объясни код # Регрессия

df = pd.read\_csv("diamonds.csv", index\_col=0)

df = pd.get\_dummies(df, columns=['cut','color','clarity'])

display(df)

y = df["price"]

X = df.drop(["price"], axis=1)

X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X, y, test\_size=0.2)

# Классификация

data = pd.read\_csv("airlines\_task.csv", index\_col=0).drop\_duplicates()

data.loc[:, ["Airline", "AirportFrom", "AirportTo", "Flight"]].nunique(axis=0)

data.dropna(inplace=True)

data\_encoded = pd.get\_dummies(data, columns=["Airline"], dtype=int).reset\_index(drop=True)

routes = pd.Series(data=(data\_encoded.AirportTo + data\_encoded.AirportFrom), name="Route")

routes

data\_encoded: pd.DataFrame = pd.concat([data\_encoded, routes], axis=1).drop(["AirportTo", "AirportFrom"], axis=1)

data\_encoded.rename({0: "Route"}, axis=1, inplace=True)

data\_encoded.Route.value\_counts()

label\_map = dict(zip(data\_encoded.Route.unique().tolist(), np.arange(0, data\_encoded.Route.unique().shape[0])))

label\_map

data\_encoded.Route

data\_encoded.Route = data\_encoded.Route.map(lambda x: label\_map.get(x))

data\_encoded.Route

df = data\_encoded

display(df)

y = df['Delay']

X = df.drop(['Delay'], axis=1)

underSampler = RandomUnderSampler(sampling\_strategy='majority')

X\_under\_sample, y\_under\_sample = underSampler.fit\_resample(X, y)

X\_train\_clf, X\_test\_clf, y\_train\_clf, y\_test\_clf = train\_test\_split(X\_under\_sample, y\_under\_sample, test\_size=0.2, random\_state = 4)

scaler = StandardScaler()

scaler.fit(X\_train\_clf, y\_train\_clf)

X\_train\_std\_clf = scaler.transform(X\_train\_clf)

X\_test\_std\_clf = scaler.transform(X\_test\_clf)

Этот код содержит две части: регрессию и классификацию.

В первой части кода происходит регрессия. Сначала загружается набор данных diamonds.csv и создается фиктивная переменная для столбцов cut, color и clarity. Затем целевая переменная y устанавливается равной столбцу price, а переменная X устанавливается равной всему датафрейму, кроме столбца price. Данные разделяются на обучающую и тестовую выборки с помощью функции train\_test\_split.

Во второй части кода происходит классификация. Загружается набор данных airlines\_task.csv, удаляются дубликаты и пропущенные значения. Создается фиктивная переменная для столбца Airline. Создается новый столбец Route, который является комбинацией столбцов AirportTo и AirportFrom. Столбцы AirportTo и AirportFrom удаляются. Затем данные разделяются на обучающую и тестовую выборки с помощью функции train\_test\_split. Данные масштабируются с помощью класса StandardScaler.

Вы хотите узнать что-то еще об этом коде?

2из20

объясни код model\_regression = tf.keras.Sequential(

[

tf.keras.layers.Input(shape=(26,)),

tf.keras.layers.Dense(256, activation="relu"),

tf.keras.layers.Dense(512, activation="relu"),

tf.keras.layers.Dropout(0.3),

tf.keras.layers.Dense(128, activation="relu"),

tf.keras.layers.Dropout(0.05),

tf.keras.layers.Dense(64, activation="relu"),

tf.keras.layers.Dropout(0.01),

tf.keras.layers.Dense(1, activation="linear")

]

)

model\_regression.summary()

Этот код создает модель регрессии с использованием библиотеки TensorFlow. Модель является последовательной и содержит несколько слоев. Первый слой является входным слоем с формой (26,). Следующие слои являются полносвязными слоями с различным количеством нейронов и функциями активации relu. Между некоторыми слоями есть слои Dropout, которые используются для предотвращения переобучения. Последний слой является полносвязным слоем с одним нейроном и линейной функцией активации. Затем вызывается метод summary для отображения информации о модели.

Вы хотите узнать что-то еще об этом коде?

3из20

объясни код

model\_regression.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.005), loss="mae")

model\_regression.fit(X\_train\_reg, y\_train\_reg, epochs=15)

Этот код компилирует модель регрессии, созданную ранее. Оптимизатором является Adam с скоростью обучения 0.005. Функция потерь установлена как mae (средняя абсолютная ошибка). Затем модель обучается на обучающей выборке X\_train\_reg и y\_train\_reg в течение 15 эпох.

Вы хотите узнать что-то еще об этом коде?

4из20

объясни код y\_pred\_reg = model\_regression.predict(X\_test\_reg)

print(mean\_absolute\_error(y\_test\_reg, y\_pred\_reg))

print(mean\_squared\_error(y\_test\_reg, y\_pred\_reg))

print(r2\_score(y\_test\_reg, y\_pred\_reg))

print(max\_error(y\_test\_reg, y\_pred\_reg))

Этот код использует обученную ранее модель регрессии для предсказания значений на тестовой выборке X\_test\_reg. Затем вычисляются различные метрики качества предсказаний: средняя абсолютная ошибка (mean\_absolute\_error), среднеквадратичная ошибка (mean\_squared\_error), коэффициент детерминации (r2\_score) и максимальная ошибка (max\_error).

Вы хотите узнать что-то еще об этом коде?

5из20

объясни код model\_classification = tf.keras.Sequential(

[

tf.keras.layers.Input(shape=(23,)),

tf.keras.layers.Dense(128, activation="relu"),

tf.keras.layers.Dense(256, activation="relu"),

tf.keras.layers.Dense(1024, activation="tanh"),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(512, activation="tanh"),

tf.keras.layers.Dropout(0.01),

tf.keras.layers.Dense(128, activation="tanh"),

tf.keras.layers.Dense(64, activation="tanh"),

tf.keras.layers.Dense(32, activation="tanh"),

tf.keras.layers.Dense(16, activation="tanh"),

tf.keras.layers.Dense(1, activation="sigmoid")

]

)

model\_classification.summary()

Этот код создает модель классификации с использованием библиотеки TensorFlow. Модель является последовательной и содержит несколько слоев. Первый слой является входным слоем с формой (23,). Следующие слои являются полносвязными слоями с различным количеством нейронов и функциями активации relu и tanh. Между некоторыми слоями есть слои Dropout, которые используются для предотвращения переобучения. Последний слой является полносвязным слоем с одним нейроном и функцией активации sigmoid. Затем вызывается метод summary для отображения информации о модели.

Вы хотите узнать что-то еще об этом коде?

6из20

объясни код model\_classification.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.00001), loss="binary\_crossentropy")

model\_classification.fit(X\_train\_std\_clf, y\_train\_clf, epochs=15)

Этот код компилирует модель классификации, созданную ранее. Оптимизатором является Adam. Функция потерь установлена как binary\_crossentropy (бинарная перекрестная энтропия). Затем модель обучается на стандартизированной обучающей выборке X\_train\_std\_clf и y\_train\_clf в течение 15 эпох.

Вы хотите узнать что-то еще об этом коде?

7из20

объясни код model\_classification.predict(X\_test\_std\_clf, verbose=None)[:5]

y\_pred = np.around(model\_classification.predict(X\_test\_std\_clf, verbose=None))

print(classification\_report(y\_test\_clf, y\_pred))

print(confusion\_matrix(y\_test\_clf, y\_pred))

Этот код использует обученную ранее модель классификации для предсказания значений на стандартизированной тестовой выборке X\_test\_std\_clf. Затем предсказанные значения округляются до ближайшего целого числа с помощью функции np.around. Вычисляются различные метрики качества классификации: отчет о классификации (classification\_report) и матрица ошибок (confusion\_matrix).

Вы хотите узнать что-то еще об этом коде?

8из20

объясни код class Activation:

def callFunction(x):

pass

def callDerivative(x):

pass

class Relu(Activation):

@staticmethod

def callFunction(x):

return np.maximum(0,x)

@staticmethod

def callDerivative(x):

return np.where(x > 0, 1.0, 0.0)

class Sigmoid(Activation):

@staticmethod

def callFunction(x):

return 1 / (1 + np.exp(-x))

@staticmethod

def callDerivative(x):

return Sigmoid.callFunction(x) \* (1- Sigmoid.callFunction(x))

class TangH(Activation):

@staticmethod

def callFunction( x):

return (np.exp(x) - np.exp(-x)) / (np.exp(x)+np.exp(-x))

@staticmethod

def callDerivative(x):

return 1 - TangH.callFunction(x) \*\* 2

Этот код определяет класс Activation и три его подкласса: Relu, Sigmoid и TangH. Класс Activation содержит два метода: callFunction и callDerivative, которые не реализованы. Подклассы Relu, Sigmoid и TangH переопределяют эти методы для реализации соответствующих функций активации и их производных.

Класс Relu реализует функцию активации ReLU (Rectified Linear Unit), которая возвращает максимум между 0 и входным значением. Производная этой функции равна 1, если входное значение больше 0, и 0 в противном случае.

Класс Sigmoid реализует сигмоидную функцию активации, которая возвращает значение от 0 до 1. Производная этой функции равна произведению значения сигмоидной функции на (1 минус значение сигмоидной функции).

Класс TangH реализует гиперболический тангенс как функцию активации, которая возвращает значение от -1 до 1. Производная этой функции равна (1 минус квадрат значения гиперболического тангенса).

Вы хотите узнать что-то еще об этом коде?

ПВы хотите узнать что-то еще об этом коде?

9из20

объясни код class Layer:

def \_\_init\_\_(self):

self.input = None

self.output = None

def forward\_propagation(self, input):

raise NotImplementedError

def backward\_propagation(self, output\_error):

raise NotImplementedError

class FCLayer(Layer):

def \_\_init\_\_(self, input\_size, output\_size):

self.weights = np.random.rand(input\_size, output\_size) - 0.5

self.bias = np.random.rand(1, output\_size) - 0.5

def forward\_propagation(self, input\_data):

self.input = input\_data

self.output = np.dot(self.input, self.weights) + self.bias

return self.output

def backward\_propagation(self, output\_error, learning\_rate):

input\_error = np.dot(output\_error, self.weights.T)#dE/dx

weights\_error = np.dot(self.input.T, output\_error)#dE/dw

self.weights -= learning\_rate \* weights\_error

self.bias -= learning\_rate \* output\_error

return input\_error #dE/dB так как равен de/dx

class ActivationLayer(Layer):

def \_\_init\_\_(self, activation, activation\_prime):

self.activation = activation

self.activation\_prime = activation\_prime

def forward\_propagation(self, input\_data):

self.input = input\_data

self.output = self.activation(self.input)

return self.output

def backward\_propagation(self, output\_error, learning\_rate):

return self.activation\_prime(self.input) \* output\_error

Этот код определяет класс Layer и два его подкласса: FCLayer и ActivationLayer. Класс Layer содержит два метода: forward\_propagation и backward\_propagation, которые не реализованы.

Подкласс FCLayer представляет собой полносвязный слой. Он содержит веса и смещения, которые инициализируются случайными значениями. Метод forward\_propagation вычисляет выходные данные слоя как произведение входных данных на веса плюс смещения. Метод backward\_propagation вычисляет ошибку входных данных и обновляет веса и смещения с использованием градиентного спуска.

Подкласс ActivationLayer представляет собой слой активации. Он содержит функцию активации и ее производную. Метод forward\_propagation вычисляет выходные данные слоя как значение функции активации, примененной к входным данным. Метод backward\_propagation вычисляет ошибку входных данных как произведение ошибки выходных данных на производную функции активации.

Вы хотите узнать что-то еще об этом коде?

10из20

объясни код class LossFunction:

def fun():

pass

def derivative():

pass

class MeanSquaredError(LossFunction):

def fun(y\_true, y\_pred):

return np.mean(np.power(y\_true-y\_pred, 2))

def derivative(y\_true, y\_pred):

return 2\*(y\_pred-y\_true)/y\_true.size

Этот код определяет класс LossFunction и его подкласс MeanSquaredError. Класс LossFunction содержит два метода: fun и derivative, которые не реализованы.

Подкласс MeanSquaredError реализует функцию потерь среднеквадратичной ошибки. Метод fun вычисляет среднее значение квадратов разностей между истинными и предсказанными значениями. Метод derivative вычисляет производную этой функции потерь по предсказанным значениям.

Вы хотите узнать что-то еще об этом коде?

Вы хотите узнать что-то еще об этом коде?

11из20

объясни код class NN:

def \_\_init\_\_(self):

self.layers = []

self.loss = None

self.loss\_prime = None

def add(self, layer):

self.layers.append(layer)

def compile(self, loss, loss\_prime):

self.loss = loss

self.loss\_prime = loss\_prime

def predict(self, input\_data):

samples = len(input\_data)

result = []

for i in range(samples):

output = input\_data[i]

for layer in self.layers:

output = layer.forward\_propagation(output)

result.append(\*output)

return result

def fit(self, x\_train, y\_train, epochs, learning\_rate):

samples = len(x\_train)

for i in range(epochs):

for j in range(samples):

output = x\_train[j]

for layer in self.layers:

output = layer.forward\_propagation(output)

error = self.loss\_prime(y\_train[j], output)

for layer in reversed(self.layers):

error = layer.backward\_propagation(error, learning\_rate)

Этот код определяет класс NN (Neural Network), который представляет собой нейронную сеть. Класс содержит список слоев и функцию потерь с ее производной. Метод add добавляет слой в список слоев. Метод compile устанавливает функцию потерь и ее производную.

Метод predict принимает на вход массив входных данных и возвращает массив предсказанных значений. Для каждого образца входных данных вычисляется выход каждого слоя с помощью метода forward\_propagation.

Метод fit обучает нейронную сеть на обучающих данных x\_train и y\_train в течение заданного количества эпох с заданной скоростью обучения. Для каждого образца обучающих данных вычисляется выход каждого слоя с помощью метода forward\_propagation. Затем вычисляется ошибка с помощью производной функции потерь и обновляются параметры каждого слоя с помощью метода backward\_propagation.

объясни код X\_clf = df.drop(["Delay"], axis=1).values

y\_clf = df["Delay"].values

Этот код использует метод drop для удаления столбца Delay из датафрейма df и сохранения оставшихся данных в переменной X\_clf. Затем он извлекает значения столбца Delay и сохраняет их в переменной y\_clf. В результате получаем две переменные: X\_clf, содержащую все данные из df, кроме столбца Delay, и y\_clf, содержащую только значения столбца Delay.

1из20

объясни код std = StandardScaler()

std.fit(X\_clf)

X\_clf = pd.DataFrame(std.transform(X\_clf), columns=df.drop(["Delay"], axis=1).columns).values

Этот код создает объект StandardScaler и сохраняет его в переменной std. Затем он использует метод fit для вычисления среднего и стандартного отклонения для каждого признака в X\_clf. Далее он преобразует данные в X\_clf, используя метод transform объекта std, и сохраняет результат в новом датафрейме с теми же именами столбцов, что и в исходном датафрейме df, за исключением столбца Delay. Наконец, он извлекает значения из этого нового датафрейма и сохраняет их в переменной X\_clf.

2из20

объясни код underSampler = RandomUnderSampler(random\_state=42)

X\_clf, y\_clf = underSampler.fit\_resample(X\_clf,y\_clf)

X\_clf= X\_clf.reshape(X\_clf.shape[0], 1, X\_clf.shape[1])

Этот код создает объект RandomUnderSampler с фиксированным random\_state и сохраняет его в переменной underSampler. Затем он использует метод fit\_resample этого объекта для выполнения случайного недооценивания данных в X\_clf и y\_clf. В результате получаем новые значения для X\_clf и y\_clf, которые содержат меньшее количество наблюдений. Наконец, он изменяет форму массива X\_clf, добавляя дополнительное измерение размером 1 между первым и вторым измерениями.

3из20

объясни код X\_clf\_train, X\_clf\_test, y\_clf\_train, y\_clf\_test = train\_test\_split(X\_clf,

y\_clf,

stratify=y\_clf,

test\_size=0.2)

Этот код использует функцию train\_test\_split для разделения данных в X\_clf и y\_clf на обучающую и тестовую выборки. Он устанавливает размер тестовой выборки равным 20% от общего количества данных и использует стратификацию по y\_clf для сохранения соотношения классов в обучающей и тестовой выборках. В результате получаем четыре новых переменных: X\_clf\_train, X\_clf\_test, y\_clf\_train и y\_clf\_test, содержащие соответственно обучающие и тестовые данные для X\_clf и y\_clf.

4из20

объясни код model\_classification = NN()

model\_classification.add(FCLayer(23, 3))

model\_classification.add(ActivationLayer(TangH.callFunction, TangH.callDerivative))

model\_classification.add(FCLayer(3, 1))

model\_classification.add(ActivationLayer(Sigmoid.callFunction, Sigmoid.callDerivative))

model\_classification.compile(MeanSquaredError.fun, MeanSquaredError.derivative)

model\_classification.fit(X\_clf\_train, y\_clf\_train, epochs=50, learning\_rate=0.001)

pred\_classification = model\_classification.predict(X\_clf\_test)

Этот код создает объект нейронной сети NN и сохраняет его в переменной model\_classification. Затем он добавляет в модель два полносвязных слоя FCLayer с размерами 23 и 3 нейронов соответственно. Между этими слоями он добавляет слой активации ActivationLayer с функцией активации TangH. Затем он добавляет еще один полносвязный слой с 1 нейроном и еще один слой активации с функцией активации Sigmoid. Далее он компилирует модель, используя функцию потерь MeanSquaredError, и обучает ее на данных X\_clf\_train и y\_clf\_train в течение 50 эпох с шагом обучения 0.001. Наконец, он использует обученную модель для предсказания значений на тестовых данных X\_clf\_test и сохраняет результат в переменной pred\_classification.

5из20

объясни код pred\_classification

pred\_classification - это переменная, которая содержит предсказанные значения модели model\_classification на тестовых данных X\_clf\_test. Эти значения были получены с помощью метода predict объекта model\_classification, который использовал обученную модель для вычисления предсказаний на тестовых данных.

6из20

объясни код print(classification\_report(y\_clf\_test, np.around(pred\_classification)))

print(confusion\_matrix(y\_clf\_test, np.around(pred\_classification)))

Этот код использует функции classification\_report и confusion\_matrix для вычисления и вывода на экран отчета о классификации и матрицы ошибок для предсказанных значений pred\_classification и истинных значений y\_clf\_test. Он округляет предсказанные значения с помощью функции np.around перед передачей их в эти функции. Отчет о классификации содержит метрики, такие как точность, полнота и F-мера для каждого класса, а матрица ошибок показывает количество верных и неверных предсказаний для каждого класса.

7из20

объясни код y\_reg = df\_reg["price"].values

X\_reg = df\_reg.drop(["price"], axis=1).values.reshape(df\_reg.shape[0], 1, df\_reg.shape[1] - 1)

Этот код извлекает значения столбца price из датафрейма df\_reg и сохраняет их в переменной y\_reg. Затем он удаляет столбец price из df\_reg, используя метод drop, и сохраняет оставшиеся данные в переменной X\_reg. Наконец, он изменяет форму массива X\_reg, добавляя дополнительное измерение размером 1 между первым и вторым измерениями. В результате получаем две переменные: y\_reg, содержащую значения столбца price, и X\_reg, содержащую все остальные данные из df\_reg.

8из20

объясни код X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X\_reg, y\_reg, test\_size=0.2,random\_state=4)

X\_train\_reg.shape, X\_test\_reg.shape, y\_train\_reg.shape, y\_test\_reg.shape

Этот код использует функцию train\_test\_split для разделения данных в X\_reg и y\_reg на обучающую и тестовую выборки. Он устанавливает размер тестовой выборки равным 20% от общего количества данных и фиксирует random\_state для воспроизводимости результатов. В результате получаем четыре новых переменных: X\_train\_reg, X\_test\_reg, y\_train\_reg и y\_test\_reg, содержащие соответственно обучающие и тестовые данные для X\_reg и y\_reg. Затем он выводит на экран формы этих массивов с помощью атрибута shape.

9из20

объясни код model\_reg = NN()

model\_reg.add(FCLayer(26, 16))

model\_reg.add(ActivationLayer(Relu.callFunction, Relu.callFunction))

model\_reg.add(FCLayer(16, 8))

model\_reg.add(ActivationLayer(Relu.callFunction, Relu.callFunction))

model\_reg.add(FCLayer(8, 4))

model\_reg.add(ActivationLayer(Relu.callFunction, Relu.callFunction))

model\_reg.add(FCLayer(4, 1))

model\_reg.compile(MeanSquaredError.fun, MeanSquaredError.derivative)

model\_reg.fit(X\_train\_reg, y\_train\_reg, epochs=50, learning\_rate=0.001)

Этот код создает объект нейронной сети NN и сохраняет его в переменной model\_reg. Затем он добавляет в модель три полносвязных слоя FCLayer с размерами 26, 16 и 8 нейронов соответственно. Между этими слоями он добавляет слои активации ActivationLayer с функцией активации Relu. Затем он добавляет еще один полносвязный слой с 4 нейронами и еще один слой активации с функцией активации Relu. Далее он добавляет последний полносвязный слой с 1 нейроном. Он компилирует модель, используя функцию потерь MeanSquaredError, и обучает ее на данных X\_train\_reg и y\_train\_reg в течение 50 эпох с шагом обучения 0.001.

10из20

объясни код pred\_reg = model\_reg.predict(X\_test\_reg)

pred\_reg - это переменная, которая содержит предсказанные значения модели model\_reg на тестовых данных X\_test\_reg. Эти значения были получены с помощью метода predict объекта model\_reg, который использовал обученную модель для вычисления предсказаний на тестовых данных.

11из20

объясни код print(mean\_absolute\_error(y\_test\_reg, pred\_reg))

print(mean\_squared\_error(y\_test\_reg, pred\_reg))

print(r2\_score(y\_test\_reg, pred\_reg))

print(max\_error(y\_test\_reg, pred\_reg))

Этот код использует функции mean\_absolute\_error, mean\_squared\_error, r2\_score и max\_error для вычисления и вывода на экран различных метрик качества предсказаний модели model\_reg на тестовых данных. Он передает в эти функции истинные значения y\_test\_reg и предсказанные значения pred\_reg и выводит на экран результаты вычислений. Эти метрики позволяют оценить качество предсказаний модели с разных сторон.