Sahil Gandhi
Programming Assignment - 02

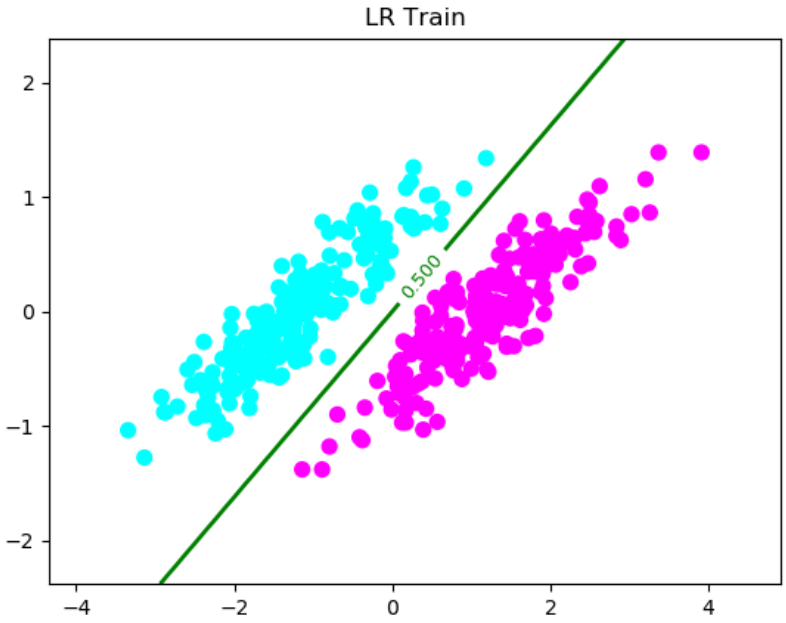
Q1. Logistic Regression

Part b. Test your implementation on the data sets provided

Linearly Separable data

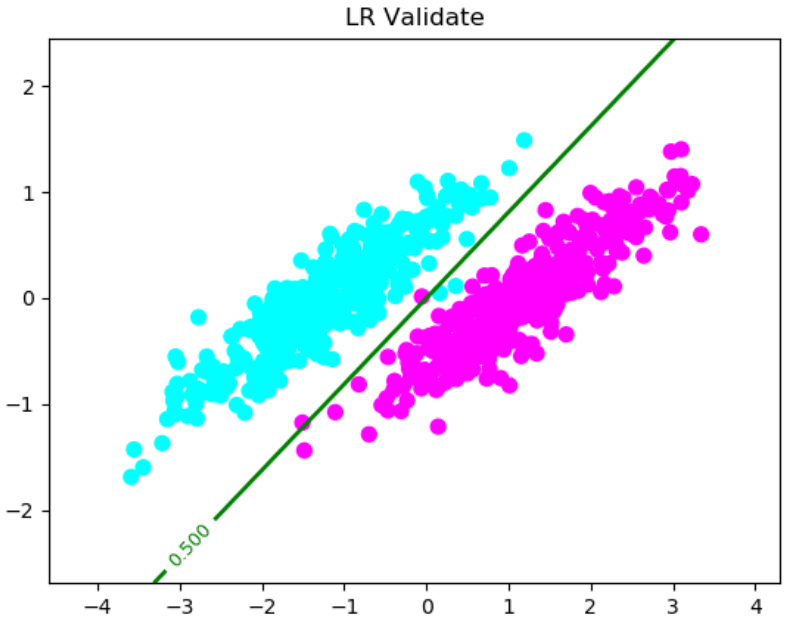
Training data:

- Misclassification on training data = 0



Validation data:

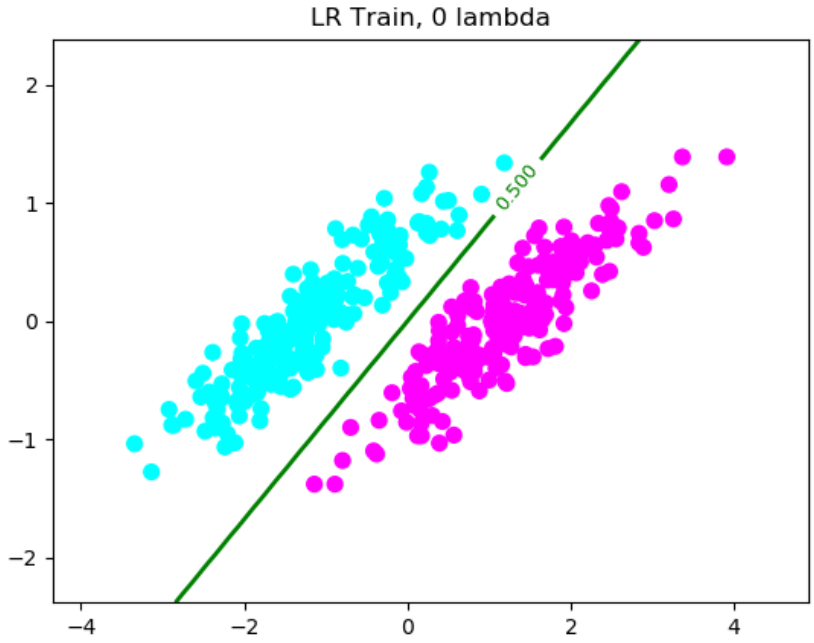
- Misclassification on validation data = 4



$\lambda = 0$

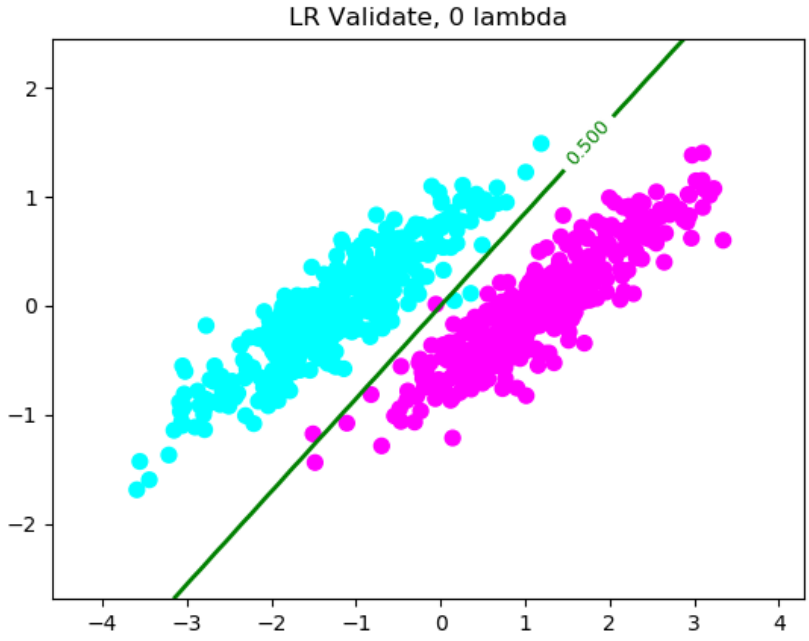
Training data:

- Misclassification on training data = 0



Validation data:

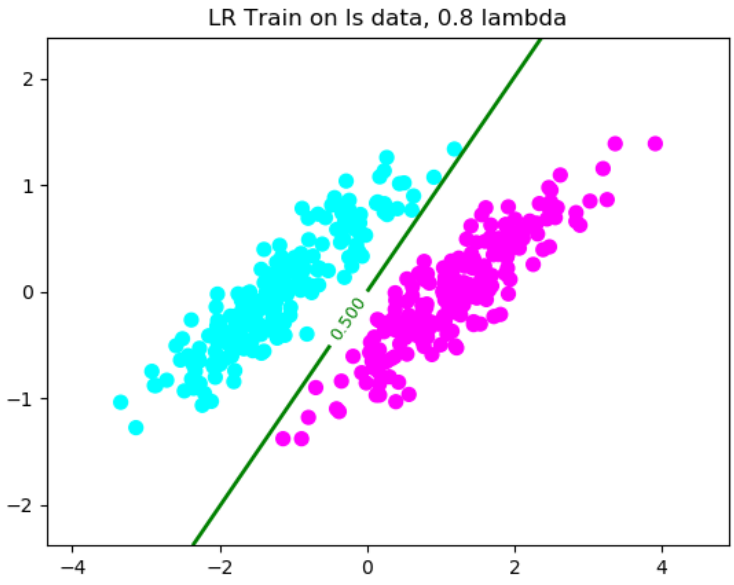
- Misclassification on validation data = 4



$\boldsymbol{\lambda = 0.8}$

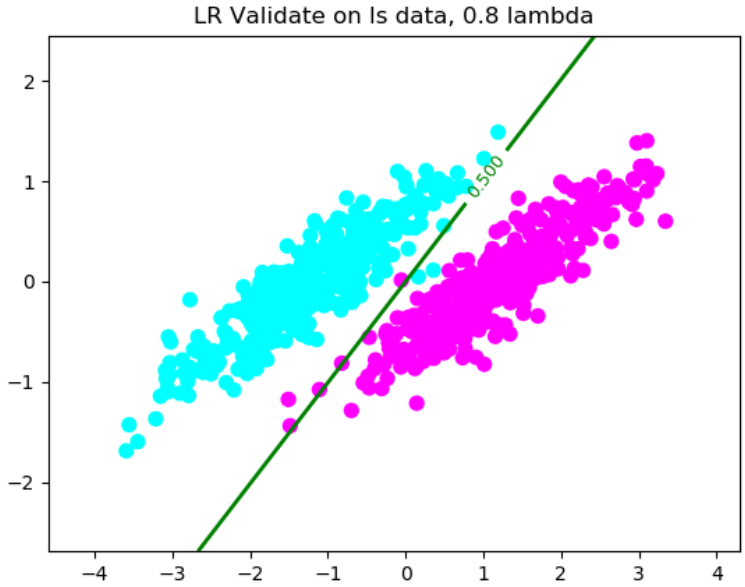
Training data:

- Misclassification on training data = 0



Validation data:

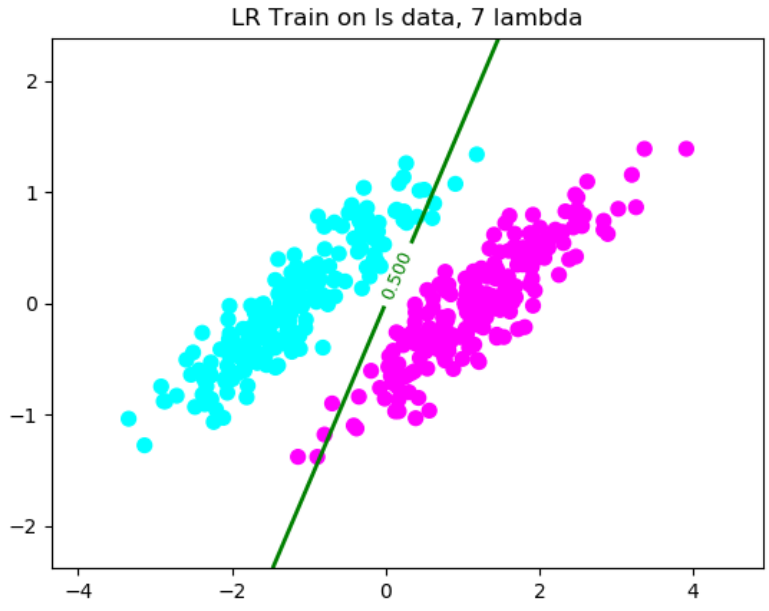
- Misclassification on validation data = 7



$\boldsymbol{\lambda = 7}$

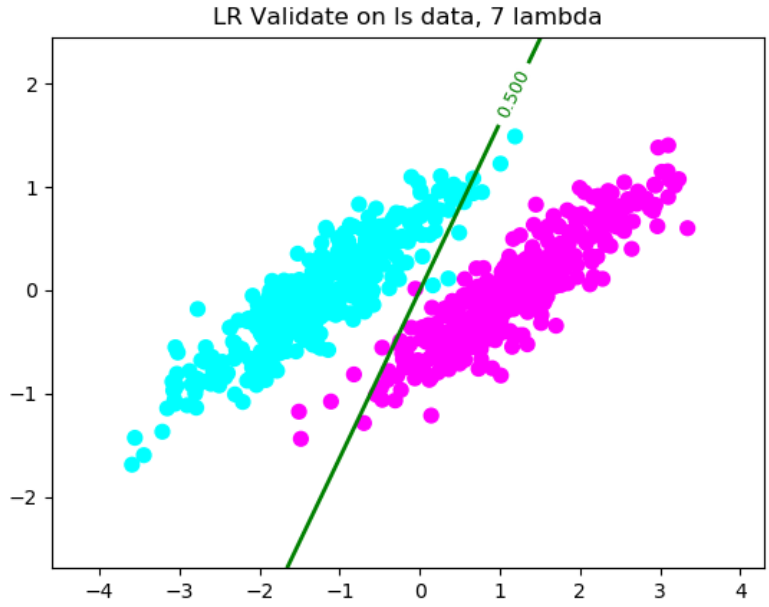
Training data:

- Misclassification on training data = 7

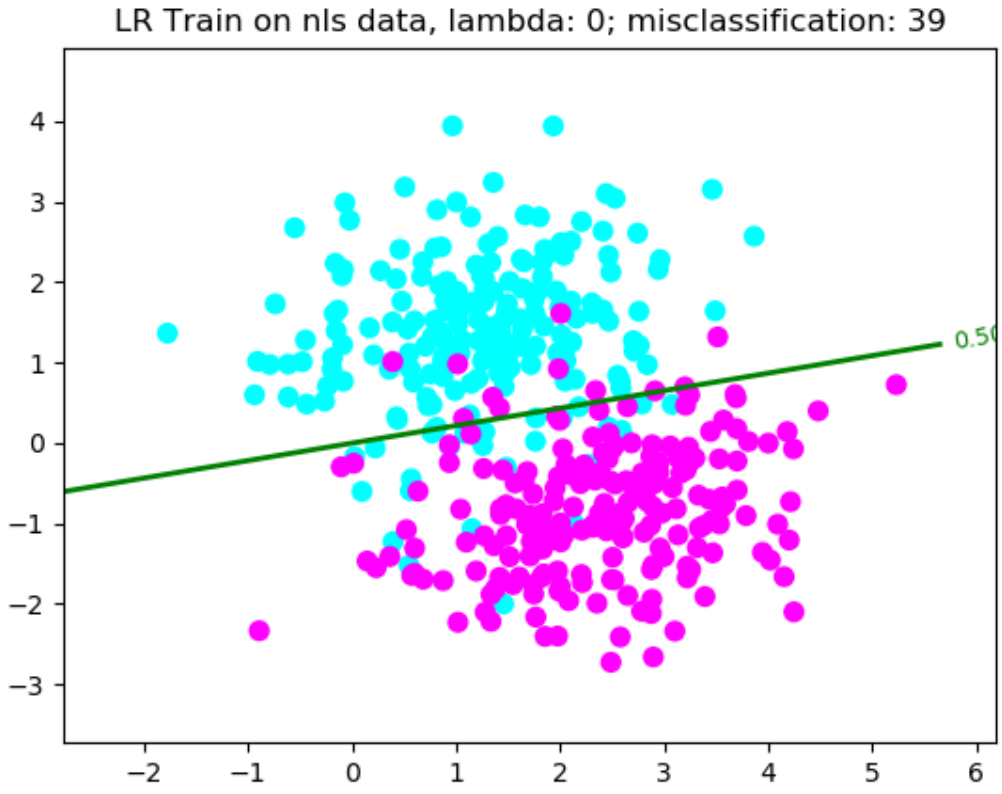


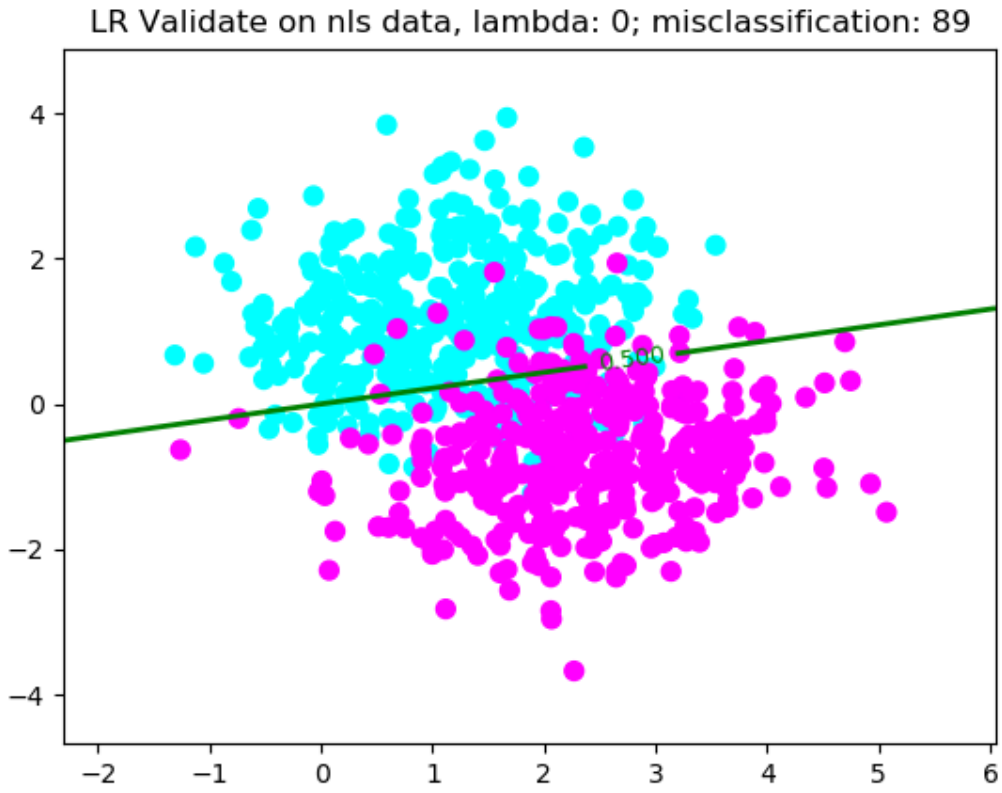
Validation data:

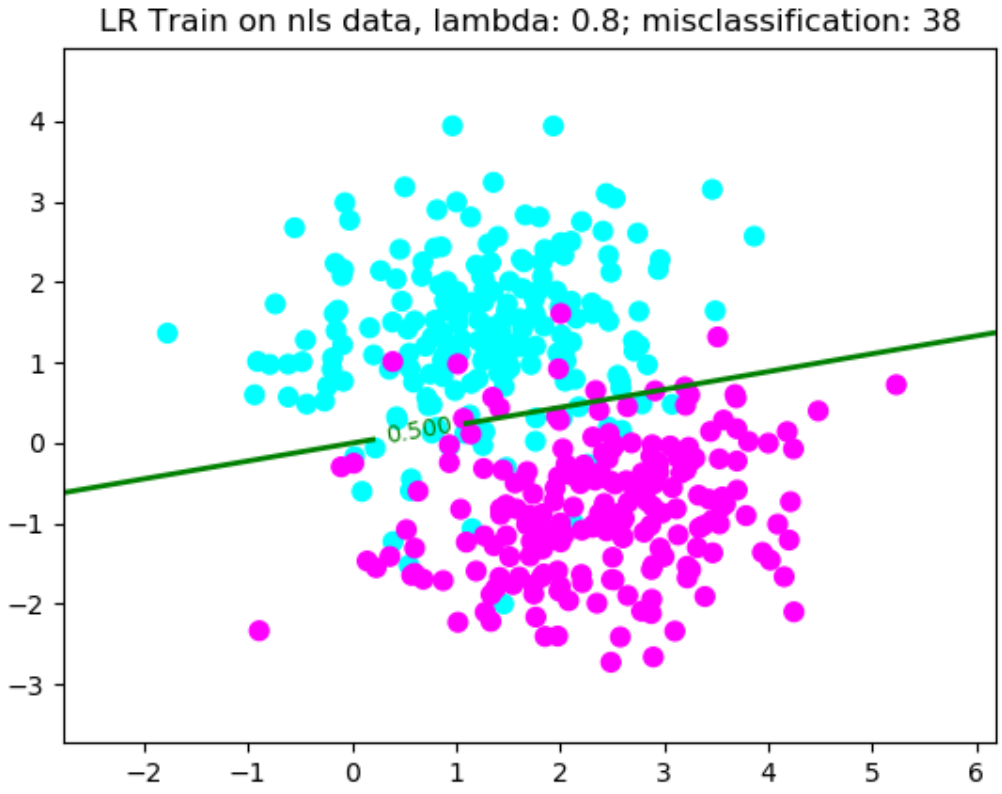
- Misclassification on validation data = 15

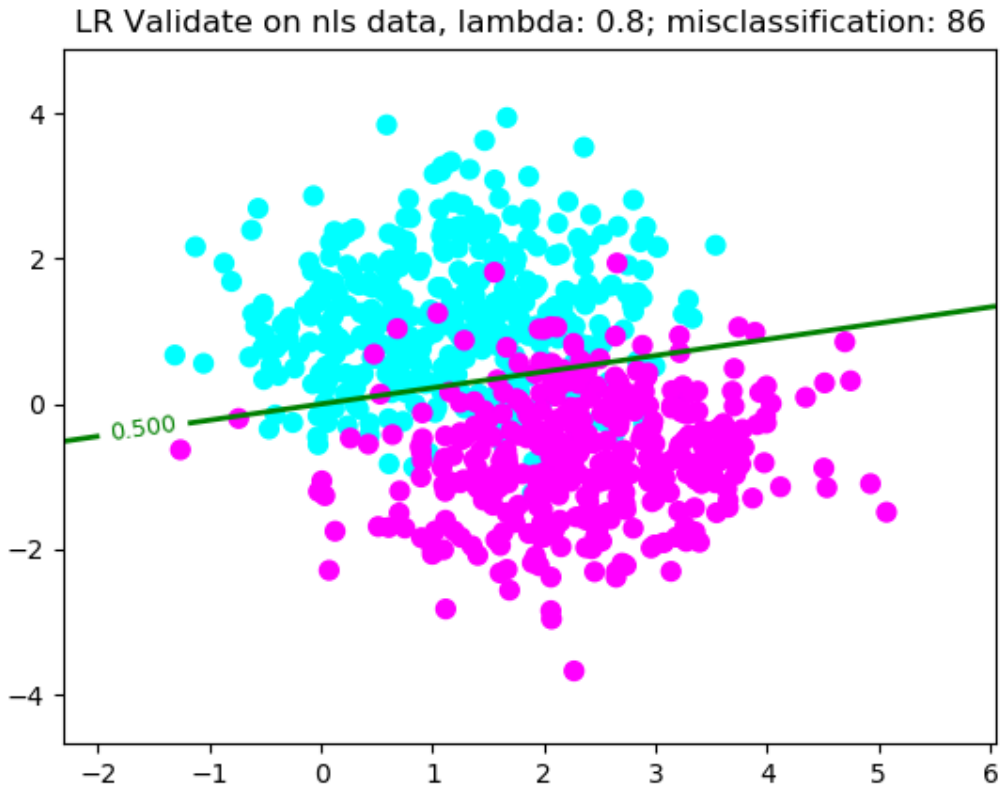


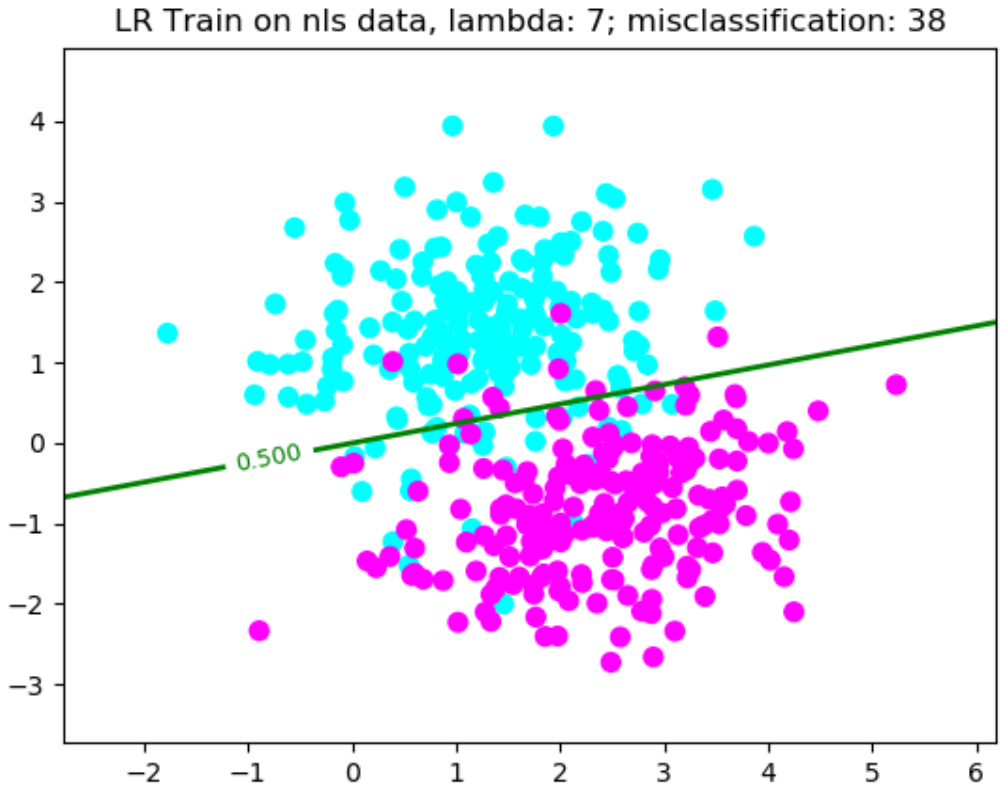
Below are plots by running logistic regression on other data sets with varying lambda. The misclassification count and lambda are in the plot title.

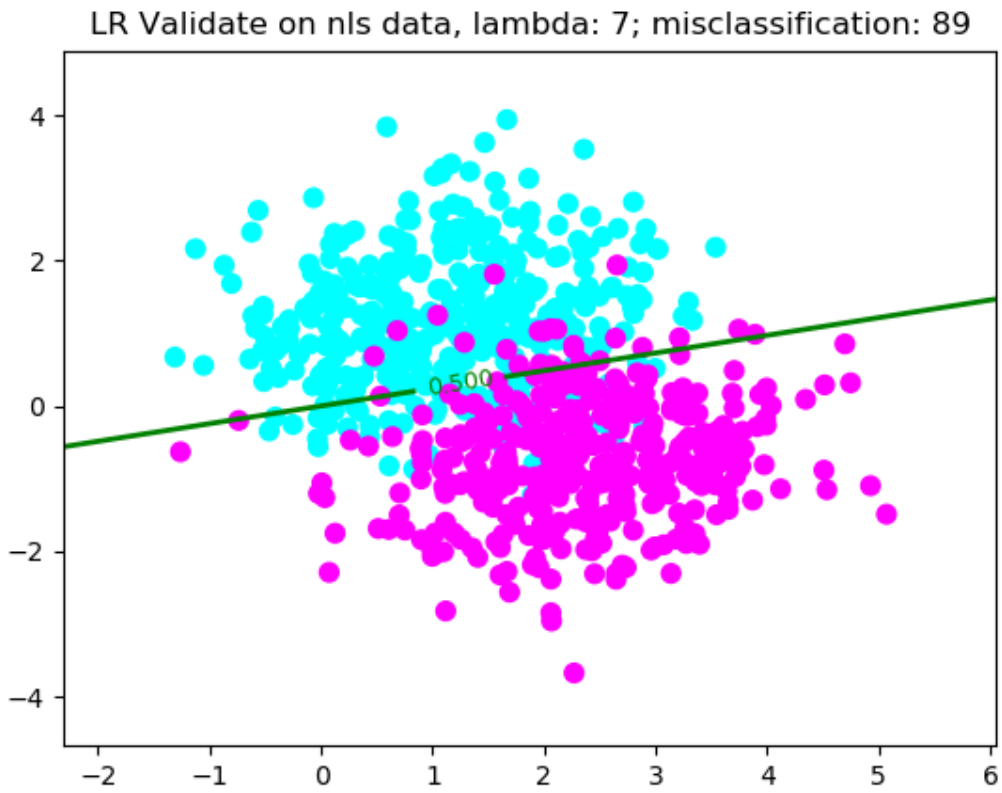

LR Train on nls data, lambda: 0; misclassification: 39
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5
6

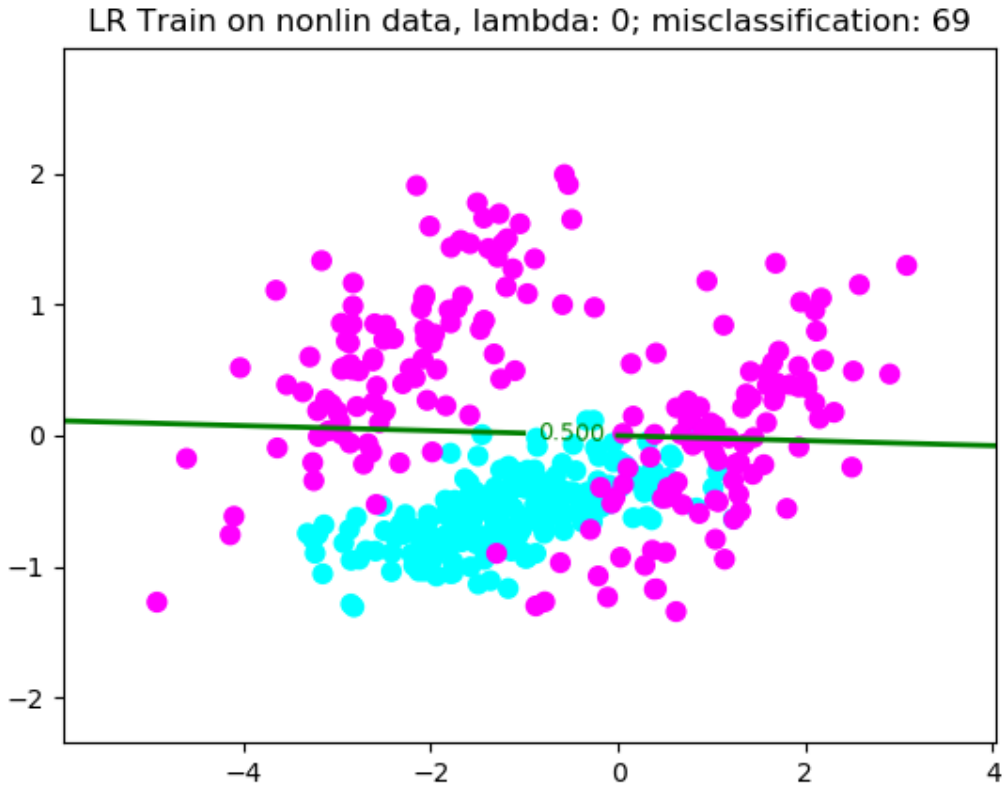

LR Validate on nls data, lambda: 0; misclassification: 89
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5
6

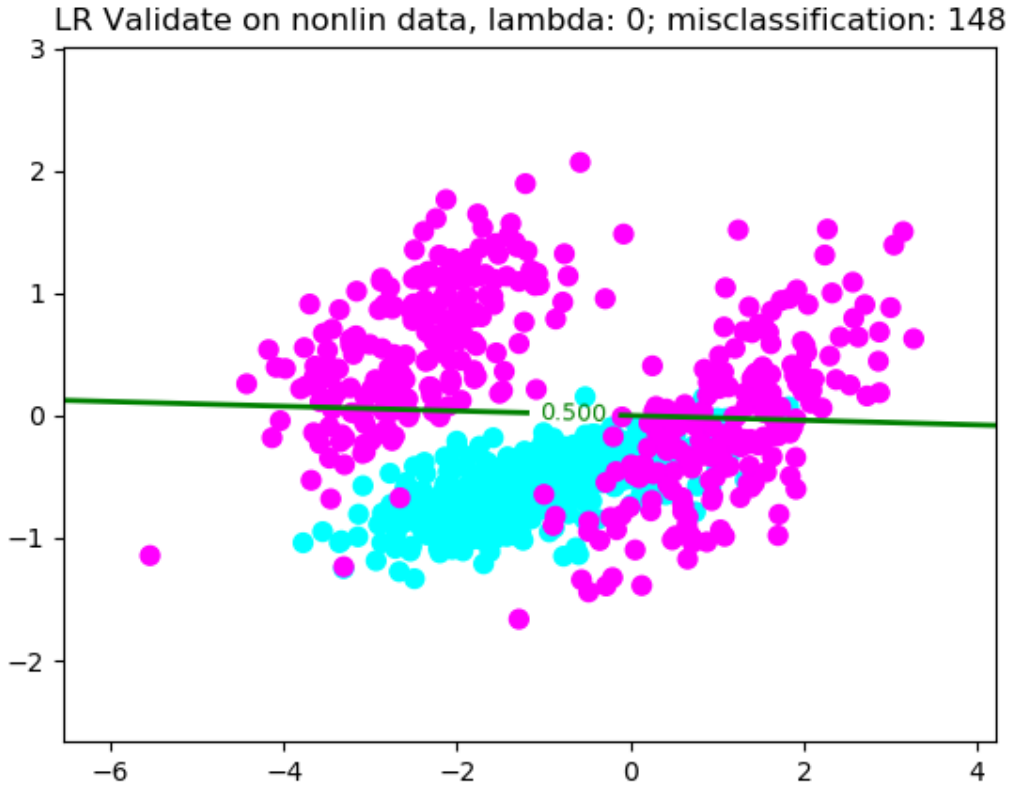

LR Train on nls data, lambda: 0.8; misclassification: 38
0.500

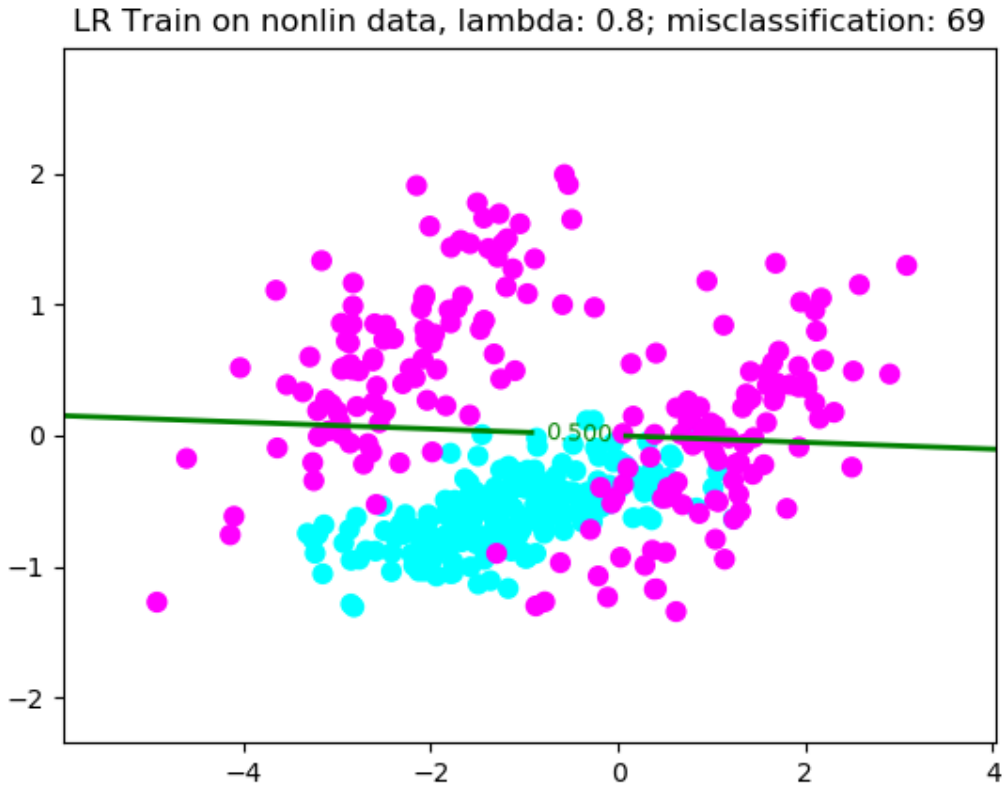

LR Validate on nls data, lambda: 0.8; misclassification: 86
0.500

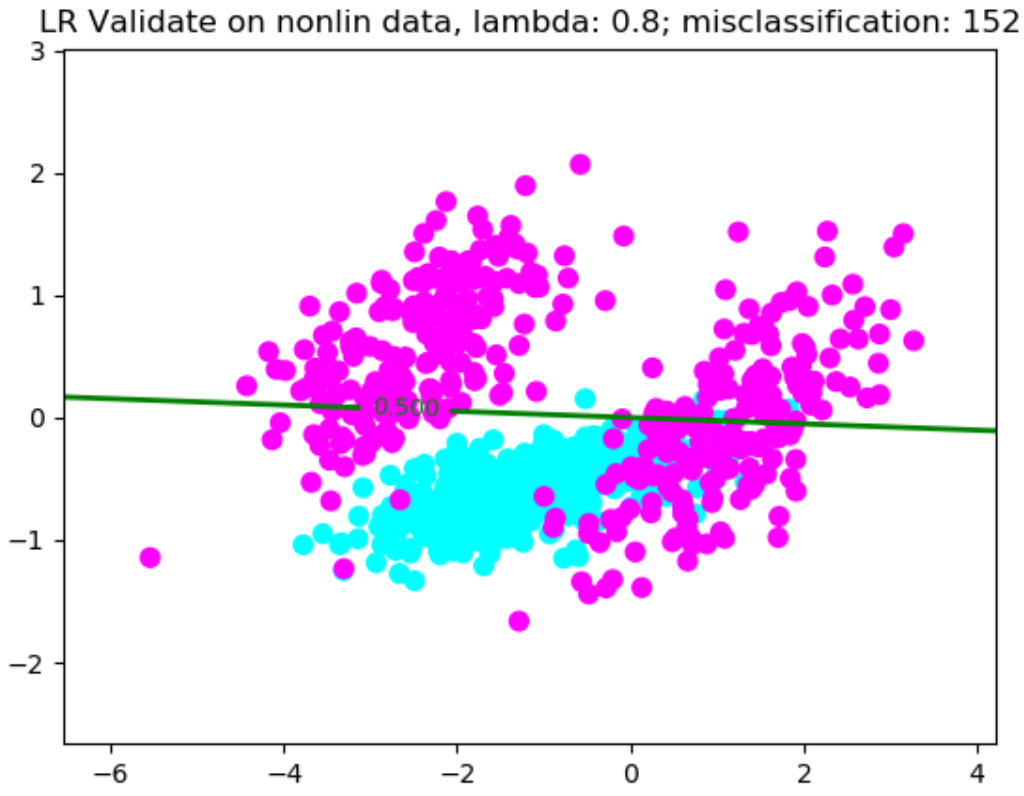

LR Train on nls data, lambda: 7; misclassification: 38
0.500

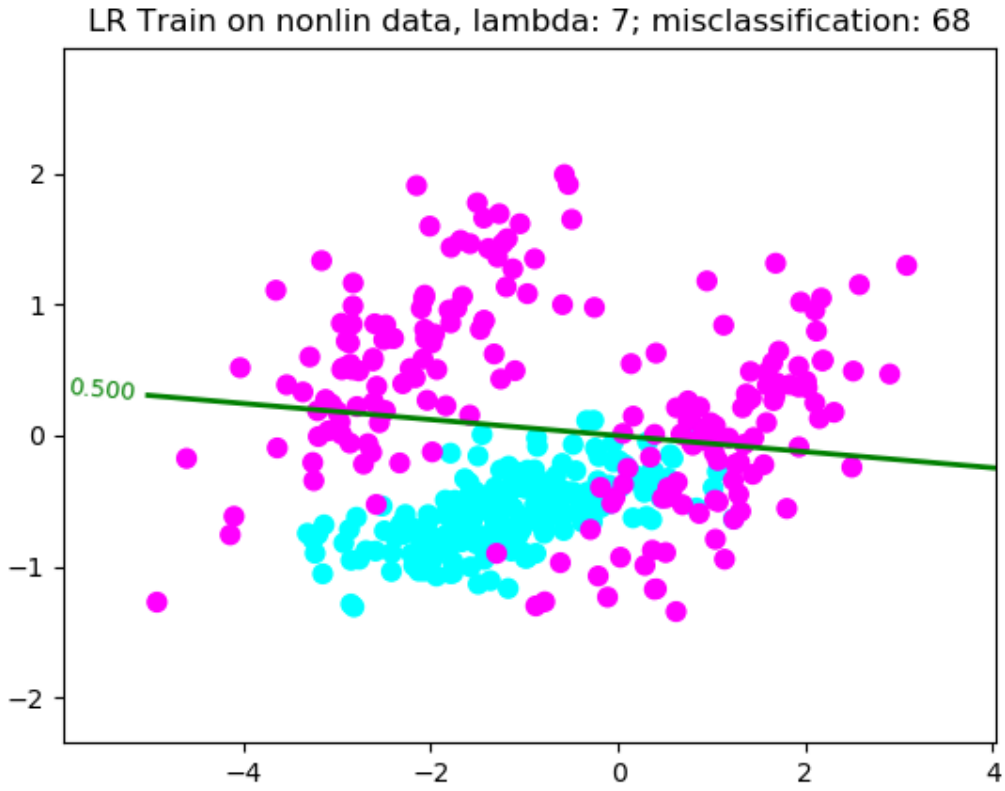

LR Validate on nls data, lambda: 7; misclassification: 89
0.500

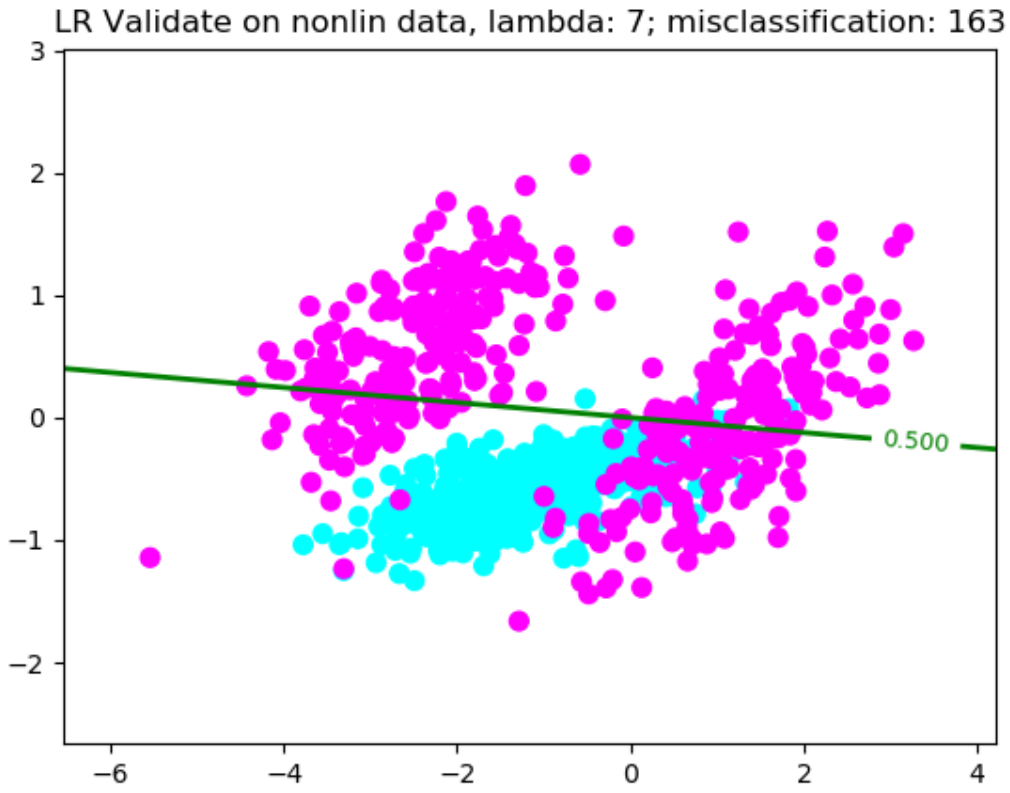

LR Train on nonlin data, lambda: 0; misclassification: 69
0.500
2
1
0
−1
−2
−4
−2
0
2
4


LR Validate on nonlin data, lambda: 0; misclassification: 148
0.500
3
2
1
0
−1
−2
−6
−4
−2
0
2
4


LR Train on nonlin data, lambda: 0.8; misclassification: 69
0.500
2
1
0
−1
−2
−4
−2
0
2
4


LR Validate on nonlin data, lambda: 0.8; misclassification: 152
0.500
3
2
1
0
−1
−2
−6
−4
−2
0
2
4


LR Train on nonlin data, lambda: 7; misclassification: 68
0.500
2
1
0
−1
−2
−4
−2
0
2
4


LR Validate on nonlin data, lambda: 7; misclassification: 163
0.500
3
2
1
0
−1
−2
−6
−4
−2
0
2
4

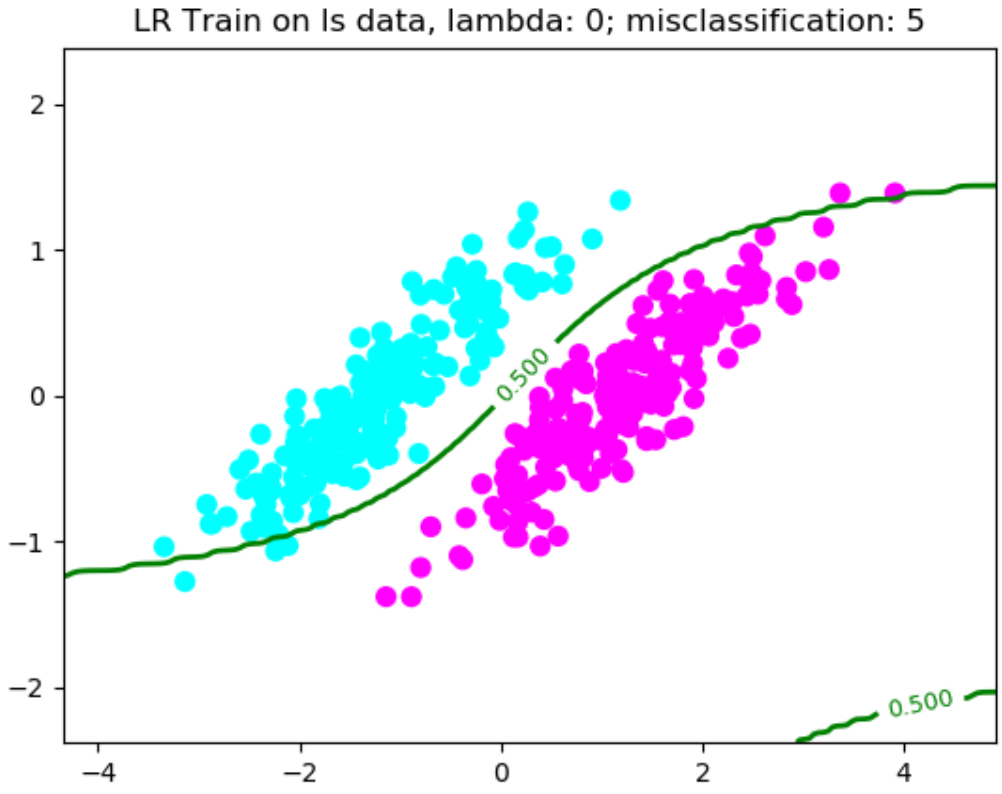
Analysis of all the plots in Q1. Part b.

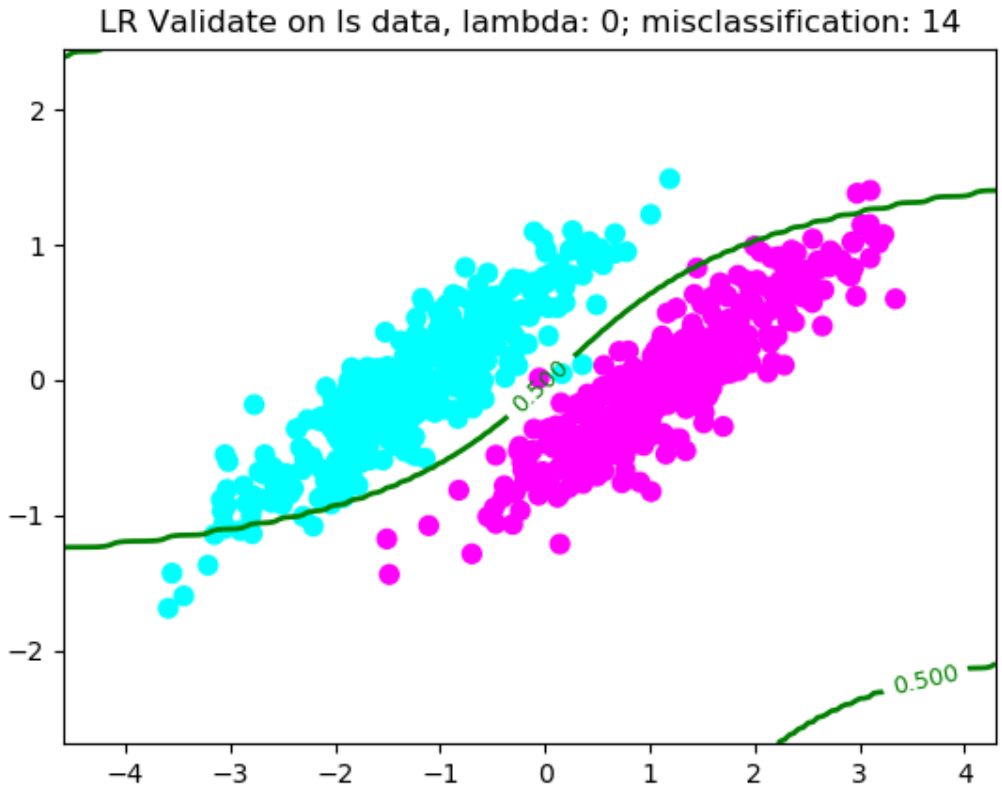
- When $\lambda = 0$, there is very little difference between training and the validation sets, even though it may be the case that the model may overfit the training data.
- Increasing λ
    - For ls data
        - Worsens the accuracy of the model. This is because the training and the validation sets are very similar and hence, even if the model over-fits the train data, it performs very well on the validation data.
    - For nls data
        - Since this data is not linearly separable, increasing lambda otherwise has no effect
    - For nonlin data
        - Similar to nls data, lambda has no effect as the data is not linearly separable, and we clearly need a higher dimension decision boundary
- One can tweak the lambda and find the best lambda that has the least number of misclassification in the validation set, but it would be futile as, the data clearly cannot be separated linearly.

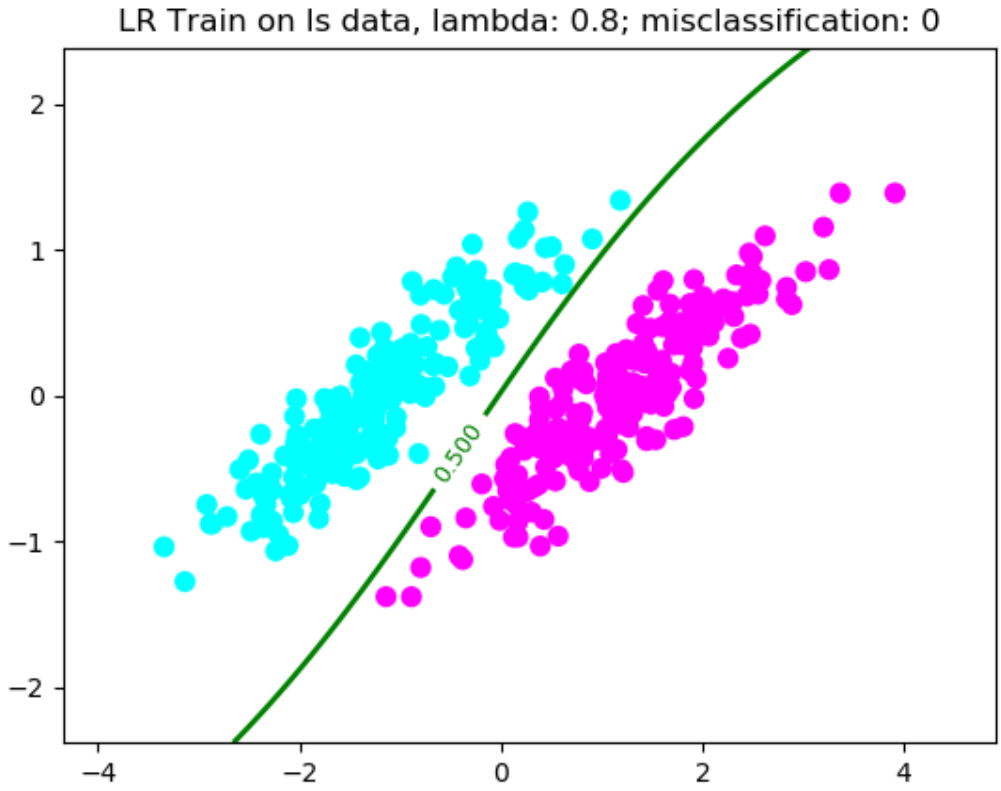
Part 3. Second order polynomial basis functions

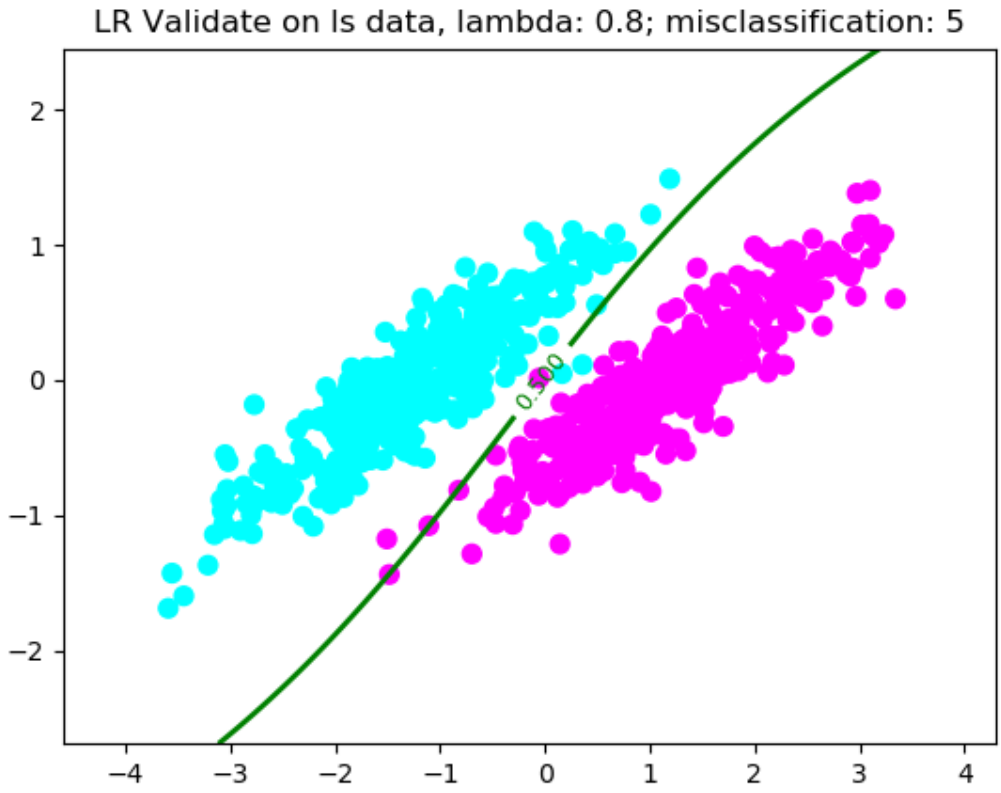
This analysis is based on the plots below (logistic regression using a polynomial basis function)

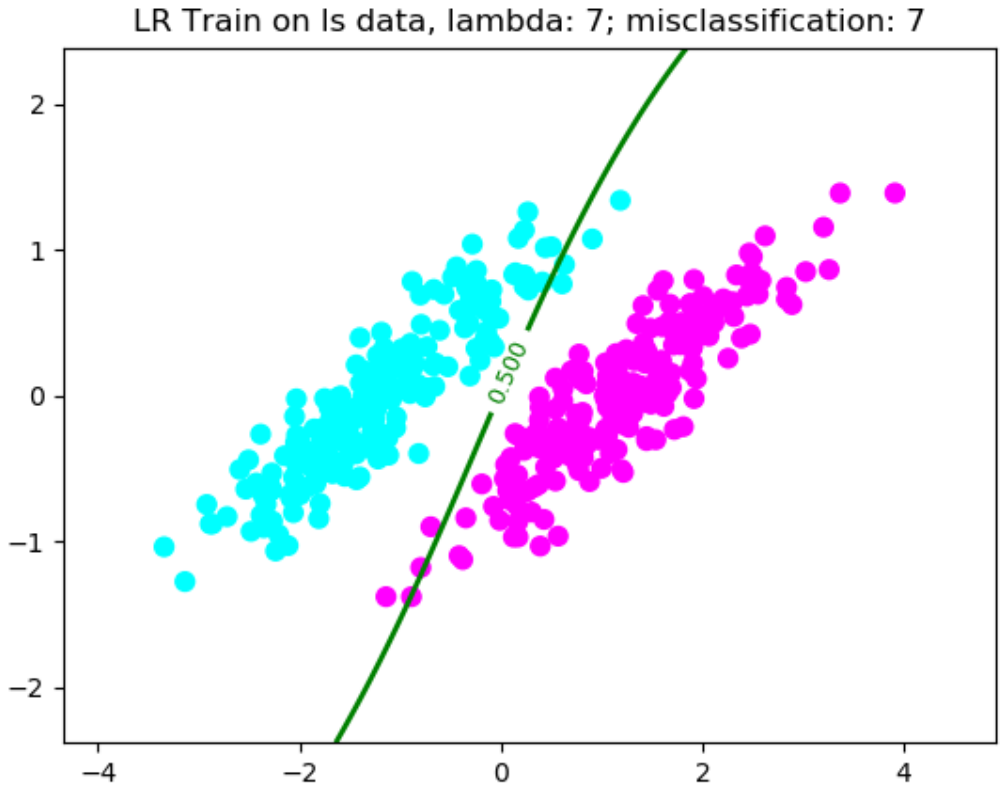
- For ls data
    - Since the data is linearly separable, using a polynomial basis function actually introduces misclassifications
    - But this is fixed by using a small lambda (regularize so the basis doesn't over-fit the training)
- For nls data
    - Poly basis function clearly fits better than a linear boundary
    - However, increasing the lambda has negligible effect on this data.
- For nonlin data
    - Poly basis function clearly fits better than a linear boundary
    - Increasing lambda improves the accuracy of the model
    - This is because, regularizing the weights, we try and generalize how the data is curved across the dimensions

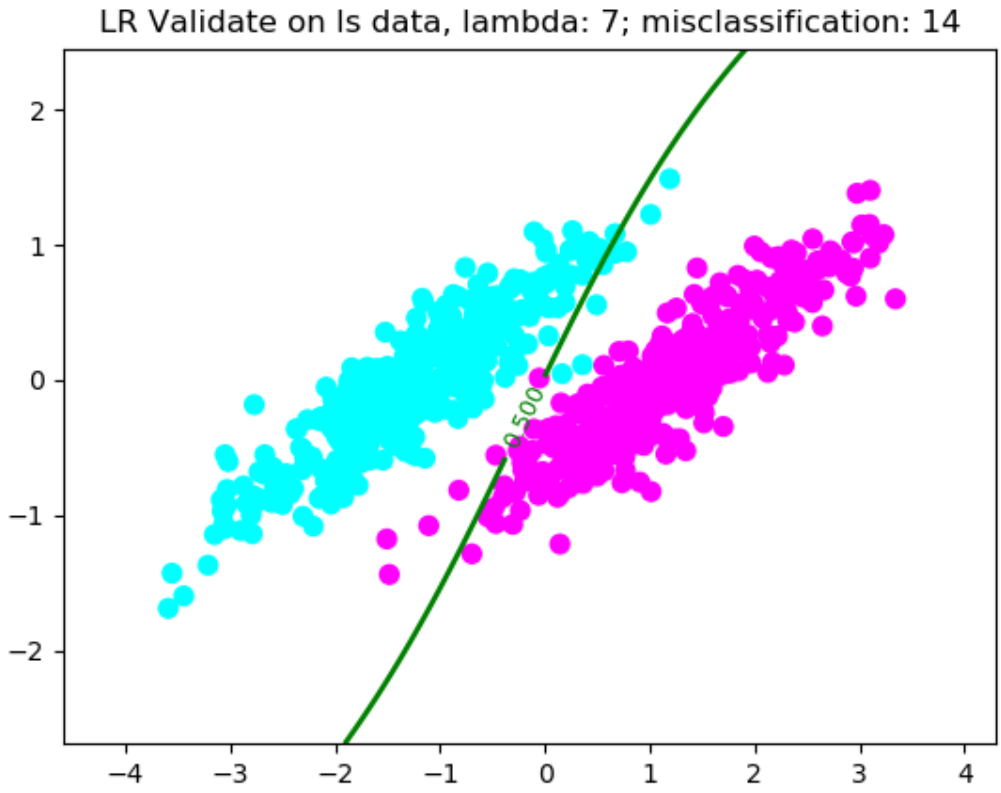

LR Train on ls data, lambda: 0; misclassification: 5
0.500
0.500

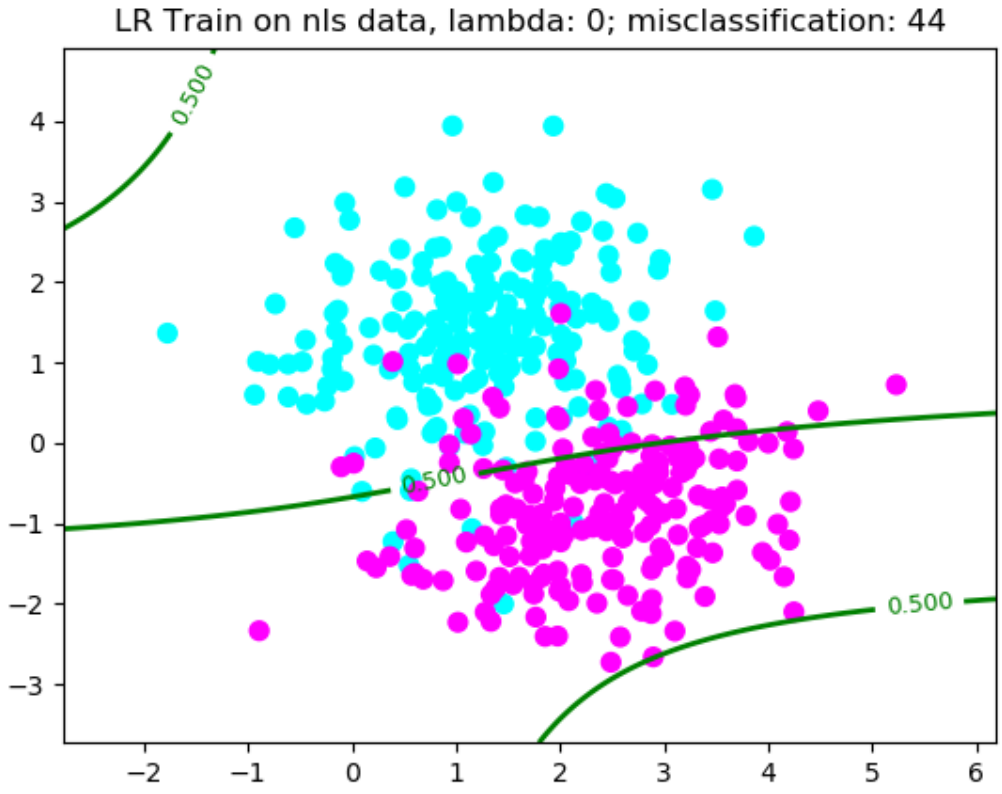

LR Validate on ls data, lambda: 0; misclassification: 14
0.500
0.500

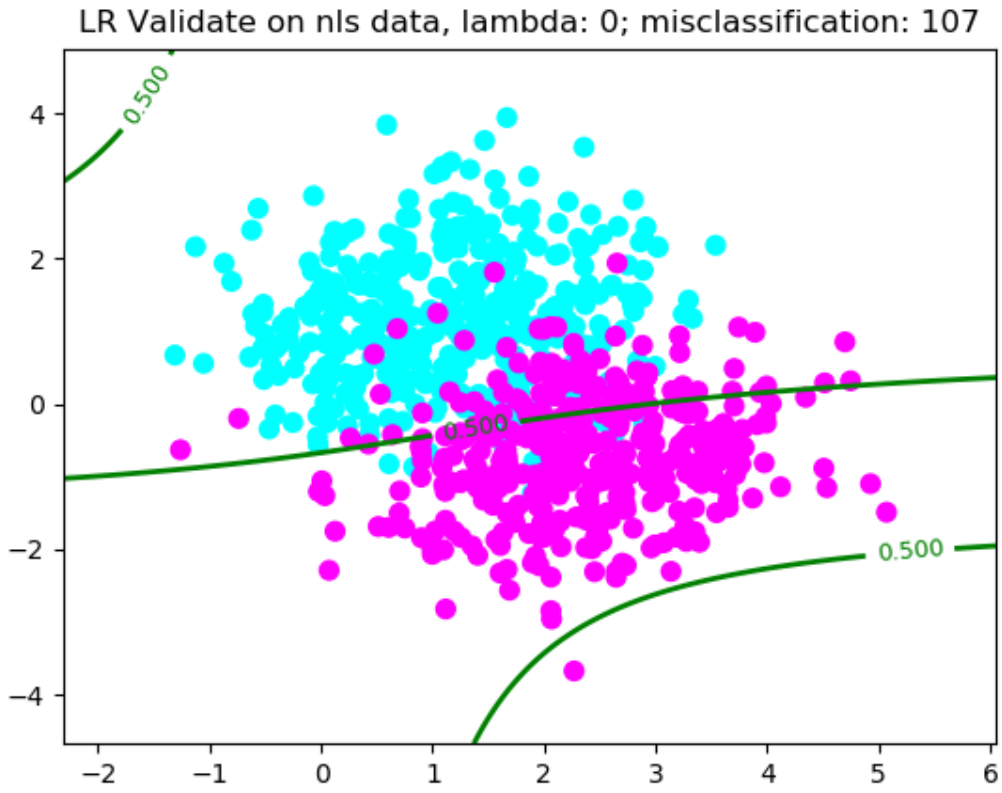

LR Train on ls data, lambda: 0.8; misclassification: 0
0.500

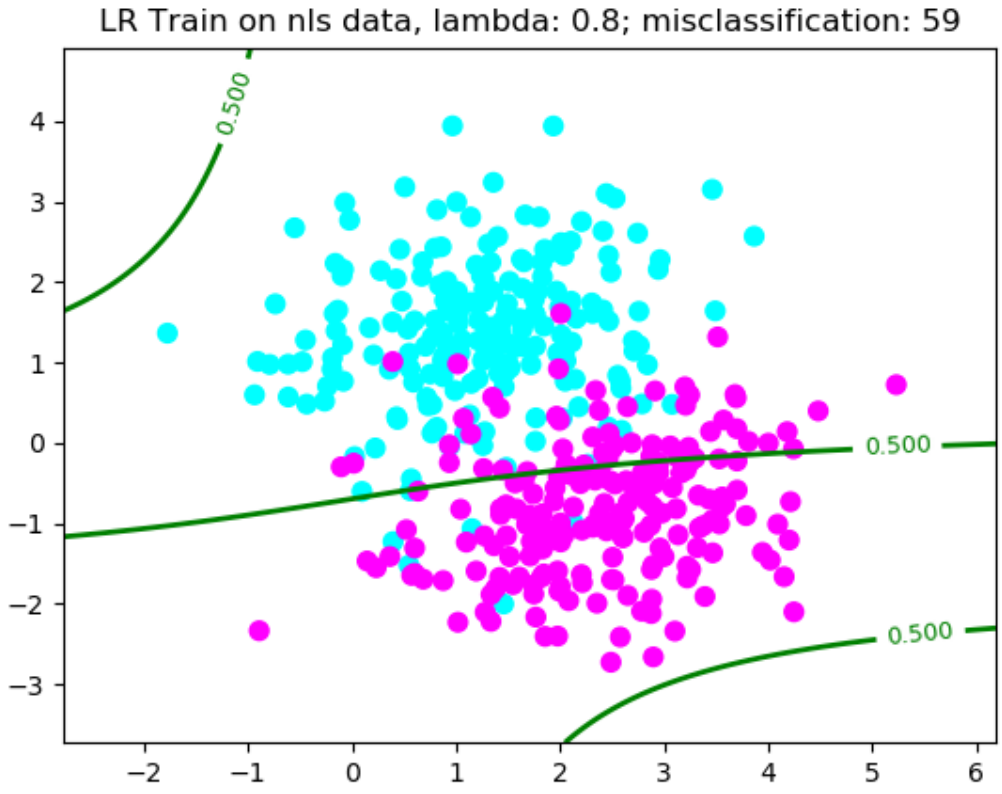

LR Validate on ls data, lambda: 0.8; misclassification: 5
0.500

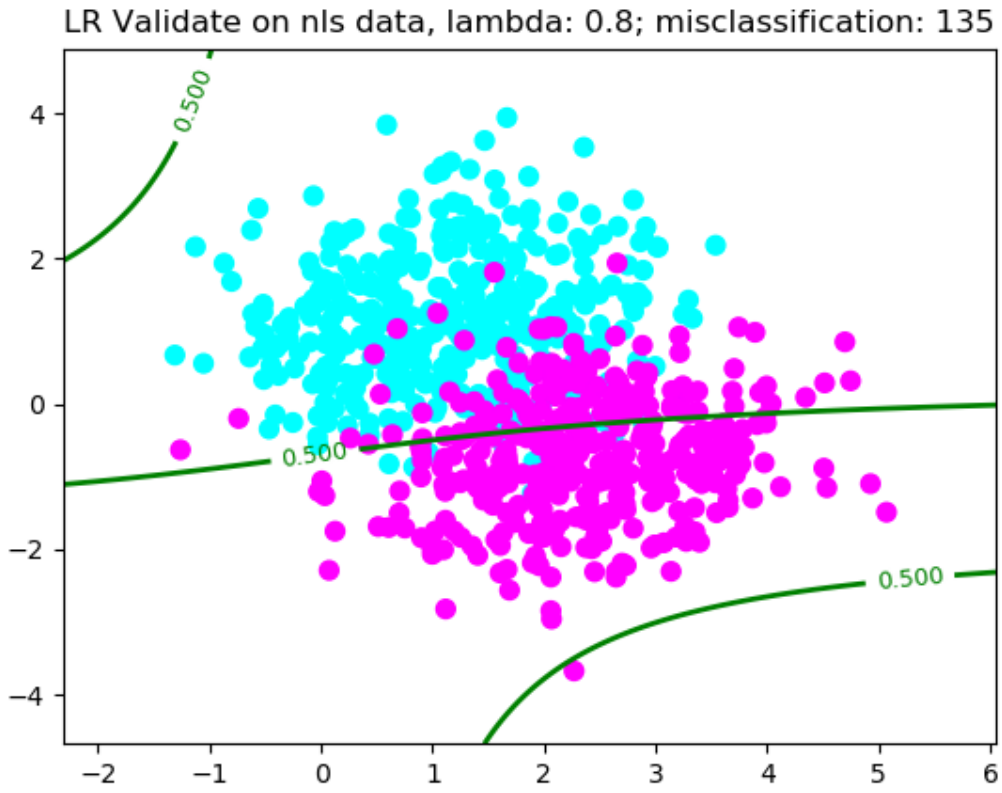

LR Train on ls data, lambda: 7; misclassification: 7
0.500
2
1
0
−1
−2
−4
−2
0
2
4

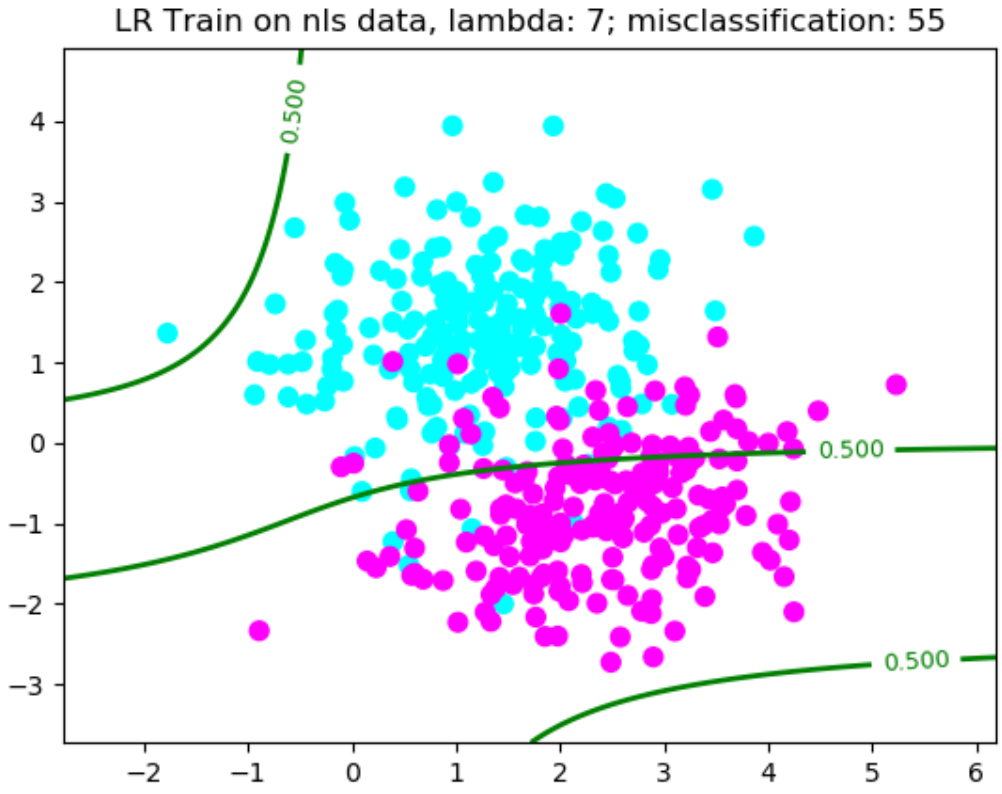

LR Validate on ls data, lambda: 7; misclassification: 14
0.500
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4

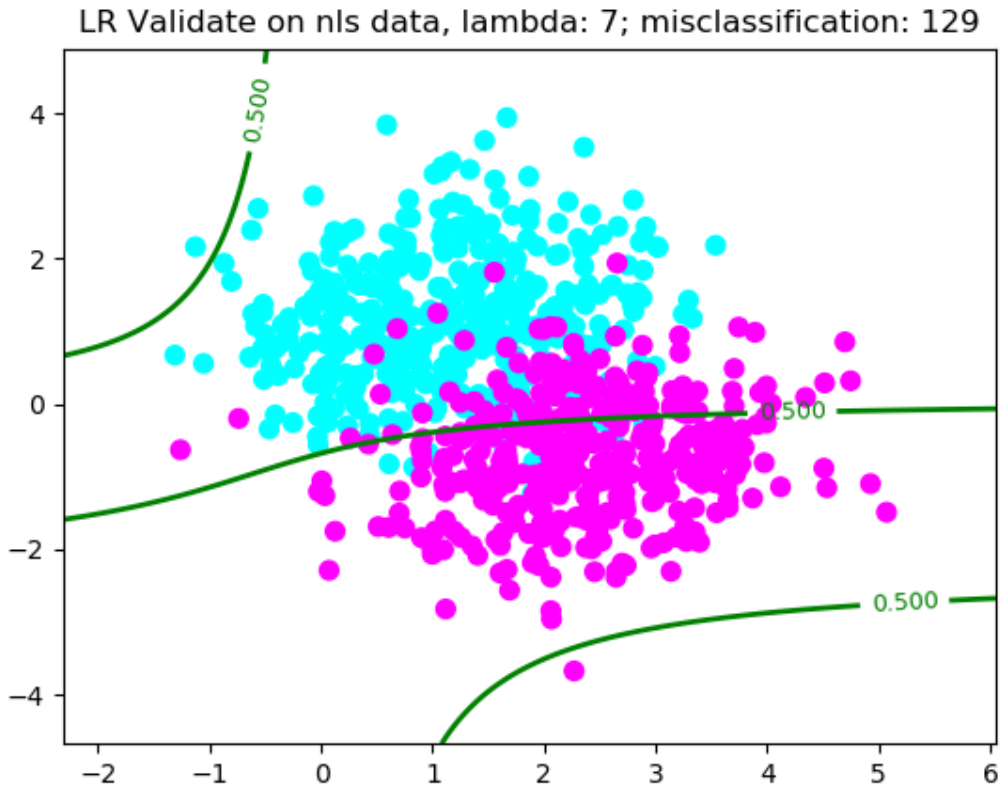

LR Train on nls data, lambda: 0; misclassification: 44
0.500
0.500
0.500
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5
6

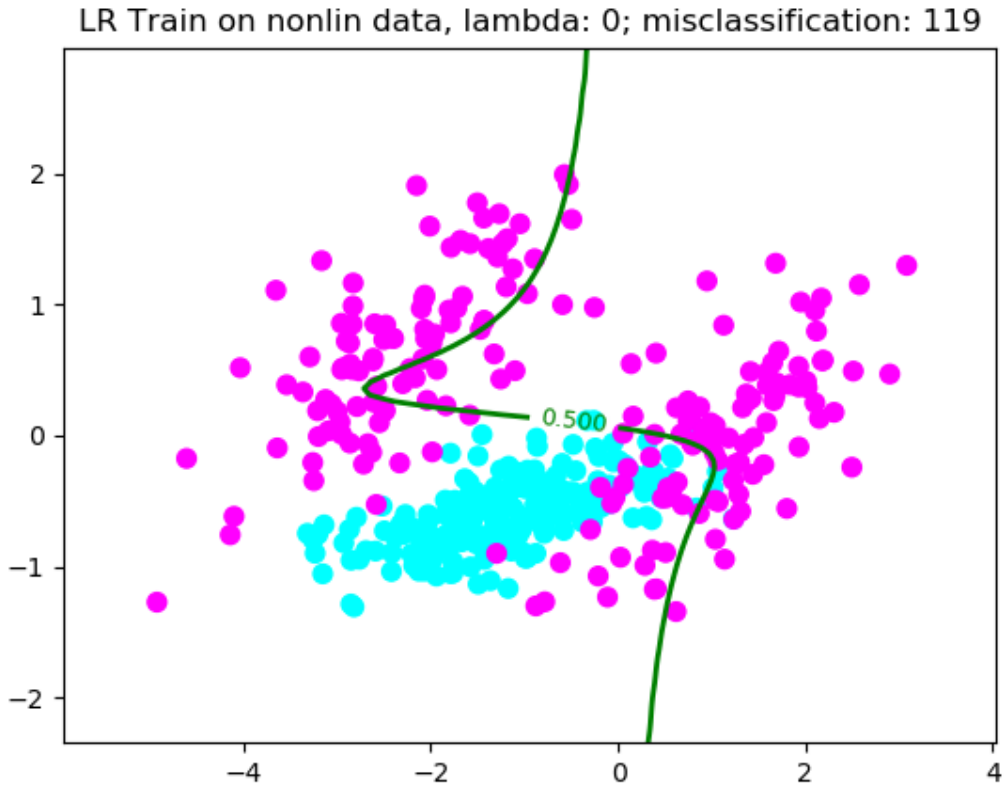

LR Validate on nls data, lambda: 0; misclassification: 107
0.500
0.500
0.500
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5
6

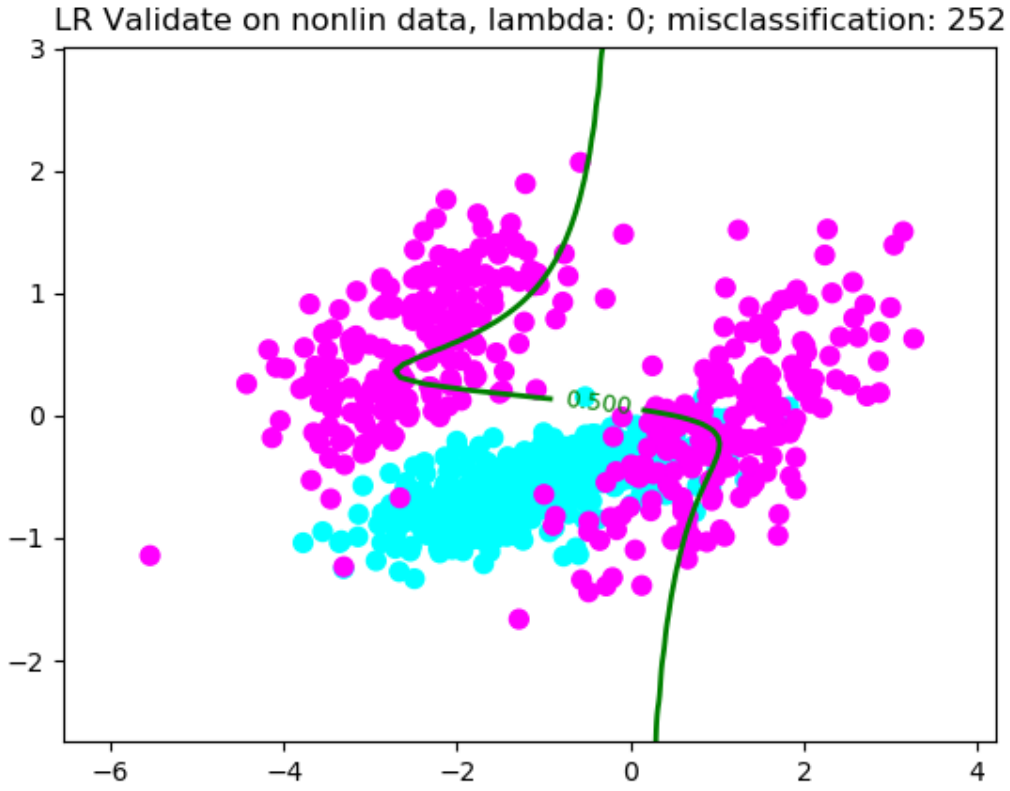

LR Train on nls data, lambda: 0.8; misclassification: 59
0.500
0.500
0.500
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5
6

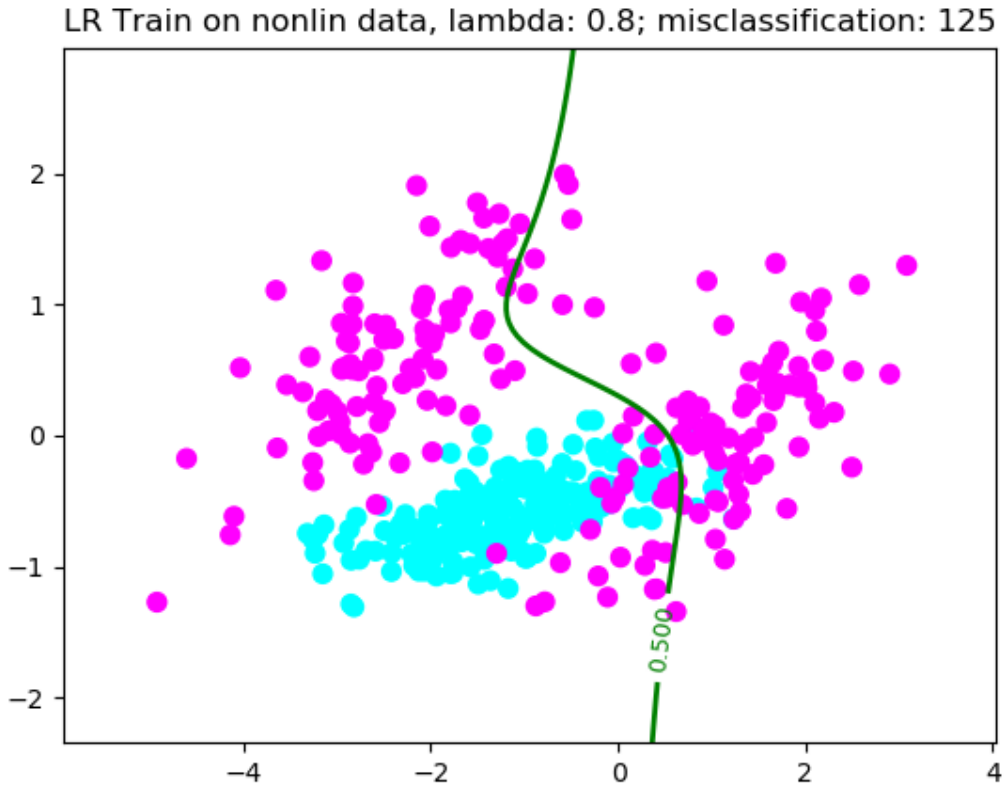

LR Validate on nls data, lambda: 0.8; misclassification: 135
0.500
0.500
0.500
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5
6

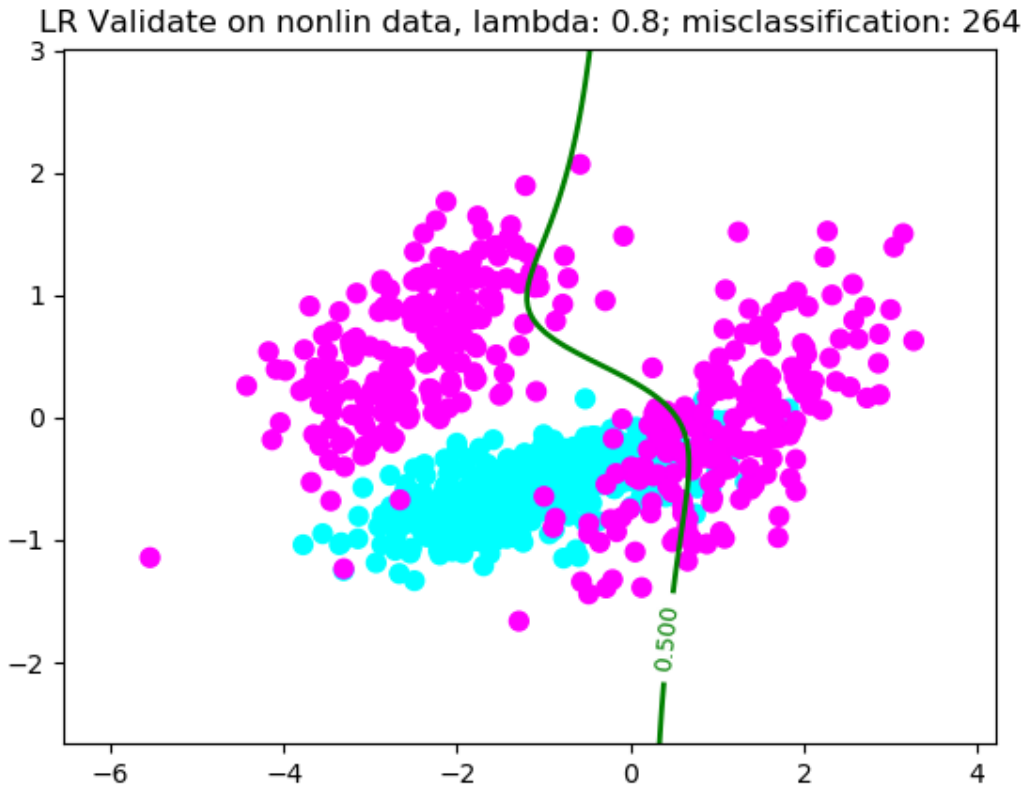

LR Train on nls data, lambda: 7; misclassification: 55
0.500
0.500
0.500
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5
6

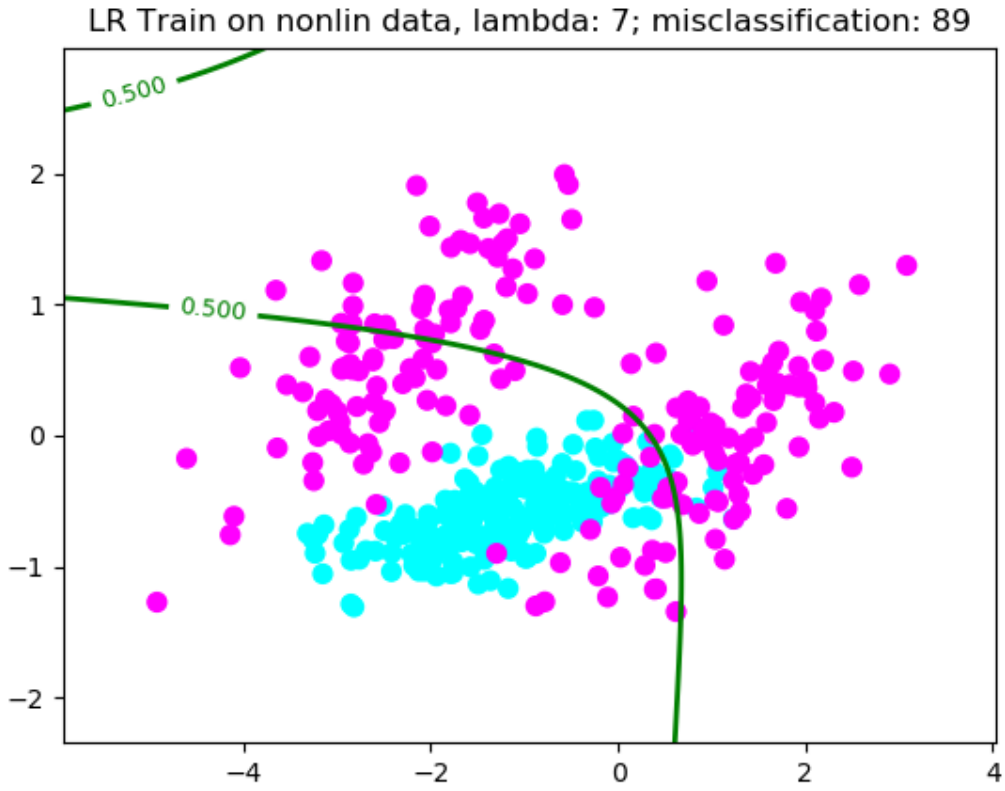

LR Validate on nls data, lambda: 7; misclassification: 129
0.500
0.500
0.500
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5
6

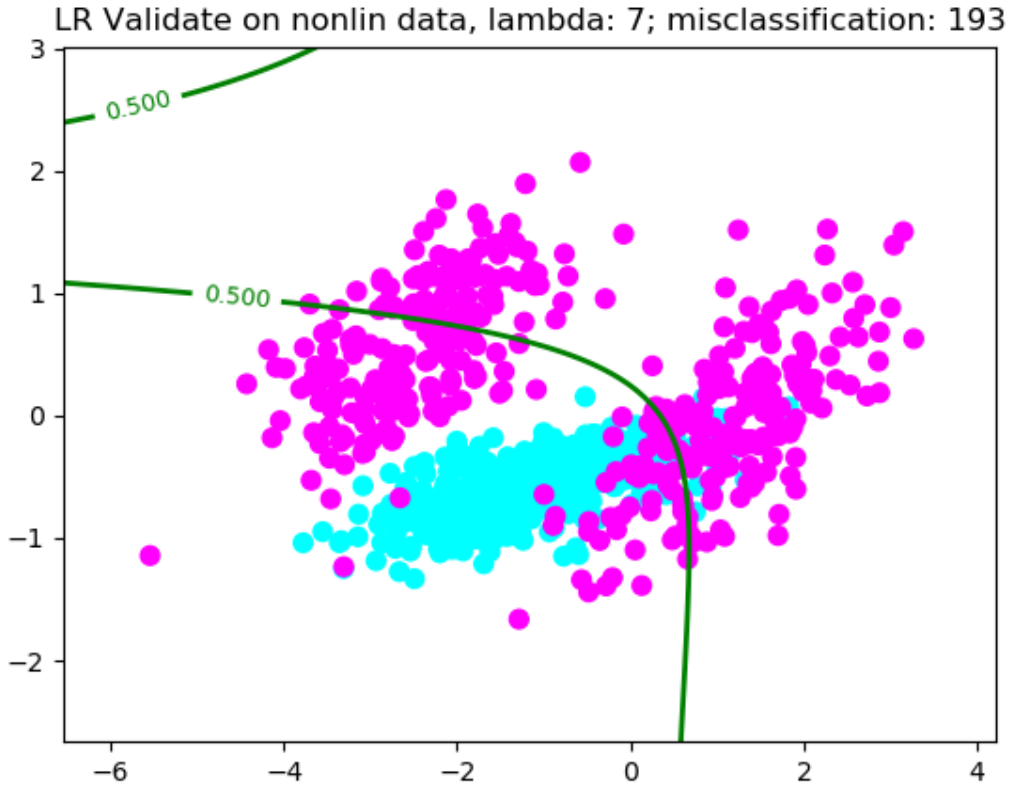

LR Train on nonlin data, lambda: 0; misclassification: 119
0.500
2
1
0
−1
−2
−4
−2
0
2
4


LR Validate on nonlin data, lambda: 0; misclassification: 252
0.500
3
2
1
0
−1
−2
−6
−4
−2
0
2
4


LR Train on nonlin data, lambda: 0.8; misclassification: 125
0.500


LR Validate on nonlin data, lambda: 0.8; misclassification: 264
0.500


LR Train on nonlin data, lambda: 7; misclassification: 89
0.500
0.500
2
1
0
−1
−2
−4
−2
0
2
4

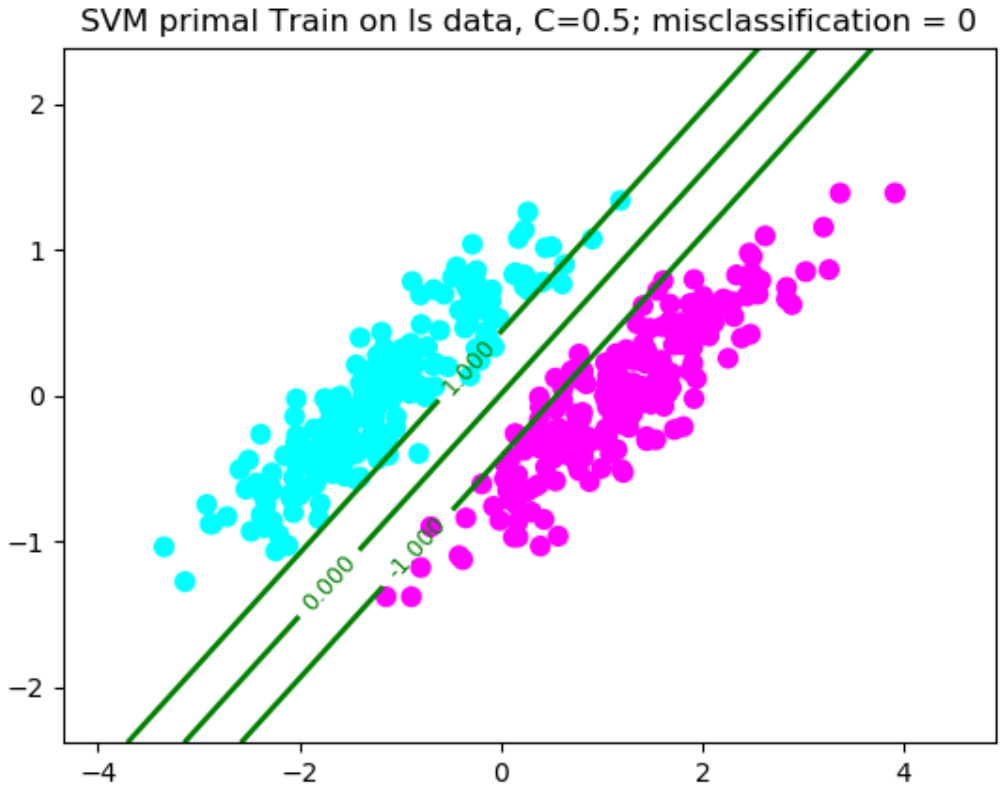

LR Validate on nonlin data, lambda: 7; misclassification: 193
0.500
0.500
3
2
1
0
−1
−2
−6
−4
−2
0
2
4

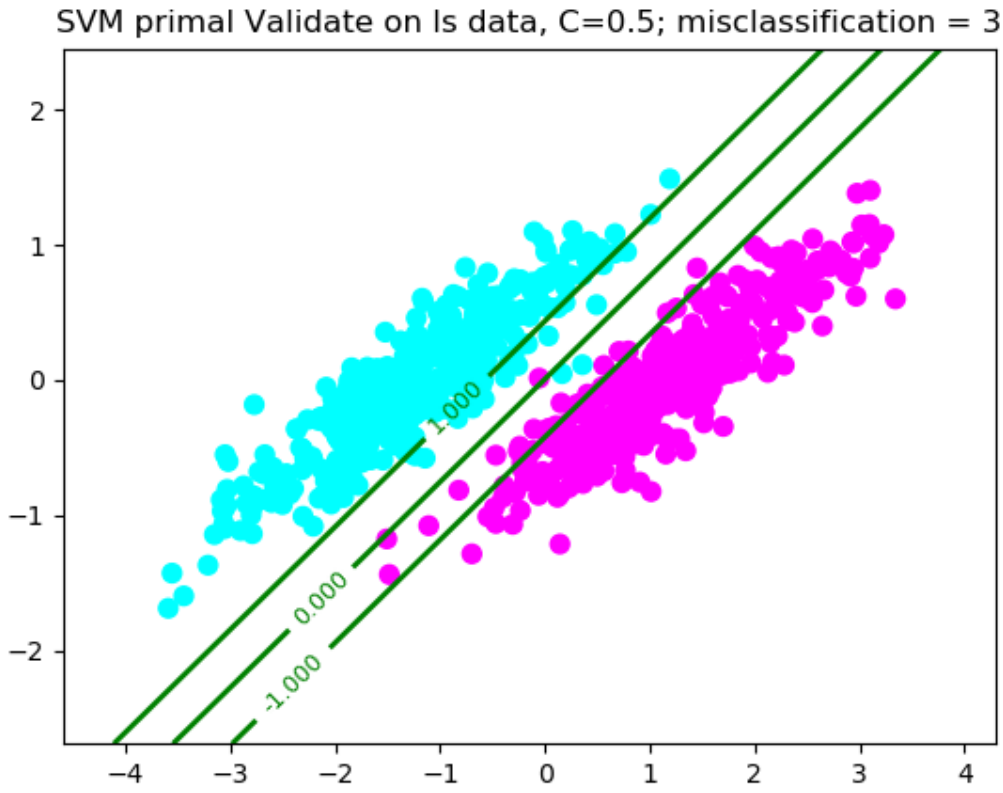
Q2. Support vector machine implementation

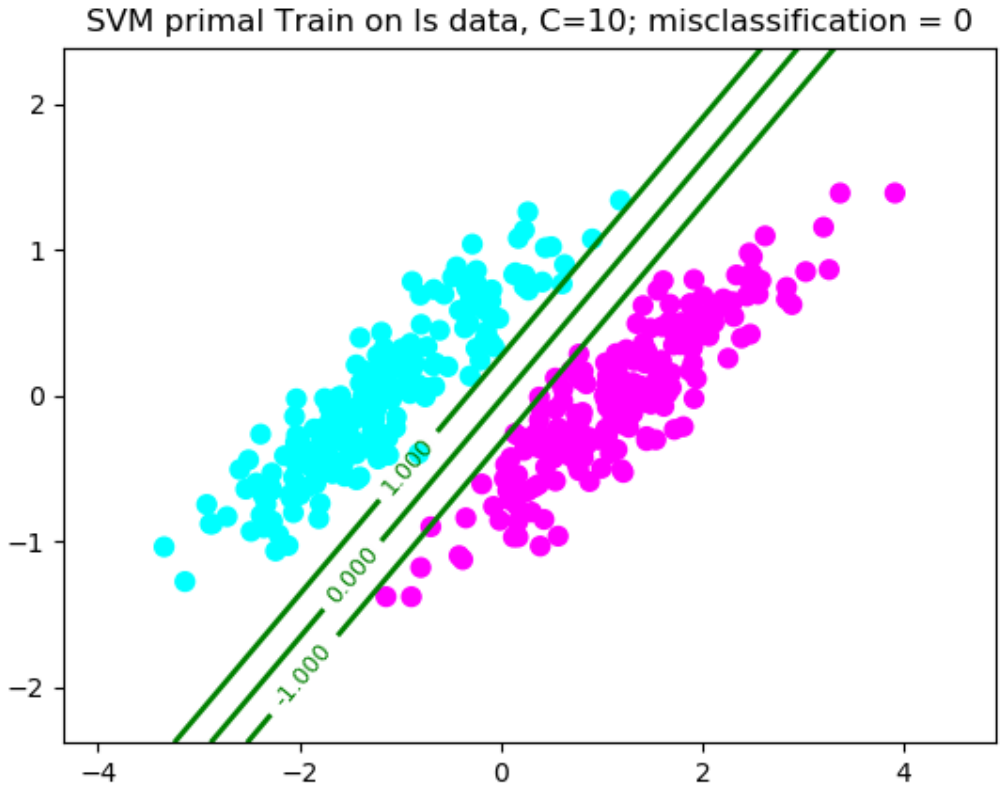
This analysis is based on the plots below:

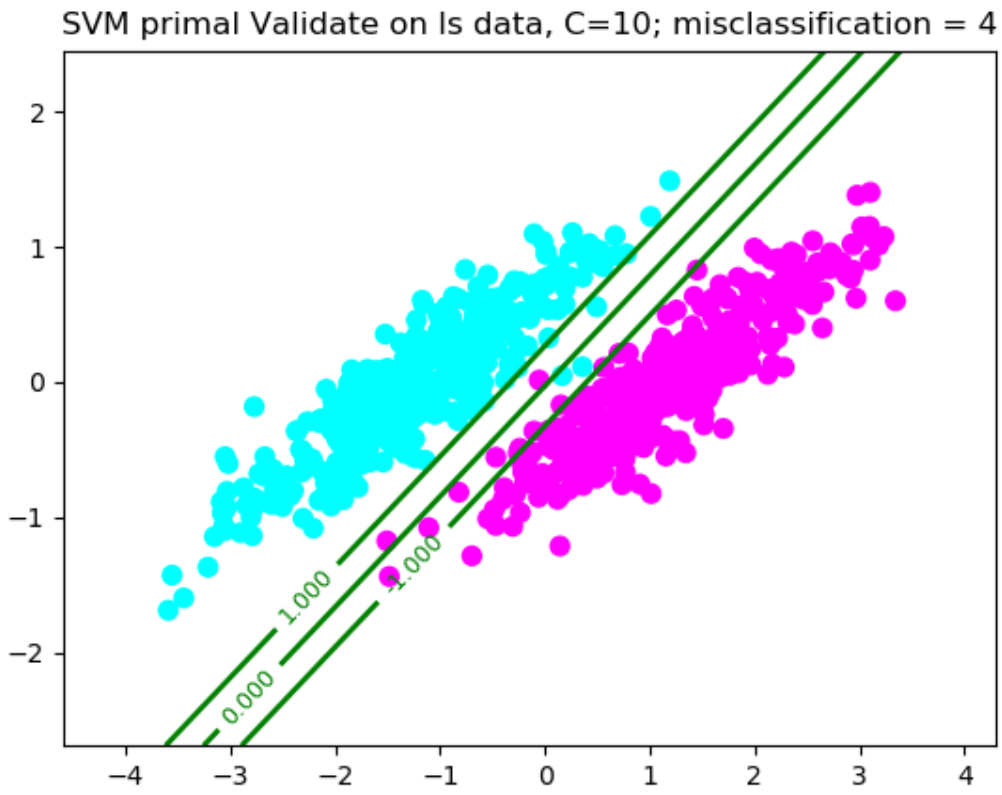
- For ls data
    - Both dual and primal behave similarly and give the same accuracy as that of logistic regression (either with a linear or polynomial basis function)
- For nls data
    - Using SVM is clearly gives a better classification. This is intuitive as in SVM we try and minimize C (number of misclassifications)
- For nonlin data:
    - This still give a really bad classifier, mainly because we are trying to fit a linear decision boundary to non-linear data
- For all data, we can see that increasing the C value, the margin reduces and vice versa.

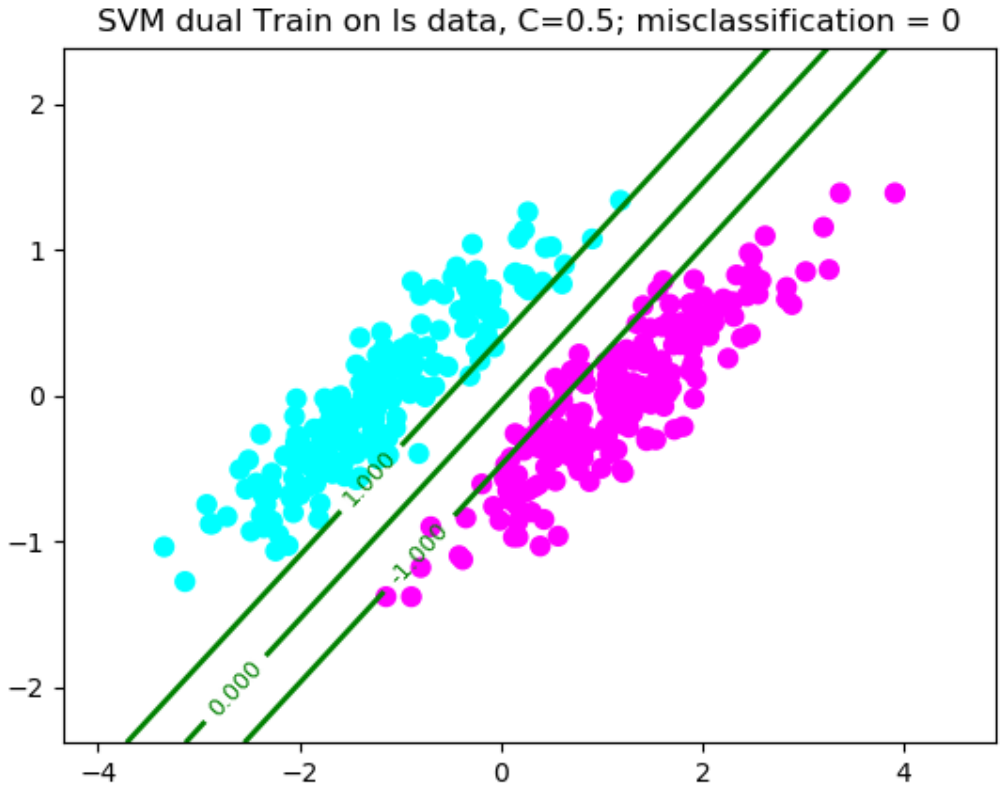
SVM Dual and Primal form for the data sets for various C:

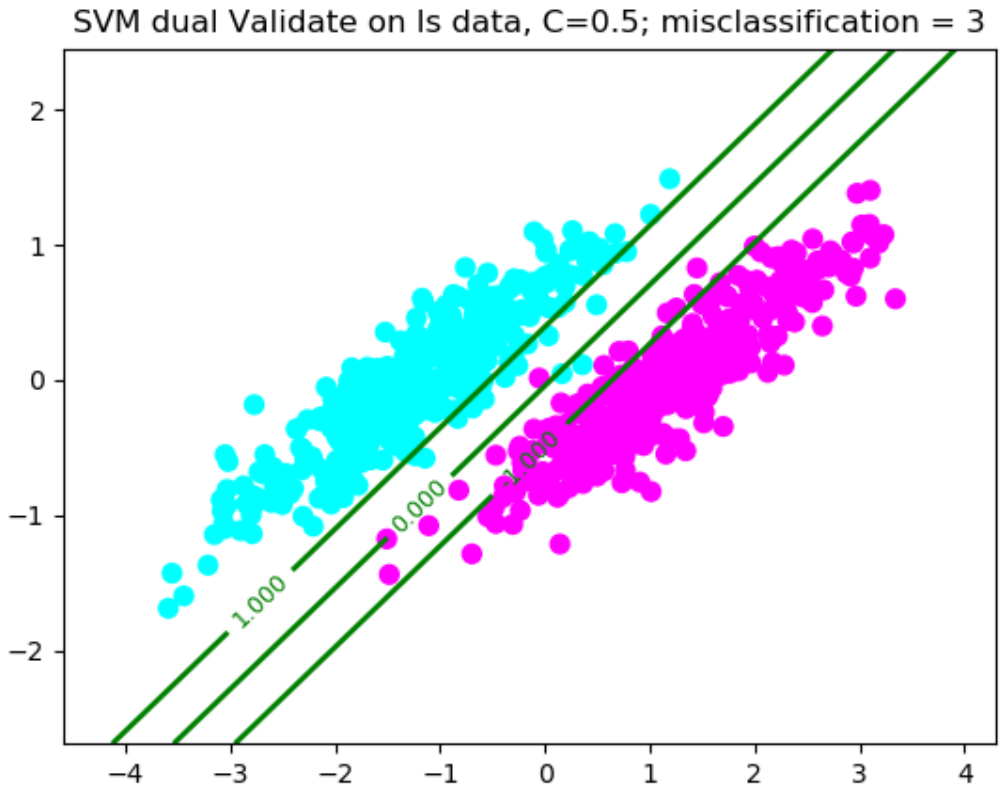

SVM primal Train on ls data, C=0.5; misclassification = 0
1.000
0.000
-1.000

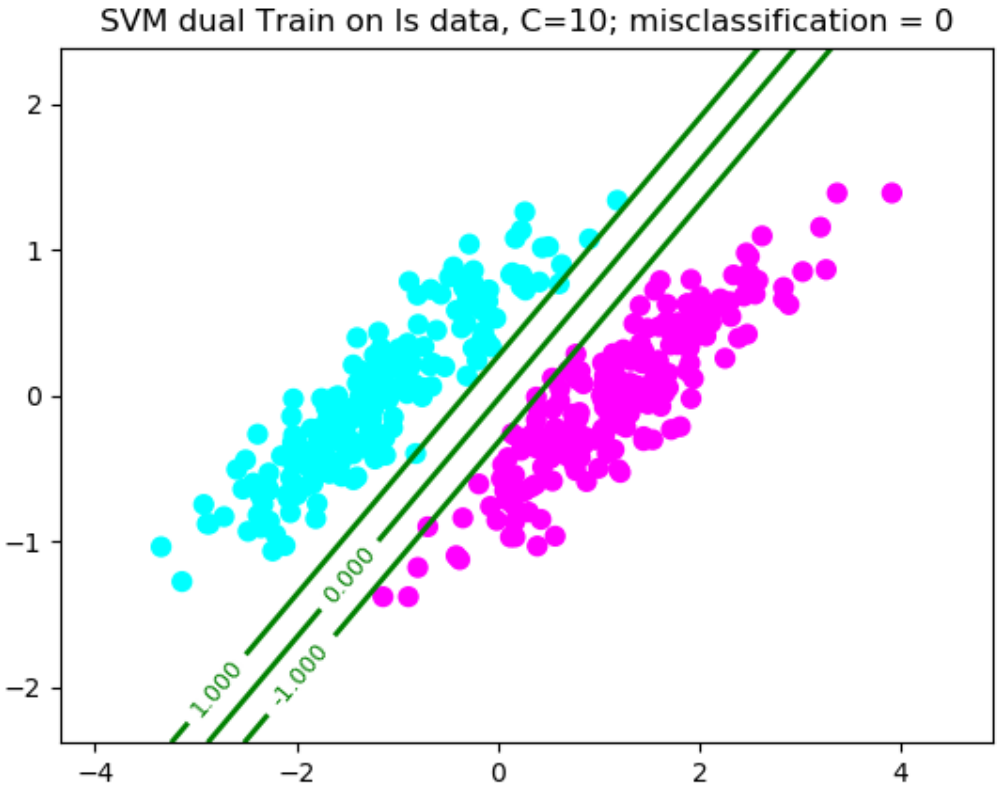

SVM primal Validate on ls data, C=0.5; misclassification = 3
1.000
0.000
-1.000

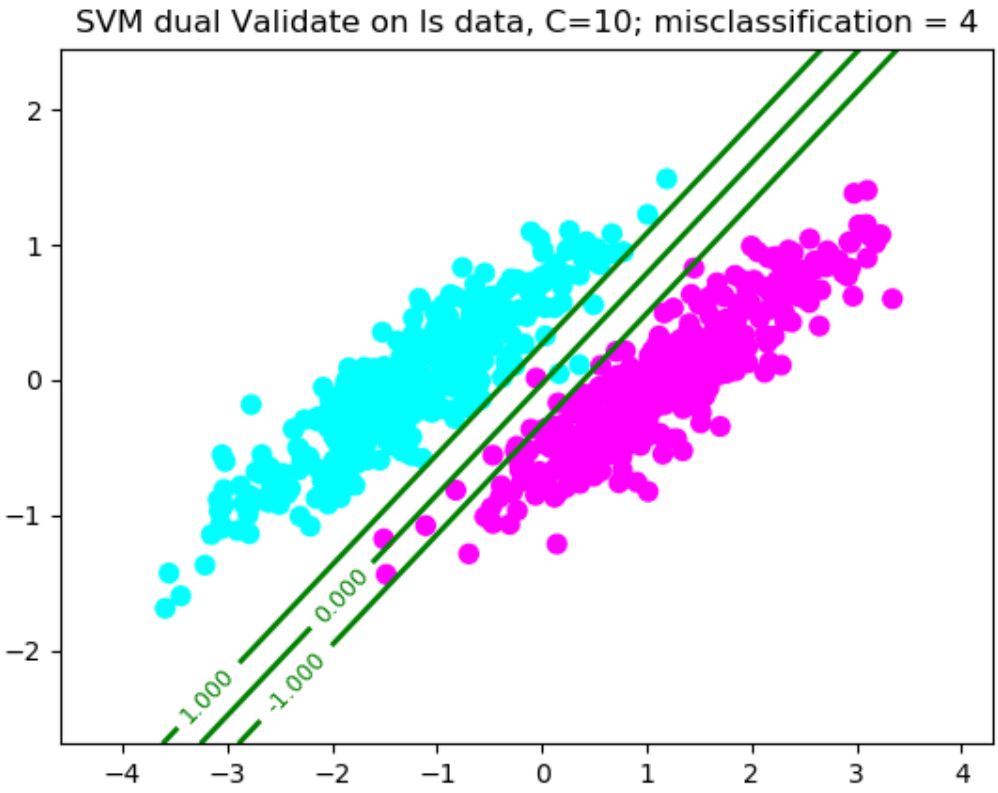

SVM primal Train on ls data, C=10; misclassification = 0
1.000
0.000
-1.000

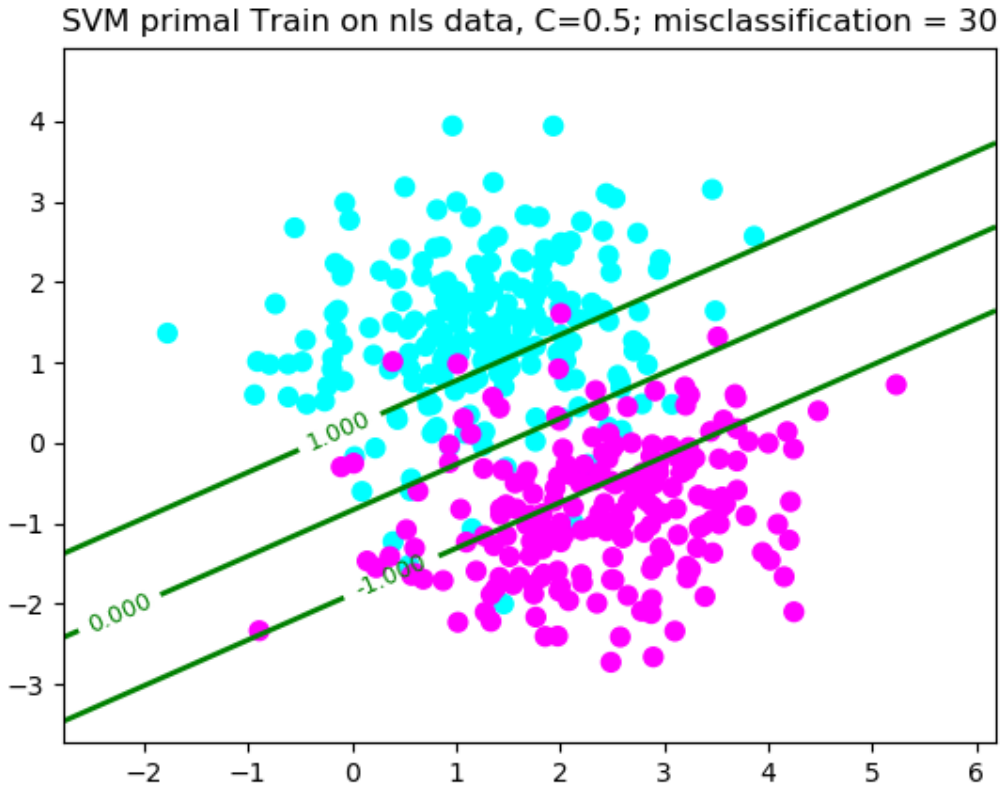

SVM primal Validate on ls data, C=10; misclassification = 4
1.000
0.000
-1.000

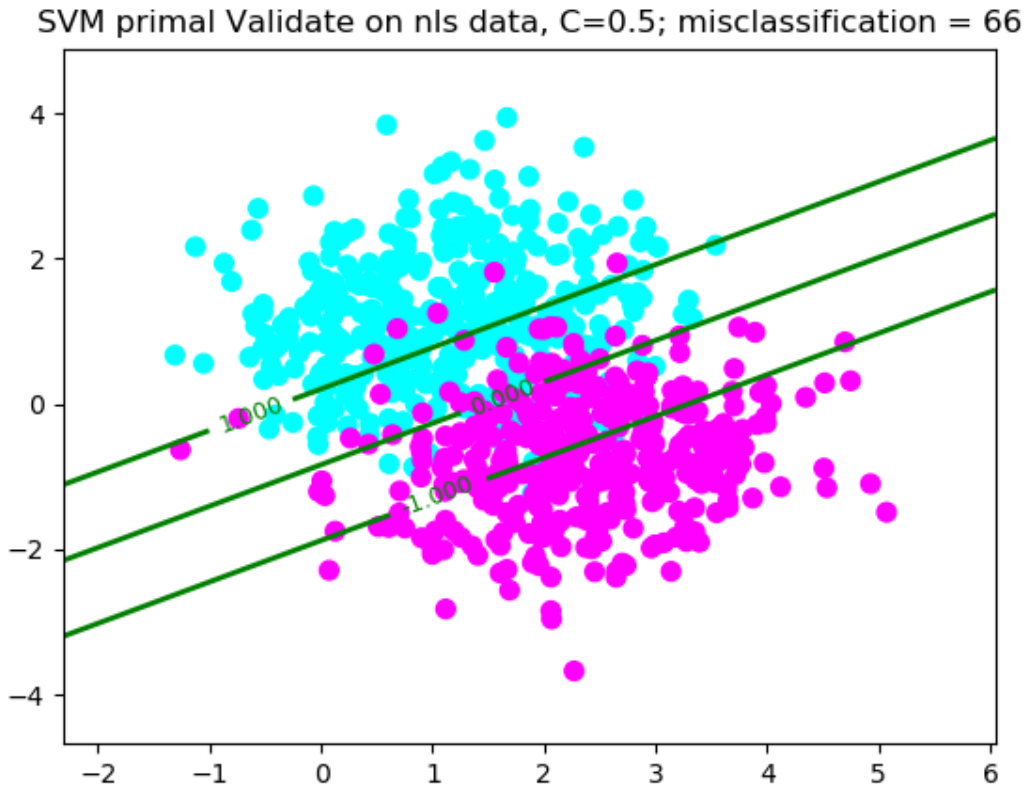

SVM dual Train on ls data, C=0.5; misclassification = 0
1.000
0.000
-1.000

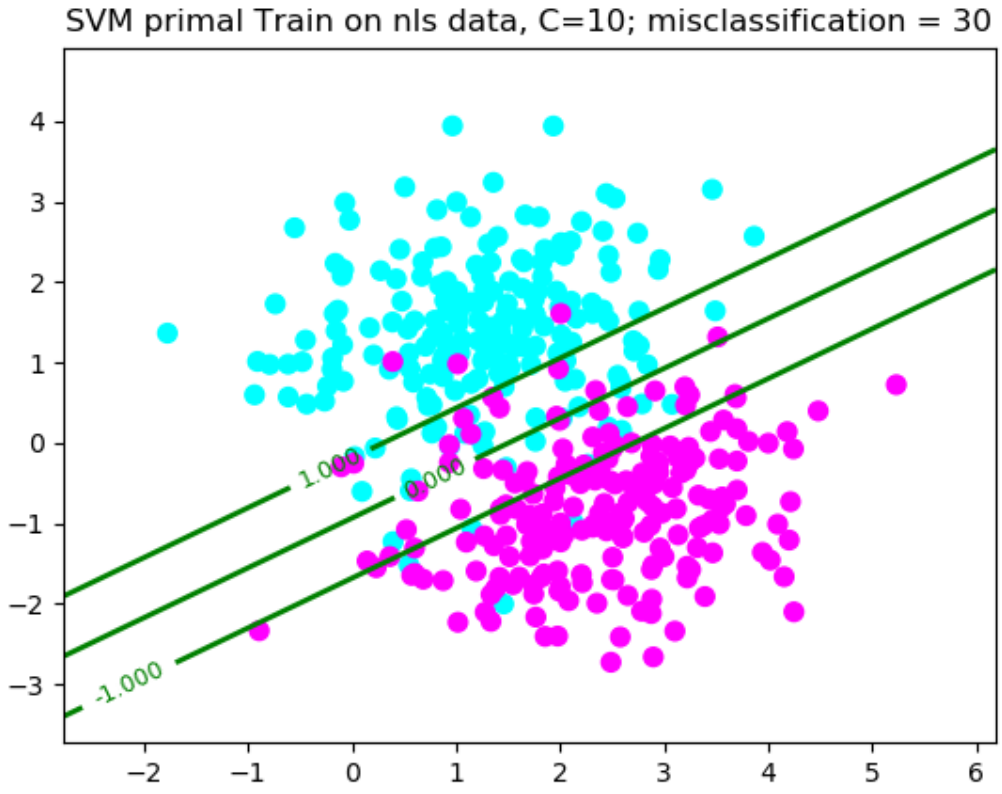

SVM dual Validate on ls data, C=0.5; misclassification = 3
1.000
0.000
-1.000

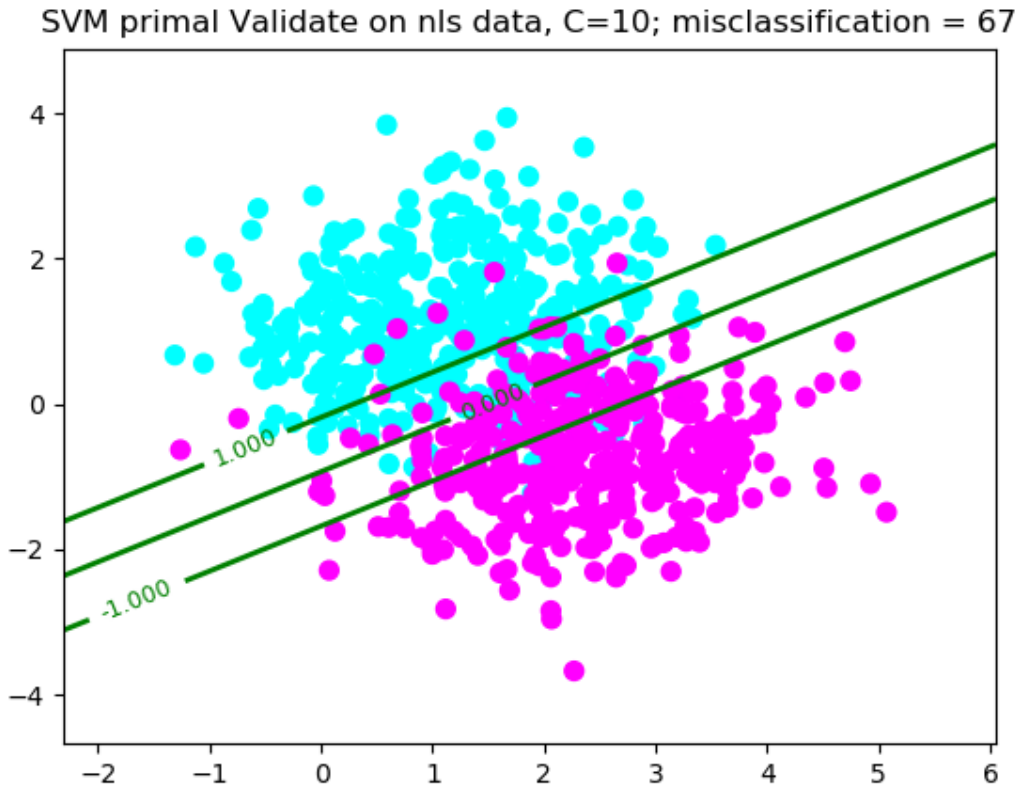

SVM dual Train on ls data, C=10; misclassification = 0
1.000
0.000
-1.000

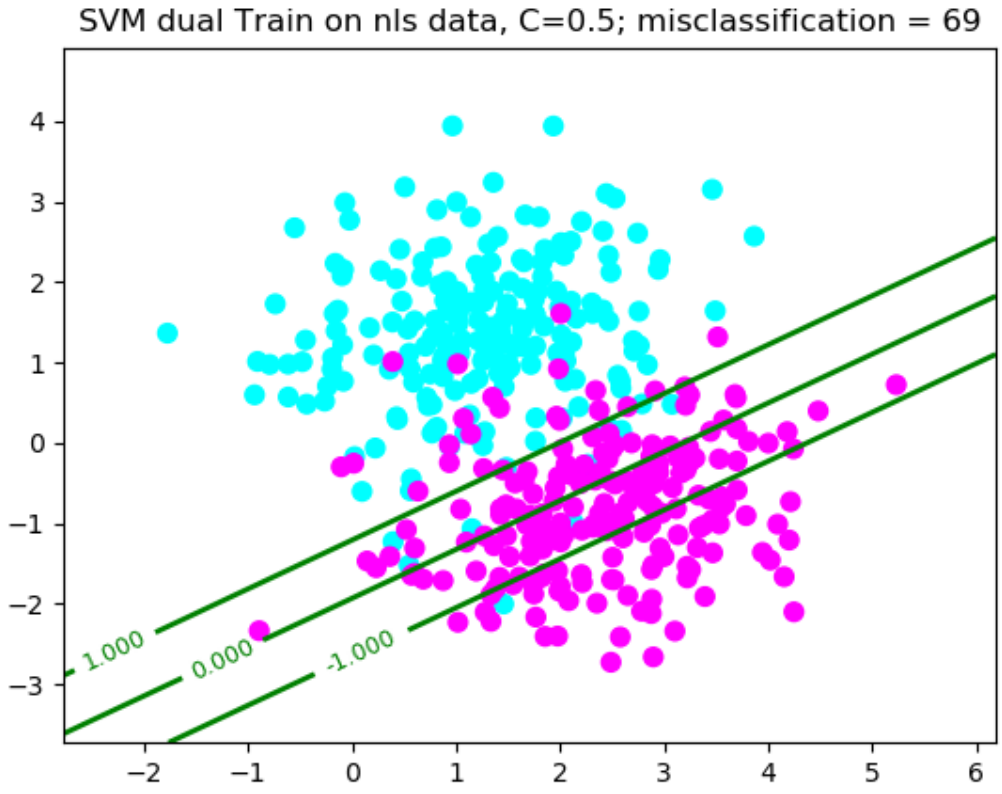

SVM dual Validate on ls data, C=10; misclassification = 4
1.000
0.000
-1.000

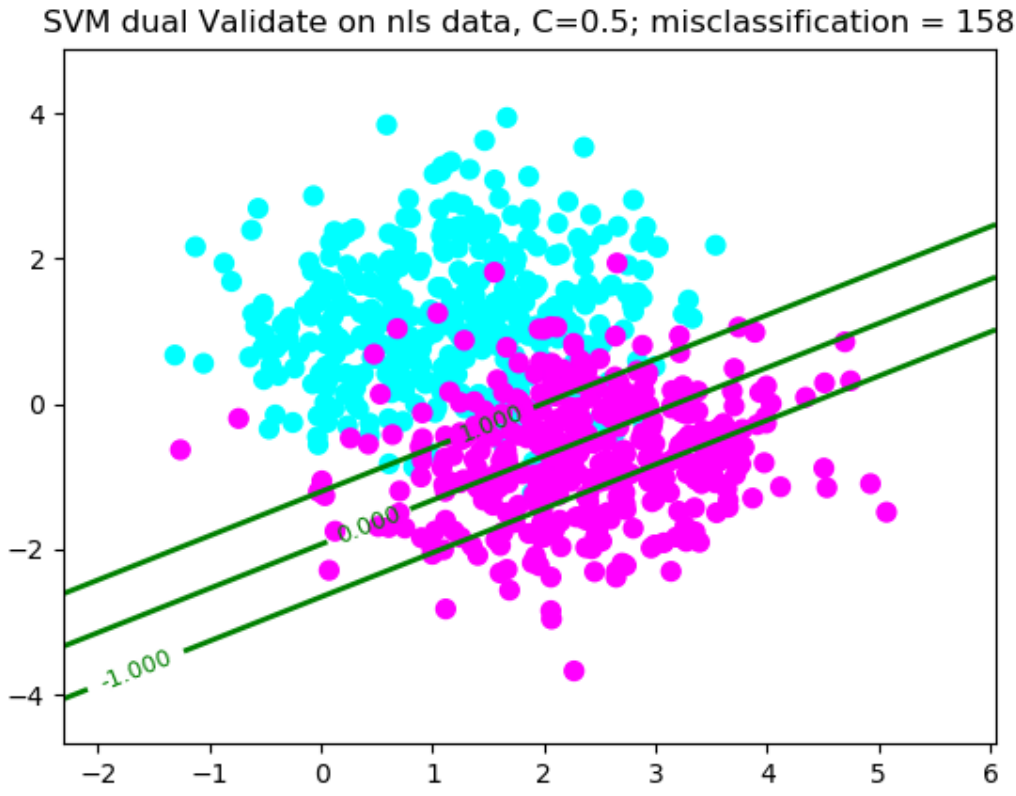

SVM primal Train on nls data, C=0.5; misclassification = 30
1.000
0.000
-1.000

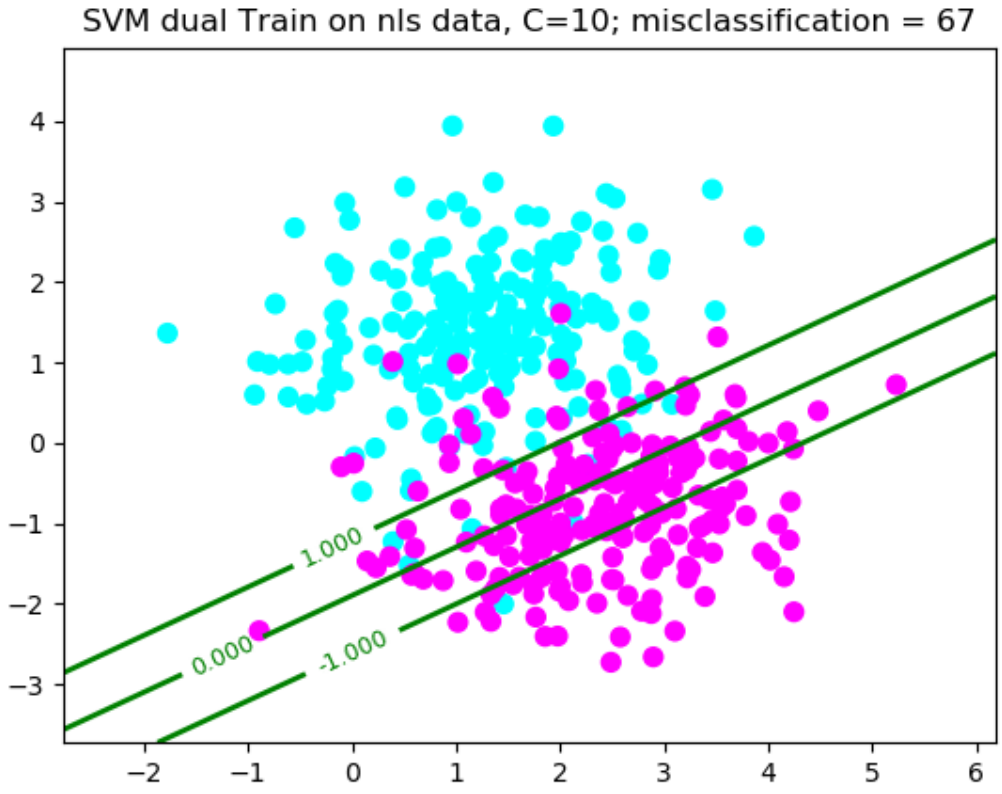

SVM primal Validate on nls data, C=0.5; misclassification = 66
1.000
0.000
-1.000

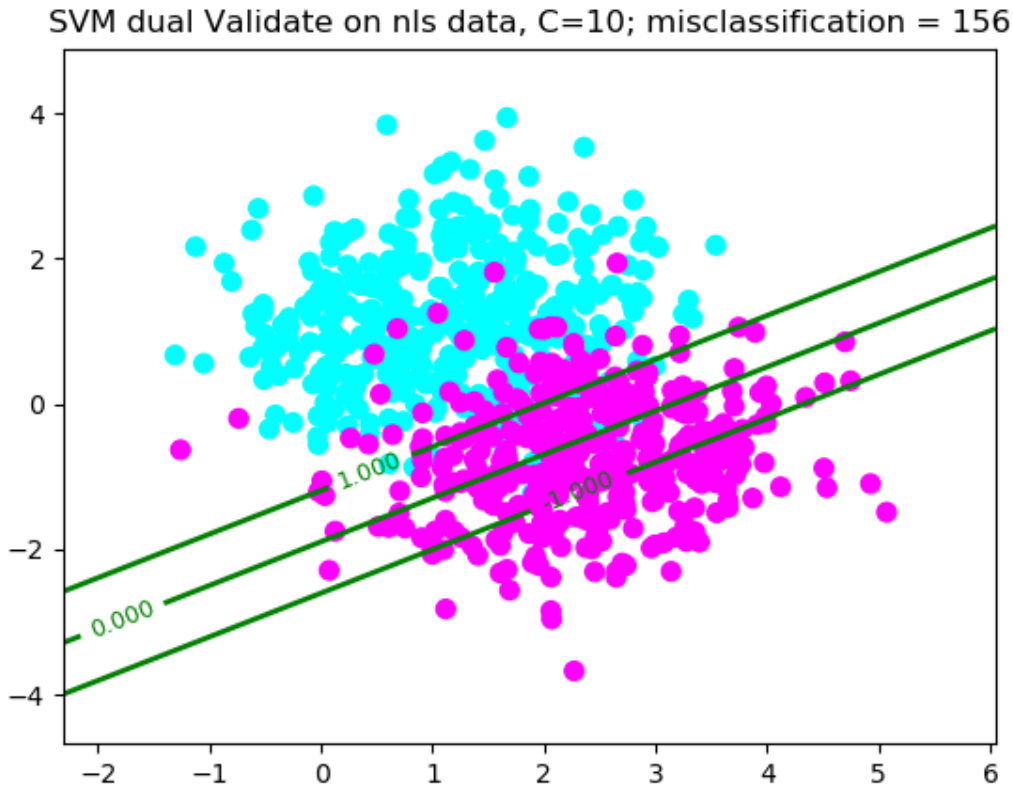

SVM primal Train on nls data, C=10; misclassification = 30
1.000
0.000
-1.000

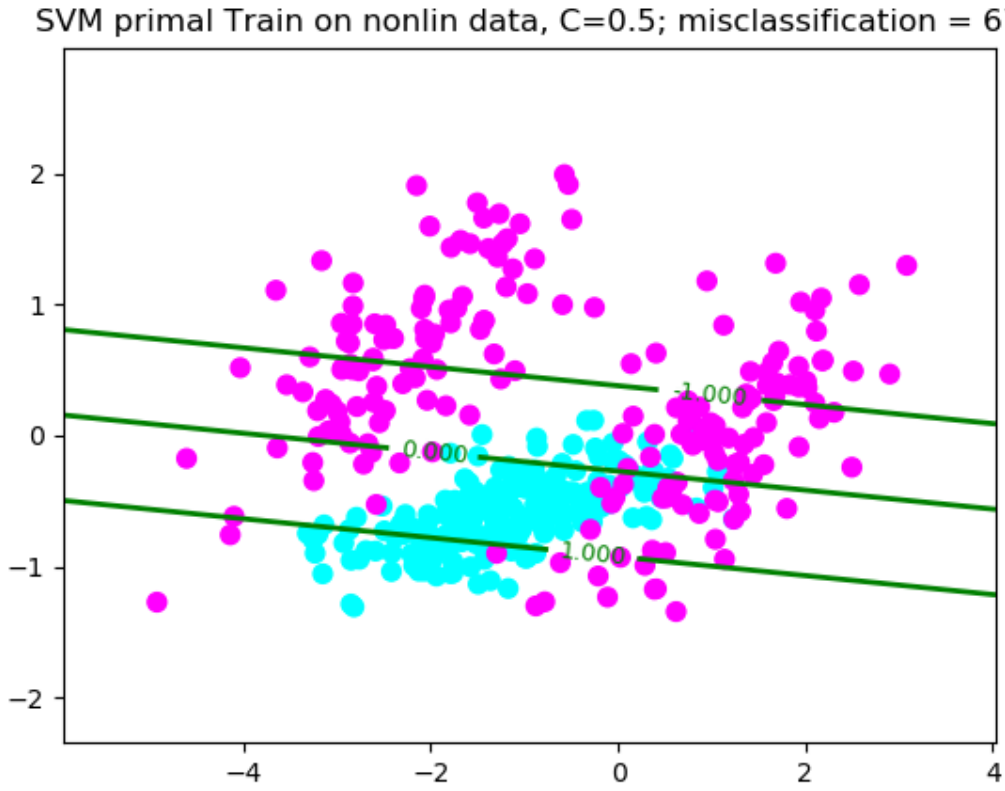

SVM primal Validate on nls data, C=10; misclassification = 67
1.000
0.000
-1.000

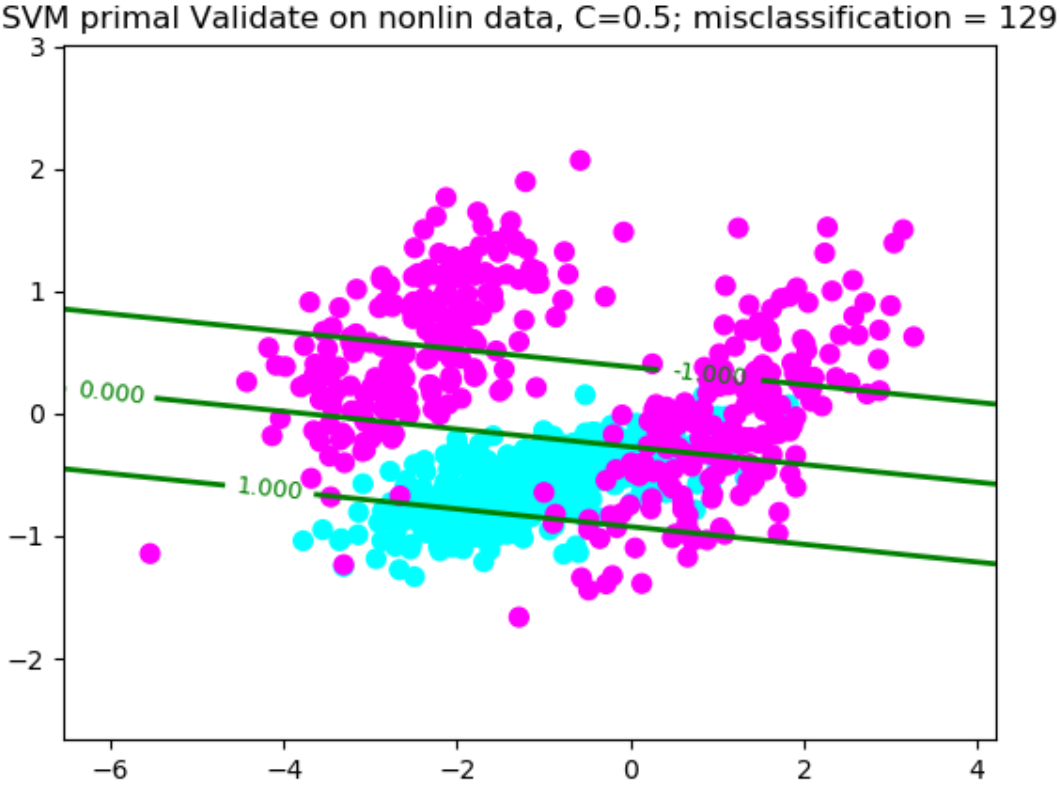

SVM dual Train on nls data, C=0.5; misclassification = 69
1.000
0.000
-1.000

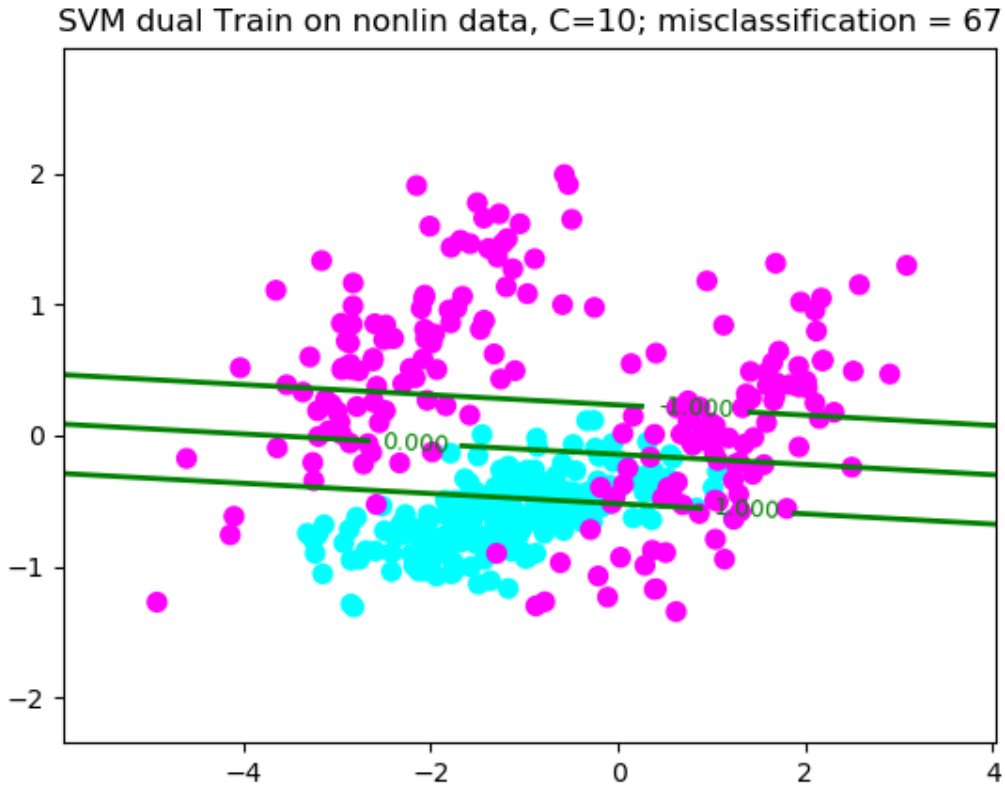

SVM dual Validate on nls data, C=0.5; misclassification = 158
1.000
0.000
-1.000

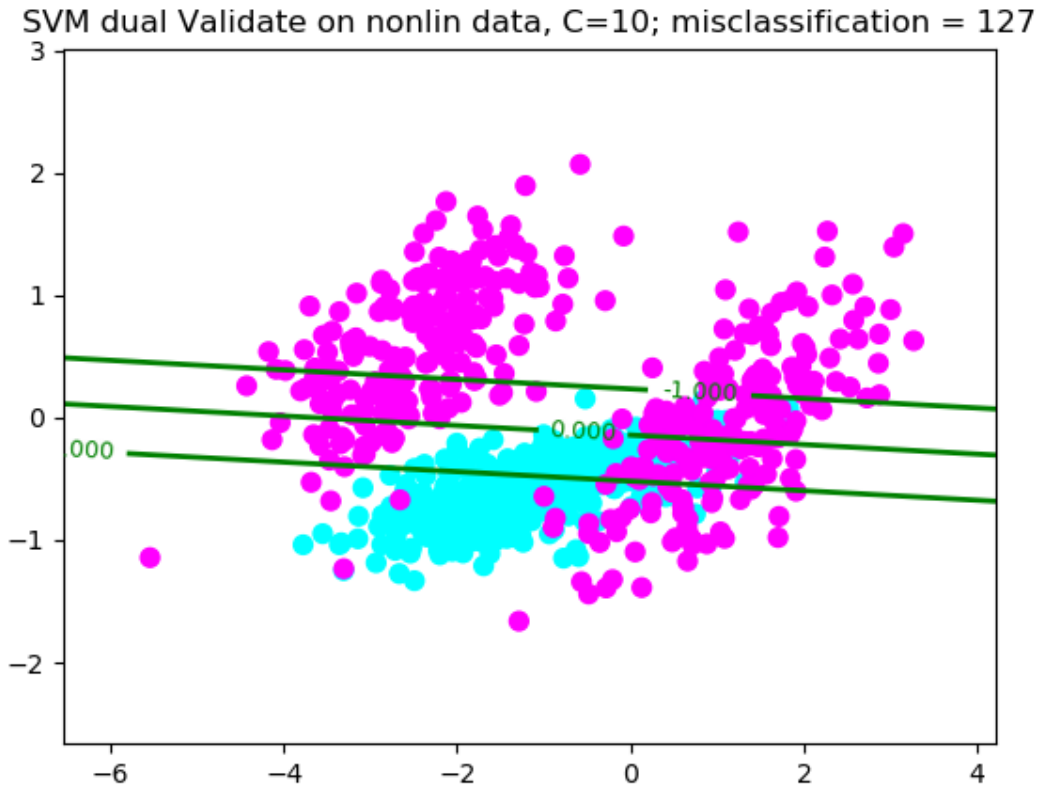

SVM dual Train on nls data, C=10; misclassification = 67
1.000
0.000
-1.000


SVM dual Validate on nls data, C=10; misclassification = 156
1.000
0.000
-1.000


SVM primal Train on nonlin data, C=0.5; misclassification = 67
-1.000
0.000
1.000
2
1
0
−1
−2
−4
−2
0
2
4


SVM primal Validate on nonlin data, C=0.5; misclassification = 129
-1.000
0.000
1.000
3
2
1
0
−1
−2
−6
−4
−2
0
2
4


SVM dual Train on nonlin data, C=10; misclassification = 67
-1.000
0.000
1.000
2
1
0
−1
−2
−4
−2
0
2
4


SVM dual Validate on nonlin data, C=10; misclassification = 127
-1.000
0.000
000
3
2
1
0
−1
−2
−6
−4
−2
0
2
4

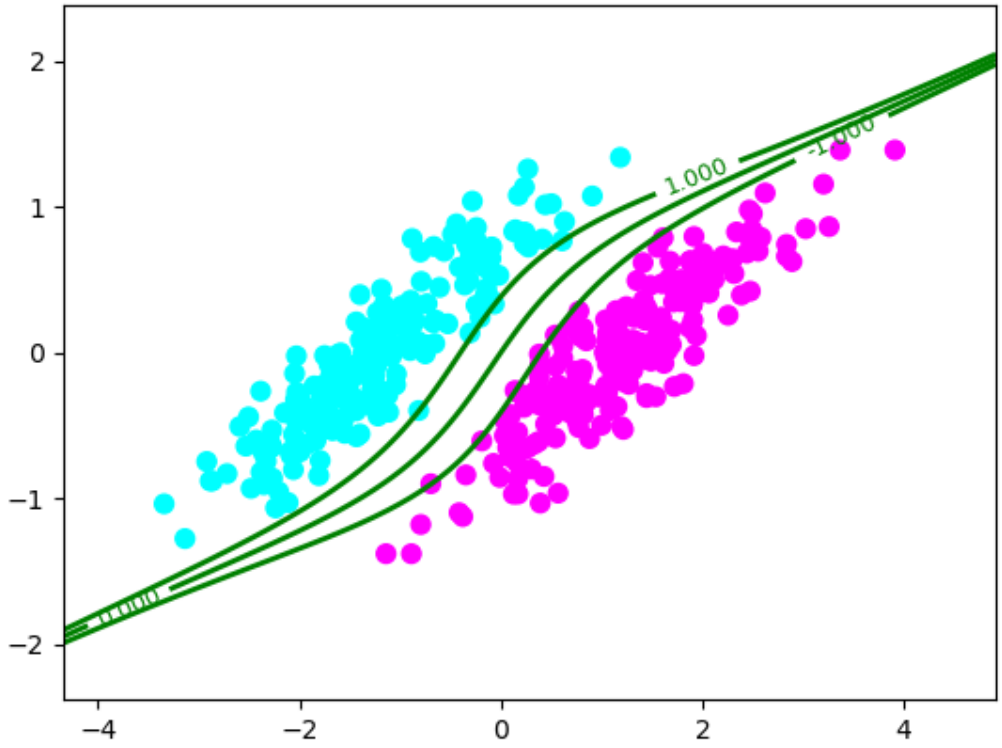
Q3. Kernel SVM

Polynomial kernel (Plots begin next page)

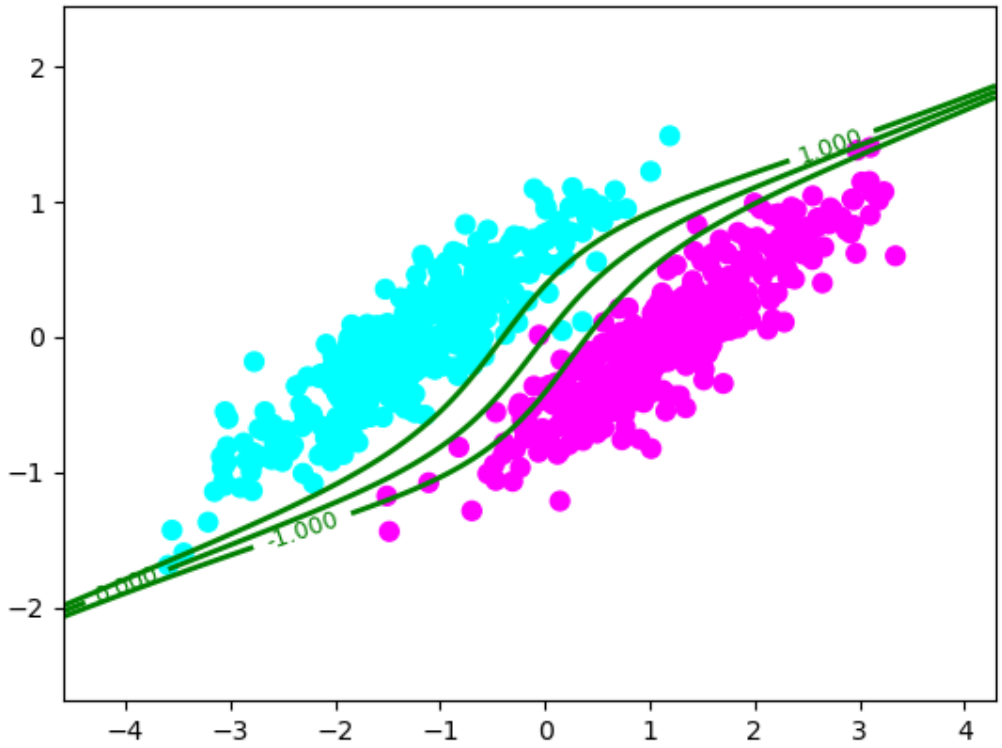
Analysis:

- So far, SVM with a polynomial kernel fits best on all the data.
- The accuracy for ls and nls is comparable to previous methods (logistic regression and primal/dual form SVM without kernels)
- The accuracy for nonlin data is surprisingly very high. However, on giving it some more thought, this result is intuitive. The data is not linear. Previously we saw logistic regression (with polynomial basis function) had given the best accuracy (even though it was pretty bad). Hence, extending that finding to use a polynomial kernel should fit the data really well (as is seen in the plots)
- Generally speaking, polynomial kernels increase the computation time for training and prediction, so using it for ls data is a definite overkill and should not be used.

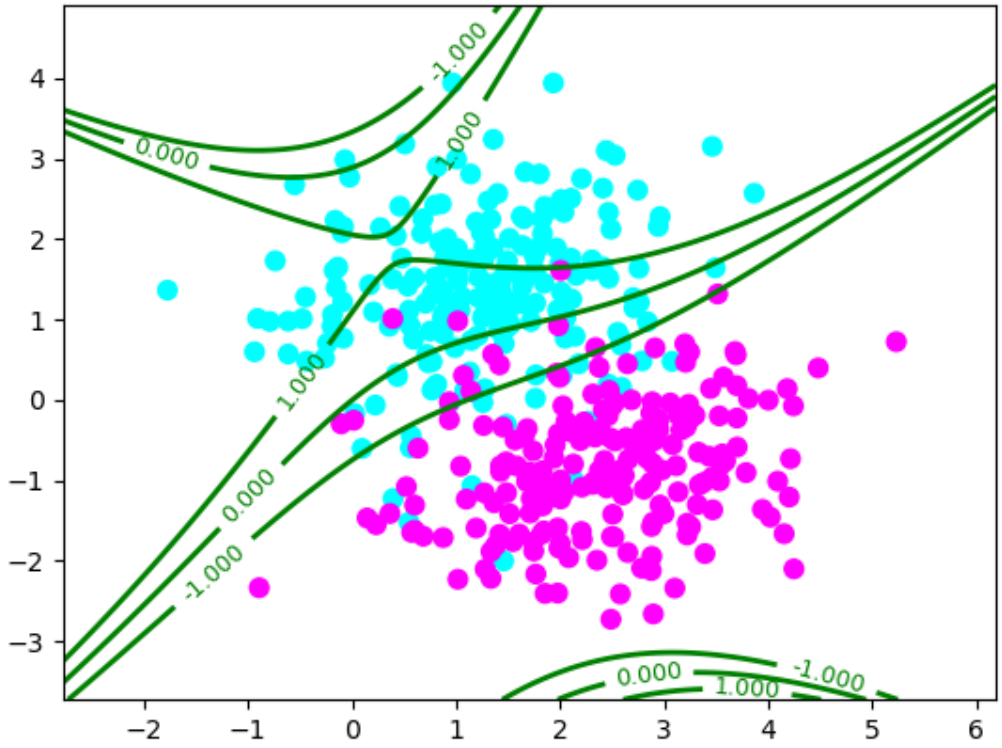
SVM dual-polynomial Train on ls data, C=10; misclassification = 0



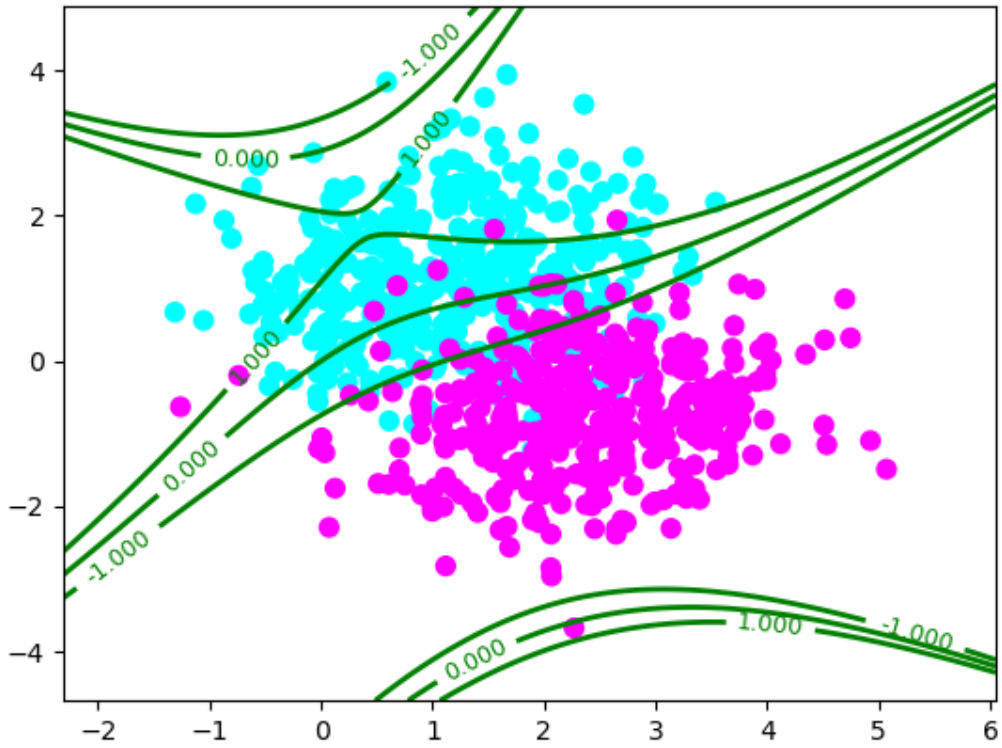
SVM dual-polynomial Validate on ls data, C=10; misclassification = 3



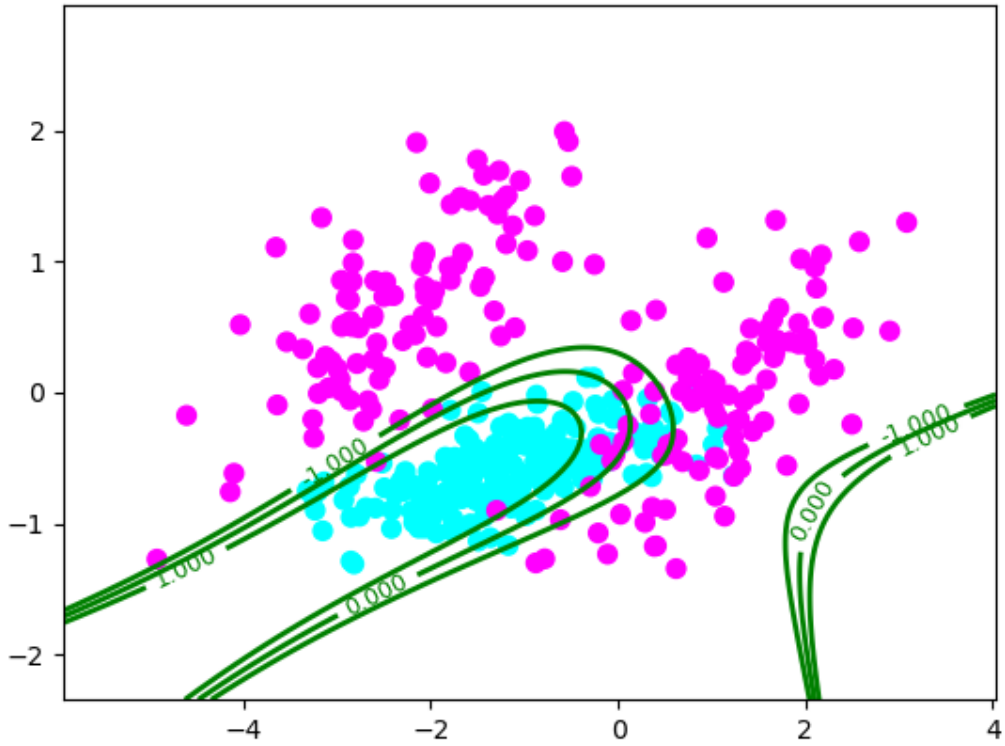
SVM dual-polynomial Train on nls data, C=10; misclassification = 55



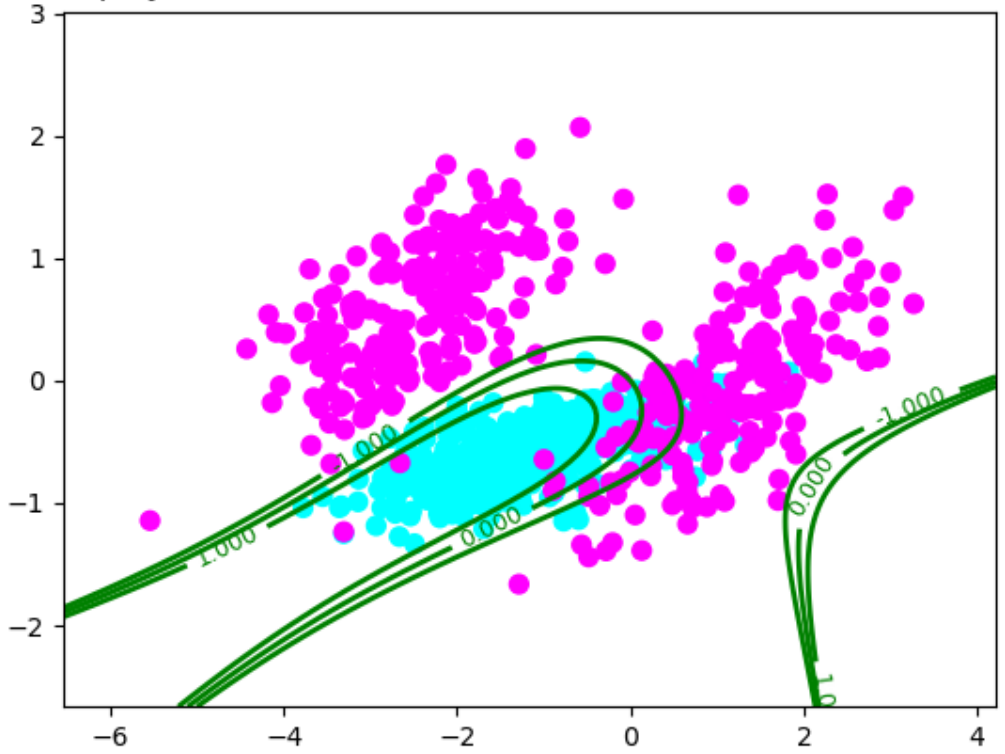
SVM dual-polynomial Validate on nls data, C=10; misclassification = 131



SVM dual-polynomial Train on nonlin data, C=10; misclassification = 29

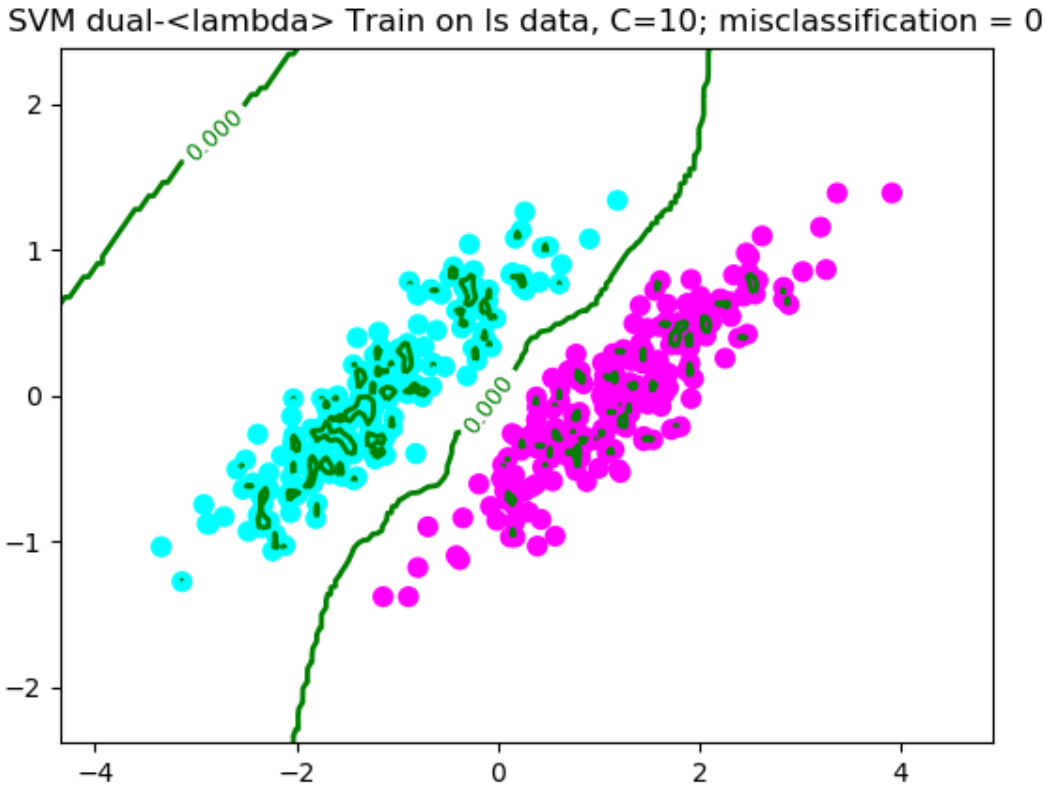


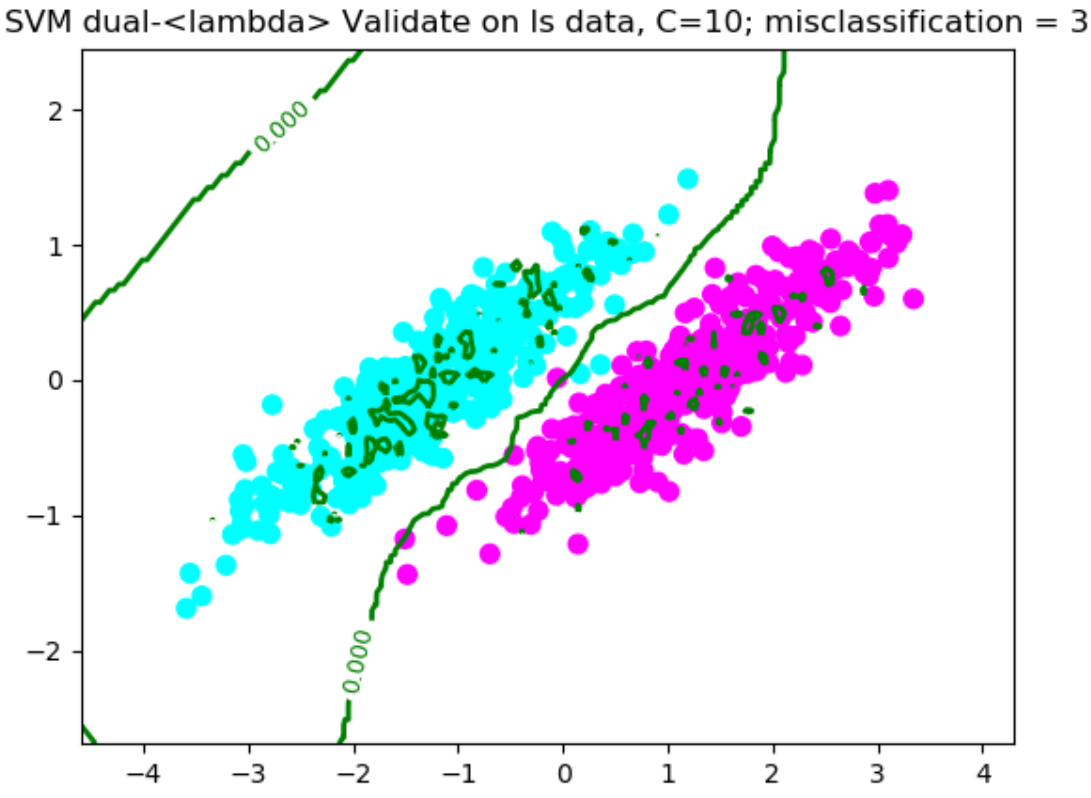
SVM dual-polynomial Validate on nonlin data, C=10; misclassification = 64

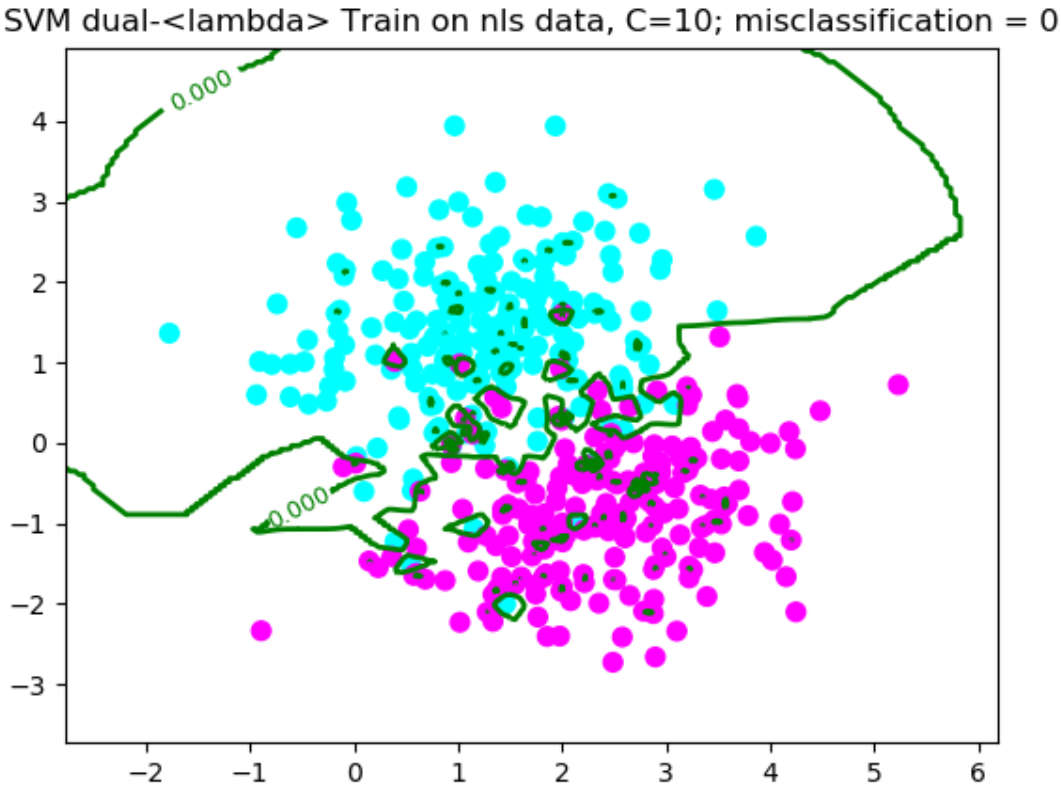


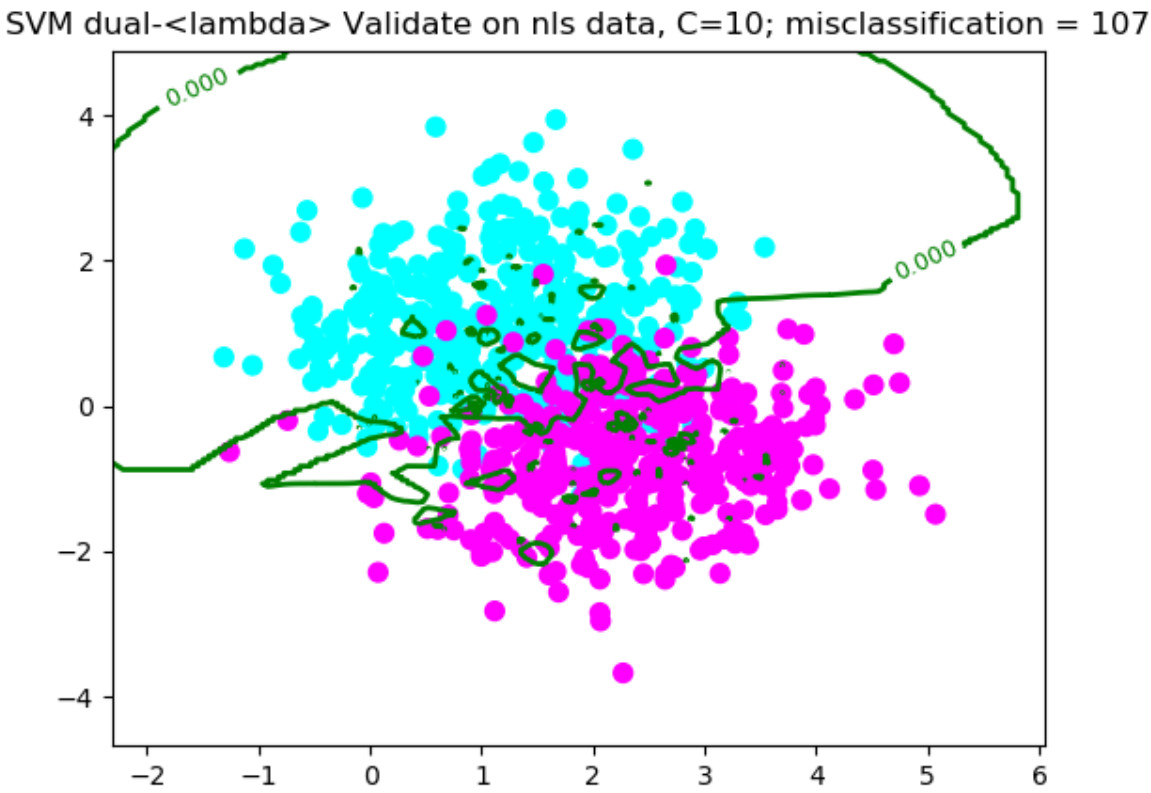
Gaussian kernel with sigma = 0.05 (plots on next page)

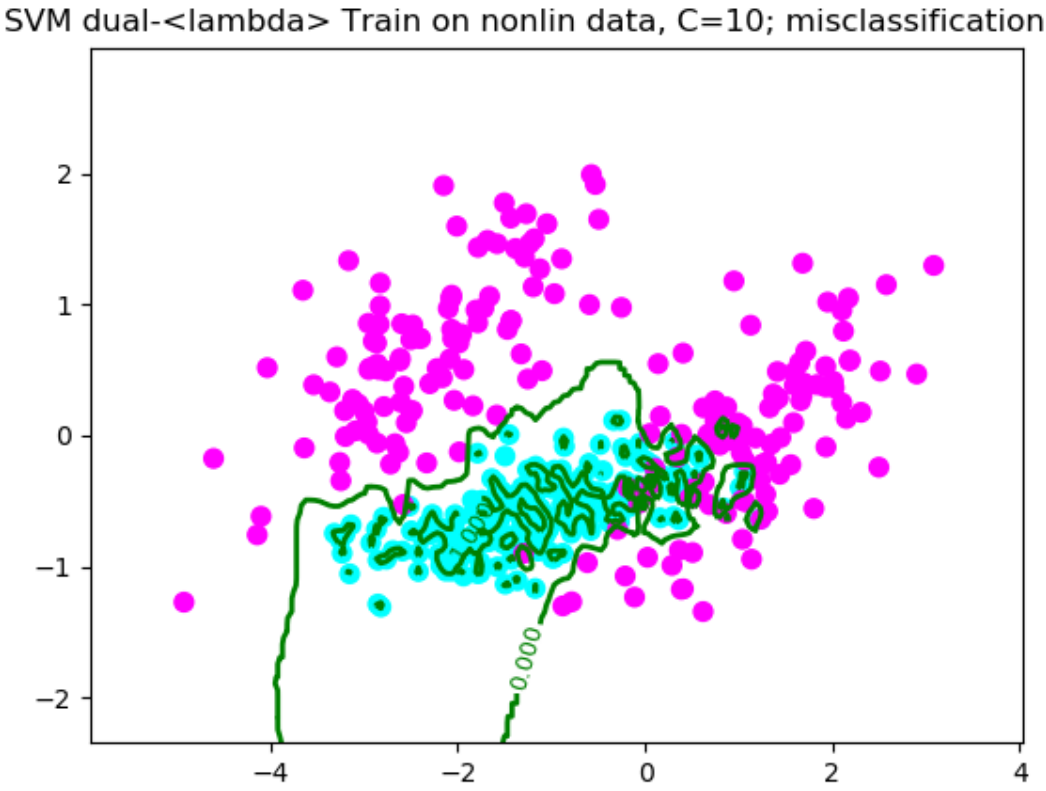
- For ls data:
    - Even though the misclassification is comparable to that of the other previous models, it seems like it has overfit the data and may not perform well in the test data
- For nls nonlin data
    - This analysis is similar to that of ls data. Even though it fits well, it seems like the model has overfit

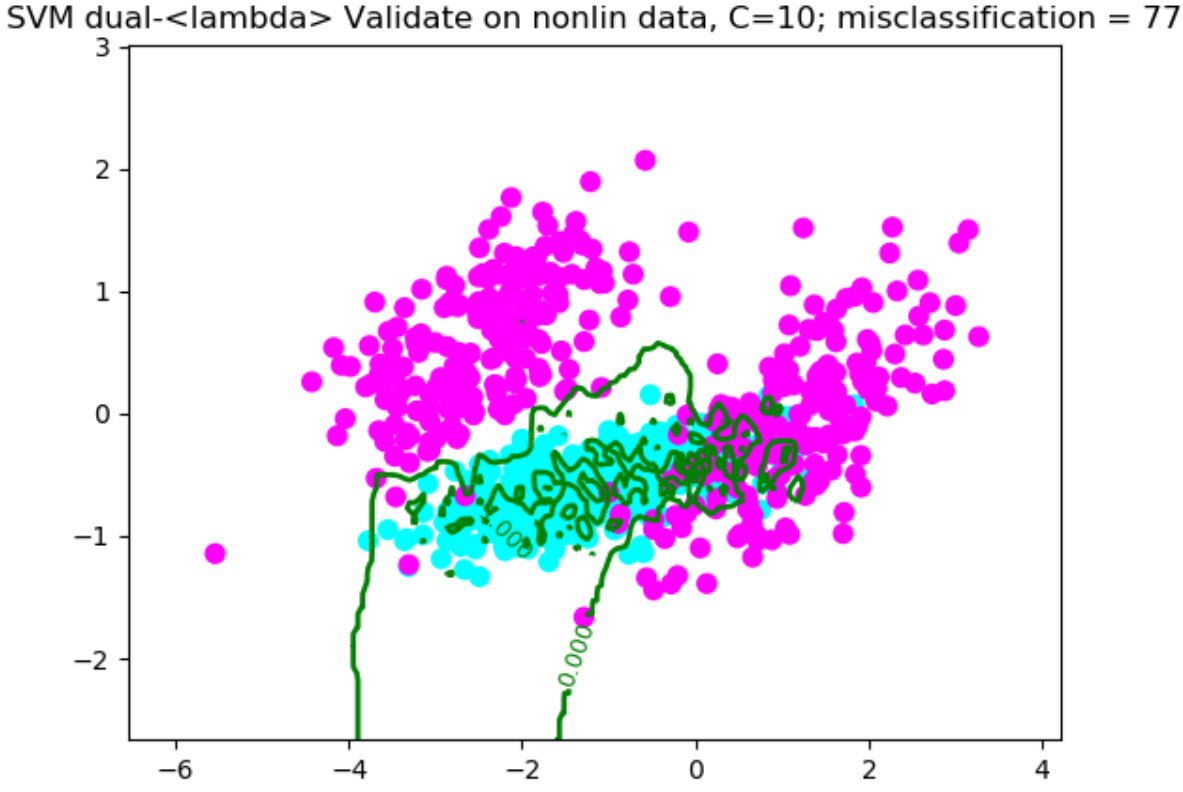

SVM dual-<lambda> Train on ls data, C=10; misclassification = 0
0.000
0.000
2
1
0
−1
−2
−4
−2
0
2
4


SVM dual-<lambda> Validate on ls data, C=10; misclassification = 3
0.000
0.000
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4


SVM dual-<lambda> Train on nls data, C=10; misclassification = 0
0.000
0.000
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5
6


SVM dual-<lambda> Validate on nls data, C=10; misclassification = 107
0.000
0.000
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5
6

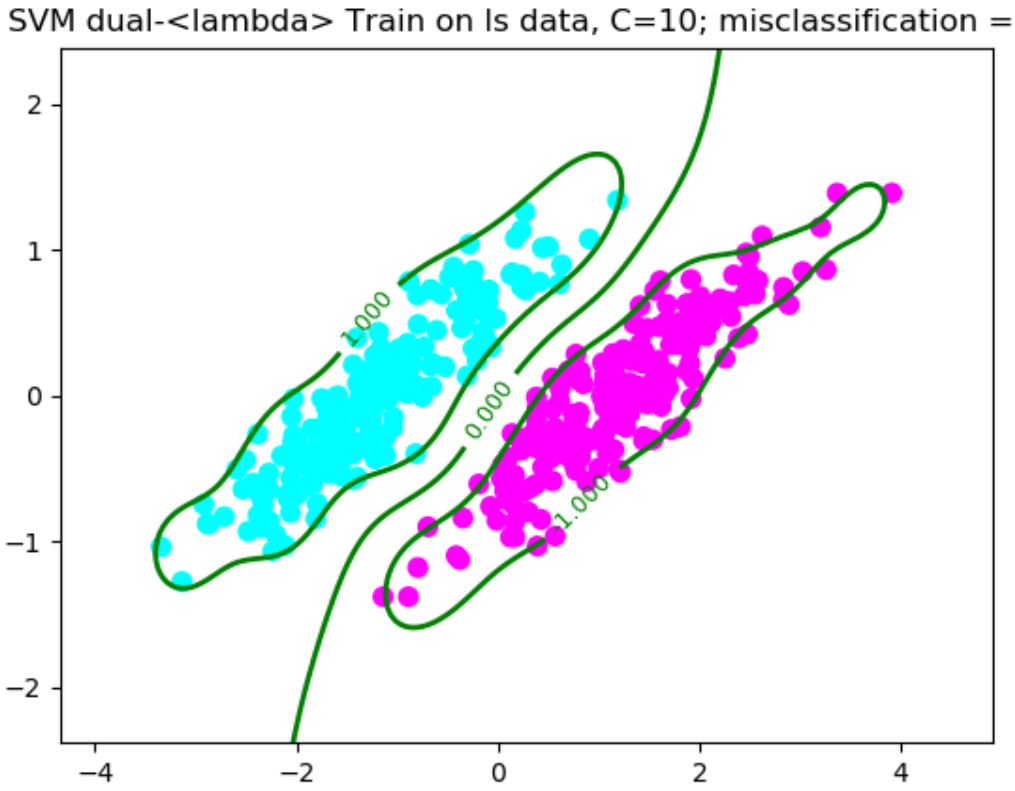

SVM dual-<lambda> Train on nonlin data, C=10; misclassification = 2
2
1
0
−1
−2
−4
−2
0
2
4
0.000
0.000

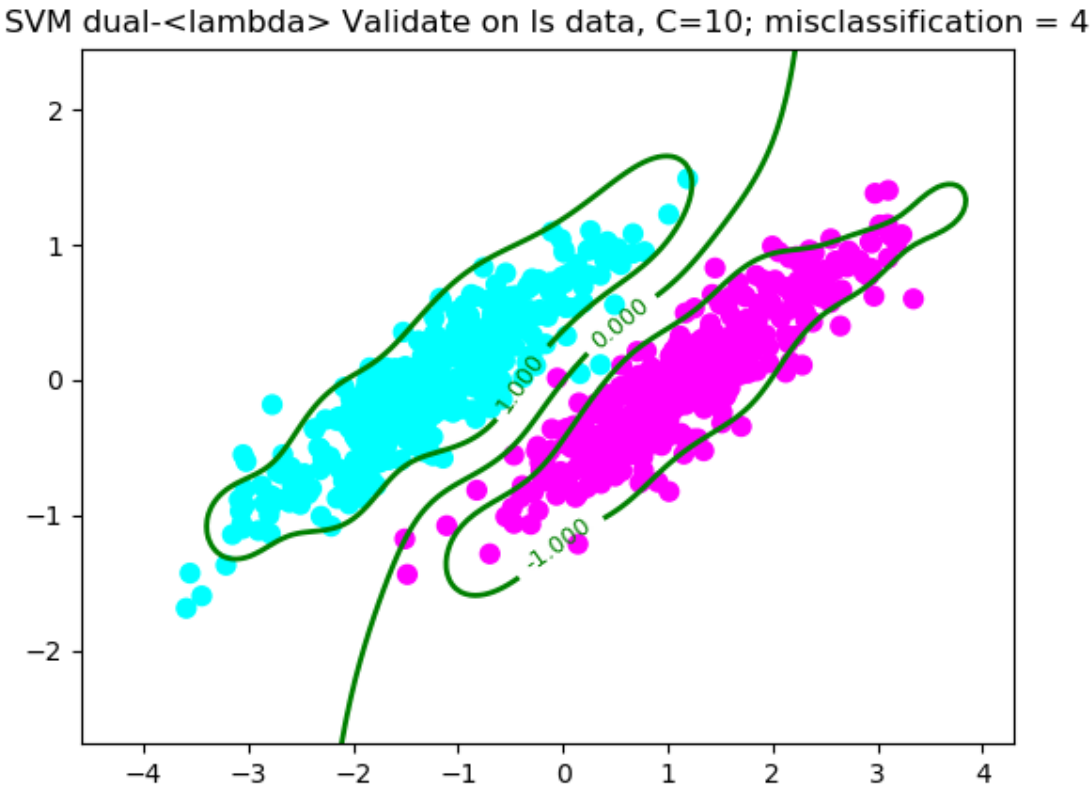

SVM dual-<lambda> Validate on nonlin data, C=10; misclassification = 77
3
2
1
0
−1
−2
−6
−4
−2
0
2
4
0.000
0.000

Gaussian kernel with sigma = 0.5 (plots on next page)

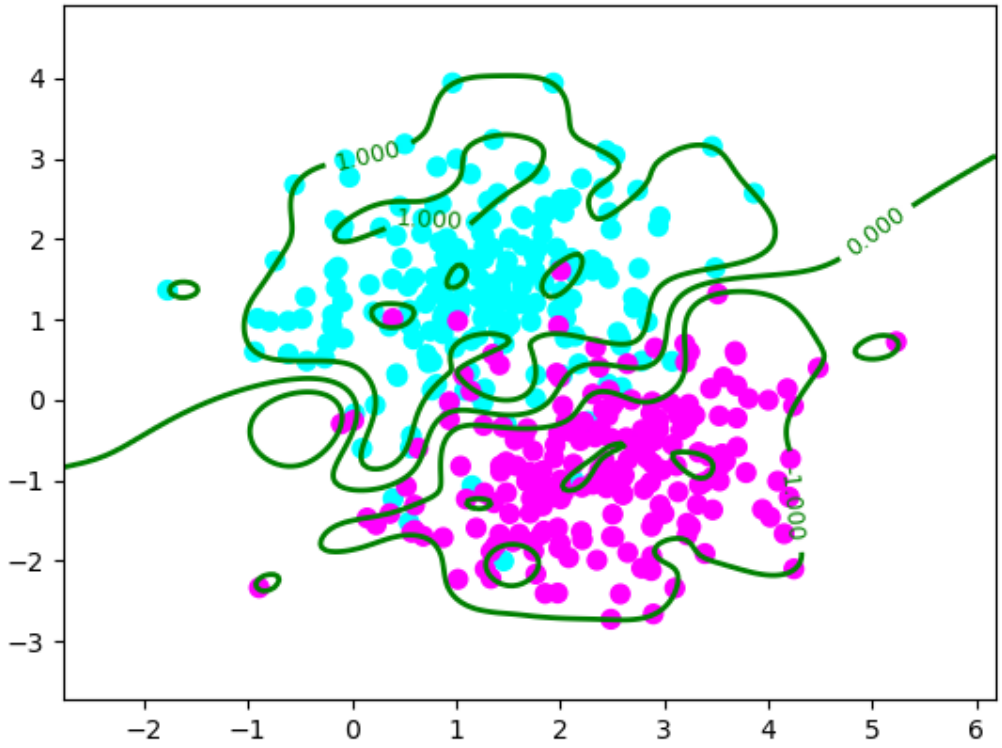
Analysis:

- This model fits the best across all the previous methods and across all datasets
- For ls data
    - It fits comparable to other models (which were already really good). So using a gaussian kernel may be an overkill
- For nls and nonlin data
    - One could tweak the C and sigma values a little to get a better fit. It seems like a good fit (best we've seen so far) but also seems like it could have worked better with a bigger sigma (or a smaller C)

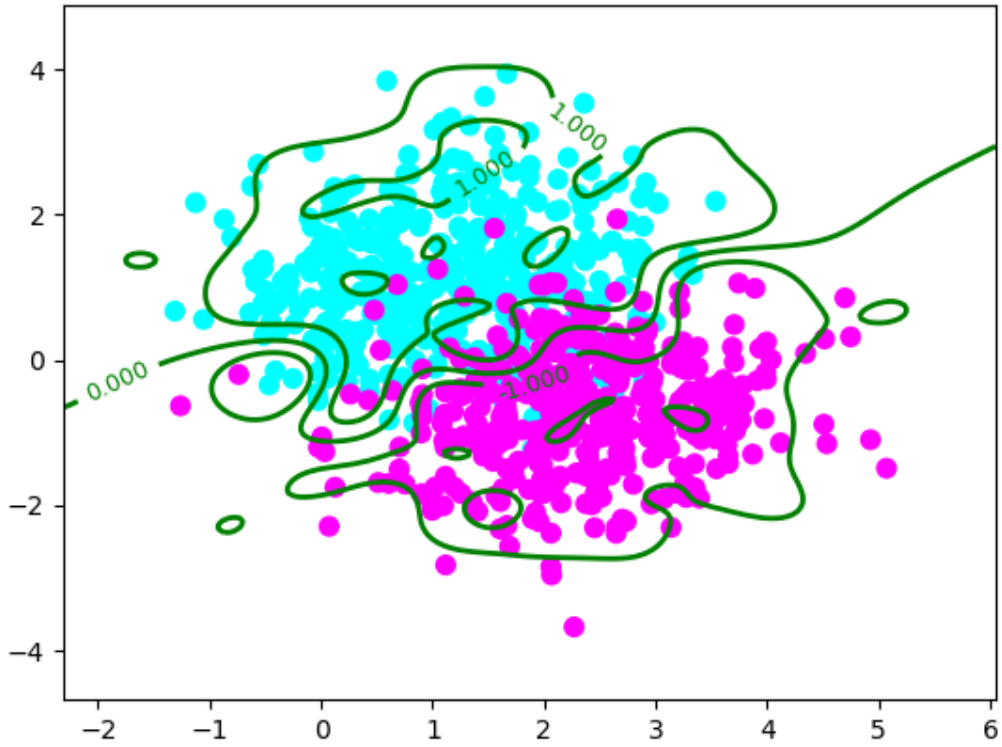

SVM dual-<lambda> Train on ls data, C=10; misclassification = 0
1.000
0.000
-1.000
2
1
0
−1
−2
−4
−2
0
2
4


SVM dual-<lambda> Validate on ls data, C=10; misclassification = 4
1.000
0.000
-1.000
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4

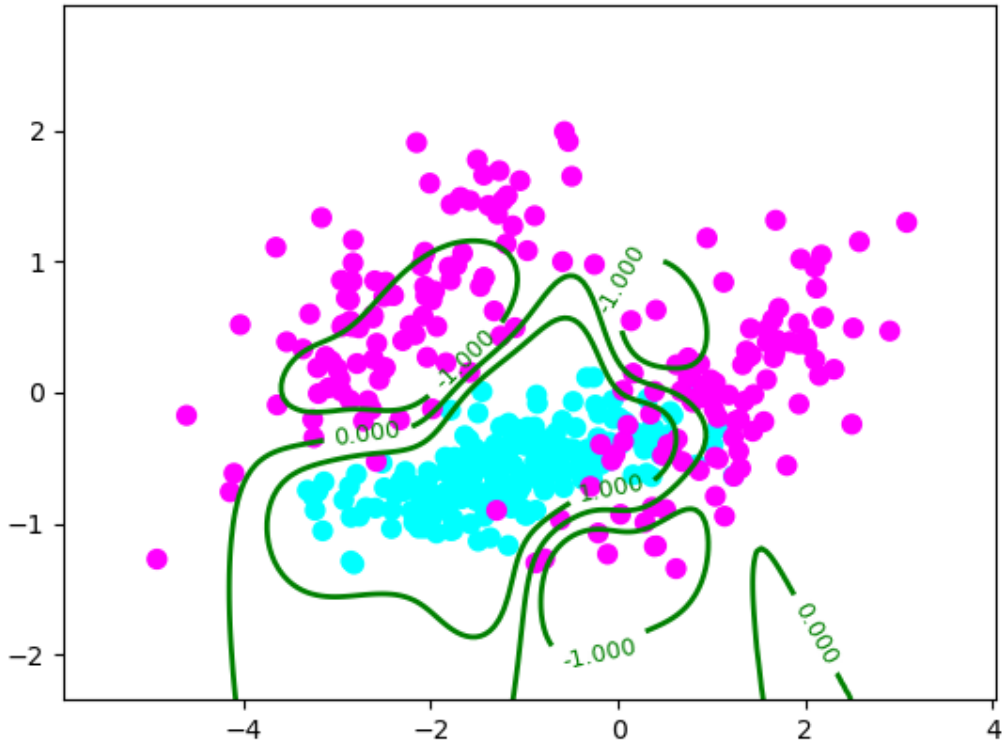
SVM dual-<lambda> Train on nls data, C=10; misclassification = 27



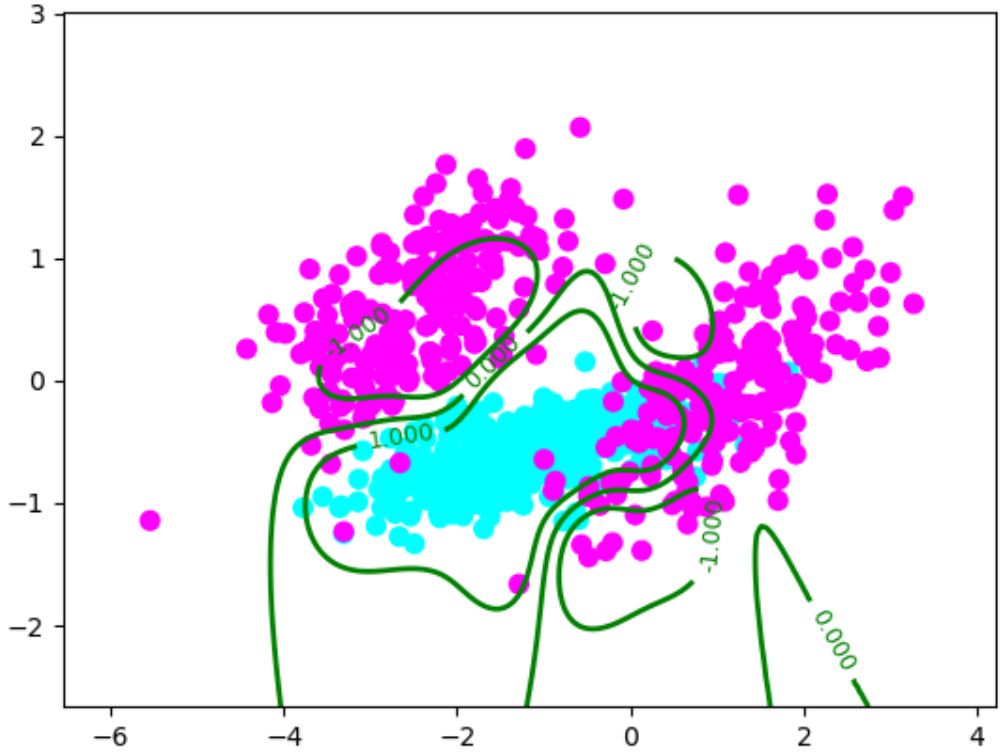
SVM dual-<lambda> Validate on nls data, C=10; misclassification = 78



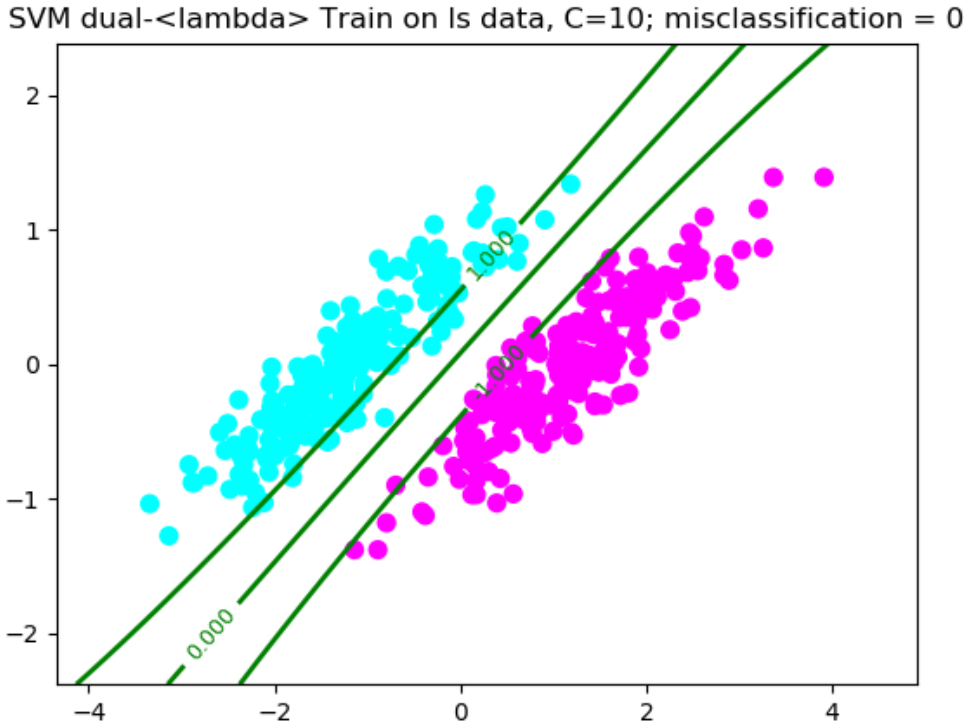
SVM dual-<lambda> Train on nonlin data, C=10; misclassification = 23

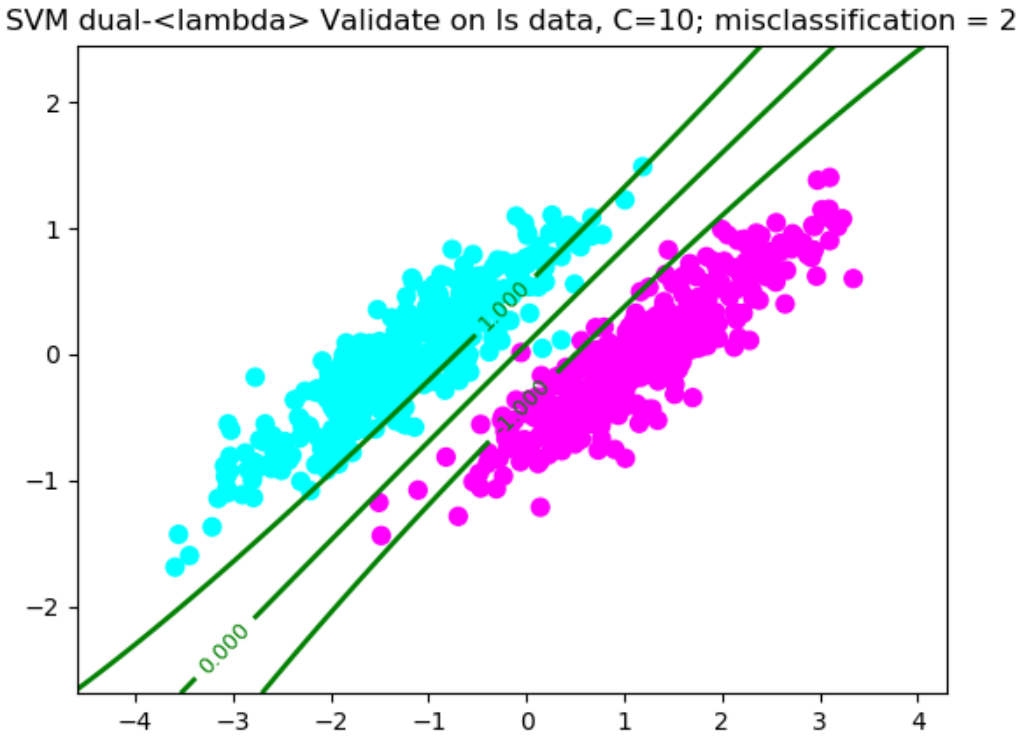


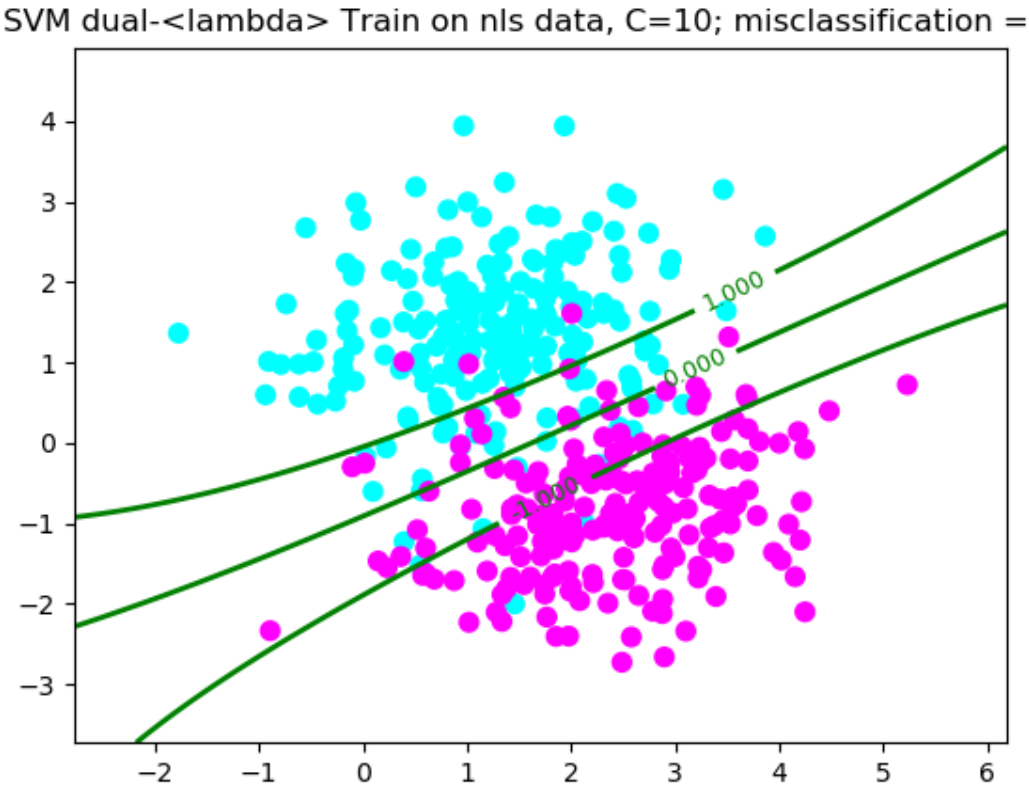
SVM dual-<lambda> Validate on nonlin data, C=10; misclassification = 74

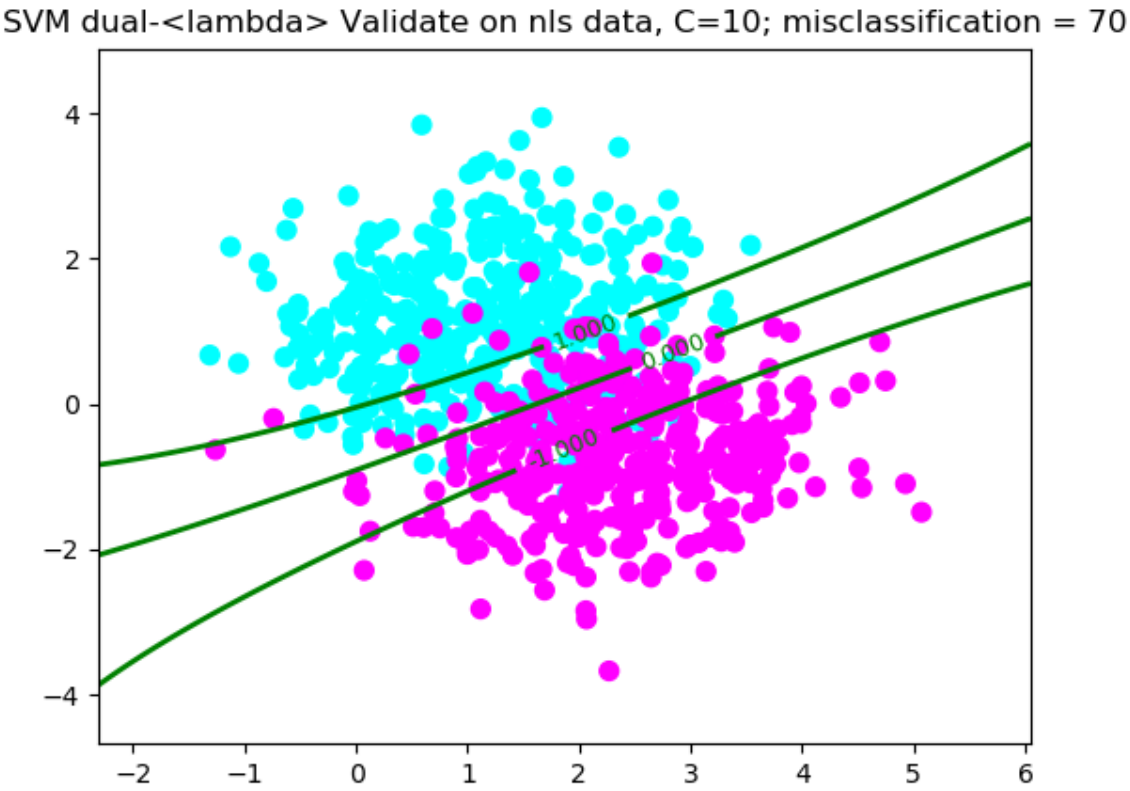


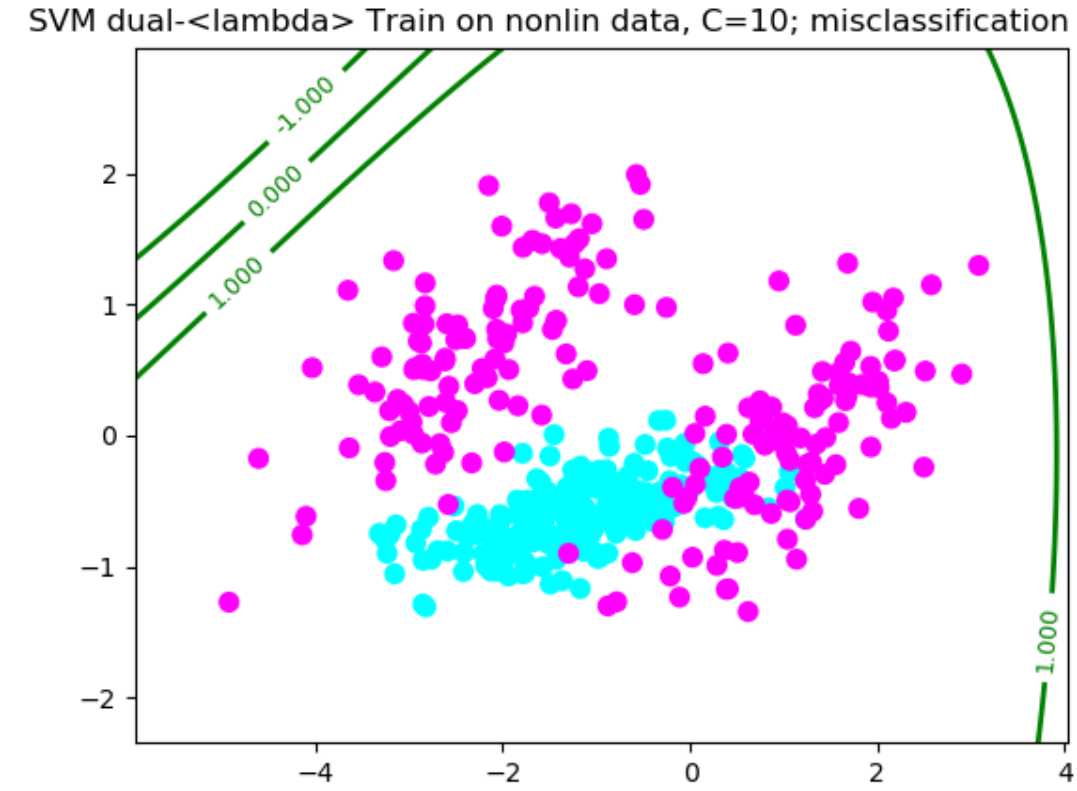
Gaussian kernel with sigma = 5
The sigma value is way too big for any concrete classification. The Gaussian doesn't fit the data correctly and grossly misclassifies for all datasets

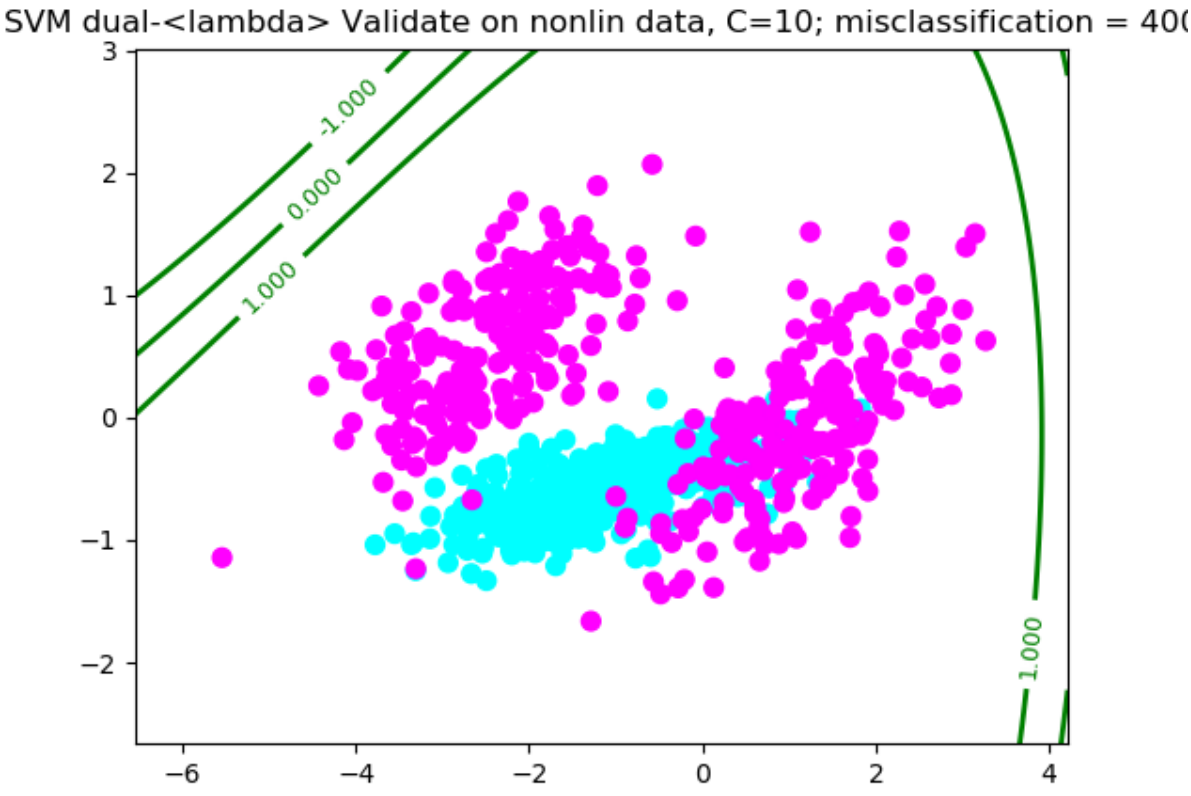





SVM dual-<lambda> Train on nls data, C=10; misclassification = 29
1.000
0.000
-1.000


SVM dual-<lambda> Validate on nls data, C=10; misclassification = 70
1.000
0.000
-1.000


SVM dual-<lambda> Train on nonlin data, C=10; misclassification = 200
-1.000
0.000
1.000
1.000
2
1
0
−1
−2
−4
−2
0
2
4


SVM dual-<lambda> Validate on nonlin data, C=10; misclassification = 40(
-1.000
0.000
1.000
1.000
3
2
1
0
−1
−2
−6
−4
−2
0
2
4