Q1.

Data is normalised using standard normalization.
Both label and one hot encoding done, no significant difference in performance between the two.

(a) predict function is implemented using np.dot(X_test,reg_clf.coef_.T) + self.model.intercept_

(b)

	mse_validation_implemented	mse_validation_inbuilt	mse_train_implemented	mse_train_inbuilt
0	0.483463	0.483463	0.468164	0.468164
1	0.423525	0.423525	0.462067	0.462067
2	0.493026	0.493026	0.459132	0.459132
3	0.447839	0.447839	0.462163	0.462163
4	0.520417	0.520417	0.450396	0.450396

(c) predictions were done using normal equations

```
np.dot(xtest,theta)
```

	mse_validation_implemented	mse_validation_inbuilt	mse_train_implemented	mse_train_inbuilt
0	0.480492	0.480492	0.470772	0.470772
1	0.422817	0.422817	0.462652	0.462652
2	0.470407	0.470407	0.462612	0.462612
3	0.442150	0.442150	0.463060	0.463060
4	0.520417	0.520417	0.450396	0.450396

(d)

There are no major variations in performances between the three approaches, also the values keep changing in each run.

	mse_validation_implemented	mse_validation_inbuilt	mse_train_implemented	mse_train_inbuilt
0	0.483463	0.483463	0.468164	0.468164
1	0.423525	0.423525	0.462067	0.462067
2	0.493026	0.493026	0.459132	0.459132
3	0.447839	0.447839	0.462163	0.462163
4	0.520417	0.520417	0.450396	0.450396

Q2.

Data is normalised before training and testing to bring all values to the same scale which helps the model to converge faster.

(a) Visualization

Datatypes:

```
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
```

No categorical data, hence does not require encoding.

Statistics:
From here we can see the statistical information of the dataset.

Clearly, scale is highly uneven for different columns, hence we normalise the data before fitting.

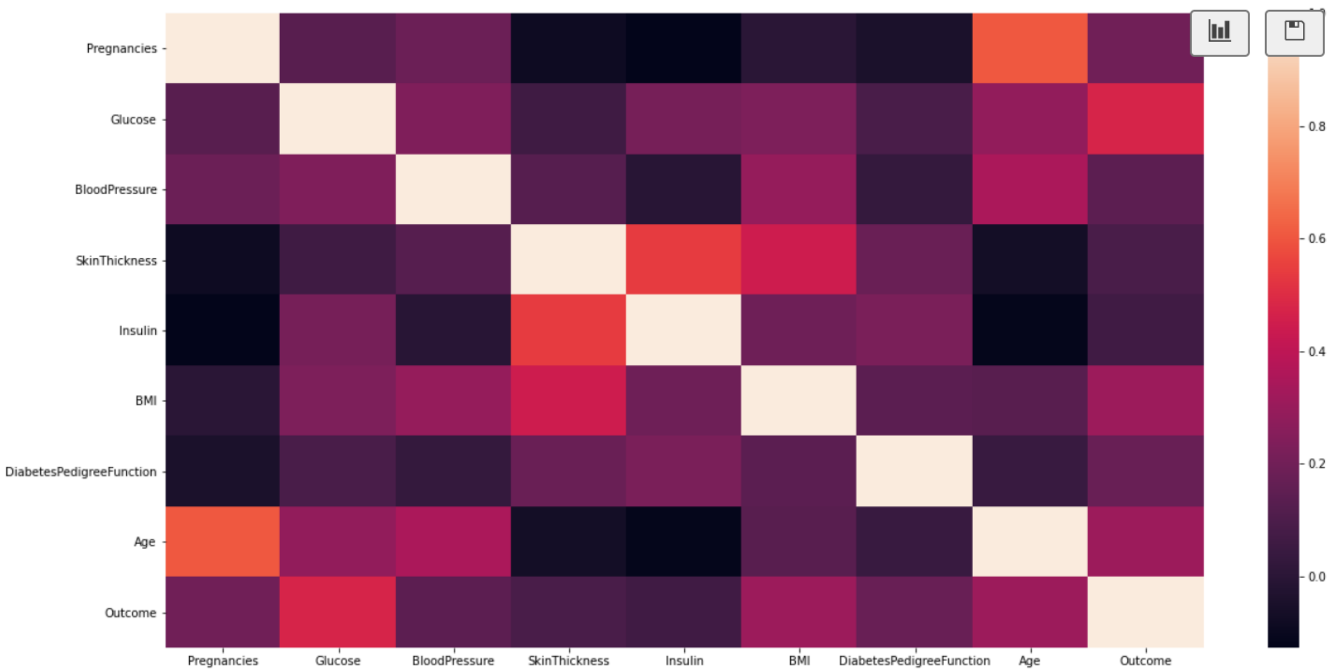
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Correlations:

The following heatmap shows the correlation between different columns.

Lightly shaded pairs depict a strong correlation.

Age, BMI, Glucose are some strongly correlated values with the outcome.



Means:
Average values of each column.

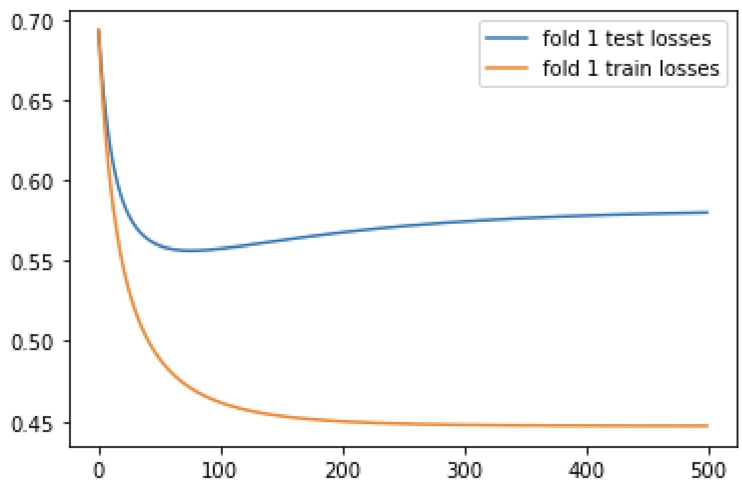
```
Pregnancies                   3.845052
Glucose                     120.894531
BloodPressure                69.105469
SkinThickness                20.536458
Insulin                      79.799479
BMI                          31.992578
DiabetesPedigreeFunction      0.471876
Age                          33.240885
Outcome                       0.348958
Name: mean, dtype: float64
```

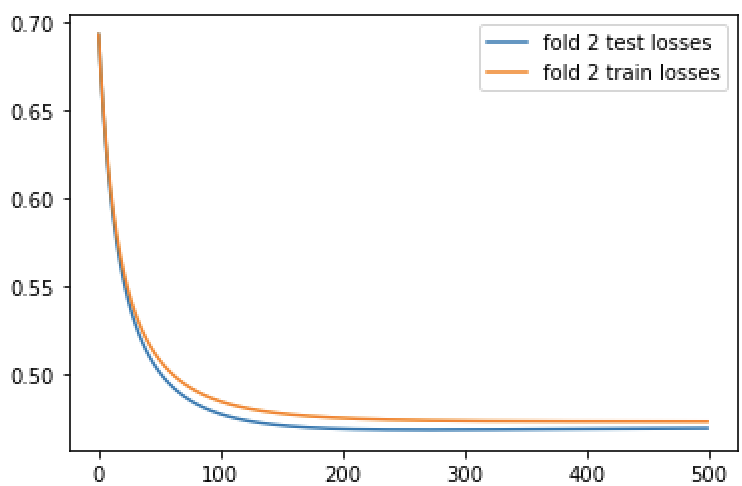
(b) the class has three main functions:
.fit, .predict, and .getProbab which fits the model, predicts the classes, and gives the probabilities respectively.

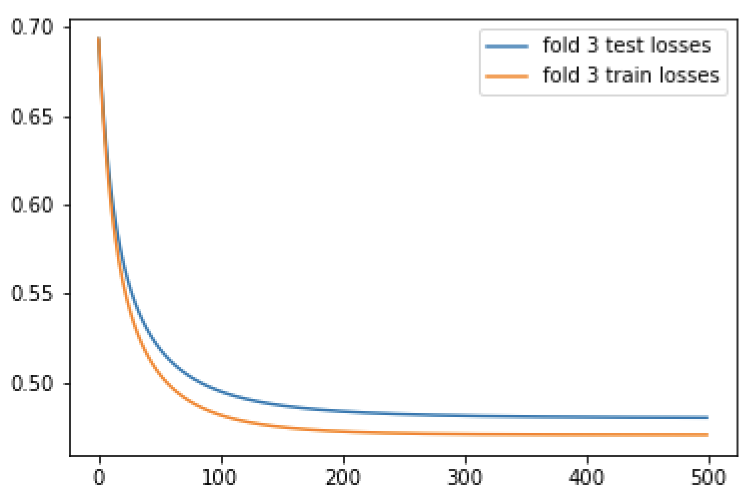
(c)

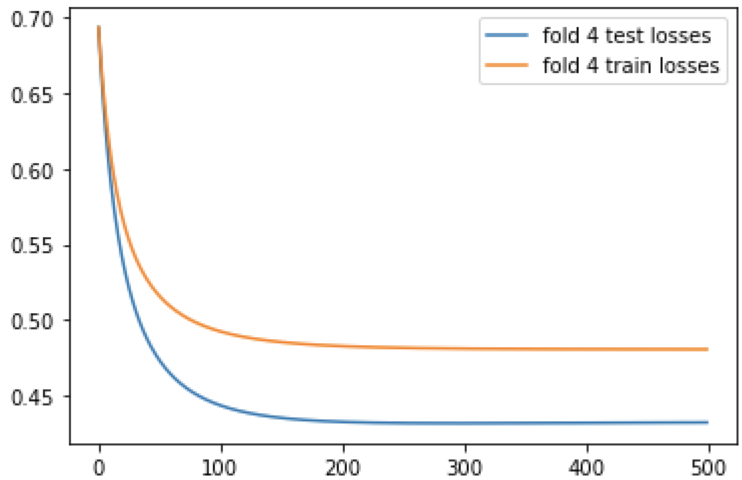
Fold	Lambda Value	Test Accuracy	Training Accuracy	Test Error	Train Error
1	0.0000	71.241830	79.512195	0.182028	0.182028
2	0.0000	83.006536	77.073171	0.131169	0.131169
3	0.0000	73.202614	77.398374	0.163485	0.163485
4	0.0000	79.738562	77.723577	0.146435	0.146435
5	0.0000	77.124183	77.886179	0.165088	0.165088

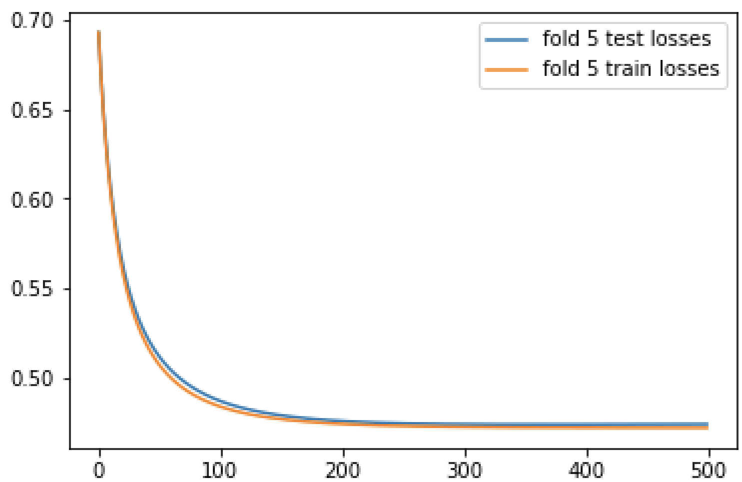
Loss Curves:


0.70
0.65
0.60
0.55
0.50
0.45
0
100
200
300
400
500
fold 1 test losses
fold 1 train losses

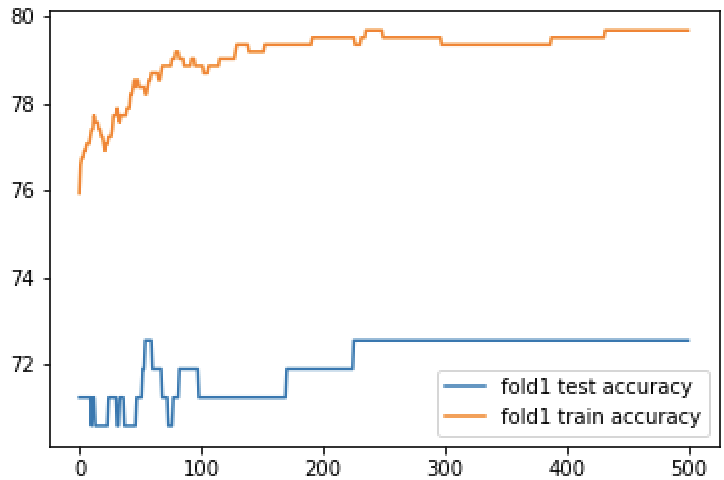

0.70
0.65
0.60
0.55
0.50
0
100
200
300
400
500
fold 2 test losses
fold 2 train losses

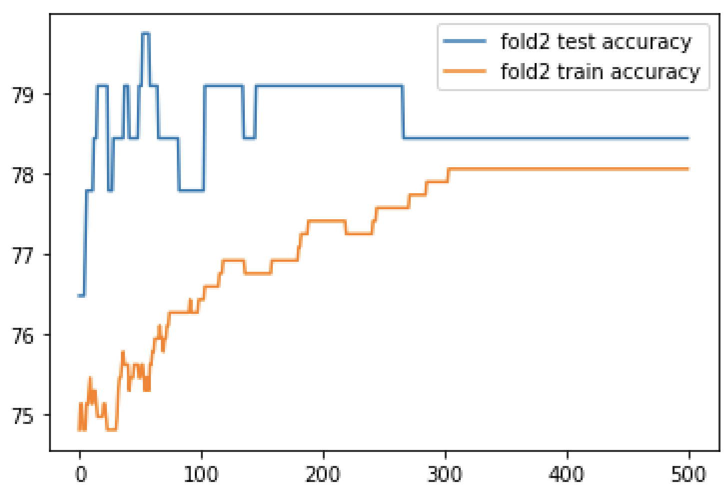

0.70
0.65
0.60
0.55
0.50
0
100
200
300
400
500
fold 3 test losses
fold 3 train losses

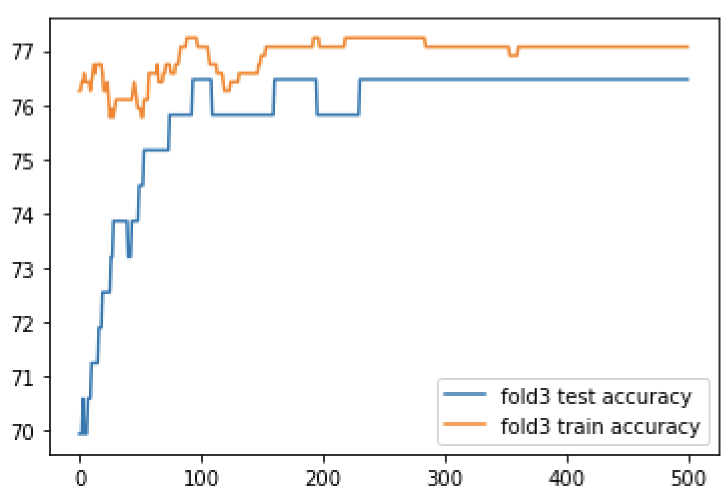

0.70
0.65
0.60
0.55
0.50
0.45
0
100
200
300
400
500
fold 4 test losses
fold 4 train losses

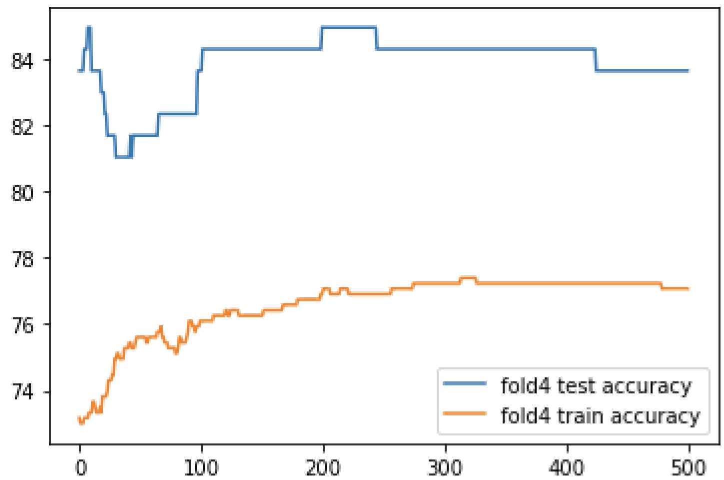

0.70
0.65
0.60
0.55
0.50
0
100
200
300
400
500
fold 5 test losses
fold 5 train losses

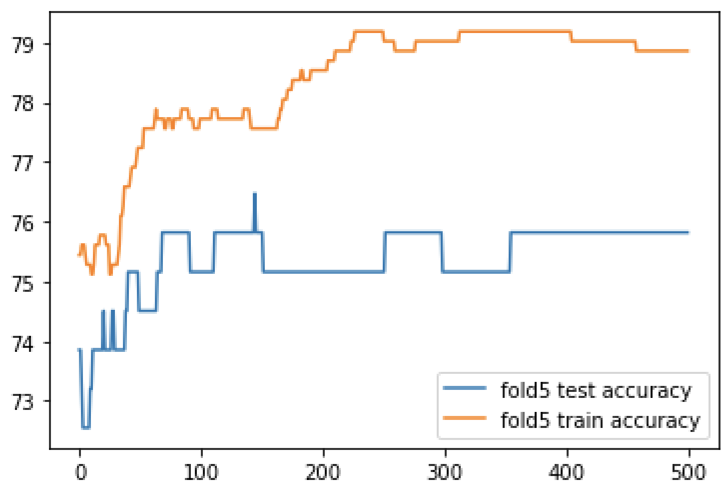
Accuracy Plots:









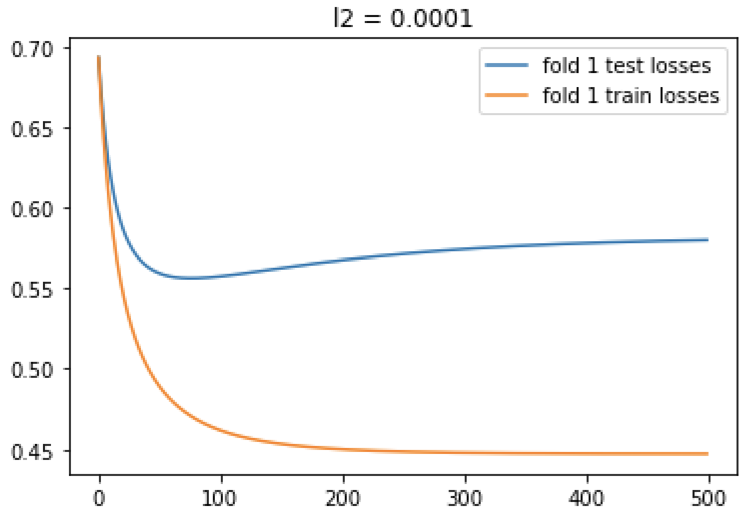


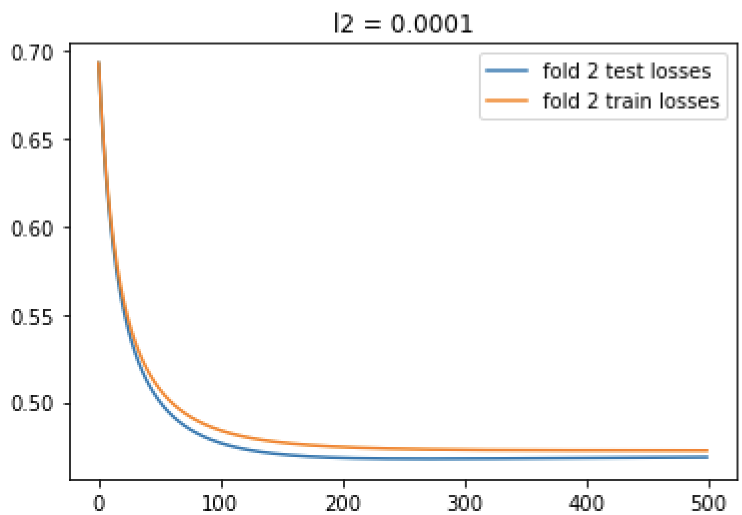
Inference: The model starts to converge at 300-400 epochs in most of the folds.

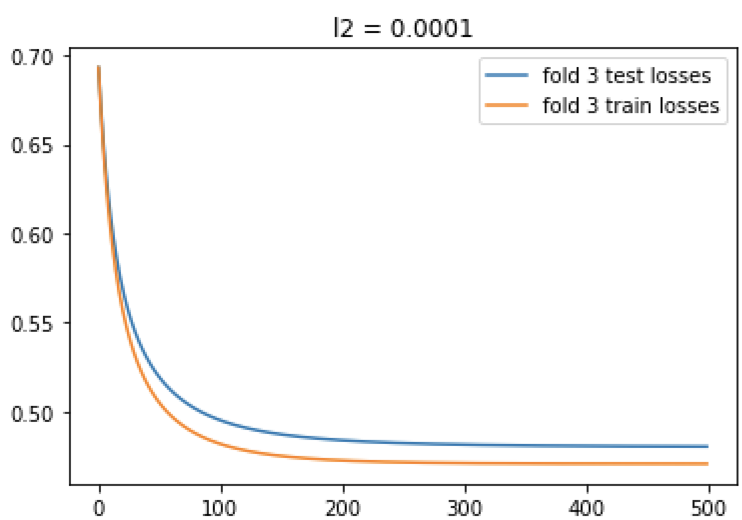
d.

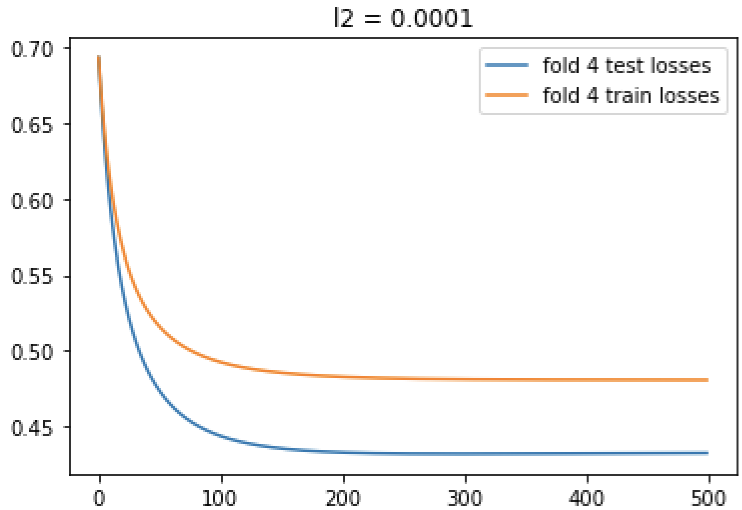
	Fold	Lambda Value	Test Accuracy	Training Accuracy	Test Error	Train Error
0	1	0.0000	71.241830	79.512195	0.182028	0.182028
1	2	0.0000	83.006536	77.073171	0.131169	0.131169
2	3	0.0000	73.202614	77.398374	0.163485	0.163485
3	4	0.0000	79.738562	77.723577	0.146435	0.146435
4	5	0.0000	77.124183	77.886179	0.165088	0.165088
5	1	0.0001	71.241830	79.512195	0.182027	0.182027
6	2	0.0001	83.006536	77.073171	0.131169	0.131169
7	3	0.0001	73.202614	77.398374	0.163485	0.163485
8	4	0.0001	79.738562	77.723577	0.146435	0.146435
9	5	0.0001	77.124183	77.886179	0.165088	0.165088
10	1	0.0006	71.241830	79.512195	0.182027	0.182027
11	2	0.0006	83.006536	77.073171	0.131169	0.131169
12	3	0.0006	73.202614	77.398374	0.163485	0.163485
13	4	0.0006	79.738562	77.723577	0.146435	0.146435
14	5	0.0006	77.124183	77.886179	0.165087	0.165087
15	1	0.0030	71.241830	79.512195	0.182026	0.182026
16	2	0.0030	83.006536	77.073171	0.131169	0.131169
17	3	0.0030	73.202614	77.398374	0.163485	0.163485
18	4	0.0030	79.738562	77.723577	0.146435	0.146435
19	5	0.0030	77.124183	77.886179	0.165087	0.165087
20	1	0.1000	71.241830	79.512195	0.181985	0.181985
21	2	0.1000	83.006536	77.073171	0.131188	0.131188
22	3	0.1000	73.202614	77.398374	0.163492	0.163492
23	4	0.1000	79.738562	77.723577	0.146450	0.146450
24	5	0.1000	77.124183	77.886179	0.165071	0.165071
25	1	0.5000	71.241830	79.512195	0.181820	0.181820
26	2	0.5000	83.006536	77.073171	0.131268	0.131268
27	3	0.5000	73.202614	77.398374	0.163520	0.163520
28	4	0.5000	79.738562	77.723577	0.146511	0.146511
29	5	0.5000	76.470588	77.886179	0.165009	0.165009
30	1	100.0000	71.241830	77.886179	0.181473	0.181473
31	2	100.0000	78.431373	75.772358	0.152838	0.152838
32	3	100.0000	73.856209	76.585366	0.175075	0.175075
33	4	100.0000	75.816993	75.934959	0.163922	0.163922
34	5	100.0000	77.124183	76.910569	0.170211	0.170211
35	1	300.0000	68.627451	72.357724	0.194917	0.194917
36	2	300.0000	72.549020	71.382114	0.175495	0.175495
37	3	300.0000	70.588235	73.008130	0.188751	0.188751
38	4	300.0000	71.895425	71.219512	0.184289	0.184289
39	5	300.0000	75.163399	72.682927	0.183598	0.183598
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

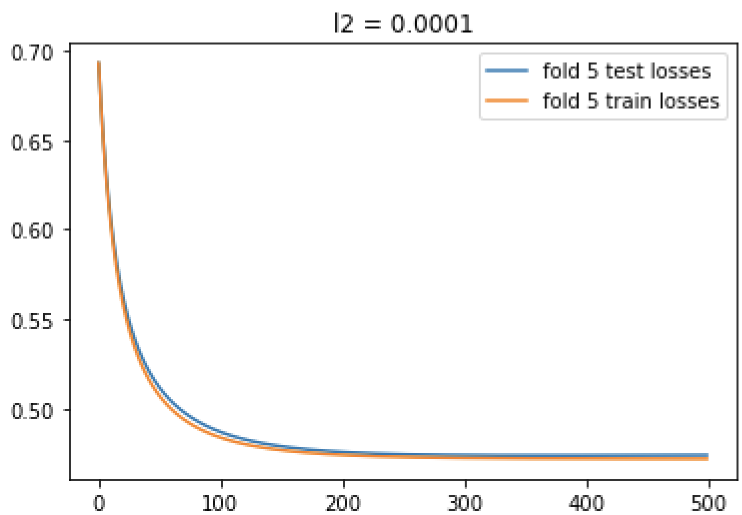
Loss curves at optimal lambda:



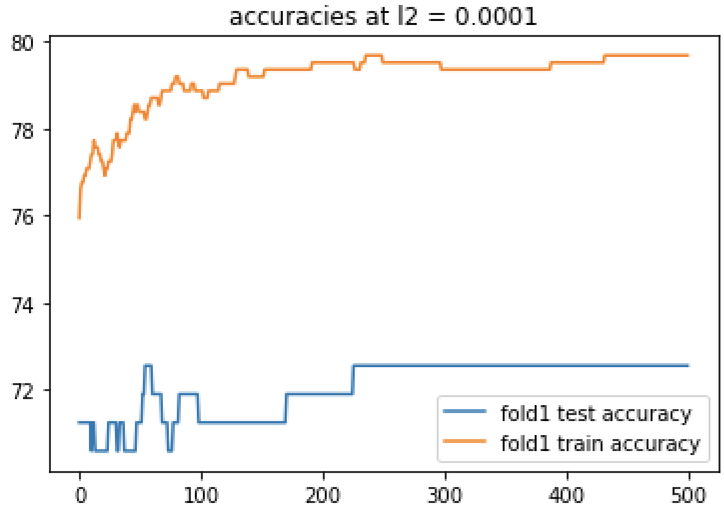


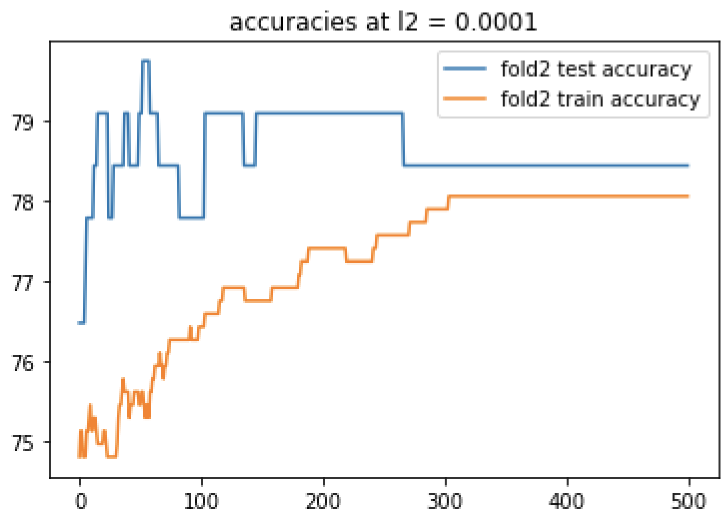


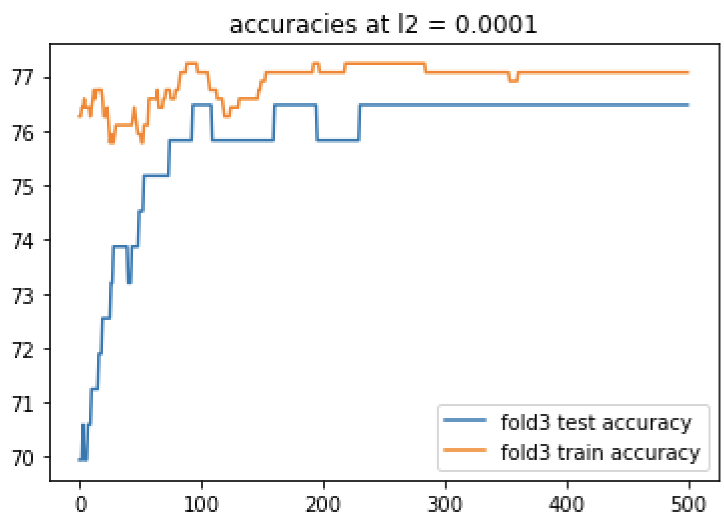

l2 = 0.0001
fold 4 test losses
fold 4 train losses
0.70
0.65
0.60
0.55
0.50
0.45
0
100
200
300
400
500

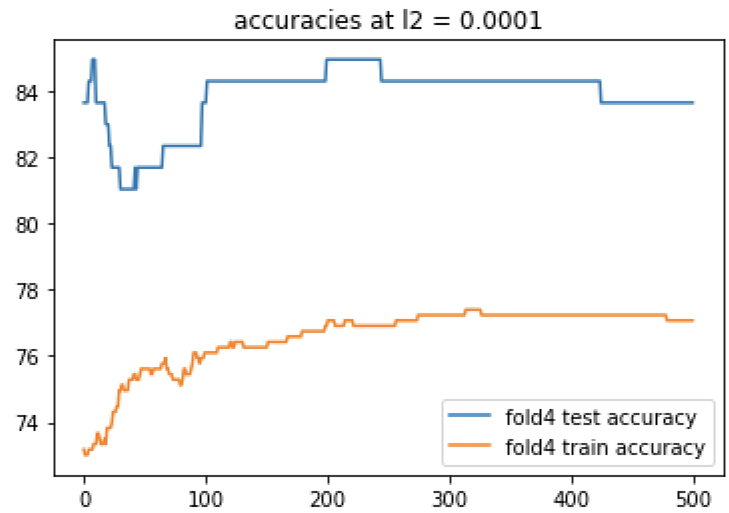

l2 = 0.0001
fold 5 test losses
fold 5 train losses
0.70
0.65
0.60
0.55
0.50
0
100
200
300
400
500

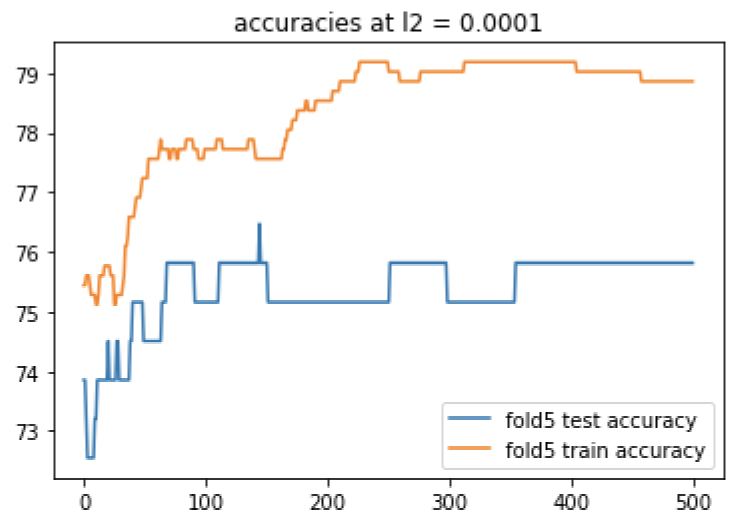
Accuracy Plots at optimal lambda:











No major differences were found as our model is not overfitted.

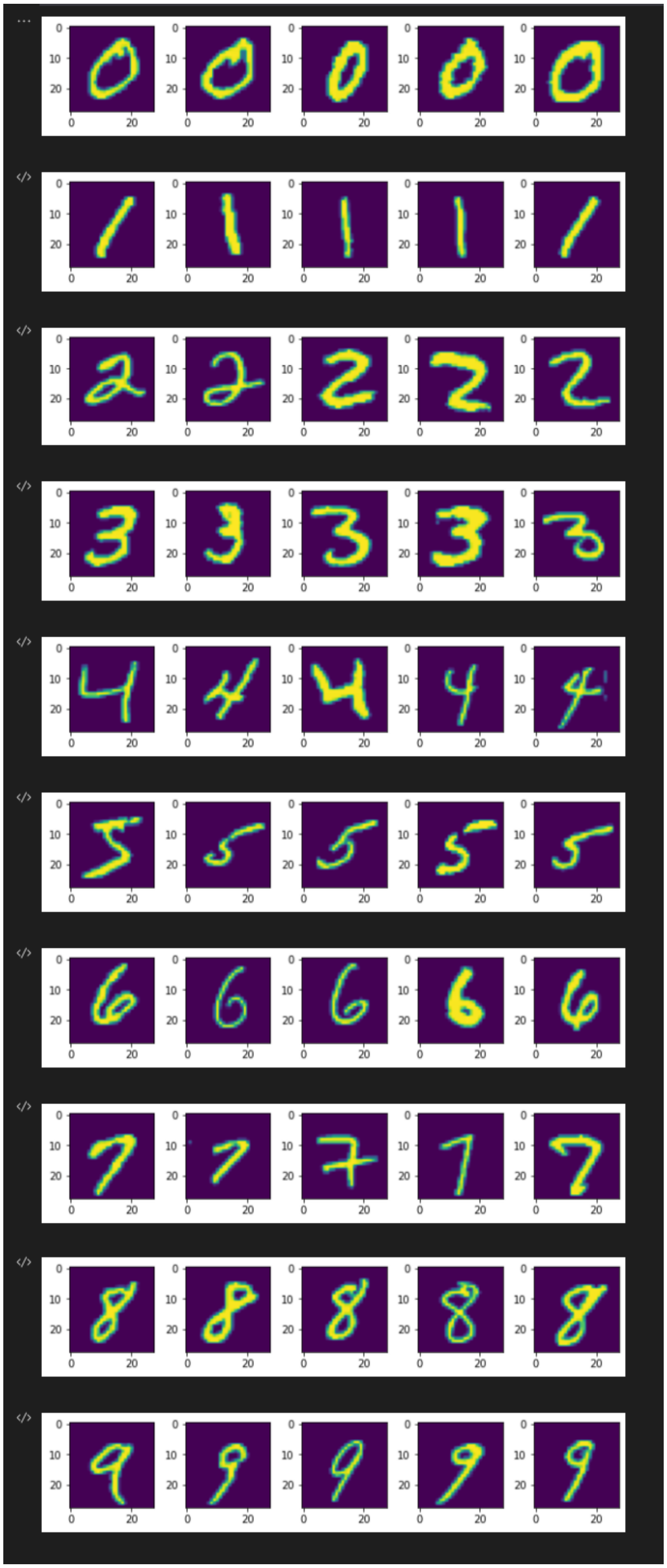
	Fold	Lambda Value	Test Accuracy	Training Accuracy	Test Error	Train Error
0	1	0.0001	71.241830	79.512195	0.182027	0.182027
1	2	0.0001	83.006536	77.073171	0.131169	0.131169
2	3	0.0001	73.202614	77.398374	0.163485	0.163485
3	4	0.0001	79.738562	77.723577	0.146435	0.146435
4	5	0.0001	77.124183	77.886179	0.165088	0.165088

(e) performance measure using sklearn in five folds.

	Fold	Test Accuracy	Training Accuracy	Test Error	Train Error
0	1	75.816993	78.211382	0.163475	0.150286
1	2	73.856209	78.699187	0.189051	0.146820
2	3	77.124183	78.536585	0.158673	0.152007
3	4	83.660131	75.934959	0.136416	0.159859
4	5	76.470588	78.699187	0.156603	0.152208

No major performances differences were spotted.

Q3. a



B. Performance table for ONE vs ONE

	class 1	class 2	Test Accuracy	Train Accuracy
0	0	1	99.684244	99.789429
1	0	2	98.249748	98.574635
2	0	3	98.971466	98.949115
3	0	4	99.422162	99.365295
4	0	5	98.060649	98.107663
5	0	6	98.345154	98.783784
6	0	7	99.507713	99.507713
7	0	8	98.607337	98.527746
8	0	9	99.258760	99.101527
9	1	2	97.732283	98.372703
10	1	3	98.570985	98.363373
11	1	4	99.300699	99.279508
12	1	5	98.980598	99.122999
13	1	6	99.431280	99.336493
14	1	7	98.831488	98.810866
15	1	8	96.602096	96.939856
16	1	9	98.739363	99.243539
17	2	3	96.625868	96.988749
18	2	4	98.677966	98.180791
19	2	5	97.574692	97.773611
20	2	6	97.777029	97.777029
21	2	7	97.807592	98.265518
22	2	8	96.207247	96.906052
23	2	9	98.387639	98.454647
24	3	4	98.997996	98.997661
25	3	5	94.667590	95.440905
26	3	6	98.705609	99.203187
27	3	7	98.322039	98.063892
28	3	8	95.894526	96.238593
29	3	9	97.781457	97.262693
30	4	5	99.041193	98.792471
31	4	6	98.673469	99.036281
32	4	7	98.116947	98.381057
33	4	8	98.803010	98.597332
34	4	9	95.997286	95.623657
35	5	6	97.636684	97.753998
36	5	7	99.075975	99.167047
37	5	8	95.280341	95.564230
38	5	9	98.206120	98.264337
39	6	7	99.934340	99.748276
40	6	8	98.844716	98.685701
41	6	9	99.662959	99.764045
42	7	8	98.580390	98.921536
43	7	9	94.499018	95.447598
44	8	9	97.491525	97.209040

Class wise Test Accuracies for One Vs One:

0	0.986735
1	0.993833
2	0.812984
3	0.856436
4	0.925662
5	0.676009
6	0.948852
7	0.922179
8	0.656057
9	0.777998

Class wise Train Accuracies for One Vs One:

0	0.982948
1	0.992139
2	0.830816
3	0.839667
4	0.909620
5	0.662977
6	0.955221
7	0.939665
8	0.616476
9	0.765003

C.

OVR performance table

	selected class	Test Accuracy	Train Accuracy	T
0	0	92.302857	92.120000	[
1	1	91.560000	91.539048	[
2	2	93.057143	93.053333	
3	3	90.400000	90.537143	
4	4	91.085714	91.009524	
5	5	93.514286	93.384762	
6	6	93.742857	93.689524	
7	7	92.617143	92.365714	
8	8	90.965714	90.643810	
9	9	90.257143	90.396190	

Classwise Test Accuracies in OVR:

	0
0	0.960204
1	0.977974
2	0.801357
3	0.855446
4	0.896130
5	0.687220
6	0.912317
7	0.880350
8	0.736140
9	0.737364

Classwise Train Accuracies in OVR:

0	0.962012
1	0.972412
2	0.812689
3	0.832654
4	0.895413
5	0.716104
6	0.923285
7	0.885874
8	0.739703
9	0.763994

D.

Comparison to sklearn's OVO and OVR classifiers with logistic regression estimator.

	Estimator	Test Accuracy	Train Accuracy
0	OneVsOne_implemented	85.970000	85.335000
1	OneVsRest_implemented	86.010000	85.335000
2	OneVsOne_sklearn	92.616162	93.325871
3	OneVsRest_sklearn	90.560606	91.231343

There are some performance differences between the implemented versions and the sklearn's version. Sklearn's version is clearly more efficient.