Лабораторна робота №3

Моделі текстових даних

Мета роботи: Ознайомитись з основними текстовими моделями та їх створення за допомогою бібліотек scikit-learn та genism.

Короткі теоретичні відомості

Попередня підготовка текстових даних очищає їх від шуму та дає краще зрозуміти структуру тексту. Для подальшого аналізу тексту часто використовують машинне навчання.

Усі моделі машинного або глибокого навчання обмежені, оскільки вони не можуть безпосередньо розуміти текстові дані та розуміють лише числові представлення ознак як вхідні дані.

Просторова векторна модель є корисною концепцією при роботі з текстовими даними та дуже популярною у задачах пошуку інформації та ранжуванні документів. Просторова векторна модель також називається векторною моделлю термів і визначається як математична та алгебраїчна модель для перетворення та представлення текстових документів як числових векторів конкретних термів, які утворюють векторні виміри.

Традиційні (на основі кількості) стратегії розробки ознак для текстових даних належать до сімейства моделей, широко відомих як модель «Сумка слів». Сюди входять частоти термів, TF-IDF (частота терму-обернена частота документа), N-грами, тематичні моделі тощо. Незважаючи на те, що вони є ефективними методами для вилучення ознак із тексту, вони містять не всю інформацію. Оскільки текст в процесі перетворюється на набір слів, то втрачається семантика, структура, послідовність і контекст навколо сусідніх слів у кожному текстовому документі.

Модель «Сумка слів» (Bag of Words)

Модель «Сумка слів» представляє кожен текстовий документ як числовий вектор, де кожен вимір є окремим словом із корпусу, а значенням може бути його частота в документі, наявність (позначена 1 або 0) або навіть зважені значення.

Для підготовки тексту, а також подальшого машинного навчання можна використати бібліотеку scikit-learn. Вона також містить модуль для розробки ознак, зокрема з тексту.

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)

cv_matrix = cv.fit_transform(p_corpus)
```

CountVectorizer() перетворює колекцію текстових документів на матрицю кількості токенів. Створює розріджене представлення кількості за допомогою scipy.sparse.csr_matrix. Параметри min_df та max_df задають мінімальну та максимальну частоту термів.

Модель «Сумка n-грам» (Bag of N-Grams)

Коли потрібно все ж таки враховувати порядок слів, модель «Сумка слів» розширюється до моделі «Сумка n-грам» , яка використовує ознаки на основі n-грам.

Наприклад, використаємо біграми, тобто сполучення двох слів.

CountVectorizer має параметр ngram_range, що задає діапазон слів у виділених словосполученнях.

bv = CountVectorizer(ngram_range=(2,2))

bv_matrix = bv.fit_transform(p_corpus)

Модель TF-IDF

Модель «Сумка слів» має деякі недоліки при використанні у великих корпусах. Оскільки вектори ознак базуються на абсолютних частотах термів, деякі терми можуть часто зустрічатися в усіх документах, і вони можуть затьмарювати інші терми в наборі ознак. Особливо слова, які зустрічаються не так часто, але можуть бути більш цікавими та ефективними як ознаки для ідентифікації конкретних категорій.

TF-IDF означає частота терму-обернена частота документа.

Бібліотека scikit-learn має TfidfTransformer(), що перетворює матрицю з частотою термів на матрицю tfidf.

from sklearn.feature_extraction.text import TfidfTransformer

tt = TfidfTransformer(norm='l2', use_idf=True)

tt_matrix = tt.fit_transform(cv_matrix)

TfidfVectorizer з бібліотеки scikit-learn дозволяє безпосередньо обчислювати вектори tfidf, він приймає необроблені документи як вхідні дані та обчислює частоти термів, а також обернені частоти документів.

from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., norm='l2',

use_idf=True, smooth_idf=True)

tv_matrix = tv.fit_transform(p_corpus)

Подібність документів – це процес використання метрики на основі відстані чи подібності, який може визначити, наскільки текстовий документ схожий на будь-який інший документ на основі ознак, отриманих із документів.

Таким чином, можна створювати ознаки на основі TF-IDF і використовувати їх для створення нових ознак. Попарна подібність документів у корпусі передбачає обчислення подібності документів для кожної пари документів у корпусі. Таким чином, якщо є C документів у корпусі, то результатом буде матриця CxC, така що кожен рядок і стовпець представляють оцінку подібності для пари документів.

Існує кілька показників подібності та відстані, які використовуються для обчислення подібності документів. До них відносяться косинусна відстань, евклідова відстань, манхеттенська відстань тощо.

Косинусна подібність дає метрику, що представляє косинус кута між представленнями вектора ознак двох текстових документів. Чим менший кут між документами, тим вони ближчі й схожі.

Її можна порахувати за допомогою бібліотеки scikit-learn.

```
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrix = cosine_similarity(tv_matrix)
```

Як вхід використовується tfidf матриця.

Також можна отримати подібності, розраховані для інших типів відстаней, оскільки подібність дорівнює 1 відняти відстань.

```
from sklearn.metrics import pairwise_distances
similarity_matrix =1-pairwise_distances(tv_matrix, metric='euclidean')
```

Розраховану подібність документів можна використати для поділу цих документів на групи, тобто кластеризації. Кластеризація ділить об’єкти на кластери на основі відстані між цими об’єктами.

На основі вже готової матриці подібності документів будується матриця зв’язку

```
from scipy.cluster.hierarchy import dendrogram, linkage
links = linkage(similarity_matrix, 'ward')
```

Зв’язки можна візуалізувати за допомогою дендрограми:

```
plt.figure(figsize=(8, 3))
plt.title('Дендрограма')
plt.xlabel('Документи')
```

plt.ylabel('Відстань')

dendrogram(links)

З дендрограми видно, на якій приблизно відстані зливаються кластери, наприклад 2, тож її можна використати для отримання міток кластерів:

from scipy.cluster.hierarchy import fcluster

max_dist = 2

cluster_labels = fcluster(links, max_dist, criterion='distance')

Бібліотека scikit-learn також має клас для агломеративної кластеризації:

from sklearn.cluster import AgglomerativeClustering

ag=AgglomerativeClustering(n_clusters=3,metric='euclidean',linkage='ward')

ag.fit(tv_matrix)

ag.labels_

Моделі, що використовують розрахунок частот і базуються на моделі «Сумка слів», не враховують контекст. Тому є сенс розглянути більш складні моделі, які можуть фіксувати цю інформацію та давати ознаки, які є векторним представленням слів. Таке представлення називається вбудовуванням.

Потрібно використовувати моделі векторного простору таким чином, щоб вбудовувати вектори слів у цей неперервний векторний простір на основі семантичної та контекстної подібності.

Модель Word2Vec

Ця модель була створена компанією Google у 2013 році та є прогностичною моделлю на основі глибокого навчання для обчислення та генерування високоякісних, розподілених і неперервних щільних векторних представлень слів, які фіксують контекстну та семантичну подібність. По суті, це моделі з навчанням без учителя, які можуть приймати масивні текстові корпуси, створювати словник можливих слів і генерувати щільні вбудовування слів для кожного слова у векторному просторі, що представляє цей словник.

Існує дві різні архітектури моделей, які Word2Vec може використовувати для створення цих представлень вбудовування слів.

Архітектура моделі неперервна сумка слів (CBOW) намагається передбачити поточне цільове слово (центральне слово, target_word) на основі слів вихідного контексту (навколишніх слів, context_window).

Архітектура моделі Skip-Gram намагається досягти зворотного до того, що робить модель CBOW. Він намагається передбачити вихідні контекстні слова (навколишні слова) з урахуванням цільового слова (центрального слова).

from gensim.models import word2vec

wpt = nltk.WordPunctTokenizer()

tokenized_corpus = [wpt.tokenize(document) for document in emma]

feature_size = 100

window_context = 30

min_word_count = 1

sample = 1e-3

w2v_model = word2vec.Word2Vec(tokenized_corpus, vector_size=feature_size,window=window_context, min_count=min_word_count,sample=sample)

Модель FastText була представлена Facebook у 2016 році як розширення та вдосконалення моделі Word2Vec.

Загалом, прогнозуючі моделі, такі як Word2Vec, розглядають кожне слово як окрему сутність і генерують щільне вбудовування для слова. Однак це серйозне обмеження для мов, які мають величезний словниковий запас і багато рідкісних слів. Модель Word2Vec зазвичай ігнорує морфологічну структуру кожного слова та розглядає слово як єдине ціле. Модель FastText розглядає кожне слово як сумку n-грам символів.

Gensim також має реалізацію цієї моделі:

from gensim.models.fasttext import FastText

wpt = nltk.WordPunctTokenizer()

tokenized_corpus = [wpt.tokenize(document) for document in emma]

ft_model = FastText(tokenized_corpus, vector_size=100, window=50, min_count=5,sample=1e-3, sg=1)

Подібно до Word2Vec, Doc2Vec також належить до класу алгоритмів з навчанням без учителя, оскільки дані, які в ньому використовуються, не позначені.

Будуємо модель Doc2Vec та навчаємо її:

model = Doc2Vec(documents, vector_size=5, min_count=1, workers=4,epochs = 40)

model.train(documents, total_examples=model.corpus_count,epochs=model.epochs)

Розмір вектора 5 означає, що кожен документ буде представлено вектором із п’яти значень з плаваючою комою. Параметр min_count встановлює порогове

значення, щоб у словнику розглядалися лише терми, які зустрічаються принаймні min_count кількість разів.

Завдання до лабораторної роботи

Створити програму, яка виконує завдання відповідно до варіанту.

Оформити звіт. Звіт повинен містити:

— титульний лист;
— код програми;
— результати виконання коду;

Продемонструвати роботу програми та відповісти на питання стосовно теоретичних відомостей та роботи програми.