# CS 276 Programming Assignment 1: K-Gram Spelling Correction and Lucene

**Todd Sullivan**
todd.sullivan@cs.stanford.edu

**Ashutosh Kulkarni**
ashuvk@stanford.edu

## 1 Introduction

In this project we developed multiple spelling correctors using the Jaccard score of k-gram overlap, Levenshtein edit distance, and word frequency for ranking and tie breaking between possible spelling corrections. After a thorough analysis of our spelling corrector's tuning parameters, we determined the best overall spelling corrector to use k-grams with 2 characters and to include the first character and last character of the word as special k-grams. A layered ranking algorithm that first generates a list of words based on Jaccard k-gram overlap, and then re-ranks the list based on Levenshtein edit distance worked best.

We also explored using Lucene for search and experimented with using Lucene's built-in spelling corrector as well as incorporating our best spelling corrector.

## 2 Spelling Correction

### 2.1 General Implementation

All of our spelling correctors can be configured by specifying four different parameters: K, SE, TB, and CS. These parameters are summarized in Table 2.1. SE is perhaps easiest described with an example. Consider the word "cow". If K = 3 and SE = 3 then the only k-gram is "cow". If SE = 2, then there are two additional k-grams of "$co" and "ow$", where $ is a special boundary symbol. If SE = 1 then there are two more additional k-grams of "$$c" and "w$$". These parameters allow us to easily create and test a range of spelling correctors.

Our spelling correctors contain a few more niceties aside from the power and convenience obtained through these four parameters. Through the indexFile() method one can incrementally build up the inverted index. Our implementation is also fairly fast. Most of our spelling correctors can spell correct the entire test set in one to three seconds with only around three to five seconds for building the provided "big.txt.gz" file's index.

Table 2.1: Spelling Correction Parameters

| Parameter | Description |
|---|---|
| K | K-gram size to use. |
| SE | Smallest extra k-gram to use from either end of the word. |
| TB | Whether or not to use a tie breaking method such as word frequencies. |
| CS | Size of returned corrections list. |

### 2.2 K-Gram Spelling Correction without Tie Breaking

Table 2.2 displays the best scoring parameters for our most basic spelling corrector, which only uses the Jaccard score of k-gram overlap with no tie breaking mechanism. Since the spelling corrector only uses k-gram overlap, the CS parameter has no effect on the score. This is because the top scoring spelling correction will always be at the front of the list.

To determine the best parameters, we varied K from two to ten and varied SE from one to K for each K. An interesting note is that all spelling correctors with SE = 1 scored better than spelling correctors with SE = 2, which scored better than spelling correctors with SE = 3, and so on. This suggests that the k-grams that can be formed with at least one $ marker

(from the front or back of the word) are consistently more important than the other k-grams and that the most benefit is realized from using all of these k-grams with $ markers.

Table 2.2: Best KGramSpellingCorrector Parameters with TB = false, CS irrelevant

| K | SE | Spellings Correct (out of 270) | Time (ms) |
|---|---|---|---|
| 2 | 1 | 188 | 1,766 |
| 4 | 1 | 187 | 727 |
| 3 | 1 | 187 | 786 |
| 5 | 1 | 186 | 785 |
| 6 | 1 | 185 | 788 |

### 2.2.1 Sample Performance Cases

Table 2.2.1 contains output from our KGramSpellingCorrector with K = 2, SE = 1, TB = false, and CS = 10, showing three interesting cases encountered in the test set. In the first case, the word "hierchy" is successfully changed to "hierarchy" mostly because the dataset used to create the k-gram index did not contain any word close to "hierarchy" except for "hierarchy" itself.

In the second case, "concider" is incorrectly changed to "cider" when the answer is clearly "consider". This case would easily be solved by applying Levenshtein edit distance to the top 10 list since "cider" is very far away from "concider" while "consider" involves swapping only one letter. The problem could also possibly be solved by blending the word frequency with the k-gram overlap score since "consider" has a much larger word frequency in comparison to "cider" and their k-gram overlap scores are relatively close.

In the third case, our spelling corrector incorrectly chose "files" instead of "fails". In this case both "files" and "fails" have the same k-gram overlap score but they have different word frequencies. The correct answer could be obtained in this case through a tie-breaking mechanism that chooses the word with the largest word frequency in the text that was indexed.

Table 2.2.1: KGramSpellingCorrector Output K = 2, SE = 1, TB = false, CS = 10

| hierchy => hierarchy | | |
|---|---|---|
| Correction | Jaccard Score | Word Frequency |
| hierarchy | 0.800 | 4 |
| kerchief | 0.417 | 11 |
| handkerchief | 0.400 | 56 |
| kerchiefs | 0.385 | 1 |
| handkerchiefs | 0.375 | 6 |
| archie | 0.364 | 2 |
| thierry | 0.333 | 3 |
| chiefly | 0.333 | 134 |
| anarchy | 0.333 | 7 |
| archery | 0.333 | 1 |

| concider => consider | | |
|---|---|---|
| Correction | Jaccard Score | Word Frequency |
| cider | 0.667 | 1 |
| consider | 0.636 | 98 |
| coincide | 0.5 | 5 |
| considers | 0.462 | 10 |
| coincided | 0.462 | 4 |
| coincides | 0.462 | 4 |
| colder | 0.455 | 2 |
| confer | 0.455 | 14 |
| cinder | 0.455 | 1 |
| concur | 0.455 | 3 |

| failes => fails | | |
|---|---|---|
| Correction | Jaccard Score | Word Frequency |
| files | 0.625 | 8 |
| fails | 0.625 | 20 |
| failures | 0.6 | 5 |
| failed | 0.556 | 63 |
| faites | 0.556 | 1 |
| faithless | 0.546 | 1 |
| fail | 0.5 | 40 |
| fairies | 0.5 | 1 |
| tailless | 0.455 | 1 |
| fairness | 0.455 | 4 |

## 2.3 Incorporating Levenshtein Edit Distance

One of the areas for improvement identified in the output of our first spelling corrector is the use of Levenshtein edit distance to rescore the original k-gram spelling corrector's output. To do this, we simply create a wrapping class that first retrieves the top CS candidates from our first spelling corrector and then reorders the list based on each word's Levenshtein edit distance with the word we are considering spelling corrections for.

### 2.3.1 Best Parameters

Table 2.3.1 shows the top 5 parameter sets for this new spelling corrector. We evaluated the spelling corrector with the same range of K and SE values as the previous corrector. Since this new spelling corrector involves rescoring all words in the list, the size of the list has an effect on performance. This is because a smaller list may not include a word that has, say, an edit distance of one while expanding the list could reveal such a word. Thus we also varied the CS parameter from 5 to 60 in increments of 5. In the table, we are allowing only the best scoring k/SE combination. For example, K = 2 and SE = 1 also gets a score of 200 when CS = 10, but we omit this since the k/SE combination has higher score elsewhere.

We see that K = 2 and K = 3 perform well again, as well as SE = 1 and SE = 2. When moving to SE = 2, the best size of the list (CS) is larger. One upside of moving to SE = 2 is that less k-grams are used. This is reflected in the reduction in processing time from K = 3 SE = 1 to K = 3 SE = 2 even as CS increases. A look back at our previous speller's performance shows that for the 2/1 k/SE combination, incorporating edit distance receives a boost in score of 13 while increasing processing time by about one second.

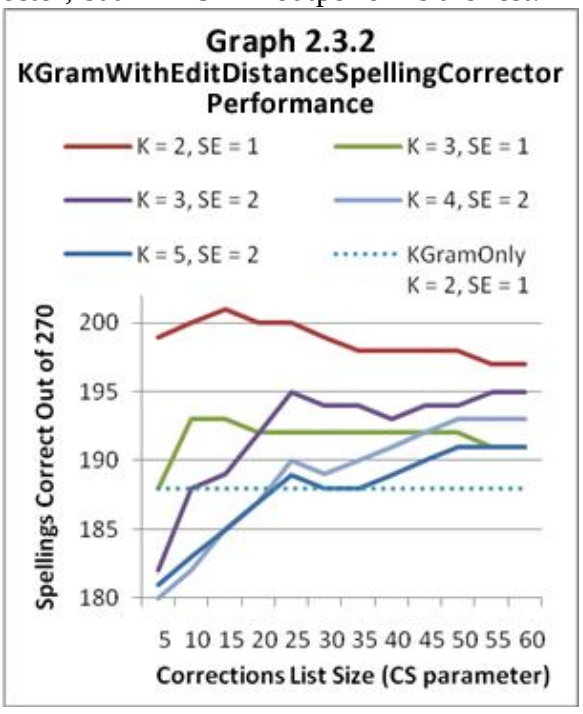
An important result that is not reflected in Table 2.3.1 is that all 2/1 k/SE combinations (for all values of CS) scored higher than any other k/SE/CS combination. The worst performing 2/1 combination was CS = 55 and CS = 60 with 197 spellings correct. Clearly, 2/1 dominates when edit distance is incorporated.

Table 2.3.1: Best KGramWithEditDistance-SpellingCorrector Parameters with TB = false

| K | SE | CS | Spellings Correct (out of 270) | Time (ms) |
|---|---|---|---|---|
| 2 | 1 | 15 | 201 | 2,664 |
| 3 | 2 | 25 | 195 | 470 |
| 4 | 2 | 50 | 193 | 727 |
| 3 | 1 | 10 | 193 | 1,109 |
| 5 | 2 | 50 | 191 | 777 |

### 2.3.2 Corrections List Size Effect on Performance

Graph 2.32 shows the edit distance corrector's performance with the parameters from Table 2.3.1 as CS is varied. The graph also includes the highest score (188) from the original corrector. The two correctors with SE = 1 exhibit a trend of peaking early and then dropping off as CS increases. The three correctors with SE = 2 follow a general increase in performance as CS increases, but they take a temporary dip somewhere in the middle and flatten out near the high end of the CS range. The graph clearly shows that all k/SE pairs are capable of outperforming the best original corrector, but K=2 SE=1 outperforms the rest.



### 2.3.3 Sample Spelling Corrections

A glance at the output in Table 2.3.3 for the best edit-distance-based corrector shows that the edit distance technique indeed corrects the previous error with the word "concider". Unfortunately, the edit distance technique does not solve the "failes" problem. As the output shows, four words are tied for first place. If we were to break ties by word frequency or somehow merge word frequency with the edit distance score, we would still not choose the correct answer because "failed" has a vastly higher word count than the other first place words.

Interestingly, this case would be solved with our original corrector using a word frequency tie breaker but it cannot be solved using a word frequency tie breaker in conjunction with our edit distance corrector. One possible solution would be to first perform tie breaks based on the original k-gram overlap scores, and then falling back to word frequency if necessary. This multi-tiered tie breaking system would solve this case because it would simply revert back to the original corrector's case of "files" and "fails" being tied for best overlap score but "fails" ultimately winning because of word frequency. In order to consistently be able to solve these types of cases, one would need some context in which the word was used because the two words in contention are the same word in different tenses.

### 2.3.4 Side-By-Side Comparison

A side-by-side comparison of the best original corrector and the best edit distance corrector (both with K = 2 and SE = 1) shows that while the edit distance corrector corrects many of the original corrector's issues, it is not without fault. Of the words that the original corrector failed to correct, the edit distance corrector successfully solved 30 of them. At the same time, the incorporation of edit distance causes the edit distance corrector to miss 17 words that the original corrector successfully solved.

Table 2.2.3: KGramWithEditDistance-SpellingCorrector Output
K = 2, SE = 1, TB = false, CS = 15

| concider => consider | | | |
|---|---|---|---|
| Correction | Edit Distance | Jaccard Score | Word Frequency |
| consider | 1 | 0.636 | 98 |
| considers | 2 | 0.462 | 10 |
| coincided | 2 | 0.462 | 4 |
| coincides | 2 | 0.462 | 4 |
| coincide | 2 | 0.500 | 5 |
| colder | 3 | 0.455 | 2 |
| confer | 3 | 0.455 | 14 |
| cider | 3 | 0.667 | 1 |
| considered | 3 | 0.429 | 164 |
| cinder | 3 | 0.455 | 1 |
| coincident | 3 | 0.429 | 4 |
| concern | 3 | 0.417 | 32 |
| concur | 3 | 0.455 | 3 |
| reconsider | 3 | 0.429 | 3 |
| confidence | 4 | 0.429 | 53 |

| failes => fails | | | |
|---|---|---|---|
| Correction | Edit Distance | Jaccard Score | Word Frequency |
| failed | 1 | 0.556 | 63 |
| files | 1 | 0.625 | 8 |
| faites | 1 | 0.556 | 1 |
| fails | 1 | 0.625 | 20 |
| piles | 2 | 0.444 | 6 |
| miles | 2 | 0.444 | 110 |
| faces | 2 | 0.444 | 162 |
| families | 2 | 0.455 | 45 |
| fades | 2 | 0.444 | 2 |
| failures | 2 | 0.600 | 5 |
| fail | 2 | 0.500 | 40 |
| fairies | 2 | 0.500 | 1 |
| tailless | 3 | 0.455 | 1 |
| fairness | 3 | 0.455 | 4 |
| faithless | 3 | 0.545 | 1 |

## 2.4 Word Frequency and Tie Breaking

We have hinted at the possibility of using word frequencies in a tie breaking mechanism to solve many of our recurrent errors. As a reminder, the TB parameter in our correctors turns tie breaking on or off. In our original corrector, the tie breaking mechanism is choosing the word with highest word frequency and arbitrarily picking the word closest to the front of the list if the words have the same word frequency. In our edit distance corrector, the tie breaking function first chooses the word or words with highest Jaccard k-gram overlap. If there is more than one such word then it picks the word with highest word frequency just as in the original spelling corrector.

We integrated word frequencies and tie breaking into both correctors because both correctors exhibit cases where tie breaking would help and word frequencies could be used to differentiate words based on importance. We expected the tie breaking functionality to have the most positive impact on the edit distance spelling corrector because Levenshtein edit distance often returns a distance of only 1, 2, or 3 for all words in the corrections list. Thus collisions for the top spot are more likely, so a tie breaking mechanism is more likely to have a large impact.

Table 2.4a shows the top five k/SE pairs for the original spelling corrector with our word frequency tie breaking turned on. The only notable difference between these results and the TB = false results is that K = 2 is not the best K value for the first time. Our tie breaking function resulted in an average of 4 additional words being spelled correctly. A side-by-side analysis of the 3/1 k/SE pair with TB = true and TB = false reveals that turning on tie breaking successfully corrected 6 additional words and did not cause the TB = true corrector to miss any words that the TB = false corrector successfully corrected.

Table 2.4a: Best KGramSpellingCorrector Parameters with TB = true, CS irrelevant

| K | SE | Spellings Correct (out of 270) | Time (ms) |
|---|---|---|---|
| 3 | 1 | 193 | 1,166 |
| 2 | 1 | 191 | 2,493 |
| 4 | 1 | 190 | 1,016 |
| 5 | 1 | 190 | 1,074 |
| 6 | 1 | 189 | 1,088 |

Table 2.4b shows the top five k/SE pairs for the edit distance spelling corrector with our previously mentioned tie breaking scheme. Again, K = 3 comes out on top. A few noticeable differences between TB = true and TB = false edit distance correctors is that the TB = true corrector's best SE values are mostly 1 and its best CS values are relatively low (mostly 10). While the CS values are mostly low, this does not result in a shorter processing time in comparison to the TB = false edit distance corrector because the multi-tiered tie breaking takes additional time. As hypothesized, the tie breaking mechanism has a much larger impact on the edit distance corrector with an average of 7.8 additional words being spelled correctly for each of the top five correctors.

Table 2.4b: Best KGramWithEditDistance-SpellingCorrector Parameters with TB = true

| K | SE | CS | Spellings Correct (out of 270) | Time (ms) |
|---|---|---|---|---|
| 3 | 1 | 10 | 206 | 1,475 |
| 2 | 1 | 10 | 203 | 2,532 |
| 4 | 1 | 10 | 202 | 979 |
| 3 | 2 | 25 | 201 | 1,548 |
| 5 | 1 | 55 | 200 | 1,817 |

We also tried using a scoring function that was a linear combination of the word frequency and Jaccard k-gram overlap score, but this did not produce any noteworthy correctors. We believe that a linear combination did not produce any good correctors because in general the Jaccard k-gram overlap score is a bet-

ter metric than word frequency, and as such word frequency should only be used in cases where tied first place words need to be ranked instead of being used all of the time.

## 2.5 Partial Credit Analysis and Summary

The scoring system used so far to evaluate our spelling correctors gives the corrector one point if the first word in the corrections list is the correct word and gives no point otherwise. This is the perfect evaluation method if one plans to use only the top returned word, but it is inadequate for evaluating the quality of the entire returned list in general. In this section we discuss spelling corrector performance when partial credit is given. In our partial credit system, 1 point is given if the first word in the corrections list is the correct word, 0.8 points if the second word is correct, 0.6 if the third, 0.4 if the fourth, 0.2 if the fifth, and 0 otherwise.

With partial credit, Table 2.2 remains unchanged except that K = 3 and K = 4 are differentiated with K = 3 having a partial score of 214.6 and K = 4 having a partial score of 210.8. Similarly, the only change to Table 2.3.1 in terms of order is the differentiation between K = 3 SE = 1 and K = 4 SE = 2 with the K = 3 corrector receiving 221.4 points and K = 4 receiving 218.6. In the tie breaking realm of Table 2.4a, the only difference is K = 2 claiming the lead with 220.2 for K = 2 and 215.8 for K = 3. In Table 2.4b, the order remains the same but K = 3 only wins by 0.4 points.

Due to the K = 2 SE = 1 corrector's favorable performance on the partial credit evaluation as well as its top scores in both non-tie breaking correctors and its second place scores in the tie breaking correctors, we believe that K = 2 SE = 1 is the best performing k/SE pair.

# 3 Lucene

## 3.1 General Implementation and Classes

The implementation of the Lucene experimentation part is mainly divided into 4 classes:

- **IMDBParser**: Already given with the assignment and we do not make any changes to the original code base.
- **IMDBIndexer**: Builds the index using the given movie database. The index is stored in a folder named cs276-index at the path defined by 'indexPath' variable.
- **IMBDSearcher**: Searches the given queries from part 2.1 and 2.2 from the assignment handout. Contains 'main' method and can be run stand-alone to get the output.
- **IMDBSearcherWithSpellCheck**: Integrates the best possible spell-checker code implemented in the first part with Lucene. It supports multiple queries with the field names and possible special characters. For example, a query such as 'PLOTS:murdered AUTHORS:George' is supported.

The spell-checker suggests the top 10 suggestions for all the words used in the query but picks up the first one from each class to continue. This is just to simplify the code base so that multiple permutations of the suggestions can be avoided.

The class also has a simple example implementation using inbuilt Lucene spellchecker, but the code has been commented out for the purposes of supporting multiple word queries. We also used Luke to play around with the multiple query structures that Lucene can support.

## 3.2 Analysis of Queries and Outputs

Lucene supports a rich query language that provides the ability to create your own queries with special clauses. The following sections show the example query output and comments.

### 3.2.1 Items that author called "Rob" has posted

Table 3.2.1 shows the Lucene query and output for the movies that have an author of Rob. The output makes sense, since the documents are sorted using the (scores, docID). The first four documents have only 'Rob' in their authors field and thus get the highest score (1.0) from Lucene. The fifth and sixth document have the same score (0.7071) since they contain the word 'Rob' twice in the authors field. The other documents have the same score (0.625) and are sorted based on docID.

### 3.2.2 Within K Words

The given query was "Name of movie for which the plotline has the words 'murdered' and 'eighteen' within 5 words of each other." Our original query of 'PLOTS:"murdered eighteen"~5' returned no results. Interestingly, swapping the order of murdered and eighteen in the query produced the output in Table 3.2.2. Using the original order but with the number 6 instead of 5 also returned the result in Table 3.2.2.

There is only one document that contains the words 'murdered' and 'eighteen' within 5 words of each other in the plot. The plot is given as – "As an eighteen year old, Tom's father was murdered. …" If the distance is increased to 10, two more documents are added to the result set: Sommarmord (1994), Ye ban qiang sheng (1932) .

One interesting issue that we discovered with Lucene was if you change the order of the words – e.g. instead of "murdered eighteen", we use "eighteen murdered", it gives slightly different results!

For example, the result set of 'murdered eighteen'~5 was empty but 'eighteen murdered'~5 actually retrieved one document. (tested on Luke 0.8).

Table 3.2.1: Query Output

| Query |
|---|
| AUTHORS:Rob |

| Output | |
|---|---|
| Title | Authors |
| "Unser Walter" (1974) | Rob |
| 1001 Arabian Knots (1994) (V) | rob |
| Doragon booru Z 6: Gekitotsu! Hyakuoku pawâ no senshi (1992) | Rob |
| Soeurs (1992) | Rob |
| Indictment: The McMartin Trial (1995) (TV) | Rob Hartill Rob <robert@bb.com.au> |
| Wednesday (2007) | Rob Sorrenti Rob Sorrenti |
| "Adventures of McGee and Me, The" (1986) | Rob Loos |
| "Crocodile Hunter" (1996) | Rob Hartill |
| "Danger Rangers" (2005) | Rob Pottorf |
| "Dr. Phil" (2002) {Nasty Neighbors (#6.8)} | Rob Whitehurst |
| "Goodnight Sweetheart" (1993) | Rob Hartill |
| "Hi-De-Hi!" (1980) | Rob Hartill |
| "Oh, Doctor Beeching!" (1995) | Rob Hartill |
| "One Foot in the Grave" (1990) | Rob Hartill |
| "Only Fools and Horses" (1981) | Rob Hartill |
| "Open All Hours" (1976) | Rob Hartill |
| "Place in the Sun, A" | Rob Hartill |
| "Pole to Pole" (1992) | Rob Hartill |
| "Porridge" (1974) | Rob Hartill |
| "Rx for Survival: A Global Health Challenge" (2005) {How Safe Are We? (#1.6)} | Rob Whittlesey |

Table 3.2.2: Query Output

| Query |
|---|
| PLOTS:"murdered eighteen"~6<br>Or<br>PLOTS:" eighteen murdered"~6 |

| Output | |
|---|---|
| Title | Plot Excerpt |
| Guilt Complex (2004) | As an eighteen year old, Tom's father was murdered. |

### 3.2.3 Documents for which the movie title is "10 items or less(2006)"

The output set is as expected. The documents are arranged in the (scores, docID) order.

Table 3.2.1: Query Output

| Query |
|---|
| TITLE:"10 items or less(2006)" |

| Output |
|---|
| Title |
| 10 Items or Less (2006) |
| "10 Items or Less" (2006) {Health Insurance (#1.3)} |
| "10 Items or Less" (2006) {The New Boss (#1.1)} |
| "10 Items or Less" (2006) {What Women Want (#1.4)} |

### 3.2.4 Keyword-weighted queries: X='Hart' and Y='Rob'

The boosting operator '^' boosts up the score of word 'Rob'. The score that is calculated for word 'Rob' by Lucene is multiplied by the factor 4 to get the final score of the document. Thus, the documents containing the word Rob will have preference but documents not containing the word Rob will still be returned. The + sign suggests that the word 'Hart' is mandatory and the document must contain it to be in the result set.

Even though, we boosted the score of term 'rob', there are no documents for which the words 'Rob' and 'Hart' go together. So, the only effect of boosting word 'Rob' here is to reduce the scores of the documents retrieved in general.

Since none of the author values contain "Rob", this example is not too interesting. If you change the query to "AUTHORS:(Rachel^4 +Hart)", then all of the results stay in the same order except the two containing "Rachel" are moved to the top two results.

Table 3.2.4: Query Output

| Query |
|---|
| AUTHORS:(Rob^4 +Hart) |

| Output | |
|---|---|
| Title | Authors |
| "Secret World of Og, The" (2006) | Geoff Hart Geoff Hart |
| 40,000 Years of Dreaming (1997) | Simon Hart |
| Glenn Miller Band Reunion, The (1989) (TV) | Perry Hart |
| Picture This: The Times of Peter Bogdanovich in Archer City, Texas (1991) | Doug Hart |
| White Shamans and Plastic Medicine Men (1996) | Daniel Hart |
| "Great Decisions" (1986) | Rachel Hart Connolly |
| Riding in Stride (2006) (TV) | Rachel Hart Connolly |
| Story of a Mother, The (2008) | Hart, James David |

### 3.2.5 Keyword-weighted queries: X='Pereyra' and Y='Rob'

After looking through the documents, we realized that there are cases where an author 'Anthony Pereyra' had shared plots with 'Rob Hartill', and in other cases, he wrote the plots by himself. We used this fact to study the effect of the boosting operator.

Though it does not support the information need that we want to search for the author whose last name is – 'Pereyra' and first name is something like 'Rob', the query does help us understand the effect of the boosting operator.

The expected output was we should see the documents where Anthony Pereyra has shared his views with Rob Hartill before the documents with views only from Anthony Pereyra. The following output shows that this was really the case.

The effect of the boosting operator can be understood for a query such as:

```
AUTHORS:"+Pereyra"    AUTHORS:anthony
PLOTS:Patagonia^4
```

The word Patagonia is present in the plot written by Rolo Pereyra. Without the boosting operator, the first document retrieved contains words – Anthony and Pereyra but not Patagonia. After the boosting is applied, the first document retrieved does contain the word Patagonia even though the document is not written by Anthony Pereyra.

Table 3.2.5: Query Output

| Query |
|---|
| AUTHORS:( Rob^4 +Pereyra) |

| Output | |
|---|---|
| Title | Authors |
| Mask, The (1994) | Anthony Pereyra <hypersonic91@yahoo.com> Ian Pugh <skypilot@ezaccess.net> Qrrbirbel Rob Hartill Chris Makrozahopoulos <makzax@hotmail.com> Robert Lynch <docrlynch@yahoo.com> |
| Species (1995) | Claudio Carvalho, Rio de Janeiro, Brazil Alexander Lum <aj_lum@postoffice.utas.edu.au> Rob Hartill Anthony Pereyra {hypersonic91@yahoo.com} |
| Oro nazi en Argentina (2004) | Pereyra, Rolo |
| "Adventures of Jimmy Neutron: Boy Genius, The" (2002) {Party at Neutron's/Ultra Sheen (#1.17)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| "Donkey Kong Country" (1997) {The Legend of the Crystal Coconut (#1.17)} | Anthony Pereyra {hypersonic91@yahoo.com} |
| "Drake & Josh" (2004) {Driver's License (#2.9)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| "Drake & Josh" (2004) {Mean Teacher (#2.11)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| "Drake & Josh" (2004) {The Bet (#2.1)} | Anthony Pereyra <hypersonic91@yahoo.com> Anonymous |
| "Drake & Josh" (2004) {Two Idiots and a Baby (#1.4)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| "Fairly OddParents, The" (2001) {The Big Problem!/Power Mad! (#1.1)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| "Malcolm in the Middle" (2000) {Lois's Birthday (#2.3)} | Anthony Pereyra <hypersonic91@yahoo.com> |
| 10,000 BC (2008) | Anthony Pereyra {hypersonic91@yahoo.com} |
| 1990: I guerrieri del Bronx (1982) | Anthony Pereyra {hypersonic91@yahoo.com} |
| 2012: The War for Souls (2010) | Anthony Pereyra {hypersonic91@yahoo.com} |
| Adventures of Elmo in Grouchland, The (1999) | Anonymous Anthony Pereyra {hypersonic91@yahoo.com} |
| Agent Cody Banks 2: Destination London (2004) | Anthony Pereyra {hypersonic91@yahoo.com} austin4577@aol.com |
| Aika (1997) (V) | Anthony Pereyra {hypersonic91@yahoo.com} Anonymous |
| All Dogs Go to Heaven 2 (1996) | Anthony Pereyra {hypersonic91@yahoo.com} Anonymous |
| Animal Crossing (2001) (VG) | axemblue Anthony Pereyra <hypersonic91@yahoo.com> |
| Ant Bully, The (2006) | Anthony Pereyra <hypersonic91@yahoo.com> movieguy3 |

## 3.3 Spelling Correction with Lucene

### 3.3.1 Support from Lucene

Lucene supports a couple of ways by which you can check and correct spelling mistakes.

**Inbuilt Spell-check class**: Lucene has a spell-check library of its own which implements n-gram spell checker method and the Levenshtein distance approach.

**Fuzzy words approach**: Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. To do a fuzzy search we can use the tilde, "~", symbol at the end of a single word term. For example to search for a term similar in spelling to "rob" use the fuzzy search: "rob~". This search will find terms like mob, rob, job etc. Thus, a normal search on ‘Trmmy’ does not give any results in the index constructed for movie database but a fuzzy search using ‘Trmmy~’ gives documents containing word ‘Timmy’.

### 3.3.2 Spelling Correction Using Our Spell Check Mechanism

We used our best spelling corrector developed in Section 2, the KGramWithEditDistanceSpellChecker with K = 2, SE = 1, TB = true, and CS = 10, to integrate with the Lucene index to get the spell correction mechanism working. The analyses of the spell checker is discussed in detail in section 2.2 and 2.3. Here we discuss the cases where the spell checker performs / fails to perform well.

For a single word query like ‘eob’, the suggestions returned by the spell check are: `[mob, sob, job, rob, erb, bob, ebb, cob, knob, jacob]` and the output for the query was a single movie called "Zombie(zero) (2001)" with an author of "Mob Boss".

Given that QWERTY style keyboards are the most popular keyboard used, we should be

able to conclude that the distance between ‘eob’ and ‘rob’ is intuitively less than with other words. Our spell-checker which implements the standard Levenshtein Distance algorithm does not take this into account and fails to perform well in such cases.

For words with longer lengths though, the basic k-gram approach helps in finding out the nearest correct word. For example, for words like "murderwd", it returns a good set of suggestions: [murdered, murder, murders, murderer, murderous, murderers, murdering, murmured, mud, murat] and produces correct output for the query as shown in Table 3.3.2.

Though our algorithm uses the similar concept of k-gram with Levenshtein Distance used in Lucene’s inbuilt spell-correction mechanism, we lack the support of field related spell-corrections. Lucene does support field related corrections and thus greatly improves the accuracy in the name fields wherein standard dictionary words seldom occur.

### 3.3.3 Summary and Possible Improvements for our Spelling Corrector

One improvement to our overlaid spelling corrector on top of Lucene would be to use a separate spelling corrector for each field index. In its current form, our spelling corrector indexes the "big.txt.gz" file provided for the project. It does not index any of the words in the IMDB data. Our spelling corrector would have much better performance if we actually used four spelling correctors: one for each of the field indexes (AUTHORS, TITLE, PLOTS) and one that indexes all three fields' words. With these four correctors, we would use the appropriate corrector when a field index is specified. For all query terms that do not contain a field index, we would use the corrector that indexed all three fields.

Another improvement would be to incorporate the standard keyboard layout as briefly mentioned in the "eof" example.

Table 3.3.2: Query Output

| Query |
|---|
| PLOTS:murderwd<br>(corrected to)<br>PLOTS:murdered |

| Output | |
|---|---|
| Title | Authors |
| "Midsomer Murders" (1997) {Death in Chorus (#9.7)} | the_mystery_man |
| Champagne Charlie (1936) | Ed Stephan <stephan@cc.wwu.edu> |
| A futura memoria: Pier Paolo Pasolini (1986) | Giancarlo Cairella <vertigo@imdb.com> |
| Courtyard, The (1995) (TV) | Anonymous |
| iMurders (2008) | JuggyGales |
| Jigsaw (1972) (TV) | frankfob2@yahoo.com |
| MacGyver: Trail to Doomsday (1994) (TV) | |
| "Baantjer" (1995) {De Cock en de moord op de moordenaar (#1.4)} | Dutch90 |
| "Moonlighting" (1985) {The Next Murder You Hear (#1.4)} | Cassandra Sumerro |
| "Nämndemans död, En" (1995) | Mattias Thuresson <mattias.thuresson@mbox300.swipnet.se> |
| "Roy Rogers Show, The" (1951) {The Treasure of Howling Dog Canyon (#1.4)} | frankfob2@yahoo.com |
| "Witse" (2004) {De bonami (#1.2)} | Rune Thandy |
| "Women's Murder Club" (2007) {The Past Comes Back to Haunt You (#1.7)} | Ron Kerrigan <mvg@whidbey.com> |
| Conspiracy of Silence (1991) (TV) | <Blythe379@cs.com> |
| Grave Situations (2007) | Anonymous |
| Lion Man, The (1936) | frankfob2@yahoo.com |
| Man from Sundown, The (1939) | Les Adams <longhorn3708@windstream.net> |
| Phantom of the West, The (1931) | Jim Beaver <jumblejim@prodigy.net> |
| Sins of the Past (1984) (TV) | frankfob2@yahoo.com |