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

ОТЧЕТ

по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: «Wine quality»

Студентка гр. 8383 Сырцова Е.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2021

Цель

Разработать и реализовать модель ИНС решающую задачу предсказания качества вина по его характеристикам на датасете wine quality.

Требования

- Модель должна быть разработана на языке Python с использованием Keras API
- Исходный код проекта должен быть быть в формате PEP8
- В исходном коде должны быть поясняющие комментарии
- Модель не должна быть избыточной (должен соблюдаться баланс между размером сети [кол-во слоев и нейронов] и качеством выдаваемого результата)
- Обучение модели должно быть стабильно (для предложенной архитектуры ИНС обучение должно приводить к примерно одним и тем же результатом, то есть не должно быть такого, что при обучении 10 сетей удовлетворительный результат дают только 5 из них)
- *Плюсом будет анализ с использованием callback'а TensorBoard*
- *Плюсом будет разработка собственных callback'ов*
- *Плюсом будет создание модели из ансамбля ИНС*

Ход работы

Описание датасета и решаемой задачи

Исходный датасет разделен на две части с красным и белым вином:

Number of Instances: red wine - 1599; white wine - 4898.

Number of Attributes: 11 + output attribute

Attribute information:

Input variables (based on physicochemical tests):

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

Вид датасета:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.17	0.74	12.8	0.045	24.0	126.0	0.99420	3.26	0.38	12.2	8
1	7.7	0.64	0.21	2.2	0.077	32.0	133.0	0.99560	3.27	0.45	9.9	5
2	6.8	0.39	0.34	7.4	0.020	38.0	133.0	0.99212	3.18	0.44	12.0	7
3	6.3	0.28	0.47	11.2	0.040	61.0	183.0	0.99592	3.12	0.51	9.5	6
4	7.4	0.35	0.20	13.9	0.054	63.0	229.0	0.99888	3.11	0.50	8.9	6

Задача заключается в предсказании качества вина на основе его химических характеристик.

Начальный анализ данных

Классы упорядочены и не сбалансированы (например, нормальных вин больше, чем отличных или плохих).

Обработка данных

Было загружено два датасета: winequality-white.csv и winequality-red.csv.

Т.к. предсказание качества вина по десятибалльной шкале более трудная задача и нет необходимости определить именно оценку вина, добавим к характеристикам вина quality_label, который будет определять вина по quality в три категории: плохое, хорошее, отличное.

Объединим данные из датасетов красных и белых вин в один и перемешаем данные.

```
#считывание датасетов
white_wine = pd.read_csv('winequality-white.csv', sep=';')
red_wine = pd.read_csv('winequality-red.csv', sep=';')
#добавим метку качества
red_wine['quality_label'] = red_wine['quality'].apply(lambda value: 'low'
                                                      if value <= 5 else 'medium'
                                                          if value <= 7 else 'high')
white_wine['quality_label'] = white_wine['quality'].apply(lambda value: 'low'
                                                          if value <= 5 else 'medium'
                                                              if value <= 7 else 'high')
#объединение датасетов
wines = np.concatenate((red_wine, white_wine), axis=0)
#перемешивание
wines = shuffle(wines)
```

Разделим загруженный набор на входные данные – химические характеристики вина, и выходные – качество вина. Т.к. качество вина определяется текстовыми метками, приводим их к категориальному виду. Разделяем данные на обучающий и тестовый набор в соотношении 4:1.

```
#разделение на химические характеристики и оценку
data = wines[:, 0:11]
targets = wines[:, 12]
data = np.array(data).astype("float32")
#переход от текстовых меток к категориальному виду
encoder = LabelEncoder()
encoder.fit(targets)
encoded_Y = encoder.transform(targets)
targets = to_categorical(encoded_Y)
#разделение на обучающий и тестовый наборы
train_x, test_x, train_y, test_y = train_test_split(data, targets, test_size=0.2)
```

Начало разработки модели

В начале тестировалась модель, для которой задавались различные параметры количества нейронов на слое (128, 256, 512, 1024), оптимизаторов (Adagrad, Adam, RMSprop, SGD), количество эпох (50, 100) и коэффициентов скорости обучения (0.01, 0.001).

```
model = Sequential()
model.add(Dense(layers, activation="relu", input_shape=(11,)))
model.add(Dense(3, activation='softmax'))
```

Также тестировалась классификация вина по десятибалльной шкале и по категориям, на задаче классификации по категориям были получены лучшие значения.

Результаты тестирования классификации по десятибалльной шкале:

Adagrad 0.001 50 128 0.54666668176651	Adam 0.001 50 128 0.5943589806556702	RMSprop 0.001 50 128 0.588205099105835	SGD 0.01 50 128 0.563076913356781
Adagrad 0.001 50 256 0.5451282262802124	Adam 0.001 50 256 0.6133333444595337	RMSprop 0.001 50 256 0.5723077058792114	SGD 0.01 50 256 0.5600000023841858
Adagrad 0.001 50 512 0.552307665348053	Adam 0.001 50 512 0.6200000047683716	RMSprop 0.001 50 512 0.6082051396369934	SGD 0.01 50 512 0.5702564120292664
Adagrad 0.001 50 1024 0.5564102530479431	Adam 0.001 50 1024 0.6338461637496948	RMSprop 0.001 50 1024 0.6215384602546692	SGD 0.01 50 1024 0.5687179565429688
Adagrad 0.001 100 128 0.5502564311027527	Adam 0.001 100 128 0.6082051396369934	RMSprop 0.001 100 128 0.5841025710105896	SGD 0.01 100 128 0.5712820291519165
Adagrad 0.001 100 256 0.5574358701705933	Adam 0.001 100 256 0.6287179589271545	RMSprop 0.001 100 256 0.573846161365509	SGD 0.01 100 256 0.5748717784881592
Adagrad 0.001 100 512 0.5594871640205383	Adam 0.001 100 512 0.6333333253860474	RMSprop 0.001 100 512 0.6117948889732361	SGD 0.01 100 512 0.5676922798156738
Adagrad 0.001 100 1024 0.5600000023841858	Adam 0.001 100 1024 0.6374359130859375	RMSprop 0.001 100 1024 0.6323077082633972	SGD 0.01 100 1024 0.5728204846382141

Результаты тестирования классификации по категориям вина:

Adagrad 0.001 50 128 0.6530769467353821	Adam 0.001 50 128 0.6915384531021118	RMSprop 0.001 50 128 0.6623076796531677	SGD 0.01 50 128 0.5930769443511963
Adagrad 0.001 50 256 0.6446154117584229	Adam 0.001 50 256 0.7323076725006104	RMSprop 0.001 50 256 0.6876922845840454	SGD 0.01 50 256 0.618461549282074
Adagrad 0.001 50 512 0.6376923322677612	Adam 0.001 50 512 0.7161538600921631	RMSprop 0.001 50 512 0.7038461565971375	SGD 0.01 50 512 0.5992307662963867
Adagrad 0.001 50 1024 0.6246153712272644	Adam 0.001 50 1024 0.7246153950691223	RMSprop 0.001 50 1024 0.5692307949066162	SGD 0.01 50 1024 0.61307692527771
Adagrad 0.001 100 128 0.6353846192359924	Adam 0.001 100 128 0.7284615635871887	RMSprop 0.001 100 128 0.7007692456245422	SGD 0.01 100 128 0.6269230842590332
Adagrad 0.001 100 256 0.6453846096992493	Adam 0.001 100 256 0.6969230771064758	RMSprop 0.001 100 256 0.6784615516662598	SGD 0.01 100 256 0.6461538672447205
Adagrad 0.001 100 512 0.6600000262260437	Adam 0.001 100 512 0.7184615135192871	RMSprop 0.001 100 512 0.6984615325927734	SGD 0.01 100 512 0.6153846383094788
Adagrad 0.001 100 1024 0.6538461446762085	Adam 0.001 100 1024 0.7292307615280151	RMSprop 0.001 100 1024 0.6561538577079773	SGD 0.01 100 1024 0.5953845977783203

Далее будем рассматривать оптимизаторы Adam и RMSprop, количество эпох 50 и 100, коэффициенты скорости обучения (0.001, 0.01) и 3 модели:

```
def build_models():
    models = []
    model_1 = Sequential()
    model_1.add(Dense(512, activation="relu", input_shape=(11,)))
    model_1.add(Dense(256, activation="relu"))
    model_1.add(Dense(3, activation='softmax'))
    models.append(model_1)

    model_2 = Sequential()
    model_2.add(Dense(256, activation="relu", input_shape=(11,)))
    model_2.add(Dense(3, activation='softmax'))
    models.append(model_2)

    model_3 = Sequential()
    model_3.add(Dense(256, activation="relu", input_shape=(11, )))
    model_3.add(Dense(256, activation="relu"))
    model_3.add(Dropout(0.3, noise_shape=None, seed=None))
    model_3.add(Dense(128, activation="relu"))
    model_3.add(Dense(128, activation="relu"))
    model_3.add(Dropout(0.2, noise_shape=None, seed=None))
    model_3.add(Dense(64, activation="relu"))
    model_3.add(Dense(3, activation="softmax"))
    models.append(model_3)
    return models
```

Результаты тестовых данных на тренированных моделях:

	Модель 1	Модель 2	Модель 3
Adam, 50, 0.01	0.717692315578	0.703076899051	0.712307691574
Adam, 50, 0.001	0.69461536407	0.703846156597	0.690769255161
Adam, 100, 0.01	0.728461563587	0.728461563587	0.702307701110
Adam, 100, 0.001	0.713076949119	0.7038461565971	0.700769245624
RMSprop, 50, 0.01	0.716923058032	0.670000016689	0.687692284584
RMSprop, 50, 0.001	0.699230790138	0.700769245624	0.709230780601
RMSprop, 100, 0.01	0.676923096179	0.691538453102	0.702307701110
RMSprop, 100, 0.001	0.698461532592	0.703846156597	0.693076908588

Результат ансамблирования по среднему арифметическому для этих моделей достигает 82%.

При варьировании параметров модели повысить точность предсказания не удалось. Модель достигает более высокой точности на тренировочных данных по сравнению с тестовыми. Это может быть обоснованно несбалансированностью набора данных (преобладают хорошие вина, меньше плохих и отличных, также в датасете преобладают красные вина).

Вывод

В процессе выполнения работы была разработана модель сети, позволяющая предсказывать качество вина на основании его химических характеристик с точностью в 82%. Код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А

```
import pandas as pd
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.optimizers import RMSprop, Adam
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder
from keras.callbacks import ModelCheckpoint, TensorBoard
import tensorflow.keras.callbacks
from keras.utils import np_utils
import matplotlib.pyplot as plt
import numpy as np

#считывание датасетов
white_wine = pd.read_csv('winequality-white.csv', sep=';')
red_wine = pd.read_csv('winequality-red.csv', sep=';')
#добавим метку качества
red_wine['quality_label'] = red_wine['quality'].apply(lambda
value: 'low'
                                                      if
value <= 5 else 'medium'
                                                          if
value <= 7 else 'high')
white_wine['quality_label'] = white_wine['quality'].apply(lambda
value: 'low'
                                                          if
value <= 5 else 'medium'

if value <= 7 else 'high')
#объединение датасетов
wines = np.concatenate((red_wine, white_wine), axis=0)
#перемешивание
wines = shuffle(wines)
#разделение на химические характеристики и оценку
data = wines[:, 0:11]
targets = wines[:, 12]
data = np.array(data).astype("float32")
#переход от текстовых меток к категориальному виду
encoder = LabelEncoder()
encoder.fit(targets)
encoded_Y = encoder.transform(targets)
targets = to_categorical(encoded_Y)
#разделение на обучающий и тестовый наборы
train_x, test_x, train_y, test_y = train_test_split(data,
```

```
targets, test_size=0.2)

res = dict()
los = dict()

def build_models():
    models = []
    model_1 = Sequential()
    model_1.add(Dense(512, activation="relu",
input_shape=(11,)))
    model_1.add(Dense(256, activation="relu"))
    model_1.add(Dense(3, activation='softmax'))
    models.append(model_1)

    model_2 = Sequential()
    model_2.add(Dense(256, activation="relu",
input_shape=(11,)))
    model_2.add(Dense(3, activation='softmax'))
    models.append(model_2)

    model_3 = Sequential()
    model_3.add(Dense(256, activation="relu", input_shape=(11,
)))
    model_3.add(Dense(256, activation="relu"))
    model_3.add(Dropout(0.3, noise_shape=None, seed=None))
    model_3.add(Dense(128, activation="relu"))
    model_3.add(Dense(128, activation="relu"))
    model_3.add(Dropout(0.2, noise_shape=None, seed=None))
    model_3.add(Dense(64, activation="relu"))
    model_3.add(Dense(3, activation="softmax"))
    models.append(model_3)
    return models


models = build_models()
k = 0
for learning_rate in [0.001, 0.01]:
    for optimizers in [Adam(), RMSprop()]:
        for epochs in [50, 100]:
            for model in models:
                optimizer_config = optimizers.get_config()
                model.compile(optimizer=optimizers,
loss='categorical_crossentropy', metrics=['accuracy'])
                                history = model.fit(train_x,
train_y, epochs=epochs, batch_size=64,
                                    validation_data=(test_x,
```

```
test_y))
                loss, acc = model.evaluate(test_x, test_y)
                print('test_acc:', acc)
                '''
                plt.title('Training and validation accuracy')
                plt.plot(history.history['accuracy'], 'b',
label='Training accuracy')
                plt.plot(history.history['val_accuracy'], 'g',
label='Validation accuracy')
                plt.xlabel('Epochs')
                plt.ylabel('Accuracy')
                plt.legend()
                plt.show()
                # plt.savefig("%s_%s_%s_%s_acc.png" %
(optimizer_config["name"], optimizer_config["learning_rate"],
epochs, acc), format='png')
                plt.clf()

                plt.title('Training and validation loss')
                plt.plot(history.history['loss'], 'b',
label='Validation loss')
                plt.plot(history.history['val_loss'], 'g',
label='Validation loss')
                plt.xlabel('Epochs')
                plt.ylabel('Loss')
                plt.legend()
                plt.show()
                # plt.savefig("%s_%s_%s_%s_loss.png" %
(optimizer_config["name"], optimizer_config["learning_rate"],
epochs, acc), format='png')
                plt.clf()
                '''
                res["%s %s %s %s" % (optimizer_config["name"],
optimizer_config["learning_rate"], epochs, acc)] = acc
                los["%s %s %s %s" % (optimizer_config["name"],
optimizer_config["learning_rate"], epochs, loss)] = loss
                model.save("model%s.h5" % k)
                k += 1


predictions = 0
print("accuracy:\n")
for i in res.keys():
    print(i, "\n", res[i], "\n")
for i in range(0, k):
    model = load_model("model%i.h5" % i)
```

```
    predictions += model.predict(test_x)
predictions = np.greater_equal(predictions, np.array([0.5]))
acc = predictions.mean()
print("Accuracy of ensemble  %s" % acc)
```