A NOVEL STRING DISTANCE FUNCTION BASED ON MOST FREQUENT K CHARACTERS
Sadi Evren SEKER, Oguz ALTUN, Ugur AYAN, Cihan MERT

Speaker: Sadi Evren SEKER
MOTIVATION

- Data Mining
- Streaming Data Mining
- Big Data (Social Networks)
- Bioinformatics (genetics)
FEATURE HASHING

- Idea is hashing the features while reducing the size and keeping the maximum available feature on the feature vectors
- String Hashing can help on Text Mining and Text Features
- Also String Distance Function helps for String operations (comparison, storing)
STRING DISTANCE

- Hashing the text without losing features (which features to carry out?)
- SDF Samples
  - Levenshtein Distance
  - Jaccard Coefficient
  - Tanimoto Distance
LEVENSHTEIN DISTANCE

\[
\text{lev}_{a,b}(i, j) = \begin{cases} 
\max(i, j) & \text{if } \min(i, j) = 0, \\
\min \left\{ \begin{align*}
\text{lev}_{a,b}(i-1, j) + 1, \\
\text{lev}_{a,b}(i, j-1) + 1, \\
\text{lev}_{a,b}(i-1, j-1) + [a_i \neq b_j]
\end{align*} \right. & \text{otherwise.}
\end{cases}
\]

\[
\text{lev}_{\text{kitten}, \text{sitting}}(|\text{kitten}|, |\text{sitting}|) = \text{lev}_{\text{kitten}, \text{sitting}}(6, 7) = 3
\]

- kitten → sitten (substitution of "s" for "k")
- sitten → sittin (substitution of "i" for "e")
- sittin → sitting (insertion of "g" at the end).
TANIMOTO DISTANCE

\[ T_s(X,Y) = \frac{\sum_i (X_i \land Y_i)}{\sum_i (X_i \lor Y_i)} \]

\[ T_d(X,Y) = -\log_2(T_s(X,Y)) \]

STR1: 10010011
STR2: 10111011

\[ T_s = \frac{4}{6} = 0.66 \]

\[ T_d = 0.58 \]

\[ \text{STR1} \land \text{STR2} : 10010011 \]
\[ \sum_{\text{STR1} \land \text{STR2}} : 4 \]
\[ \text{STR1} \lor \text{STR2} : 10111011 \]
\[ \sum_{\text{STR1} \lor \text{STR2}} : 6 \]
JACCARD COEFFICIENT

\[
J(A, B) = \frac{|A \cap B|}{|A \cup B|}
\]

\[
d_j(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}
\]

STR1 : kitten
STR2 : sitting
Bi-gram (STR1) : {ki, it, tt, te, en}
Bi-gram (STR2) : {si, it, tt, ti, in, ng}
Bi-gram (STR1) \cap Bi-gram (STR1) : {it, tt,}
Bi-gram (STR1) \cup Bi-gram (STR1) : {ki, si, it, tt, te, ti, in, en, ng}
J = 2 / 9 = 0.22
d_j = 1 - 0.22 = 0.78
HAMMING DISTANCE BETWEEN BINARY STRINGS

HammingD (0000, 0001) = 1
HammingD (0001, 1111) = 3
LIMITATIONS

- Levenshtein distance has a high memory complexity
- Bit-wise operators (like Tanimoto Distance) loses the feature relation between input and output (we can include classical hashing functions (not tuned for string operations) into this group like MD5, SHA1)
- MD5 (‘A sample string’) = 15D086957773DBCEB6ABB944320D8CCE
- SHA1 (‘A sample string’) = 22A3432DAC42F2ED5FAE0AD8A0DD0D1B4D2ECFC5
A NOVEL STRING SIMILARITY METRIC

- **Max Frequent Characters**

- Get the max frequent n chars from any string. Keep the chars with their frequencies. If there are two chars with equal frequency, the firstly occurring char has priority.
MaxFreq2Hashing ALGORITHM

- 2 steps
  - Hashing
  - Distance Calculation

Algorithm 1: MaxFreq2Hashing

1. \( X \leftarrow h(str) \)
2. \( \text{for } i \leftarrow 0 \text{ to } \text{length}(str1) \)
3. \( \text{putHashMap}(str_i, \text{count(getHashMap}(str_i)+1) \)
4. \( c1 \leftarrow \text{getChar(maxHashMap,1)} \)
5. \( n1 \leftarrow \text{getCount(maxHashMap,1)} \)
6. \( c2 \leftarrow \text{getChar(maxHashMap,2)} \)
7. \( n2 \leftarrow \text{getCount(maxHashMap,2)} \)
8. \( x1 \leftarrow \text{concat}(c1,n1,c2,n2) \)
9. \( \text{return } x1 \)
Algorithm 2: Novel SDF

1. Let str1 and str2 be two strings to measure the distance between
2. \(X \leftarrow f(str1, str2, \text{limit})\)
3. \(x1 := h(str1)\)
4. \(x2 := h(str2)\)
5. def similarity := 0
6. if \(x1[0] == x2[0]\) then
7. similarity := similarity + \(x1[1] + x2[1]\)
8. if \(x1[0] == x2[2]\) then
9. similarity := similarity + \(x1[1] + x2[3]\)
10. if \(x1[2] == x2[0]\) then
11. similarity := similarity + \(x1[3] + x2[1]\)
12. if \(x1[2] == x2[2]\) then
13. similarity := similarity + \(x1[3] + x2[3]\)
14. return limit - similarity
SAMPLES FOR NOVEL SDF

- $h('research') = r2e2$
- because we have 2 ‘r’ and 2 ‘e’ characters with the highest frequency and we return in the order they appear in the string.
- $h('seeking') = e2s1$

**FASTA sample:**

```
Str1 = LCLYTHIGRNIYYGSYLYSETWNTGIMLLITMATAFMGYVLPWQGMSFWGATVITNLFSAIPIGTNLV
Str2 = EWIWGGFSVDKATLRFFAFHFILPFTMVAlAGVHHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
```

- $h(str1) = L9T8$
- $h(str2) = F9L8$
- $f(str1, str2, 100) = 83$

### Table

<table>
<thead>
<tr>
<th></th>
<th>Hashing Outputs</th>
<th>SDF Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>'night'</td>
<td>n1i1</td>
<td>9</td>
</tr>
<tr>
<td>'nacht'</td>
<td>n1a1</td>
<td></td>
</tr>
<tr>
<td>'my'</td>
<td>m1y1</td>
<td>10</td>
</tr>
<tr>
<td>'a'</td>
<td>a1NULL0</td>
<td></td>
</tr>
<tr>
<td>'research'</td>
<td>r2e2</td>
<td>6</td>
</tr>
<tr>
<td>'research'</td>
<td>r2e2</td>
<td></td>
</tr>
<tr>
<td>'aaaaabbbb'</td>
<td>a5b4</td>
<td>1</td>
</tr>
<tr>
<td>'ababababa'</td>
<td>a5b4</td>
<td></td>
</tr>
<tr>
<td>'significant'</td>
<td>i3n2</td>
<td>5</td>
</tr>
<tr>
<td>'capabilities'</td>
<td>i3a2</td>
<td></td>
</tr>
</tbody>
</table>
A SAMPLE PROBLEM

- IMDB62 Dataset

<table>
<thead>
<tr>
<th></th>
<th>IMDB62</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>62000</td>
</tr>
<tr>
<td>Texts per Author</td>
<td>1000</td>
</tr>
<tr>
<td>Average number of words per entry</td>
<td>300</td>
</tr>
<tr>
<td>Std. Dev. of words per author</td>
<td>198</td>
</tr>
<tr>
<td>Number of distinct words in corpus</td>
<td>139,434</td>
</tr>
</tbody>
</table>

Memory Requirement = 139,434 words x 62,000 posts x 300 average word length x 2 bytes for each character = ~ 4830 GByte
Memory Requirement (for texts) = 139,434 words x 62,000 posts x 300 average word length x 2 bytes for each character = ~ 4830 Gbyte = ~ 4 Tbyte

Memory Requirement (for frequency) = 139,434 words x 62,000 posts x 2 bytes for each number = 16.10 Gbyte

Machine Learning Algorithms requires more memory

Most of the algorithms (like SVM or ANN) are beyond the capacity of personal computers

Real life scenarios requires much much more memory
SUCCESS OF MaxFreqK ON IMDB’62

**TABLE III: ERROR RATES OF DISTANCE METHODS**

<table>
<thead>
<tr>
<th>Method</th>
<th>RMSE</th>
<th>RAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levenshtein Distance</td>
<td>29</td>
<td>0.47</td>
</tr>
<tr>
<td>Jaccard Index</td>
<td>45</td>
<td>0.68</td>
</tr>
<tr>
<td>Novel SDF</td>
<td>32</td>
<td>0.49</td>
</tr>
</tbody>
</table>

**TABLE IV: CUMULATIVE RUNNING TIMES**

<table>
<thead>
<tr>
<th>Method</th>
<th>Running Time</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levenshtein Distance</td>
<td>3647286.54sec</td>
<td>O(n*m) = O(n^2)</td>
</tr>
<tr>
<td>Jaccard Index</td>
<td>228647.22sec</td>
<td>O(n+m) = O(n)</td>
</tr>
<tr>
<td>Novel SDF</td>
<td>2712323.51sec</td>
<td>O(nlog n + mlog m) = O(nlog n)</td>
</tr>
</tbody>
</table>
SUCCESS OF MAXFREQ2HASHING

Fig. 5. Effect of K parameter on the success rate for MaxKFreqHashing

Fig. 6. Effect of K parameter on running time performance.
THANK YOU

Dr. Sadi Evren SEKER

www.SadiEvrenSEKER.com