Least Squares; Sequence Alignment

1. Segmented Least Squares
   - multi-way choices
   - applying dynamic programming

2. Sequence Alignment
   - matching similar words
   - applying dynamic programming
   - analysis of the algorithm
1 Segmented Least Squares
   - multi-way choices
   - applying dynamic programming

2 Sequence Alignment
   - matching similar words
   - applying dynamic programming
   - analysis of the algorithm
fitting data in the least squares sense

Input: \( n \) points \((x_i, y_i), \ x_1 < x_2 < \cdots < x_n\).

Output: \( a, b \), coefficients of \( y = ax + b \), so
\[
\sum_{i=1}^{n} (y_i - ax_i - b)^2
\]
is minimal over all \( a \) and \( b \).
data close to two lines
fitting data close to two lines

- Red line: $y = 0.521206102392563x + 1.0143995144942668$
- Green line: $y = 1.9641573446039344x + 1.0216814605887747$

[Graph showing the data points and the two fitted lines]
explicit formulas for a linear least squares fit

The coefficients $a$ and $b$ of the line $y = ax + b$ for the least squares fit of the $n$ points $(x_i, y_i)$ satisfy

$$a = \frac{n \sum_{i=1}^{n} x_i y_i - \left( \sum_{i=1}^{n} x_i \right) \left( \sum_{i=1}^{n} y_i \right)}{n \sum_{i=1}^{n} x_i^2 - \left( \sum_{i=1}^{n} x_i \right)^2}$$

and

$$b = \frac{1}{n} \left( \sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i \right).$$

Exercise 1:
Express the cost to compute $a$ and $b$ as $O(f(n))$, for some function $f$. Justify your formula for $f(n)$. 

Computer Algorithms I (CS 401/MCS 401)  Least Squares; Sequence Alignment  L-13  18 July 2018  6 / 38
problem: detect the direction of change
solved by three linear least squares fits

- Red line: $y = 0.47421649844266295x + 0.9887188901904131$
- Green line: $y = 2.00727212429138x + 0.9974128519452048$
- Blue line: $y = 0.2330720294759922x + 2.860956329656263$
partition the data into segments

Input: \( n \) points \((x_i, y_i), \ x_1 < x_2 < \cdots < x_n\).

Each point is represented by an index \( i, \ i \in \{ 1, 2, \ldots, n \} \).

**Definition**

Given a set of \( n \) indices \( \{ 1, 2, \ldots, n \} \), a segment \((i, j)\), with \( i \leq j \), defines the sequence of points \( i, i + 1, \ldots, j \).

The data will be segmented into a partition.

**Definition**

Given a set of \( n \) indices \( \{ 1, 2, \ldots, n \} \), a partition into segments is a sequence of \( m \) segments \((i_k, j_k)\), for \( k = 1, 2, \ldots, m \), with

- \( i_1 = 1 \) and \( i_m = n \);
- \( i_{k+1} = j_k + 1 \), for \( k = 1, 2, \ldots, m - 1 \).
the segmented least squares problem

Input: \( n \) points \((x_i, y_i), x_1 < x_2 < \cdots < x_n\).
Output: a partition of the points in segments with minimal penalty.

**Definition**

Given \( n \) points \((x_i, y_i), x_1 < x_2 < \cdots < x_n\) and a partition into \( m \) segments \((i_k, j_k)\), *the penalty* is the sum of

1. \( m \times C \), where \( C > 0 \);
2. \( \sum_{\ell=i_k}^{j_k} (y_\ell - ax_\ell - b)^2 \) is the sum of squared distances of the linear fit to the data points in the \( k \)-th segment.
Is there a unique solution?

Exercise 2:
Does the segmented least squares problem have a unique solution?
Justify your answer with proof and/or examples.
Segmented Least Squares
- multi-way choices
- applying dynamic programming

Sequence Alignment
- matching similar words
- applying dynamic programming
- analysis of the algorithm
guidelines for dynamic programming

For dynamic programming to apply, we need a collection of subproblems derived from the original problem that satisfies the properties:

1. There are a polynomial number of subproblems.
2. The solution to the original problem can be easily computed from the solutions to the subproblems.
3. There is a natural order from smallest to largest subproblem, jointly with an easy-to-compute recurrence.

There is a chicken-and-egg relationship between

- the decomposition into subproblems; and
- the recurrence linking the subproblems.
a useful observation

The last segment has to end in \( n \), so \( j_m = n \).

What about the start of the last segment? What is \( i_m \)?

The penalty is the sum of

1. \( C \), from the segment \( (i_m, n) \),
2. \( \sum_{\ell=i_m}^{n} (y_{\ell} - a_m x_{\ell} - b_m)^2 \), the error of the fit, and
3. the penalty of the partition of \( \{ 1, 2, \ldots, i_m - 1 \} \).

Observe that the value of \( m \) does not matter. Abbreviate \( i_m \) by \( i \).
finding a good notation for the observation

For \( i \in \{ 1, 2, \ldots, n \} \) denote by \( \text{OPT}(i) \) the penalty for the partition of \( \{ 1, 2, \ldots, i \} \) in an optimal solution. In the trivial case, \( \text{OPT}(0) = 0 \).

Let \( e_{i,n} = \sum_{\ell=i}^{n} (y_\ell - a_m x_\ell - b_m)^2 \) denote the error of the fit.

The useful observation translates then into a recurrence.

**Lemma**

*Let the last segment in an optimal partition of \( n \) data points begin at \( i \). Then, the value of that optimal solution is*

\[
\text{OPT}(n) = C + e_{i,n} + \text{OPT}(i - 1).
\]
designing a general recurrence

The previous recurrence has its focus on the last segment. We still have not answered the *What is i?* question.

What if we tried all values for \( i \), for \( i = 1, 2, \ldots, n \)? For every tried value for \( i \), consider:

- taking \( i \) as the start of a segment has the cost \( C + e_{i,n} + \text{OPT}(i - 1) \),
- optimizing happens by taking the minimum cost over all \( i \).

This leads then to a more general recurrence:

**Lemma**

*The value of an optimal solution of a partition of \( n \) points into segments is*

\[
\text{OPT}(n) = \min_{1 \leq i \leq n} (C + e_{i,n} + \text{OPT}(i - 1)).
\]
Exercise 3:
The recurrence

\[
\text{OPT}(n) = \min_{1 \leq i \leq n} (C + e_{i,n} + \text{OPT}(i - 1)).
\]

suggests a recursive algorithm.

1. Formulate the recursive algorithm in pseudo code.
2. For \( n = 5 \), draw the tree of subproblems.
3. Extrapolate your drawing of the tree of subproblems for \( n = 5 \) into a general formula for the cost of the recursive algorithm, for any \( n \).
an iterative algorithm

Segmented-Least-Squares($n, x, y, C$)
Input: $n$, the number of data points,
$x$, $x$-coordinates $x_1 < x_2 < \cdots < x_n$,
$y$, $y$-coordinates of the data points,
$C$, the penalty for a segment.
Output: returns $\text{OPT}(n)$.

Let $M$ be an array indexed by $0, 1, \ldots, n$
$M[0] := 0$

for all pairs $(i,j)$, $i \leq j$, do $e_{i,j} := \sum_{\ell=i}^{j} (y_{\ell} - a_{i,j}x_{\ell} - b_{i,j})^2$

for $j = 1, 2, \ldots, n$ do
$M[j] := \min_{1 \leq i \leq j} (C + e_{i,j} + M[i - 1])$
return $M[n]$
Exercise 4:
Prove that Segmented-Least-Squares is correct.

1. Formulate a precise statement on the correctness.
2. Use the recurrence relation in your proof.
reporting segments in an optimal solution

Report-Segments\((n, C, e_{i,j}, M)\)

Input: \(n\), the number of data points,
\(C\), the penalty for a segment,
\(e_{i,j}\), fitting errors for all segments \((i, j)\),
\(M\), the array in \textit{Segmented-Least-Squares}.

Output: prints all segments in the optimal partition.

\[
\text{if } n = 0 \text{ then}
\quad \text{print nothing}
\]
\[
\text{else}
\quad \text{find } i \text{ that minimizes } C + e_{i,j} + M[i - 1]
\quad \text{print } (i, n)
\quad \text{Report-Segments}(i - 1, C, e_{i,j}, M)
\]

Exercise 5:
What is the cost of \textit{Report-Segments}? Justify your \(O(f(n))\) formula.
The cost of Segmented-Least-Squares

**Theorem**

*For n points, Segmented-Least-Squares runs in $O(n^3)$ time.*

**Proof.** The computation of $e_{i,j}$, for all pairs $(i, j)$, for $1 \leq i \leq j \leq n$ requires $O(n^2)$ steps. By exercise 1, each step requires $O(n)$ operations to compute the coefficients $a_{i,j}$ and $b_{i,j}$. The first for loop in Segmented-Least-Squares thus runs in $O(n^3)$ time.

The second for loop does $n$ steps, for all $j = 1, 2, \ldots, n$. The operation in the loop \( \min_{1 \leq i \leq j} \) requires

- $j$ sums of three elements to compute $C + e_{i,j} + M[i - 1]$,
- $j$ comparisons of the current minimum with $C + e_{i,j} + M[i - 1]$.

So the second loop runs in $O(n^2)$ time.

As $O(n^3) + O(n^2)$ is $O(n^3)$, Segmented-Least-Squares runs in $O(n^3)$ time. Q.E.D.
1. Segmented Least Squares
   - multi-way choices
   - applying dynamic programming

2. Sequence Alignment
   - matching similar words
   - applying dynamic programming
   - analysis of the algorithm
Consider the following situations.

- **Search an online dictionary for a word.**
  Often a user may be so unfamiliar with a word that the exact spelling of the word is not known.

- **Compile a program.**
  A helpful parser will match misspelled keywords or variable names and offer suggestions to correct the errors.

Words for us will be sequences of characters $x \in \{ 'a', 'b', \ldots, 'z' \}$. 
mismatched characters

Our alphabet is $\mathcal{A} = \{ 'a', 'b', \ldots, 'z' \}$.

As an example of a mismatch, consider $X = \text{occurrence}$ and $Y = \text{occurrence}$.

There is a mismatch at positions 6 and 6 of both $x$ and $y$, as $X_6 = a$ and $Y_6 = e$.

**Definition**

Let $X$ and $Y$ be two words of lengths respectively $m$ and $n$: $X = x_1x_2\cdots x_m$ and $Y = y_1y_2\cdots y_n$. A *mismatch* is a pair $(i, j)$, $i \in \{ 1, 2, \ldots, m \}$ and $j \in \{ 1, 2, \ldots, n \}$, for which $x_i \neq y_j$. 
Misspelled words may have missing characters.

As an example of a gap, consider \( X = o\text{---}currence \) and \( Y = occurrence \). There is a gap at positions 2 and 2 of both \( x \) and \( y \), as \( x_2 = - \) and \( y_2 = c \).

**Definition**

Let \( X \) and \( Y \) be two words of lengths respectively \( m \) and \( n \):

\[
X = x_1x_2 \cdots x_m \quad \text{and} \quad Y = y_1y_2 \cdots y_n.
\]

A gap is a pair \((i, j)\), \( i \in \{1, 2, \ldots, m\} \) and \( j \in \{1, 2, \ldots, n\} \),
for which \( x_i = - \) and \( y_j \neq - \) or \( x_i \neq - \) and \( y_j = - \).
an alignment between two words

In the definitions below we consider two words $X$ and $Y$, of lengths respectively $m$ and $n$:

$X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$.

**Definition**

A *matching* is a sequence of pairs $(i, j)$, $i \in \{ 1, 2, \ldots, m \}$ and $j \in \{ 1, 2, \ldots, n \}$, such that each $x_i$ and $y_j$ occurs in at most one pair.

**Definition**

A matching $M$ is *an alignment* if there are no crossing pairs: for all $(i, j), (k, \ell) \in M$: if $i < k$, then $j < \ell$.

For example, for $x = \text{stop-}$ and $y = \text{--tops}$, the alignment is $(2, 1), (3, 2), (4, 3)$. 
Suppose $M$ is a given alignment between $X$ and $Y$:

1. The *gap cost* $\delta$ is applied for each pair $(i, j) \in M$ which is a gap.

2. The *mismatch cost* $\alpha_{x_i, y_j}$ is applied for each pair $(i, j) \in M$ which is a mismatch.

The *cost* of $M$ is the sum of all gap and mismatch costs.

Note that the parameters determine which alignment of “occurance” and “occurrence” we should prefer:

- “occurance” and “occurrence” have a cost of $\delta + \alpha_{a,e}$,
- “ocurrance” and “occurence” have a cost of $3\delta$. 

1 Segmented Least Squares
   - multi-way choices
   - applying dynamic programming

2 Sequence Alignment
   - matching similar words
   - applying dynamic programming
   - analysis of the algorithm
a dichotomy

Let $X$ and $Y$ be two words of length $m$ and $n$ respectively.

In an optimal solution $M$,

- either $(m, n) \in M$,
- or $(m, n) \notin M$.

We need a stronger statement.

**Lemma**

Let $M$ be any alignment of $X$ and $Y$. If $(m, n) \notin M$, then

- either the $m$-th position of $X$ is not matched in $M$,
- or the $n$-th position of $Y$ is not matched in $M$. 
the lemma with proof

**Lemma**

Let $M$ be any alignment of $X$ and $Y$. If $(m, n) \notin M$, then

- either the $m$-th position of $X$ is not matched in $M$,
- or the $n$-th position of $Y$ is not matched in $M$.

**Proof.** Assume by contradiction there is a matching of the $m$-th position of $X$ and the $n$-th position of $Y$: we have $i < m$ and $j < n$ so that

- $(m, j) \in M$,
- $(i, n) \in M$;

and $(m, n) \notin M$.

Recall that in an alignment there is no crossing, but we have $(i, n), (m, j) \in M$ with $i < m$ and $n > i$, a crossing. This crossing contradicts the assumption $(m, n) \notin M$. Q.E.D.
Lemma

Let $M$ be any alignment of $X$ and $Y$. If $(m, n) \notin M$, then
- either the $m$-th position of $X$ is not matched in $M$, or
- the $n$-th position of $Y$ is not matched in $M$.

In our way to a recurrence, we restate the lemma:

Lemma

In an optimal alignment $M$, at least one of the following holds:
1. $(m, n) \in M$; or
2. the $m$-th position of $X$ is not matched; or
3. the $n$-th position of $Y$ is not matched.
Let $\text{OPT}(i,j)$ denotes the minimum cost of an alignment between

- $x_1 x_2 \cdots x_i$, and
- $y_1 y_2 \cdots y_j$.

The restated lemma has three cases:

1. If $(m, n) \in M$, then

   $$\text{OPT}(m, n) = \alpha_{x_m, y_n} + \text{OPT}(m - 1, n - 1).$$

2. If the $m$-th position of $X$ is not matched, we have a gap, and then

   $$\text{OPT}(m, n) = \delta + \text{OPT}(m - 1, n).$$

3. If the $n$-th position of $Y$ is not matched, we have a gap, and then

   $$\text{OPT}(m, n) = \delta + \text{OPT}(m, n - 1).$$
Theorem

The minimum alignment costs satisfy the following recurrence, for \( i \geq 1 \) and \( j \geq 1 \):

\[
\text{OPT}(i, j) = \min \left( \alpha_{x_i, y_j} + \text{OPT}(i - 1, j - 1), \right.
\left. \delta + \text{OPT}(i - 1, j), \delta + \text{OPT}(i, j - 1) \right).
\]

The trivial cases are
- \( \text{OPT}(i, 0) = i\delta \),
- \( \text{OPT}(0, i) = i\delta \),

as the only way to line up an \( i \)-letter word with a 0-letter word is to use \( i \) gaps.

With the recurrence we can formulate an algorithm.
the algorithm

Alignment($m$, $n$, $X$, $Y$, $\delta$, $\alpha_{a,b}$)

Input: $m$ and $n$, the number of letters in $X$ and $Y$, $X$ and $Y$, words of length $m$ and $n$, $\delta$, the gap cost, $\alpha_{a,b}$, the mismatch cost between $a$ and $b$.

Output: returns $\text{OPT}(m, n)$.

let $A[0 \ldots m, 0 \ldots n]$ be an array of arrays
for $i$ from 0 to $m$ do $A[i, 0] := i\delta$
for $j$ from 0 to $n$ do $A[0, j] := j\delta$
for $j$ from 1 to $n$ do
  for $i$ from 1 to $m$ do
    $A[i, j] = \min\left(\begin{array}{c}
      \alpha_{x_i, y_j} + A[i - 1, j - 1], \\
      \delta + A[i - 1, j], \\
      \delta + A[i, j - 1]
    \end{array}\right)$
return $A[m, n]$. 
1. Segmented Least Squares
   - multi-way choices
   - applying dynamic programming

2. Sequence Alignment
   - matching similar words
   - applying dynamic programming
   - analysis of the algorithm
the cost of the algorithm

Theorem

For words of length m and n, **Alignment** runs in $O(mn)$ time.

Exercise 6:
Prove the theorem about the cost of **Alignment**.
In the sequence alignment problem, we build a graph $G_{XY}$:

- the nodes are labeled by $(i, j)$, for $i = 0, 1, \ldots, m$, and for $j = 0, 1, \ldots, n$,
- the weights on the edges are the costs:
  - the cost of each vertical and horizontal edge is $\delta$,
  - the cost of each diagonal edge from $(i - 1, j - 1)$ to $(i, j)$ is $\alpha_{x_i, y_j}$.
Proposition

Let $f(i, j)$ denote the minimum cost of a path from $(0, 0)$ to $(i, j)$ in $G_{XY}$. Then, for all $i, j$, we have $f(i, j) = \text{OPT}(i, j)$. 

the minimum cost path in $G_{XY}$