8.3.1. Word2vec-признаки

Проект word2vec предложила компания Google как относительно новый подход к обработке естественного языка. Сам инструмент word2vec представляет собой модель с машинным обучением, построенную на базе глубоких нейронных сетей — раздела ML, в котором в последнее время получают потрясающие результаты, особенно в областях, связанных с деятельностью человека, например с естественным языком, речью и изображениями.

Для построения word2vec-модели на базе нашей обучающей выборки воспользуемся библиотекой Gensim языка Python, в которую встроена прекрасная реализация word2vec. Мы уже прибегали к ней в главе 7 при изучении латентного размещения Дирихле — другой тематической модели, сходной с word2vec.

Нам потребуется подготовить документ к моделированию, так как Gensim-алгоритмы работают с предложениями (списками слов), а не с произвольными документами. Эта дополнительная работа заодно позволяет посмотреть, что именно попадает в модель. Листинг 8.7 демонстрирует простую функцию токенизации, которая удаляет шумовые слова и знаки препинания, а также преобразует все символы в нижний регистр. Разумеется, все эти операции автоматически выполняются векторизаторами слов из библиотеки scikit-learn; сходную функциональность можно было получить, воспользовавшись инструментарием NLTK для языка Python, но для наглядности мы решили написать функцию с нуля.

Листинг 8.7. Токенизация документа

```
import re, string

stop_words = set(['all', "she'll", "don't", 'being', 'over', 'through',
'yourselves', 'its', 'before', "he's", "when's", "we've", 'had', 'should',
"he'd", 'to', 'only', "there's", 'those', 'under', 'ours', 'has',
"haven't", 'do', 'them', 'his', "they'll", 'very', "who's", "they'd",
'cannot', "you've", 'they', 'not', 'during', 'yourself', 'him', 'nor',
"we'll", 'did', "they've", 'this', 'she', 'each', "won't", 'where',
"mustn't", "isn't", "i'll", "why's", 'because', "you'd", 'doing', 'some',
'up', 'are', 'further', 'ourselves', 'out', 'what', 'for', 'while',
"wasn't", 'does', "shouldn't", 'above', 'between', 'be', 'we', 'who',
"you're", 'were', 'here', 'hers', "aren't", 'by', 'both', 'about', 'would',
```

```
'of', 'could', 'against', "i'd", "weren't", "i'm", 'or', "can't", 'own',
'into', 'whom', 'down', "hadn't", "couldn't", 'your', "doesn't", 'from',
"how's", 'her', 'their', "it's", 'there', 'been', 'why', 'few', 'too',
'themselves', 'was', 'until', 'more', 'himself', "where's", "i've", 'with',
"didn't", "what's", 'but', 'herself', 'than', "here's", 'he', 'me',
"they're", 'myself', 'these', "hasn't", 'below', 'ought', 'theirs', 'my',
"wouldn't", "we'd", 'and', 'then', 'is', 'am', 'it', 'an', 'as', 'itself',
'at', 'have', 'in', 'any', 'if', 'again', 'no', 'that', 'when', 'same',
'how', 'other', 'which', 'you', "shan't", 'our', 'after', "let's", 'most',
'such', 'on', "he'll", 'a', 'off', 'i', "she'd", 'yours', "you'll", 'so',
"we're", "she's", 'the', "that's", 'having', 'once'])

def tokenize(docs):
    pattern = re.compile('[\W_]+', re.UNICODE)
    sentences = []
    for d in docs:
        sentence = d.lower().split(" ")
        sentence = [pattern.sub('', w) for w in sentence]
        sentences.append( [w for w in sentence if w not in stop_words] )
return sentences
```

После преобразования всех символов в нижний регистр разбивает документ на слова

Удаляет все символы, не являющиеся словами, например знаки препинания

Удаляет английские стоп-слова

С этой функцией в токены можно превратить любой список документов, поэтому перейдем к построению нашей первой word2vec-модели. Дополнительную информацию о параметрах используемого алгоритма можно найти в документации на сайте библиотеки Gensim[1].

Листинг 8.8. Word2vec-модель

```
from gensim.models.word2vec import Word2Vec

sentences = tokenize(d_train.review)
model = Word2Vec(sentences, size=300, window=10, min_count=1,
    sample=1e-3, workers=2)
model.init_sims(replace=True)
print model['movie']
#> array([ 0.00794919, 0.01277687, -0.04736909, -0.02222243, …])
```

Генерирует предложения с помощью функции токенизации

Строит и нормализует word2vec-модель

Выводит вектор word2vec-модели для слова «movie»

Вы видите, каким образом одно слово преобразуется в вектор (в рассматриваемом случае он состоит из 300 чисел). Чтобы word2vec-модель смогла сгенерировать признаки для нашего ML-алгоритма, обзоры сле-

[1] *https://radimrehurek.com/gensim/models/word2vec.html*

дует превратить в векторы признаков. Вы уже умеете представлять в векторном виде отдельные слова, а в данном случае требуется представить документ обзора (список слов) как средний вектор всех составляющих его слов. Следующий листинг демонстрирует построение решающей эту задачу функции.

Листинг 8.9. Генерация Word2vec-признаков

```
def featurize_w2v(model, sentences):
    f = zeros((len(sentences), model.vector_size))
    for i,s in enumerate(sentences):
        for w in s:
            try:
                vec = model[w]
            except KeyError:
                continue
            f[i,:] = f[i,:] + vec
        f[i,:] = f[i,:] / len(s)
    return f
```

Инициализирует массив NumPy для векторов признаков

Циклически просматривает каждое предложение, добавляет векторы для каждого слова и берет среднее

Все готово для построения модели на основе сгенерированных word2vec-признаков. Возможно, из нашего обсуждения в разделе 8.2.2 вы помните, что наивный байесовский классификатор хорошо работает с разреженными данными, но не очень подходит для плотных данных. В результате преобразования мы перешли от ~65 000 разреженных признаков, полученных путем подсчета слов, к сотням плотных word2vec-признаков. Модель глубокого обучения изучила темы высокого уровня (листинг 8.8), и теперь каждый документ можно представить как комбинацию тем (листинг 8.9).

8.3.2. Модель на базе алгоритма «случайный лес»

Наивный байесовский алгоритм, с которым мы работали в предыдущих разделах, несовместим с новыми word2vec-признаками, так как их нельзя представить как порождения полиномиального распределения. Можно поменять распределение и продолжить работу с наивным байесовским алгоритмом, но вместо этого мы обратимся к старому знакомому — алгоритму «случайный лес». В следующем листинге строится модель «случайного леса» из 100 деревьев на базе word2vec-признаков и, как обычно, анализируется ее производительность на тестовой выборке.

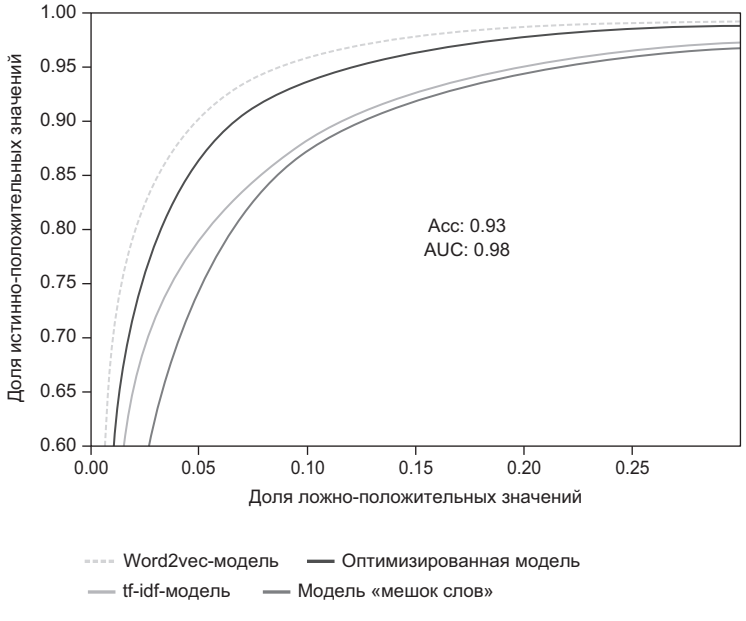
Листинг 8.10. Построение модели на базе алгоритма «случайный лес» и word2vec-признаков

```
features_w2v = featurize_w2v(model, sentences)

model4 = RandomForestClassifier(n_estimators=100, n_jobs=-1)
model4.fit(features_w2v, d_train.sentiment)

test_sentences = tokenize(d_test.review)
test_features_w2v = featurize_w2v(model, test_sentences)
pred4 = model4.predict_proba(test_features_w2v)
performance(d_test.sentiment, pred4, color="c")
```

На илл. 8.9 производительность word2vec-модели на базе алгоритма «случайный лес» сравнивается с показателями предыдущих моделей. Легко заметить, что точность новой модели и в самом деле выше, как в выбранных оценочных метриках, так и во всех точках ROC-кривой.



Илл. 8.9. Кривые рабочей характеристики приемника word2vec-модели и ранее построенных моделей. Легко заметить улучшение для всех точек ROC-кривой, также отражающееся в увеличившейся точности и величине параметра AUC

Производительность последнего варианта модели нас вполне устраивает, так что работу по оптимизации можно прекратить. Но вы можете попро-

бовать другие варианты увеличения точности. С большой вероятностью даже человек будет не в состоянии корректно классифицировать тональность всех обзоров; могут как появиться неверные метки, так и обзоры, понять тональность которых крайне сложно.

Но модель может работать намного лучше, чем в настоящий момент. Поэтому мы составили для вас, наши дорогие читатели, список того, что можно попробовать сделать в порядке приоритетности:

- *Воспользоваться немаркированными данными для построения лучшей тематической модели.*

  Посвященный данным раздел на сайте Kaggle содержит немаркированный набор рецензий, которые можно использовать для обучения. Так как мы реализуем обучение с учителем, на первый взгляд эти данные кажутся бесполезными. Но мы строим word2vec-модель, которая должна изучить нюансы встречающихся в обзорах на сайте IMDb слов, а особенно связи между различными словами и понятиями, соответственно эти данные помогут улучшить ее первый вариант, с признаками на базе обучающей выборки (которые оснащены метками), до того, как вы построите модель целиком.

- *Оптимизировать параметры.*

  Вы видели, как сильно увеличивалась производительность первых вариантов нашей модели после подбора оптимальных значений гиперпараметров. Но мы уже успели построить новую модель (word2vec) и взять новый ML-алгоритм («случайный лес»), так что у вас появилось множество новых параметров, которые можно оптимизировать.

- *Распознать фразы.*

  Библиотека Gensim включает в себя поддержку распознавания фраз, таких как «New York City», которые упускает наша «глупая» функция токенизации отдельных слов. В английском языке распространены понятия, состоящие из нескольких слов, и их имеет смысл включить в функцию генерации предложений.

- *Обработать разные языки.*

  Если не все обзоры написаны на одном языке (в рассматриваемом случае на английском), придется вставлять в различные места на-

шего конвейера обработку различных языков. Первым делом нужно понять, на каком языке написана рецензия, то есть распознать язык (существует несколько библиотек, с разным успехом решающих эту задачу). Затем с помощью этой информации процесс токенизации следует заставить пользоваться различными шумовыми словами и, возможно, разными знаками препинания. В особо неудачных случаях приходится иметь дело с совершенно другой структурой предложений, как, например, в китайском языке, где отличить одно слово от другого можно только по наличию пробела.

Теперь представим, что полученная модель вас устраивает. В реальном мире на этой стадии модель запускают в производство. При этом учитываются следующие аспекты:

- *Насколько велик размер обучающей выборки и не улучшится ли модель, если мы добавим в эту выборку дополнительные данные?*

  Этот аспект влияет на выбор ML-алгоритма, так как готовая модель должна хорошо масштабироваться при росте обучающей выборки. К примеру, наивный байесовский классификатор поддерживает так называемое динамическое обучение, в то время как алгоритм «случайный лес» сложно адаптировать к увеличившемуся набору данных.

- *Каков объем прогнозов и должны ли они делаться в реальном времени?*

  Подробно масштабирование прогнозов по объему и скорости будет обсуждаться в следующей главе, а пока следует учесть, что ответ на этот вопрос тоже влияет на выбор алгоритма и инфраструктуры, в которой будет разворачиваться модель.

8.4. Заключение

В этой главе мы на практическом примере рассмотрели процесс машинного обучения от начала до конца, заодно обсудив основы обработки естественного текста и параметры оптимизации модели.

Вот то, что следует запомнить:

- Важно правильно формулировать задачу. Работу всегда следует начинать с ответа на вопрос: «Какова практическая ценность решения данной задачи?».

- Анализируйте данные, чтобы понять, хватит ли их для решения поставленной задачи.
- Начинайте работу с простых стандартных алгоритмов, позволяющих построить первый вариант модели. В нашем примере тональность рецензий была предсказана с почти 90%-й точностью.
- Точность предсказаний можно увеличивать путем тестирования и оценки альтернативных моделей, а также комбинаций параметров модели.
- Зачастую приходится искать компромисс между различными параметрами модели и оценочными критериями. Мы посмотрели, как для нашей модели оценки тональности компромисс между долями ложно-положительных и ложно-отрицательных значений выражается с помощью ROC-кривой.
- Ультрасовременные техники обработки естественного языка и ML-моделирования, такие как word2vec, — примеры того, как усовершенствованное проектирование признаков улучшает точность работы моделей.
- Выбор алгоритма может зависеть не только от точности модели, а, например, от времени обучения, необходимости добавления новых данных или получения прогнозов в реальном времени.
- Реальные модели всегда можно улучшить.

8.5. Терминология

Термин	Определение
word2vec	Программный инструмент для анализа текстов, выпущенный Google и используемый множеством ультрасовременных систем машинного обучения, работающих с естественным языком
Оптимизация гиперпараметров (hyperparameter optimization)	Различные техники выбора параметров, обеспечивающие максимальную производительность ML-алгоритмов