1997年，Czech等人[6]所提出的最小完美哈希（minimal perfect hashing）解决了碰撞的问题，但同时也要求一个非常大的基本上和词典规模相当的哈希表。

另外一个经常用来存储词典的数据结构是前缀树（Trie）[5]。这是一种有序树，用于保存关联数组，其中的键通常是字符串，与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。前缀树能够有效加速词典的查询，但它的大小也是依赖于词典的大小，为了减小前缀树的规模，很多研究者也做出了自己的努力，如著名的C-trie[7]，PATRICIA[8]，以及Bonsai[9]。

研究者使用了一种无环确定性有限自动机（Acyclic Deterministic Finite Automaton，ADFA）[1]，可以看做是前缀树的扩展。如果一颗完全前缀树的等价子树已经全部合并，那么我们就得到了一颗极小ADFA。

还有一种方法是直接保存每个单词的词根和相应的生成规则，这样我们就知道了怎么生成每个单词的正确形式。这种方法广泛使用在开源的拼写检查器如Ispell[10]和Aspell[11]中。如在Ispell中的一些例子：boolean/S，这意味着这个单词的复数形式booleans也是正确的；frizzle/DGS，则意味着frizzle,frizzled,frizzles,以及frizzling都是正确的。

3.0.2 独立单词拼写纠错

独立单词的纠错意味着要为**非词错误**提供一个或多个正确的单词建议，如果有多个建议，则它们应该按照可能性大小依次排序。如对于一个非词错误*speling*，*spelling*比*spilling*可能性更大，但是也不能完全确定。

一种最简单的方法是基于这样一个假设：人们在打字时通常只会犯几种错误，因此对每一个错误单词，我们需要弄明白使用一个正确单词来产生它的最小的基本操作数（如插入、删除和替换），所需的步数越小，则该正确单词产生它的概率越大。这个方法叫做最小编辑距离（minimal edit distance）方法[12]，Damerau提出后又由Wagner作了进一步研究[13]。1966年，一种建立在插入、删除和移位操作上的相似方法被提出[14]。但是这种方法只能解决三大拼写错误中的一种，即**误敲**，而不能同时解决其他错误。一种直接的实现方法是对词典中的每一个单词都和非词错误进行对比，这种作法比较耗时；一种更快的做法[15]可以使得词典中要进行比较的单词数量减少一个数量级，如果非词中的错误个数较小的话。

在**相似性线索技术（similarity key technique）**中，词典中的每个单词都被赋予一个线索，在和非词的比较中，只有线索才会进行比较，那些和非词的线索非常相似的线索对应的单词会被选为建议单词。由于只有相似的线索才会参与比较，所以这个方法更加有效。这个方法首先是基于SOUNDEX系统[16]提出的，这个系统是用来对按发音转译的名字进行纠错。后来，一个相似的但是更加准确的方法被开发出来，称为SPEEDCOP[17]。这个系统提供了两种线索的实现，即概要（skeleton）和省略（omission）。但这种方法只适应于误敲的情况。

基于规则（Rule-based）的方法适用于更加普遍的错误类型，这种方法考虑的是非词向词典中正确单词的转变。第一个基于知识的系统提出于1983年[18, 19]，所依赖的规则是从1000个拼写错误中得到的。这种方法有两个问题，首先是知识或规则必须通过对真实拼写错误进行实验才能得到，并且必须融合进算法之中，其次是对所获得的单词列表进行验证，判断它们是否属于词典非常困难，这使得整个过程非常耗时。但是这种方法具有的一个很好的优点是它对所有三种拼写错误都能进行处理。

在基于概率的方法（Probabilistic techniques）中，不同的编辑操作产生错误的概率是不同的。对字母替代、字母插入、字母删除和字母移位这些基本操作的的概率研究构成的概率方法的基础。在Church和Gale提出的算法[20]中，这些单词通过分析语料库中的大量的单词获得，建议单词是按照生成非词的概率大小来进行排序的。

语音相似性（phonetic similarity）方法适用于对拼写错误的单词进行纠错。这种方法有一个假设，即是打字者确切地知道所要打的单词的发音，但是不知道正确的拼写方式，这样的人常常使用不正确的字形来表达音素。语音相似性可以有很多种表达方式，如SOUNDEX编码[16]即是一种，其他的如PHONIX[21]，MetaPhone[22]，以及Double MetaPhone[23]。

参考文献

[1] S. Deorowicz and M. G. Ciura, "Correcting spelling errors by modelling their causes," *International journal of applied mathematics and computer science*, vol. 15, no. 2, p. 275, 2005.

[2] F. J. Damerau and E. Mays, "An examination of undetected typing errors," *Information Processing & Management*, vol. 25, no. 6, pp. 659–664, 1989.

[3] F. J. Damerau, "Evaluating computer-generated domain-oriented vocabularies," *Information processing & management*, vol. 26, no. 6, pp. 791–801, 1990.

[4] J. L. Peterson, "A note on undetected typing errors," *Communications of the ACM*, vol. 29, no. 7, pp. 633–637, 1986.

[5] D. E. Knuth, "The art of computer programming, vol. 3: Sorting and searching," 1973," *This book describes the binary search on*, pp. 409–426.

[6] Z. J. Czech, G. Havas, and B. S. Majewski, "Perfect hashing," *Theoretical Computer Science*, vol. 182, no. 1-2, pp. 1–143, 1997.

[7] K. Maly, "Compressed tries," *Communications of the ACM*, vol. 19, no. 7, pp. 409–415, 1976.

[8] D. R. Morrison, "Patricia—practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

[9] J. J. Darragh, J. G. Cleary, and I. H. Witten, "Bonsai: a compact representation of trees," *Software: Practice and Experience*, vol. 23, no. 3, pp. 277–291, 1993.

[10] R. Gorin, P. Willisson, W. Buehring, and G. Kuenning, "2003. international ispell," 1971.

[11] K. Atkinson, "Gnu aspell. 2003," *URL http://aspell. net*, 2011.

[12] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[13] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[14] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[15] R. Baeza-Yates and G. Navarro, "Fast approximate string matching in a dictionary," in *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pp. 14–22, IEEE, 1998.

[16] M. Odell and R. Russell, "The soundex coding system," *US Patents*, vol. 1261167, 1918.

[17] J. J. Pollock and A. Zamora, "System design for detection and correction of spelling errors in scientific and scholarly text," *Journal of the American Society for Information Science (pre-1986)*, vol. 35, no. 2, p. 104, 1984.

[18] E. J. Yannakoudakis and D. Fawthrop, “An intelligent spelling error corrector,” *Information Processing & Management*, vol. 19, no. 2, pp. 101–108, 1983.

[19] E. J. Yannakoudakis and D. Fawthrop, “The rules of spelling errors,” *Information Processing & Management*, vol. 19, no. 2, pp. 87–99, 1983.

[20] K. W. Church and W. A. Gale, “Probability scoring for spelling correction,” *Statistics and Computing*, vol. 1, no. 2, pp. 93–103, 1991.

[21] T. Gadd, “Phonix: The algorithm,” *Program*, vol. 24, no. 4, pp. 363–366, 1990.

[22] L. Philips, “Hanging on the metaphone,” *Computer Language*, vol. 7, no. 12 (December), 1990.

[23] L. Philips, “The double metaphone search algorithm,” *C/C++ users journal*, vol. 18, no. 6, pp. 38–43, 2000.