Perceptron Learning

February 27, 2019

1 Perceptrons Learning

by Sagar Jain (sj735)

Generating Data Set >>>

```
In [49]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
```

```
In [157]: def dataframe_function(k=20, m=100,epsilon=1):

              data = []

              def vector_generation_function(k=k,epsilon=epsilon):
                  X = []
                  Y = 0
                  for x in range(1,k+1):
                      if x >= 1 and x <= k-1:
                          mean = 0
                          sd = 1
                          X.append(np.random.normal(mean, sd))
                      else:
                          X1 = np.random.exponential(1)
                          X.append(np.random.choice([X1+epsilon,-(X1+epsilon)], 1, p=[0.5, 0.5]

                  if X[k-1] > 0:
                      X.append(1)
                  else:
                      X.append(-1)

                  return X

              for x in range(1, m+1):
                  data.append(vector_generation_function(k))

              #Create header list
              headers = ['X'+str(x) for x in range(1,k+1)] + ['Y']
```

```
        dataframe = pd.DataFrame(data, columns=headers)

        return dataframe
```

```
In [158]: df = dataframe_function(20,100,epsilon=1)
```

```
In [159]: print(df)
```

```
          X1        X2        X3        X4        X5        X6        X7  \
0   1.535817 -1.419101  0.016050 -0.021452  1.684866 -1.332180 -1.619824
1  -0.933247 -1.336097  0.069984 -0.449854  0.722462  0.095052 -0.265306
2  -1.135426  0.862697  0.863542 -0.499595 -0.518582 -0.751083 -0.781932
3  -0.149086 -1.534156 -1.096022  0.714272  0.223301  0.248707  0.514388
4  -0.199361  1.179314 -1.268321  0.090409 -0.599559  0.761337 -1.259422
5   1.254271  0.607360 -1.242738  0.890274 -1.564233 -1.223744  0.765984
6  -1.513953  0.074638  0.728523 -0.689135 -0.119806 -1.570104 -1.049375
7  -2.023537  0.892188 -0.309214  0.560868 -2.067710  0.816765  0.535350
8   0.009481  0.617670 -0.301038  1.445177  0.029085  1.392239 -0.901956
9  -0.290127  0.927900  1.401828  1.359908  0.482172  0.258394  1.088674
10  0.773078  0.164436  0.215175 -0.780082  2.233510 -0.588813  0.027732
11  1.694182  0.576981 -0.435074  1.283452  1.452752  2.365205 -1.699005
12 -0.082600 -1.469900 -1.265428 -2.219238  0.160911  0.880268 -0.618289
13 -0.606438  2.230559  0.992614 -0.366950 -0.722049 -0.000069 -0.446467
14  0.863901 -0.491102 -1.117681  0.706323 -0.076111  1.659486  0.947813
15  0.404884 -0.338032  0.775611  1.137594  2.234050 -1.160878 -0.396923
16  1.160416  0.816825 -0.324093 -0.415729 -0.756309  0.444077  2.156621
17  0.573003 -0.172841 -0.560713  0.561434  0.206086 -1.401452  0.427229
18 -0.587279 -2.239901 -0.721535 -0.410061 -2.164068 -0.586267 -0.367914
19  1.129677  0.077777  0.277238 -0.449291 -0.684495 -1.520179 -1.999676
20  0.102063  0.292102  1.632110 -1.355097 -0.696446  1.242605  0.519781
21  0.492292  0.210710 -1.147535 -0.805025  0.378355 -1.349253 -1.180767
22  0.183631 -0.801360 -0.487433 -0.720364 -0.145609  0.466516 -0.387494
23 -0.375199  1.196749  1.157582 -0.256042 -0.791533  0.973055 -1.766066
24  1.354649 -0.671441  0.082304  0.852767 -0.253959  0.218592  0.365721
25  0.235470  1.088696 -0.731011  1.125210 -0.937767  0.872427  0.067069
26 -1.075841 -0.392285 -0.450100  0.480928 -0.138313 -0.180344 -1.739589
27  0.965862 -2.820598  0.467628 -0.922432  0.801797 -2.207561 -3.911880
28 -0.871933 -2.259202 -0.772828  0.059059  0.317196 -1.011249  0.114017
29  0.533063 -1.823843  0.339928 -1.519395 -1.471347  0.946030 -1.153715
..       ...       ...       ...       ...       ...       ...       ...
70  0.228474 -2.272568 -0.586011 -0.138747 -2.047411 -0.054626 -1.108776
71 -0.232164 -0.431479 -0.889738 -0.445929  0.208665  0.192233  1.527591
72 -0.255224  1.053022  0.331288  0.881576 -0.708996  0.806739  0.429382
73  1.105198 -0.142744  0.047223 -0.879209  0.550618  0.650624 -0.525492
74  1.041927 -0.565227 -2.094045 -0.368636 -0.167534 -1.067250  1.391636
75  2.218026 -1.135586 -0.056334 -0.401059  0.839797  1.330440 -0.361564
76  0.629758 -0.033198  0.930807  0.381683  1.752751 -0.488225 -1.468641
```

77	1.639613	-0.221588	-1.077892	-0.409405	-0.502600	-0.531014	-1.533565
78	0.655481	0.395476	-0.285088	1.511467	0.119043	-1.554160	-0.913247
79	-0.025730	0.224182	0.550662	-0.603153	-1.254247	0.660419	0.114135
80	-1.619123	-0.099738	-0.956523	0.573105	0.570394	0.346095	0.059610
81	0.949289	0.885167	-0.696787	-0.747221	-0.851655	-1.595308	-1.573865
82	-0.012536	0.390098	1.792077	1.374411	0.038230	1.646716	-0.266116
83	-1.058239	-1.504467	-0.068503	0.720249	-0.019838	-0.097464	0.413147
84	0.709179	-1.412176	0.025726	1.930025	2.200724	-2.029496	-1.026095
85	-0.795579	0.186339	2.282425	0.740441	0.773815	-0.990607	0.725351
86	-1.098786	-0.559732	1.328667	-1.483963	-1.251557	-0.274948	0.759456
87	-0.291053	-1.111744	-1.433321	0.401052	-0.257667	-0.036484	-1.358490
88	0.562757	1.883950	-1.307309	-0.359735	1.198147	1.855311	-0.245621
89	-0.254578	1.642521	-0.509485	-0.118705	0.697767	-0.884757	-1.701519
90	1.542125	-0.548169	-0.518468	1.737394	1.688015	1.218437	-0.236060
91	1.218322	0.090557	0.018371	-0.436709	-1.397109	0.054808	-0.235144
92	-0.447425	-1.063971	0.650865	1.134665	-0.784260	1.077094	1.908885
93	-2.277840	0.943037	0.672734	-0.537178	-1.176886	1.033394	1.351107
94	0.334519	2.390546	-0.670493	-0.083270	0.405738	-0.998541	-0.270342
95	0.005817	0.198330	-0.060185	0.528978	0.922171	-0.076635	0.426478
96	-0.369162	0.162771	1.287196	0.606610	2.010138	0.331861	-0.156470
97	-0.703713	-0.745664	0.256499	0.633306	-0.113174	1.069847	0.298819
98	0.296243	-1.153907	1.785567	-0.887815	-0.315995	-2.153775	-0.794411
99	0.672182	-0.103402	1.067955	1.649859	0.254091	1.793822	-0.756709

	X8	X9	X10	...	X12	X13	X14	X15	\
0	0.025468	-1.450834	0.888356	...	0.442380	0.105203	0.834319	-0.478613	
1	0.593864	1.136802	-0.206657	...	-1.041813	0.305410	0.712904	1.704276	
2	0.719329	1.150456	0.612948	...	-1.213056	0.093513	-0.410289	-1.358519	
3	-0.735098	0.236947	-1.868782	...	0.092592	-1.452506	-1.254920	0.067823	
4	-1.086473	0.149841	0.323467	...	-0.339502	-0.312341	1.275347	-0.401146	
5	0.467984	-1.147057	-0.853201	...	-0.024872	0.170505	0.422821	0.541833	
6	1.662931	-1.002620	-0.995524	...	-1.372608	-0.084437	0.436705	0.632355	
7	-2.310296	0.500844	-0.123123	...	-0.221790	1.065209	-0.279663	0.489284	
8	-0.857276	0.106111	0.489864	...	0.180625	-0.720701	-0.451008	1.407206	
9	-0.460005	-0.769071	1.716316	...	-0.200036	0.071072	-0.969965	1.266447	
10	0.036950	-1.429909	0.234495	...	0.381887	-0.348961	0.140988	-1.333756	
11	1.109204	0.502828	-0.164233	...	-0.788268	-2.871252	0.033722	1.545170	
12	1.015401	-1.544890	0.040192	...	0.673948	0.512896	-1.465763	-0.168417	
13	-1.251130	-0.520203	0.746330	...	0.951292	-0.059592	-1.764163	0.584253	
14	-0.368384	-0.685492	-0.157262	...	1.443107	-1.110129	0.826405	-2.244372	
15	0.350754	0.278748	-1.276145	...	0.149569	0.478426	1.508804	-1.195854	
16	0.380779	0.358925	0.912254	...	2.068085	-1.821035	-0.094960	0.799891	
17	0.235605	0.399429	0.080173	...	-0.120344	-0.519655	-0.211918	0.835111	
18	1.795879	2.445252	-1.447472	...	-0.665683	-1.198303	-1.823444	-0.118264	
19	-0.089649	0.490101	-0.640121	...	-1.447591	-0.274033	-1.005712	-0.877800	
20	-0.169944	-0.615112	-0.065463	...	0.592506	1.262068	0.660922	0.312333	
21	0.774924	-1.355360	-0.344326	...	-0.904621	-1.628341	0.223989	0.194612	
22	1.058961	0.541948	-1.978839	...	0.681967	-0.232020	-0.220840	-0.103127	

```
23 -1.323876 -0.080042 -0.657279 ... -1.094393 -0.550951 -1.234929 -0.037521
24 -0.677066 -1.466584 -1.890322 ...  0.120183 -0.352210  0.225573  1.031241
25  0.153567 -0.004581 -0.138550 ... -0.559200  2.095961  0.234779 -0.707788
26 -0.549529 -0.691815 -0.140374 ... -1.674702  1.985760  1.649605  0.145370
27  1.037246  1.037619 -1.288361 ... -0.589299 -0.980078  0.987582  1.122147
28 -0.233353  1.467475 -0.371649 ...  0.728324 -0.048471 -0.302596 -0.313801
29  2.007782 -0.124257 -0.364859 ...  0.096220 -0.402159 -0.352479  1.012389
..       ...       ...       ... ...       ...       ...       ...       ...
70  0.399714 -1.814921  0.596181 ... -1.049811 -0.305725 -0.196439 -0.996503
71  1.248657  0.857511  0.123373 ... -0.401187 -0.296976  1.267618  1.407632
72  0.030046  0.418403 -0.844470 ... -1.017495 -0.310232  1.185204 -0.094041
73 -0.563457  0.645620 -0.072238 ... -0.462390  0.859341 -0.673302 -0.775462
74  1.998429  1.330004  0.904255 ... -0.439623 -2.511650 -0.864090 -0.000300
75  0.090197  0.419132 -0.411887 ... -0.280226  0.341549  0.750836 -0.481730
76 -0.003909 -0.608779  1.380819 ... -0.479099 -0.477545  0.064289  0.188386
77 -0.260208 -1.228691  0.293633 ...  1.054654  0.278712  0.835792 -0.669672
78 -1.970891  1.859660 -1.336751 ... -1.023968  1.747803 -1.398047  3.054109
79 -0.371840 -1.454054  0.689402 ...  2.159556  2.828567  0.787448  3.357980
80  1.336730 -0.173222  0.528489 ...  0.705145 -0.236390  0.794127  0.034032
81 -1.071889 -1.327804  1.455120 ... -0.706613  0.037127  0.821836 -0.208053
82  0.494363 -1.357284 -0.881724 ... -0.918066  1.006673  0.152670  0.794414
83 -0.345121 -0.606346 -0.250060 ...  0.345311 -0.297554 -1.612845 -0.920126
84  0.206828  0.726186 -0.634634 ...  0.882847 -2.036136 -1.174613 -0.735347
85 -0.365361  2.106710 -1.153328 ... -1.292370  0.517513 -1.235261  1.036895
86  0.784701  0.671629 -0.010766 ... -1.747557  1.974905  1.114480 -0.933930
87  1.242480  0.927010 -0.124062 ...  0.495329 -1.039336  1.546429 -0.083968
88 -1.220621 -0.590916  0.867684 ... -0.095283  0.137494 -0.032359  2.499759
89 -0.694359 -2.552955 -0.620113 ... -0.930475  0.548321  0.935686 -0.054190
90  1.364984 -0.152651 -0.085216 ...  2.170426  0.354861 -0.213907 -0.800140
91 -0.381461 -0.496112  1.450530 ...  0.157826  0.965310  1.317044 -1.544242
92 -0.432571 -1.160908 -0.975361 ... -0.034817 -0.033289  0.974470 -1.444899
93  0.887200 -0.479244  1.392512 ... -0.501300  2.396208  2.851629  0.377902
94  0.304066 -0.140139  1.821381 ... -0.753021  2.632727  0.270002  0.624940
95  1.562867  0.336065  0.486314 ... -0.678658 -0.553460  1.761389  1.639796
96 -0.584791  1.329236 -1.222141 ...  0.858936 -0.450724 -0.976011  0.200169
97  0.154183  0.960068  0.830254 ...  0.619807 -1.078051  0.235374  0.519606
98 -0.373598 -0.864674 -0.380980 ...  0.202298  1.552599  1.054328 -0.123826
99 -0.063116 -0.582223 -0.601134 ...  0.139241 -1.293498 -0.117857  0.075175

         X16       X17       X18       X19       X20  Y
0   0.055139 -0.047043 -0.288166  0.071933 -1.984860 -1
1  -0.207358  0.187353 -3.294822 -0.412425 -1.091548 -1
2   1.077672 -0.449392  0.102625  0.061146  1.051049  1
3   0.548626 -0.294320 -0.114500  0.116142  2.240178  1
4   0.320545 -0.343446 -0.594816  1.521063 -2.657491 -1
5  -0.025377 -0.510531 -2.290886  0.319804 -2.383328 -1
6  -0.148917  0.700197 -0.849276  2.104047 -3.119065 -1
7   1.982642 -0.418662  1.267801 -1.641752 -1.316907 -1
```

8	-0.412671	0.074028	-1.711044	2.100226	1.006713	1
9	0.560043	0.437294	-0.678661	-1.040659	2.434224	1
10	-0.253823	-1.320237	0.700549	-0.308352	1.423215	1
11	-1.505190	-1.183918	0.353564	1.446184	-1.220614	-1
12	0.633386	0.821052	0.823824	0.818807	2.896950	1
13	-0.606534	-0.571134	-0.643407	-1.985450	-1.811177	-1
14	-0.080778	0.334225	-0.038178	0.160714	4.967474	1
15	1.010167	-0.447635	0.986069	2.047337	1.044141	1
16	1.042621	0.568763	1.043503	-0.955457	-4.687003	-1
17	0.489210	0.217868	-1.625947	0.246208	2.903925	1
18	-1.483798	0.029979	0.449189	0.431714	1.367687	1
19	-0.802404	1.031833	0.888534	-0.824319	1.757657	1
20	1.190499	-0.503493	-0.538494	0.111919	1.298365	1
21	-1.464304	0.670811	-0.829861	-1.101805	-4.976539	-1
22	1.215006	-0.174391	-2.508358	0.091237	5.317371	1
23	-0.249310	-0.325728	2.105976	1.034660	3.457764	1
24	0.712562	1.216753	0.495720	-0.480912	-2.581034	-1
25	-1.563459	-0.731774	0.382575	0.776534	2.537464	1
26	-1.057274	-0.219270	1.484963	-1.559365	-1.404652	-1
27	-0.802862	2.238939	-0.834604	0.061543	1.710767	1
28	-0.803636	-0.037722	0.658589	-0.975986	1.373730	1
29	0.657998	-0.293870	1.934462	-0.504609	-2.052451	-1
..	...	...	...	...	...	..
70	0.605669	0.731935	0.138165	-1.051564	1.639227	1
71	0.857659	-0.419239	0.442897	-0.040977	1.296584	1
72	0.429328	-0.369674	-0.737510	0.144583	1.906660	1
73	1.212283	0.636853	0.615222	-0.836833	1.067185	1
74	0.268267	0.516379	0.119914	1.429355	1.453480	1
75	-1.105067	-0.450184	-0.381562	-1.730800	-2.298988	-1
76	-0.117863	1.721397	-0.601319	0.062108	2.395851	1
77	-0.886188	-1.416172	-0.655284	-0.818575	2.261014	1
78	-0.981708	0.486741	2.054171	0.478623	4.436638	1
79	0.029352	0.990176	0.552975	1.846770	-1.148492	-1
80	0.272622	1.531233	-0.258871	0.291405	1.857374	1
81	-1.129532	0.735854	0.531243	-0.068959	-2.078339	-1
82	0.541198	-0.249286	-0.103320	-0.284817	-2.274288	-1
83	1.159215	0.018376	-0.957934	1.421915	3.827221	1
84	-0.834657	-0.672974	0.999040	-0.143372	1.900989	1
85	-1.586371	-0.163510	-0.218254	-0.138241	-1.450802	-1
86	2.519440	0.806379	-0.079915	-2.135720	-1.986785	-1
87	-0.086441	-1.573187	0.438781	2.397325	-1.143259	-1
88	-1.066084	0.283830	-0.743245	-0.564458	1.318809	1
89	0.815712	0.545581	1.446164	-0.019871	-1.337086	-1
90	0.671328	-0.278517	0.126612	0.201741	-1.513812	-1
91	0.416140	0.674789	-1.712449	0.440772	1.366612	1
92	0.741001	-1.264833	-0.900301	0.262629	1.467125	1
93	1.720848	1.652874	0.430679	0.198464	-1.598991	-1
94	0.117179	-0.183047	0.544243	-0.076814	1.944161	1

```
95 -0.312238  0.025654  0.543542  0.223127  2.233729  1
96  0.985217 -0.670479  0.837273  0.141904 -4.166485 -1
97  0.677437  0.251633 -0.423527  3.015709 -1.113356 -1
98 -0.390625 -0.809361 -0.550079  0.965275 -1.931530 -1
99  0.613467  0.344205 -0.495027 -1.926405 -2.069869 -1

[100 rows x 21 columns]
```

In [98]:
```python
length = len(df)

for row_index in range(0, length):
    row = print(dataframe[row_index: row_index+1])
```

```
         X1        X2        X3        X4        X5        X6       X7  \
0  1.024317  1.606267 -0.469453 -0.307574 -0.753898 -0.332596 -0.58312

         X8        X9       X10  Y
0 -1.174982 -0.715254  0.447657  1
         X1        X2        X3        X4        X5        X6        X7  \
1 -0.573116  0.961008  0.501592 -0.363208 -0.735952  2.846852  0.990659

         X8        X9       X10  Y
1  0.718813 -0.676832 -1.289605 -1
         X1        X2        X3        X4        X5        X6        X7  \
2 -2.109406  0.180483  1.409474 -1.195209 -0.082716  1.616118  0.604639

         X8        X9       X10  Y
2 -0.110708 -0.001893  1.010693  1
         X1        X2        X3        X4        X5       X6        X7  \
3 -1.015835  0.171674  0.868645 -0.633455 -1.034178 -0.48988  1.217603

         X8       X9       X10  Y
3  0.783461 -0.49431  1.652137  1
         X1        X2        X3       X4        X5        X6        X7  \
4  1.197106 -1.597415  1.724866 -1.35963 -0.388301 -1.639981  1.456671

         X8        X9      X10  Y
4  0.439257 -0.033563 -1.01174 -1
```

In [76]:
```python
df.shape[0]
```

Out[76]: 5

2 Function to Train a Perceptron on given Data set

In [144]:
```python
#Create perceptron, returns weight vector and bias
def perceptron_fit_function(dataframe, steps=10):
```

```
features = list(dataframe)[:-1]
size = dataframe.shape[0]
feature_size = dataframe.shape[1] - 1
flag = 0
count_step = 0
steps = steps
#print('step size: ',steps)

w = np.zeros((1, feature_size))
b = np.zeros((1,1))

while count_step < steps and flag < 3:

    classified = 0
    misclassified = 0

    for i in range(0, size):

        df = dataframe[i: i+1]
        xi = df[features].values
        fx = np.dot(w, xi.T) + b
        fx = fx[0][0]
        Y = df['Y'].values

        #Checking if the point fxi is misclassified
        if (fx > 0 and Y == 1) or (fx < 0 and Y == -1):
            classified += 1
        else:
            #Updating the weights and bias
            w = w + (Y * xi)
            b = b + Y
            misclassified += 1

    if misclassified == 0:
        #If all data is classified 3 times then exit, ie early stopping if solut
        flag += 1
        #print('flag')

    count_step += 1
    if count_step == steps:
        print('within count steps')
        if misclassified == 0:
            print('Linear Seperator exists')
        else:
            print('Linear Seperator does not exists')
            w = None
            b = None
```

```
                flag = 10

        if count_step != steps:
            count_step -= flag

        return w, b, count_step
```

In [150]:
```
dataframe = dataframe_function(k=10, m=50, epsilon=0.1)
model = perceptron_fit_function(dataframe)
print('w: ',model[0])
print('b: ',model[1])
print('step count: ',model[2])
```

```
w:  [[-0.74660775  0.06873336 -1.82815349 -1.13711481  0.95665413  0.18813585
   2.12113467  1.28534675 -0.5025711   5.11448027]]
b:  [[0.]]
step count:  1
```

3 Question 1

Show that there is a perceptron that correctly classifies this data. Is this perceptron unique? What is the 'best' perceptron for this data set, theoretically?

4 Answer

We created a data set using k = 10, m = 50 and epsilon = 0.1. We generated a perceptron using the perceptron_fit_function. What we get in return is a unique perceptron for every data set. There is a missclassified tag within the perceptron_fit_function which keeps a track of correctly classifying the data set, using this we can see that final resulting perceptron has 0 misclassifed value. Hence we can conclude that there is a perceptron which correctly classifies the data and provides a unique result.

What is the best perceptron for this data set ? The data set's classification value ie Y depends solely on value of Xk (in our case it is on normal and exponential distribution). It does not depend on any other dimensions, hence we can say if we can find a seperator between Xk values we can classify the data set. Xk is taken in such a way that probability of it containing +ve and -ve values are equal ie 1/2 - 1/2. Thus we get an even distribution of Xk values around 0, we can conclude the best perceptron for this kind of data set is a line which passes throught the orign.

5 Question 2

We want to consider the problem of learning perceptrons from data sets. Generate a set of data of size m = 100 with k = 20, epsilon = 1.

- Implement the perceptron learning algorithm. This data is separable, so the algorithm will terminate. How does the output perceptron compare to your theoretical answer in the previous problem?

Generating data set with specification m = 100, k = 20 & epsilon = 1

```
In [122]: dataframe = dataframe_function(k=20, m=100, epsilon=1)

          #Display a sample of the dataset.
          dataframe.head()
```

```
Out[122]:          X1        X2        X3        X4        X5        X6        X7  \
          0  0.340212  0.485803 -1.181850  1.400515  1.657566  1.208631 -1.070483
          1 -1.597158 -1.967528  1.502941 -0.130621 -0.687851  0.639631  0.759493
          2  1.086853 -0.822808  0.091870 -0.577329 -0.771169 -1.471926 -1.550960
          3 -0.544034 -1.101588  1.552439  0.638534  0.479416  0.647569  0.996788
          4  0.271434 -2.071821  2.226086  0.652093  1.401903 -1.060997 -1.448512

                   X8        X9       X10  ...       X12       X13       X14       X15  \
          0  1.303207  0.112928  0.908023  ...  0.443499  0.149570 -2.444234 -0.776719
          1 -0.046555  1.337165 -1.955479  ...  0.917504 -0.084746  0.139968 -0.074476
          2 -1.483968  0.425939 -0.617225  ...  0.313743 -1.675454  1.959581 -1.493245
          3 -0.172508 -0.588969  0.927808  ...  1.779191  0.009583 -0.951522 -0.975144
          4  0.424875 -0.808284 -0.114838  ...  0.683795  1.405966  0.070741 -0.074328

                  X16       X17       X18       X19       X20  Y
          0  2.385420  0.656723 -0.211792 -0.790755 -2.345192 -1
          1 -0.034353 -0.777148  1.344744  0.172613  1.880840  1
          2  0.004931 -0.224782 -0.155077  0.136446  1.630811  1
          3  0.016425 -0.197770  0.966589  1.688775  2.335050  1
          4  0.343087  2.116701  0.464243  0.832187 -1.542629 -1

          [5 rows x 21 columns]
```

Fit Perceptron Model on the Data Set

```
In [146]: model = perceptron_fit_function(dataframe)
          print('w: ',model[0])
          print('b: ',model[1])
          print('step count: ',model[2])
```

```
w:  [[ 1.62337312  0.07112198 -1.41545305  1.64707975  0.52439475 -0.99106275
   1.19796553  1.33156467  0.931362   -0.74197748  2.92093541 -1.47774099
   0.55843342  2.40515051 -0.90398341 -0.04597894 -1.14035018  1.46474577
  -1.54823483  9.39215437]]
b:  [[0.]]
step count:  2
```

6 Conclusion

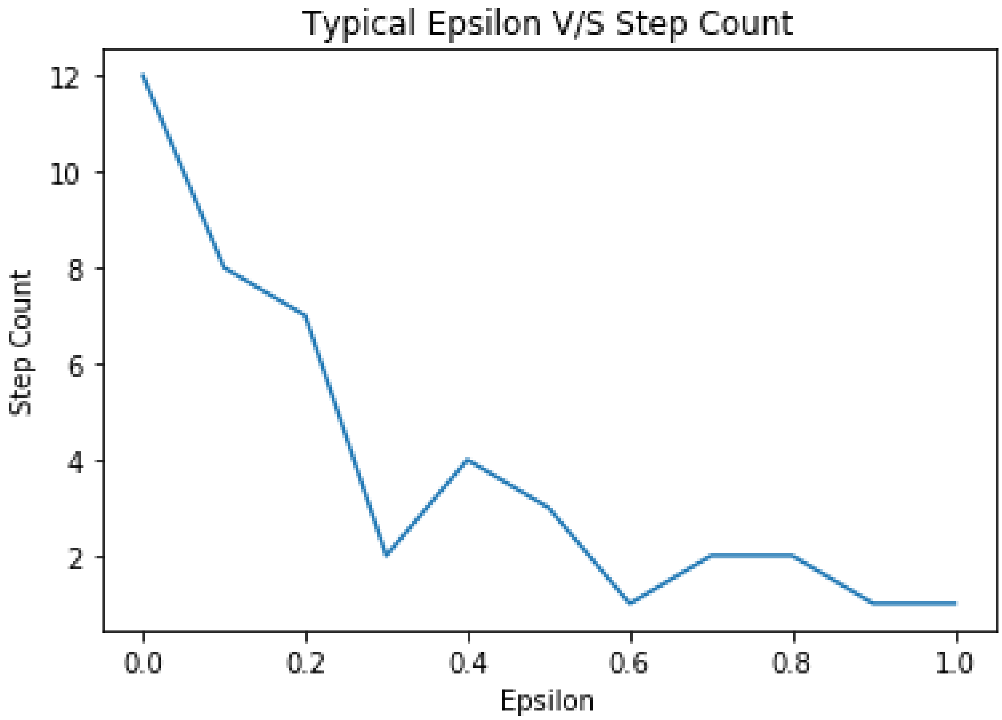
We observed that b = 0 which indicates that our perceptron's seperating line has intercept of 0, which tells us that the seperating line passes through the origin for this data set. Similar to what we have concluded in question 1 that theoretically our line seperator passes through the origin.

7 Question 3

For any given data set, there may be multiple separators with multiple margins - but for our data set, we can effectively control the size of the margin with the parameter epsilon - the bigger this value, the bigger the margin of our separator.

- For m = 100, k = 20, generate a data set for a given value of epsilon and run the learning algorithm to completion. Plot, as a function of epsilon within [0, 1], the average or typical number of steps the algorithm needs to terminate. Characterize the dependence.

```
In [161]: m = 100
          k = 20
          step_threshold = 100
          epsilon_list = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
          eclist = {}

          for epsilon in epsilon_list:
              eclist[epsilon] = perceptron_fit_function(dataframe_function(k=k, m=m, epsilon=ep

          plot_list = sorted(eclist.items())
          x,y = zip(*plot_list)

          plt.plot(x,y)
          plt.title('Typical Epsilon V/S Step Count')
          plt.xlabel('Epsilon')
          plt.ylabel('Step Count')
          plt.show()
```



8 Conclusion

I have classified the data set using perceptrons with varying value of epsilon between 0 & 1. We plot the step count for every epsilon value. Observe that for values of epsilon below 0.1 needs a lot of computation steps to fit a correct perceptron, but the step count decreases linearly as we go from 0 to 0.2. Moving from epsilon 0.2 to 1 we see that the step count decreases very slowly. Where we observe that for epsilon = 1 it only takes 1 step to compute the correct perceptron for this data set.

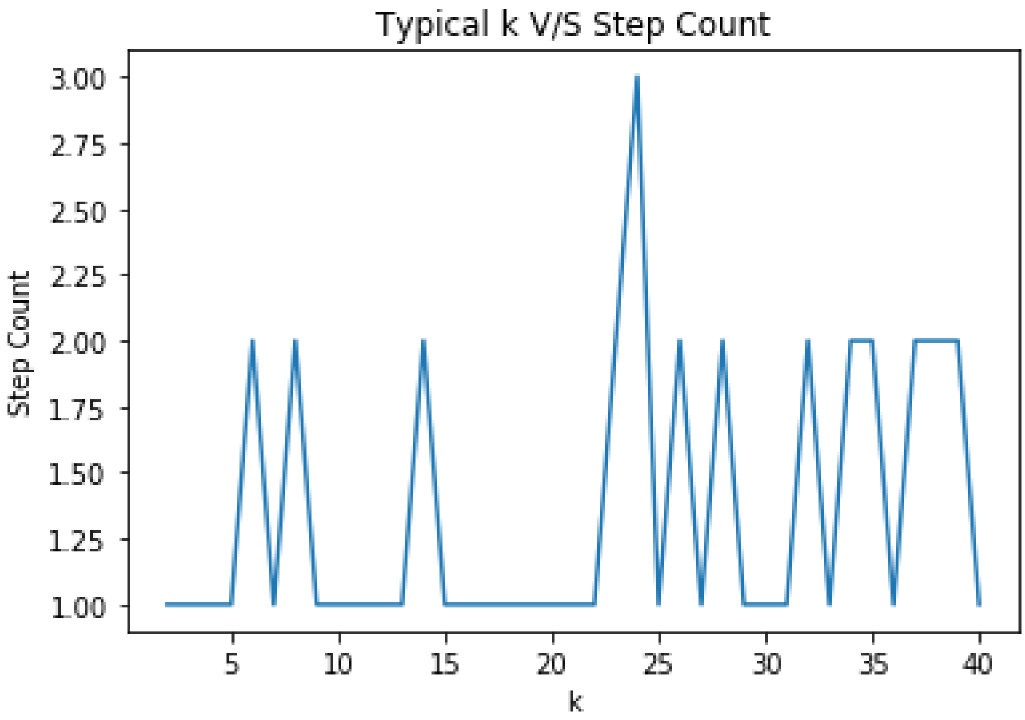
9 Question 4

One of the nice properties of the perceptron learning algorithm (and perceptrons generally) is that learning the weight vector w and bias value b is typically independent of the ambient dimension. To see this, consider the following experiment:

- Fixing m = 100; epsilon = 1, consider generating a data set on k features and running the learning algorithm on it. Plot, as a function k (for k = 2, ..., 40), the typical number of steps to learn a perceptron on a data set of this size. How does the number of steps vary with k? Repeat for m = 1000.

Computing for m = 100

In [162]:
```
m = 100
step_threshold = 100
epsilon = 1
kclist = {}

for k in range(2, 41):
    kclist[k] = perceptron_fit_function(dataframe_function(k=k, m=m, epsilon=epsilon

plot_list = sorted(kclist.items())
x,y = zip(*plot_list)

plt.plot(x,y)
plt.title('Typical k V/S Step Count')
plt.xlabel('k')
plt.ylabel('Step Count')
plt.show()
```

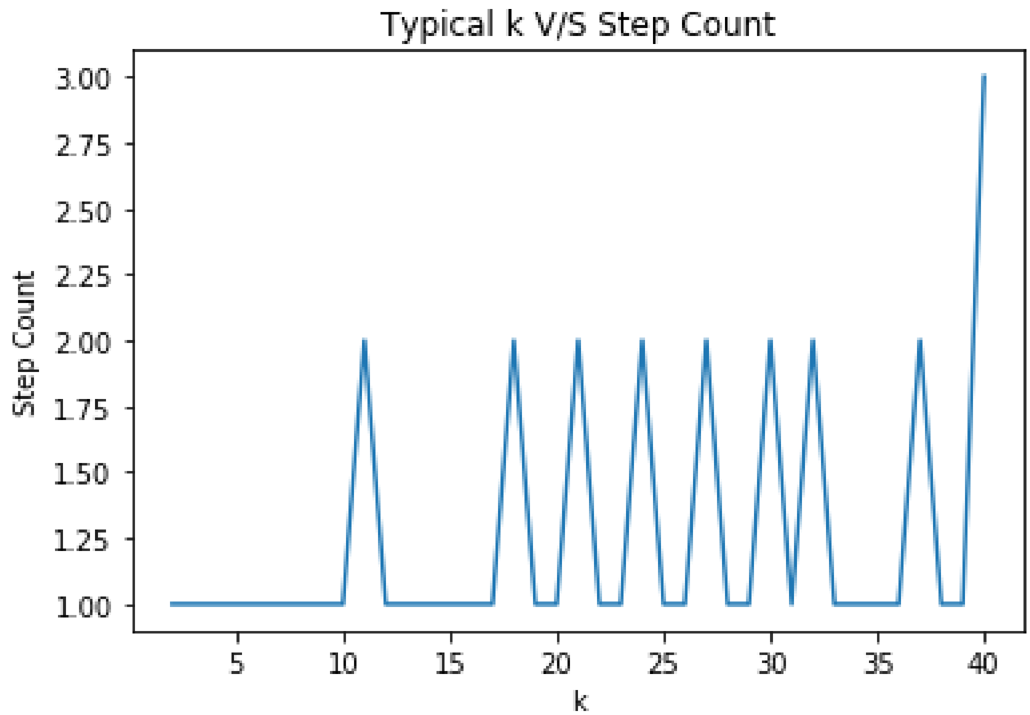


Conclusion

We can observe that, when we vary the value of k ranging from 2 to 40 we see a random step counts between 1 to 3. Which leads us to conclude that the number of steps taken to correctly classify a perceptron does not depend on value of k. Also we have observed that from the definition of our data set we can see that the value of Y depends solely on value of Xk, we can say that value of k is not correlated with the number of step count.

Computing for m = 1000

```
In [163]: m = 1000
          step_threshold = 100
          epsilon = 1
          kclist = {}

          for k in range(2, 41):
              kclist[k] = perceptron_fit_function(dataframe_function(k=k, m=m, epsilon=epsilon)

          plot_list = sorted(kclist.items())
          x,y = zip(*plot_list)

          plt.plot(x,y)
          plt.title('Typical k V/S Step Count')
          plt.xlabel('k')
          plt.ylabel('Step Count')
          plt.show()
```



Conclusion
By Changing the value of m, the value of step count is not affected for different value of k.

10 Question 5

As shown in class, the perceptron learning algorithm always terminates in finite time - if there is a separator. Consider generating non-separable data in the following way: generate each X1, ... , Xk as i.i.d. standard normals N(0; 1). Define Y by:

```
Y = +1 if summationOf(Xi^2) >= k ;
   else -1
```

For data defined in this way, there is no universally applicable linear separator.

For k = 2, m = 100, generate a data set that is not linearly separable. (How can you verify this?) Then run the perceptron learning algorithm. What does the progression of weight vectors and bias values look like over time? If there is no separator, this will never terminate - is there any condition or heuristic you could use to determine whether or not to terminate the algorithm and declare no separator found?

Solution: Modifying generate_data set function to implement the changes mentioned above

In [172]:
```
#Function to generate dataset
def modified_dataframe_function(k=2, m=100, epsilon=1):

    data = []
```

```
    def vector_generation_function(k=k, epsilon=epsilon):
        X = []

        #Assign standard normal values to X1 .. Xk-1.
        for i in range(1, k+1):
            X.append(np.random.standard_normal())

        if sum([i ** 2 for i in X]) >= k: X.append(1)
        else: X.append(-1)

        return X

    for x in range(1, m+1):
        data.append(vector_generation_function(k=k, epsilon=epsilon))

    #Create header list
    headers = ['X'+str(x) for x in range(1, k+1)] + ['Y']

    dataframe = pd.DataFrame(data, columns=headers)

    return dataframe
```

Generate modified dataset

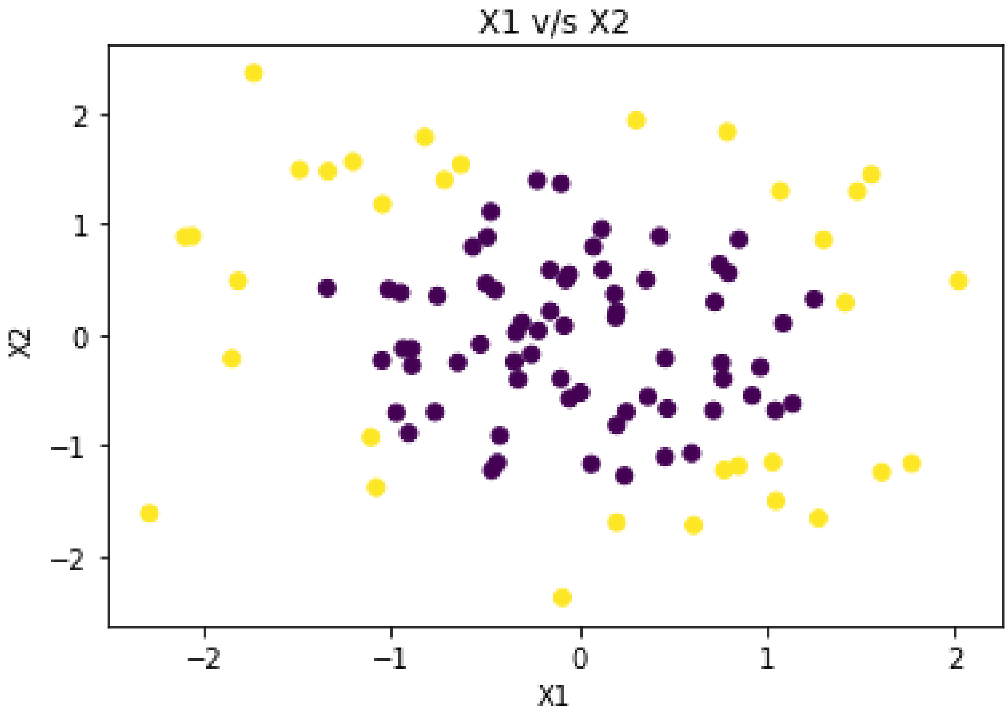
```
In [173]: mod_dataset = modified_dataframe_function(k=2, m=100, epsilon=1)
          mod_dataset.head()
```

```
Out[173]:          X1        X2  Y
          0 -0.974027 -0.702063 -1
          1  1.033643 -1.146976  1
          2  1.089418  0.106023 -1
          3  0.201295 -0.813970 -1
          4 -0.096179  1.363796 -1
```

Plot X1 vs X2 and Y as color coded

```
In [185]: X1 = mod_dataset['X1'].values
          X2 = mod_dataset['X2'].values
          Y = mod_dataset['Y'].values

          plt.scatter(X1, X2, c=Y)
          plt.title('X1 v/s X2')
          plt.xlabel('X1')
          plt.ylabel('X2')
          plt.show
```

```
Out[185]: <function matplotlib.pyplot.show(*args, **kw)>
```



How can you verify the data set is not linearly seperable ?

We plot the X1 v/s X2 data set and color code the Y values for every X1, X2 pair; we get the above plot. What we can infer by looking at the above plot is that for value of Y = -1 we have a different color and for Y = 1 the point has different color; we can say that there exit no single line which can seperate values of Y = -1 and Y = 1. Hence the dataset is not linearly seperable

Modify the perceptron function to print vectors of w and b on every time step

```
In [186]: #Create perceptron, returns weight vector and bias
          def modified_perceptron_fit_function(dataframe, steps=10):

              features = list(dataframe)[:-1]
              size = dataframe.shape[0]
              feature_size = dataframe.shape[1] - 1
              flag = 0
              count_step = 0
              steps = steps
              wstep = {}
              bstep = {}
              #print('step size: ',steps)

              w = np.zeros((1, feature_size))
              b = np.zeros((1,1))

              print('Step: ',count_step,' ie Initial vector.')
              print('w: ',w)
```

```
print('b: ',b)
print('\n')

wstep[count_step] = w
bstep[count_step] = b

while count_step < steps and flag < 3:

    classified = 0
    misclassified = 0

    for i in range(0, size):

        df = dataframe[i: i+1]
        xi = df[features].values
        fx = np.dot(w, xi.T) + b
        fx = fx[0][0]
        Y = df['Y'].values

        #Checking if the point fxi is misclassified
        if (fx > 0 and Y == 1) or (fx < 0 and Y == -1):
            classified += 1
        else:
            #Updating the weights and bias
            w = w + (Y * xi)
            b = b + Y
            misclassified += 1

    print('Step: ',count_step+1)
    print('w: ',w)
    print('b: ',b)
    print('\n')

    wstep[count_step+1] = w
    bstep[count_step+1] = b

    if misclassified == 0:
        #If all data is classified 3 times then exit, ie early stopping if solut
        flag += 1
        #print('flag')

    count_step += 1
    if count_step == steps:
        print('within count steps')
        if misclassified == 0:
            print('Linear Seperator exists')
        else:
            print('Linear Seperator does not exists')
```

```python
                w = None
                b = None
            flag = 10

    if count_step != steps:
        count_step -= flag

    return w, b, count_step, wstep, bstep
```

Run mod fit perceptron using above mod data set

```python
In [187]: model = modified_perceptron_fit_function(mod_dataset, steps=100)
```

```
Step:  0  ie Initial vector.
w:  [[0. 0.]]
b:  [[0.]]


Step:  1
w:  [[-1.60079193 -2.09692983]]
b:  [[-1.]]


Step:  2
w:  [[-1.59873226 -1.91094554]]
b:  [[-1.]]


Step:  3
w:  [[ 0.09940789 -3.26048422]]
b:  [[-1.]]


Step:  4
w:  [[ 0.10460107 -3.24355867]]
b:  [[-1.]]


Step:  5
w:  [[ 0.10979425 -3.22663313]]
b:  [[-1.]]


Step:  6
w:  [[ 0.11498742 -3.20970758]]
b:  [[-1.]]
```

```
Step:  7
w:  [[ 0.43334519 -0.84627994]]
b:  [[0.]]


Step:  8
w:  [[-0.29883711 -2.75090507]]
b:  [[0.]]


Step:  9
w:  [[-0.17768112 -2.5591303 ]]
b:  [[0.]]


Step:  10
w:  [[-0.18786287 -2.14586417]]
b:  [[0.]]


Step:  11
w:  [[ 0.25545252 -1.41233671]]
b:  [[-1.]]


Step:  12
w:  [[-0.26047987 -2.37975641]]
b:  [[0.]]


Step:  13
w:  [[-1.72675973 -1.91798363]]
b:  [[-1.]]


Step:  14
w:  [[ 0.35507548 -1.89568201]]
b:  [[-1.]]


Step:  15
w:  [[ 0.08659326 -3.18451462]]
b:  [[-1.]]


Step:  16
w:  [[ 0.40495103 -0.82108698]]
b:  [[0.]]
```

```
Step:  17
w:  [[-1.08042527 -2.7184865 ]]
b:  [[-1.]]


Step:  18
w:  [[ 0.16784788 -1.42565924]]
b:  [[-1.]]


Step:  19
w:  [[-0.40179671 -2.09312266]]
b:  [[-1.]]


Step:  20
w:  [[-0.96865813 -2.52538382]]
b:  [[-1.]]


Step:  21
w:  [[-0.94676074 -2.53726376]]
b:  [[-1.]]


Step:  22
w:  [[ 0.92024799 -1.39946594]]
b:  [[-1.]]


Step:  23
w:  [[ 0.19584903 -2.14077727]]
b:  [[0.]]


Step:  24
w:  [[ 0.02722717 -2.04366315]]
b:  [[-1.]]


Step:  25
w:  [[-0.04613775 -2.00942301]]
b:  [[0.]]


Step:  26
```

```
w:  [[-0.61304193 -2.27627052]]
b:  [[-1.]]


Step:  27
w:  [[-1.44333016 -1.75728035]]
b:  [[-1.]]


Step:  28
w:  [[-0.7062768  -2.29912032]]
b:  [[-1.]]


Step:  29
w:  [[-1.53656503 -1.78013015]]
b:  [[-1.]]


Step:  30
w:  [[-0.79951168 -2.32197013]]
b:  [[-1.]]


Step:  31
w:  [[ 0.06628058 -3.33850292]]
b:  [[-1.]]


Step:  32
w:  [[-0.01464533 -2.22948708]]
b:  [[-1.]]


Step:  33
w:  [[-0.70179878 -2.01898983]]
b:  [[-1.]]


Step:  34
w:  [[ 0.65607264 -2.50410936]]
b:  [[-1.]]


Step:  35
w:  [[-0.32888071 -1.72581104]]
b:  [[-1.]]
```

```
Step:  36
w:  [[-0.8927944  -2.78384921]]
b:  [[-1.]]


Step:  37
w:  [[-0.4936238  -2.52308818]]
b:  [[-1.]]


Step:  38
w:  [[ 0.19087485 -2.31280696]]
b:  [[0.]]


Step:  39
w:  [[-0.66610154 -3.27162182]]
b:  [[-1.]]


Step:  40
w:  [[-1.54170297 -1.76396654]]
b:  [[-1.]]


Step:  41
w:  [[-1.58818905 -2.13490822]]
b:  [[-1.]]


Step:  42
w:  [[-1.58612938 -1.94892393]]
b:  [[-1.]]


Step:  43
w:  [[ 0.11201078 -3.29846261]]
b:  [[-1.]]


Step:  44
w:  [[ 0.11720395 -3.28153706]]
b:  [[-1.]]


Step:  45
w:  [[ 0.12239713 -3.26461152]]
```

```
b:  [[-1.]]


Step:  46
w:  [[ 0.12759031 -3.24768597]]
b:  [[-1.]]


Step:  47
w:  [[ 0.13278349 -3.23076043]]
b:  [[-1.]]


Step:  48
w:  [[ 0.13797667 -3.21383488]]
b:  [[-1.]]


Step:  49
w:  [[ 0.45633444 -0.85040724]]
b:  [[0.]]


Step:  50
w:  [[-0.01016004 -1.99373253]]
b:  [[0.]]


Step:  51
w:  [[ 0.19992342 -2.12289546]]
b:  [[-1.]]


Step:  52
w:  [[-0.19372208 -2.28652169]]
b:  [[-1.]]


Step:  53
w:  [[ 0.51875776 -0.77553417]]
b:  [[0.]]


Step:  54
w:  [[ 0.05226328 -1.91885946]]
b:  [[0.]]
```

```
Step:  55
w:  [[ 0.50910016 -1.37549405]]
b:  [[0.]]


Step:  56
w:  [[ 0.13366109 -3.14148468]]
b:  [[-1.]]


Step:  57
w:  [[ 0.45201886 -0.77805705]]
b:  [[0.]]


Step:  58
w:  [[-0.01447562 -1.92138234]]
b:  [[0.]]


Step:  59
w:  [[-0.56212948 -2.02746741]]
b:  [[-1.]]


Step:  60
w:  [[-1.49350565 -1.7728113 ]]
b:  [[-1.]]


Step:  61
w:  [[-0.7564523  -2.31465128]]
b:  [[-1.]]


Step:  62
w:  [[ 0.10933996 -3.33118407]]
b:  [[-1.]]


Step:  63
w:  [[-0.64986163 -1.8092015 ]]
b:  [[-1.]]


Step:  64
w:  [[ 0.40616249 -1.39189615]]
b:  [[-1.]]
```

```
Step:  65
w:  [[-0.24110764 -2.13782449]]
b:  [[0.]]


Step:  66
w:  [[ 0.20220774 -1.40429703]]
b:  [[-1.]]


Step:  67
w:  [[-0.26941063 -3.04269826]]
b:  [[0.]]


Step:  68
w:  [[-0.29498297 -1.78097294]]
b:  [[-1.]]


Step:  69
w:  [[-0.93670171 -2.72990569]]
b:  [[-1.]]


Step:  70
w:  [[-0.55594721 -2.04773455]]
b:  [[-1.]]


Step:  71
w:  [[-1.48732338 -1.79307844]]
b:  [[-1.]]


Step:  72
w:  [[-0.75027003 -2.33491842]]
b:  [[-1.]]


Step:  73
w:  [[-1.58055826 -1.81592825]]
b:  [[-1.]]


Step:  74
```

```
w:  [[-0.8435049  -2.35776822]]
b:  [[-1.]]


Step:  75
w:  [[ 0.36465372 -0.81905435]]
b:  [[0.]]


Step:  76
w:  [[ 0.16274323 -2.21307247]]
b:  [[0.]]


Step:  77
w:  [[-1.84366999 -2.06946943]]
b:  [[-1.]]


Step:  78
w:  [[-0.33926395 -2.76317125]]
b:  [[0.]]


Step:  79
w:  [[-0.21810796 -2.57139648]]
b:  [[0.]]


Step:  80
w:  [[-0.2282897  -2.15813035]]
b:  [[0.]]


Step:  81
w:  [[ 0.21502568 -1.42460289]]
b:  [[-1.]]


Step:  82
w:  [[-0.25659269 -3.06300413]]
b:  [[0.]]


Step:  83
w:  [[-0.9194066  -2.40775765]]
b:  [[-2.]]
```

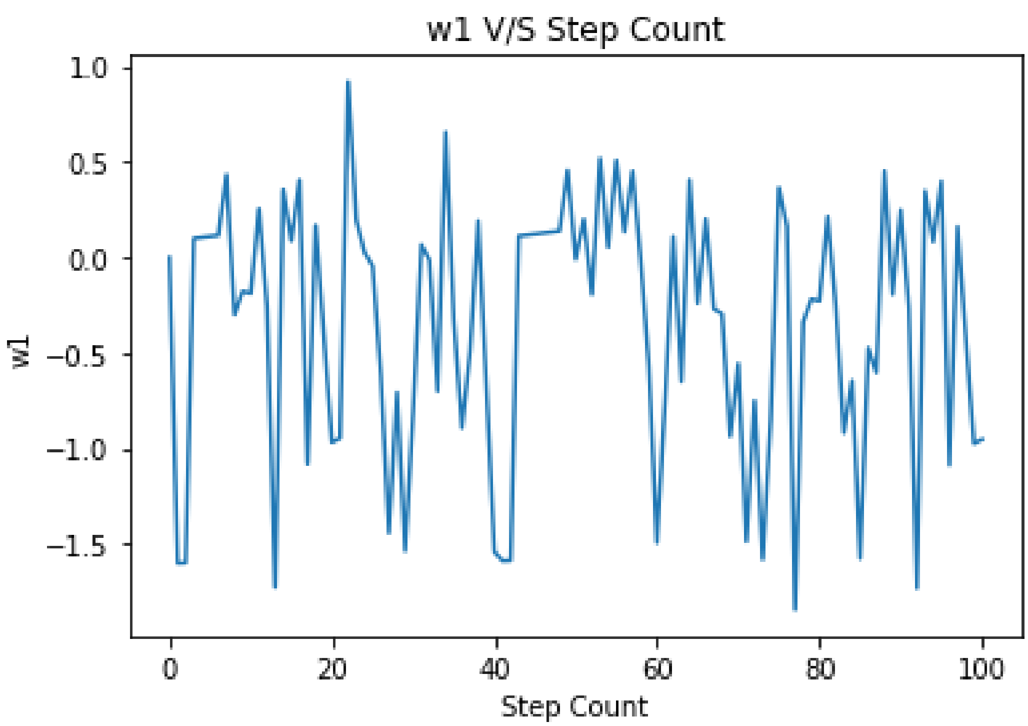
```
Step:  84
w:  [[-0.6449651  -2.03275113]]
b:  [[-1.]]


Step:  85
w:  [[-1.57634127 -1.77809502]]
b:  [[-1.]]


Step:  86
w:  [[-0.47339052 -3.25145461]]
b:  [[-1.]]


Step:  87
w:  [[-0.60266885 -1.81967774]]
b:  [[-1.]]


Step:  88
w:  [[ 0.45335526 -1.40237238]]
b:  [[-1.]]


Step:  89
w:  [[-0.19391487 -2.14830072]]
b:  [[0.]]


Step:  90
w:  [[ 0.24940052 -1.41477326]]
b:  [[-1.]]


Step:  91
w:  [[-0.26653187 -2.38219296]]
b:  [[0.]]


Step:  92
w:  [[-1.73281173 -1.92042018]]
b:  [[-1.]]


Step:  93
w:  [[ 0.34902348 -1.89811856]]
```

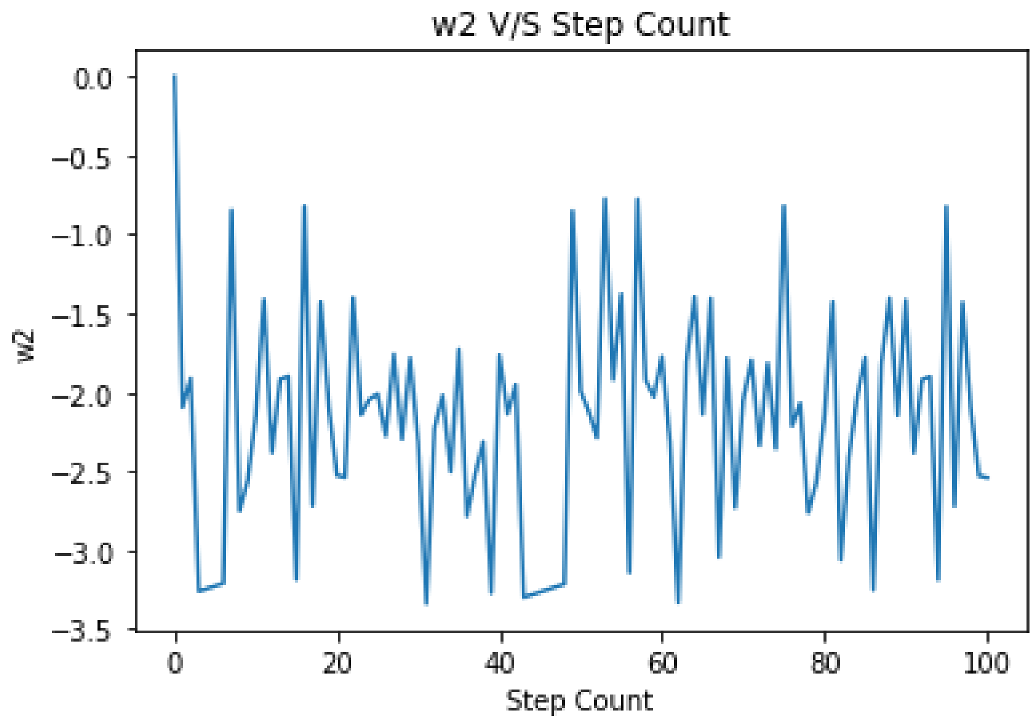
```
b:  [[-1.]]


Step:  94
w:  [[ 0.08054126 -3.18695117]]
b:  [[-1.]]


Step:  95
w:  [[ 0.39889903 -0.82352353]]
b:  [[0.]]


Step:  96
w:  [[-1.08647727 -2.72092305]]
b:  [[-1.]]


Step:  97
w:  [[ 0.16179587 -1.42809579]]
b:  [[-1.]]


Step:  98
w:  [[-0.40784871 -2.09555921]]
b:  [[-1.]]


Step:  99
w:  [[-0.97471014 -2.52782037]]
b:  [[-1.]]


Step:  100
w:  [[-0.95281274 -2.53970032]]
b:  [[-1.]]


within count steps
Linear Seperator does not exists
```

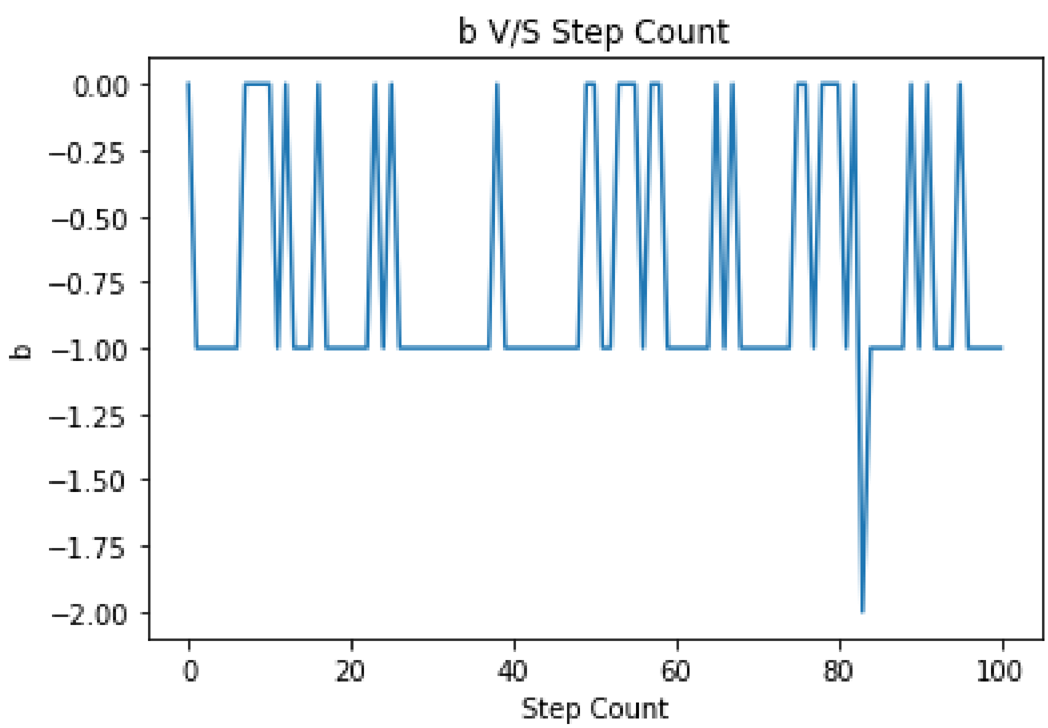
Lets look at how the values of w1, w2, b changes over time

```
In [188]: w = model[3]
          plot_list = sorted(w.items())
          x,y = zip(*plot_list)
          w1 = [i[0][0] for i in y]
```

```
w2 = [i[0][1] for i in y]

plt.plot(x, w1)
plt.title('w1 V/S Step Count')
plt.xlabel('Step Count')
plt.ylabel('w1')
plt.show()

plt.plot(x, w2)
plt.title('w2 V/S Step Count')
plt.xlabel('Step Count')
plt.ylabel('w2')
plt.show()

b = model[4]
plot_list = sorted(b.items())
x,y = zip(*plot_list)
y = [i[0][0] for i in y]

plt.plot(x,y)
plt.title('b V/S Step Count')
plt.xlabel('Step Count')
plt.ylabel('b')
plt.show()
```




w2 V/S Step Count
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
w2
0
20
40
60
80
100
Step Count


b V/S Step Count
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
b
0
20
40
60
80
100
Step Count

11 Conclusion

We have plotted the change in value of w1, w2 & b over time. Observing that we can infer that there is a pattern between w1, w2 & over time. This pattern is repeated for 3 to 4 time as seen above.

Fitting perceptron for 1000 steps did not lead us to a correctly classified perceptron. Futhermore using the heuristics found above i.e. the reoccuring pattern for the values of w1, w2 & b. We can say if the same pattern occurs more than 2 time we can terminate the algorithm and declare no seperator found!