20180115_COGS118a_Hw1

January 12, 2018

1

```
In [1]: import numpy as np
        A = np.array([[1,2], [3,4], [5,6]])
        B = np.array([[1,-1], [-1,1], [1,-1]])
```

1.1

$A + B$

```
In [2]: A + B

Out[2]: array([[2, 1],
               [2, 5],
               [6, 5]])
```

1.2

$A \circ B$

```
In [3]: np.multiply(A,B)

Out[3]: array([[ 1, -2],
               [-3,  4],
               [ 5, -6]])
```

1.3

$A^T B$

```
In [4]: np.dot(A.T, B)

Out[4]: array([[ 3, -3],
               [ 4, -4]])
```

1.4

AB^T

```
In [5]: np.dot(A, B.T)
```

```
Out[5]: array([[-1,  1, -1],
               [-1,  1, -1],
               [-1,  1, -1]])
```

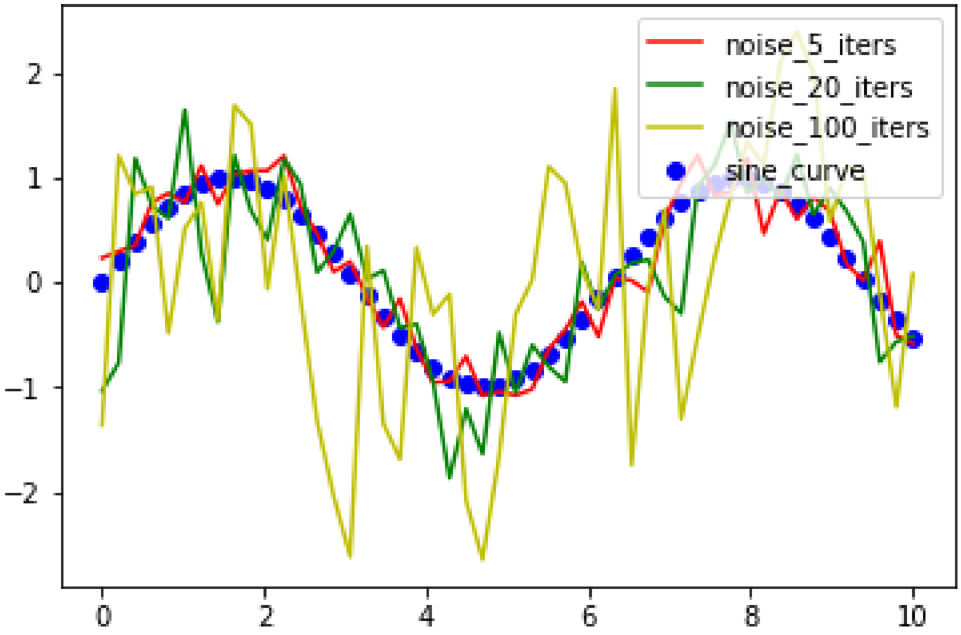
1.5

AB

impossible

2

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt

        np.random.seed(0)
        space = np.linspace(0, 10, num = 50)
        sine = np.sin(space)
        sine_5 = sine
        sine_20 = sine
        sine_100 = sine

        for i in range(5):
            sine_5 = sine_5 + np.random.normal(scale = 0.1, size = 50)

        for i in range(20):
            sine_20 = sine_20 + np.random.normal(scale=0.1, size = 50)

        for i in range(100):
            sine_100 = sine_100 + np.random.normal(scale=0.1, size = 50)

        plt.scatter(space, sine, color = 'b', label = 'sine_curve')
        plt.plot(space, sine_5, color = 'r', label = 'noise_5_iters')
        plt.plot(space, sine_20, color = 'g', label = 'noise_20_iters')
        plt.plot(space, sine_100, color = 'y', label = 'noise_100_iters')
        plt.legend(loc = 'upper right')
        plt.savefig('./Q2.png')
        plt.show()
```



3

3.1

```
In [7]: import matplotlib.pyplot as plt
        img = plt.imread("cat.jpg")
        plt.imshow(img)
        plt.show()
```

3.2

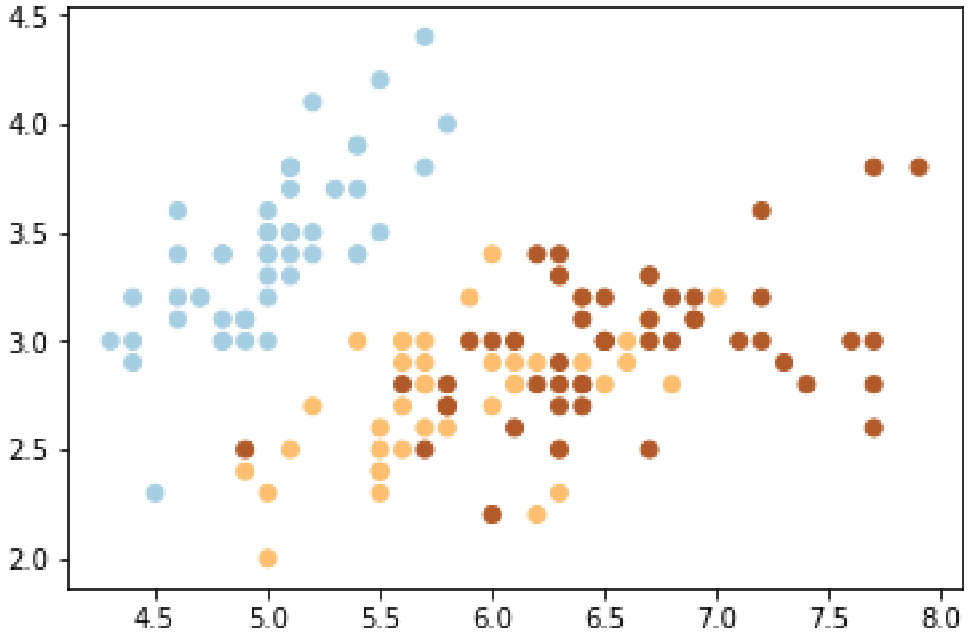
```
In [8]: img.shape

Out[8]: (183, 275, 3)
```

4

```
In [9]: import matplotlib.pyplot as plt
        from sklearn import datasets
        from mpl_toolkits.mplot3d import Axes3D

        iris = datasets.load_iris()
        X = iris.data
        Y = iris.target
```

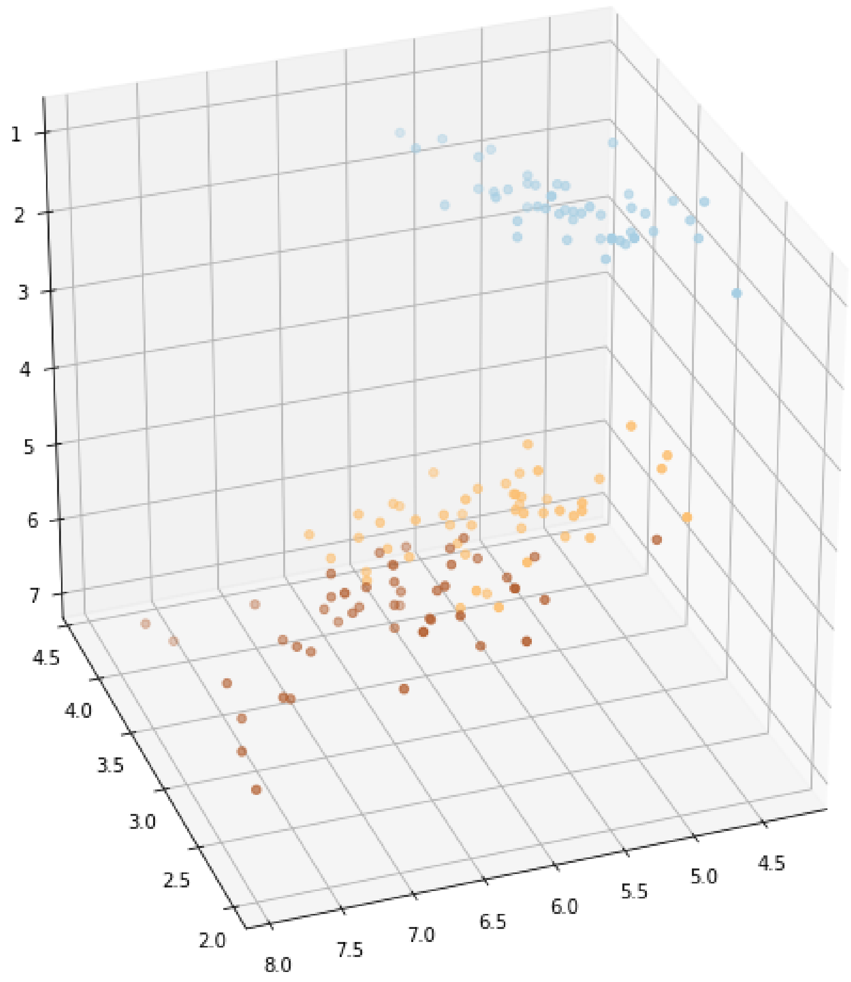
4.1

```
In [10]: plt.scatter(X[:,0], X[:,1], c = Y, cmap = plt.cm.Paired)
         plt.show()
```



4.2

```
In [11]: fig = plt.figure(1, figsize = (8, 8))
         ax = Axes3D(fig, elev = -150, azim = 110)
         ax.scatter(X[:, 0], X[:, 1], X[:, 2], c = Y, cmap = plt.cm.Paired)
         plt.show()
```



5

5.1

With unlimited computing power I would design an automatic debugger that would be able to catch all my coding errors before I make them.

5.2

As input to this personal Watson I would use source code that others have written that were known to have bugs and then the source code after the bug fix.

5.3

Because source code is already in machine readable format, Watson would already be able to process the input.

5.4

1. Watson would have to be able to discern between different programming languages, even those with very similar syntax (C vs C++).
2. Some bugs may be too complex to determine from before and after snapshots. For instance, context (domain knowledge) may be needed to deem a particular code change as necessary for fixing a bug.
3. May be theoretically impossible (Turing's halting problem).

6

6.1

```
In [12]: print(X[:5, :3])
```

```
[[ 5.1  3.5  1.4]
 [ 4.9  3.   1.4]
 [ 4.7  3.2  1.3]
 [ 4.6  3.1  1.5]
 [ 5.   3.6  1.4]]
```

6.2

```
In [13]: print(f"Mean: {X[:, 2].mean()}")
         print(f"Variance: {X[:, 2].var()}")
```

```
Mean: 3.758666666666666
Variance: 3.092424888888889
```

6.3

```
In [14]: w = np.array([1,2,3,4])
         np.dot(X, w).mean()
```

```
Out[14]: 28.022000000000006
```

6.4

```
In [15]: for i in range(4):
             idx = np.random.randint(0, 150)
             print(f"Index: {idx} | Data: {X[idx]} ")
```

```
Index: 91 | Data: [ 6.1  3.   4.6  1.4]
Index: 29 | Data: [ 4.7  3.2  1.6  0.2]
Index: 7 | Data: [ 5.   3.4  1.5  0.2]
Index: 2 | Data: [ 4.7  3.2  1.3  0.2]
```

6.5

```
In [16]: X1 = np.ones((150, 1))
         X = np.hstack((X, X1))

         print(X[0])
```

```
[ 5.1  3.5  1.4  0.2  1. ]
```