Отчет по работе - Лупенко Н.А. по направлению Data Science

Задачи:

1. **Обучить модель на языке Python для классификации отзывов.**
2. **Разработать веб-сервис на базе фреймворка Django для ввода отзыва о фильме с автоматическим присвоением рейтинга (от 1 до 10) и статуса комментария (положительный или отрицательный).**
3. **Развернуть сервис в открытом доступе для оценки работоспособности прототипа.**

Выполнение:

1. В результате проделанной работы было создано две модели нейросетей с различной архитектурой на базе фреймворка Keras.

Обучение проводилось в Jupyter Lab. После модель сохранялась и использовалась непосредственно внутри сервиса.

Суть задачи состоит в определении тональности отзыва - {Отрицательный, Положительный}, а также присвоении рейтинга по шкале от 1 до 10 каждому отзыву. То есть все сводится к задаче классификации. Тем не менее, на мой взгляд, необычность моего подхода в том, что модели работают только на бинарную классификацию с последующим вероятностным отнесением вектора токенов к классу 0, 1. Суть в том, что рейтинг выставляется на основании усреднения вероятностного вывода двух моделей, то есть прим. (0 – 0.1 - рейтинг 1, …, 0.9 - 1 - рейтинг 10).

Теперь об обработке. В начале текст очищается от ненужных символов, таких как {.,\")(>+<*/'}. А все весомые знаки препинания {!?} отделяются от слов. После происходит разбиение на токены, все незнакомые языковые конструкции, которые пережили токенизацию, в последствии были заменены на 0.

Внутри архива были файлы: imdb.vocab, imdbEr.txt - посредством их и происходит замена токенов на числовые значения. В первом случае происходит замена токенов на ожидаемый рейтинг(вес). Во втором замена на номер токена в словаре (предполагалось, что словарь хранит токены в порядке частоты появления в датасете). Все это необходимо для того, чтобы обучить две модели, которые ищут разные паттерны в текстах. И в последствии провести бэггинг, то есть усреднить значения для более качественной классификации.

В процессе чтения файлов происходит извлечение оценки отзыва из имени файла прим. ({100_10.txt} - оценка 10). Отдельно собираются тренировочная и тестовая выборка в каждой по 25000 значений из отрицательных и положительных отзывов. В начале, из каждой папки с негативными и положительными отзывами происходит потоковое чтение, в это же время все тексты обрабатываются алгоритмами, описанными выше. После собирается датафрейм с двумя столбцами - вектор, верный лейбл. Далее все кортежи перемешиваются. И датафреймы опять разбиваются на группы массивов.

После все полученные векторы “усекаются” до определенного размера. Эмпирическим путем было выявлено, что для модели, которая принимает векторы с ожидаемым рейтингом токенов оптимальный размер - 375, для другой 300.

Первая модель состоит из 6-ти обычных слоев, где:

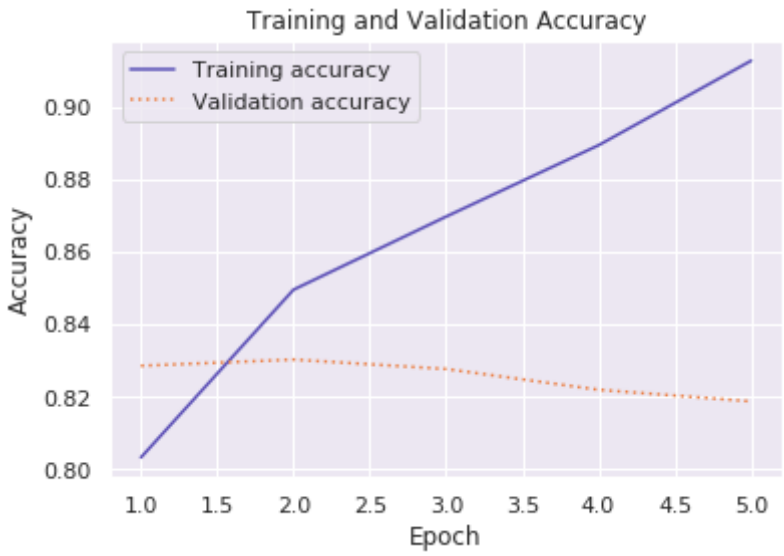
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 375)               141000
_________________________________________________________________
dense_8 (Dense)              (None, 50)                18800
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_9 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout_5 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_10 (Dense)             (None, 50)                2550
_________________________________________________________________
dropout_6 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_11 (Dense)             (None, 50)                2550
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 51
=================================================================
Total params: 167,501
Trainable params: 167,501
Non-trainable params: 0
_________________________________________________________________
```

После каждого слоя Dense, кроме последнего (sigmoid), идет функция активации relu.

В качестве оптимизатора выбран - adam, функция потерь - binary_crossentropy, т.к. решается задача бинарной классификации, метрика оценки - accuracy.

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/5
25000/25000 [==============================] - 4s 151us/step - loss: 0.4357 - acc: 0.8033 - val_loss: 0.3968 - val_acc: 0.8286
Epoch 2/5
25000/25000 [==============================] - 3s 140us/step - loss: 0.3654 - acc: 0.8495 - val_loss: 0.3939 - val_acc: 0.8303
Epoch 3/5
25000/25000 [==============================] - 3s 137us/step - loss: 0.3251 - acc: 0.8696 - val_loss: 0.4036 - val_acc: 0.8277
Epoch 4/5
25000/25000 [==============================] - 3s 138us/step - loss: 0.2791 - acc: 0.8893 - val_loss: 0.4233 - val_acc: 0.8220
Epoch 5/5
25000/25000 [==============================] - 3s 135us/step - loss: 0.2306 - acc: 0.9126 - val_loss: 0.4571 - val_acc: 0.8188
```

Во время обучения (за 5 эпох, каждый батч размером 100), можно заметить, что модель подвергается переобучению уже после 2-ой эпохи, данный вывод можно сделать по графику:



Следовательно модели хватает на обучение всего 2 эпохи. По результатам проверки на валидационной выборке точность модели составляет примерно ~83%.

Вторая модель имеет совершенно другую архитектуру:

Layer (type)	Output Shape	Param #
embedding_16 (Embedding)	(None, 300, 32)	2880000
conv1d_14 (Conv1D)	(None, 300, 32)	3104
max_pooling1d_14 (MaxPooling	(None, 150, 32)	0
lstm_16 (LSTM)	(None, 100)	53200
dense_16 (Dense)	(None, 1)	101

Total params: 2,936,405
Trainable params: 2,936,405
Non-trainable params: 0

Первый слой - слой внедрения производит сопоставление многомерных массивов с индексами слов в виде целых чисел с массивами дробных чисел с меньшей размерностью. Далее происходит свертка с размерностью ядра - 3, после операцией подвыборки произведем сжатие

для последующей подачи на слой LSTM. Завершает все слой с одним узлом и функцией активации - sigmoid.

В качестве оптимизатора выбран - adam, функция потерь - binary_crossentropy, т.к. решается задача бинарной классификации, метрика оценки - accuracy.

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/2
25000/25000 [==============================] - 117s 5ms/step - loss: 0.4627 - acc: 0.7576 - val_loss: 0.3003 - val_acc: 0.8739
Epoch 2/2
25000/25000 [==============================] - 100s 4ms/step - loss: 0.2033 - acc: 0.9265 - val_loss: 0.3480 - val_acc: 0.8595
```

Для обучения данной модели достаточно одной эпохи, после чего начинается процесс оверфиттинга, что явно можно заметить по данному графику:

На валидационной выборке точность составляет ~87%.

Теперь проверим точность ансамбля моделей.

```
Accuracy: 87.7%
```

Как можно заметить, значительного прироста точности не произошло, но бэггинг в то же время увеличивает шансы правильного ответа при выставлении рейтинга от 1 до 10.

2. Был собран проект Django - review_estimator, внутри которого было собрано приложение estimator. Приложение подгружает заранее обученные модели, и выставляет тональность и рейтинг введенному отзыву. Для стилизации проекта был использован bootstrap.
3. Адрес развернутого сервиса - http://review-estimator.morell.lclients.ru/

   Поиск подходящего хостинга был довольно сложной задачей, потому что бесплатные предоставляют только 512 мб на диске, а сервис со всеми зависимостями весит около 612 мб. Как следствие - пришлось использовать пробную версию хоста, которая будет активна до 13-го июля.