

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Центр цифровых
образовательных технологий

09.04.01 Информатика и вычислительная техника

Работа с табличными данными с помощью библиотеки Keras
Вариант 2

ЛАБОРАТОРНАЯ РАБОТА № 2

по дисциплине:
Машинное обучение

Исполнитель:

студент группы

8BM42

Текере Ричард

Руководитель:

доцент

ОИТ, ИШИТР

Друки А.А.

Томск - 2025

ВВЕДЕНИЕ

Глубокое обучение стало одним из ведущих направлений в машинном обучении, а библиотека Keras (высокоуровневая API над TensorFlow) получила широкую популярность благодаря удобству интерфейса и гибкости построения нейронных сетей. Хотя нейронные сети в первую очередь ассоциируются с задачами распознавания изображений и речи, они также эффективно применяются к табличным данным (структурированным наборам данных в формате строк и столбцов).

В данной лабораторной работе исследуется процесс работы с табличными данными в Keras, с акцентом на задачи классификации. В частности, рассматриваются два практических примера: задача мультиклассовой классификации и задача бинарной классификации на основе реальных табличных наборов данных.

В Части 1 проводится мультиклассовая классификация на наборе данных "Body Performance" с платформы Kaggle. Этот набор содержит физические параметры и результаты тестов физической подготовки людей, а целевая метка отражает уровень их физической формы (классы A, B, C, D).

В Части 2 решается задача бинарной классификации на основе набора данных "Bank Marketing" из репозитория UCI Machine Learning. Этот набор включает демографические данные клиентов банка и характеристики маркетинговой кампании; целью является предсказание того, откроет ли клиент срочный депозит (да или нет).

Цель работы

получить навыки решения задач многоклассовой классификации табличных данных с использованием искусственных нейронных сетей в библиотеке машинного обучения Keras.

Ход работы

Методика выполнения каждой части лабораторной работы соответствует стандартному конвейеру машинного обучения. На первом

этапе проводится исследование и предварительная обработка данных для изучения особенностей набора данных и подготовки признаков для построения модели.

Затем разрабатывается модель с использованием библиотеки Keras, где определяется архитектура нейронной сети, подходящая для задачи классификации. После этого модель обучается на тренировочной выборке (с использованием части данных для валидации с целью настройки гиперпараметров и предотвращения переобучения) и в конечном итоге оценивается на отдельной тестовой выборке.

В процессе выполнения работы применяются библиотеки Python, такие как pandas для обработки данных, scikit-learn для разбиения данных и масштабирования признаков, а также Keras (TensorFlow) для построения и обучения нейронной сети.

ЧАСТЬ 1: Мультиклассовая классификация на наборе Body Performance

Для решения задачи мультиклассовой классификации использован набор данных Body Performance. Данный датасет содержит различные физические измерения и результаты тестов физической подготовки людей. Целью является предсказание класса физического развития человека (А, В, С или D).

1.1 Загрузка и исследование данных

На этом этапе загружаем датасет и знакомимся с его структурой.

```
import pandas as pd
df1 = pd.read_csv('bodyPerformance.csv')
df1.head()
```

Вывод:

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm	class
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4	60.0	217.0	C
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3	53.0	229.0	A
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0	49.0	181.0	C
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2	53.0	219.0	B
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1	45.0	217.0	B

Рисунок 1.1 — Загруженный набор данных.

1.2 Предобработка данных

Производится кодирование категориальных признаков, нормализация числовых данных и разбиение на обучающую, валидационную и тестовую выборки.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.utils import to_categorical

df1 = pd.get_dummies(df1, columns=['gender'])
le = LabelEncoder()
df1['class_label'] = le.fit_transform(df1['class'])
X1 = df1.drop(columns=['class', 'class_label'])
y1 = df1['class_label']

X1_train, X1_temp, y1_train, y1_temp = train_test_split(X1, y1, test_size=0.3,
stratify=y1, random_state=42)
X1_val, X1_test, y1_val, y1_test = train_test_split(X1_temp, y1_temp,
test_size=0.5, stratify=y1_temp, random_state=42)

scaler = StandardScaler()
numeric_cols = ['age', 'height_cm', 'weight_kg', 'body fat_%', 'diastolic', 'systolic',
'gripForce',
'sit and bend forward_cm', 'sit-ups counts', 'broad jump_cm']
X1_train[numeric_cols] = scaler.fit_transform(X1_train[numeric_cols])
X1_val[numeric_cols] = scaler.transform(X1_val[numeric_cols])
X1_test[numeric_cols] = scaler.transform(X1_test[numeric_cols])

y1_train_cat = to_categorical(y1_train)
y1_val_cat = to_categorical(y1_val)
y1_test_cat = to_categorical(y1_test)
```

1.3 Создание модели нейронной сети

Разрабатывается полносвязная нейронная сеть для задачи классификации.

```
model1 = Sequential([
    Dense(64, activation='relu', input_shape=(X1_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(4, activation='softmax')
])
model1.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model1.summary()
```

Вывод:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	832
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 4)	132

Total params: 3,044 (11.89 KB)
Trainable params: 3,044 (11.89 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 1.2 — Архитектура модели.

1.4 Обучение модели

Модель обучается на тренировочных данных с валидацией на отдельной выборке.

```
history1 = model1.fit(X1_train, y1_train_cat, epochs=30, batch_size=32,
validation_data=(X1_val, y1_val_cat), verbose=1)
```

Вывод:

```

Epoch 21/30 ----- 1s 3ms/step - accuracy: 0.7543 - loss: 0.5926 - val_accuracy: 0.7377 - val_loss: 0.6350
Epoch 22/30 ----- 1s 3ms/step - accuracy: 0.7553 - loss: 0.5978 - val_accuracy: 0.7307 - val_loss: 0.6357
Epoch 23/30 ----- 1s 3ms/step - accuracy: 0.7500 - loss: 0.5926 - val_accuracy: 0.7417 - val_loss: 0.6337
Epoch 24/30 ----- 1s 3ms/step - accuracy: 0.7582 - loss: 0.5848 - val_accuracy: 0.7417 - val_loss: 0.6313
Epoch 25/30 ----- 1s 3ms/step - accuracy: 0.7564 - loss: 0.5960 - val_accuracy: 0.7397 - val_loss: 0.6345
Epoch 26/30 ----- 1s 3ms/step - accuracy: 0.7562 - loss: 0.5858 - val_accuracy: 0.7402 - val_loss: 0.6295
Epoch 27/30 ----- 1s 3ms/step - accuracy: 0.7597 - loss: 0.5863 - val_accuracy: 0.7327 - val_loss: 0.6295
Epoch 28/30 ----- 1s 3ms/step - accuracy: 0.7616 - loss: 0.5784 - val_accuracy: 0.7432 - val_loss: 0.6398
Epoch 29/30 ----- 1s 3ms/step - accuracy: 0.7665 - loss: 0.5754 - val_accuracy: 0.7372 - val_loss: 0.6368
Epoch 30/30 ----- 1s 3ms/step - accuracy: 0.7691 - loss: 0.5642 - val_accuracy: 0.7367 - val_loss: 0.6386
Epoch 30/30 ----- 2s 4ms/step - accuracy: 0.7629 - loss: 0.5772 - val_accuracy: 0.7307 - val_loss: 0.6274

```

Рисунок 1.3 —Графики обучения (точность/потери в обучающих и валидационных выборках)..

1.5 Оценка модели

Оценивается качество модели на тестовой выборке.

```

test_loss1, test_acc1 = model1.evaluate(X1_test, y1_test_cat)
print(f"Точность на тестовой выборке: {test_acc1:.4f}")

```

1.5 Матрица ошибок и отчёт о классификации

Выводится матрица ошибок и отчёт о метриках классификации.

```

import numpy as np
from sklearn.metrics import confusion_matrix, classification_report

y1_pred = np.argmax(model1.predict(X1_test), axis=1)
print(confusion_matrix(y1_test, y1_pred))
print(classification_report(y1_test, y1_pred, target_names=le.classes_))

```

Вывод:

```

63/63 ----- 0s 3ms/step
[[442  57   3   0]
 [119 299  74  10]
 [ 46  90 351  16]
 [ 10  19  63 410]]
      precision    recall  f1-score   support

     A       0.72       0.88       0.79       502
     B       0.64       0.60       0.62       502
     C       0.71       0.70       0.71       503
     D       0.94       0.82       0.87       502

 accuracy          0.75
 macro avg         0.75
 weighted avg      0.75

```

Рисунок 1.4 —Матрица ошибок и отчет о классификации.

ЧАСТЬ 2: Бинарная классификация на наборе Bank Marketing

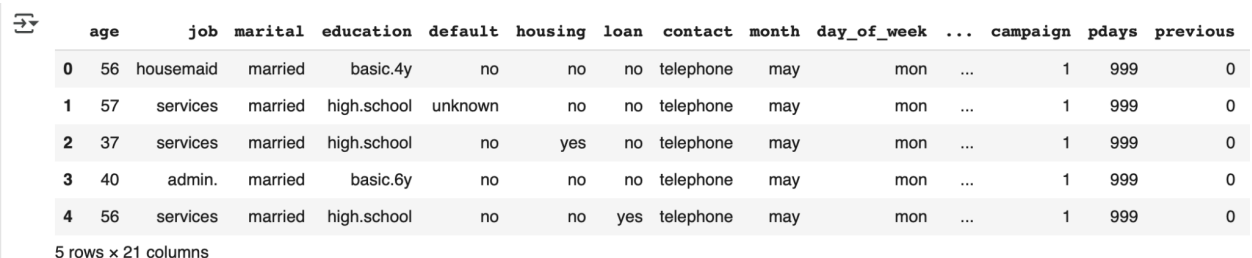
В этой части работы решается задача бинарной классификации: необходимо спрогнозировать, откроет ли клиент банка срочный депозит на основе информации о нём и данных маркетинговой кампании.

2.1 Загрузка и исследование данных

На первом этапе загружается набор данных Bank Marketing и проводится его предварительный просмотр.

```
import pandas as pd
df2 = pd.read_csv('bank-additional-full.csv', sep=';')
df2.head()
```

Вывод:



	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0

5 rows x 21 columns

Рисунок 2.1 — загрузка данных.

2.2 Предобработка данных

Производится кодирование категориальных признаков, нормализация числовых признаков и создание целевой переменной.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer

# Создание целевой переменной
lb = LabelBinarizer()
df2['label'] = lb.fit_transform(df2['y'])

# Кодирование категориальных признаков
df2 = pd.get_dummies(df2.drop(columns=['y']))

X2 = df2.drop(columns=['label'])
y2 = df2['label']
```

```
# Разделение на обучающую, валидационную и тестовую выборки
X2_train, X2_temp, y2_train, y2_temp = train_test_split(X2, y2, test_size=0.3,
stratify=y2, random_state=42)
X2_val, X2_test, y2_val, y2_test = train_test_split(X2_temp, y2_temp,
test_size=0.5, stratify=y2_temp, random_state=42)

# Масштабирование признаков
scaler = StandardScaler()
X2_train = scaler.fit_transform(X2_train)
X2_val = scaler.transform(X2_val)
X2_test = scaler.transform(X2_test)
```

2.3 Создание модели нейронной сети

Построение полносвязной модели для бинарной классификации.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model2 = Sequential([
    Dense(64, activation='relu', input_shape=(X2_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
model2.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model2.summary()
```

Вывод:

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 64)	4,096
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

```
Total params: 6,209 (24.25 KB)
Trainable params: 6,209 (24.25 KB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 2.2 — Архитектура модели 2.

2.4 Обучение модели

Процесс обучения модели с валидацией на отложенной выборке.

```
history2 = model2.fit(X2_train, y2_train, epochs=30, batch_size=32,  
                    validation_data=(X2_val, y2_val), verbose=1)
```

Вывод:

```
Epoch 21/30  
901/901 ————— 4s 2ms/step - accuracy: 0.9474 - loss: 0.1175 - val_accuracy: 0.9079 - val_loss: 0.2191  
Epoch 22/30  
901/901 ————— 2s 2ms/step - accuracy: 0.9490 - loss: 0.1138 - val_accuracy: 0.9097 - val_loss: 0.2255  
Epoch 23/30  
901/901 ————— 3s 3ms/step - accuracy: 0.9501 - loss: 0.1155 - val_accuracy: 0.9081 - val_loss: 0.2252  
Epoch 24/30  
901/901 ————— 5s 3ms/step - accuracy: 0.9523 - loss: 0.1111 - val_accuracy: 0.9110 - val_loss: 0.2346  
Epoch 25/30  
901/901 ————— 3s 3ms/step - accuracy: 0.9530 - loss: 0.1101 - val_accuracy: 0.9040 - val_loss: 0.2377  
Epoch 26/30  
901/901 ————— 5s 2ms/step - accuracy: 0.9555 - loss: 0.1068 - val_accuracy: 0.9069 - val_loss: 0.2409  
Epoch 27/30  
901/901 ————— 5s 5ms/step - accuracy: 0.9556 - loss: 0.1048 - val_accuracy: 0.9058 - val_loss: 0.2453  
Epoch 28/30  
901/901 ————— 4s 3ms/step - accuracy: 0.9563 - loss: 0.1021 - val_accuracy: 0.9061 - val_loss: 0.2507  
Epoch 29/30  
901/901 ————— 3s 4ms/step - accuracy: 0.9575 - loss: 0.0996 - val_accuracy: 0.9053 - val_loss: 0.2533  
Epoch 30/30  
901/901 ————— 6s 6ms/step - accuracy: 0.9611 - loss: 0.0939 - val_accuracy: 0.9051 - val_loss: 0.2593
```

Рисунок 2.3 —Графики обучения

2.5 Оценка модели

Проверка точности модели на тестовой выборке.

```
test_loss2, test_acc2 = model2.evaluate(X2_test, y2_test)  
print(f'Точность на тестовой выборке: {test_acc2:.4f}')
```

2.6 Матрица ошибок и отчёт о классификации

Анализ качества предсказаний модели.

```
from sklearn.metrics import confusion_matrix, classification_report  
import numpy as np  
  
y2_pred = (model2.predict(X2_test) > 0.5).astype("int32")  
print(confusion_matrix(y2_test, y2_pred))  
print(classification_report(y2_test, y2_pred))
```

Вывод:

194/194 — 1s 4ms/step

[[5142 341]
[309 387]]

	precision	recall	f1-score	support
0	0.94	0.94	0.94	5483
1	0.53	0.56	0.54	696
accuracy			0.89	6179
macro avg	0.74	0.75	0.74	6179
weighted avg	0.90	0.89	0.90	6179

Рисунок 2.4 — Матрица ошибок и отчет о классификации.

2.7 Визуализация процесса обучения нейронной сети

Для анализа процесса обучения модели в части 2 были построены графики изменения точности (accuracy) и функции потерь (loss) на обучающей и валидационной выборках по эпохам.

```
import matplotlib.pyplot as plt

# Построение графиков accuracy и loss
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history2.history['accuracy'], label='Training Accuracy')
plt.plot(history2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history2.history['loss'], label='Training Loss')
plt.plot(history2.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Вывод:

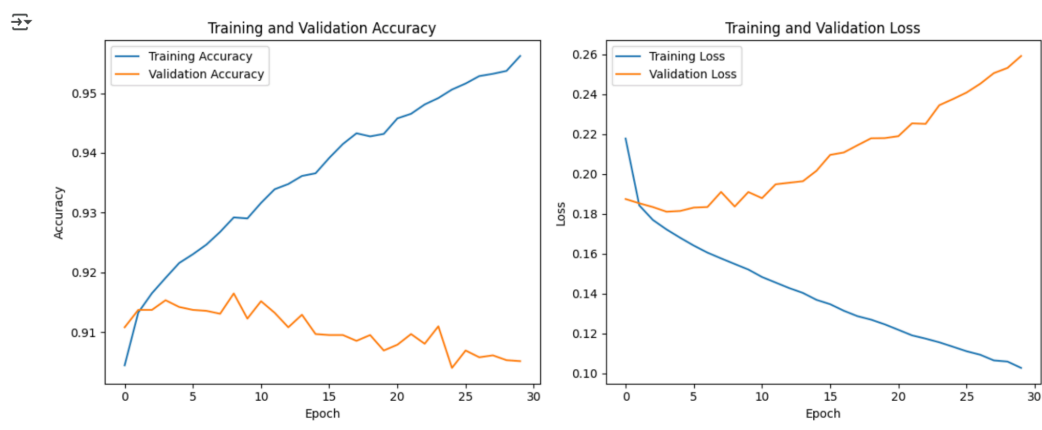


Рисунок 2.5 — Визуализация для обучения модели.

Заключение

В ходе выполнения лабораторной работы была проведена мультиклассовая классификация на датасете Body Performance и бинарная классификация на датасете Bank Marketing. По результатам первой задачи модель достигла точности 75%. Классы A и D распознавались лучше всего, в то время как класс B показал наименьшую точность. Модель продемонстрировала хорошее качество классификации при минимальной переобученности.

Во второй задаче бинарной классификации модель достигла точности 89%, уверенно распознавая отрицательный класс. При этом распознавание положительного класса оказалось менее точным из-за дисбаланса данных. Тем не менее, модель показала хорошую общую сбалансированность между precision и recall. Анализ графиков обучения показал наличие признаков переобучения: наблюдается рост функции потерь на валидационных данных и ухудшение качества валидации после определённого количества эпох. Это свидетельствует о необходимости применения методов регуляризации или ранней остановки для дальнейшего улучшения модели.

Приложение А

[Google_Colab_Link](#)

```
# Импорт библиотек
df2 = pd.read_csv('bank-additional-full.csv', sep=';')

from sklearn.preprocessing import LabelBinarizer

# Создание целевой переменной
lb = LabelBinarizer()
df2['label'] = lb.fit_transform(df2['y'])

# Кодирование категориальных признаков
df2 = pd.get_dummies(df2.drop(columns=['y']))

X2 = df2.drop(columns=['label'])
y2 = df2['label']

# Разделение на обучающую, валидационную и тестовую выборки
X2_train, X2_temp, y2_train, y2_temp = train_test_split(X2, y2, test_size=0.3, stratify=y2,
random_state=42)
X2_val, X2_test, y2_val, y2_test = train_test_split(X2_temp, y2_temp, test_size=0.5,
stratify=y2_temp, random_state=42)

# Нормализация данных
scaler = StandardScaler()
X2_train = scaler.fit_transform(X2_train)
X2_val = scaler.transform(X2_val)
X2_test = scaler.transform(X2_test)

# Создание модели
model2 = Sequential([
    Dense(64, activation='relu', input_shape=(X2_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Обучение модели
history2 = model2.fit(X2_train, y2_train, epochs=30, batch_size=32,
validation_data=(X2_val, y2_val), verbose=1)

# Оценка модели
test_loss2, test_acc2 = model2.evaluate(X2_test, y2_test)
print(f"Test Accuracy: {test_acc2:.4f}")

# Матрица ошибок и отчёт
y2_pred = (model2.predict(X2_test) > 0.5).astype("int32")
print(confusion_matrix(y2_test, y2_pred))
```

```
print(classification_report(y2_test, y2_pred))

# Визуализация обучения
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history2.history['accuracy'], label='Training Accuracy')
plt.plot(history2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history2.history['loss'], label='Training Loss')
plt.plot(history2.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```