CPSC 340 A1

Zachary Chua 48995336

January 14 2020

1 Linear Algebra

1.1 Basic Operations

1. 15
2. 0
3. $\begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix}$
4. $\sqrt{5}$
5. $\begin{bmatrix} 6 \\ 5 \\ 7 \end{bmatrix}$
6. 19
7. $\begin{bmatrix} 11 & 10 & 10 \\ 10 & 14 & 10 \\ 10 & 10 & 14 \end{bmatrix}$

1.2 Matrix Algebra Rules

1. True
2. True
3. False
4. True
5. False
6. True
7. False

8. True
9. True
10. True

2 Probability

2.1 Rules of probability

1. 2.5
2. 0.55
3. 0.917

2.2 Bayes Rule and Conditional Probability

1. 0.010097
2. False positives
3. 0.0096
4. No
5. Repeat drug tests for positive users

3 Calculus

3.1 One-variable derivatives

1. 6x-2
2. 1-2x
3. $\exp(x)$

3.2 Multi-variable derivatives

1. $\begin{bmatrix} 2x_1 + \exp(x_1 + 2x_2) \\ 2\exp(x_1 + 2x_2) \end{bmatrix}$
2. $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
3. a^T
4. $0.5(A + A^T)x$
5. 2x

3.3 Optimization

1. 4.67
2. 0.25
3. 0
4. 0.5
5. $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$
6. $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

3.4 Derivatives of code

```
def foo_grad(x): # getting specific gradient vector so for function foo
    grad = np.zeros((len(x),), float)
    for i,x_i in enumerate(x):
        grad[i] = 4*x_i**3
    return grad

def bar(x): #Return the product of array elements over a given axis. ie [1,
    return np.prod(x) #xyz

def bar_grad(x):
    grad = np.zeros((len(x),), float)
    for i,x_i in enumerate(x):
        grad[i] = np.prod(x)/x_i
    return grad
```

4 Algorithms and Data Structures Review

4.1 Trees

1. 6
2. 6

4.2 Common Runtimes

1. O(n)
2. O(lg n)
3. O(1)
4. O(d)
5. $O(d^2)$

4.3 Running times of code

1. O(N) 2. O(N) 3. O(1) 4. $O(N^2)$

5 Data Exploration

5.1 Summary Statistics

Minimum:
NE 0.428
MidAtl 0.483
ENCentral 0.452
WNCentral 0.464
SAt1 0.468
ESCentral 0.554
WSCentral 0.456
Mtn 0.352
Pac 0.377
WtdILI 0.606

Maximum:
NE 2.310
MidAtl 2.205
ENCentral 2.515
WNCentral 3.115
SAt1 2.714
ESCentral 3.859
WSCentral 3.219
Mtn 4.862
Pac 2.660
WtdILI 3.260

Mean:
NE 1.223346
MidAtl 1.233538
ENCentral 1.275269
WNCentral 1.460212

SAt1 1.298827
ESCentral 1.562519
WSCentral 1.292346
Mtn 1.270019
Pac 1.063212
WtdILI 1.566962

Median:
NE 1.1295
MidAtl 1.1160
ENCentral 1.2650
WNCentral 1.2775
SAt1 1.1025
ESCentral 1.4165
WSCentral 1.1075
Mtn 0.9785
Pac 0.9570
WtdILI 1.3035

Mode:

	NE	MidAtl	ENCentral	WNCentral	SAt1	ESCentral	WSCentral	Mtn	Pac	WtdILI
0	0.428	0.49	0.452	0.505	0.542	0.554	0.499	0.352	0.377	0.715
1	0.432	NaN	0.454	1.286	0.616	0.558	NaN	0.380	0.385	NaN
2	0.434	NaN	0.468	1.794	1.012	0.566	NaN	0.391	0.388	NaN
3	0.437	NaN	0.470	NaN	NaN	0.587	NaN	0.404	0.409	NaN
4	0.444	NaN	0.488	NaN	NaN	0.595	NaN	0.430	0.414	NaN
5	0.465	NaN	0.490	NaN	NaN	0.600	NaN	0.439	0.422	NaN
6	0.480	NaN	0.491	NaN	NaN	0.612	NaN	0.452	0.442	NaN
7	0.495	NaN	0.522	NaN	NaN	0.642	NaN	0.459	0.452	NaN
8	0.498	NaN	0.526	NaN	NaN	0.653	NaN	0.474	0.453	NaN
9	0.525	NaN	0.540	NaN	NaN	0.721	NaN	0.521	0.499	NaN
10	0.570	NaN	0.589	NaN	NaN	0.769	NaN	0.525	0.502	NaN
11	0.600	NaN	0.604	NaN	NaN	0.829	NaN	0.564	0.540	NaN
12	0.684	NaN	0.667	NaN	NaN	0.893	NaN	0.568	0.541	NaN
13	0.743	NaN	0.726	NaN	NaN	0.932	NaN	0.643	0.561	NaN
14	0.770	NaN	0.792	NaN	NaN	0.974	NaN	0.649	0.617	NaN
15	0.795	NaN	0.896	NaN	NaN	0.987	NaN	0.685	0.652	NaN
16	0.868	NaN	0.906	NaN	NaN	1.006	NaN	0.744	0.667	NaN
17	0.890	NaN	0.913	NaN	NaN	1.024	NaN	0.770	0.709	NaN
18	0.911	NaN	0.954	NaN	NaN	1.033	NaN	0.846	0.736	NaN
19	0.949	NaN	0.994	NaN	NaN	1.039	NaN	0.884	0.759	NaN
20	0.964	NaN	1.065	NaN	NaN	1.144	NaN	0.903	0.770	NaN
21	0.970	NaN	1.141	NaN	NaN	1.208	NaN	0.912	0.820	NaN
22	1.095	NaN	1.184	NaN	NaN	1.279	NaN	0.914	0.894	NaN
23	1.096	NaN	1.228	NaN	NaN	1.385	NaN	0.937	0.895	NaN
24	1.121	NaN	1.232	NaN	NaN	1.394	NaN	0.964	0.917	NaN
25	1.124	NaN	1.258	NaN	NaN	1.400	NaN	0.976	0.943	NaN
26	1.135	NaN	1.272	NaN	NaN	1.433	NaN	0.981	0.971	NaN
27	1.136	NaN	1.278	NaN	NaN	1.457	NaN	1.056	1.018	NaN
28	1.212	NaN	1.295	NaN	NaN	1.556	NaN	1.102	1.082	NaN
29	1.285	NaN	1.356	NaN	NaN	1.644	NaN	1.275	1.105	NaN
30	1.354	NaN	1.460	NaN	NaN	1.662	NaN	1.321	1.118	NaN
31	1.365	NaN	1.482	NaN	NaN	1.794	NaN	1.371	1.137	NaN
32	1.390	NaN	1.506	NaN	NaN	1.805	NaN	1.388	1.204	NaN
33	1.470	NaN	1.530	NaN	NaN	1.813	NaN	1.403	1.350	NaN
34	1.477	NaN	1.557	NaN	NaN	1.820	NaN	1.453	1.351	NaN
35	1.502	NaN	1.576	NaN	NaN	1.894	NaN	1.628	1.384	NaN
36	1.520	NaN	1.581	NaN	NaN	1.903	NaN	1.641	1.406	NaN
37	1.541	NaN	1.617	NaN	NaN	1.952	NaN	1.678	1.443	NaN
38	1.632	NaN	1.638	NaN	NaN	2.059	NaN	1.694	1.453	NaN
39	1.756	NaN	1.711	NaN	NaN	2.159	NaN	1.701	1.473	NaN
40	1.784	NaN	1.822	NaN	NaN	2.194	NaN	1.770	1.497	NaN
41	1.815	NaN	1.901	NaN	NaN	2.220	NaN	1.826	1.506	NaN
42	1.860	NaN	1.914	NaN	NaN	2.230	NaN	1.849	1.516	NaN
43	1.984	NaN	1.955	NaN	NaN	2.252	NaN	1.852	1.554	NaN
44	2.102	NaN	2.050	NaN	NaN	2.315	NaN	1.856	1.566	NaN
45	2.114	NaN	2.065	NaN	NaN	2.498	NaN	1.867	1.569	NaN
46	2.144	NaN	2.068	NaN	NaN	2.657	NaN	1.943	1.573	NaN
47	2.147	NaN	2.131	NaN	NaN	2.866	NaN	2.132	1.717	NaN
48	2.208	NaN	2.151	NaN	NaN	3.027	NaN	2.364	2.094	NaN
49	2.280	NaN	2.370	NaN	NaN	3.318	NaN	3.352	2.181	NaN
50	2.303	NaN	2.463	NaN	NaN	3.480	NaN	3.890	2.595	NaN
51	2.310	NaN	2.515	NaN	NaN	3.859	NaN	4.862	2.660	NaN

5.2 Data Visualization

1. C. Each column
2. D. Every value
3. B. Boxplot
4. A. Simple plot
5. F. Linear relationship
6. E. No clear relationship

6 Decision Trees

6.1 Splitting rule

Binary features would make equality-based splitting make sense

6.2 Decision Stump Implementation

Decision Stump with inequality rule error: 0.265

```
for d in range(D): # first longitude then latitude
    for n in range(N):
        # Choose value to equate to
        value = X[n, d] # for each longitude and latitude, this is threshho

        # Find most likely class(1 or 0) for each split(the split here is e
        y_sat = utils.mode(y[X[:,d] < value]) #X[:,d] means keep column d
        # so for each longitude see how many 1 or 0 and then take the most
        y_not = utils.mode(y[X[:,d] >= value])

        # Make predictions
        y_pred = y_sat * np.ones(N) #0 or 1 * array of ones
        y_pred[X[:, d] >= value] = y_not # replace index of cities not at l
        # print(y_pred)
        # places where the city does not match longitude find the index and
        # X[:, d] != value is true or false
        # Compute error
        errors = np.sum(y_pred != y)

        # Compare to minimum error so far
        if errors < minError:
```

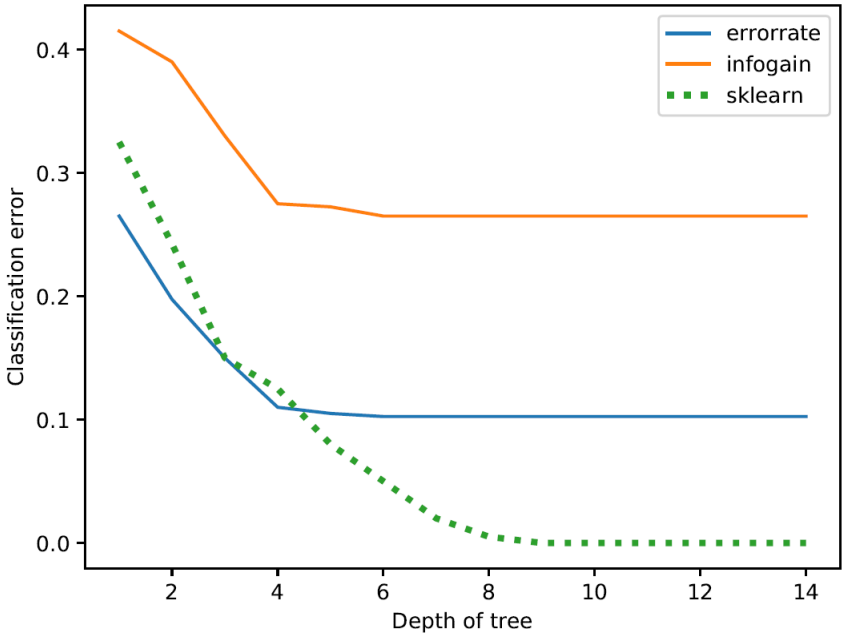
```python
def predict(self, X):

    M, D = X.shape
    X = np.round(X)

    if self.splitVariable is None:
        return self.splitSat * np.ones(M)

    yhat = np.zeros(M)

    for m in range(M):
        if X[m, self.splitVariable] < self.splitValue:
            yhat[m] = self.splitSat
        else:
            yhat[m] = self.splitNot

    return yhat
```

6.3 Decision Stump Info Gain Implementation

Error is higher than expected at 0.410.

6.4 Constructing Decision Trees

6.5 Decision Tree Training Error



The error rate method does not converge closely to 0 and flattens at 0.1.

6.6 Comparing implementations

No one might have a longer running time than the other. I would test running times of both on the same dataset

6.7 Cost of Fitting Decision Trees

There are $2^m - 1$ trees so $O(2^{(m-1)}nd \log n)$
However only one object appears at depth so it is $O(mnd \log n)$