МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №7
по дисциплине «Искусственные нейронные сети»
Тема: «Классификация обзоров фильмов»

Студент гр. 8383 _____ Муковский Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы

Классификация последовательностей — это проблема прогнозирующего моделирования, когда у вас есть некоторая последовательность входных данных в пространстве или времени, и задача состоит в том, чтобы предсказать категорию для последовательности.

Проблема усложняется тем, что последовательности могут различаться по длине, состоять из очень большого словарного запаса входных символов и могут потребовать от модели изучения долгосрочного контекста или зависимостей между символами во входной последовательности.

В данной лабораторной работе также будет использоваться датасет IMDb, однако обучение будет проводиться с помощью рекуррентной нейронной сети.

Задачи

1. Ознакомиться с рекуррентными нейронными сетями
2. Изучить способы классификации текста
3. Ознакомиться с ансамблированием сетей
4. Построить ансамбль сетей, который позволит получать точность не менее 97%

Требования

1. Найти набор оптимальных ИНС для классификации текста
2. Провести ансамблирование моделей
3. Написать функцию/функции, которые позволят загружать текст и получать результат ансамбля сетей
4. Провести тестирование сетей на своих текстах (привести в отчете)

Ход работы

Были выбраны и реализованы три различные архитектуры ИНС для классификации текста. Архитектуры моделей представлены описаны в функциях get_model_* и представлены в приложении А.

Ансамблирование моделей

Все модели были обучены в цикле и оценены на тестовых данных. Далее была написана функция, которая оценивает результат предсказания ансамбля моделей, accuracy_score – принимает вектор тестовых результатов и среднее предсказание по трем предсказаниям моделей и возвращает точность ансамбля. Результат работы приведен ниже:

```
Epoch 1/2
188/188 [==============================] - 12s 47ms/step - loss: 0.0219 - accuracy: 0.9920 - val_loss: 0.5949 - val_accuracy: 0.8568
Epoch 2/2
188/188 [==============================] - 8s 43ms/step - loss: 0.0129 - accuracy: 0.9963 - val_loss: 0.6995 - val_accuracy: 0.8636
model accuracy: 86.29%
Epoch 1/2
188/188 [==============================] - 7s 30ms/step - loss: 0.0203 - accuracy: 0.9946 - val_loss: 0.7295 - val_accuracy: 0.8553
Epoch 2/2
188/188 [==============================] - 5s 28ms/step - loss: 0.0115 - accuracy: 0.9971 - val_loss: 0.7797 - val_accuracy: 0.8523
model accuracy: 84.70%
Epoch 1/2
188/188 [==============================] - 3s 13ms/step - loss: 0.0120 - accuracy: 0.9958 - val_loss: 1.0092 - val_accuracy: 0.8688
Epoch 2/2
188/188 [==============================] - 2s 13ms/step - loss: 0.0112 - accuracy: 0.9959 - val_loss: 0.8973 - val_accuracy: 0.8703
model accuracy: 85.87%
Ensemble accuracy: 88.38%
```

Точность ансамбля моделей больше каждой конкретной ОНС. Она составляет 88.38%.

Функция ввода пользовательского текста

Функция ввода пользовательского текста (`read_text_from_input`) представлена в исходном коде в Приложении А. Функция получает словарь с ключами словами и их индексами, далее удаляем все не буквы и не цифры с помощью регулярки. Заменяем слова на их индексы и оставляем только 10000 самых частых слов. Далее расширяем текст до длины 500 и выводим результат предсказания ансамбля нейронных сетей. Пример работы представлен ниже.

```
This is a horrible movie I ever seen, disgusting. I really hate it. Delete it from internet.
[[0.0020228]]
[[0.00044117]]
[[0.0291345]]
[0]

 I love this movie. Director is genius. Please give me more like that.
 [[0.988782]]
 [[0.8887325]]
 [[0.984209]]
 [1]
```

Выводы

В ходе лабораторной работы был реализован прогноз успеха фильмов по обзорам. Также был изучен и применен метод ансамблирования моделей для более точного семантического анализа текста.

Приложение А. Исходный код программы

```
import numpy as np
from keras.layers import Dense, Dropout, LSTM, Conv1D, MaxPool1D, Flatten
from keras.models import Sequential
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from sklearn.metrics import accuracy_score
import re
from keras.datasets import imdb

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=10000)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)

X_test = data[:10000]
Y_test = targets[:10000]
X_train = data[10000:]
Y_train = targets[10000:]

max_review_length = 500
embedding_vector_length = 32
top_words = 10000

X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)


def get_model_a():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(128, 3, padding='same', activation='relu'))
    model.add(MaxPool1D(2))
    model.add(Dropout(0.4))
    model.add(LSTM(40, return_sequences=True, dropout=0.5))
    model.add(LSTM(20, dropout=0.3))
    model.add(Dense(1, activation='sigmoid'))
    return model


def get_model_b():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(32, 3, padding='same', activation='relu'))
    model.add(MaxPool1D(2))
    model.add(Dropout(0.2))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    return model


def get_model_c():
```

```
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(32, 3, padding='same', activation='relu'))
    model.add(MaxPool1D(2))
    model.add(Dropout(0.3))
    model.add(Conv1D(32, 3, padding='same', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    return model


train_size, test_size = len(X_train) // 3, len(X_test) // 3


def get_ensemble_predictions(all_models, x_test, X):
    result = []
    for m in all_models:
        if X:
            print(m.predict(x_test, verbose=0))
        result.append(np.round(m.predict(x_test, verbose=0)))
    result = np.asarray(result)
    result = [np.round(np.mean(x)) for x in zip(*result)]
    return np.asarray(result).astype('int')


all_models = [get_model_a(), get_model_b(), get_model_c()]

for i, model in enumerate(all_models):
    x_train = X_train[i * train_size: (i + 1) * train_size]
    y_train = Y_train[i * train_size: (i + 1) * train_size]
    x_test = X_test[i * test_size: (i + 1) * test_size]
    y_test = Y_test[i * test_size: (i + 1) * test_size]
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(x_train, y_train, validation_split=0.1, epochs=2,
batch_size=64)
    scores = model.evaluate(x_test, y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores[1] * 100))

result_predictions = get_ensemble_predictions(all_models, X_test, False)
acc = accuracy_score(Y_test, result_predictions)
print("Accuracy: %.2f%%" % (acc * 100))


def read_text_from_input():
    dictionary = imdb.get_word_index()
    words = input()
    words = re.sub(r"[^\w']", " ", words).split(' ')

    valid = []
    for word in words:
        word = dictionary.get(word)
```

```
            if word in range(1, 10000):
                valid.append(word + 3)

        X = []
        X.append(valid)
        result = sequence.pad_sequences(X, maxlen=max_review_length)
        print(get_ensemble_predictions(all_models, result, True))


read_text_from_input()
```