Module 6 - CKCS 113 Intro to Machine Learning

Median Example

To find the median value in a list with an **odd** amount of numbers, one would find the number that is in the middle with an equal amount of numbers on either side of the median. To find the median, first arrange the numbers in order, usually from lowest to highest.

For example, in a data set of {3, 13, 2, 34, 11, 26, 47}, the sorted order becomes {2, 3, 11, 13, 26, 34, 47}.

The median is the number in the middle {2, 3, 11, **13**, 26, 34, 47}, which in this instance is 13 since there are three numbers on either side.

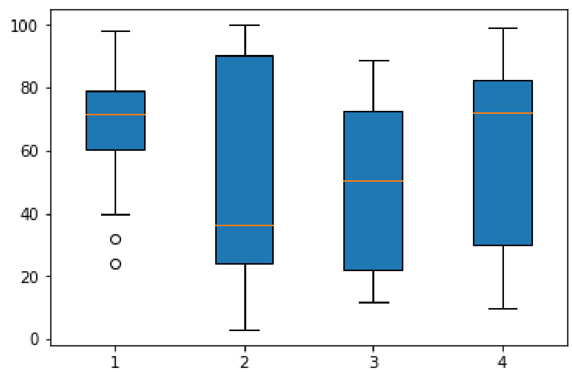
To find the median value in a list with an **even** amount of numbers, one must determine the middle pair, add them, and divide by two. Again, arrange the numbers in order from lowest to highest.

For example, in a data set of {3, 13, 2, 34, 11, 17, 27, 47}, the sorted order becomes {2, 3, 11, 13, 17, 27, 34, 47}. The median is the average of the two numbers in the middle {2, 3, 11, **13, 17**, 26 34, 47}, which in this case is fifteen {(13 + 17) ÷ 2 = 15}.

In [1]:
```
import matplotlib.pyplot as plt

value1 = [82,76,24,40,67,62,75,78,71,32,98,89,78,67,72,82,87,66,56,52]
value2=[62,5,91,25,36,32,96,95,3,90,95,32,27,55,100,15,71,11,37,21]
value3=[23,89,12,78,72,89,25,69,68,86,19,49,15,16,16,75,65,31,25,52]
value4=[59,73,70,16,81,61,88,98,10,87,29,72,16,23,72,88,78,99,75,30]

box_plot_data=[value1,value2,value3,value4]
plt.boxplot(box_plot_data, patch_artist=True)
plt.show()
```



In [2]:
```
import statistics

print(statistics.median(value1))
```

```
71.5
```

Class Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

In [3]:
```
class Employee():
    """A class representing an Employee."""
    def __init__(self,n):
        print("Name of the Employee is:",n)
```

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

In [4]:
```
class Manager(Employee):
    """A class representing a Manager."""
    def __init__(self):
        print('This is printed from the Manager Class')
```

In [5]:
```
s = Manager()
```

```
This is printed from the Manager Class
```

In [6]:
```
class Manager(Employee):
    def __init__(self):
        super().__init__("John")
        print('This is printed from the Manager Class')
```

In [7]:
```
s = Manager()
```

```
Name of the Employee is: John
This is printed from the Manager Class
```

Manual Neural Network

Manually building out a neural network that mimics the TensorFlow API. This will greatly help your understand when working with the real TensorFlow!

Operation

In [8]:
```python
class Operation():
    """
    An Operation is a node in a "Graph". TensorFlow will also use this concept of a Graph.

    This Operation class will be inherited by other classes that actually compute the specific
    operation, such as adding or matrix multiplication.
    """

    def __init__(self, input_nodes = []):
        """
        Intialize an Operation
        """
        self.input_nodes = input_nodes # The list of input nodes
        self.output_nodes = [] # List of nodes consuming this node's output

        # For every node in the input, we append this operation (self) to the list of
        # the consumers of the input nodes
        for node in input_nodes:
            node.output_nodes.append(self)

        # There will be a global default graph (TensorFlow works this way)
        # We will then append this particular operation
        # Append this operation to the list of operations in the currently active default graph
        _default_graph.operations.append(self)

    def compute(self):
        """
        This is a placeholder function. The inheritting classes will override by the actual specific operation

        """
        pass
```

Example Operations

Addition

In [9]:
```python
class add(Operation):

    def __init__(self, x, y):

        super().__init__([x, y])

    def compute(self, x_var, y_var):

        self.inputs = [x_var, y_var]
        return x_var + y_var
```

Multiplication

In [10]:
```python
class multiply(Operation):

    def __init__(self, a, b):

        super().__init__([a, b])

    def compute(self, a_var, b_var):

        self.inputs = [a_var, b_var]
        return a_var * b_var
```

Matrix Multiplication

In [11]:
```
class matmul(Operation):

    def __init__(self, a, b):

        super().__init__([a, b])

    def compute(self, a_mat, b_mat):

        self.inputs = [a_mat, b_mat]
        return a_mat.dot(b_mat)
```

Placeholders

In [12]:
```
class Placeholder():
    """
    A placeholder is a node that needs to be provided a value for computing the output in the Graph.
    """

    def __init__(self):
        self.output_nodes = []
        _default_graph.placeholders.append(self)
```

Variables

In [13]:
```
class Variable():
    """
    This variable is a changeable parameter of the Graph.
    """
    def __init__(self, initial_value = None):
        self.value = initial_value
        self.output_nodes = []
        _default_graph.variables.append(self)
```

Graph

In [14]:
```
class Graph():
    def __init__(self):
        self.operations = []
        self.placeholders = []
        self.variables = []

    def set_as_default(self):
        """
        Sets this Graph instance as the Global Default Graph
        """
        global _default_graph
        _default_graph = self
```

A Basic Graph

$$z = Ax + b$$

With A=10 and b=1

$$z = 10x + 1$$

Just need a placeholder for x and then once x is filled in we can solve it!

In [15]:
```
g = Graph()
```

In [16]:
```
g.set_as_default()
```

In [17]:
```
A = Variable(10)
```

In [18]:
```
b = Variable(1)
```

In [19]:
```
# Will be filled out later
x = Placeholder()
```

In [20]:
```
y = multiply(A,x)
```

In [21]:
```
z = add(y,b)
```

Creating Session

In [22]:
```
import numpy as np
```

Traversing Operation Nodes

In [23]:
```
def traverse_postorder(operation):
    """
    PostOrder Traversal of Nodes. Basically makes sure computations are done in
    the correct order (mx first , then mx + c). Feel free to copy and paste this code.
    It is not super important for understanding the basic fundamentals of deep learning.
    """

    nodes_postorder = []
    def recurse(node):
        if isinstance(node, Operation):
            for input_node in node.input_nodes:
                recurse(input_node)
        nodes_postorder.append(node)

    recurse(operation)
    return nodes_postorder
```

In [24]:
```
class Session:

    def run(self, operation, feed_dict = {}):
        """
          operation: The operation to compute
          feed_dict: Dictionary mapping placeholders to input values (the data)
        """

        # Puts nodes in correct order
        nodes_postorder = traverse_postorder(operation)

        for node in nodes_postorder:

            if type(node) == Placeholder:

                node.output = feed_dict[node]

            elif type(node) == Variable:

                node.output = node.value

            else: # Operation

                node.inputs = [input_node.output for input_node in node.input_nodes]


                node.output = node.compute(*node.inputs)

            # Convert lists to numpy arrays
            if type(node.output) == list:
                node.output = np.array(node.output)

        # Return the requested node value
        return operation.output
```

Let's explore how a session will work

In [25]:
```
sess = Session()
```

In [26]:
```
result = sess.run(operation=z,feed_dict={x:10})
```

In [27]:
```
result
```

Out[27]: 101

In [28]:
```
10*10 + 1
```

Out[28]: 101

Lets try this with a Matrix multiplications, which is more common in tensorflow

In [29]:
```
g = Graph()

g.set_as_default()
#Here variables are matrix
A = Variable([[10,20],[30,40]])
b = Variable([1,1])
#x is a placeholder which is waiting a value
x = Placeholder()

y = matmul(A,x)

z = add(y,b)
```

In [30]:
```
sess = Session()
```

In [31]:
```python
result = sess.run(operation=z,feed_dict={x:10})
```

In [32]:
```python
result
```

Out[32]:
```
array([[101, 201],
       [301, 401]])
```

Activation Function

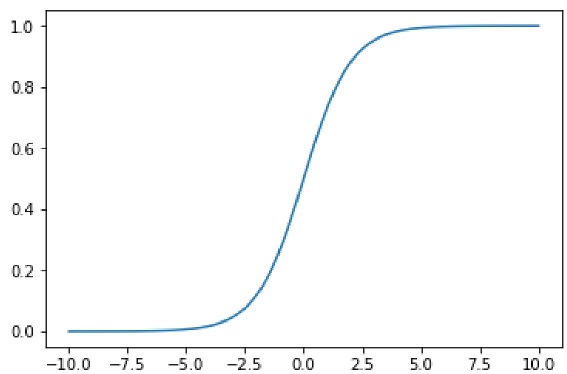
In [33]:
```python
import matplotlib.pyplot as plt
%matplotlib inline
```

In [34]:
```python
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

In [35]:
```python
sample_z = np.linspace(-10,10,100)
sample_a = sigmoid(sample_z)
```

In [36]:
```python
plt.plot(sample_z,sample_a)
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x25c4fcc7860>]
```



Sigmoid as an Operation

In [37]:
```python
class Sigmoid(Operation):


    def __init__(self, z):

        # a is the input node
        super().__init__([z])

    def compute(self, z_val):

        return 1/(1+np.exp(-z_val))
```

Classification Example

In [38]:
```python
from sklearn.datasets import make_blobs
```

In [39]:
```python
data = make_blobs(n_samples = 50,n_features=2,centers=2,random_state=75)
```

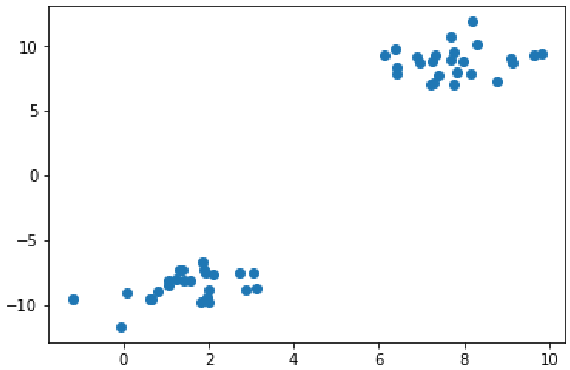
In [40]:
```
data
```

Out[40]:
```
(array([[  7.3402781 ,   9.36149154],
        [  9.13332743,   8.74906102],
        [  1.99243535,  -8.85885722],
        [  7.38443759,   7.72520389],
        [  7.97613887,   8.80878209],
        [  7.76974352,   9.50899462],
        [  8.3186688 ,  10.1026025 ],
        [  8.79588546,   7.28046702],
        [  9.81270381,   9.46968531],
        [  1.57961049,  -8.17089971],
        [  0.06441546,  -9.04982817],
        [  7.2075117 ,   7.04533624],
        [  9.10704928,   9.0272212 ],
        [  1.82921897,  -9.86956281],
        [  7.85036314,   7.986659  ],
        [  3.04605603,  -7.50486114],
        [  1.85582689,  -6.74473432],
        [  2.88603902,  -8.85261704],
        [ -1.20046211,  -9.55928542],
        [  2.00890845,  -9.78471782],
        [  7.68945113,   9.01706723],
        [  6.42356167,   8.33356412],
        [  8.15467319,   7.87489634],
        [  1.92000795,  -7.50953708],
        [  1.90073973,  -7.24386675],
        [  7.7605855 ,   7.05124418],
        [  6.90561582,   9.23493842],
        [  0.65582768,  -9.5920878 ],
        [  1.41804346,  -8.10517372],
        [  9.65371965,   9.35409538],
        [  1.23053506,  -7.98873571],
        [  1.96322881,  -9.50169117],
        [  6.11644251,   9.26709393],
        [  7.70630321,  10.78862346],
        [  0.79580385,  -9.00301023],
        [  3.13114921,  -8.6849493 ],
        [  1.3970852 ,  -7.25918415],
        [  7.27808709,   7.15201886],
        [  1.06965742,  -8.1648251 ],
        [  6.37298915,   9.77705761],
        [  7.24898455,   8.85834104],
        [  2.09335725,  -7.66278316],
        [  1.05865542,  -8.43841416],
        [  6.43807502,   7.85483418],
        [  6.94948313,   8.75248232],
        [ -0.07326715, -11.69999644],
        [  0.61463602,  -9.51908883],
        [  1.31977821,  -7.2710667 ],
        [  2.72532584,  -7.51956557],
        [  8.20949206,  11.90419283]]),
 array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
        1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
        1, 0, 0, 0, 0, 1]))
```

Here a tuple of features and labels is generated.

In [41]:
```
features = data[0]
plt.scatter(features[:,0],features[:,1])
```

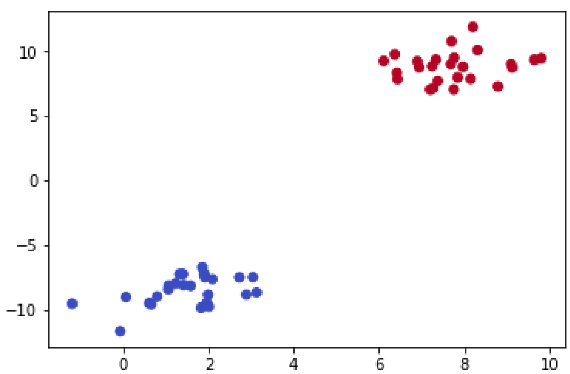
Out[41]: <matplotlib.collections.PathCollection at 0x25c5083f240>



To give different color to each feature c = labels and cmap='coolwarm'

In [42]:
```
labels = data[1]
plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')
```

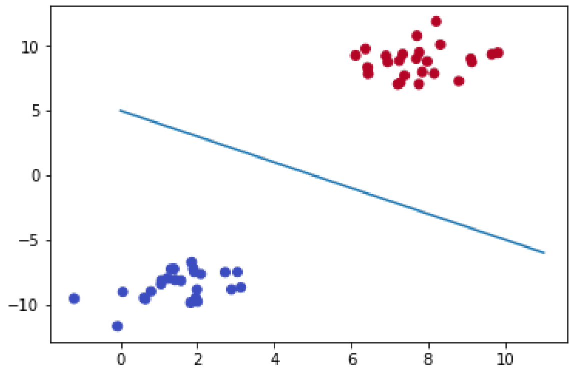
Out[42]: <matplotlib.collections.PathCollection at 0x25c508a20b8>



Lets try to DRAW A LINE THAT SEPERATES CLASSES

In [43]:
```
x = np.linspace(0,11,10)
y = -x + 5
plt.scatter(features[:,0],features[:,1],c=labels,cmap='coolwarm')
plt.plot(x,y)
```

Out[43]: `[<matplotlib.lines.Line2D at 0x25c50868198>]`



Defining the Perceptron

$$y = mx + b$$

$$y = -x + 5$$

$$f1 = mf2 + b, m = 1$$

$$f1 = -f2 + 5$$

$$f1 + f2 - 5 = 0$$

Convert to a Matrix Representation of Features

$$w^T x + b = 0$$

$$\begin{pmatrix} 1, 1 \end{pmatrix} f - 5 = 0$$

Then if the result is > 0 its label 1, if it is less than 0, it is label=0

Example Point

Let's say we have the point f1=2 , f2=2 otherwise stated as (8,10). Then we have:

$$\begin{pmatrix} 1, 1 \end{pmatrix} \begin{pmatrix} 8 \\ 10 \end{pmatrix} + 5 =$$

In [44]:
```
np.array([1, 1]).dot(np.array([[8],[10]])) - 5
```

Out[44]: `array([13])`

Or if we have (4,-10)

In [45]:
```
np.array([1,1]).dot(np.array([[4],[-10]])) - 5
```

Out[45]: `array([-11])`

Using an Example Session Graph

In [46]:
```
g = Graph()
```

In [47]:
```
g.set_as_default()
```

x is a feature array

In [48]:
```
x = Placeholder()
```

In [49]:
```
w = Variable([1,1])
```

In [50]:
```
b = Variable(-5)
```

In [51]:
```
z = add(matmul(w,x),b)
```

In [52]:
```
a = Sigmoid(z)
```

Activation function

In [53]:
```
sess = Session()
```

In [54]:
```
sess.run(operation=a,feed_dict={x:[8,10]})
```

Out[54]:
```
0.999997739675702
```

In [55]:
```
sess.run(operation=a,feed_dict={x:[0,-10]})
```

Out[55]:
```
3.059022269256247e-07
```

[http://playground.tensorflow.org/ (http://playground.tensorflow.org/)](http://playground.tensorflow.org/)