Thesis Meeting 7

kense, for the thesis

Comparing Traces

- Comparing two traces requires a distance metric.
  - Bag of Activities
  - K-gram Model
  - Hamming Distance
  - Levenshtein Distance
- Levenshtein distance makes a lot of sense, however, there are issues with cost of operations (affecting length mismatch, label context, etc).

Modified Levenshtein Distance

- Base modification (following the paper)
  - Derive substitution costs
  - Derive Insertion costs
  - Modify edit distance with costs
- Additional modifications (new things to try)
  - Sub-conversations
  - Archetypes/Areas of similarity

Substitution Costs - Overview

1. Define the symbols
2. List the 3-grams, and frequency
3. Define the context for each symbol
4. Determine the pair-contexts
5. Determine the co-occurrence counts for each pair-context
6. Determine the co-occurrence counts total
7. Define the norm on the count of co-occurrence total
8. Normalize co-occurrence counts total
9. Determine the probability of all symbols
10. Determine the expected value of occurrence for pairs of symbols
11. Determine the substitution scores

Substitution Costs - Define Symbols

An event log $\boldsymbol{E}$ is defined as

$\boldsymbol{E = \{t_1, t_2, t_3, \ldots t_n\}}$ where $\boldsymbol{n}$ is the number of traces in the event log, and $\boldsymbol{t_i}$ is defined as a trace.

A trace $\boldsymbol{t_i}$ is defined as

$\boldsymbol{t_i = (\ell_1, \ell_2, \ell_3, \ldots \ell_w)}$ where $\boldsymbol{w}$ is the number of actions in the given trace.

A label $\boldsymbol{\ell_i}$ is defined from the set of labels $\boldsymbol{\ell}$, the 11 labels predetermined.

Substitution Costs - Define Symbols

- The list of symbols are obtained by taking each trace in the eventlog and obtaining the union of the sets of symbols in each trace.

Function: Define Symbols

Input: list (of traces)

Output: list (of symbols)

Substitution Costs - List the 3-grams and Freq

- The list of 3-grams are obtained by taking each trace in the event log, iterating in increments of three to record the 3-gram, and incrementing the frequency for each recorded 3-gram using a dictionary.

Function: 3-grams and frequency

Input: list (of traces)

Output: dictionary, dictionary (3-grams, frequency)

Substitution Costs - Defining Context

A context $\mathcal{X}_{\mathbf{a}}$ is the set of all contexts for a symbol ***a***, where $\mathbf{a} \in \boldsymbol{\ell}$

A context is in the form ***xy***, such that ***xay*** is in the list of 3-grams

Substitution Costs - Defining Context

- Context is obtained by taking each 3-gram in the form ***xay***, and storing ***xy*** in the dictionary of list of symbols ***a***, for all 3-grams in the list of 3-grams.

Function: Defining Context

Input: list (of 3-grams)

Output: dictionary (of contexts, where the key is the symbol, and the value is a list of contexts for that symbol)

Substitution Costs - Determine pair-contexts

A pair-context is a context $\mathcal{X}_{(a,b)}$, which is the set of contexts common to symbols ***a*** and ***b***.

- This is obtained via a union of the two sets of contexts for symbols ***a*** and ***b***.

Function: Determine pair-contexts

Input: dictionary (of contexts, where the key is the symbol, and the value is the list of contexts for that symbol)

Output: dictionary (of pair-contexts, where the key is the pair of symbols, and the value is the list of contexts for that pair of symbols)

Substitution Costs - Determine Co-Occurrence Counts

A Co-Occurrence is defined as $C_{xy}(a,b)$ where ***xy*** denotes the context for the symbols ***a*** and ***b***. $xy \in \mathcal{X}_{(a,b)}$

- Algorithmically, $C_{xy}(a,b) = n_i * n_j$, where n_i and n_j are the frequency counts for the 3-grams ***xay*** and ***xby*** respectively.
- Algorithmically, $C_{xy}(a,a) = n(n-1)/2$, where n is the frequency count for the 3-gram ***xay***.

Input: list (of symbols), dictionary (of context-pairs), dictionary (of 3-gram freq)

Output: dictionary (where the key is the co-occurrence definition, and the value is the count/value for co-occurrence.

Substitution Costs - Determine Co-Occurrence Counts

- Co-Occurrence is the count for when symbols appear in contexts that are similar to one another.
- For instance, the sequence ***‘abbabbabba’*** and ***‘cbbcbbcbbc'*** have symbols ***a*** and ***c*** appear in similar contexts.
- This helps the issue of context in modifying the edit distance as discussed previously.

Substitution Costs - Co-Occurrence Totals

Co-Occurrence totals $\boldsymbol{C(a,b)}$ is defined as the co-occurrence for symbols $\boldsymbol{a}$ and $\boldsymbol{b}$ for all contexts in $\mathcal{X}_{(a,b)}$

Substitution Costs - Normalization

- All co-occurrence totals ***C(a,b)*** are then summed to be the norm.
- All co-occurrence totals are then normalized, and organized in matrix format

M(a,b) = [C(a,b)/norm]

Substitution Costs - Probability of Occurrence

- $\boldsymbol{p_a}$, the probability of occurrence for symbol ***a*** is defined as

  $$\boldsymbol{p_a = M(a,a) + \sum_{a!=b} M(a,b)}$$

Substitution Costs - Expected Values

- The expected value of occurrence of a pair of symbols is defined:

  $\boldsymbol{E(a,b) = p_a^2}$, if $\boldsymbol{a = b}$

  $\boldsymbol{= 2p_a p_b}$, otherwise

Substitution Costs - Substitution Score

- The final score for substitution of a symbol a with b is defined as a matrix of values, derived as a likelihood ratio.

  $$S(a,b) = \log_2(M(a,b)/E(a,b))$$

Substitution Costs - Usage

- We would take a event log of lots of traces, and apply the algorithm defined in order to obtain a matrix of costs for substitutions of symbols ***a*** and ***b***.
- Thus, we can use these values to determine how a substitution inside the edit-distance should be evaluated (instead of a generic unit cost as per Levenshtein).

Insert Costs - Overview

- Steps 1-3 are the same for Insert Costs

  4. Determine the occurrence of the 3-gram ***xay***

  5. Determine the counts of ‘Right Given Left’

  6. Define the norm for the counts of RGL

  7. Determine the probability of occurrence of symbols

  8. Normalize the values for counts of RGL

  9. Calculate the Insert Costs as a likelihood ratio

Insert Costs - Count of ‘Right Given Left’

For each symbol $\boldsymbol{a, x} \in \boldsymbol{\ell}$, where $\boldsymbol{O_{xy}(a)}$ is the count of the occurrence of ***xay***, we can access the value via the 3-gram function.

$$\boldsymbol{RGL(a/x) = \sum_{y|xy \in \mathcal{X}_a} O_{xy}(a)}$$

- This function will take the counts for all 3-grams that begin with ***xa*** in the set of contexts $\mathcal{X}_a$.
- Essentially, finds the counts for when ***a*** appears after ***x***.

Insert Costs - Insertion Score

- The final score for insertion of a symbol ***b*** given symbol ***a*** is defined also as a likelihood ratio:

  $$InsSc\ (a/b) = \log_2(RGL(a/b)/p_a*p_b)$$

Insert Costs - Usage

- Finds similarity for sequential labels occurring
- Shows that both functions (sub and indel) are likelihood ratios, for probability of occurrence of the labels vs trends of occurrence
- Provided correct implementation, we can use the same logic to apply bigger trends

Modifying the Edit Distance

- In edit-distance, specifically Levenshtein, all operations are unit cost.
- For the modification, instead of unit cost, the cost will correspond to the scores derived for EACH operation.

→ insert +1

↓ delete +1

↘ sub +1

trace2 j

trace1 i

min	_	a	b	c	a	a	c	b	c
_	0	1	2	3	4	5	6	7	8
a	1	0	1	2	3	4	5	6	7
b	2	2	0	1	2	3	4	5	6
c	3	3	1	0	1	2	3	4	5
b	4	4	3	1	1	2	3	3	4

if insert, compare trace1[i] w/ trace2[j]

insert(b/a)

Modifying the Edit Distance - Similarity Version

- Use the algorithm, but in reverse for similarity (argmax instead)
  - Special case of 1000 as “no operations taken” or perfect similarity (arbitrary starting threshold)
- A cost is then applied whenever the operation is chosen, according to the cost matrix derived from the event log
- The result is a score that takes into account context and trends present in the traces that we fed in (much akin to “training”)

Considerations for the algorithm

- Algorithm probably assumes that the event log consists of traces that are largely obtained from the same set up (i.e: event log is from recorded hospital processes)
- This means the cost scores will be affected if we treat it similar to ML (in the most extreme cases)
- But the metric still captures short-term dependencies (co-occurrence, context)
- We can modify to capture long-term dependencies (sub-conversations?)

Sub-conversations

- We can describe a long-term dependency as a sub-conversation, a sequence of labels, given by LTL terms.
- Monologue: (give.statement OR give.opinion) UNTIL NOT(give.statement OR give.opinion)
- Share.Memory: (recall) UNTIL (closed.question OR open.question)
- We can describe these sequences through frequency.

Sub-conversations

- Similar conversations should share similar occurrences (normalized to length) of these sub-conversations.
- This assumption holds depending on how strictly we define the sub-conversations to look for. (i.e: interview-based conversation vs query-based)

Tasks

1. Look at describing longer sequences using LTL
2. Examine using LTL descriptions as an additional factor in similarity
3. <<Reserved>>