Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовая работа

по курсу «Методы, средства и технологии мультимедиа»

Студент: Обыденкова Ю.Ю.
Группа: М8О-408Б-18
Преподаватель: Б. В. Вишняков.
Оценка:

Москва, 2022

Постановка задачи: Выбрать задачу (классификация или регрессия), датасет и метрику качества. Выбранные данные необходимо визуализировать и проанализировать. После этого выполнить препроцессинг. Затем реализовать алгоритм линейной регрессии, проверить качество обучения, сравнить с моделью из sklearn.

Вариант: Линейная регрессия. Будем предсказывать оценку покемонов на основании остальных признаков.

Входные данные

Датасет представляет из себя информацию о покемонах. Содержит следующие признаки:

- Имя
- Стихия (тип 1)
- Ядовитость (тип 2)
- Здоровье
- Атака
- Урон
- Защита
- Скорость атаки
- Скорость защиты
- Скорость
- Поколение
- Легендарность

Описание

Линейная регрессия — модель зависимости переменной x от одной или нескольких других переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

Линейная регрессия относится к задаче определения «линии наилучшего соответствия» через набор точек данных и стала простым предшественником нелинейных методов, которые используют для обучения нейронных сетей. В этой статье покажем вам примеры линейной регрессии.

Градиентный спуск — метод численной оптимизации, который может быть использован во многих алгоритмах, где требуется найти экстремум функции — нейронные сети, SVM, k-средних, регрессии. Однако проще его воспринять в чистом виде (и проще модифицировать).

Регуляризация — метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Чаще всего эта информация имеет вид штрафа за сложность модели.

Регуляризация L2 – это тип линейной регрессии, который позволяет регуляризовать модель, основан на выборе как можно меньших значений веса. Другими словами, регуляризация ограничивает модель, уменьшая влияние входов на выход, тем самым модель становится регуляризованной и избегает переоснащения этими ограничениями.

Регуляризация L1 – оператор наименьшей абсолютной усадки и выбора (LACCO), при этом методе веса некоторых значений принимаются равными модулю числа, предполагая, что они не влияют на результат.

Ход работы

Загружаем датасет и изучаем его данные:

```
B [1]: # 1. Выбрать задачу (классификация или регрессия), датасет (пересечений не должно быть - у каждого студента свой да
       # и метрику качества

       # Мой выбор - регрессия и метрика R2
       import matplotlib.pyplot as plt
       import seaborn as sns
       import pandas as pd
       import numpy as np
       import sklearn
       import warnings
       warnings.simplefilter("ignore")
       %matplotlib inline

       sns.set(style="darkgrid")
       data = pd.read_csv("Pokemon.csv")
       data.head()
```

Out[1]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

```
B [2]: data.shape
```

```
Out[2]: (800, 13)
```

Производим One Hot Encoding на категориальные признаках

```
B [3]: # 2. Сделать препроцессинг, фичеинжиниринг и т.д. \\ разрешается использование любых пакетов
       data.isna().sum()
```

```
Out[3]: #               0
        Name            0
        Type 1          0
        Type 2        386
        Total           0
        HP              0
        Attack          0
        Defense         0
        Sp. Atk         0
        Sp. Def         0
        Speed           0
        Generation      0
        Legendary       0
        dtype: int64
```

```
B [4]: categorical = list(data.dtypes[data.dtypes == "object"].index)

       a = data[categorical]
       a = pd.get_dummies(a)

       def change(a):
           if np.isnan(a)==True:
                   return 0
           else:
               return a

       data = data.drop(columns=categorical)

       data = data.join(a)
```

Далее мы разделяем данные на обучающую и тестовую выборку, а также нормализуем X-данные с помощью StandardScaler.

B [6]:
```
y = data["Speed"]
X = data.drop(columns=["Speed","Legendary"])

from sklearn.model_selection import train_test_split

numeric_features = data.select_dtypes(include=np.number).columns.tolist()
X_nf = data[numeric_features]
numeric_features = X_nf.columns
X_nf = X_nf.fillna(method='ffill')
X_nf.isna().mean()

X_train, X_test, y_train, y_test = train_test_split(X_nf, y, test_size=0.3, random_state=10)
```

B [7]:
```
for i in list(X_train.columns):
    X_test[i] = X_test[i].agg(change)
    X_train[i] = X_train[i].agg(change)
    y_train = y_train.agg(change)
    y_test = y_test.agg(change)
```

B [8]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

sc.fit(X_train)
X_train_sc = sc.transform(X_train)
X_test_sc  = sc.transform(X_test)
```

Реализуем метрику качества:

B [19]:
```
# 3. Реализовать метрику качества \\ аргументировать выбор метрики качества
# метрика r2 используется в реализации лин регрессии и поэтому очень удобна для сравнения
def r2(real, predicted):
    Sum1 = np.sum((predicted - real) ** 2)
    Sum2 = np.sum((real - np.mean(real)) ** 2)

    r2 = 1 - Sum1 / Sum2

    return r2
```

Реализуем линейную регрессию:

B [20]:
```
# 4. Реализовать и обучить логистическую регрессию или линейную регрессию \\ !!!не разрешается использовать sklearn!!!
def predict_y(x, weight, bias):
    return np.array(weight).dot(np.array(x.T)) + np.array([bias[0]+i*0 for i in range(len(np.array(weight).dot(np.array(x.T))))])

def cost_function(x, y, weight, bias):
    pred = np.array(weight).dot(np.array(x.T)) + np.array([bias[0]+i*0 for i in range(len(np.array(weight).dot(np.array(x.T))))])
    error = (np.array(y)-pred)**2
    error = np.mean(error)

    return error
    '''companies = len(x)
    total_error = 0.0
    for i in range(companies):
        total_error += (y[i] - (weight*x[i] + bias))**2
    return total_error / companies'''
```

B [21]:
```
def update_weights(x, y, weight, bias, learning_rate):
    weight_deriv = 0
    bias_deriv = 0
    companies = len(x)

    for i in range(companies):
        # Вычисление частных производных
        # -2x(y - (mx + b))
        weight_deriv += -2*x[i] * (y[i] - (weight*x[i] + bias))

        # -2(y - (mx + b))
        bias_deriv += -2*(y[i] - (weight*x[i] + bias))

    # Мы вычитаем, потому что производные указывают в направлении самого крутого подъема
    weight -= (weight_deriv / companies) * learning_rate
    bias -= (bias_deriv / companies) * learning_rate

    return weight, bias
```

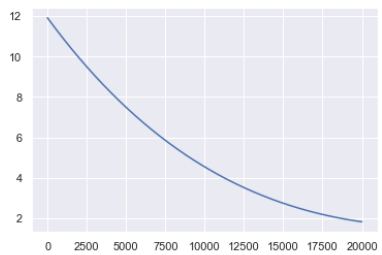
B [22]:
```
def train(x, y, weight, bias, learning_rate, iters):
    cost_history = []

    weight = [weight + i*0 for i in range (len(x[0]))]

    fade = 1

    for i in range(iters):
        '''if (i < 120):
            weight,bias = update_weights(x, y, weight, bias, learning_rate)
        else:
            weight,bias = update_weights(x, y, weight, bias, learning_rate)
            learning_rate = learning_rate/fade
            fade = fade + 0.01'''

        weight,bias = update_weights(x, y, weight, bias, learning_rate)

        #Calculate cost for auditing purposes
        cost = cost_function(x, y, weight, bias)
        cost_history.append(cost)

        #Calculate score for auditing purposes
        score_iter = r2(x, weight)
        score.append(score_iter)

        # Log Progress
        if i % 50 == 0:
            print("iter={}    weight={}    bias={}    cost={}".format(i, weight, bias, cost))


    return weight, bias, cost_history, score
```

Обучим линейную регрессию:

B [23]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
weight = 0.4 # веса
bias = 0.3 # сдвиг
lr = 0.00001 # качество обучения
iters = 20000 # итерации
score = [] # оценка
weight, bias, cost_history, score = train(x,y,weight,bias,lr,iters)
```

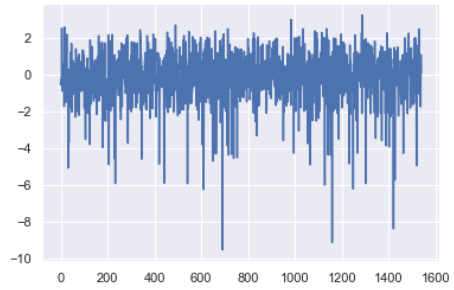
```
 0.3415446  0.29128321 0.35230717 0.36035257 0.30389308 0.34531412
 0.30237983 0.3368577  0.33666274 0.32234642 0.32291886 0.30780045
 0.34681128 0.233676   0.37565059 0.0143729 ]    bias=[0.20149378 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378
 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378
 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378
 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378
 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378 0.20149378
 0.20149378 0.20149378 0.20149378 0.20149378]    cost=1.8414600158648782
iter=19950    weight=[0.08924271 0.20370177 0.07740824 0.1054241  0.04252733 0.09885282
 0.09719316 0.09867684 0.0993108  0.1166665  0.59741524 0.19998554
 0.30886411 0.34492786 0.32860939 0.38391991 0.30725167 0.30996426
 0.34142509 0.29106094 0.35220967 0.36027151 0.30369659 0.34520231
 0.30218025 0.3367286  0.33653324 0.32218766 0.32276127 0.30761195
 0.34670254 0.23333596 0.37560081 0.01358449]    bias=[0.20129238 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238
 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238
 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238
 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238
 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238 0.20129238
 0.20129238 0.20129238 0.20129238 0.20129238]    cost=1.835839074737468
```

```
plt.show()
#Изменения ошибки на каждом градиентном шаге
```



In [45]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y(x, weight, bias))
```

Out[45]:
```
[<matplotlib.lines.Line2D at 0x29d9e2a3610>]
```

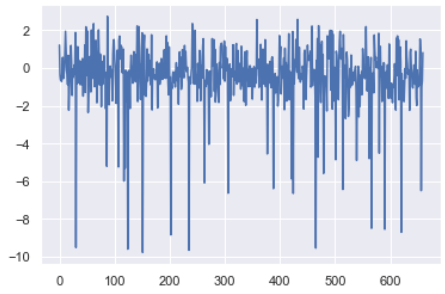


In [46]:
```
np.mean(abs(y - predict_y(x, weight, bias)))
#Ошибка является минимальной
```

Out[46]:
```
0.9629833960838762
```

In [47]:
```
x = X_test_sc
y = y_test.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y(x, weight, bias))
```

Out[47]:
```
[<matplotlib.lines.Line2D at 0x29d9e2f8fd0>]
```



In [48]:
```
np.mean(abs(y - predict_y(x, weight, bias)))
#Ошибка является минимальной
```

Out[48]:
```
1.0516599987645703
```

Из результатов наблюдаем, что у нас есть небольшое переобучение, поэтому мы производим регуляризацию нашего градиента

В [24]:
```
# 4*. Добавить регуляризацию \\ не разрешается использовать sklearn

def l1(w):
    summa = 0
    for i in w:
        summa = summa + abs(i)

    return summa

def l2(w):
    summa = 0
    for i in w:
        summa = summa + (i)**2

    return summa

def predict_y_reg(x, weight, bias):
    return np.array(weight).dot(np.array(x.T)) + np.array([bias[0]+i*0 for i in range(len(np.array(weight).dot(np.array(x.T))))])

def cost_function_reg(x, y, weight, bias):
    pred = np.array(weight).dot(np.array(x.T)) + np.array([bias[0]+i*0 for i in range(len(np.array(weight).dot(np.array(x.T))))])
    error = (np.array(y)-pred)**2
    error = np.mean(error)

    return error
```

```
def update_weights_reg(x, y, weight, bias, learning_rate, lam, t = 1): #lam - сила регуляризации, t - тип регуляризации
    #базовое значение t = 1
    weight_deriv = 0
    bias_deriv = 0
    companies = len(x)

    for i in range(companies):
        # Вычисление частных производных
        # -2x(y - (mx + b))
        weight_deriv += -2*x[i] * (y[i] - (weight*x[i] + bias))

        # -2(y - (mx + b))
        bias_deriv += -2*(y[i] - (weight*x[i] + bias))

    weight = np.array(weight)
    if t == 1:
        weight = weight - lam * l1(weight)
        weight = weight / len(weight)
    elif t == 2:
        weight = weight - lam * l2(weight)
        weight = weight / len(weight)
    else:
        print ("Such l does not exist.")
    # Мы вычитаем, потому что производные указывают в направлении самого крутого подъема
    weight -= (weight_deriv / companies) * learning_rate
    bias -= (bias_deriv / companies) * learning_rate
```

```
def train_reg(x, y, weight, bias, learning_rate, iters, lam, t = 1):
    cost_history = []

    weight = [weight + i*0 for i in range (len(x[0]))]

    fade = 1

    for i in range(iters):
        '''if (i < 120):
            weight,bias = update_weights_reg(x, y, weight, bias, learning_rate, lam, t)
        else:
            weight,bias = update_weights_reg(x, y, weight, bias, learning_rate, lam, t)
            learning_rate = learning_rate/fade
            fade = fade + 0.01'''

        weight,bias = update_weights_reg(x, y, weight, bias, learning_rate, lam, t)

        #Calculate cost for auditing purposes
        cost = cost_function_reg(x, y, weight, bias)
        cost_history.append(cost)

        #Calculate score for auditing purposes
        score_iter = r2(x, weight)
        score.append(score_iter)

        # Log Progress
        if i % 5 == 0:
            print("iter={}    weight={}    bias={}    cost={}".format(i, weight, bias, cost))


    return weight, bias, cost_history, score
```

Обучаем регулизацию L1:

B [25]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
weight = 0.4 # веса
bias = 0.3 # сдвиг
lr = 0.01 # качество обучения
iters = 200 # итерации
score = [] # оценка
lam = 0.1 # лямбда
t = 1
weight, bias, cost_history, score = train_reg(x,y,weight,bias,lr,iters,lam, t)
```

```
  0.00384062  0.00075011  0.00450239  0.00499709  0.00152548  0.0040724
  0.00143243  0.00355242  0.00354044  0.00266015  0.00269535  0.00176574
  0.00416446 -0.00279207  0.00593774 -0.01627671]    bias=[0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851]    cost=0.8051269099860349
iter=195    weight=[-0.01163408 -0.00461052 -0.01236028 -0.01064114 -0.01450068 -0.01104438
 -0.01114622 -0.01105517 -0.01101627 -0.00995127  0.01954897 -0.00483856
  0.00184257  0.00405556  0.0030542   0.00644823  0.00174363  0.00191008
  0.00384062  0.00075011  0.00450239  0.00499709  0.00152548  0.0040724
  0.00143243  0.00355242  0.00354044  0.00266015  0.00269535  0.00176574
  0.00416446 -0.00279207  0.00593774 -0.01627671]    bias=[0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047]    cost=0.8051195837367082
```

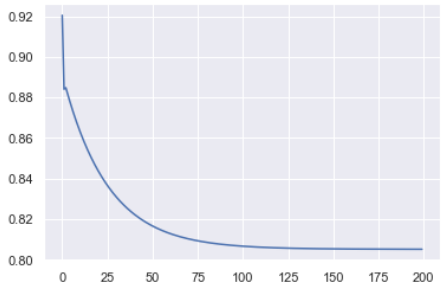
B [26]:
```
# 5. Оценить качество модели на обучающей и тестовой выборках \\ не разрешается использовать sklearn
cost_function(x, y, weight, bias)
```

Out[26]: 0.8051147001306329

B [27]:
```
x = X_test_sc
y = y_test.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
cost_function(x, y, weight, bias)
#Наблюдается переобучение, но незначительное
#При добавлении регуляризации разница будет куда меньше
```

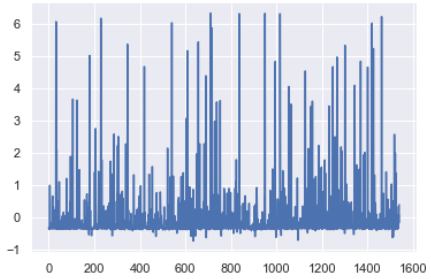
Out[27]: 0.7944286825857141

B [28]:
```
# 5*. Сделать график ошибки модели на обучающей и тестовой выборках
# 5*. Сделать график точности модели на обучающей и тестовой выборках
plt.plot(cost_history)
plt.grid(True)
plt.show()
#Изменения ошибки на каждом градиентном шаге
```



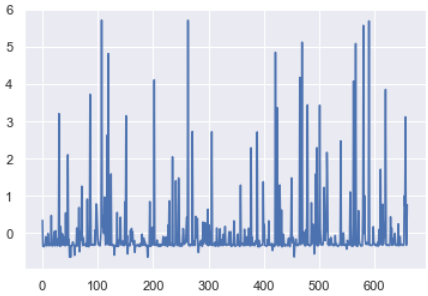
В [53]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y_reg(x, weight, bias))
#точечная разница - (истинное значение - предсказанное)
#y - разница, x - номер элемента
```

Out[53]:
```
[<matplotlib.lines.Line2D at 0x29d9c4cef40>]
```



В [55]:
```
x = X_test_sc
y = y_test.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y_reg(x, weight, bias))
```

Out[55]:
```
[<matplotlib.lines.Line2D at 0x29d9c52ee80>]
```



В [56]:
```
np.mean(abs(y - predict_y_reg(x, weight, bias)))
#Ошибка является минимальной
```

Out[56]:
```
0.4737185974742898
```

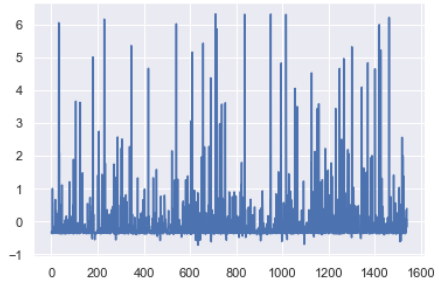
Обучаем регулизацию L2:

В [79]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
weight = 0.4 # веса
bias = 0.3 # сдвиг
lr = 0.01 # качество обучения
iters = 200 # итерации
score = [] # оценка
lam = 0.1 # лямбда
t = 2
weight, bias, cost_history, score = train_reg(x,y,weight,bias,lr,iters,lam, t)
```

```
  0.00447541  0.00138491  0.00513719  0.00563189  0.00216028  0.0047072
  0.00206723  0.00418722  0.00417524  0.00329495  0.00333014  0.00240053
  0.00479925 -0.00215727  0.00657254 -0.01564191]    bias=[0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851
 0.00632851 0.00632851 0.00632851 0.00632851 0.00632851]    cost=0.8060406813252322
iter=195    weight=[-0.01099928 -0.00397572 -0.01172548 -0.01000634 -0.01386588 -0.01040958
 -0.01051142 -0.01042037 -0.01038147 -0.00931647  0.02018377 -0.00420376
  0.00247737  0.00469036  0.003689    0.00708303  0.00237843  0.00254488
  0.00447541  0.00138491  0.00513719  0.00563189  0.00216028  0.0047072
  0.00206723  0.00418722  0.00417524  0.00329495  0.00333014  0.00240053
  0.00479925 -0.00215727  0.00657254 -0.01564191]    bias=[0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047 0.00572047
 0.00572047 0.00572047 0.00572047 0.00572047]    cost=0.8060333550759058
```

В [80]:
```
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y_reg(x, weight, bias))
```

Out[80]:
```
[<matplotlib.lines.Line2D at 0x29da173f310>]
```

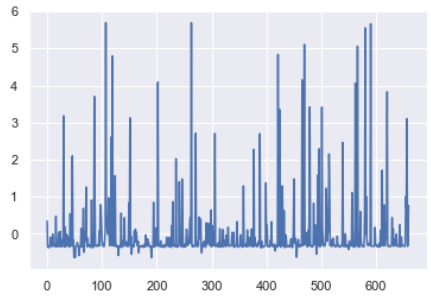


В [81]:
```
np.mean(abs(y - predict_y_reg(x, weight, bias)))
#Ошибка является минимальной
```

Out[81]:
```
0.469232759995526
```

В [82]:
```
x = X_test_sc
y = y_test.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
plt.plot([i for i in range(len(y))],y - predict_y_reg(x, weight, bias))
```

Out[82]: [<matplotlib.lines.Line2D at 0x29da276b280>]



В [83]:
```
np.mean(abs(y - predict_y_reg(x, weight, bias)))
#Ошибка является минимальной
```

Out[83]: 0.4755444683207061

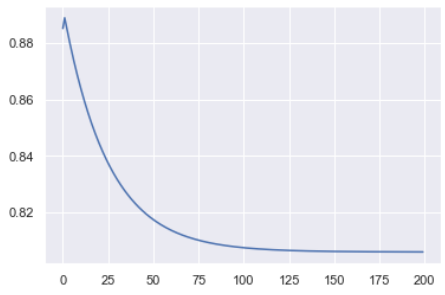
В [84]:
```
# 5. Оценить качество модели на обучающей и тестовой выборках \\ не разрешается использовать sklearn
x = X_train_sc
y = y_train.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
cost_function(x, y, weight, bias)
```

Out[84]: 0.8060284714698305

В [85]:
```
x = X_test_sc
y = y_test.values
for i in range(len(x)):
    for k in range(len(x[i])):
        if np.isnan(x[i][k])==True:
            x[i][k]=0
for i in range(len(y)):
    if np.isnan(y[i])==True:
            y[i]=0
y = (np.array(y)-np.mean(y))/np.std(y)
y = list(y)
cost_function(x, y, weight, bias)
#Наблюдается переобучение, но незначительное
#При добавлении регуляризации разница будет куда меньше
```

Out[85]: 0.7937310307734364

В [67]:
```
# 5*. Сделать график ошибки модели на обучающей и тестовой выборках
# 5*. Сделать график точности модели на обучающей и тестовой выборках
plt.plot(cost_history)
plt.grid(True)
plt.show()
#Изменения ошибки на каждом градиентном шаге
#График ошибки от итерации градиентного спуска
```



Нам удалось устранить переобучение.
Теперь сделаем то же самое, но с помощью sklearn:

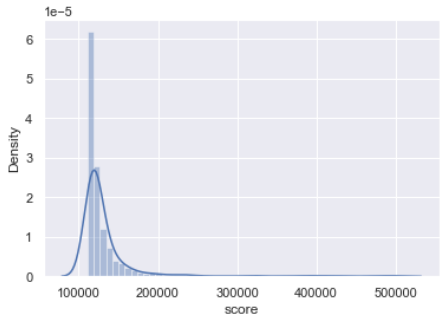
B [78]:
```
# 6. Обучить логистическую регрессию или линейную регрессию из sklearn, оценить качество модели на
# обучающей и тестовой выборках и сравнить со своей моделью
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

lr = linear_model.LinearRegression()
lr.fit(X_train_sc, y_train)
print ("MAE (Mean Absolute Error) Train:", mean_squared_error(lr.predict(X_train_sc), y_train))
print ("MAE (Mean Absolute Error) Test:", mean_squared_error(lr.predict(X_test_sc), y_test))
print ("LR Train RMSE:", np.sqrt(mean_squared_error(lr.predict(X_train_sc), y_train)))
print ("LR Test RMSE:", np.sqrt(mean_squared_error(lr.predict(X_test_sc), y_test)))
```

```
MAE (Mean Absolute Error) Train: 5.639835029031725e-27
MAE (Mean Absolute Error) Test: 6.317312872414258e-27
LR Train RMSE: 7.509883507106968e-14
LR Test RMSE: 7.94815253528407e-14
```

B [30]:
```
# 6*. Сделать график ошибки модели из sklearn на обучающей и тестовой выборках и сравнить со своей моделью
# 6*. Сделать график точности модели из sklearn на обучающей и тестовой выборках и сравнить со своей моделью
```

B [31]:
```
error = (y_train - lr.predict(X_train)) ** 2
sns.distplot(error)
```

Out[31]: <AxesSubplot:xlabel='score', ylabel='Density'>



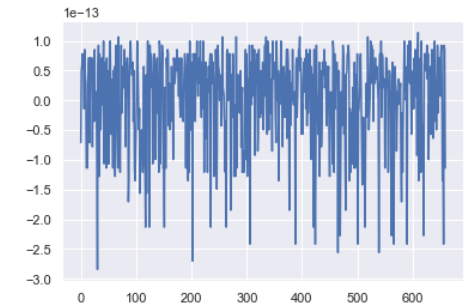
B [32]:
```
# в sklearn нет истории данных
# потому мы делаем поточечную разницу
```

B [33]:
```
y_test - lr.predict(X_test_sc)
```

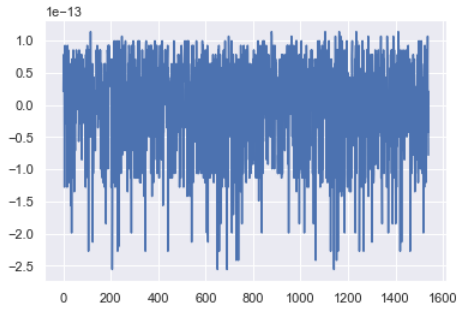
Out[33]:
```
299    -7.105427e-14
1859    4.973799e-14
1939    5.684342e-14
2194    7.815970e-14
1861    6.394885e-14
           ...
256    -1.065814e-13
112    -2.415845e-13
2062    9.237056e-14
1126    8.526513e-14
1247   -1.136868e-13
Name: score, Length: 660, dtype: float64
```

B [34]:
```
plt.plot([i for i in range(len(y_test))],y_test - lr.predict(X_test_sc))
```

Out[34]: [<matplotlib.lines.Line2D at 0x2df85d4ca30>]



```
B [35]: plt.plot([i for i in range(len(y_train))],y_train - lr.predict(X_train_sc))
Out[35]: [<matplotlib.lines.Line2D at 0x2df85d5fac0>]
```



Вывод

В ходе работы я узнала о линейной регрессии, которая является простым, но мощным механизмом для аппроксимации линейных зависимостей.

Модель является предшественником нелинейных методов, которые используют для обучения нейронных сетей. Также удалось улучшить реализацию модели, добавить регуляризацию, и добиться лучших оценок.