SVM Classifier

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	kernel	C	gamma		
1	linear	1	Default	7.06	20.054
2	linear	10	Default	7.33	183.115
3	linear	100	Default	7.64	2118.856
4	rbf	1	0.0001	6.34	9.671
5	rbf	1	0.0010	3.42	8.685
6	rbf	1	0.0100	14.23	25.757
7	rbf	5	0.0001	5.02	8.073
8	rbf	5	0.0010	2.72	8.424
9	rbf	5	0.0100	13.31	27.123
10	rbf	10	0.0001	4.74	7.82
11	**rbf**	**10**	**0.0010**	**2.68**	**8.046**
12	rbf	10	0.0100	13.31	27.092
13	sigmoid	1	0.0001	7.01	818.726
14	sigmoid	1	0.0010	8.65	393.926
15	sigmoid	1	0.0100	43.67	821.123
16	sigmoid	5	0.0001	5.91	525.568
17	sigmoid	5	0.0010	10.63	228.977
18	sigmoid	5	0.0100	43.41	899.302
19	sigmoid	10	0.0001	5.61	342.351
20	sigmoid	10	0.0010	11.17	235.85
21	sigmoid	10	0.0100	43.47	789.173

Used the “sklearn.svm.SVC” classifier.
For the linear kernel applied using the “sklearn.svm.LinearSVC” with regularization hyper-parameter “C” and values of [1, 10, 100]
Used the hyper-parameters – kernel, C, gamma.
Hyper-parameters:
kernel: Specifies the kernel type to be used in the algorithm. (rbf, sigmoid, poly)
gamma: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.
C: Regularization parameter. Must be strictly positive.

Proper choice of C and gamma is critical to the SVM’s performance. The values of Gamma and C should not be very high (causes overfitting) and should not be very small (causes underfitting).
C trades off misclassification of training examples against simplicity of the decision surface. A high C aims at model choosing more data points and get higher variance and lower bias. A low C, makes the model chooses less data points and get less variance and higher bias.
Gamma defines influence of training examples. The larger gamma is, the closer other examples must be to be affected.
Best error rate obtained for the set the hyper-parameters chosen is below.

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	kernel	C	gamma		
11	**rbf**	**10**	**0.0010**	**2.68**	**8.046**

MLP Classifier

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	hidden_layer_sizes	activation	solver		
1	(300,)	tanh	sgd	3.47	551.679
2	(300,)	tanh	adam	2.86	108.31
3	(300,)	relu	sgd	2.56	463.205
4	(300,)	relu	adam	2.77	95.349
5	(200, 200)	tanh	sgd	3.52	603.176
6	(200, 200)	tanh	adam	2.73	104.326
7	(200, 200)	relu	sgd	2.69	414.932
8	**(200, 200)**	**relu**	**adam**	**2.00**	**86.073**
9	(200, 100, 50)	tanh	sgd	3.51	435.265
10	(200, 100, 50)	tanh	adam	2.88	86.132
11	(200, 100, 50)	relu	sgd	2.90	251.742
12	(200, 100, 50)	relu	adam	2.19	108.129

MLP - Multi-layer Perceptron.

Used the “sklearn.neural_network.MLPClassifier” neural network classifier.
MLPClassifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

Used the hyper-parameters – hidden_layer_sizes, activation and solver.
Hyper-parameters:

hidden_layer_sizes: A tuple of values. The tuple length defines the number of hidden layers where as the values in the tuple defines the number of hidden units in that layer.

activation: Activation function for the hidden layer.
‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
‘relu’, the rectified linear unit function, returns f(x) = max (0, x)

solver: Solver of the weight optimization.
‘sgd’ refers to stochastic gradient descent.
‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

As per the documentation, the default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score.

The best error rate obtained from the set of hyper-parameters chosen is below.

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	hidden_layer_sizes	activation	solver		
8	**(200, 200)**	**relu**	**adam**	**2.00**	**86.073**

KNN Classifier

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	n_neighbors	metric	weights		
1	3	euclidean	uniform	5.50	4.001
2	3	euclidean	distance	5.34	2.923
3	**3**	**manhattan**	**uniform**	**4.09**	**2654.114**
4	3	manhattan	distance	4.11	1582.564
5	5	euclidean	uniform	5.59	1.875
6	5	euclidean	distance	5.50	1.809
7	5	manhattan	uniform	4.27	1526.046
8	5	manhattan	distance	4.13	1479.739
9	9	euclidean	uniform	5.72	1.895
10	9	euclidean	distance	5.60	1.889
11	9	manhattan	uniform	4.53	1482.351
12	9	manhattan	distance	4.47	1481.343

Used the "sklearn.neighbors.KNeighborsClassifier" k-nearest neighbor classifier.

Used the hyper-parameters – n_neighbors, metric and weights.
Hyper-parameters:
n_neighbours: Number of neighbors to use by default for k-neighbors queries.
metric: The distance metric to use for the tree.

'euclidean': `sqrt(sum((x-y)^2))`

'manhattan': `sum(|x-y|)`

weights: Weight function used in prediction.

'uniform': uniform weights. All points in each neighborhood are weighted equally.

'distance': weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away

Best error rate results obtained from the set of hyper-parameters chosen is below.

Index	Hyper-Parameters			Error Rate (%)	Run Time (Sec)
	n_neighbors	metric	weights		
3	**3**	**manhattan**	**uniform**	**4.09**	**2654.114**