Практическое занятие 2.3.4. Классификация текстовых данных

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

Определим общую частоту и обратную частоту документа

Вообще говоря, нам бы хотелось 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?

Давайте 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
]
```

```
[('co', 6800),
 ('http', 6154),
 ('https', 618),
 ('û_', 514),
 ('amp', 510),
 ('like', 492),
 ('fire', 365),
 ('get', 336),
 ('new', 329),
 ('via', 325),
 ('2', 310),
 ('news', 288),
 ('people', 283),
 ('emergency', 229),
 ('video', 227),
 ('disaster', 220),
 ('would', 214),
 ('3', 202),
 ('police', 199),
```

```
 ('still', 180)]
```

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 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. Как вы думаете, почему это имеет смысл?

Интуитивный вывод известен как "общая частота слов (`tf`) – обратная частота документов (`idf`)", или `tfidf`. Это простой и удобный способ оценить важность термина для какого-либо документа относительно всех остальных документов. Принцип такой – 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.

Формула `tfidf` имеет вид:

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

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

В нашем 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

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 твитов (num
docs) 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
```

```
[]: (500,)
```

```
idf
```

```
[]:
array([0.64339282, 0.74842242, 2.87774463, 3.07225142, 3.18581871,
       3.15295704, 3.47871106, 3.50288142, 3.5245206 , 3.51829005,
       3.65448692, 3.67244073, 3.6760705 , 3.88566982, 3.91294424,
       4.02498343, 3.9601228 , 4.05109458, 4.08883491, 4.1223576 ,
       4.1223576 , 4.18085381, 4.17484778, 4.14534712, 4.18085381,
       4.18689612, 4.19297517, 4.19909139, 4.25588899, 4.20524526,
       4.2822063 , 4.27556176, 4.26240367, 4.24298559, 4.28889529,
       4.24941648, 4.26240367, 4.3440817 , 4.28889529, 4.30240901,
       4.31610785, 4.32302829, 4.32302829, 4.33701454, 4.31610785,
       4.38756682, 4.37286067, 4.44081133, 4.32999696, 4.35119917,
       4.41003967, 4.44865451, 4.40249247, 4.38756682, 4.41764427,
       4.41764427, 4.50534985, 4.45655969, 4.46452786, 4.52215697,
       4.47256003, 4.46452786, 4.47256003, 4.5137181 , 4.46452786,
       4.50534985, 4.53066766, 4.50534985, 4.53066766, 5.07333389,
       4.5392514 , 4.56545377, 4.54790947, 4.59236123, 4.61071037,
       4.56545377, 4.91081496, 4.62001276, 4.60149371, 4.62001276,
       4.61071037, 4.6294025 , 4.6294025 , 4.6294025 , 4.64845069,
. . . . . .
       5.50865196, 5.4861791 , 5.4861791 , 5.4861791 , 5.57926953,
       5.55517198, 5.53164148, 5.53164148, 5.55517198, 5.53164148,
       5.55517198, 5.57926953, 5.50865196, 5.50865196, 5.55517198,
       5.50865196, 5.50865196, 5.53164148, 5.50865196, 5.53164148,
       5.53164148, 5.55517198, 5.55517198, 5.55517198, 5.70932266,
       5.60396214, 5.53164148, 5.53164148, 5.53164148, 5.53164148])
```

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 (пре
дставленную мешком слов) с idf
for doc in range(numdocs):
    tfidf[doc, :]=bag_o[doc, :]*idf

print (tfidf)
```

```
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
```

```
[0.         0.         0.         ... 0.         0.         0.        ]
...
[0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
[0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
[0.         0.         0.         ... 0.         0.         0.        ]]
```

Как бы вы описали массив `tfidf`? Он состоит из значений `tfidf` каждого из 500 токенов в 10860 твитах.

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: TF-IDF –универсальная метрика для измерения важности.

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

Будем использовать библиотеку scikit-learn, которая предоставляет ряд моделей машинного обучения.

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split #для разделения
 данных на обучающий и тестовый наборы
from sklearn.model_selection import GridSearchCV #чтобы узнать лучши
й 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,shuffle=True)

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

```
[]: 8145 2715 - 75% : 25%.
```

```
print (tfidf)
print (df['relevance'])
```

```
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
 [0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
0        1
1        1
2        1
3        1
4        1
       ..
10871    1
10872    1
10873    1
10874    1
10875    1
Name: relevance, Length: 10860, dtype: int64
```

```
print(X_train.shape, X_test.shape)

print(y_train.shape, y_test.shape)
```

```
(8145,) (2715,)
```

```
# Создаем экземпляр модели логистической регрессии для
# прогнозирования результатов столбца 'choose_one' тестового набора
model = SVC()
# обучаем модель на тренировочном наборе - обучение с учителем
model.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

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

```
[1 0 0 ... 1 0 0]
```

```
# Используем метод score для получения точности модели
score = model.score(X_test, y_test)
print(score)

print('Точность классификатора логистической регрессии на тестовом н
аборе: {:.3f}'.format(score))
```

```
0.8007366482504604
Точность классификатора логистической регрессии на тестовом наборе: 0.801
```

```
score = model.score(X_train,y_train)
print(score)

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

```
0.8788213627992634
Точность классификатора логистической регрессии на обучающем наборе: 0.879
```