Przetwarzanie języka naturalnego

Projekt: Wyszukiwarka semantyczna

- Karol Nowacki - D/125744
- Repozytorium projektu: https://bitbucket.org/karolnowacki/pjn/

Opis projektu

Celem projektu było stworzenie wyszukiwarki semantycznej dla Wikipedii. Wyszukiwarka pobiera strony z angielskiej Wikipedii z podanej kategorii. Pobrane strony służą do jako korpus do stworzenia wektorów TFIDF, następnie przy pomocy analiza głównych składowych (PCA) dokonywana jest dekompozycja tych wektorów. Wyszukiwana fraza podobnie jak korpus zamieniana jest na wektory następnie na składowe. Wyszukiwanie następuje przez obliczenie podobieństwa cosinusowego składowych wyszukiwanej frazy z każdym artykułem, następnie poprzez wybranie najbardziej podobnych stron.

Uruchomienie projektu

Wymagane moduły języka Python: wikipedia-api pandas nltk tkinter

Uruchomienie projektu:

```
$ python3 main.py
```

Implementacja

Pobieranie stron z Wikipedii

Do pobierania stron z wikipedii użyłem Wikipedia-API

```
    def learnCategory(self, category, depth = 0, max_comp = 50):
        cat = self.wiki.page(category)
        self.documents = pd.DataFrame(columns=['url', 'summary', 'title'])
        self.documents.set_index('url')
        self.loadCategoryPages(cat.categorymembers, 0, depth)

    def loadCategoryPages(self, categorymembers, level=0, max_level=1):
        for c in categorymembers.values():
            if c.ns == wikipediaapi.Namespace.CATEGORY and level < max_level:
                self.loadCategoryPages(c.categorymembers, level=level + 1, max_level=max_level)
            if c.ns == wikipediaapi.Namespace.MAIN:
                self.addPage(key = c.pageid, url = c.fullurl, title = c.title, summary = c.summary)
```

Tworzenie wektorów TFIDF i dekompozycja

Do stworzenia wektorów TFIDF użyłem wektorizera TfidfVectorizer z biblioteki scikit-learn. Do dekompozycji używam PCA z tej samej biblioteki

```
    def fit(self, max_comp = 50):
        nltk.download('stopwords')
        self.vectorizer = TfidfVectorizer(stop_words = nltk.corpus.stopwords.words("english"))

        self.pca = PCA(n_components=min(max_comp, self.count()))
        self.pca_topics = pd.DataFrame(
            self.pca.fit_transform(self.vectorizer.fit_transform(self.getCorpus()).toarray()))
```

Wyszukiwanie

Wyszukiwanie to wykonanie następujących kroków: zamiana zapytania na wektory TFIDF, dekompozycja, porównywanie cosinusowe i zwrócenie najlepszych wyników.

```
def query(self, que, min_score = 0.5):
    Y = self.pca.transform(self.vectorizer.transform([que]).toarray())

    Z = cosine_similarity(self.pca_topics, Y)
    result = pd.DataFrame(self.documents)
    result['score'] = Z
    result = result.query("`score` >= {:.2f}".format(min_score)).nlargest(n=10, columns="score")
    return result
```

Przykład użycia

Pobranie stron, stworzeni wektorów i dekompozycja

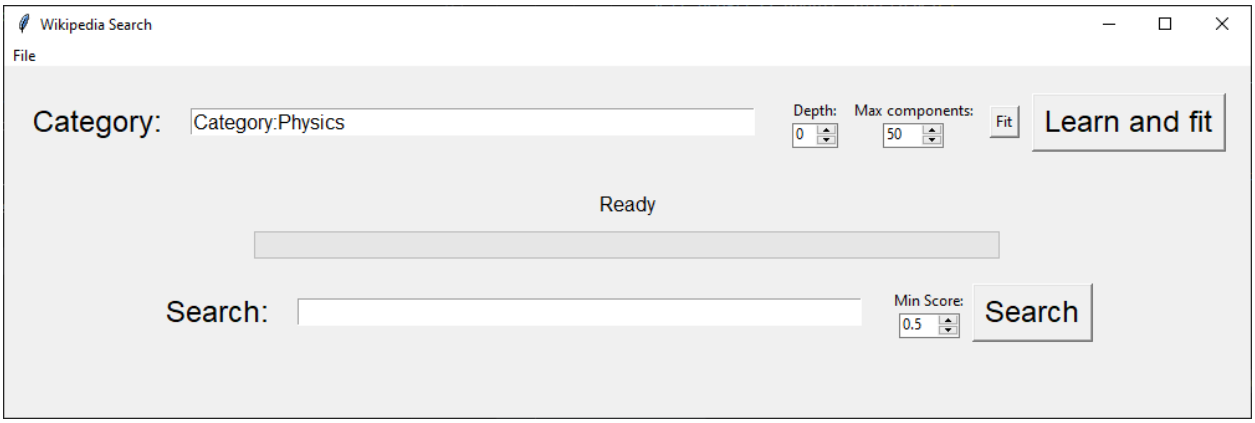
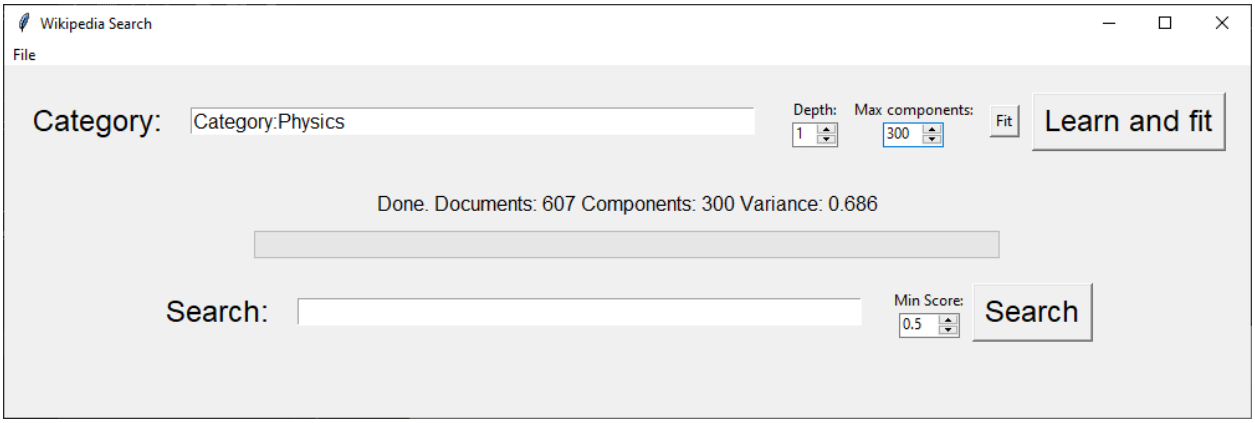


Figure 1: snap1

- W polu `Category` podajemy nazwę kategorii z angielskiej Wikipedii
- W polu `Depth` podajemy jak głęboko w podkategorie chcemy wchodzić
- W polu `Max components` podajemy na ile maksymalnie komponentów chcemy dokonać dekompozycji
- Przycisk `Learn and fit` rozpoczyna proces



Po skończonym procesie pobierania w otrzymujemy informację ile stron zostało pobrany, liczbę komponentów oraz sumę wariancji poszczególnych komponentów. Determinuje to procent zachowanych informacji po procesie dekompozycji. W tym przypadku 300 komponentów z 607 artykułów daje zachowuje 69% informacji. Tak wyuczoną bazę możemy zapisać z menu `File`.

Wyszukiwanie

W celu wyszukania strony w polu `Search` podajemy poszukiwaną frazę. Parametr `Min Score` określa minimalne podobieństwo artykułów jakich oczekujemy w wynikach. Przycisk `Search` rozpoczyna wyszukiwanie.

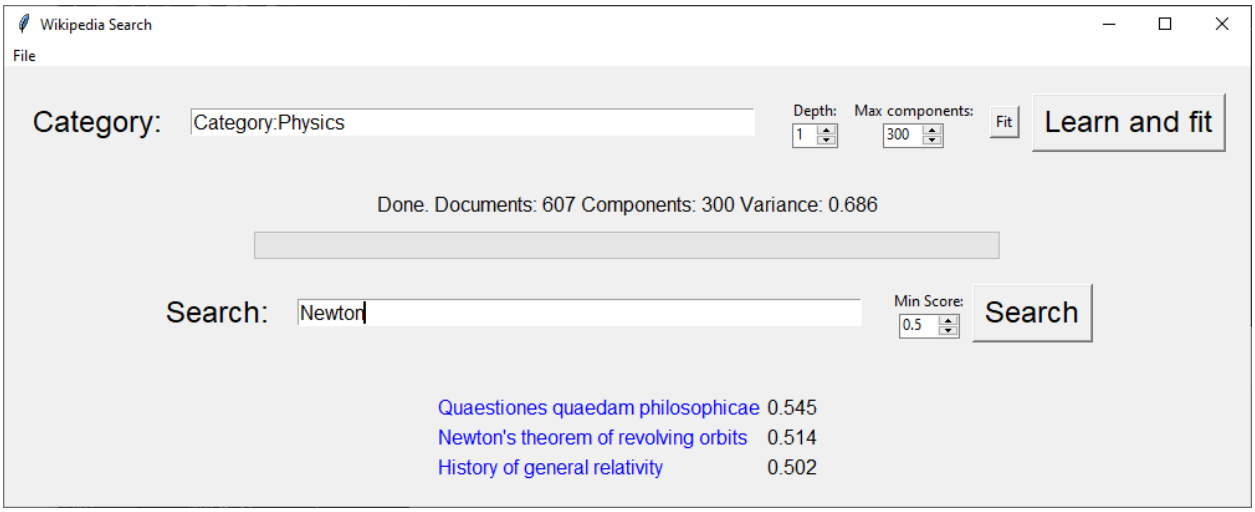


Figure 2: snap3

10 najlepszych wyników prezentowanych jest w kolejności malejącego podobieństwa. Kliknięcie w nazwę strony otwiera przeglądarkę z artykułem.