Data Analysis in R

By
Dr. Hussein Hazimeh

Lebanese University
Faculty of Information 1

Data Science Departement
2rd year – Data Analysis in R
Spring – 2022 – Chapter 10

Agenda

- Dynamic programming
- Edit distance
- Applications

1

Dynamic Programming

Introduction

Dynamic Programming

1. General Design and Problem Solving Strategies
2. More about Dynamic Programming
   - Example: Edit Distance

Design Strategies

- Dynamic Programming Design Strategy
  - Solve an “easy” sub-problem
  - Store the solution
  - Use stored solution to solve a more difficult sub-problem.
  - Repeat until you solve the “big” hard problem

Overview

- What is edit distance? Why do we care?
- A few interesting properties
- Some algorithms from the past for finding exact edit distance
- More (complicated) algorithms from today for approximate edit distance
- Embedding edit distance into l1

What is that?

Suppose we have two strings x,y

e.g. x = kitten

y = sitting

and we want to transform x into y.

We use *edit operations*:

1. insertions
2. deletions
3. substitutions

2

Edit Distance

Introduction

What is that?

A closer look:

k i t t e n

s i t t i n g

1st step: kitten →sitten (substitution)

2nd step: sitten→sittin (substitution)

3rd step: sittin→sitting (insertion)

Minimum Edit Distance

- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N

d s s   i s
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

What is that?

Can we do better?

Answer here is no (obviously)

What about:

x = darladidirladada

y = marmelladara

Tough…

Why do we care?

A lot of applications depend on the similarity of two strings

- Computational Biology:

  …ATGCATACGATCGATT…
  …TGCAATGGCTTAGCTA…

  Animal species from the same family are bound to have more similar DNAs

  What about evolutionary biology?

Why do we care?

- searching keywords through the net: usually by “mtallica” we mean “metallica”:

Other uses of Edit Distance in NLP

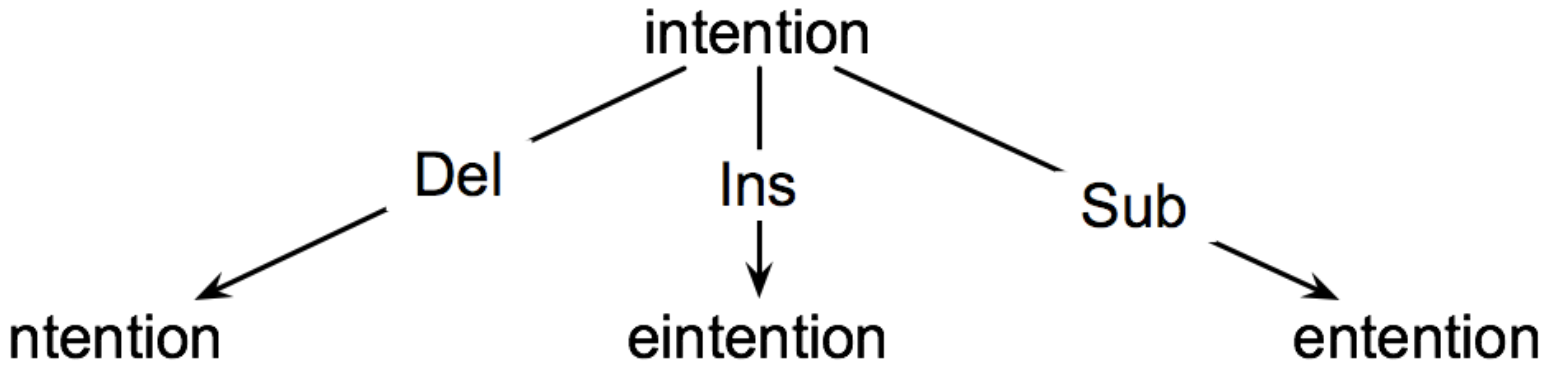
- Evaluating Machine Translation and speech recognition

R	Spokesman		confirms			senior	government	adviser	the	senior	was shot
H	Spokesman		said							adviser	was shot dead
			S		I			D			I

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announcedyesterday
  - for Stanford University President John Hennessy

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(*n*,*m*)
- Solving problems by combining solutions to subproblems.
- Bottom--up
  - We compute D(i,j) for small *i*,*j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i*,*j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Defining Min Edit Distance (Levenshtein)

- Initialization `D(i,0) = i`

  `D(0,j) = j`

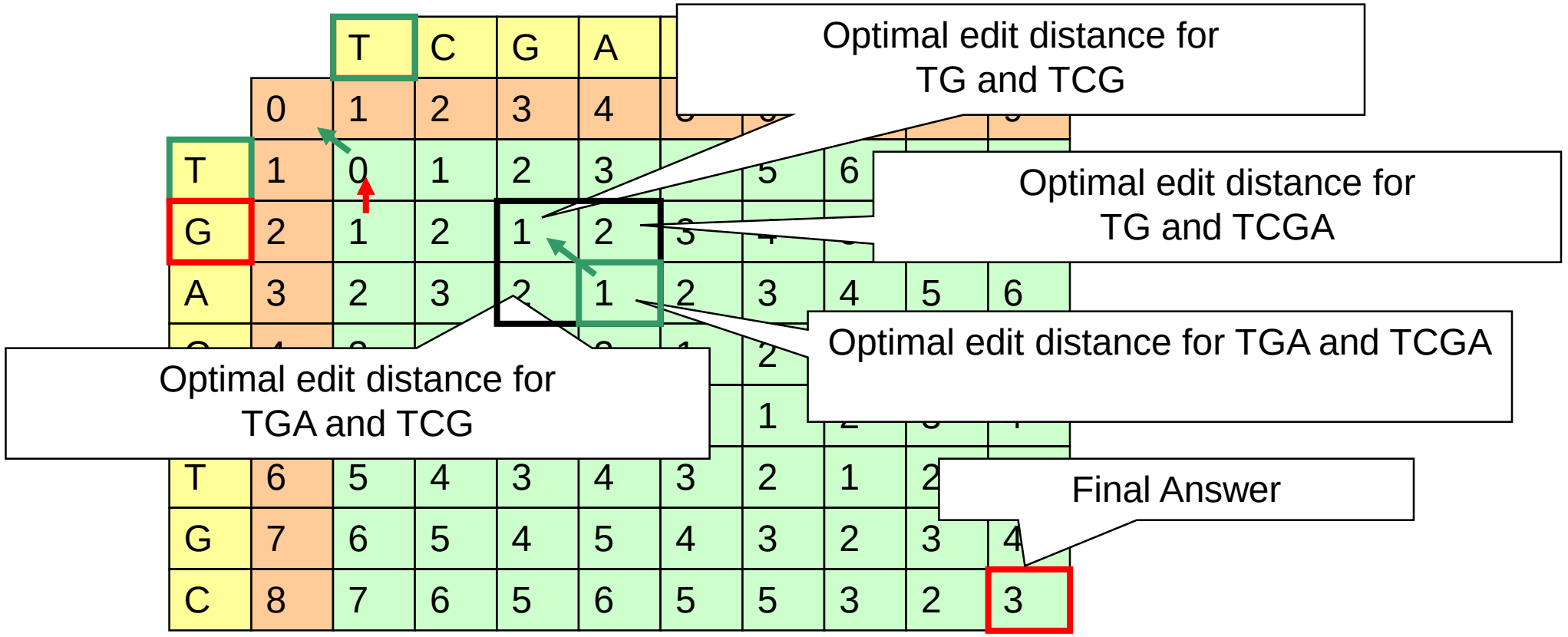
- Recurrence Relation*:*

```
For each  i = 1…M
      For each  j = 1…N
                                  D(i-1,j) + 1
                                  D(i,j-1) + 1
            D(i,j)= min           D(i-1,j-1)   +1;  if X(i) ≠ Y(j)
                                                0;  if X(i) = Y(j)
```

- Termination*:*

  `D(N,M) is distance`

Edit Distance – Dynamic Programming



Example

I want to find the edit distance between «abcdef» and «azced»

	Null	a	b	c	d	e	f	Exp.
Null	0	1	2	3	4	5	6	Null -> abcdef, costs 8 D(null, abcdef) = 8
a	1	0	1	2 D(a,abc)	3	4	5	D(A , abcdef) = 5
z	2	1	1	2	3	4	5	az -> abcdef
c	3	2	2	1	2	3	4	
e	4	3	3	2	2	2	3	
d	5	4	4	3	2	3	3	D(abcdef, azced) = 3
Exp.	**Null -> azced D(null, azced) = 5**							Transfer f to d Delete d Transfer b to z =3

Computing alignments

- Edit distance isn't sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a "backtrace"
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment

Edit distance in R

```
#load stringdist package
library(stringdist)


#calculate Levenshtein distance between two strings
stringdist("string1", "string2", method = "lv")
```

Java Code

```java
package org.arpit.java2blog;

import java.util.Scanner;

public class EditDistance {

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        String s1 = scn.nextLine();
        String s2 = scn.nextLine();

        System.out.println(editDist(s1, s2));

    }

    public static int editDist(String s1, String s2) {
        int[][] dp = new int[s1.length() + 1][s2.length() +

        /* this is the last column of the matrix which
         *  represents the result of empty string1
         *  and string2 starting from current row.
         */
        for (int row = s2.length(); row >= 0; row--) {
            dp[row][s1.length()] = s2.length() - row;
        }
        /* this is the last row of the matrix which
        * represents the result of empty string2
        *  and string1 starting from current col.
        */
        for (int col = s1.length(); col >= 0; col--) {
            dp[s2.length()][col] = s1.length() - col;
        }

        for (int col = s1.length() - 1; col >= 0; col--) {
            for (int row = s2.length() - 1; row >= 0; row--) {

                /* If characters are same, then the solution will be without
                 * these characters */
                if (s1.charAt(col) == s2.charAt(row)) {
                    dp[row][col] = dp[row + 1][col + 1];
                } else {
                /* else it will be minimum of these three operations
                */
                dp[row][col] = 1 + Math.min(dp[row + 1][col + 1], // replace
                                Math.min(dp[row][col + 1]  //   removal
                                , dp[row + 1][col]));   // insertion
                }
            }
        }

        return dp[0][0];
    }
```

String similarity measures

- Token-based
  - Examples
    - Jaccard
    - TF-IDF Cosine similarities
  - Suitable for large documents
- Character-based
  - Examples:
    - Edit-distance and variants like Levenshtein, Jaro-Winkler
    - Soundex
  - Suitable for short strings with spelling mistakes
- Hybrids