МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | В. Ю. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
| классификация табличных данных на основе нейросетевых моделей |
| по курсу: интеллектуальный анализ данных на основе методов машинного обучения |
|  |
|  |

РАБОТУ ВЫПОЛНИЛА

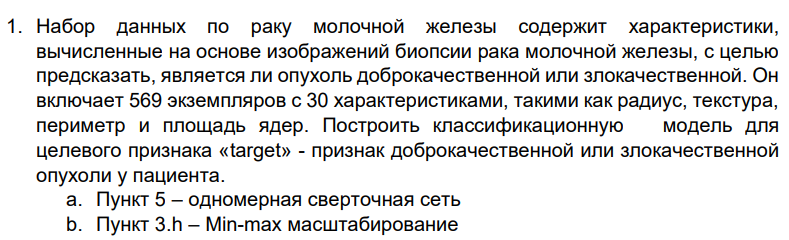
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4136 |  |  |  | Н.С. Бобрович |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

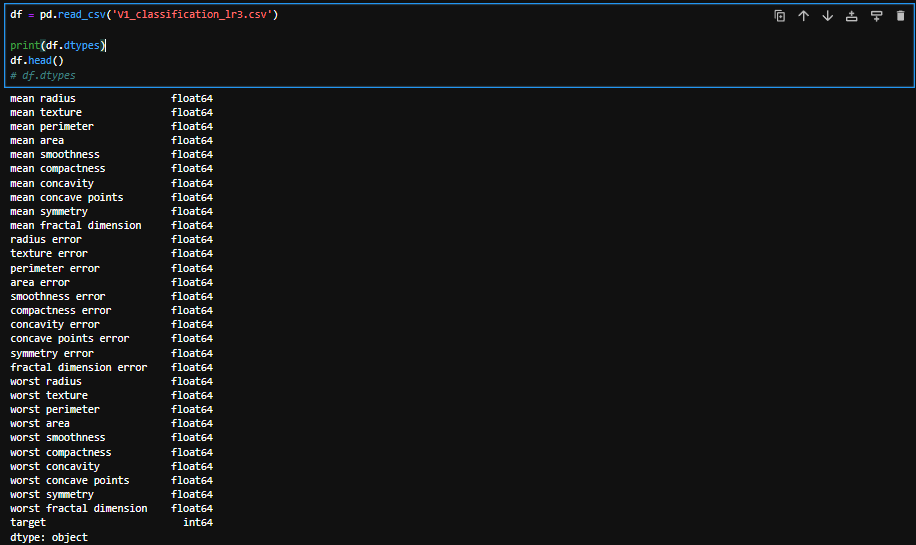
1. **Цель работы**

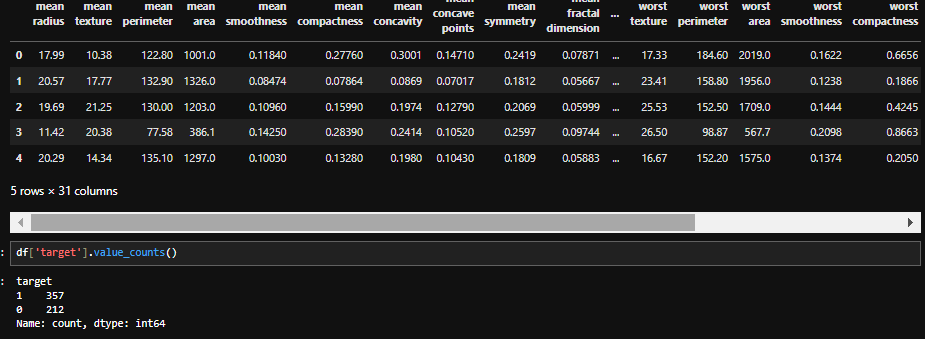
Необходимо выполнить классификационный анализ данных по указанному целевому признаку

**Вариант 1:**

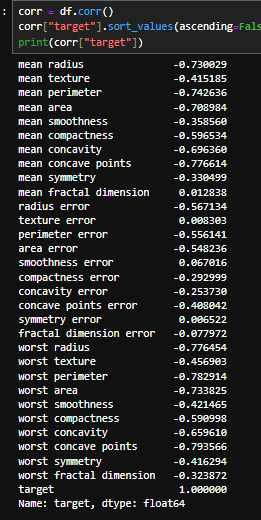


1. **Ход работы**

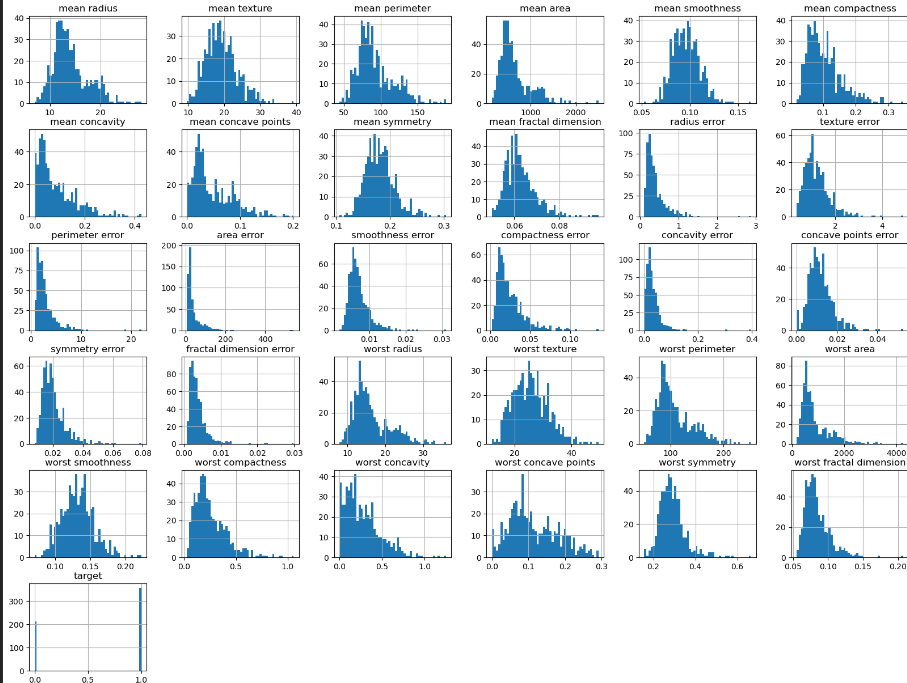


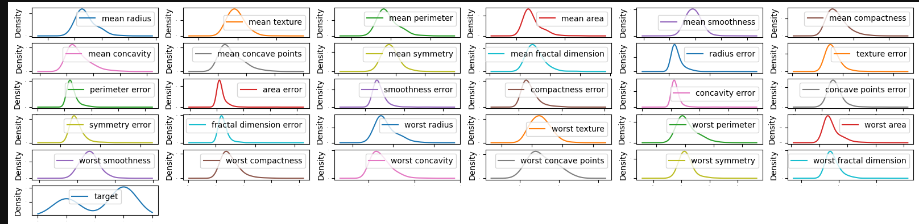


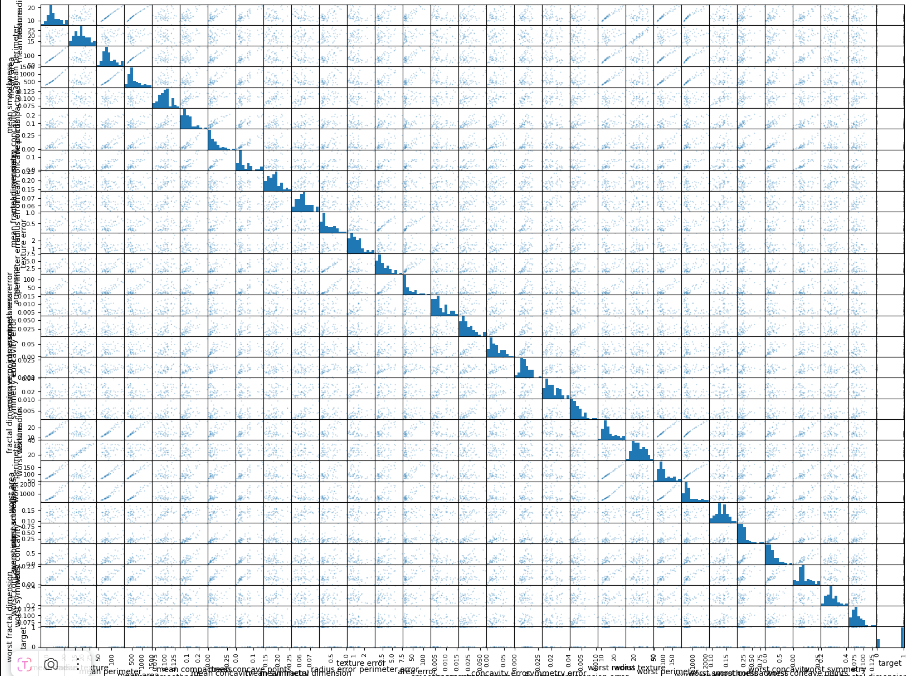
Выполним анализ корреляционной зависимости для целевого признака (target)



Построим гистограммы распределения и плотнотей и зависимостей признаков

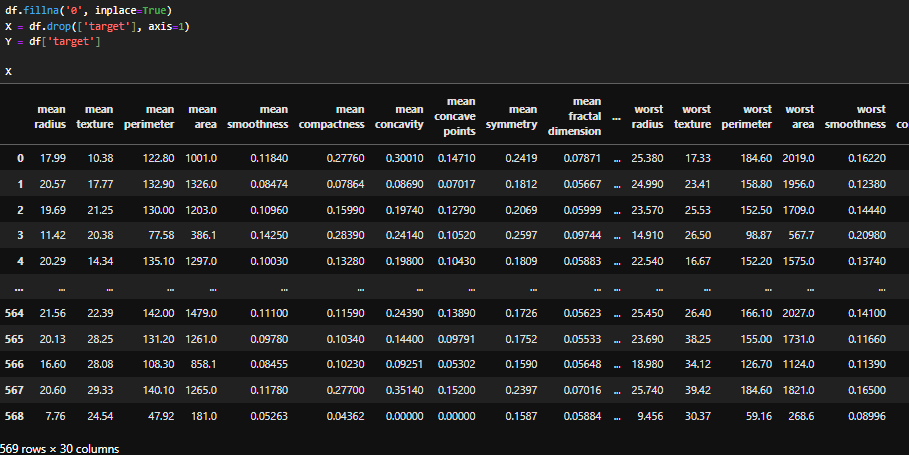






Как можно заметить, признаки нормально распределены

Разделим данные на входные и выходные значения



Создадим набор с дополнительным признаком, представляющим собой сумму части негативных величин

X\_extra = X

Y\_extra = Y

X\_extra["worst"] = X\_extra["worst smoothness"] + X\_extra["worst concavity"] + X\_extra["worst concave points"] + X\_extra["worst fractal dimension"]

X\_with\_new\_atribbute = df.drop(['target'], axis=1)

Y\_with\_new\_atribbute = df['target']

Используем масштабирование для обоих наборов данных.

scaler = MinMaxScaler(feature\_range = (0, 1))

X\_scaled = scaler.fit\_transform(X)

Y = np.array(Y)

Y = Y.reshape(-1, Y.shape[0])

Y\_scaled = Y

X\_with\_new\_atribbute\_scaled = scaler.fit\_transform(X\_with\_new\_atribbute)

Y\_with\_new\_atribbute = np.array(Y\_with\_new\_atribbute)

Y\_with\_new\_atribbute = Y\_with\_new\_atribbute.reshape(-1, Y\_with\_new\_atribbute.shape[0])

Y\_with\_new\_atribbute\_scaled = Y\_with\_new\_atribbute

Разобъём выборку на обучающую, тестовую и валидационную в отношении 60/20/20 %

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scaled, Y\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=42)

Создадим одномерную сверточная сеть и полносвязную модель для последующего обучения

model\_dense = Sequential()

model\_dense.add(Input(shape=(X\_train.shape[1],)))

model\_dense.add(Dense(128, activation='relu'))

model\_dense.add(Dense(64, activation='relu'))

model\_dense.add(Dense(1, activation='sigmoid'))

model = Sequential()

model.add(Input(shape=(1, X\_train.shape[1])))

model.add(Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=32)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=32)

model\_dense\_2 = Sequential()

model\_dense\_2.add(Input(shape=(X\_train.shape[1],)))

model\_dense\_2.add(Dense(128, activation='relu'))

model\_dense\_2.add(Dense(64, activation='relu'))

model\_dense\_2.add(Dense(1, activation='sigmoid'))

model\_2 = Sequential()

model\_2.add(Input(shape=(1, X\_train.shape[1])))

model\_2.add(Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'))

model\_2.add(MaxPooling1D(pool\_size=1))

model\_2.add(Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'))

model\_2.add(MaxPooling1D(pool\_size=1))

model\_2.add(Flatten())

model\_2.add(Dense(64, activation='relu'))

model\_2.add(Dense(1, activation='sigmoid'))

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scaled, Y\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=42)

labels = [0, 1]

X\_test.shape

model\_dense.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model\_dense.fit(X\_train, Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = model\_dense.evaluate(X\_test, Y\_test)

predicted = model\_dense.predict(X\_test)

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Dense, stand mse: {mse[0]}, balansed accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

mse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Conv, stand mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=42)

model\_dense\_2.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model\_dense\_2.fit(X\_train, Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = model\_dense\_2.evaluate(X\_test, Y\_test)

predicted = model\_dense\_2.predict(X\_test)

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Dense, stand, extra mse: {mse[0]}, accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

model\_2.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model\_2.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

mse = model\_2.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model\_2.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Conv, stand, extra mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=42)

model\_dense.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model\_dense.fit(X\_train, Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = model\_dense.evaluate(X\_test, Y\_test)

predicted = model\_dense.predict(X\_test)

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Dense mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

mse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Conv mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_with\_new\_atribbute, Y\_with\_new\_atribbute, test\_size=0.4, random\_state=42)

X\_test, X\_dev, Y\_test, Y\_dev = train\_test\_split(X\_test, Y\_test, test\_size=0.5, random\_state=42)

model\_dense\_2.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model\_dense\_2.fit(X\_train, Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(X\_dev, Y\_dev))

mse = model\_dense\_2.evaluate(X\_test, Y\_test)

predicted = model\_dense\_2.predict(X\_test)

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Dense, extra mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')

model\_2.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

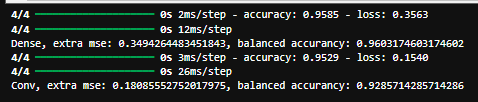
history = model\_2.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

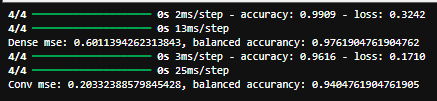
mse = model\_2.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

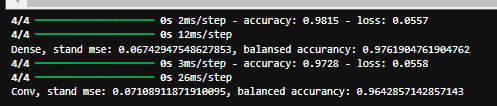
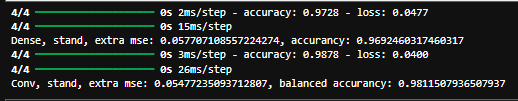
predicted = model\_2.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

print(f'Conv, extra mse: {mse[0]}, balanced accurancy: {balanced\_accuracy\_score(Y\_test, predicted\_classes)}')







Лучший результат показала сверточная модель c масштабированием и с дополнительными признаками. Выполним Grid поиск гиперпараметров для неё

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

Deg = [2,3,4,5,6,7]

res = []

res\_r2 = []

names = []

for deg in Deg:

model = Sequential()

model.add(Input(shape=(1, X\_train.shape[1])))

model.add(Conv1D(filters=2\*\*deg, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Conv1D(filters=2\*\*(deg-1), kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

rmse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

acc = balanced\_accuracy\_score(Y\_test, predicted\_classes)

res.append(rmse)

res\_r2.append(acc)

names.append(deg)

print("Deg = ", deg)

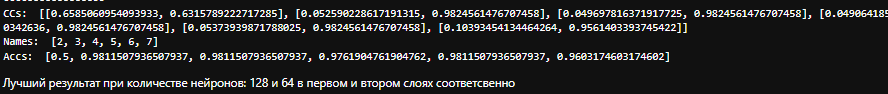
print("CC = ", rmse)

print("Acc = ", acc, "\n-----------------")

print ("CCs: ", res)

print ("Names: ", names)

print ("Accs: ", res\_r2)



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

Epohs = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

res = []

res\_r2 = []

names = []

for ep in Epohs:

model = Sequential()

model.add(Input(shape=(1, X\_train.shape[1])))

model.add(Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=ep, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

rmse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

acc = balanced\_accuracy\_score(Y\_test, predicted\_classes)

res.append(rmse)

res\_r2.append(acc)

names.append(ep)

print("Epohs = ", ep)

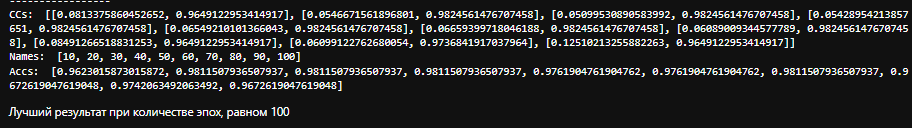
print("CC = ", rmse)

print("Acc = ", acc, "\n-----------------")

print ("CCs: ", res)

print ("Names: ", names)

print ("Accs: ", res\_r2)



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

Opts = ['adam', 'SGD', 'Adamax']

res = []

res\_r2 = []

names = []

for op in Opts:

model = Sequential()

model.add(Input(shape=(1, X\_train.shape[1])))

model.add(Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=op, loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=50, batch\_size=32, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

rmse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

acc = balanced\_accuracy\_score(Y\_test, predicted\_classes)

res.append(rmse)

res\_r2.append(acc)

names.append(op)

print("Ops = ", op)

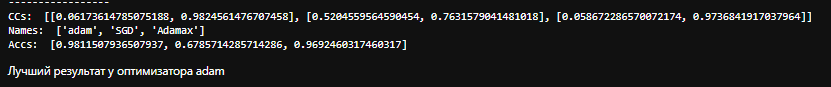
print("CC = ", rmse)

print("Acc = ", acc, "\n-----------------")

print ("CCs: ", res)

print ("Names: ", names)

print ("Accs: ", res\_r2)



Итоговая сверточная модель без стандартизации и датасетом с дополнительным признаком и следующими гиперпараметрами:

- 128, 64 нейронов в первом и втором слоях соответсвенно

- Оптимизатор Adam

- Количество эпох 100

Выполним прогнозирования для модели с лучшими параметрами:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_with\_new\_atribbute\_scaled, Y\_with\_new\_atribbute\_scaled, test\_size=0.4, random\_state=42)

X\_test, X\_dev, y\_test, y\_dev = train\_test\_split(X\_test, y\_test, test\_size=0.5, random\_state=42)

model = Sequential()

model.add(Input(shape=(1, X\_train.shape[1])))

model.add(Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'))

model.add(MaxPooling1D(pool\_size=1))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=op, loss='binary\_crossentropy', metrics=['accuracy'])

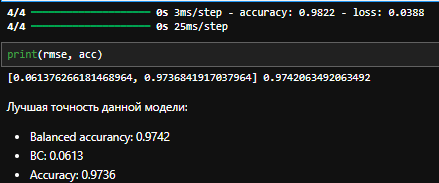
history = model.fit(np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1])), Y\_train, epochs=100, batch\_size=4, verbose=0, validation\_data=(np.reshape(X\_dev, (X\_dev.shape[0], 1, X\_dev.shape[1])), Y\_dev))

rmse = model.evaluate(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])), Y\_test)

predicted = model.predict(np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1])))

predicted\_classes = (predicted > 0.5).astype(int)

acc = balanced\_accuracy\_score(Y\_test, predicted\_classes)



1. **Выводы**

В ходе лабораторной работы была выбрана модель, лучше всего классифицирующая данные на 2 класса.