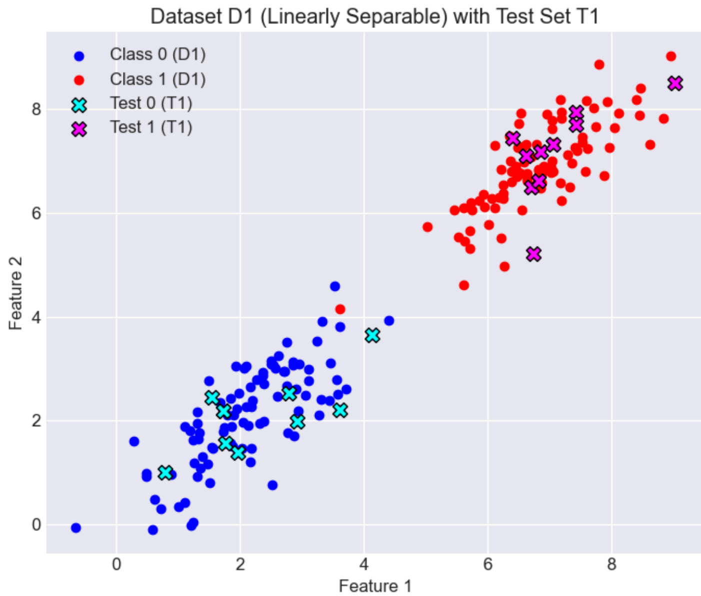
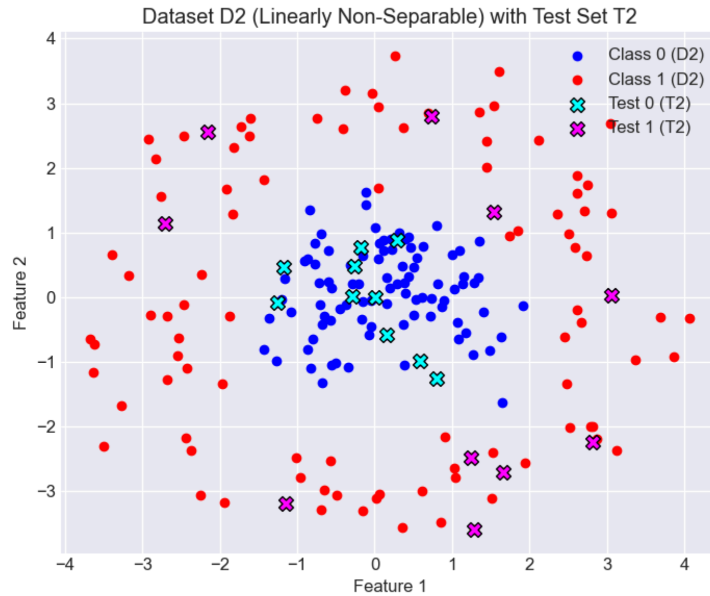
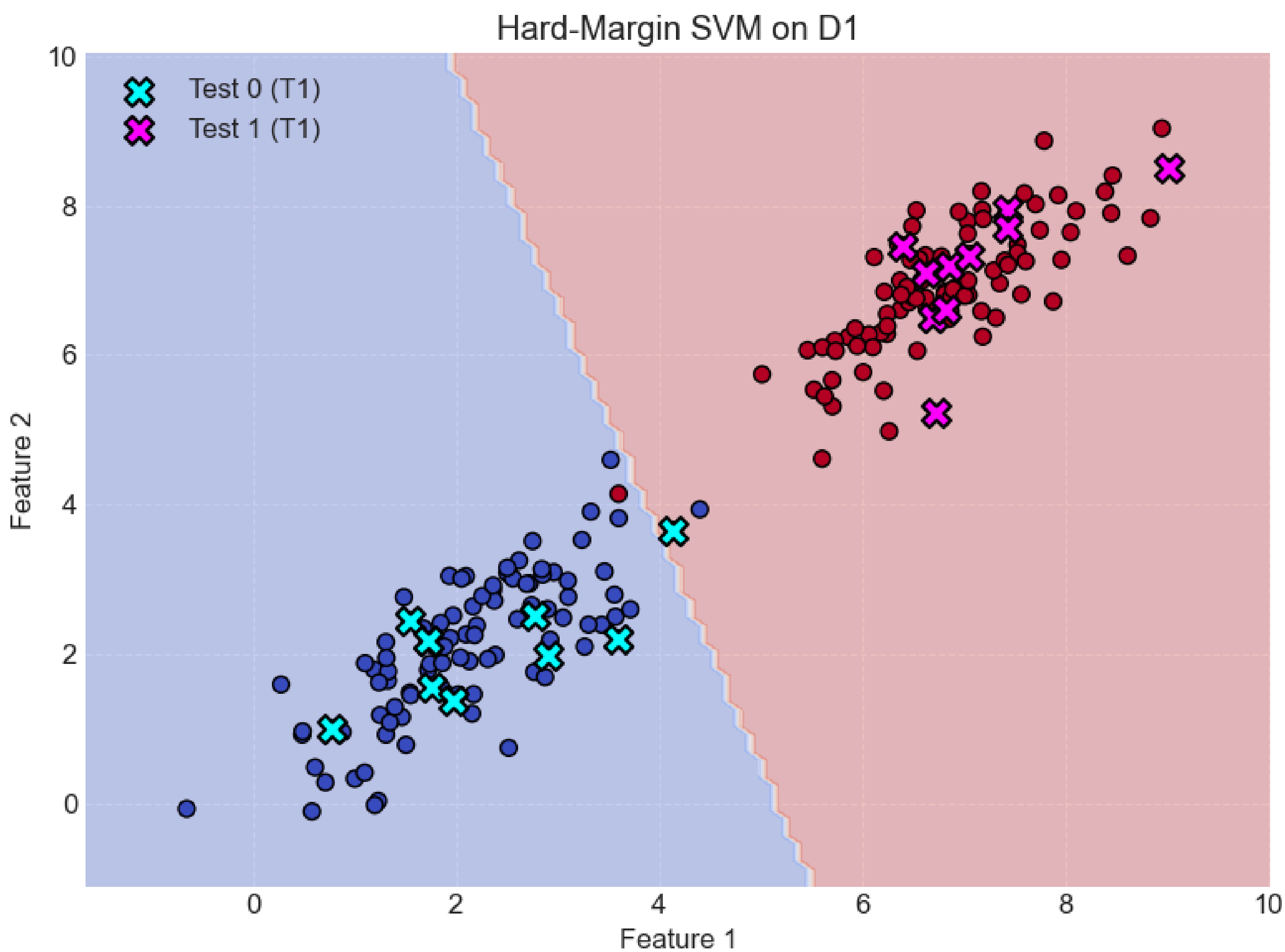
Omar Qawasmi 150210905

4. Plot the data points in D1 and in D2 as separate scatter plots. (5 points)





5. Implement the hard-margin SVM and obtain the results for D1. Report also your test set results on T1.



--- Hard-Margin SVM for D1 ---

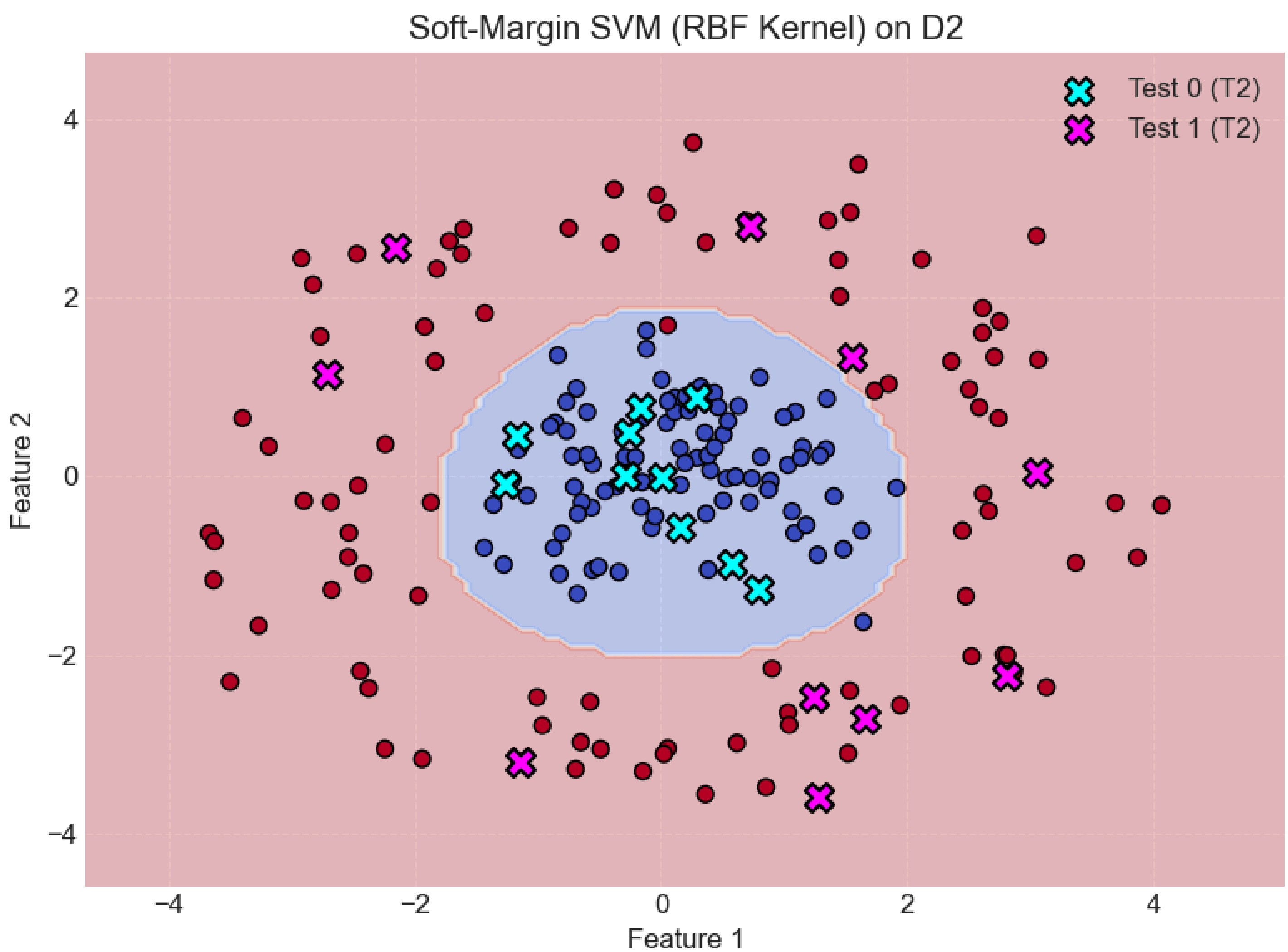
Hard-Margin SVM Training Accuracy (D1): 0.9889

Hard-Margin SVM Test Accuracy (T1): 0.9500

Hard-Margin SVM Classification Report (T1):

	precision	recall	f1-score	support
0.0	1.00	0.90	0.95	10
1.0	0.91	1.00	0.95	10
accuracy			0.95	20
macro avg	0.95	0.95	0.95	20
weighted avg	0.95	0.95	0.95	20

6. Implement the soft-margin SVM and obtain the results for D2. Report also your test set results on T2.



--- Soft-Margin SVM for D2 ---
Soft-Margin SVM Training Accuracy (D2): 0.9889
Soft-Margin SVM Test Accuracy (T2): 1.0000

Soft-Margin SVM Classification Report (T2):

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	10
1.0	1.00	1.00	1.00	10

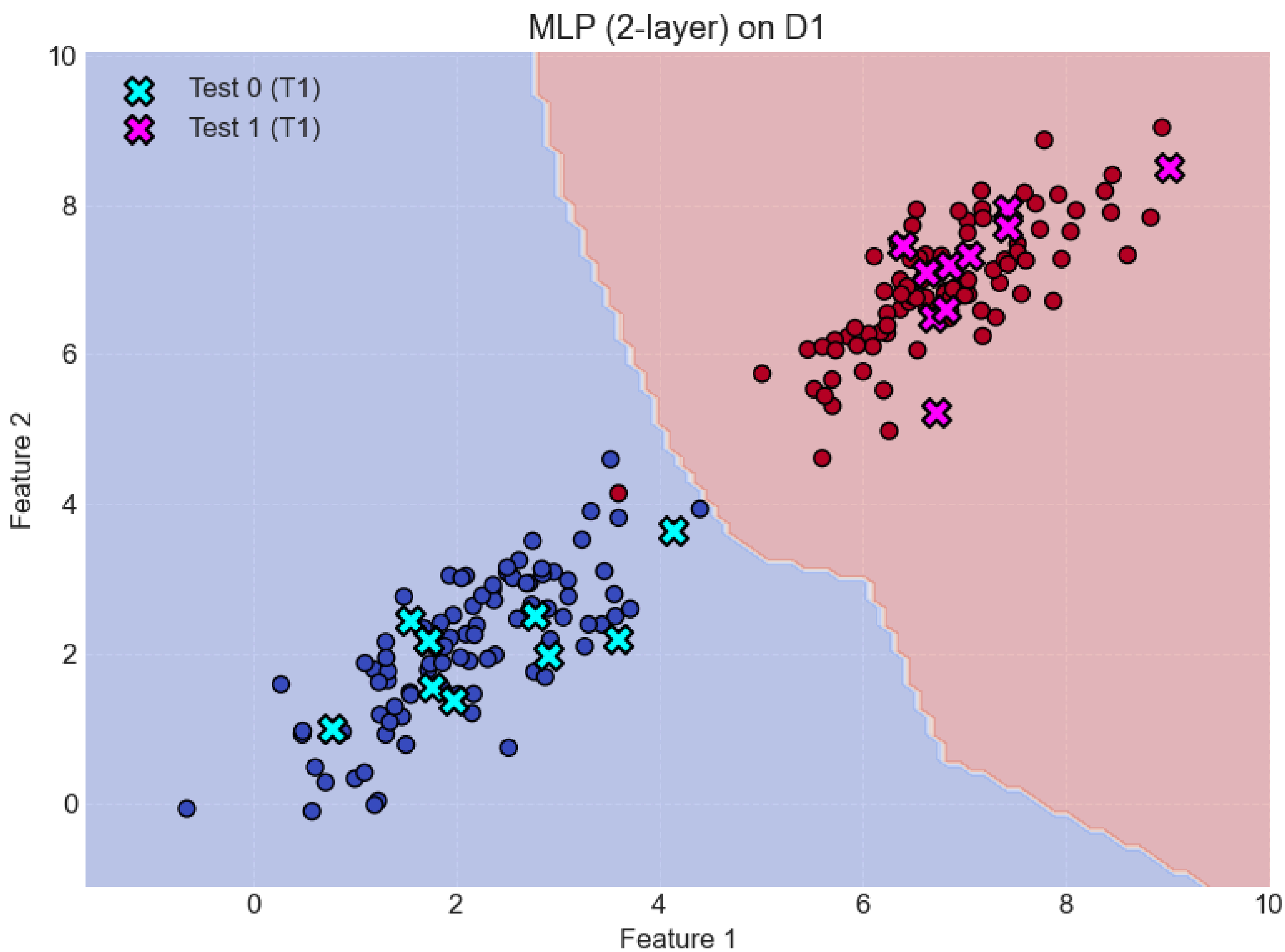
	precision	recall	f1-score	support
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

7. Implement a two-layer multi-layer-perceptron (MLP) structure and use it to classify the data points in D1 and D2. Compare your SVM results, MLP results, and comment.

--- Two-layer MLP for D1 ---
MLP Training Accuracy (D1): 0.9944
MLP Test Accuracy (T1): 1.0000

MLP Classification Report (T1):

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	10
1.0	1.00	1.00	1.00	10
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

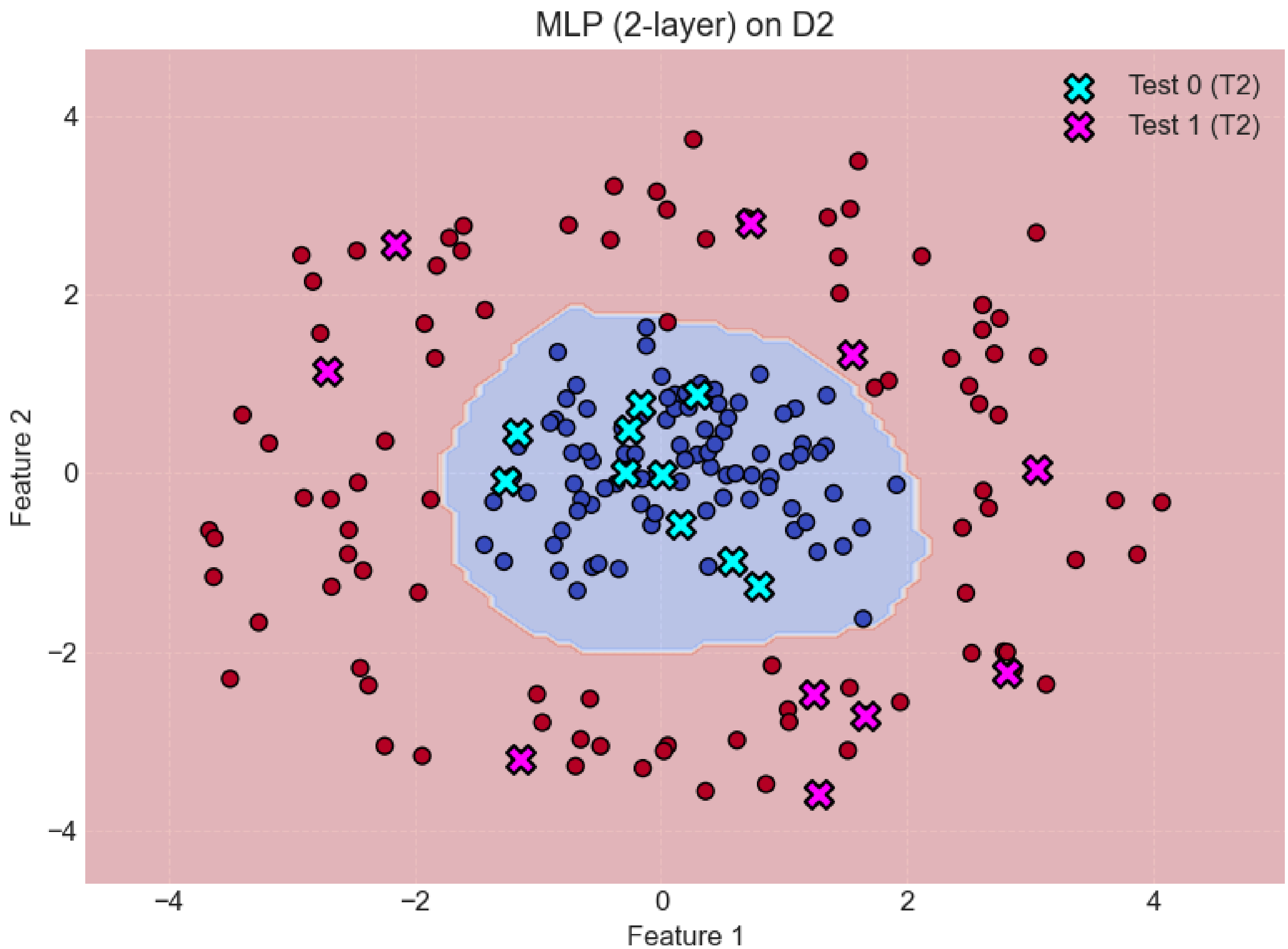


--- Two-layer MLP for D2 ---
MLP Training Accuracy (D2): 0.9944

MLP Test Accuracy (T2): 1.0000

MLP Classification Report (T2):

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	10
1.0	1.00	1.00	1.00	10
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20



Summary
Dataset D1 (Linearly Separable):
Hard-Margin SVM Training Accuracy: 0.9889
Hard-Margin SVM Test Accuracy (T1): 0.9500
MLP Training Accuracy: 0.9944
MLP Test Accuracy (T1): 1.0000

Dataset D2 (Linearly Non-Separable):
Soft-Margin SVM Training Accuracy: 0.9889
Soft-Margin SVM Test Accuracy (T2): 1.0000
MLP Training Accuracy: 0.9944

MLP Test Accuracy (T2): 1.0000

Comments: It is necessary to distinguish the kernel when using SVM linear and RBF. Both functions work excellently on the data with great accuracy, though mlp preformed better over all.

Link to files: https://github.com/itu-itis-qawasmi21/Blg454E.git
Github link itu-itis-qawasmi21/Blg454E: contains the answers to blg454e final
https://github.com/itu-itis-qawasmi21/Blg454E

Here is the data in case the linked did not work: data.txt:

--- D1 Data ---

Feature1	Feature2	Label
2.773679	1.759865	0.000000
3.464832	3.104079	0.000000
2.100730	2.261926	0.000000
1.489174	2.760904	0.000000
1.947464	2.214487	0.000000
2.390370	1.986899	0.000000
1.683416	2.343836	0.000000
3.605251	2.210101	0.000000
2.532438	3.059256	0.000000
2.728222	2.945447	0.000000
0.488276	0.924721	0.000000
3.605916	3.817451	0.000000
2.607042	2.466323	0.000000
2.964310	3.094500	0.000000
1.468938	1.156644	0.000000
3.563317	2.794046	0.000000
0.614374	0.483589	0.000000
2.368618	2.858241	0.000000
2.781519	2.519280	0.000000
2.855725	3.061299	0.000000
1.550906	2.448100	0.000000
2.316236	1.932797	0.000000
2.848069	3.136156	0.000000
2.747297	2.654351	0.000000
2.100452	3.041337	0.000000
3.439165	2.392636	0.000000
2.134972	1.904924	0.000000
2.170184	2.637934	0.000000
1.333068	1.644112	0.000000
0.716149	0.283716	0.000000
1.312724	0.923408	0.000000
2.522418	0.745601	0.000000
2.379907	2.713708	0.000000
1.254854	1.184404	0.000000
2.624235	3.251084	0.000000
1.236439	0.034717	0.000000
1.796074	2.099651	0.000000
2.160610	1.204182	0.000000
1.727981	1.788396	0.000000
1.974117	2.515867	0.000000
1.352147	1.085854	0.000000

1.005754	0.332948	0.000000
0.583772	-0.105917	0.000000
1.102405	0.413516	0.000000
1.196467	1.792125	0.000000
2.261166	2.776532	0.000000
2.904946	2.601109	0.000000
1.745614	1.869091	0.000000
1.335291	1.768080	0.000000
-0.650276	-0.070628	0.000000
1.868285	1.879509	0.000000
1.890189	2.098289	0.000000
1.937538	3.045330	0.000000
2.024877	1.459283	0.000000
2.368658	2.918387	0.000000
1.887986	1.490575	0.000000
3.057477	2.486122	0.000000
1.767361	1.553022	0.000000
3.242989	3.527443	0.000000
4.405310	3.935180	0.000000
0.491949	0.966225	0.000000
4.138977	3.657932	0.000000
2.507251	3.091609	0.000000
1.312287	2.160353	0.000000
1.243994	1.619835	0.000000
3.109018	2.763263	0.000000
2.916539	1.988809	0.000000
0.783463	1.001310	0.000000
3.331854	3.904646	0.000000
2.563750	3.011051	0.000000
2.931386	2.191126	0.000000
3.574175	2.499414	0.000000
3.269972	2.099663	0.000000
1.966950	1.368504	0.000000
2.208049	2.381675	0.000000
3.721012	2.598292	0.000000
1.202829	-0.020475	0.000000
2.696850	2.941225	0.000000
1.852502	1.568136	0.000000
2.757402	3.511315	0.000000
1.548917	1.483313	0.000000
1.851860	2.417618	0.000000
0.885891	0.954276	0.000000
1.729717	2.179342	0.000000
3.526535	4.597694	0.000000
3.103655	2.981785	0.000000
1.555355	1.450176	0.000000
0.278763	1.591750	0.000000
1.721860	2.190317	0.000000
1.106970	1.877982	0.000000
2.045500	1.953069	0.000000
2.184764	2.253606	0.000000
1.510316	0.786442	0.000000
1.400844	1.290809	0.000000
2.880109	1.691372	0.000000

2.508453	3.154730	0.000000
2.056963	3.010082	0.000000
3.308864	2.394263	0.000000
2.176023	1.462878	0.000000
1.316841	1.948327	0.000000
6.642361	6.625148	1.000000
7.712352	8.019989	1.000000
6.698255	6.508910	1.000000
7.358676	6.959780	1.000000
6.822242	6.848577	1.000000
7.051376	7.324878	1.000000
6.377982	6.604588	1.000000
7.027290	6.877656	1.000000
6.247956	6.284141	1.000000
5.866745	6.240124	1.000000
5.528978	5.535782	1.000000
6.862615	7.182484	1.000000
6.633926	6.658659	1.000000
6.626734	7.098378	1.000000
7.322010	6.499435	1.000000
7.178982	8.190892	1.000000
7.535247	7.473966	1.000000
7.298175	7.130960	1.000000
6.627988	7.335552	1.000000
7.052270	6.802626	1.000000
6.705127	7.091294	1.000000
7.796232	8.867578	1.000000
7.935785	8.139827	1.000000
5.466058	6.066449	1.000000
5.735512	6.191755	1.000000
6.538541	7.936226	1.000000
3.604541	4.143510	1.000000
6.464761	6.705557	1.000000
6.632256	6.764231	1.000000
6.013602	5.771836	1.000000
5.711025	5.314637	1.000000
8.112033	7.928218	1.000000
6.858098	6.718941	1.000000
6.827089	6.621194	1.000000
7.533033	7.367491	1.000000
8.058197	7.642570	1.000000
7.885782	6.717314	1.000000
6.269613	4.979574	1.000000
7.070826	7.353690	1.000000
6.564895	7.284137	1.000000
6.873268	6.587027	1.000000
6.377137	6.996691	1.000000
5.708187	5.664504	1.000000
6.189392	6.298743	1.000000
8.618225	7.329071	1.000000
7.966920	7.274237	1.000000
8.956960	9.032518	1.000000
6.250827	6.550019	1.000000
8.457784	7.898298	1.000000

8.396901	8.185189	1.000000
7.407112	7.263875	1.000000
7.042357	7.790382	1.000000
6.910567	6.895828	1.000000
6.357262	7.485198	1.000000
6.784599	7.319736	1.000000
7.185921	7.942977	1.000000
7.191796	6.246725	1.000000
6.216162	5.522659	1.000000
7.432073	7.948333	1.000000
6.069425	6.276078	1.000000
6.817712	6.816708	1.000000
6.872557	6.755971	1.000000
6.956883	7.917677	1.000000
6.396215	7.447271	1.000000
5.017876	5.743003	1.000000
6.547385	6.057371	1.000000
7.615808	7.255286	1.000000
7.044325	7.623876	1.000000
7.602223	8.163497	1.000000
5.613409	6.101491	1.000000
6.735596	5.221189	1.000000
6.584656	6.992768	1.000000
6.497756	6.779981	1.000000
6.899905	6.877563	1.000000
6.479644	7.270181	1.000000
7.754171	7.672441	1.000000
6.109893	6.105067	1.000000
7.178285	6.587037	1.000000
7.429836	7.699810	1.000000
9.030499	8.506198	1.000000
6.536809	6.757198	1.000000
6.446538	6.912364	1.000000
8.844221	7.830187	1.000000
7.572820	6.812243	1.000000
8.471305	8.402479	1.000000
6.388605	6.807012	1.000000
7.016310	6.791191	1.000000
5.635018	5.447478	1.000000
7.440641	7.205131	1.000000
7.049534	6.997479	1.000000
6.859997	6.485687	1.000000
6.222646	6.845609	1.000000
6.250666	6.388410	1.000000
6.497477	7.722904	1.000000
6.122950	7.310052	1.000000
5.609320	4.613433	1.000000
5.742407	6.057906	1.000000
7.195666	7.821518	1.000000
5.934864	6.354919	1.000000
5.954862	6.120604	1.000000

--- End of D1 Data ---

--- D2 Data ---

Feature1	Feature2	Label
0.155496	0.309417	0.000000
-0.706110	-0.122922	0.000000
1.914559	-0.133163	0.000000
1.031048	0.123813	0.000000
0.005167	1.078090	0.000000
0.384739	-1.048603	0.000000
-0.556727	0.134795	0.000000
0.004193	-0.009878	0.000000
1.339554	0.298938	0.000000
-0.050540	-0.056342	0.000000
-0.687418	0.981218	0.000000
0.114787	0.877041	0.000000
-0.600243	0.718295	0.000000
0.159568	-0.098372	0.000000
-0.860352	0.598231	0.000000
0.402734	0.064494	0.000000
-0.365671	-0.121099	0.000000
-0.456006	-0.173144	0.000000
-0.174881	0.770615	0.000000
0.505989	-0.276723	0.000000
0.291012	0.878779	0.000000
-0.679869	-1.317387	0.000000
-1.207555	-0.039577	0.000000
0.357452	0.484986	0.000000
-0.119000	1.423920	0.000000
0.508503	0.464321	0.000000
-0.076773	-0.584415	0.000000
1.629501	-0.611392	0.000000
0.323033	0.998516	0.000000
-0.049321	-0.452745	0.000000
1.403738	-0.226886	0.000000
-0.870351	-0.806748	0.000000
-1.360791	-0.322498	0.000000
1.084082	-0.640359	0.000000
-0.294294	0.213508	0.000000
-0.722108	0.222455	0.000000
-0.118703	1.626882	0.000000
-0.595389	0.237429	0.000000
0.739232	-0.025480	0.000000
-0.158749	0.639016	0.000000
-0.313967	0.498318	0.000000
1.148939	0.322700	0.000000
-1.086941	-0.220501	0.000000
0.631125	0.787041	0.000000
0.397873	0.894969	0.000000
-0.566277	-0.356446	0.000000
1.289682	0.226690	0.000000
1.347053	0.868235	0.000000
0.803771	1.103249	0.000000
0.044094	0.594421	0.000000
-1.159461	0.294908	0.000000
-0.900932	0.559805	0.000000
0.812494	0.214433	0.000000

-0.164203	-0.348114	0.000000
0.112594	0.718902	0.000000
-0.805441	-0.646281	0.000000
0.223918	0.731530	0.000000
-0.258170	0.484032	0.000000
0.204243	0.893077	0.000000
0.056224	0.839791	0.000000
0.534017	-0.028243	0.000000
-1.262529	-0.086550	0.000000
1.091864	0.721055	0.000000
-0.557439	-1.051109	0.000000
0.293252	0.202720	0.000000
-1.276385	-0.992563	0.000000
0.796601	-1.255761	0.000000
-0.284981	0.010799	0.000000
0.888393	-0.054042	0.000000
-0.838998	1.352138	0.000000
1.271804	-0.884333	0.000000
-0.154636	-0.066736	0.000000
-0.645846	-0.294481	0.000000
0.161402	-0.575903	0.000000
-0.676529	-0.425998	0.000000
0.434034	0.930337	0.000000
-1.435487	-0.805644	0.000000
-0.508788	-1.016471	0.000000
1.134822	0.203560	0.000000
0.368290	-0.424387	0.000000
-0.826425	-1.097106	0.000000
1.063295	-0.396921	0.000000
0.465494	0.773499	0.000000
-1.175240	0.457009	0.000000
0.545481	0.616523	0.000000
-0.208476	0.210793	0.000000
0.382904	0.229829	0.000000
1.480530	-0.821336	0.000000
0.582199	-0.985585	0.000000
-0.343549	-1.073788	0.000000
1.642580	-1.632460	0.000000
0.606481	-0.006527	0.000000
0.873718	-0.152260	0.000000
0.721882	-0.298555	0.000000
0.437497	0.320622	0.000000
-0.768757	0.830385	0.000000
-0.768132	0.506108	0.000000
0.996386	0.663030	0.000000
0.196039	0.146353	0.000000
1.179504	-0.550639	0.000000
0.361818	-3.560644	1.000000
-0.964969	-2.793334	1.000000
1.243175	-2.478717	1.000000
1.452575	2.009344	1.000000
-1.821256	2.318934	1.000000
-2.714400	1.135568	1.000000
-3.672378	-0.641871	1.000000

-1.008015	-2.476171	1.000000
2.786664	-1.998515	1.000000
-3.636136	-1.163362	1.000000
2.743487	0.648101	1.000000
1.545215	1.319984	1.000000
2.478233	-1.344091	1.000000
1.285890	-3.591057	1.000000
-1.620696	2.485126	1.000000
-0.034306	3.146877	1.000000
-3.268784	-1.673493	1.000000
-0.491639	-3.057830	1.000000
-1.870964	-0.298193	1.000000
-2.245849	-3.055507	1.000000
-2.462227	-0.108375	1.000000
-0.694260	-3.280285	1.000000
1.537106	2.953328	1.000000
1.944231	-2.565189	1.000000
-2.241736	0.355386	1.000000
3.371101	-0.974072	1.000000
-0.750309	2.774453	1.000000
0.054307	1.686124	1.000000
0.365015	2.615213	1.000000
-0.577220	-2.527460	1.000000
2.119250	2.424410	1.000000
-1.835788	1.281143	1.000000
1.850271	1.030797	1.000000
1.663109	-2.711159	1.000000
0.262571	3.731819	1.000000
2.524746	-2.017061	1.000000
-2.542758	-0.909336	1.000000
1.031258	-2.648050	1.000000
0.048305	2.945117	1.000000
-0.652769	-2.981784	1.000000
-2.471961	2.486373	1.000000
-2.536484	-0.638291	1.000000
-3.629247	-0.732401	1.000000
3.130661	-2.365494	1.000000
-1.720663	2.630674	1.000000
-3.502311	-2.303638	1.000000
-0.381639	3.208670	1.000000
-3.185114	0.331396	1.000000
4.070833	-0.328972	1.000000
2.586828	0.771362	1.000000
3.050669	2.687917	1.000000
-0.411046	2.607138	1.000000
-1.605241	2.764501	1.000000
-2.828651	2.141653	1.000000
1.040945	-2.787143	1.000000
0.054731	-3.050796	1.000000
2.449487	-0.613011	1.000000
1.356143	2.859615	1.000000
2.814653	-2.233229	1.000000
-2.769006	1.561927	1.000000
3.692560	-0.304831	1.000000

2.611875	1.602091	1.000000
1.519075	-3.105356	1.000000
3.059746	0.032940	1.000000
3.871361	-0.913385	1.000000
1.532658	-2.404596	1.000000
2.614271	1.879368	1.000000
0.854286	-3.482065	1.000000
0.902219	-2.153802	1.000000
-2.922265	2.436812	1.000000
-1.147116	-3.197697	1.000000
2.752218	1.729148	1.000000
3.064062	1.301476	1.000000
-3.401112	0.649617	1.000000
0.693425	2.855299	1.000000
-2.444618	-2.183341	1.000000
-1.921157	1.670882	1.000000
-2.902527	-0.279632	1.000000
-2.157694	2.557040	1.000000
0.729057	2.803427	1.000000
2.865103	-2.194803	1.000000
-2.421196	-1.093495	1.000000
-2.683516	-0.295348	1.000000
2.710118	1.332310	1.000000
2.505057	0.972452	1.000000
1.604235	3.489176	1.000000
-1.940924	-3.166905	1.000000
1.736706	0.953630	1.000000
-2.376731	-2.376431	1.000000
-2.677203	-1.272392	1.000000
0.617932	-2.990617	1.000000
2.663637	-0.394006	1.000000
-0.147815	-3.304965	1.000000
0.023608	-3.111239	1.000000
2.812549	-2.004257	1.000000
2.618861	-0.199292	1.000000
-1.971601	-1.338809	1.000000
1.440824	2.417369	1.000000
-1.430882	1.822333	1.000000
2.360030	1.281283	1.000000

--- End of D2 Data ---

--- T1 Data ---

Feature1	Feature2	Label
1.767361	1.553022	0.000000
1.550906	2.448100	0.000000
1.729717	2.179342	0.000000
2.781519	2.519280	0.000000
4.138977	3.657932	0.000000
1.721860	2.190317	0.000000
3.605251	2.210101	0.000000
2.916539	1.988809	0.000000
0.783463	1.001310	0.000000
1.966950	1.368504	0.000000
6.735596	5.221189	1.000000

6.862615	7.182484	1.000000
6.396215	7.447271	1.000000
7.432073	7.948333	1.000000
7.429836	7.699810	1.000000
6.698255	6.508910	1.000000
6.827089	6.621194	1.000000
6.626734	7.098378	1.000000
7.051376	7.324878	1.000000
9.030499	8.506198	1.000000

--- End of T1 Data ---

--- T2 Data ---

Feature1	Feature2	Label
-0.258170	0.484032	0.000000
0.291012	0.878779	0.000000
-1.175240	0.457009	0.000000
-0.174881	0.770615	0.000000
-1.262529	-0.086550	0.000000
0.582199	-0.985585	0.000000
0.004193	-0.009878	0.000000
0.796601	-1.255761	0.000000
-0.284981	0.010799	0.000000
0.161402	-0.575903	0.000000
-1.147116	-3.197697	1.000000
1.545215	1.319984	1.000000
3.059746	0.032940	1.000000
2.814653	-2.233229	1.000000
-2.157694	2.557040	1.000000
1.243175	-2.478717	1.000000
1.663109	-2.711159	1.000000
1.285890	-3.591057	1.000000
-2.714400	1.135568	1.000000
0.729057	2.803427	1.000000

--- End of T2 Data ---

Here is the code in case the link did not work:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

plt.rcParams.update({'font.size': 12})
plt.style.use('seaborn-v0_8-darkgrid')

np.random.seed(33)

# Class 0 for D1
mean_d1_c0 = [2, 2]
cov_d1_c0 = [[0.8, 0.6], [0.6, 0.8]]
X_d1_c0 = np.random.multivariate_normal(mean_d1_c0, cov_d1_c0, 100)
y_d1_c0 = np.zeros(100)

# Class 1 for D1
mean_d1_c1 = [7, 7]
cov_d1_c1 = [[0.8, 0.6], [0.6, 0.8]]
X_d1_c1 = np.random.multivariate_normal(mean_d1_c1, cov_d1_c1, 100)
y_d1_c1 = np.ones(100)

# Combine D1 classes
X_d1 = np.vstack((X_d1_c0, X_d1_c1))
y_d1 = np.hstack((y_d1_c0, y_d1_c1))

# Class 0 for D2 (inner circle-ish)
theta_d2_c0 = 2 * np.pi * np.random.rand(100)
r_d2_c0 = 1 + 0.5 * np.random.randn(100)
X_d2_c0 = np.array([r_d2_c0 * np.cos(theta_d2_c0), r_d2_c0 * np.sin(theta_d2_c0)]).T
y_d2_c0 = np.zeros(100)

# Class 1 for D2 (outer circle-ish)
theta_d2_c1 = 2 * np.pi * np.random.rand(100)
r_d2_c1 = 3 + 0.5 * np.random.randn(100)
X_d2_c1 = np.array([r_d2_c1 * np.cos(theta_d2_c1), r_d2_c1 * np.sin(theta_d2_c1)]).T
y_d2_c1 = np.ones(100)

# Combine D2 classes
X_d2 = np.vstack((X_d2_c0, X_d2_c1))
y_d2 = np.hstack((y_d2_c0, y_d2_c1))
```

```python
# For D1
# Temporarily split to get 20 test samples, then extract 10 from each class
X_train_d1, X_test_d1_temp, y_train_d1, y_test_d1_temp = train_test_split(
    X_d1, y_d1, test_size=20, stratify=y_d1, random_state=33
)
# Then, explicitly select the first 10 points from each class from the temporary test
set
# to guarantee exactly 10 Class 0 and 10 Class 1 samples for T1.
T1_X = np.vstack((X_test_d1_temp[y_test_d1_temp == 0][:10],
X_test_d1_temp[y_test_d1_temp == 1][:10]))
T1_y = np.hstack((y_test_d1_temp[y_test_d1_temp == 0][:10],
y_test_d1_temp[y_test_d1_temp == 1][:10]))

# For D2
# Temporarily split to get 20 test samples, then extract 10 from each class
X_train_d2, X_test_d2_temp, y_train_d2, y_test_d2_temp = train_test_split(
    X_d2, y_d2, test_size=20, stratify=y_d2, random_state=33
)
T2_X = np.vstack((X_test_d2_temp[y_test_d2_temp == 0][:10],
X_test_d2_temp[y_test_d2_temp == 1][:10]))
T2_y = np.hstack((y_test_d2_temp[y_test_d2_temp == 0][:10],
y_test_d2_temp[y_test_d2_temp == 1][:10]))

# Define a helper function for plotting decision boundaries (will be used in later
cells)
def plot_decision_boundary(model, X, y, title, ax, resolution=100):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, resolution),
                         np.linspace(y_min, y_max, resolution))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap=plt.cm.coolwarm,
edgecolors='k')
    ax.set_title(title)
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.legend(*scatter.legend_elements(), title="Classes")
    ax.grid(True, linestyle='--', alpha=0.7)
```

```python
# --- Code to export data to data.txt ---
def export_data_segment(filename, data_name, X_data, y_data, open_mode):
    """
    Exports a segment of data (X and y) to a text file with a header and footer.
    """
    y_data_reshaped = y_data.reshape(-1, 1)
    combined_data = np.hstack((X_data, y_data_reshaped))

    with open(filename, open_mode) as f:
        f.write(f"--- {data_name} Data ---\n")
        f.write("Feature1\tFeature2\tLabel\n")
        np.savetxt(f, combined_data, fmt='%.6f', delimiter='\t')
        f.write(f"--- End of {data_name} Data ---\n\n")

# Export D1 data (write mode to create/overwrite the file)
export_data_segment('data.txt', 'D1', X_d1, y_d1, 'w')

# Export D2 data (append mode)
export_data_segment('data.txt', 'D2', X_d2, y_d2, 'a')

# Export T1 data (append mode)
export_data_segment('data.txt', 'T1', T1_X, T1_y, 'a')

# Export T2 data (append mode)
export_data_segment('data.txt', 'T2', T2_X, T2_y, 'a')
```

```python
# Cell 2: Scatter Plots of D1 and D2

plt.figure(figsize=(14, 6))

# Plot D1 with its test set T1
plt.subplot(1, 2, 1)
plt.scatter(X_d1[y_d1 == 0, 0], X_d1[y_d1 == 0, 1], color='blue', label='Class 0
(D1)')
plt.scatter(X_d1[y_d1 == 1, 0], X_d1[y_d1 == 1, 1], color='red', label='Class 1
(D1)')
plt.scatter(T1_X[T1_y == 0, 0], T1_X[T1_y == 0, 1], color='cyan', marker='X', s=100,
label='Test 0 (T1)', edgecolors='black')
plt.scatter(T1_X[T1_y == 1, 0], T1_X[T1_y == 1, 1], color='magenta', marker='X',
s=100, label='Test 1 (T1)', edgecolors='black')
plt.title('Dataset D1 (Linearly Separable) with Test Set T1')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
```

```python
plt.grid(True)

# Plot D2 with its test set T2
plt.subplot(1, 2, 2)
plt.scatter(X_d2[y_d2 == 0, 0], X_d2[y_d2 == 0, 1], color='blue', label='Class 0 (D2)')
plt.scatter(X_d2[y_d2 == 1, 0], X_d2[y_d2 == 1, 1], color='red', label='Class 1 (D2)')
plt.scatter(T2_X[T2_y == 0, 0], T2_X[T2_y == 0, 1], color='cyan', marker='X', s=100, label='Test 0 (T2)', edgecolors='black')
plt.scatter(T2_X[T2_y == 1, 0], T2_X[T2_y == 1, 1], color='magenta', marker='X', s=100, label='Test 1 (T2)', edgecolors='black')
plt.title('Dataset D2 (Linearly Non-Separable) with Test Set T2')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

```python
# Cell 3: Hard-Margin SVM for D1

print("--- Hard-Margin SVM for D1 ---")
# C=1e10 effectively makes it a hard-margin SVM for perfectly separable data.
# 'linear' kernel is appropriate for linearly separable data.
svm_hard_margin = SVC(kernel='linear', C=1e10, random_state=33)
svm_hard_margin.fit(X_train_d1, y_train_d1)

# Training results for D1
y_pred_d1_svm_train = svm_hard_margin.predict(X_train_d1)
train_accuracy_d1_svm = accuracy_score(y_train_d1, y_pred_d1_svm_train)
print(f"Hard-Margin SVM Training Accuracy (D1): {train_accuracy_d1_svm:.4f}")

# Test set results on T1
y_pred_t1_svm = svm_hard_margin.predict(T1_X)
test_accuracy_t1_svm = accuracy_score(T1_y, y_pred_t1_svm)
print(f"Hard-Margin SVM Test Accuracy (T1): {test_accuracy_t1_svm:.4f}")

print("\nHard-Margin SVM Classification Report (T1):")
print(classification_report(T1_y, y_pred_t1_svm))

# Plotting Hard-Margin SVM for D1
fig, ax = plt.subplots(figsize=(8, 6))
```

```
plot_decision_boundary(svm_hard_margin, X_d1, y_d1, 'Hard-Margin SVM on D1', ax)
# Plot test points on top
ax.scatter(T1_X[T1_y == 0, 0], T1_X[T1_y == 0, 1], color='cyan', marker='X', s=150,
label='Test 0 (T1)', edgecolors='black', linewidth=1.5)
ax.scatter(T1_X[T1_y == 1, 0], T1_X[T1_y == 1, 1], color='magenta', marker='X',
s=150, label='Test 1 (T1)', edgecolors='black', linewidth=1.5)
ax.legend()
plt.tight_layout()
plt.show()
```

```
# Cell 4: Soft-Margin SVM for D2

print("\n--- Soft-Margin SVM for D2 ---")
# Dataset D2 is linearly non-separable (concentric circles).
# A linear kernel would fail to find an effective decision boundary for this type of
data,
# resulting in very low accuracy. This is because a linear kernel can only draw a
straight line
# or hyperplane, which cannot adequately separate an inner circle from an outer
circle.
# Therefore, to correctly classify D2, a non-linear kernel is essential.
# 'rbf' kernel (Radial Basis Function) is good for non-linear separation.
# notes on rbf in nural networks document from class notes
svm_soft_margin = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=33)
svm_soft_margin.fit(X_train_d2, y_train_d2)

# Training results for D2
y_pred_d2_svm_train = svm_soft_margin.predict(X_train_d2)
train_accuracy_d2_svm = accuracy_score(y_train_d2, y_pred_d2_svm_train)
print(f"Soft-Margin SVM Training Accuracy (D2): {train_accuracy_d2_svm:.4f}")

# Test set results on T2
y_pred_t2_svm = svm_soft_margin.predict(T2_X)
test_accuracy_t2_svm = accuracy_score(T2_y, y_pred_t2_svm)
print(f"Soft-Margin SVM Test Accuracy (T2): {test_accuracy_t2_svm:.4f}")

print("\nSoft-Margin SVM Classification Report (T2):")
print(classification_report(T2_y, y_pred_t2_svm))

# Plotting Soft-Margin SVM for D2
fig, ax = plt.subplots(figsize=(8, 6))
plot_decision_boundary(svm_soft_margin, X_d2, y_d2, 'Soft-Margin SVM (RBF Kernel) on
D2', ax)
# Plot test points on top
```

```
ax.scatter(T2_X[T2_y == 0, 0], T2_X[T2_y == 0, 1], color='cyan', marker='X', s=150,
label='Test 0 (T2)', edgecolors='black', linewidth=1.5)
ax.scatter(T2_X[T2_y == 1, 0], T2_X[T2_y == 1, 1], color='magenta', marker='X',
s=150, label='Test 1 (T2)', edgecolors='black', linewidth=1.5)
ax.legend()
plt.tight_layout()
plt.show()
```

```
# Cell 5: MLP, Comparison, and Comments

print("\n--- Two-layer MLP for D1 ---")
# Two layers, e.g., (10, 5) means 1st hidden layer with 10 neurons, 2nd with 5.
# Activation function 'relu' optimizer 'SGD'.
mlp_d1 = MLPClassifier(hidden_layer_sizes=(10, 5), activation='relu', solver='sgd',
                           learning_rate_init=0.01, momentum=0.9,
                           max_iter=2000, random_state=33, verbose=False)
mlp_d1.fit(X_train_d1, y_train_d1)

# Training results for D1
y_pred_d1_mlp_train = mlp_d1.predict(X_train_d1)
train_accuracy_d1_mlp = accuracy_score(y_train_d1, y_pred_d1_mlp_train)
print(f"MLP Training Accuracy (D1): {train_accuracy_d1_mlp:.4f}")

# Test set results on T1
y_pred_t1_mlp = mlp_d1.predict(T1_X)
test_accuracy_t1_mlp = accuracy_score(T1_y, y_pred_t1_mlp)
print(f"MLP Test Accuracy (T1): {test_accuracy_t1_mlp:.4f}")

print("\nMLP Classification Report (T1):")
print(classification_report(T1_y, y_pred_t1_mlp))

# Plotting MLP for D1
fig, ax = plt.subplots(figsize=(8, 6))
plot_decision_boundary(mlp_d1, X_d1, y_d1, 'MLP (2-layer) on D1', ax)
# Plot test points on top
ax.scatter(T1_X[T1_y == 0, 0], T1_X[T1_y == 0, 1], color='cyan', marker='X', s=150,
label='Test 0 (T1)', edgecolors='black', linewidth=1.5)
ax.scatter(T1_X[T1_y == 1, 0], T1_X[T1_y == 1, 1], color='magenta', marker='X',
s=150, label='Test 1 (T1)', edgecolors='black', linewidth=1.5)
ax.legend()
plt.tight_layout()
plt.show()
```

```python
print("\n--- Two-layer MLP for D2 ---")
# Slightly larger layers for D2 to handle more complexity
mlp_d2 = MLPClassifier(hidden_layer_sizes=(20, 10), activation='relu', solver='sgd',
                           learning_rate_init=0.01, momentum=0.9,
                           max_iter=2000, random_state=33, verbose=False)
mlp_d2.fit(X_train_d2, y_train_d2)

# Training results for D2
y_pred_d2_mlp_train = mlp_d2.predict(X_train_d2)
train_accuracy_d2_mlp = accuracy_score(y_train_d2, y_pred_d2_mlp_train)
print(f"MLP Training Accuracy (D2): {train_accuracy_d2_mlp:.4f}")

# Test set results on T2
y_pred_t2_mlp = mlp_d2.predict(T2_X)
test_accuracy_t2_mlp = accuracy_score(T2_y, y_pred_t2_mlp)
print(f"MLP Test Accuracy (T2): {test_accuracy_t2_mlp:.4f}")

print("\nMLP Classification Report (T2):")
print(classification_report(T2_y, y_pred_t2_mlp))

# Plotting MLP for D2
fig, ax = plt.subplots(figsize=(8, 6))
plot_decision_boundary(mlp_d2, X_d2, y_d2, 'MLP (2-layer) on D2', ax)
# Plot test points on top
ax.scatter(T2_X[T2_y == 0, 0], T2_X[T2_y == 0, 1], color='cyan', marker='X', s=150,
label='Test 0 (T2)', edgecolors='black', linewidth=1.5)
ax.scatter(T2_X[T2_y == 1, 0], T2_X[T2_y == 1, 1], color='magenta', marker='X',
s=150, label='Test 1 (T2)', edgecolors='black', linewidth=1.5)
ax.legend()
plt.tight_layout()
plt.show()

# --- Comparison Report ---
print("\n--- Summary of Results ---")

print("\nDataset D1 (Linearly Separable):")
print(f"Hard-Margin SVM Training Accuracy: {train_accuracy_d1_svm:.4f}")
print(f"Hard-Margin SVM Test Accuracy (T1): {test_accuracy_t1_svm:.4f}")
print(f"MLP Training Accuracy: {train_accuracy_d1_mlp:.4f}")
print(f"MLP Test Accuracy (T1): {test_accuracy_t1_mlp:.4f}")

print("\nDataset D2 (Linearly Non-Separable):")
print(f"Soft-Margin SVM Training Accuracy: {train_accuracy_d2_svm:.4f}")
print(f"Soft-Margin SVM Test Accuracy (T2): {test_accuracy_t2_svm:.4f}")
```

```
print(f"MLP Training Accuracy: {train_accuracy_d2_mlp:.4f}")
print(f"MLP Test Accuracy (T2): {test_accuracy_t2_mlp:.4f}")
```