Hľadanie podobností v textoch ľudových piesní

Michal Mižák

Školiteľ: doc. RNDr. Stanislav Krajči PhD

Výstup práce

- **Vložím pieseň** alebo úryvok textu
- Dostanem **zoznam podobných piesní** alebo pieseň, ktorá úryvok v nejakej forme obsahuje
- Dostanem **štatistické vyhodnotenie podobnosti**
- Porovnanie rôznych **konfigurácií** nášho **„Vector space“** algoritmu

Využitie

- Od dediny ku dedine, od domu k domu **pretvorený/ skomolený text**
- **Piesne získavajú od starších ľudí**, ktorí spievajú tak, ako si spomenú
  - Síce **autentické**, ale odborníci to môžu **opraviť**
- Porovnávanie **regionálnych rozdielov** v piesňach
- „Skonzistentnenie“ záznamov o piesňach
  - Nesprávny zápis hlások v nárečí

Využitie

- **Dzifče** počarovne, **ňezaľup** śe do mňe
  ja chlopec vandrovni, co ci budze zo mňe

  Ja chlapec vandrovni, zo šireho poľa
  co ci budze zo mňe, frajirečko moja?

- **Dzifče** počarovne, **nezaľub** se do mne
  ja chlapec vandrovny, co ci budze zo mňe

  Ja chlapec vandrovny, zo šireho poľa
  co ci budze zo mne, frajirečko moja?

- **Dzivče** počarovne, **nepopatraj** na mne
  ja chlapec vandrovny, co ci budze zo mne

  Ja chlapec vandrovný zo šireho poľa
  co ci budze zo mne draha duša moja?

Technológie

- Programovací jazyk Java
  - Knižnica JOOQ
- Databáza
  - MySQL

JOOQ v skratke

```
Long id = create.select()
        .from(T_SONG)
        .where(T_SONG.LYRICS.eq(song.getLyrics()))
        .fetchOne(T_SONG.SONGID);
```

```
SongRecord songRecord = create
        .insertInto(T_SONG, T_SONG.TITLE, T_SONG.LYRICS, T_SONG.CLEANLYRICS,
                    T_SONG.SONGSTYLE, T_SONG.REGION, T_SONG.SOURCE)
        .values(s.getTitle(), s.getLyrics(), s.getCleanLyrics(),
                s.getSongStyle(), s.getRegion(), s.getSource())
        .onDuplicateKeyIgnore()
        .returning(T_SONG.SONGID)
        .fetchOne();
```

Príprava databázy

- Scraping a rôzne formy skriptovania
  - Dáta bolo treba niekde zohnať
    - Viktor Gliganič - primáš (1. huslista) Ľudovej hudby FS Zemplín
    - Piesne372

„Čistenie“ textov

- Opakovačky
  - Štandardná štruktúra ľudovky
    - [: Sloha :] [: Refrén :]
  - Pre samotnú pieseň nie je opakovaný text refrénu/ slohy zaujímavý
    - Zátvorky a špeciálne znaky (teda {([/:.,‘“])}) tým pádom môžeme ignorovať
- Čísla ignorujeme tiež

Čistenie a parsovanie textov

- „ … “
  - Ignorovaný nedopísaný text
- Čiastočne zatriedenie piesní podľa regiónov, tónin (dur, mol) a charakteru (valčík, polka...)

Algoritmus

- Potrebujeme algoritmicky porovnať dokumenty
  - Information retrieval
- Myšlienka:
  - Vytvorím tabuľky
    - 1. riadok = slová, 2. riadok = početnosť
  - Získam vektor = druhý riadok
  - Vektory porovnám a získam ich algebraický vzťah

Vector space algorithm v rýchlosti

- „Ja som veta“

ja	som	veta
1	1	1

- <1, 1, 1>

- „Ja som veta, veta som ja“

ja	som	veta
2	2	2

- <2, 2, 2>

TermScheme - n-gramy

- Poradie?
  - Porovnávanie n-tíc slov = n-gramov

Vector space algorithm v rýchlosti

- „Un-gram

Ja	som	veta
1	1	1

- Bi-gram

Ja som	som veta
1	1

- Tri-gram

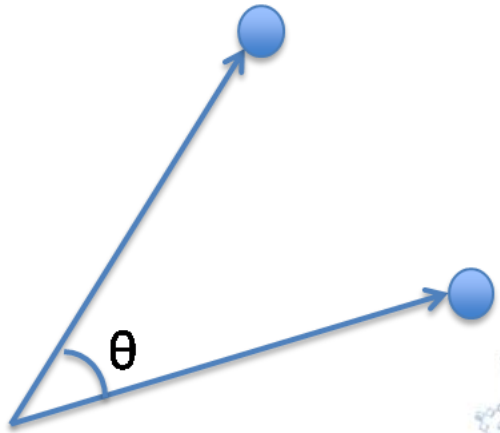
Ja som veta
1

- Máme teda vektory...

Vzdialenosť dvoch vektorov

- Kosínusová miera
  - Podiel skalárneho súčinu a súčinu euklidovských dĺžok vektorov

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

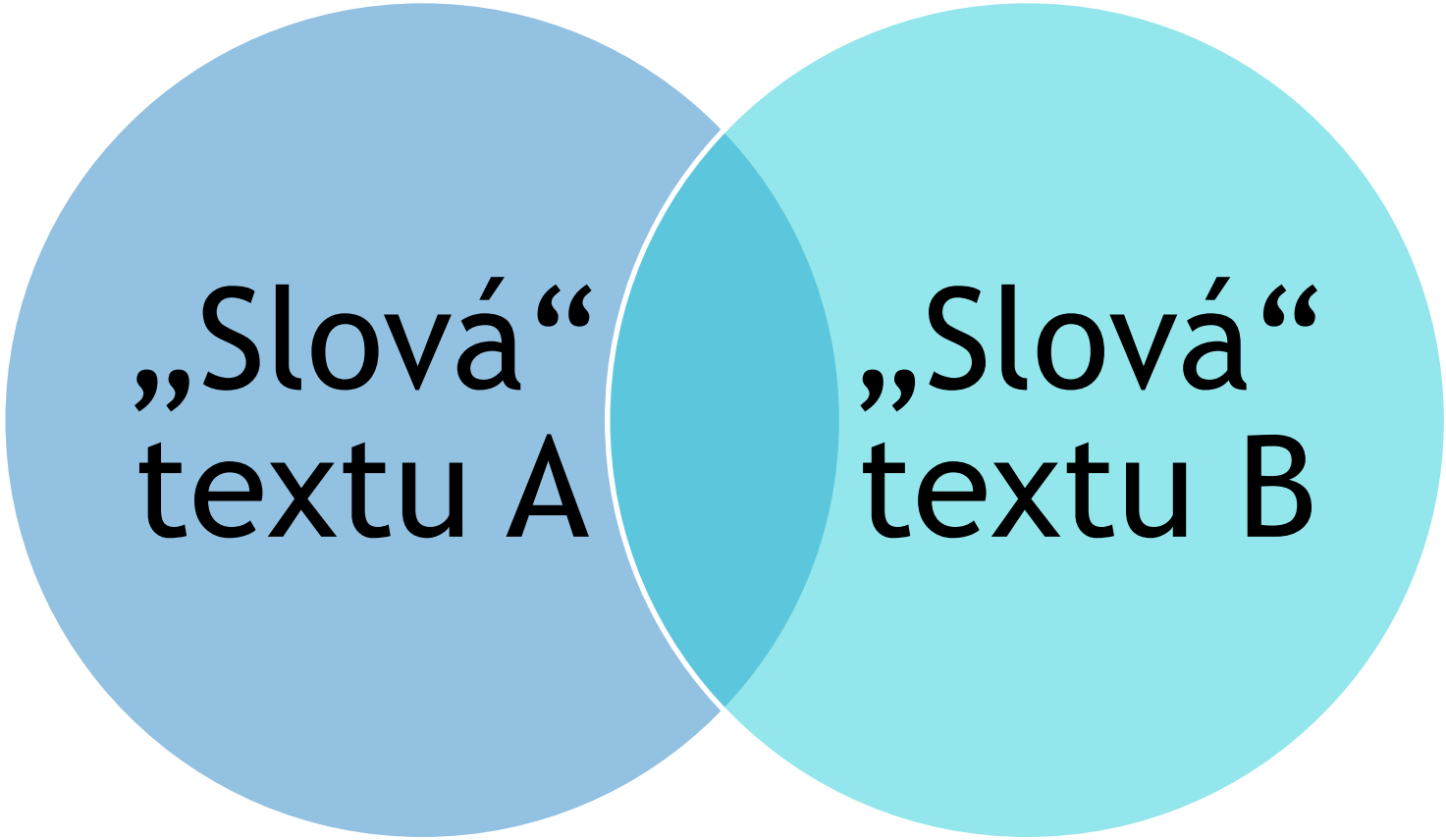


Vector inclusion

Poznámka – pracujeme s dimenziami vektora

Vector inclusion

- Vezmime vektory s dimenziami
  - <šírka, výška, hĺbka>
  - <výška, šírka, hĺbka, čas>
  - <šírka, výška>
- Chcem nalepiť plagát na skriňu...
- Musíme vektory „našteľovať“ podľa dimenzií


„Slová“ textu A
„Slová“ textu B

Vector inclusion

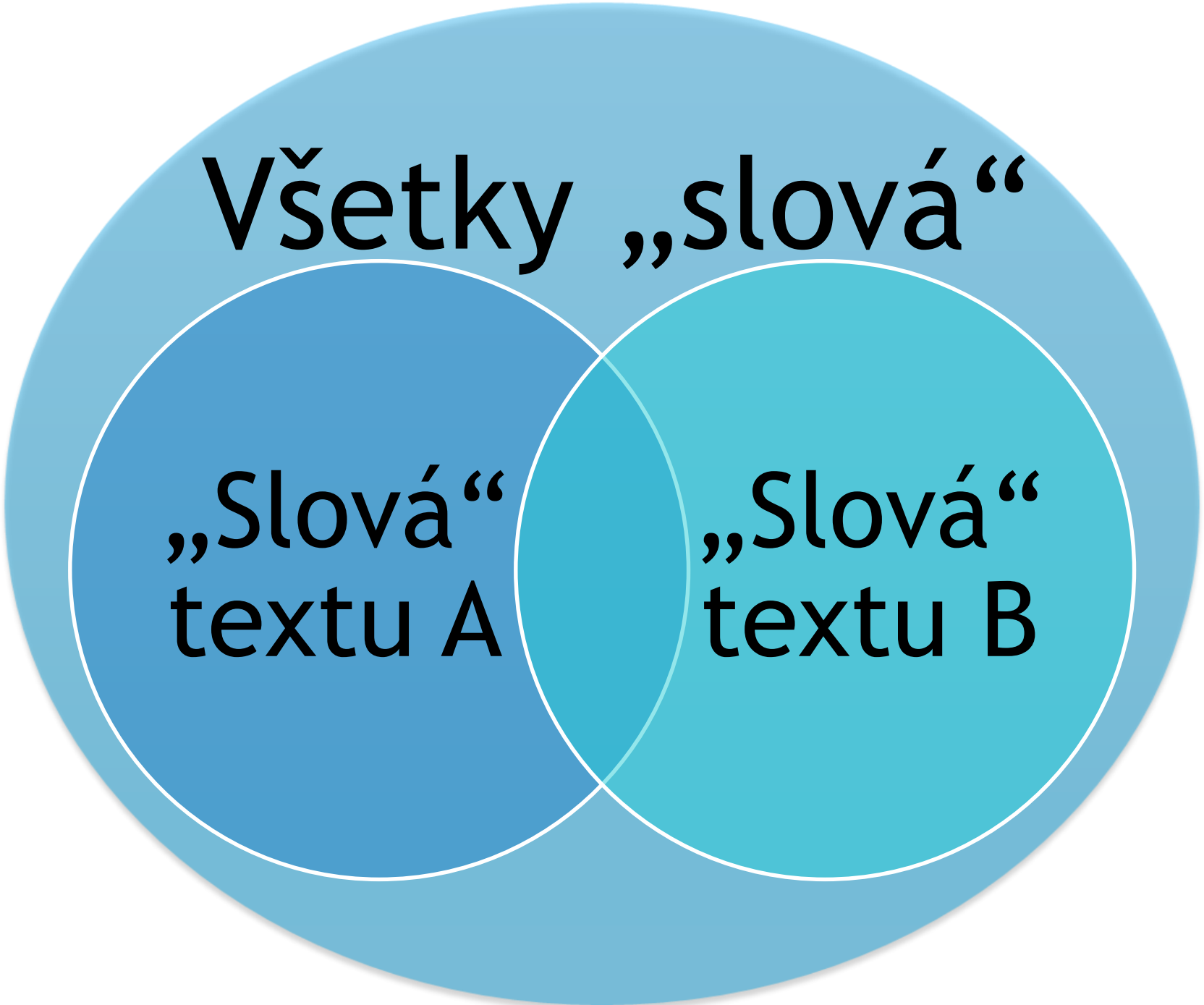
- Vektor v1 = <A, B, C> | < a1, b1, c >
  - Reprezentuje text A
- Vektor v2 = <B, A, E> | < b2, a2, e >
  - Reprezentuje text B

Vector inclusion

- A forma – Slovu C vo v2 priradíme 0
  - < A, B, C >
  - v1 = < a1, b1, c >
  - v2 = < a2, b2, 0 >
  - < B, A, E > zmeníme na < A, B, - >

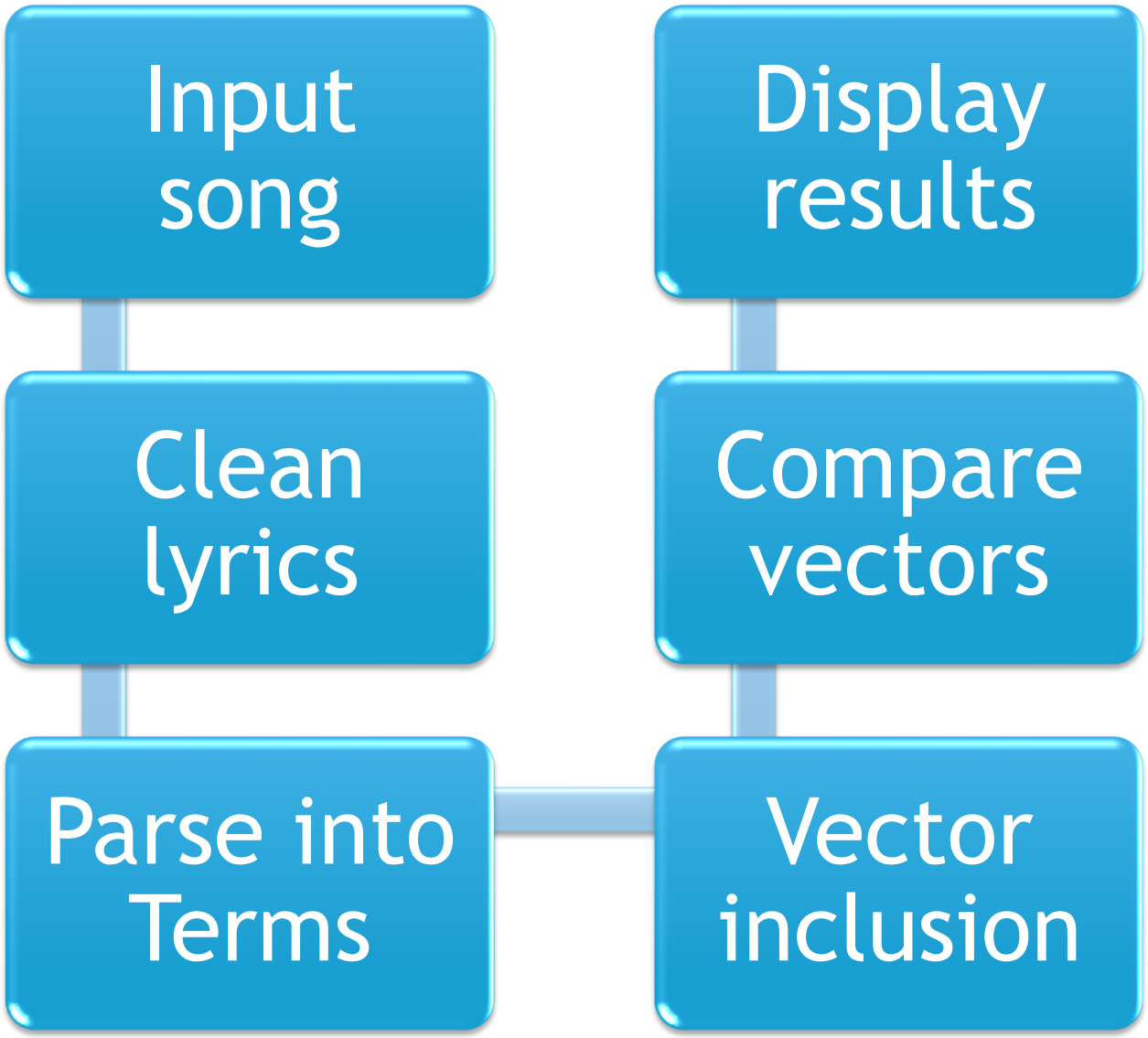
Vector inclusion

- B forma – Slovu E vo v1 priradíme 0
  - < B, A, E >
  - v1 = < b1, a1, 0 >
  - v2 = < b2, a2, e >
  - < A, B, C > zmeníme na < B, A, - >


Všetky „slová“
„Slová“ textu A
„Slová“ textu B

Vector inclusion

- Zjednotenie
  - < A, B, C, E >
- Prienik
  - < A, B>
- Všetky slová databázy
  - < A, B, C, D, E, F, G, H, I, J, K... >


Input song
Clean lyrics
Parse into Terms
Vector inclusion
Compare vectors
Display results

Zobrazenie výsledkov

- Pre každú pieseň *p* vrátim mapu
  s -> % podobnosť,
  kde *s* je každá pieseň rôzna od *p*

Pokročilé porovnávanie

Term weighting

- „Slová“ v texte sú pre nás rôzne zaujímavé
  - „A ja taka **čarna** jak **čarna čarnica**...“
  - „A ja taka **dzivočka cingi lingi bom**...“

TF

- Zohľadnenie frekvencie pojmu v **jednom dokumente**
- Rôzne vyhladzovacie techniky

TF

Logarithm tf

Idea logaritmického tf je využiť logaritmus ako škálovací mechanizmus. Nech $n_{p,d}$ je početnosť pojmu p v dokumente d. Potom

$$\text{logtf}(p, d) = 1 + \log(n_{p,d})$$

IDF – inverse document frequency

- Zohľadnenie frekvencie pojmu v rámci celej kolekcie
- Rôzne vyhladzovacie techniky

IDF – inverse document frequency

Idf

Nech n_{doc} je počet dokumentov v našej kolekcii. Nech df_p je početnosť pojmu p vo všetkých dokumentoch. Potom

$$\mathrm{idf}(p) = \log(n_{doc}/df_p)$$

TfIdf

- Zohľadnenie oboch
- Napríklad tf * idf

TermScheme - zovšeobecnenie

- „Ja chlapec vandrovny, zo šireho poľa“
  - Dvojice
    - <Ja, vandrovny>, <chlapec, zo>, <vandrovny, šireho>
    - <Ja, poľa>
  - Trojice
    - <Ja, chlapec, zo>, <chlapec, vandrovny, šireho>
  - Mnoho ďalších...

Term comparison algorithm

- Keď už porovnávame vektory, môžeme porovnávať aj „slová“
- Neoplatí sa zohľadňovať sémantiku slov

Jazykovedný prístup

- Lemmatization, stemming
  - Analýza efektivity existujúcich riešení pre náš problém, napr. STUBA
  - Ohýbanie riešení na ľudový jazyk
- Korpus nárečí Slovenského národného korpusu

Tvaroslovník

- Tvaroslovník – vytváranie vlastného ľudového „tvaroslovníka“
  - Podobnosti sú často veľmi veľké, stačí vymeniť „y za „i“, „š“ za „ś“, „h“ za „g“ a pod.
  - Generovanie koncoviek

Term similarity

- Levenshteinova vzdialenosť
  - Počet operácií potrebných na zmenu *s1* na *s2*
  - Vloženie, vymazanie a zmena písmena

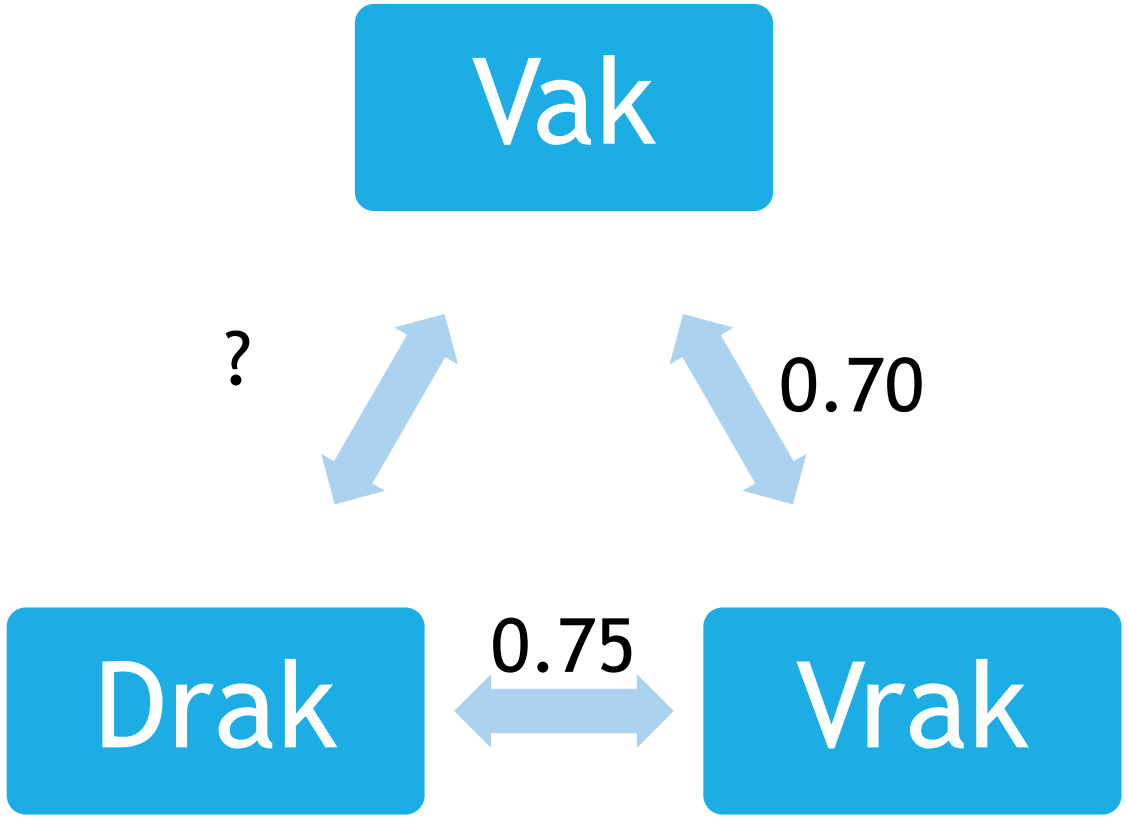
1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

- Mnoho ďalších...

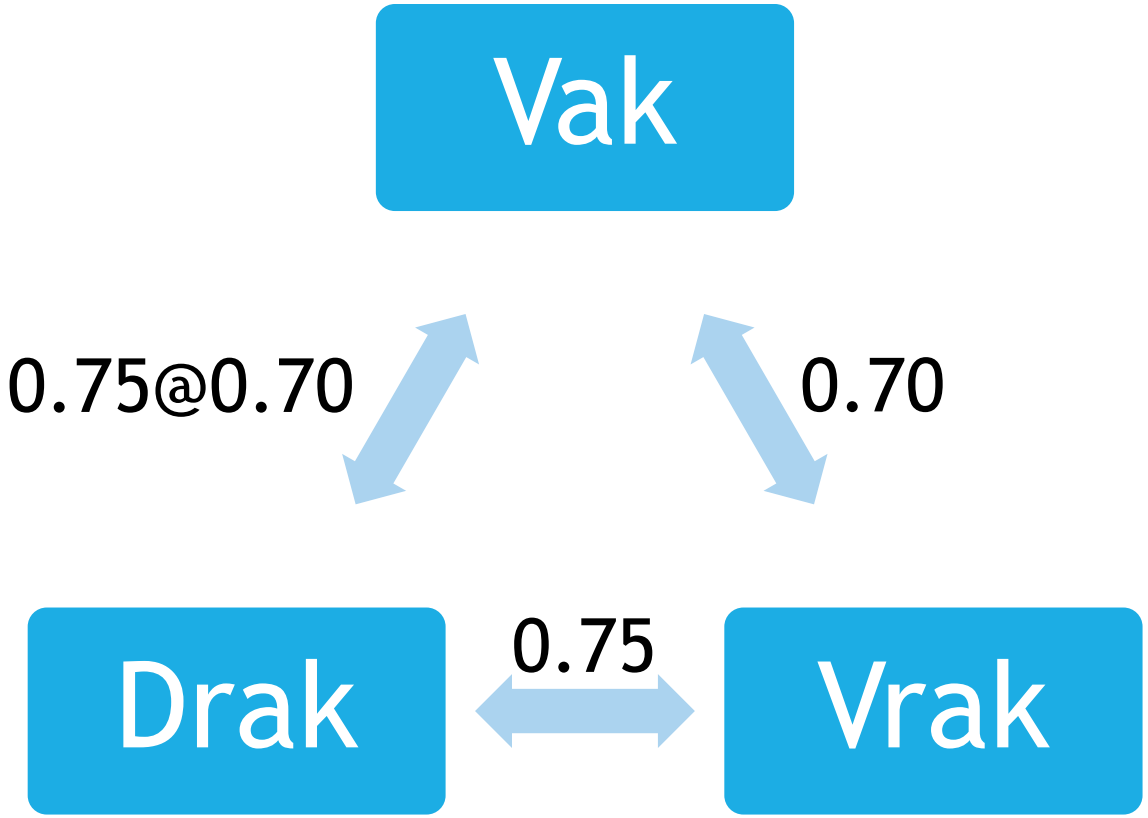
Indirect term similarity

- Idea – „slová“ si nemusia byť podobné priamo, ale cez nejakého suseda
- Ilustrácia - číslo naznačuje, nakoľko sú si „slová“ podobné v danom porovnávacom algoritme

Indirect term similarity



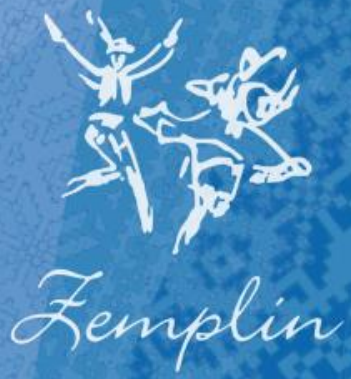
Indirect term similarity



Zemplin

Indirect term similarity

- Dostaneme teda vzorec...
- $Pi(a,b) =$
  $\max(\ P(a,b),\ \max(\ Pi(a,c)\ @\ Pi(b,c))),\ \ kde\ c\ \epsilon\ Vsetky"Slova"$
- Kde Pi je Indirect term similarity, P je Term similarity a a, b sú „slová“



Praktickejšie riešenie:

Term comparison + Tolerance

- Ak je podobnosť dvoch „slov“ v tolerancii, označíme ich za totožné
- Dáme ich do rovnakej **triedy ekvivalencie**

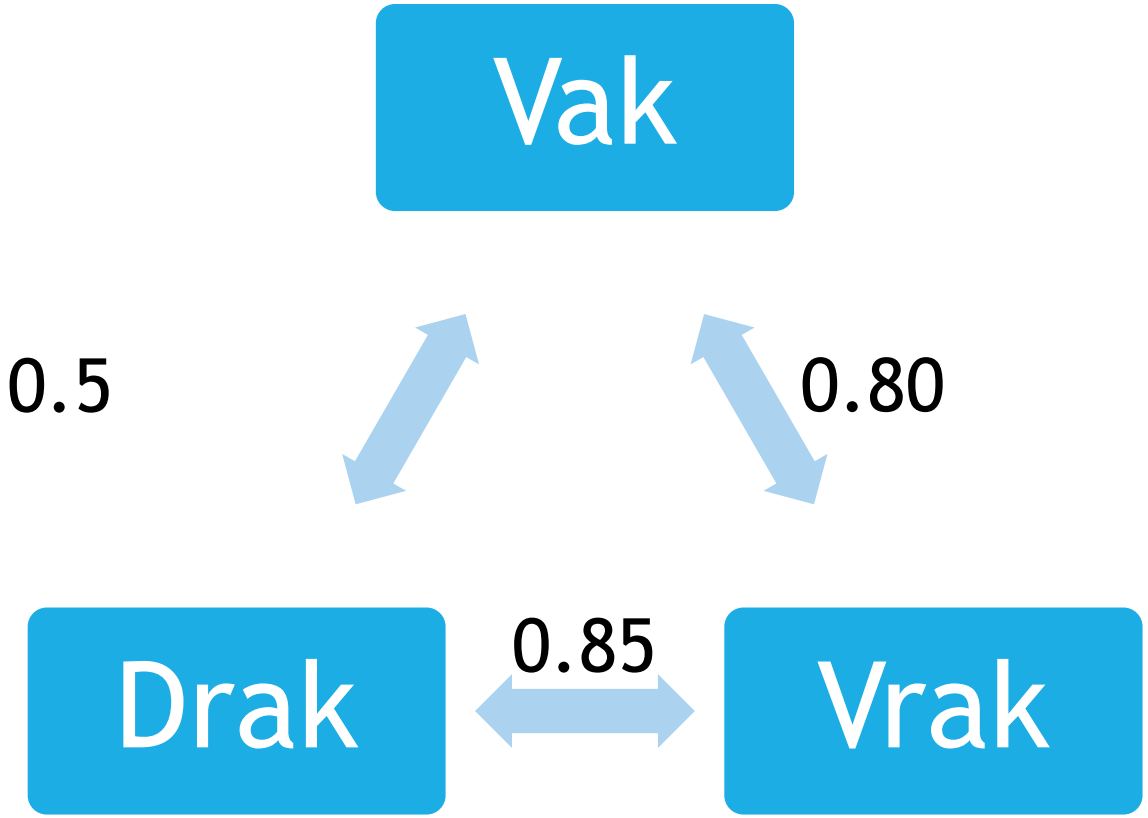
Tolerancia

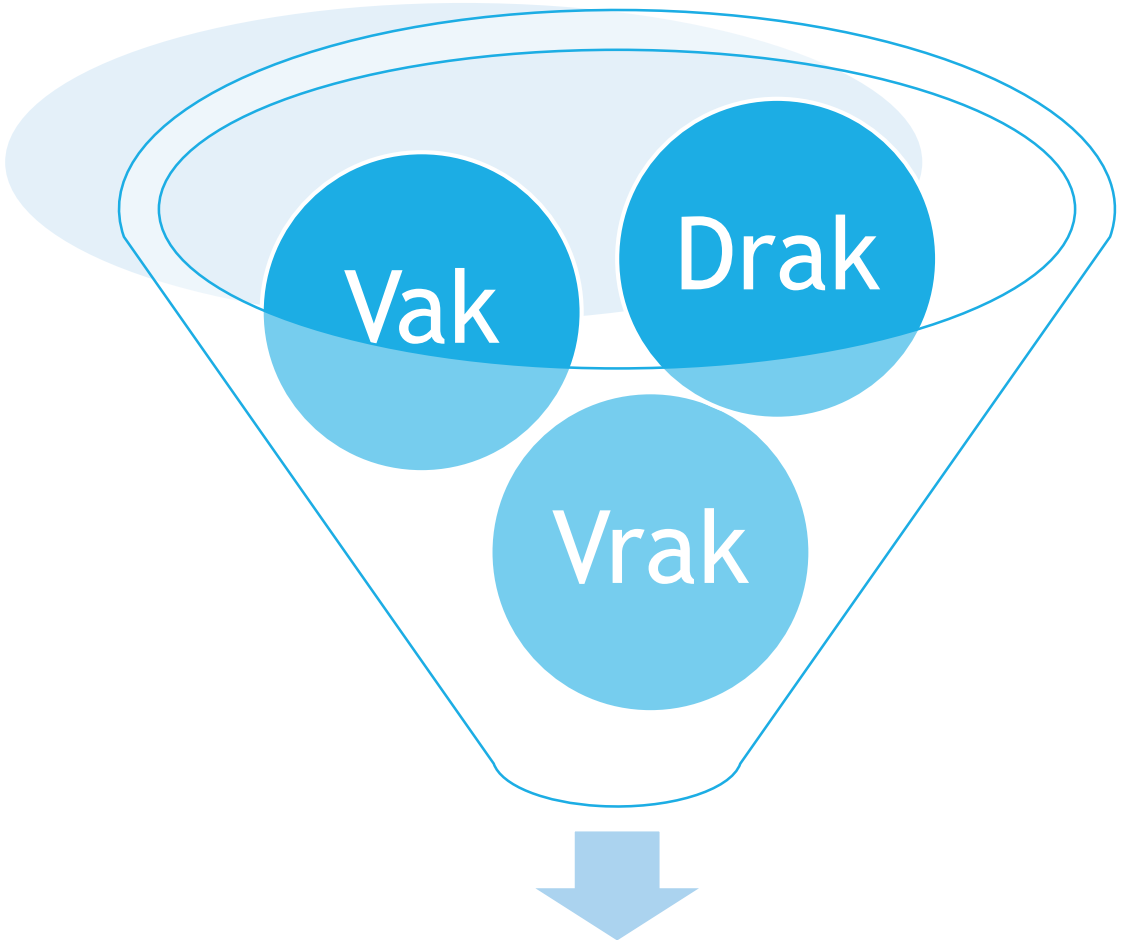
```
public enum Tolerance {
    NONE, LOW, MEDIUM, HIGH
}

public interface IToleranceCalculator {

    Double calculateTolerance(Tolerance tolerance,
                              TermComparisonAlgorithm termComparisonAlgorithm);
}
```

Term grouping

- TermComparisonAlgorithm = naivný
- Tolerance = MEDIUM = 0.8

Term grouping





TermGroupXYZ

Term grouping

- Teda {Vak, Drak, Vrak} označíme ako triedu ekvivalencie pri tolerancii = 0.8 pri naivnom porovnávacom algoritme

Rôzne prístupy a taktiky

```
public enum TermGroupMatchingStrategy {
    MATCH_ALL, MATCH_ONE
}

public enum TermGroupMergingStrategy {
    MERGE_ANY, MERGE_ALL
}
```

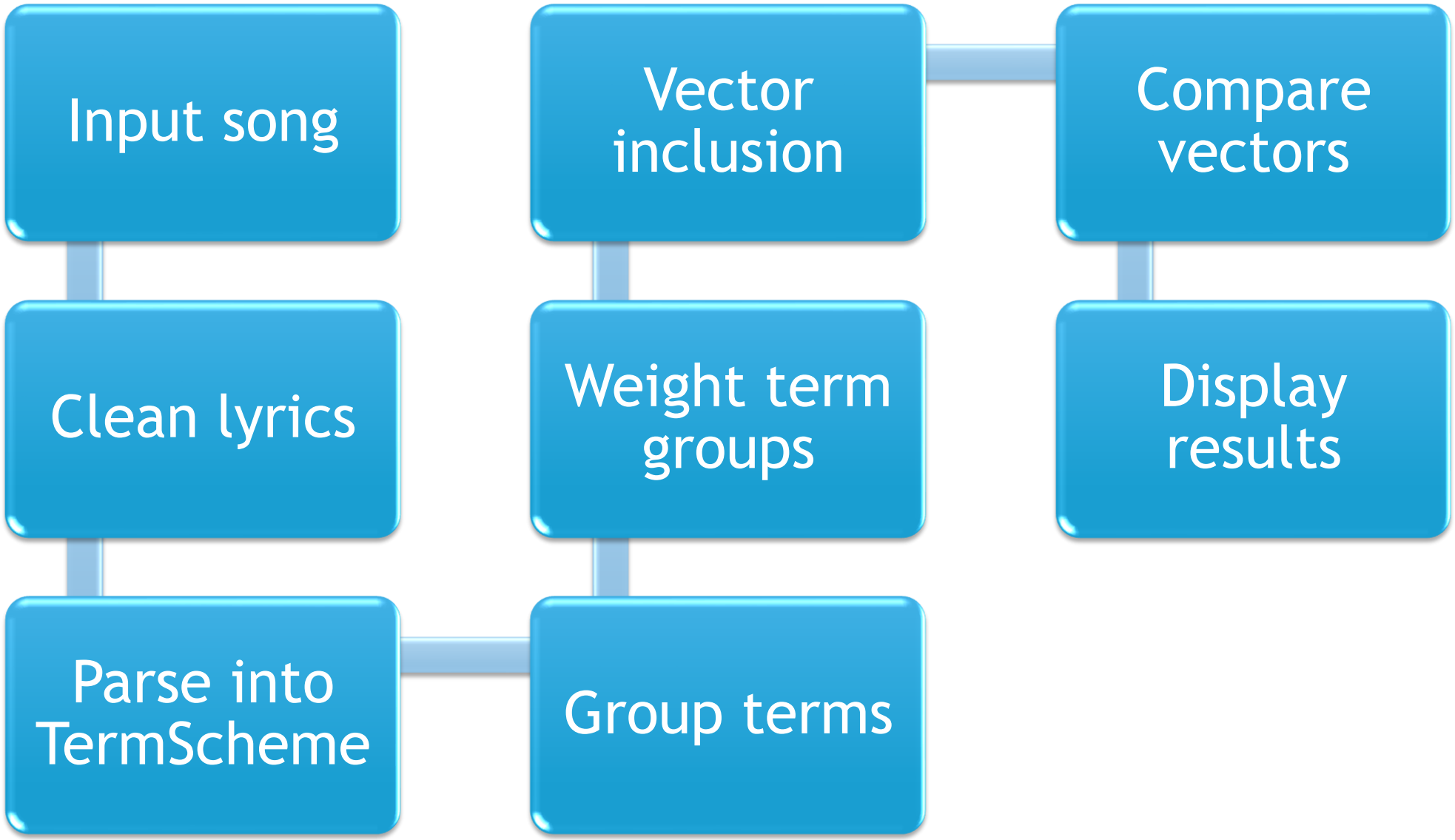
Vector comparison algorithm

$\lvert Q \cap D \rvert$	Simple matching (coordination level match)
$2\frac{\lvert Q \cap D \rvert}{\lvert Q \rvert + \lvert D \rvert}$	Dice's Coefficient
$\frac{\lvert Q \cap D \rvert}{\lvert Q \cup D \rvert}$	Jaccard's Coefficient
$\frac{\lvert Q \cap D \rvert}{\lvert Q \rvert^{1/2} \times \lvert D \rvert^{1/2}}$	Cosine Coefficient (what we studied)
$\frac{\lvert Q \cap D \rvert}{\min(\lvert Q \rvert, \lvert D \rvert)}$	Overlap Coefficient

Zdroj: https://courses.cs.washington.edu/courses/cse573/12sp/lectures/17-ir.pdf

Vector algorithm configuration

- Objekt, ktorá definuje každú fázu algoritmu na základe enum konštanty


Input song
Clean lyrics
Parse into TermScheme
Group terms
Weight term groups
Vector inclusion
Compare vectors
Display results

```java
/**
 * N-gram...
 */
private TermScheme termScheme;
private Integer termDimension;

/**
 * TF, IDF, TF-IDF
 */
private TermWeightType termWeightType;

/**
 * NAIVE, LEVENSHTEIN_DISTANCE
 */
private TermComparisonAlgorithm termComparisonAlgorithm;
private Tolerance tolerance;

private TermGroupMatchingStrategy termGroupMatchingStrategy;
private TermGroupMergingStrategy termGroupMergingStrategy;

/**
 * A, B, INTERSECTION, UNIFICATION, ALL
 */
private VectorInclusion vectorInclusion;

/**
 * SIMPLE_MATCHING, COS_COEFFICIENT, DICE'S_COEFICIENT, JACCARD'S_COEFFICIENT...
 */
private VectorComparisonAlgorithm vectorComparisonAlgorithm;
```

Plány do budúcna

- Štatistické vyhodnotenie výsledkov behu pri rôznych konfiguráciách
- „Mergovací“ systém
- API jednoducho využiteľné v iných aplikáciách
  - Prepojenie so spevníkom pre Android

Zdroje

- Texty ľudových piesní
  - Rôzne stránky
  - Kontakty u nadšencov
- https://en.wikipedia.org/wiki/N-gram
- https://en.wikipedia.org/wiki/Levenshtein_distance
- https://courses.cs.washington.edu/courses/cse573/12sp/lectures/17-ir.pdf
- https://nlp.stanford.edu/IR-book/

Ďakujem za pozornosť

Zemplín