Практическое занятие 2.3.4. Классификация текстовых данных

Предполагается, что вы работаете в том же блокноте, где выполняли задания Части 1.

5. Использование статистической меры для оценки важности слова в контексте документа – `tf-idf`

Определим общую частоту и обратную частоту документа

Нам хотелось бы работать со словами, которые несут наибольший смысл в предложениях / твитах. Но возникает вопрос: а слова, которые встречаются чаще всего, они действительно несут наибольший смысл и важны для классификации и построения предсказательной модели?

1). Посмотрим на слова внутри нашего мешка слов. Выведем 20 самых популярных слов и сколько раз они встречаются:

```
sorted(hash_map.items(), key=lambda item: item[1], reverse=
True)[:20]
```

[**Результат**]:

Как следует из вывода, 20 самых распространенных слов преимущественно не дают никакой информации о твитах. Это остатки URL-адресов твитов, а также некоторые общие слова, часто встречающиеся во всем тексте. Их едва ли можно рассматривать в качестве важных характеристик. Для улучшения нашей модели необходимо сделать нечто большее, чем просто выявить наиболее часто встречающиеся слова. Возможно, следует искать слова, которые часто встречаются в некоторых документах, но не во всех.

Простой и удобный способ оценить важность термина для какого-либо документа относительно всех остальных документов – если слово встречается в каком-либо документе часто, при этом встречаясь редко во всех остальных документах, – это слово имеет большую значимость для того самого документа. Этот способ известен как принцип "общая частота слов (`tf`) – обратная частота документов (`idf`)", или `tfidf`.

Формула `tfidf` имеет вид:

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right),$$

i – это индекс слова, j – индекс документа, N – общее количество документов (токенов), df_i – количество документов, в которых появляется слово с индексом i.

В мешке слов каждая строка – это документ, а каждый столбец – частота слова в этом документе. Это уже часть `'term frequency'` (частоты слова – $tf_{i,j}$) в `tfidf`.

Обратная частота документа

2). Вычислим обратную частоту документа по следующему правилу: для каждого слова подсчитаем количество документов, в которых оно появляется, а затем возьмем логарифм от инверсии этого числа.

Построим вектор `idf`, реализуя следующие два шага:

1. Определим частоту появления для каждого слова.
2. Разделим количество документов (N) на количество документов, в которых встречается слово с индексом `i`, и возьмем логарифм от результата.

```
#инициализируйте 2 переменные, представляющие количество тв
итов (numdocs) и количество токенов/слов (numwords)
numdocs, numwords = np.shape(bag_o)

#Переход к формуле tfidf, как указано выше
N = numdocs
word_frequency = np.empty(numwords)

#Подсчет количества документов, в которых появляется слово
for word in range(numwords):
    word_frequency[word]=np.sum((bag_o[:,word]>0))

idf = np.log(N/word_frequency)
idf.shape
```

[**Результат**]:

```
idf
```

[**Результат**]:

Завершим расчет `tfidf`, умножив наш мешок слов (частоту слов) на `idf`.

```
#инициализирует массив tfidf
tfidf = np.empty([numdocs, numwords])

#циклически перебирает твиты, перемножая частоту появления
слов (представленную мешком слов) с idf
for doc in range(numdocs):
    tfidf[doc, :]=bag_o[doc, :]*idf

tfidf.shape
```

[**Результат**]:

Массив `tfidf` состоит из значений каждого из 500 токенов в 10860 твитах. TF-IDF –универсальная метрика для измерения важности.

Пример–пояснение

Пусть в некотором документе X, содержащем 100 слов, есть слово «интеллект», которое встречается 5 раз. Таким образом, TF слова «интеллект» равняется 5/100 или 0.05. А теперь представим, что всего у нас есть 1000 документов (включая документ X), и слово «интеллект» встречается в 10 из них. Таким образом, IDF слова «интеллект» равняется lg(1000/10) или 2. Таким образом, TF-IDF слова «интеллект» равняется 2 * 0.05 или 0.1.

Вместо документов у нас могут быть какие-нибудь абстрактные категории или конкретные ящики, а вместо слов – абстрактные объекты или конкретные фрукты:

3). Теперь у нас есть массив `tfidf`. Начнем обучать нашу модель и делать прогнозы. *Обучаться*, или, как говорят, *строить модель*, мы будем на *обучающей выборке*, а проверять качество построенной модели – на *тестовой*.

Подключим библиотеку scikit-learn, которая содержит ряд моделей машинного обучения.

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split #для р
азделения данных на обучающий и тестовый наборы
from sklearn.model_selection import GridSearchCV #чтобы узн
ать лучший параметр для нашей модели
```

Пояснения:

– Метод опорных векторов (Support Vector Machines – SVM**)** – это набор методов обучения, используемых для классификации, регрессии и обнаружения выбросов.

– GridSearchCV – состоит из двух частей: GridSearch и CV – поиск по сетке и перекрестная проверка.
В моделе машинного обучения параметры, которые необходимо найти, называются гиперпараметрами. Для выполнения этого поиска можно использовать Scikit-Learn GridSearchCV.
– Модуль train_test_split разбивает датасет на данные для обучения и тестирования. Разбиение на тестовую и обучающую выборку должно быть случайным. Обычно используют разбиения в пропорции 50% : 50%, 60% : 40%, 75% : 25% и т.д. Мы воспользуемся функцией train_test_split и разобьем данные на обучающую/тестовую выборки в отношении 75% : 25%.

X_train, y_train – это обучающая выборка; X_test, y_test – тестовая выборка.

```
# разбиваем X_all и y_all на обучающие и тестовые наборы
X_train, X_test, y_train,
y_test = train_test_split(tfidf,df['relevance'].values,shuf
fle=True)

# выведем количественное соотношение обучающей и тестовой
выборок
N_train, _ = X_train.shape
N_test,  _ = X_test.shape
print(N_train, N_test)
```

[**Результат**]: **Оцените, в какой пропорции разбиты данные на тестовую и обучающую выборки. Ответ вставьте в виде комментария.**

```
print (tfidf)
print (df['relevance'])
```
[**Результат**]:

```
print(X_train.shape, X_test.shape)
```
[**Результат**]:

```
print(y_train.shape, y_test.shape)
```
[**Результат**]:

```
# Создаем экземпляр модели
model = SVC()
# обучаем модель на тренировочном наборе - обучение с
учителем
model.fit(X_train,y_train)
```
[**Результат**]:

```
y_pred = model.predict(X_test) # предсказать значение
целевой переменной
```

```
print(y_pred)
```
[**Результат**]:

```
# Используем метод score для получения точности модели
score = model.score(X_test, y_test)
print(score)
```
[**Результат**]:

```
print('Точность классификатора логистической регрессии на т
естовом наборе: {:.3f}'.format(score))
```

[**Результат**]:

```
score = model.score(X_train,y_train)
print(score)
```
[**Результат**]:

```
print('Точность классификатора логистической регрессии на о
бучающем наборе: {:.3f}'.format(score))
```

[**Результат**]:

Оформите результаты выполнения заданий Части 1 и Части 2 практического занятия 2.3.4. Классификация текстовых данных в одном файле или одном блокноте в google.Colab!!!