Sakshi Udeshi 1003197
Lu Wei
18 October 2017

Machine Learning Assignment 2

SVM's and Logistic Regression

Linear SVM with $\lambda = 1$ (Q1)

I implemented a linear SVM assuming the data is slightly linearly inseparable for the primal and the dual forms. Below are my results

Linear SVM

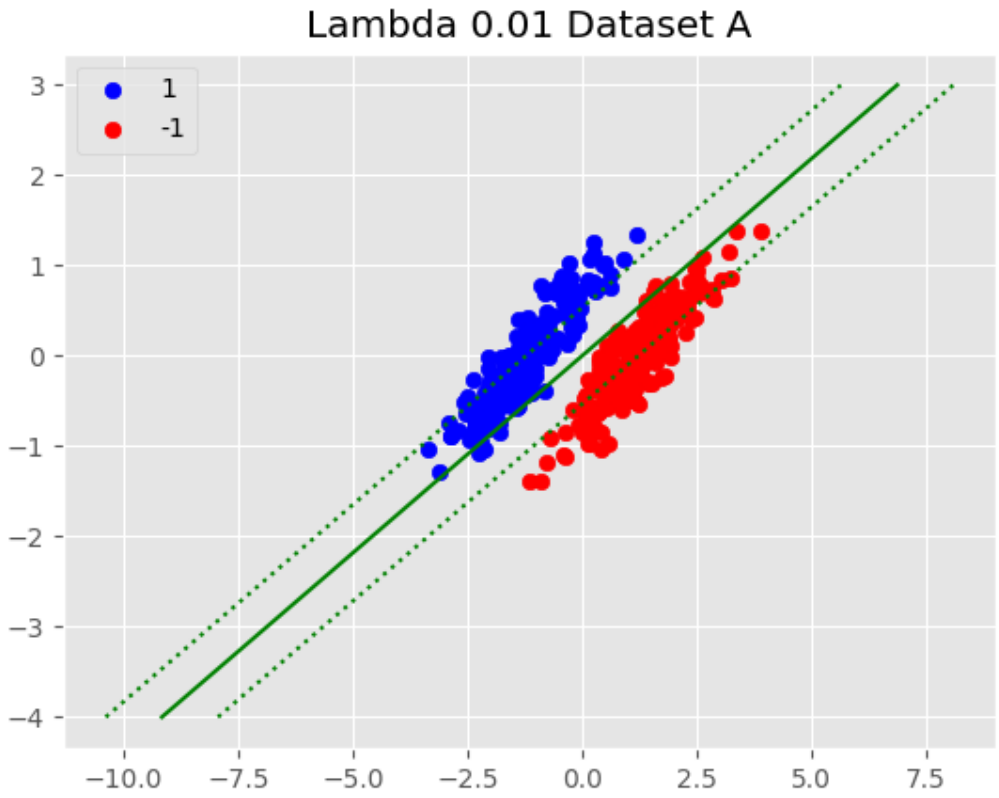
	A	B	C
Primal form accuracy	99.625%	89.125%	81.75%
Primal form θ	[-1.6858861147661965, 2.1827859871827324]	[-0.34193883889544385, 1.51121894904954]	[-0.00441286024071002, -2.0088805796235283]
Dual form accuracy	99.625%	83.125%	80.375%
Dual form θ	[-1.9434663601647042, 2.5783461656184423]	[-0.84091143452701445, 1.3896707451600212]	[-0.11933243849778954, -2.5415402272135763]

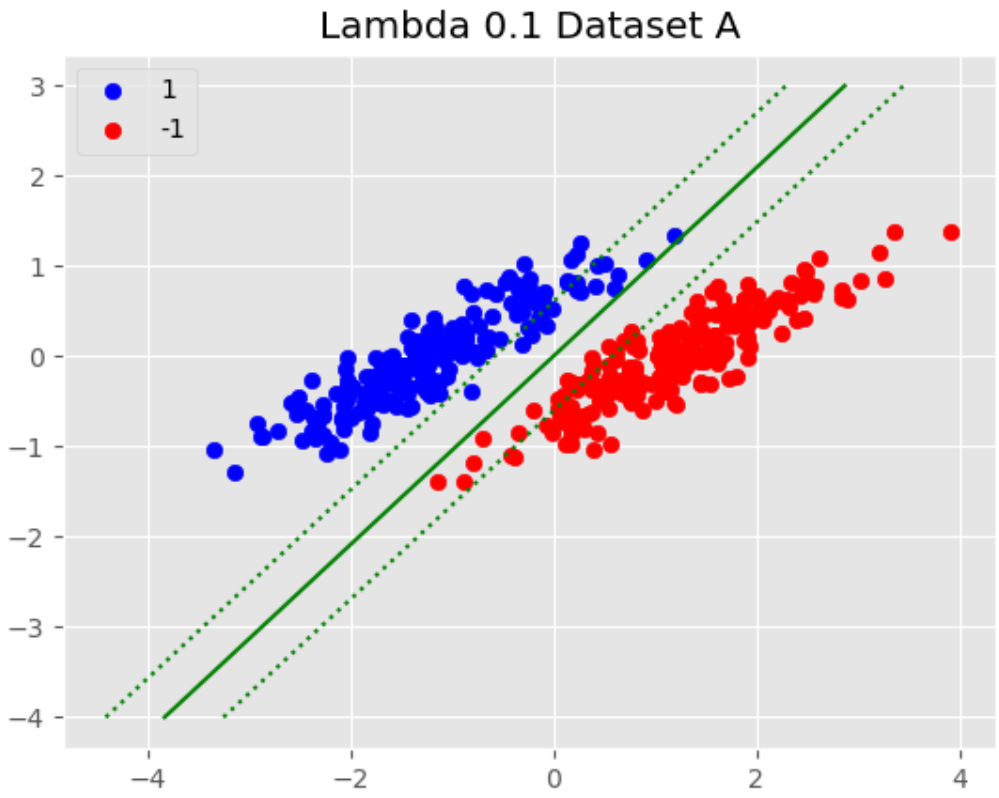
Modifying λ (Q2)

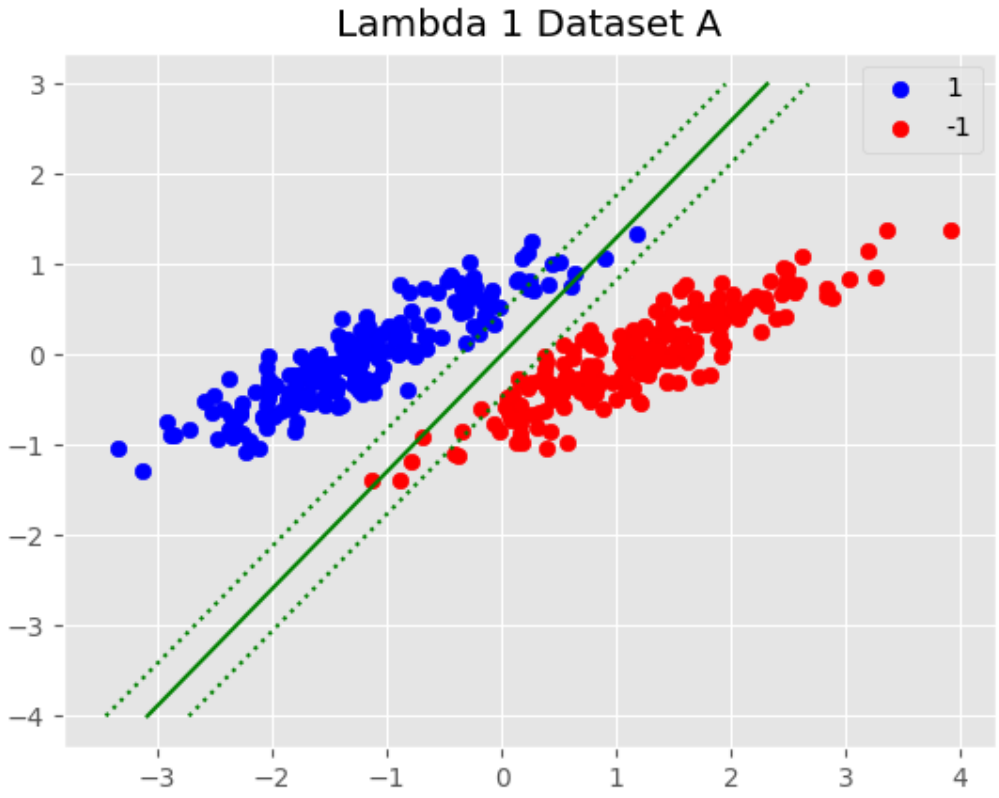
In my code, I have modified λ, which is essentially the same as modifying C.

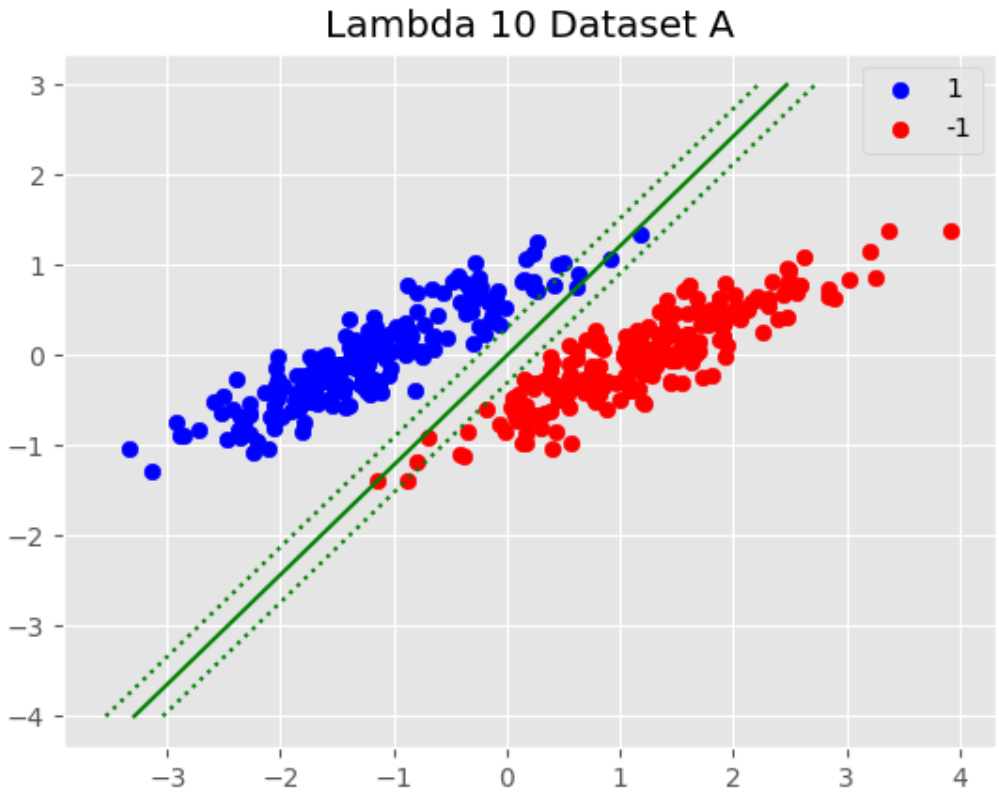
The C parameter tells the SVM optimisation how much you want to avoid misclassifying each training example. For large values of C, the optimisation will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimiser to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points

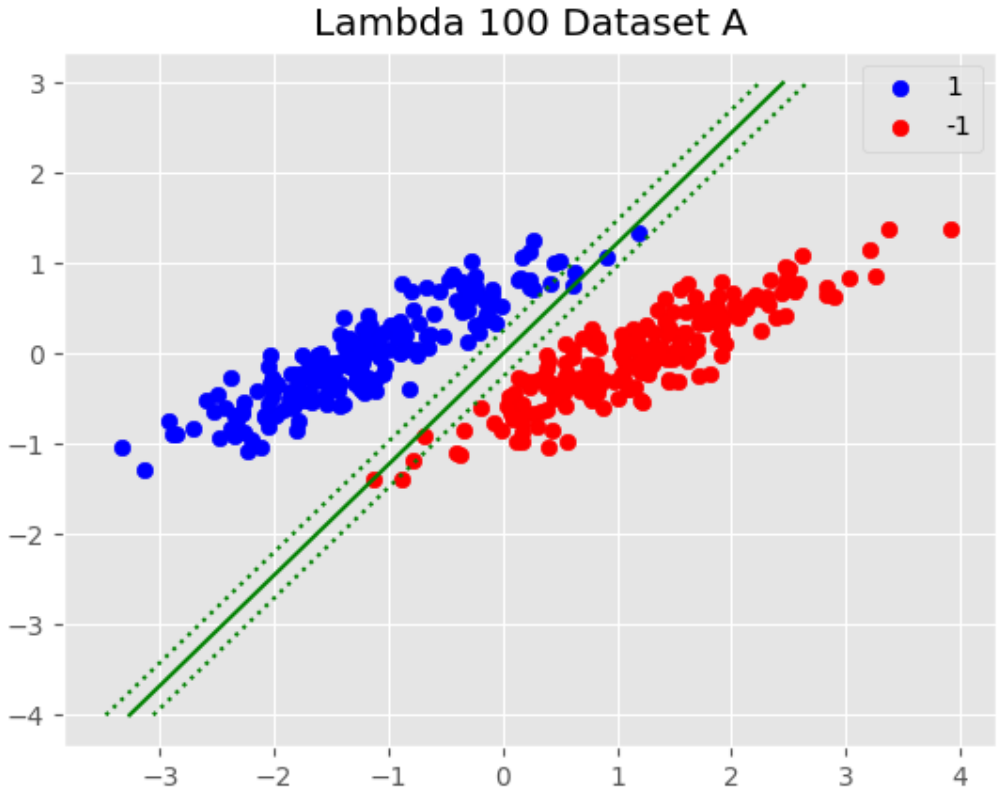
The number of support vectors decreases as C increases.




Lambda 0.1 Dataset A
1
-1
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4


Lambda 1 Dataset A
1
-1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4


Lambda 10 Dataset A
1
-1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4


Lambda 100 Dataset A
1
-1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

Optimal C (Q3)

The most obvious way to optimise C really depends on the data. If we are using a kernel, the most common way is grid search, where we jointly optimise the parameters of the kernel and the C. For linear SVM's we can just search run a search given a range and step size for C and pick the C which gives the highest accuracy on the test set.

Eventually, it will converge to the most optimal C.

For the three datasets, the λ vs accuracy metrics are as follows.

Optimal C

	A	B	C
λ = 0.01	96.5%	88.75%	74.75%
λ = 0.1	99.5%	88.875%	81.0%
λ = 1	99.625%	89.125%	81.75%
λ = 10	99.5%	89.125%	81.875%
λ = 100	99.5%	89.125%	81.875%
λ = 1000	99.5%	89.125%	81.875%

Implementing Kernels (Q4)

I implemented the RBF kernel and a polynomial kernel. Implementing the RBF kernel was simple, but it took a lot of time to figure out optimal σ values.

The performance is as follows.

Kernels

	A	B	C
RBF	92.375%	90.5%	72.125%
Polynomial	52.875%	51.125%	46.375%

Product of Kernels (Q5)

Since k_1, k_2 are valid kernels, we know (via Mercer) that they must admit an inner product representation. Let a denote the feature vector for k_1 and b for k_2.

$$k_1(x,y) = a(x)^T a(y), \qquad a(z) = [a_1(z), a_2(z), \ldots a_M(z)]$$
$$k_2(x,y) = b(x)^T b(y), \qquad b(z) = [b_1(z), b_2(z), \ldots b_N(z)]$$

$$\begin{aligned} k_p(x,y) &= k_1(x,y)k_2(x,y) \\ &= \Big(\sum_{m=1}^{M} a_m(x)a_m(y)\Big)\Big(\sum_{n=1}^{N} b_n(x)b_n(y)\Big) \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N} [a_m(x)b_n(x)][a_m(y)b_n(y)] \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N} c_{mn}(x)c_{mn}(y) \\ &= c(x)^T c(y) \end{aligned}$$

Here, $c(z)$ is a $M \cdot N$ vector, such that $c_{mn}(z) = a_m(z)b_n(z)$

Hence, proved

Logistical Regression (Q6)

I used a learning rate of 0.001 and the algorithm terminates when the absolute difference of between consecutive θ vectors is below 0.0001

The table below represents the results and the final stage cost function for various different λ

Interestingly, when I take a random value of θ, there are instances where I get very high accuracy for low λ and very low accuracy for high λ.

	A		B		C	
λ	**Accuracy**	**Cost**	**Accuracy**	**Cost**	**Accuracy**	**Cost**
0.001	81.875%	-0.037182933090320267	46.25%	0.67080134478101117	67.625%	0.6930951549527 9705
0.01	81.875%	0.0078572242921312908	46.25%	0.6712410995545 8189	67.625%	0.6930951560777 9704
0.1	81.875%	0.2351917777843 7897	46.25%	0.6742802528682 0614	67.625%	0.6930951673277 9708
1	81.875%	0.5680874905591 1145	46.25%	0.6851270044921 1894	67.625%	0.6930952798277 9706
10	81.875%	0.6776371600697 9513	46.25%	0.6919117914536 1814	67.625%	0.6930964048277 97
100	81.875%	0.6914790452282 9462	46.25%	0.6929437967572 2539	67.625%	0.6931076548277 9702
1000	81.875%	0.6930509031414 146212	46.25%	0.6931454227853 3342	67.625%	0.6932201548277 9709

Varying Learning Rates (Q7)

I used $\lambda = 1$ and the algorithm terminates when the absolute difference of between consecutive θ vectors is below 0.0001

The table below represents the results and the final stage cost function for various different learning rate

I get very low accuracy if I use a random theta initialisation for very low learning rates. For higher learning rates, the accuracy converges to 81.875%, 46.25% and 67.625% for datasets A, B and C respectively.

The optimal learning rate seems to be around 0.1

	A		B		C	
Learning Rate	**Accuracy**	**Cost**	**Accuracy**	**Cost**	**Accuracy**	**Cost**
0.001	81.875%	0.674823631 07796972	46.25%	0.693212586 50457784	67.625%	0.693056055 4436872
0.01	81.875%	0.658148399 79515372	46.25%	419.7587612 9494679	67.625%	0.681917610 00155704
0.1	81.875%	56317.29481 5448469	46.25%	inf	67.625%	inf
1	81.875%	inf	46.25%	inf	67.625%	inf
10	81.875%	inf	46.25%	inf	67.625%	inf
100	81.875%	inf	46.25%	inf	67.625%	inf
1000	81.875%	inf	46.25%	inf	67.625%	inf

Changing Cost Function (Q8)

The cost function now seems to reward the misclassification of the points. As a result we get very low accuracy across the 3 datasets.

Since we have flipped the function, the cost will be minimised when the points are misclassified.

	A	B	C
Accuracy	20.875%	14.75%	30.5%

The code for this assignment can be found here