Algorithms&Complexity - In Class Assignments (Week 8)

Gradient Descent and Convex Optimization

Manuela Cleves

In [1]:
```
#Import neccesary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from ucimlrepo import fetch_ucirepo
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

In [12]:
```
#Read data
df = pd.read_csv('agaricus-lepiota.data', header=None)

#Encode features
labelencoder = LabelEncoder()
for col in df.columns:
    df[col] = labelencoder.fit_transform(df[col])

#Separate target
X = df.iloc[:, 1:]
y = df.iloc[:, 0]

#Separate test from training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randon

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [13]:
```
#Explore our variables
X
```

Out[13]:

	1	2	3	4	5	6	7	8	9	10	...	13	14	15	16	17	18	19	20	21	22
0	5	2	4	1	6	1	0	1	4	0	...	2	7	7	0	2	1	4	2	3	5
1	5	2	9	1	0	1	0	0	4	0	...	2	7	7	0	2	1	4	3	2	1
2	0	2	8	1	3	1	0	0	5	0	...	2	7	7	0	2	1	4	3	2	3
3	5	3	8	1	6	1	0	1	5	0	...	2	7	7	0	2	1	4	2	3	5
4	5	2	3	0	5	1	1	0	4	1	...	2	7	7	0	2	1	0	3	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	3	2	4	0	5	0	0	0	11	0	...	2	5	5	0	1	1	4	0	1	2
8120	5	2	4	0	5	0	0	0	11	0	...	2	5	5	0	0	1	4	0	4	2
8121	2	2	4	0	5	0	0	0	5	0	...	2	5	5	0	1	1	4	0	1	2
8122	3	3	4	0	8	1	0	1	0	1	...	1	7	7	0	2	1	0	7	4	2
8123	5	2	4	0	5	0	0	0	11	0	...	2	5	5	0	1	1	4	4	1	2

8124 rows × 22 columns

In [4]:
```
#Explore our target (we will assume p=poisonous)
y
```

Out[4]:

	poisonous
0	p
1	e
2	e
3	p
4	e
...	...
8119	e
8120	e
8121	e
8122	p
8123	e

8124 rows × 1 columns

In [ ]:

In [ ]:

In [14]:
```
#Separating the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor
```

In [15]:
```
#Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [16]:
```
#Steps followed in clasee in example
#Define Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
#Define loss function
def loss_fun(y, y_hat):
    m = len(y)
    return -1/m * np.sum(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
#Define descent
def gradient_descent(X, y, w, b, learning_rate):
    m = len(y)
    y_hat = sigmoid(np.dot(X, w) + b)
    dw = 1/m * np.dot(X.T, (y_hat - y))
    db = 1/m * np.sum(y_hat - y)
    w -= learning_rate * dw
    b -= learning_rate * db
    return w, b
```

In [19]:
```
#Example Usage

learning_rate = 0.01
num_iterations = 4000
num_features = X_train.shape[1]

w = np.zeros((num_features, 1))
b = 0

y_train = y_train.reshape(-1, 1)

for i in range(num_iterations):
    w, b = gradient_descent(X_train, y_train, w, b, learning_rate)
    if i % 100 == 0:
        print("Loss after iteration {}: {}".format(i, loss_fun(y_train, sigmoi
```

```
Loss after iteration 0: 0.6886093198465015
Loss after iteration 100: 0.44252145011725397
Loss after iteration 200: 0.35843541906059734
Loss after iteration 300: 0.3177299015161403
Loss after iteration 400: 0.2937939432538207
Loss after iteration 500: 0.27796802973403534
Loss after iteration 600: 0.266668282778348
Loss after iteration 700: 0.2581539297450821
Loss after iteration 800: 0.2514779698152658
Loss after iteration 900: 0.24608032750188064
Loss after iteration 1000: 0.2416080824806741
Loss after iteration 1100: 0.23782760842361633
Loss after iteration 1200: 0.23457805706499124
Loss after iteration 1300: 0.23174508775859423
Loss after iteration 1400: 0.22924522654696053
Loss after iteration 1500: 0.22701613917964786
Loss after iteration 1600: 0.22501035826935697
Loss after iteration 1700: 0.22319111221523813
Loss after iteration 1800: 0.22152947820906135
Loss after iteration 1900: 0.22000239454184398
Loss after iteration 2000: 0.2185912450601683
Loss after iteration 2100: 0.21728083319423336
Loss after iteration 2200: 0.21605862652553992
Loss after iteration 2300: 0.21491419256360983
Loss after iteration 2400: 0.21383877181826666
Loss after iteration 2500: 0.21282495088156556
Loss after iteration 2600: 0.21186640932260914
Loss after iteration 2700: 0.21095772172318458
Loss after iteration 2800: 0.21009420136889462
Loss after iteration 2900: 0.20927177573722733
Loss after iteration 3000: 0.2084868864935977
Loss after iteration 3100: 0.207736408549267
Loss after iteration 3200: 0.2070175840717723
Loss after iteration 3300: 0.20632796831839562
Loss after iteration 3400: 0.20566538488867983
Loss after iteration 3500: 0.20502788853413545
Loss after iteration 3600: 0.20441373407199517
Loss after iteration 3700: 0.20382135026055787
Loss after iteration 3800: 0.20324931773170374
Loss after iteration 3900: 0.20269635025989702
```

In [21]:

```
#Make predictions
y_pred = sigmoid(np.dot(X_test, w) + b) >= 0.5
print (y_pred)
```

```
[[False]
 [ True]
 [ True]
 ...
 [ True]
 [ True]
 [ True]]
```

In [ ]:

```
```

Other code used in class (not for grading purposes, just useful notes for me to complete this assignment):

Gradient Descent - Linear Regression

Let's try to implement a simple linear regression task with gradient descent! Hint: We don't need chain rule.

In [ ]:
```
# Load data from CSV file
data = pd.read_csv('Housing.csv')
```

In [ ]:
```
theta
```

In [ ]:
```
#FINISH THIS, YOU WERE USING CHATGPT "ALPHABETICAL ADVENTURE"

# Assume your CSV file has columns 'X' and 'y'
#The -1 in reshape(-1, 1) is a placeholder that means "whatever is needed."
#The goal is to reshape the array into two dimensions with one column.
#Reshaping is necessary because many machine learning libraries expect the inpu

theta = np.random.randn(2, 1)
x = data['area'].values.reshape(-1, 1)
y = data['price'].values.reshape(-1, 1)




# Add a bias term to X
X_b = np.c_[np.ones((len(X), 1)), X]

# Initialize model parameters

# Set hyperparameters
learning_rate = 0.01
n_iterations = 1000

#Calculkate derivative of cost function with respect to 1st theta and 2nd theta
#Then calculate final uysing FIND DIFFERENCES, NOT CHAIN!!
```

In [ ]:
```
#What Ilia did

def cost(b0,b1,y,x):
    y_pred=finc(b0,b1,x)
    return (1/(2*len(y))*np.sum((y_pred-y)**2))

b0=5
b1=2

h=1e-6
alpha=0.01

fig, ax=plt.subplots()

print(cost(b0,b1,price,area))

for i in range(10):
    y_pred=func(b0,b1,area)
    dj0=(cost(b0+h,b1,price,area)-cost(b0,b1,price,area))/h
    dj1=(cost(b0,b1+1,price,area)-cost(b0,b1,price,area))/h
```

In [ ]:

```
```