Санкт-Петербургский Государственный Университет
Факультет Прикладной математики – Процессов управления
Кафедра технологий программирования

Ефремов Иван Дмитриевич

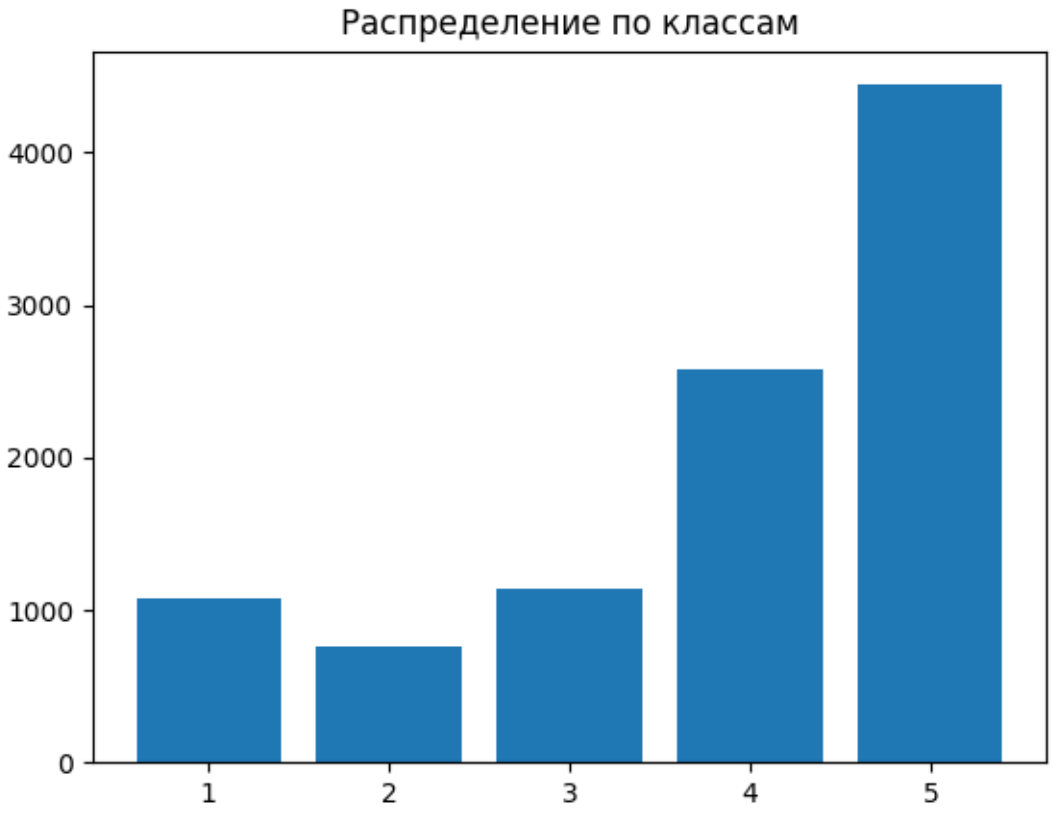
Предобработка и классификация отзывов с помощью нейронных сетей на примере датасета YELP

Постановка задачи

В данном проекте осуществляется классификация отзывов о различных бизнесах на основе датасета YELP (Yelp Dataset).
Датасет содержит в себе около 600000 значений, содержащий данные о бизнесе и человеке, оставившем отзыв и текст отзыва на английском языке с присвоенным ему рейтингом от 1 до 5.

Задача состоит в предобработке текста отзыва, составлении архитектуры нейронной сети и её обучении.

Т.к. данные имеют большой вес – 4.98 G, было принято взять выборку из 10000 первых строк датасета для возможности предобработки и обучения с учётом возможностей ПК.



Разделение на обучающую, тестовую и валидационную выборки взято в соотношении 80/10/10

В качестве критерия качества была выбрана метрика accuracy как самая простая и в то же время дающая достаточные данные для оценки

Использованные программные средства

Весь код проекта был написан на языке Python 3.12.

Для обработки текста были использованы библиотеки JSON, re, pandas, nltk и sklearn

Для отображения результатов была использована библиотека matplotlib и pandas.

Для создания и обучения нейросетей была использована библиотека PyTorch. Выбор обоснован возможностью использования графического ускорителя Nvidia для текущей версии CUDA. Библиотека Tensorflow не поддерживает работу с текущей версией Python и CUDA, а позволяет использовать только CPU.

Очистка текста

Данные в датасете представлены в виде естественного языка, что не совсем подходит для дальнейшего использования для обучения.

Естественный язык содержит множество слов, не несущих существенного смысла (например, «и», «на», «в»). Такие слова в NLP называются stopwords. Очистка датасета от подобных слов позволяет снизить шум и повысить точность модели, фокусируя внимания на ключевых словах.

Помимо этого, слова несут одинаковые значения в независимости от их формы («хорошо» = «хороший»). Процесс приведения слов к их основам, путём удаления окончаний, суффиксов и префиксов называется stemming. Он позволяет сократить количество уникальных слов в данных.

Очистка датасета производится на языке Python с использованием библиотеки для работы с естественным языком nltk. Библиотека позволяет загрузить корпус stopwords для английского языка, а также предоставляет инструмент для stemming.

Текст отзыва до очистки:

If you decide to eat here, just be aware it is going to take about 2 hours from beginning to end. We have tried it multiple times, because I want to like it! I have been to it's other locations in NJ and never had a bad experience. \n\nThe food is good, but it takes a very long time to come out. The waitstaff is very young, but usually pleasant. We have just had too many experiences where we spent way too long waiting. We usually opt for another diner or restaurant on the weekends, in order to be done quicker.

Текст отзыва после очистки:

decid eat awar go take hour begin end tri multipl time want like locat nj never bad experi food good take long time come waitstaff young usual pleasant mani experi spent way long wait usual opt anoth diner restaur weekend order done quicker

Предобработка текста – статический подход

Для работы нейросети с данными естественного языка требуется составить их векторное представление.

Для представления текстов в виде статического числового вектора была выбран метод TF-IDF (Term Frequency – Inverse Document Frequency).

Метод выполняется следующими шагами:

1. Подсчёт частоты слов в отзыве (TF)
2. Вычисление логарифма от отношения общего числа документов к количеству документов, содержащих это слово. (IDF)
3. Умножение TF*IDF

TF-IDF позволяет не только векторизовать текст, но так же даёт оценку словам, что помогает выделять ключевые слова для моделей классификации.

Векторизация производится с помощью модуля библиотеки sklearn.feature_extraction.text. Гиперпараметр MAX_FEATURES позволяет ограничит максимальную длину векторного представления, тем самым исключив редкие, неинформативные слова.

Текст отзыва:

decid eat awar go take hour begin end tri multipl time want like locat nj never bad experi food good take long time come waitstaff young usual pleasant mani experi spent way long wait usual opt anoth diner restaur weekend order done quicker

Ненулевые элементы векторного представления:

0.15856954372008786 0.20184382421206926 0.25257581505447746 0.20506595160034494 0.21636631931811498 0.18420592804295424 0.30604474023570005 0.31067250866837276 0.16797831912936365 0.2159354220283532 0.17181248249654196 0.26250521365817736 0.2352883381836684 0.16669302454348023 0.30430838455539727 0.20109652324223762 0.11187647476223385 0.1454462009428051 0.17609658334817269 0.15055261430723973 0.2528766126945296 6 0.12877753946869233

Архитектура – статический подход

Для классификации текста в статическом векторном представлении была выбрана простая линейная полносвязная архитектура.

Простота обучения позволяет провести сравнительный анализ разных уровней предобработки датасетов и сформировать несколько вариантов обученных моделей на их основе.

```
class Linear_model(nn.Module):
    def __init__(self, output_size = 5, hidden_size = 5):
        super(Linear_model, self).__init__()
        self.fc1 = torch.nn.Linear(MAX_FEATURES, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, hidden_size)
        self.fc3 = torch.nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = torch.relu((self.fc1(x)))
        x = torch.relu((self.fc2(x)))
        x = self.fc3(x)

        return F.log_softmax(x, dim=1)
```

В качестве функции активации на скрытых слоях была взята relu, на выходном - log_softmax для нормализации выходных значений в вероятности.

В качестве функции ошибки стандартная для много классовой классификации кросс-энтропия, оптимизатор Adam и стандартный для него lr=0.01.

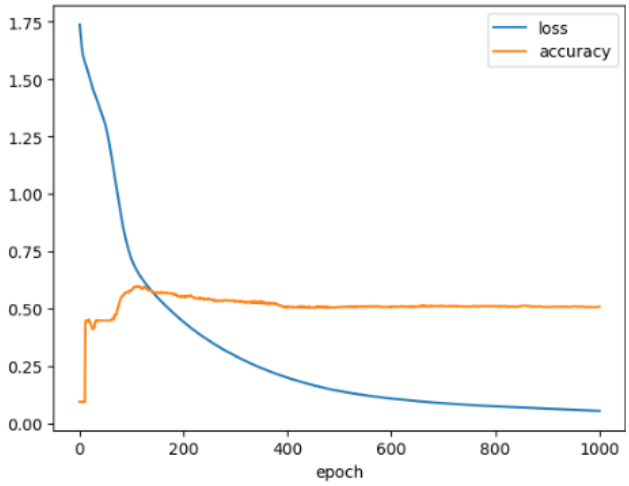
При обучении данные не делятся на батчи учитывая небольшой размер и то, что вычисления производятся на GPU.

Гиперпараметром сети в данном случае является только количество нейронов в скрытых слоях.

Сравнение моделей для различных вариантов предобработки

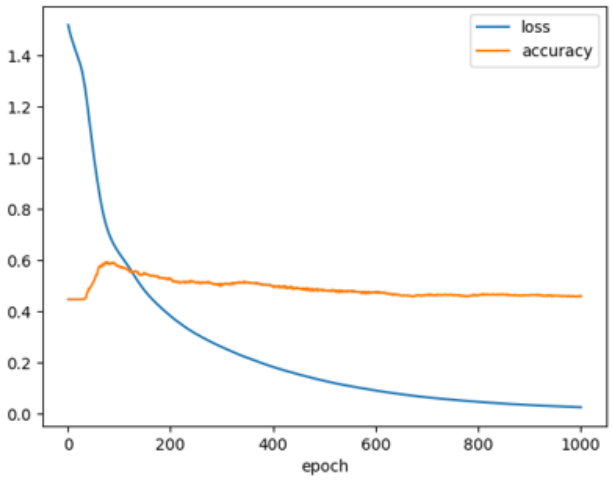
Очищенный датасет:

```
Epochs: 1000
Training time: 1.68 seconds
Test acc: 0.492
```



Неочищенный датасет

```
Epochs: 1000
Training time: 1.65 seconds
Test acc: 0.483
```



Как видно из результатов эксперимента, точность на тестовой выборке у модели, обученной на очищенном датасете выше, пусть и незначительно.

Максимальное значение метрики достигается примерно на 150 эпохе обучения.

Предобработка текста – последовательный подход

Слова в тексте не являются самодостаточными смысловыми единицами, чтобы верно оценить их значение, требуется понимать контекст слова в тексте.

Для этого можно представить текст в виде последовательности токенов.

Для токенизации датасета был использован токенизатор BERT из библиотеки transformers, который работает по следующему алгоритму:

1. Текст разбивается на слова
2. Создаётся словарь токенов – все возможные токены, которые могут встретиться в тексте
3. Каждому токену в словаре соответствует уникальный индекс

Различные токенизаторы определяют различные алгоритмы разбиения текста на токены, BERT был выбран из-за эффективности в задачах много классовой классификации.

Архитектура – последовательный подход

Для работы с последовательными данными была выбрана простая двунаправленная LSTM архитектура:

```
class LSTM_model(nn.Module):
    def __init__(self, vocabulaire_size2, embedding_dim, hidden_size1=128,
hidden_size2=64):
        super(LSTM_model, self).__init__()
        self.embedding = nn.Embedding(vocabulaire_size2, embedding_dim)

        self.lstm1 = nn.LSTM(embedding_dim, hidden_size1, bidirectional=True)
        self.lstm2 = nn.LSTM(hidden_size1 * 2, hidden_size2, bidirectional=True)

        self.dropout = nn.Dropout(0.2)

        self.dense = nn.Linear(hidden_size2 * 2, 5)
        self.softmax = nn.Softmax(dim=1)
```

В качестве функции активации на выходном слое была выбрана softmax для нормализации выходных значений в вероятности.

В качестве функции ошибки стандартная для многоклассовой классификации кросс-энтропия, но был взят другой оптимизатор – SGD с lr=0.01, т.к. он показал себя лучше на небольшом количестве эпох.

Гиперпараметры сети и обучения:

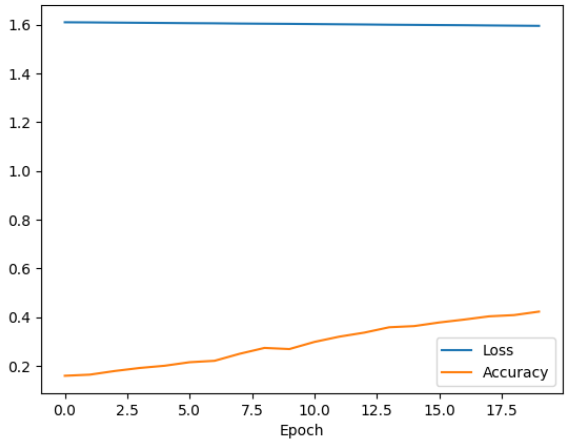
Batch_size = 128
Embedding dim = 128 – размерность векторного представления каждого токена
Hidden_size = 128, 64 – размерность скрытого состояния LSTM
Dropout = 0.2 – вероятность отключения нейронов для предотвращения переобучения

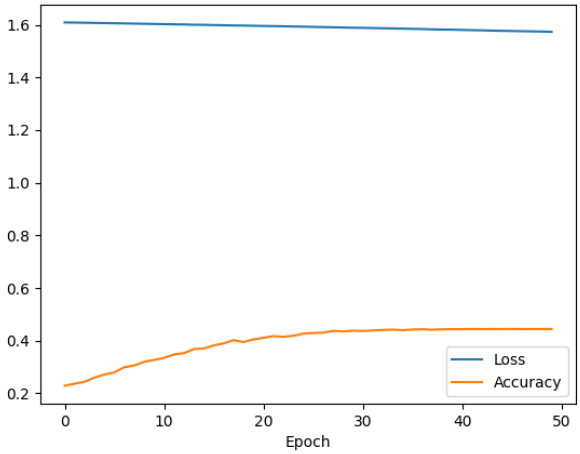
Результаты обучения

LSTM – модели обычно позволяют точнее классифицировать текст, но требуют очень больших вычислений и большого количества эпох для обучения

```
Epochs = 20
Training time: 62.9 seconds
Final acc: 0.403
Test acc: 0.4155
```

```
Epochs = 50
Training time: 150.7 seconds
Final acc: 0.4445
Test acc: 0.4415
```





Как видно Loss-функция меняется крайне медленно, но с увеличением числа эпох точность на тестовой выборке возрастает. При этом сильно вырастает время обучения модели.

Если сравнивать два рассмотренных подхода на основе точности, то видно, что LSTM модель достигла точности линейной уже к 30 эпохе, но при этом время обучения больше в 100 раз.

Так же отличается и вес сохранённых моделей: 32 KB у линейной и 16 MB у LSTM.