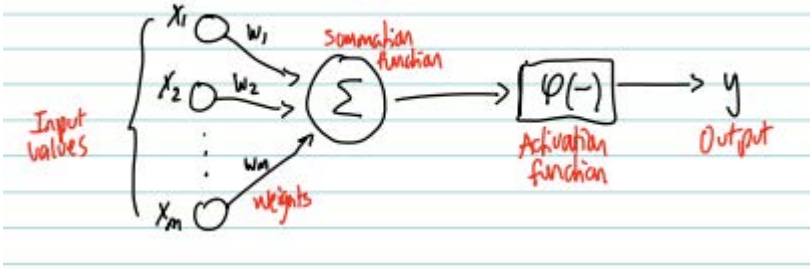
Assignment 1

CAP6619 - Deep Learning Summer 2024

Benjamin Luo

May 27, 2024

Problem 1. Perceptron Model



Inputs: Values that the perceptron receives from either other perceptrons or an existing model (i.e. a dataset, an image, etc.)
Weights: Parameters of the perceptron model that can be adjusted to determine how much influence each incoming input value will have on the final output
Summing function: Function that serves to calculate a weighted sum for the model by adding the products of the corresponding input values and weights together
Activation Function: Function that is employed to determine how the weighted sum calculated by the summing function will be interpreted in the the final output value
Output: The outgoing value of the perceptron, after the weights and input values have been summated and put through the activation function

Problem 2. Activation Functions

(a) Hard-limiter

$$\phi(v) = \begin{cases} 1 \text{ if } v \geq 0 \\ 0 \text{ if } v \leq 0 \end{cases}$$

In the hard-limiter activation function, the output is not continuous; it can only be 1 of 2 possible outputs, which is denoted by the horizontal sections of the graph.

(b) Sigmoid

$$\sigma = \frac{1}{1 + e^{-x}}.$$

The sigmoid function is a function bounded between values 0 and 1. It is continuous, as the output can be any input within the bounds of the function.

(c) Hyperbolic tangent

$$\phi = \tanh(v)$$

The hyperbolic tangent function is similar to the sigmoid function with some key differences; it is zero-centered, meaning that positive values will be positive, and negative values will similarly be negative. The base hyperbolic tangent function lies within the range of -1 and 1.

(d) Rectified Linear Unit (ReLU)

$$R(x) = max(0, x)$$

The rectified linear unit (ReLU) function is simply a linear function that normalizes the negative values into 0. Essentially, it is a linear function that creates different output based on the positive values of the activation inputs. For any results that are less than or equal to 0, the activation function output is 0.

Problem 3. Activation Functions

(a) Figure 1(b) - Sigmoid

For a given input x, the output can be calculated through this equation:

$$\sigma = \frac{1}{1 + e^{-x}}.$$

Using the quotient rule, we get:

$$\frac{d}{dx}\sigma(x) = \frac{(1 + e^{-x})(0) + (1)(-e^{-x})}{(1 + e^{-x})^2} = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$\frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1 + e^{-x} - 1}{(1 + e^{-x})^2} = \frac{1 + e^{-x}}{(1 + e^{-x})^2} - \frac{1}{(1 + e^{-x})^2}$$

$$\frac{1}{(1 + e^{-x})^2} - \frac{1}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}}\left(1 - \frac{1}{1 + e^{-x}}\right)$$

Substituting σ in for the function, we get:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - (\sigma(x))$$

- It is continuous and differentiable as well as bounded, meaning that it will always have a meaningful output (between 0 and 1).
- However, the gradient of the function vanishes as output approaches the extremities of the function (0,1); in addition to that, sigmoid can be computationally expensive as well.

(b) Figure 1(d) - Rectified Linear Unit (ReLU)

The derivative of the gradient of the ReLU activation function is split into two parts. We can first treat the function $ReLU = max(0, x)$ like a piece-wise function, with the sections where the max is determined based on 0 and x, respectively. With this in mind, the derivative of 0 is similarly 0, while the derivative of x is 1. We can represent this as:

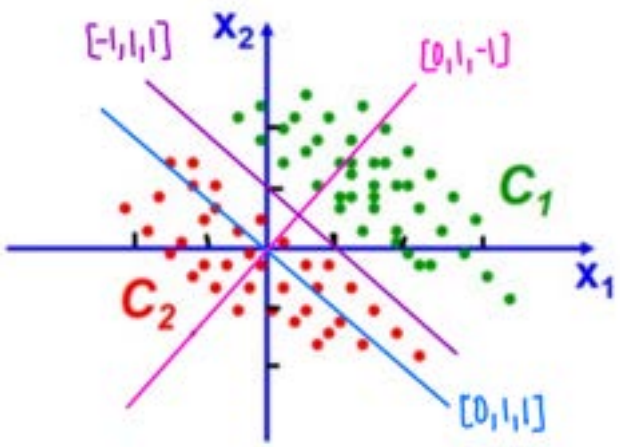
$$R(x) = max(0, x)$$

$$R'(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

The advantages of ReLU is that its gradient is consistent, due to the derivatives being constant in nature, as well as being computationally efficient. However, it is disadvantaged by the fact that it is not differentiable at every point ($x = 0$). ReLU is also not bounded, as the gradient can blow up, since for positive values, it will keep going up.

Problem 4. Linear Line Decision

- The weight values of the neuron are designed to determine the optimal level of emphasis to place on various inputs to the neuron in order to have the most accuracy on the decision surface when it comes to determining the correct class for data inputs.



- w_0, or b, controls the origin that our decision surface is centered about.
- w_1 and w_2 control how large or small that specific dimension of the decision surface must be. In our case, with only two dimensions, it is easy to visualize with a line.
- Among the three decision surfaces mentioned in the question, the decision surface defined by $[-1, 1, 1]$ (depicted as the purple line in the above image) is the best decision to separate instances in this scenario. This is because, we notice that the data points tend to be differentiated into the different classes, either have both x_1 and x_2 values greater than 1 or not. Comparing the decision surface created, shown by the line $x_1 + x_2 - 1 \geq 0$ it seems to most accurately divide data points into their respective classes.

Problem 5. Gradient Descent Part 1

(a) Squared Error

The squared error of our instance x_n with respect to the network is as follows:

$$e(w) = d - (w_0 + w_1x_1 + w_2x_2)$$

Taking our error here and turning it into a second order function, we get the squared error:

$$E(w) = \frac{1}{2} \cdot e^2$$

$$E(w) = \frac{1}{2}(d - (w_0 + w_1x_1 + w_2x_2))^2$$

(b) Weight updating rules for w_0, w_1, w_2

$$f(w_0, w_1, w_2) = \frac{1}{2}(d - [w_0 + w_1x_1 + w_2x_2])^2$$

$$\text{for } w_0, \text{ gradient} = \frac{\partial f(w_0, w_1, w_2)}{\partial w_0}$$

$$= \frac{1}{2} \cdot 2 \cdot (d - [w_0 + w_1x_1 + w_2x_2]) \cdot \frac{\partial (d - [w_0 + w_1x_1 + w_2x_2])}{\partial w_1}$$

$$= (d - [w_0 + w_1x_1 + w_2x_2]) \cdot (-1)$$

$$w_0(k+1) = w_0(k) + (d - [w_0 + w_1x_1 + w_2x_2])$$

Based on Gradient = $(d - [w_0 + w_1x_1 + w_2x_2])$, the weight updating rules for w_1 and w_2 are:

$$w_1(k+1) = w_1(k) + (d - [w_0 + w_1x_1 + w_2x_2]) \cdot x_1$$

$$w_2(k+1) = w_2(k) + (d - [w_0 + w_1x_1 + w_2x_2]) \cdot x_2$$

Problem 6. Gradient Descent Part 2

To find the gradient at a point on the graph, we need to find the derivative first:

$$y = 2x^2 - 4x + 1$$

$$y' = 4x - 4$$

$$y'(-1) = 4(-1) - 4 = -8$$

The gradient at point $x = -1$ is equal to -8.

Following the gradient descent principle, we get:

$$w(k+1) = w(k) - \eta(\text{gradient})$$

$$w(k+1) = -1 - 0.1(-8) = -0.2$$

The next step towards the global minimum using the gradient descent principle is $x = -0.2$.

Problem 7. Gradient Descent Learning Rule Part 1

Mean squared errors $E(W)$ corresponding to the initial weights: $11/2$ $(= 5.5)$

First Round

Input	Weight	v	Desired, $d(n)$	Output, $o(n)$	Δw_i
$(1, 1, 0)$	$(1, 1, 1)$	$[1, 1, 0]$	1	2	$[-0.1, -0.1, 0]$
$(1, -1, 0)$	$(1, 1, 1)$	$[1, -1, 0]$	0	0	$[0, 0, 0]$
$(1, 0, -1)$	$(1, 1, 1)$	$[1, 0, -1]$	1	0	$[0.1, 0, -0.1]$
$(1, 0, 1)$	$(1, 1, 1)$	$[1, 0, 1]$	0	2	$[-0.2, 0, -0.2]$
$(1, 1, 1)$	$(1, 1, 1)$	$[1, 1, 1]$	1	3	$[-0.2, -0.2, -0.2]$
$(1, -1, -1)$	$(1, 1, 1)$	$[1, -1, -1]$	0	-1	$[0.1, -0.1, -0.1]$

New weight after first round: $[0.7, 0.6, 0.4]$
Mean squared errors $E(W)$ after the first-round weight updating: $2.38/2$ $(= 1.19)$

Second Round

Input	Weight	v	$d(n)$	$o(n)$	Δw_i
$(1, 1, 0)$	$(0.7, 0.6, 0.4)$	$[0.7, 0.6, 0]$	1	1.3	$[-0.03, -0.03, 0]$
$(1, -1, 0)$	$(0.7, 0.6, 0.4)$	$[0.7, -0.6, 0]$	0	0.1	$[0.09, -0.09, 0]$
$(1, 0, -1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0, -0.4]$	1	0.3	$[0.07, 0, -0.07]$
$(1, 0, 1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0, 0.4]$	0	1.1	$[-0.01, 0, -0.01]$
$(1, 1, 1)$	$(0.7, 0.6, 0.4)$	$[0.7, 0.6, 0.4]$	1	1.7	$[-0.07, -0.07, -0.07]$
$(1, -1, -1)$	$(0.7, 0.6, 0.4)$	$[0.7, -0.6, -0.4]$	0	-0.3	$[0.13, -0.13, -0.13]$

New weight after first round: $[0.88, 0.28, 0.12]$
Mean squared errors $E(W)$ after the second-round weight updating: $1.752/2$ $(= 0.876)$

Problem 8. Gradient Descent Learning Rule Part 2

Mean squared errors $E(W)$ corresponding to the initial weights: $11/2$ $(= 5.5)$

Input	Weight	v	d	o	Δw	New Weight
$(1, 1, 0)$	$(1, 1, 1)$	$[1, 1, 0]$	1	2	$[-0.1, -0.1, 0]$	$[0.9, 0.9, 1]$
$(1, -1, 0)$	$(0.9, 0.9, 1)$	$[0.9, -0.9, 0]$	0	0	$[0, 0, 0]$	$[0.9, 0.9, 1]$
$(1, 0, -1)$	$(0.9, 0.9, 1)$	$[0.9, 0, -1]$	1	-0.1	$[0.09, 0, -0.09]$	$[0.99, 0.9, 0.91]$
$(1, 0, 1)$	$(0.99, 0.9, 0.91)$	$[0.99, 0, 0.91]$	0	1.9	$[-0.19, 0, -0.19]$	$[0.8, 0.9, 0.72]$
$(1, 1, 1)$	$(0.8, 0.9, 0.72)$	$[0.8, 0.9, 0.72]$	1	2.42	[-.242,-.242,-.242]	[.558, .658, .478]
(1,-1,-1)	(.558, .658, .478)	[.558,-.658,-.478]	0	-.578	[.0578,-.0578,-.0578]	[.6158,.6002,.4302]

Mean squared errors $E(W)$ after the weight updating of the last instance: $\frac{1}{2}[1.98033] \approx 0.99016$

Problem 9. Programming Exercise 1

See the also additional attached file a1q9.pdf

Problem 10. Programming Exercise 2

See the also additional attached file a1q10.pdf for the scripts written.

Gradient descent learning can be very useful in searching for solutions to $f_1(x, y)$ and $f_2(x, y)$ as it helps us identify local and global maxima/minima. Learning rate is incredibly important in gradient descent learning, as it determines how well our model "learns". If the learning rate is too small, learning will be minimal or slow, and if the learning rate is too large, there is a high likelihood of oscillation and lack of real optimization due to being too imprecise in gradient descent.

Assignment 1 Question 9

Benjamin Luo

CAP6619 Summer Term 2024

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sb
```

In [ ]:
```
# Read housing.header.txt as a dataframe.
df = pd.read_csv('housing.header.txt')
df

# Report number of instances and features
# There are 506 instances and 14 features in the dataset
```

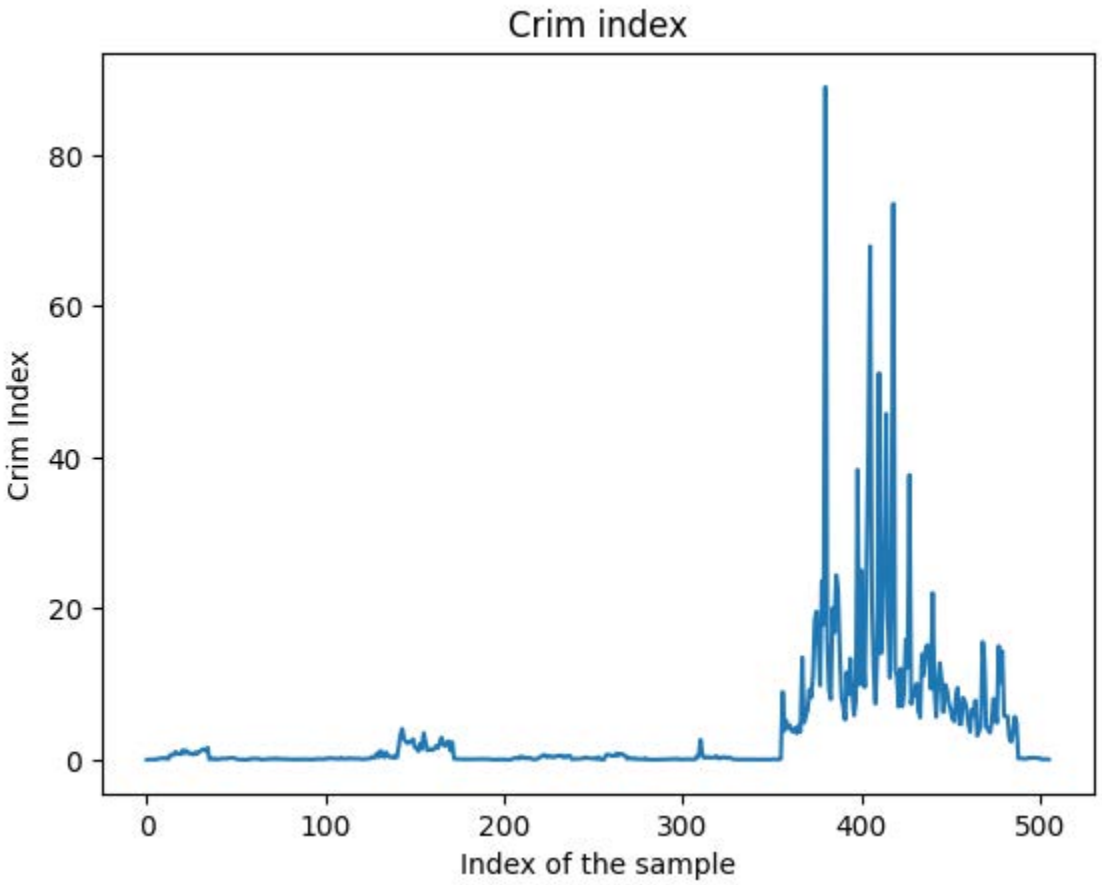
Out[ ]:

	Crim	Zn	Indus	Chas	Nox	Rm	Age	Dis	Rad	Tax	Ptratio	B	Lsta
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.9
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.0
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.9
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.6
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.0
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.6
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.4
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.8

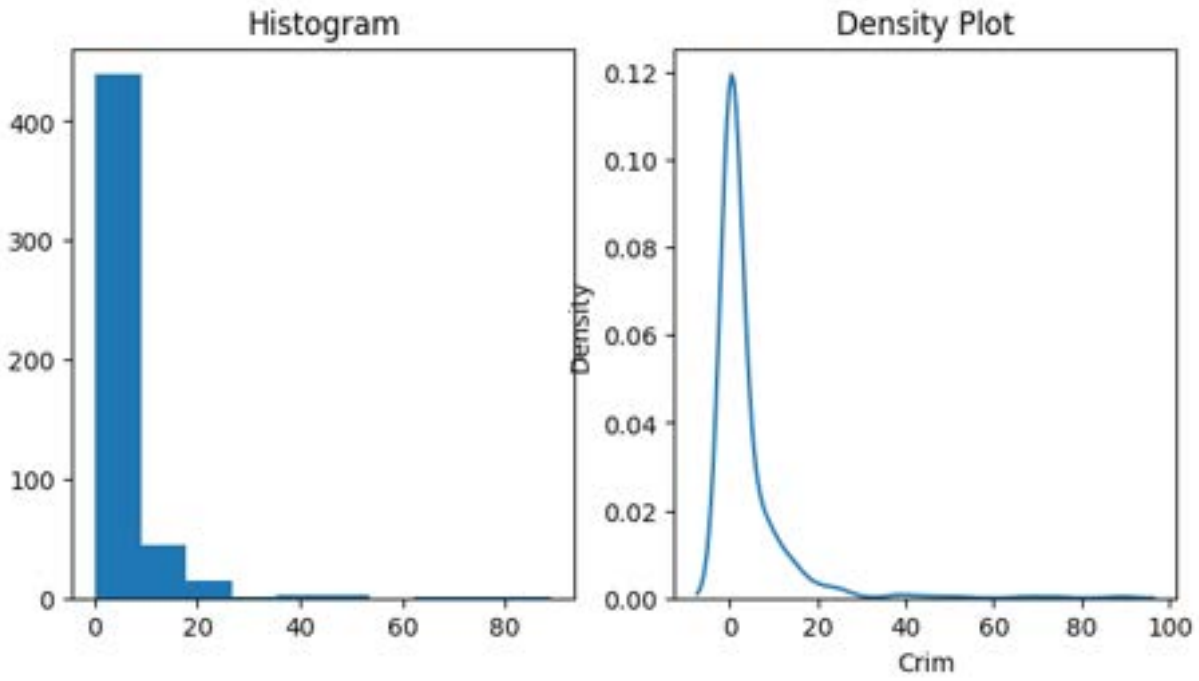
506 rows × 14 columns

In [ ]:
```
#Report all samples with respect to the Crim index on a plot
# (the x-axis shows the index of the sample, and the
# y-axis shows the Crim index of the sample).
y = df['Crim']
x = df.index
plt.plot(x,y)
plt.title('Crim index')
plt.xlabel('Index of the sample')
```

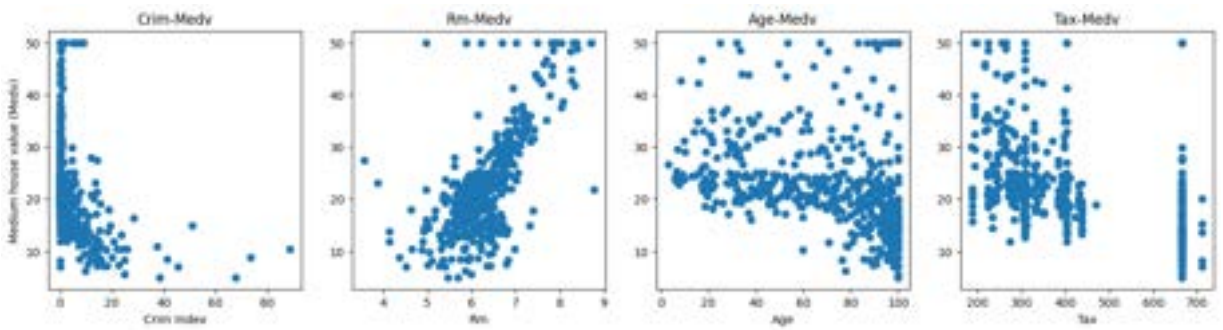
```
plt.ylabel('Crim Index')
plt.show()
```



In [ ]:
```
# Show both histogram of the Crim Index and the density of the
# Crim index on a 1x2 frame (one row two columns).
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(8, 4))


ax1.hist(y)
ax1.set_title('Histogram')
ax2 = plt.subplot(1,2,2)
ax2.set_title('Density Plot')
sb.kdeplot(y)
plt.show()
```



In [ ]:
```
# Show following four scatter plots in one frame (1x4),
# - crim-medv, Rm-medv, Age-medv, Tax-medv

fig, (ax1, ax2, ax3, ax4) = plt.subplots(1,4, figsize=(18, 4))

ax1.scatter(df['Crim'],df['Medv'])
ax1.set_xlabel('Crim Index')
ax1.set_ylabel('Medium house value (Medv)')
ax1.set_title('Crim-Medv')

ax2.scatter(df['Rm'], df['Medv'])
ax2.set_xlabel('Rm')
ax2.set_title('Rm-Medv')

ax3.scatter(df['Age'], df['Medv'])
ax3.set_xlabel('Age')
ax3.set_title('Age-Medv')

ax4.scatter(df['Tax'], df['Medv'])
ax4.set_xlabel('Tax')
ax4.set_title('Tax-Medv')
plt.show()
```



Explain how are they (Crim, Rm, Age, Tax) correlated to the medium house value

(Medv).

Crim index does not seem to be indicative of value at less than 1, but as the value of the Crim index grows, Medv trends downwards.

As *Rm* increases, Medv also increases in value.

Age does not seem to be related in any particular way to Medv.

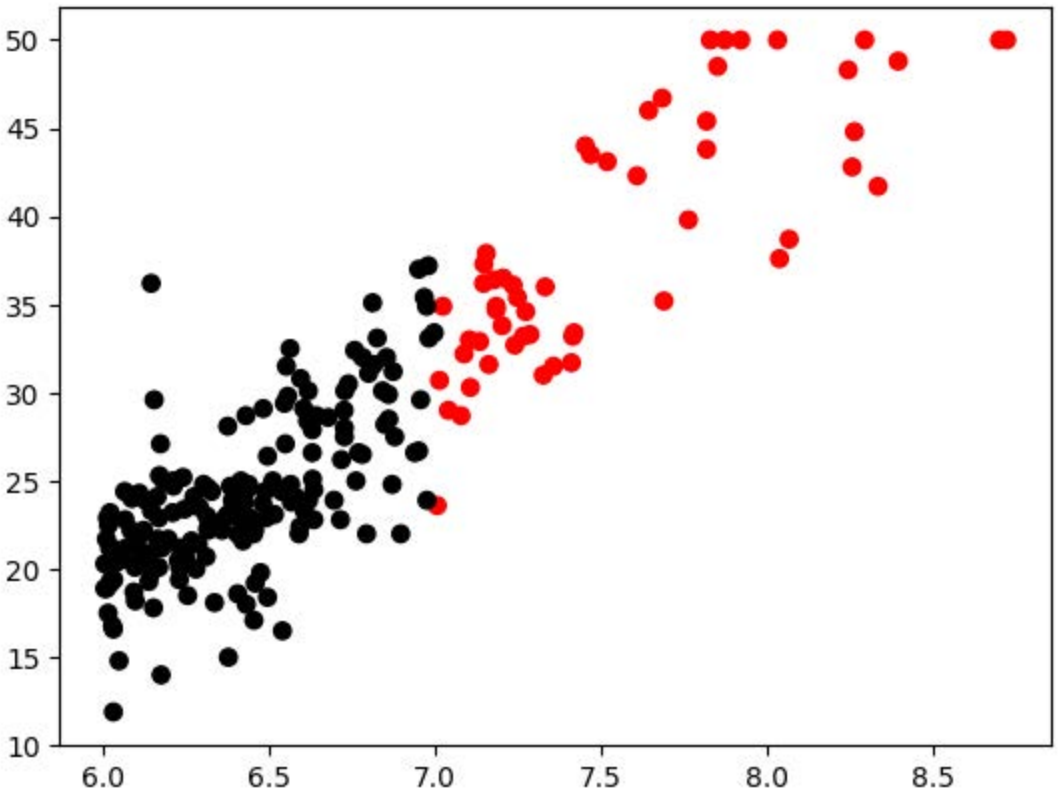
Tax also does not seem to be related, but over a certain threshold, there does seem to be a decrease in value of Medv.

In [ ]:
```
# Create a subset which only includes properties with Crim less than 1 (inclusive)
# and Rm greater than 6 (inclusive).

subset = df[(df['Crim']<=1) & (df['Rm'] >= 6)]
subset.head()
```

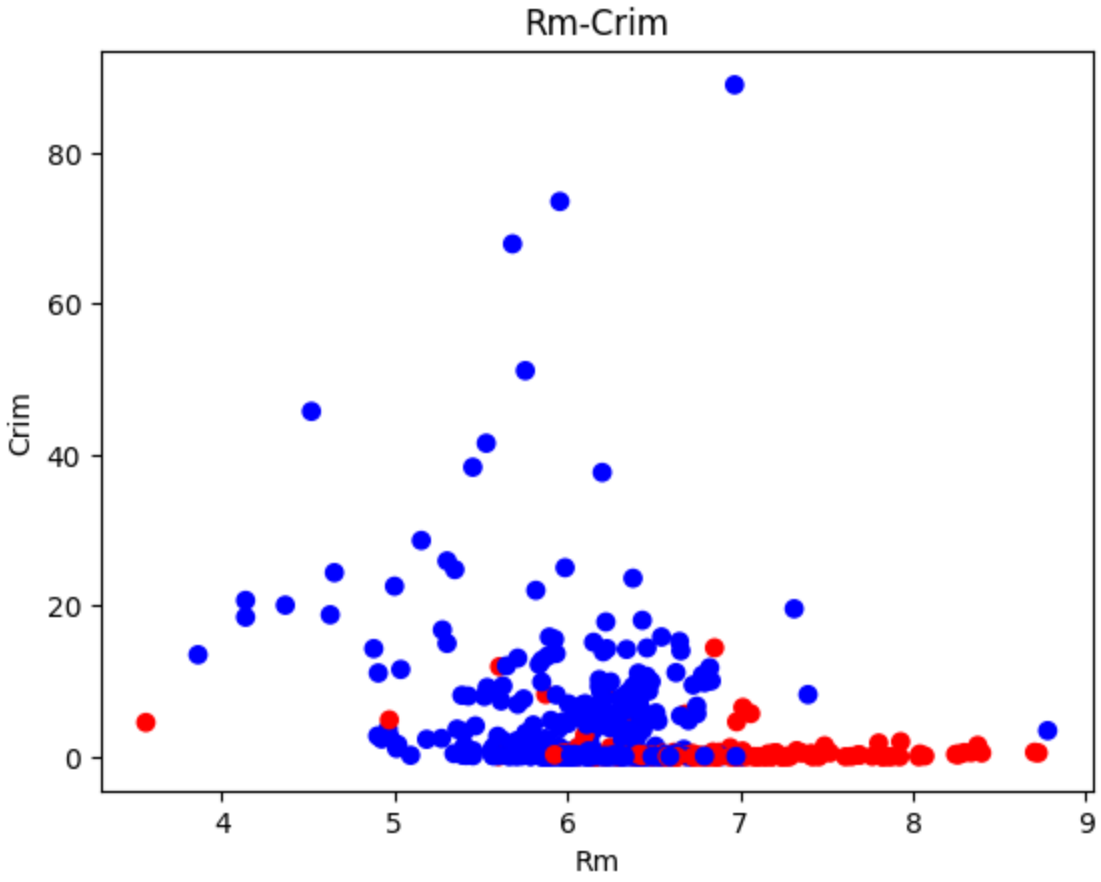
Out[ ]:

	Crim	Zn	Indus	Chas	Nox	Rm	Age	Dis	Rad	Tax	Ptratio	B	Lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

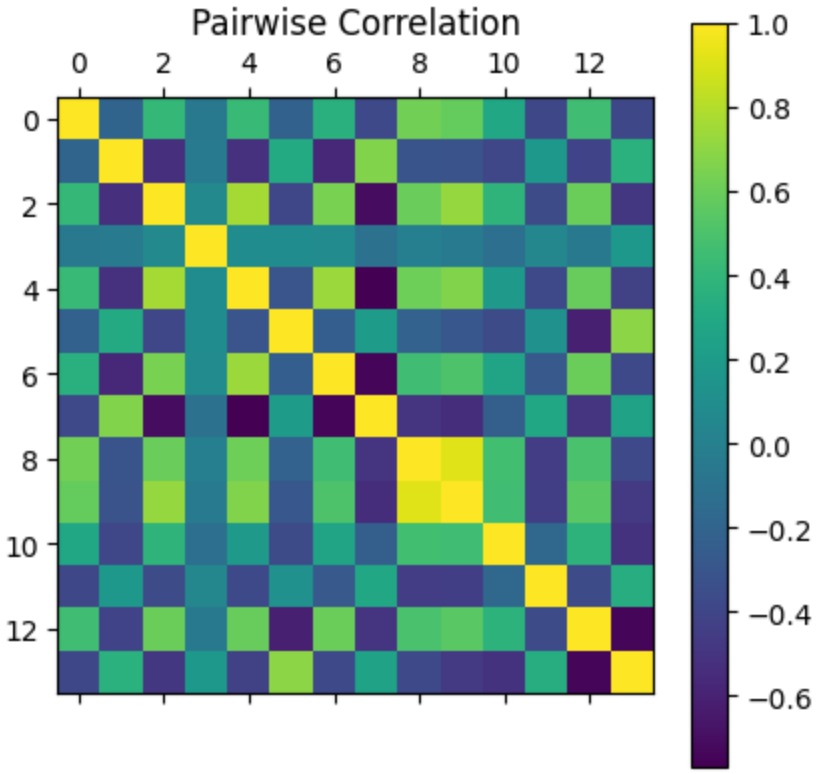
In [ ]:
```
# Show a scatter plot between Rm and Medv (x=Rm, y=Medv)
# Color >=7 Rm values as red, rest as black

color = np.where(subset['Rm'] >= 7, 'r', np.where(subset['Medv'], 'k','r'))
plt.scatter(subset['Rm'], subset['Medv'], c=color)
plt.show()
```



In [ ]:
```
# Create a scatter plot between Rm and Crim and show all 506 properties on the plot
# Color properties where Medv >= 24 as red, and rest as blue.

color = np.where(df['Medv'] >= 24, 'r', np.where(df['Crim'], 'b','r'))
scatter = plt.scatter(df['Rm'], df['Crim'], c=color)
plt.title('Rm-Crim')
plt.xlabel("Rm")
plt.ylabel("Crim")
plt.show()
```



In [ ]:
```
# Report the pairwise correlation between every two variables
# (either as a matrix or as a level plot)

pc = df.corr('pearson')
plt.matshow(pc)
plt.title('Pairwise Correlation')
cb = plt.colorbar()
cb.ax.tick_params(labelsize=10)
plt.show()
```

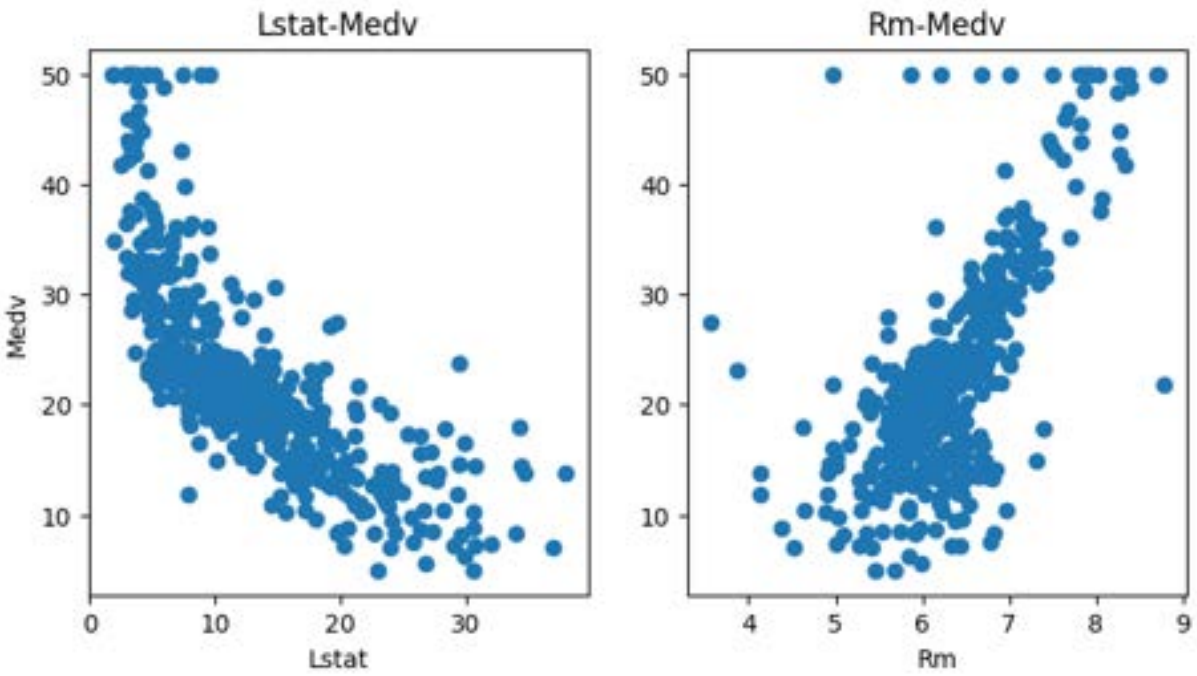


Please explain which variable is most positively correlated to Medv, and which variable is most negatively correlated to Medv.

The variable that is most positively correlated to Medv seems to be Rm; based on the pairwise correlation level plot, it is the closest to 1.0 of all the features, which suggests that it is the most positively correlated to Medv.

The variable that is most negatively correlated to Medv is Lstat; it has a correlation value of between roughly -0.6 and -0.8, which means that when Medv value goes up, Lstat value trends downwards, and vice versa.

In [ ]:
```
# Draw scatterplots to show relationship between each attribute and Medv
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(8, 4))
ax1.scatter(df['Lstat'], df['Medv'])
ax1.set_title('Lstat-Medv')
ax1.set_xlabel('Lstat')
ax1.set_ylabel('Medv')
ax2.scatter(df['Rm'], df['Medv'])
ax2.set_title('Rm-Medv')
ax2.set_xlabel('Rm')
plt.show()
```



Explain how to use scatterplots to find attributes which are positively correlated, negatively correlated, or independent of Medv, respectively.

Scatterplots can give us a good indication or visualization how the data points trend with regards to two variables. If the general slope of the data points are positive, and most points are in the trajectory of the best fit line, then there is a strong correlation between the attribute and Medv. The strength of the correlation depends on how many data points seem to adhere to this line. This is the same for negatively correlated attributes, just based on a negative/downwords slope.

If there is no pattern that can be seen, it is a good indicator that the attribute is simply independent of Medv.

In [ ]:
```
# Create a new instance with the mentioned attribute values.

tmp = {'Crim':1.0, 'Zn':0.2,'Indus':6, 'Chas':0.1, 'Nox':6.5, 'Rm':5,
       'Age':100, 'Dis':4.1,'Rad':4.5,'Tax':21,'Ptratio':20,'B':300,
       'Lstat':12,'Medv':20.5}
df.loc[len(df)] = tmp
print(df.tail(1))
```

```
     Crim   Zn  Indus  Chas  Nox   Rm    Age  Dis  Rad  Tax  Ptratio      B  \
506   1.0  0.2    6.0   0.1  6.5  5.0  100.0  4.1  4.5   21     20.0  300.0

     Lstat  Medv
506   12.0  20.5
```

In [ ]:
```
# Report the number of instances and features of the new dataframe.
df

# There are 507 instances and 14 features.
```

Out[ ]:

	Crim	Zn	Indus	Chas	Nox	Rm	Age	Dis	Rad	Tax	Ptratio	B	Ls
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222	18.7	396.90	5.
...	...	...	...	...	...	...	...	...	...	...	...	...	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273	21.0	396.90	7.
506	1.00000	0.2	6.00	0.1	6.500	5.000	100.0	4.1000	4.5	21	20.0	300.00	12.

507 rows × 14 columns

In [ ]:

```
# Create a new feature (named 'Dummy'), and include the new feature into the
# new dataframe as the last feature.
# the values of the Dummy feature are to be randomly generated within range [0,5].
import random

rand = [random.randrange(0,5) for _ in range(507)]
df['Dummy'] = rand
print(df)
```

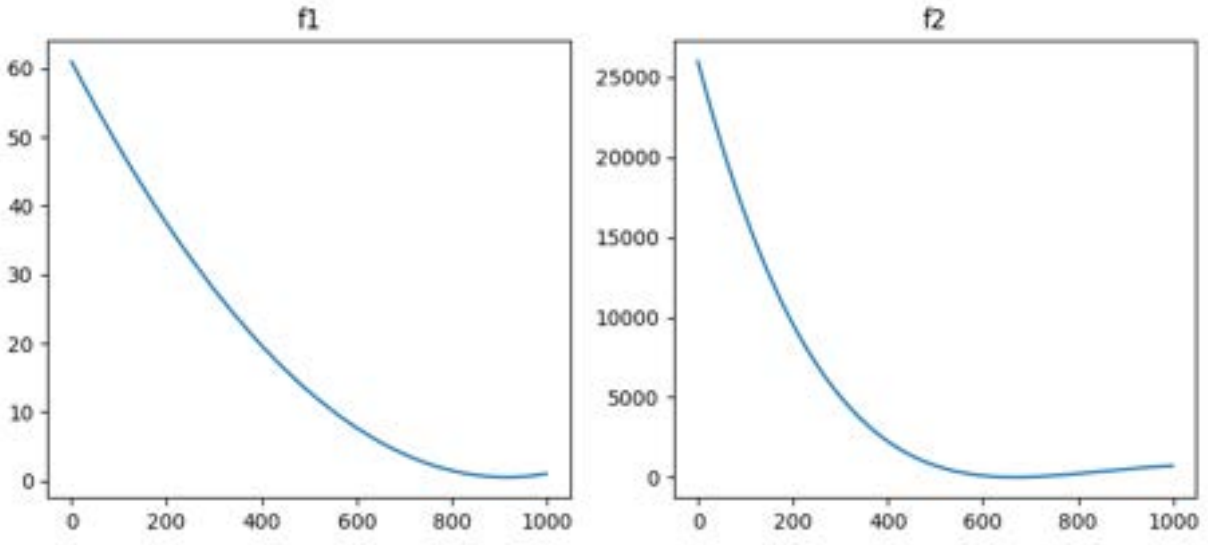
	Crim	Zn	Indus	Chas	Nox	Rm	Age	Dis	Rad	Tax	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222	
..	...	...	...	...	...	...	...	...	...	...	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273	
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273	
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273	
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273	
506	1.00000	0.2	6.00	0.1	6.500	5.000	100.0	4.1000	4.5	21	

	Ptratio	B	Lstat	Medv	Dummy
0	15.3	396.90	4.98	24.0	0
1	17.8	396.90	9.14	21.6	2
2	17.8	392.83	4.03	34.7	3
3	18.7	394.63	2.94	33.4	3
4	18.7	396.90	5.33	36.2	2
..	...	...	...	...	...
502	21.0	396.90	9.08	20.6	4
503	21.0	396.90	5.64	23.9	4
504	21.0	393.45	6.48	22.0	2
505	21.0	396.90	7.88	11.9	3
506	20.0	300.00	12.00	20.5	0

[507 rows x 15 columns]

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sb
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,4))
x = np.linspace(-3,3,1000)
y = np.linspace(-3,3,1000)

def fx1(x, y):
    return (x-2)**2+(y-3)**2
def fx2(x, y):
    return (1-(y-3))**2+20*((x+3)-(y-3)**2)**2
ax1.plot(fx1(x,y))
ax1.set_title('f1')
ax2.plot(fx2(x,y))
ax2.set_title('f2')

plt.show()
```



In [ ]:
```
# specify initial values
x1_c = 0
x2_c = 0
y1_c = 0
y2_c = 0
learning_rate = 0.5
T = 100
MIN_VAL = 99999

def dfx1(x,y):
    return 2*x-4
def dfy1(x,y):
    return 2*y-6
def dfx2(x,y):
    return 40*(x-(y-3)**2+3)
def dfy2(x,y):
```

```
    return (-80)*(-1*(y-3)**2+x+3)*(y-3)-2*(4-y)

# this gives you the result of the derivative (the gradient)
def calc_gradient(fx, x=None, y=None):
    if x is not None and y is not None:
        return fx(x, y)
    elif x is not None:
        return fx(x)
    elif y is not None:
        return fx(y)
    else:
        return None

# with gradient calculated, use
def gradient_descent(val, lr, grad):
    # return x+1
    return val - (lr * grad)

# run T amount of iterations
def run_iter(fx, dfx, dfy, xi, yi, T):
    x_c,y_c = xi, yi
    MIN_VAL = 99999.9
    for iter in range(T):
        gradx = calc_gradient(dfx, x_c, y_c)
        grady = calc_gradient(dfy, x_c, y_c)
        x_c = gradient_descent(x_c, learning_rate, gradx)
        y_c = gradient_descent(y_c, learning_rate, grady)
        print(x_c,y_c)
        if min(MIN_VAL,fx(x_c,y_c)) is fx(x_c,y_c):
            MIN_VAL = fx(x_c,y_c)
    return (x_c, y_c, MIN_VAL)
```

In [ ]:

```
# Test different inputs
x1, y1, min1 = run_iter(fx1, dfx1, dfy1, x1_c, y1_c,T)
x2, y2, min2 = run_iter(fx2, dfx2, dfy2, x2_c, y2_c, T)

print("Results for f1: x: " + x1 + "; y: " + y1 + "; optimal minimum: " + min1)
print("Results for f2: x: " + x2 + "; y: " + y2 + "; optimal minimum: " + min2)
```

```
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
```

```
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
2.0 3.0
120.0 724.0
10394480.0 -14988667116.0
4.493202840083638e+21 1.3469424333631776e+32
3.6285078375886363e+65 -9.774782352477998e+97
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[57], line 3
      1 # Test different inputs
      2 x1, y1, min1 = run_iter(fx1, dfx1, dfy1, x1_c, y1_c,T)
----> 3 x2, y2, min2 = run_iter(fx2, dfx2, dfy2, x2_c, y2_c, T)
      5 print("Results for f1: x: " + x1 + "; y: " + y1 + "; optimal minimum: " + mi
n1)
      6 print("Results for f2: x: " + x2 + "; y: " + y2 + "; optimal minimum: " + mi
n2)

Cell In[56], line 45, in run_iter(fx, dfx, dfy, xi, yi, T)
     43     y_c = gradient_descent(y_c, learning_rate, grady)
     44     print(x_c,y_c)
---> 45     if min(MIN_VAL,fx(x_c,y_c)) is fx(x_c,y_c):
     46         MIN_VAL = fx(x_c,y_c)
     47 return (x_c, y_c, MIN_VAL)

Cell In[51], line 8, in fx2(x, y)
      7 def fx2(x, y):
----> 8     return (1-(y-3))**2+20*((x+3)-(y-3)**2)**2

OverflowError: (34, 'Result too large')
```

In [ ]: