

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

**ОТЧЕТ**

**по индивидуальному домашнему заданию**  
**по дисциплине «Искусственные нейронные сети»**  
**Тема: Определение прочности бетона на сжатие**

Студент гр. 8382

\_\_\_\_\_

Нечепуренко Н.А.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2021

## **Цели работы.**

Построить регрессионную модель, позволяющую по заданному набору компонентов в растворе и их количественному содержанию спрогнозировать прочность полученного кубометра бетона.

## **Требования к разрабатываемой модели.**

- Модель должна быть разработана на языке Python с использованием Keras API
- Исходный код проекта должен 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 результатам, 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

## **Выполнение работы.**

Для проведения работы был выбран датасет «Concrete Compressive Strength», доступный по адресу <https://archive.ics.uci.edu/ml/datasets/concrete+compressive+strength>

### *Анализ и обработка исходных данных.*

В наборе данных представлено 1030 наблюдений, каждое из которых описывается следующими характеристиками:

- Cement (цемент), кг в м<sup>3</sup> раствора
- Blast Furnace Slag (доменный шлак), кг в м<sup>3</sup> раствора
- Fly Ash (пепел), кг в м<sup>3</sup> раствора
- Water (вода), кг в м<sup>3</sup> раствора
- Superplasticizer (добавка, позволяющая использовать меньше воды), кг в м<sup>3</sup> раствора
- Coarse Aggregate (гравий), кг в м<sup>3</sup> раствора
- Fine Aggregate (песок вперемешку с очень мелкими камнями), кг в м<sup>3</sup> раствора
- Age (время), в днях [1:365]
- Concrete compressive strength (прочность бетона на сжатие), МПа, изучаемая величина

С помощью библиотеки `pandas` изучим датасет и соберем некоторые описательные статистики для каждого компонента.

С помощью команды `df = pd.read_csv(path)` считаем набор данных в `pandas dataframe`. Посмотрим на первые пять наблюдений с помощью команды `df.head()`.

df.head()

	Cement (component 1)(kg in a m <sup>3</sup> mixture)	Blast Furnace Slag (component 2) (kg in a m <sup>3</sup> mixture)	Fly Ash (component 3)(kg in a m <sup>3</sup> mixture)	Water (component 4)(kg in a m <sup>3</sup> mixture)	Superplasticizer (component 5)(kg in a m <sup>3</sup> mixture)	Coarse Aggregate (component 6) (kg in a m <sup>3</sup> mixture)	Fine Aggregate (component 7) (kg in a m <sup>3</sup> mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075

Рисунок 1 – Первые пять наблюдений датасета

Для начала убедимся, что в датасете нет пропущенных данных с помощью `df.info()`.

```

RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
#   Column                                                                                               Non-
   Null Count  Dtype
---  -
0   Cement (component 1) (kg in a m^3 mixture)                  1030
non-null    float64
1   Blast Furnace Slag (component 2) (kg in a m^3 mixture)     1030
non-null    float64
2   Fly Ash (component 3) (kg in a m^3 mixture)                 1030
non-null    float64
3   Water (component 4) (kg in a m^3 mixture)                   1030
non-null    float64
4   Superplasticizer (component 5) (kg in a m^3 mixture)        1030
non-null    float64
5   Coarse Aggregate (component 6) (kg in a m^3 mixture)        1030
non-null    float64
6   Fine Aggregate (component 7) (kg in a m^3 mixture)          1030
non-null    float64
7   Age (day)                                                    1030

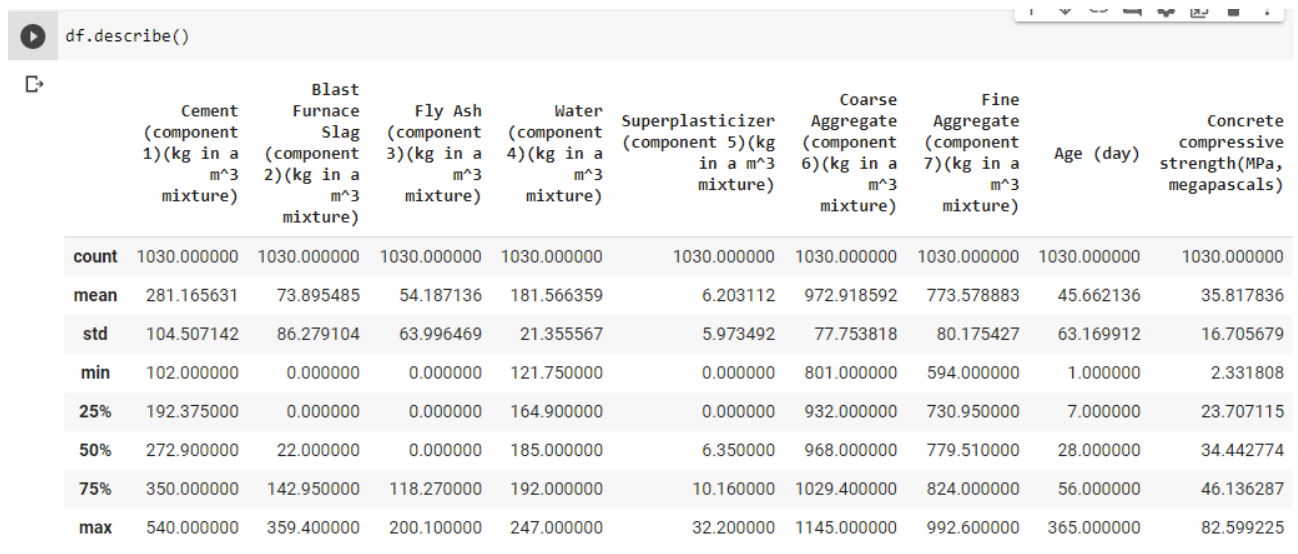
```

```

non-null    int64
8    Concrete compressive strength(MPa, megapascals)    1030
non-null    float64
dtypes: float64(8), int64(1)

```

С помощью вызова `df.describe()` получим данные о среднем значении каждого компонента, стандартном отклонении, минимальном и максимальном значении, а также Q1-Q3 квантили.



	Cement (component 1)(kg in a m <sup>3</sup> mixture)	Blast Furnace Slag (component 2)(kg in a m <sup>3</sup> mixture)	Fly Ash (component 3)(kg in a m <sup>3</sup> mixture)	Water (component 4)(kg in a m <sup>3</sup> mixture)	Superplasticizer (component 5)(kg in a m <sup>3</sup> mixture)	Coarse Aggregate (component 6)(kg in a m <sup>3</sup> mixture)	Fine Aggregate (component 7)(kg in a m <sup>3</sup> mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.165631	73.895485	54.187136	181.566359	6.203112	972.918592	773.578883	45.662136	35.817836
std	104.507142	86.279104	63.996469	21.355567	5.973492	77.753818	80.175427	63.169912	16.705679
min	102.000000	0.000000	0.000000	121.750000	0.000000	801.000000	594.000000	1.000000	2.331808
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.707115
50%	272.900000	22.000000	0.000000	185.000000	6.350000	968.000000	779.510000	28.000000	34.442774
75%	350.000000	142.950000	118.270000	192.000000	10.160000	1029.400000	824.000000	56.000000	46.136287
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.599225

Рисунок 2 – Описательные статистики компонентов

Отделим столбец с зависимой переменной

```

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

```

Заметим, что некоторых компонентов необходимо сравнительно много в кг (таких как цемент, вода, гравий), а других совсем немного (шлак, пел, суперпластификатор). Например, средняя масса цемента примерно 281 кг, в то время как шлака и суперпластификатора примерно 73 и 6 кг соответственно. И дисперсия у этих величин будет отличаться на порядки (дисперсия примерно равна возведенному в квадрат значению в строке `std`). Исходя

из этого, необходимо «перемасштабировать» эти данные, например, медленнорастущей монотонной функцией. Так как компонент может отсутствовать в растворе (например шлак), его значение в наблюдении будет равно нулю, что видно в строке `min`, поэтому выберем функцию  $\log(1 + x)$ .

С помощью функции `train_test_split` разделим датасет на тренировочное и тестовое множества в соотношении 4/1.

```
for column in X.columns:
    X[column] += 1
    X[column] = np.log(X[column])
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.20)
```

Первые пять наблюдений в `X_train` представлены на рисунке ниже.

`X_train.head()`

	Cement (component 1)(kg in a m <sup>3</sup> mixture)	Blast Furnace Slag (component 2)(kg in a m <sup>3</sup> mixture)	Fly Ash (component 3)(kg in a m <sup>3</sup> mixture)	Water (component 4)(kg in a m <sup>3</sup> mixture)	Superplasticizer (component 5)(kg in a m <sup>3</sup> mixture)	Coarse Aggregate (component 6)(kg in a m <sup>3</sup> mixture)	Fine Aggregate (component 7)(kg in a m <sup>3</sup> mixture)	Age (day)
339	5.697630	0.000000	4.775250	5.169347	2.353278	6.931276	6.625989	1.386294
248	5.476673	0.000000	4.555034	5.234632	2.079442	6.857419	6.742892	4.615121
163	5.750348	5.572914	0.000000	5.173321	2.261763	6.954543	6.418039	4.521789
633	5.620401	0.000000	0.000000	5.214936	0.000000	6.993015	6.695799	2.079442
740	5.697093	0.000000	0.000000	5.231109	0.000000	6.947937	6.599870	2.079442

Рисунок 3 – Данные тренировочного множества

С помощью класса `StandardScaler` из библиотеки `sklearn` нормируем входные данные по формуле

$$z = \frac{x - \mu}{\sigma}$$

после этого входные данные будут распределены нормально.

```

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train[:5, :])

```

получаем следующий вывод

```

array([[ 0.33232043, -1.05564555,  1.11186648, -0.2553189 ,
         0.77315492,
         0.64895097, -0.18181869, -1.65203224],
       [-0.24843048, -1.05564555,  1.01916728,  0.28977139,
         0.53558222,
        -0.26799118,  0.93612811,  1.21592172],
       [ 0.4708803 ,  1.24199643, -0.89825645, -0.22213922,
         0.69375908,
         0.93781251, -2.17044715,  1.13302109],
       [ 0.12933554, -1.05564555, -0.89825645,  0.12532078,
        -1.26848059,
         1.41544201,  0.48577312, -1.03635518],
       [ 0.33090962, -1.05564555, -0.89825645,  0.26035474,
        -1.26848059,
         0.85579379, -0.43158984, -1.03635518]])

```

Данные подготовлены для обучения моделей.

### ***Разработка и обучение моделей.***

Поставленная задача относится к классу регрессионных задач. Подобная задача решалась в лабораторной работе номер 3, поэтому базовая архитектура модели была выбрана аналогично модели в ЛР №3.

```

relu2l = Sequential()
relu2l.add(Dense(64, activation='relu', input_shape=(X_train.
    shape[1],)))
relu2l.add(Dense(64, activation='relu'))
relu2l.add(Dense(1))

```

```
relu2l.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

модель содержит два полносвязных слоя на 64 нейрона с функцией активации `relu` и линейным выходным нейроном для получения непрерывного значения.

В качестве оптимизатора будем использовать `adam`, функция потерь `mse`

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

в качестве метрики `mae`

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \tilde{y}_i|$$

Обучим ее со следующими параметрами:

```
h1 = relu2l.fit(
    X_train, y_train, epochs=100, batch_size=1,
    validation_split=0.2, verbose=1, callbacks=[
        tensorboard_callback,
        ModelCheckpoint("relu2l" + filepath, monitor='mae',
            verbose=1, save_best_only=True, mode='min'),
    ]
)
```

`callback`'и будут описаны позже.

На случайно разделенном на тренировочные и тестовые данные датасете модель показала результат  $mae = 3.6351$ . Это не надежная оценка, о надежности оценки и применении кросс-валидации написано в следующем разделе. Просто зафиксируем текущее разбиение и обучим несколько моделей, сравнивая, насколько они точно предугадывают значение целевой величины относительно друг друга.

Рассмотрим вторую модель, в которую добавим скрытый слой на 96 нейронов, веса которых проинициализируем с помощью `glorot_normal`.



```

relu3l = Sequential()
relu3l.add(Dense(64, activation='relu', input_shape=(X_train.
    shape[1],), kernel_initializer='glorot_normal'))
relu3l.add(Dense(96, activation='relu', kernel_initializer='
    glorot_normal'))
relu3l.add(Dense(64, activation='relu', kernel_initializer='
    glorot_normal'))
relu3l.add(Dense(1))
relu3l.compile(optimizer='adam', loss='mse', metrics=['mae'])

```

Обучим ее с теми же параметрами и получим  $mae = 3.5711$ . Результат на тестовых данных улучшился, но не сопоставимо с изменением архитектуры модели.

Рассмотрим метод решения задачи регрессии не связанный с ИНС. Рассмотрим `RandomForestRegressor` из библиотеки `sklearn`. Это лес решающих деревьев, по умолчанию состоящий из сотни. У него можно варьировать множество параметров, причем сделать это можно с помощью `GridSearchCV`. В работе будем использовать `RandomForestRegressor` с параметрами по умолчанию.

```

random_forest = RandomForestRegressor()
random_forest.fit(X_train, y_train)
print(mean_absolute_error(y_test, random_forest.predict(
    X_test)))

```

Получаем на тех же данных  $mae = 3.2712$ , что значительно лучше моделей с использованием ИНС. Возможно, что для решения данной задачи решающие леса подходят лучше, либо были рассмотрены неудачные архитектуры ИНС. Лес решающих деревьев помимо лучшей точности обучается гораздо быстрее.

### ***Ансамблирование моделей.***

Проведем ансамблирование рассмотренных моделей с целью улучшения предсказаний. Интуитивно хочется дать чуть больший вес решающему лесу. Для моделей с использованием ИНС зададим примерно равные коэффициенты. Функция получения результата ансамбля приведена ниже.

```
def get_ensembled():
    return 0.5 * random_forest.predict(X_test) + 0.2 * relu21
        .predict(X_test)[: , 0] + 0.3 * relu31.predict(X_test)
        [: , 0]
```

На рассмотренных выше данных результат ансамбля оказался гораздо лучше, чем просто результат решающего леса,  $mae = 3.0133$ .

```
print(relu21.evaluate(X_test, y_test))
print(relu31.evaluate(X_test, y_test))
print(mean_absolute_error(y_test, random_forest.predict(
    X_test)))
print(mean_absolute_error(y_test, get_ensembled()))
```

#### **с выводом**

```
7/7 [=====] - 0s 3ms/step - loss:
    24.5908 - mae: 3.6352
[24.590801239013672, 3.63517165184021]
7/7 [=====] - 0s 3ms/step - loss:
    23.4865 - mae: 3.5711
[23.486515045166016, 3.571138620376587]
3.2712339184814683
3.0133112764115593
```

Опять же, это не надежный показатель, потому что случайный сплит функции `train_test_split` мог оказаться либо удачным, либо неудачным. Но сам факт того, что ансамблирование моделей с использованием ИНС и моделей, использующих другие математические подходы, дает лучшие ре-

зультаты, нежели просто использование их по-отдельности, является неплохой гипотезой. Проверим ее в разделе, посвященному кросс-валидации.

### ***Использование Keras Callback.***

Для сбора статистики по ходу обучения модели воспользуемся коллбэком TensorBoard. Также для сохранения весов модели, при которых достиглось наименьшее `mae` ходе обучения, воспользуемся `ModelCheckpoint`.

В коде это выглядит так:

```
tensorboard_callback = TensorBoard(log_dir="./logs",
    histogram_freq=1)
# %load_ext tensorboard
# !rm -rf ./logs/
...
callbacks=[
    tensorboard_callback,
    ModelCheckpoint("relu31" + filepath, monitor='mae',
        verbose=1, save_best_only=True, mode='min')
]
```

### ***Оценка качества и кросс-валидация.***

Выше уже упоминалось, что полученные результаты ненадежны и зависят от того, как данные были перемешаны и разделены функцией разбиения. Более надежную оценку позволяет получить кросс-валидация. Можно использовать классический подход, при котором сначала данные перемешиваются, а потом в цикле перебираются блоки тестовых данных. В данной работе реализуем кросс-валидацию немного по-другому. Так как `train_test_split` разбивает данные случайно, но в фиксированной пропорции с 20% тестовых данных, можно просто запустить программу 5 раз, записать результаты, а потом усреднить их.

В данных о запуске получаются следующим образом

```
print(relu2l.evaluate(X_test, y_test))
print(relu3l.evaluate(X_test, y_test))
print(mean_absolute_error(y_test, random_forest.predict(
    X_test)))
print(mean_absolute_error(y_test, get_ensembled()))
```

**Первый запуск:**

```
7/7 [=====] - 0s 2ms/step - loss:
    24.9161 - mae: 3.7003
[24.916112899780273, 3.7003490924835205]
7/7 [=====] - 0s 2ms/step - loss:
    17.7427 - mae: 3.1191
[17.742652893066406, 3.1190884113311768]
3.4922293781571194
2.978705184585904
```

График динамики MAE второй модели, полученный с использованием TensorBoard, приведен на рисунке ниже

epoch\_mae  
tag: epoch\_mae

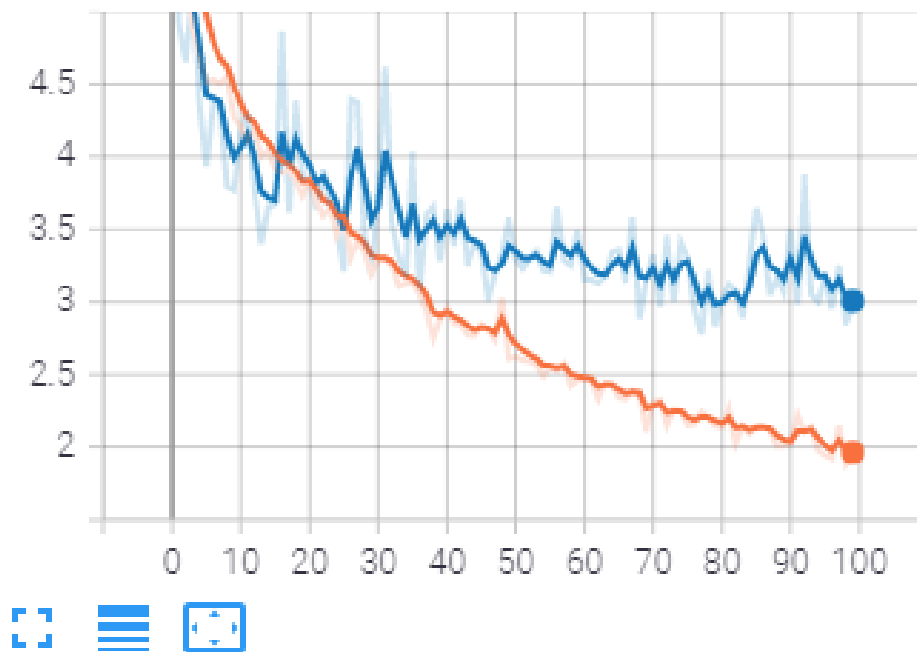


Рисунок 4 – График mae второй модели

Модель немного переобучилась, хотя в конце все равно достигла оптимальных значений `val_mae`.

Второй запуск:

```
7/7 [=====] - 0s 2ms/step - loss:
      33.8612 - mae: 3.7521
[33.86116027832031, 3.752112865447998]
7/7 [=====] - 0s 3ms/step - loss:
      32.3284 - mae: 3.5929
[32.32839584350586, 3.5929012298583984]
3.719320858469332
3.2111038613283864
```

Третий запуск:

```
7/7 [=====] - 0s 2ms/step - loss:
      23.4703 - mae: 3.5142
```

```
[23.47028923034668, 3.5141773223876953]
7/7 [=====] - 0s 2ms/step - loss:
      23.3283 - mae: 3.6826
[23.328298568725586, 3.682640552520752]
3.304281311628491
2.8608690783688346
```

#### Четвертый запуск:

```
7/7 [=====] - 0s 2ms/step - loss:
      30.8526 - mae: 3.7865
[30.85256004333496, 3.7865188121795654]
7/7 [=====] - 0s 2ms/step - loss:
      25.2605 - mae: 3.2373
[25.260467529296875, 3.237285614013672]
3.178326676188121
2.8622070232315875
```

#### Пятый запуск:

```
7/7 [=====] - 0s 2ms/step - loss:
      31.3261 - mae: 3.7417
[31.326065063476562, 3.741673231124878]
7/7 [=====] - 0s 2ms/step - loss:
      31.3487 - mae: 3.6371
[31.348711013793945, 3.637115240097046]
3.7255232919464554
3.279796148904787
```

#### Получаем следующие результаты:

- Средний MAE первой модели: 3.6989
- Средний MAE второй модели: 3.4538
- Средний MAE случайного леса: 3.4839
- Средний MAE ансамбля (0.2, 0.3, 0.5): 3.0385

### *Анализ полученных результатов.*

По результатам усреднения было установлено, что модель с тремя скрытыми слоями в среднем точна как случайный лес с параметрами по умолчанию, но обучение леса происходит во много раз быстрее. MAE ансамбля при каждом запуске получался лучше, чем точность отдельных моделей. Средний MAE ансамбля с выбранными коэффициентами оказался равным 3.0385.

#### Решенные проблемы:

- Входные данные были изучены, перемасштабированы и нормализованы.
- Были реализованы модели на основе ИНС, позволяющие решить задачу регрессии.
- Была изучена модель, использующая лес решающих деревьев.

#### Возникшие проблемы:

- Для датасета с 1030 наблюдениями необходимо проводить кросс-валидацию, чтобы получать более надежные результаты.
- Для решения данной задачи не-ИНС модели обучаясь гораздо быстрее дают такую же, либо более высокую точность предсказаний.

#### Нерешенные проблемы:

- Не была найдена архитектура ИНС, при которой модель имела бы точность лучше, чем не-ИНС модель, при адекватном числе параметров и времени обучения.
- Кросс-валидация не была проведена в классическом варианте, осуществлялись последовательные запуски, что можно было бы автоматизировать.

#### Возможные улучшения:

- Рассмотреть другие не-ИНС подходы к решению задачи регрессии.
- Добавить больше моделей к ансамблю и точнее настроить весовые ко-

эффиценты.

### **Выводы.**

В результате выполнения работы была решена задача регрессии, были описаны модели, позволяющие предсказать прочность бетона по его составу и времени затвердевания. Было произведено ансамблирование полученных моделей. Было установлено, что ансамбль моделей дает более точные результаты, нежели модели по-отдельности. Были сохранены конфигурации моделей с целью повторного использования.



## ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import TensorBoard,
    ModelCheckpoint
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
import pickle

#read and analyze
df = pd.read_csv('sample_data/Concrete Compressive Strength.csv')
# df.head()
# df.describe()
# df.info()

#split to x and y
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

#split to train and test; to normal
for column in X.columns:
    X[column] += 1
    X[column] = np.log(X[column])

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.20)
print(X_train.head())
```

```

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train[:5, :])

#configure callbacks
# %load_ext tensorboard
# !rm -rf ./logs/
filepath = "-{epoch:02d}-{mae:.4f}.hdf5"
tensorboard_callback = TensorBoard(log_dir="./logs",
    histogram_freq=1)

#first model
relu2l = Sequential()
relu2l.add(Dense(64, activation='relu', input_shape=(X_train.
    shape[1],)))
relu2l.add(Dense(64, activation='relu'))
relu2l.add(Dense(1))
relu2l.compile(optimizer='adam', loss='mse', metrics=['mae'])
h1 = relu2l.fit(
    X_train, y_train, epochs=100, batch_size=1,
    validation_split=0.2, verbose=1, callbacks=[
        tensorboard_callback,
        ModelCheckpoint("relu2l" + filepath, monitor='mae',
            verbose=1, save_best_only=True, mode='min'),
    ]
)
print(relu2l.evaluate(X_test, y_test))

#second model
relu3l = Sequential()

```

```

relu3l.add(Dense(64, activation='relu', input_shape=(X_train.
    shape[1],), kernel_initializer='glorot_normal'))
relu3l.add(Dense(96, activation='relu', kernel_initializer='
    glorot_normal'))
relu3l.add(Dense(64, activation='relu', kernel_initializer='
    glorot_normal'))
relu3l.add(Dense(1))
relu3l.compile(optimizer='adam', loss='mse', metrics=['mae'])
h2 = relu3l.fit(
    X_train, y_train, epochs=100, batch_size=1,
    validation_split=0.2, verbose=1, callbacks=[
        tensorboard_callback,
        ModelCheckpoint("relu3l" + filepath, monitor='mae',
            verbose=1, save_best_only=True, mode='min')
    ]
)
print(relu3l.evaluate(X_test, y_test))

# non-ann model
random_forest = RandomForestRegressor()
random_forest.fit(X_train, y_train)
print(mean_absolute_error(y_test, random_forest.predict(X_test)))
# with open('rf.pickle', 'wb') as f:
#     pickle.dump(random_forest, f)

def get_ensembled():
    return 0.5 * random_forest.predict(X_test) + 0.2 * relu2l.
        predict(X_test)[: , 0] + 0.3 * relu3l.predict(X_test)[: , 0]

# relu2l.load_weights("relu2l-93-2.2260.hdf5")

```

```
# relu3l.load_weights("relu3l-99-1.8773.hdf5")
# with open('rf.pickle', 'rb') as f:
#     random_forest = pickle.load(f)

#evaluation
print(relu2l.evaluate(X_test, y_test))
print(relu3l.evaluate(X_test, y_test))
print(mean_absolute_error(y_test, random_forest.predict(X_test)))
print(mean_absolute_error(y_test, get_ensembled()))

# %tensorboard --logdir logs/
```