

# Support Vector Machines

Emma Lathouwers, Mahdieh Malekian, Ish Mukul, Ulas Ozdemir

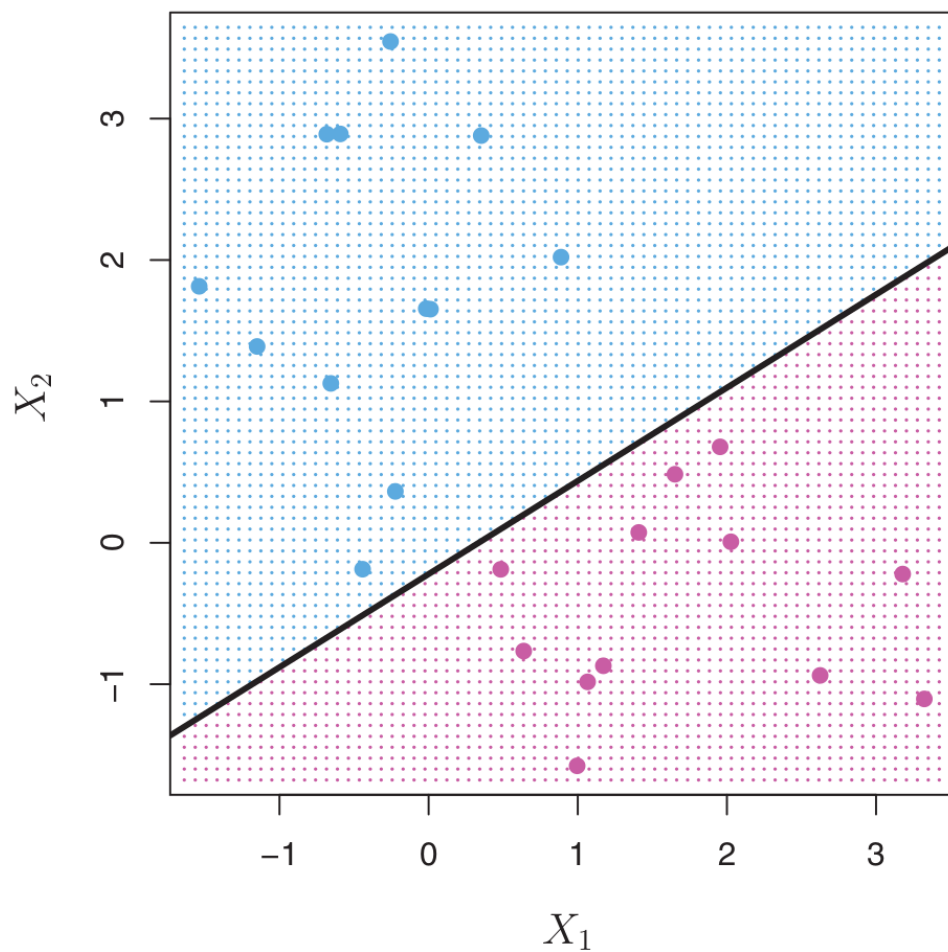
AQM

March 4, 2020

# Outline

- SVM Theory (Ulas)
- Code Walkthrough (Ish)
- Results and Discussion (Emma)

# Hyperplane Classification



- N training examples

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$$

$$y_i \in \{\pm 1\}$$

- Separating hyperplane

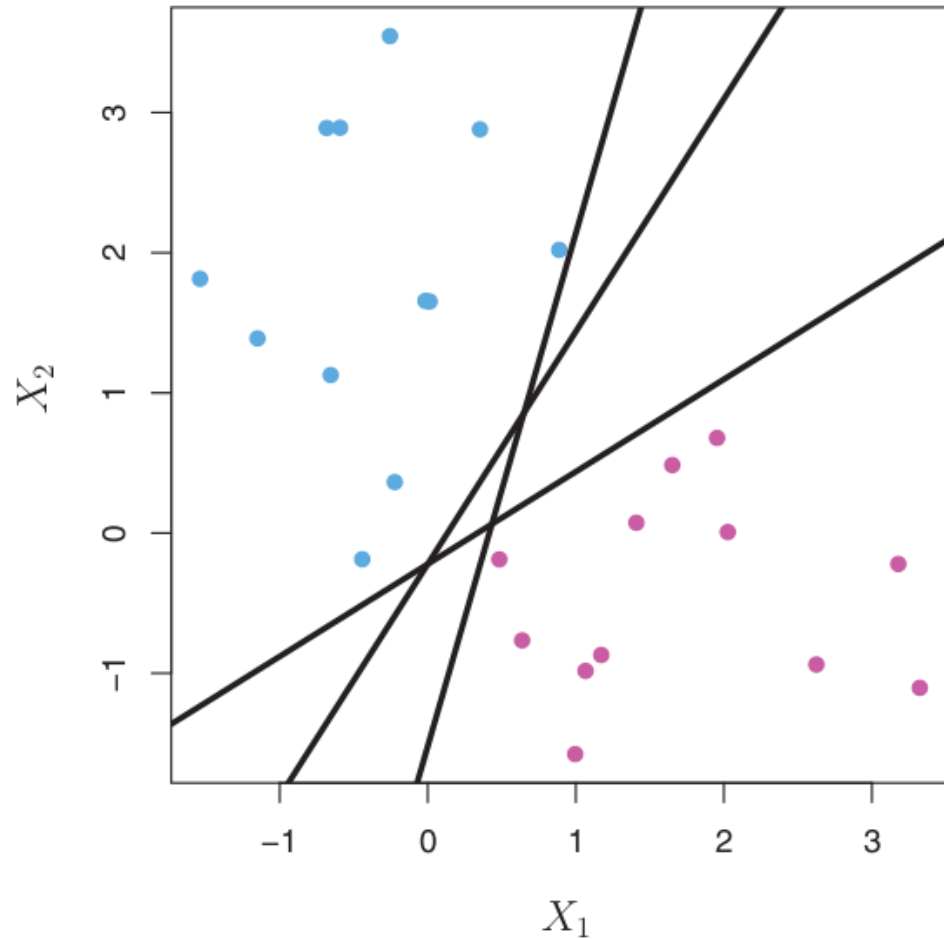
$$f(x) = x^T \beta + \beta_0 = 0$$

- Classification

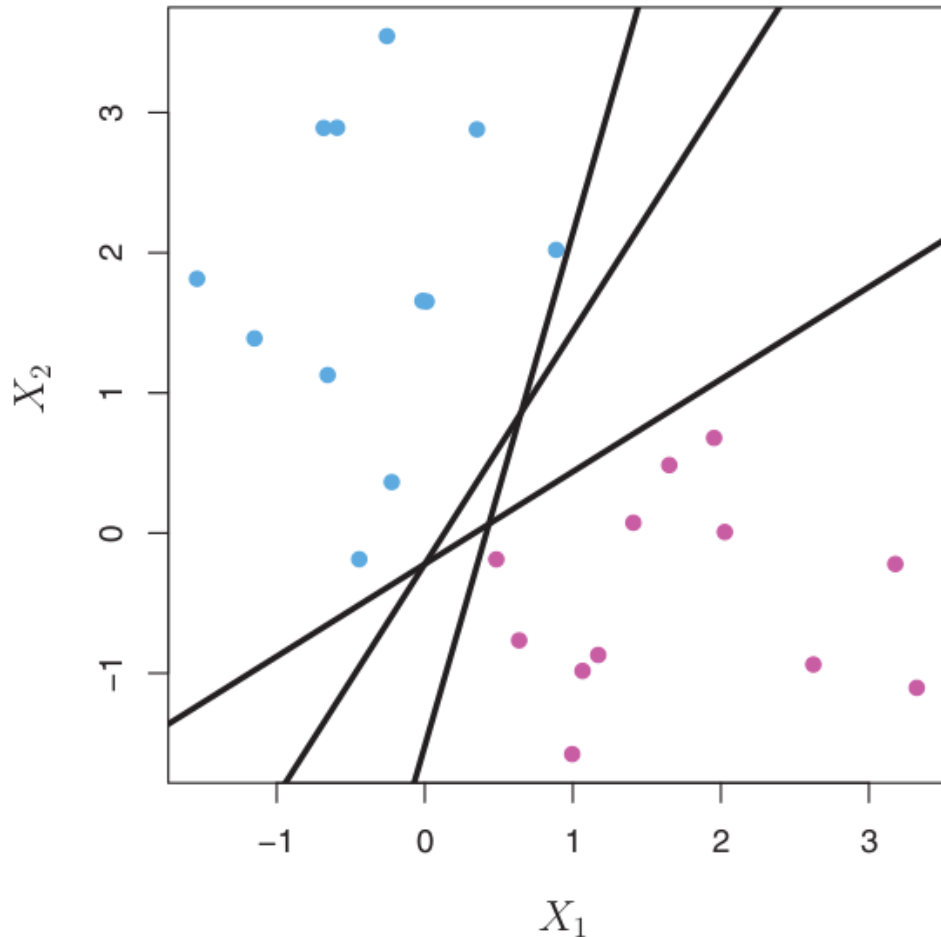
$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} > 0 \text{ if } y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} < 0 \text{ if } y_i = -1$$

# Hyperplane Classification



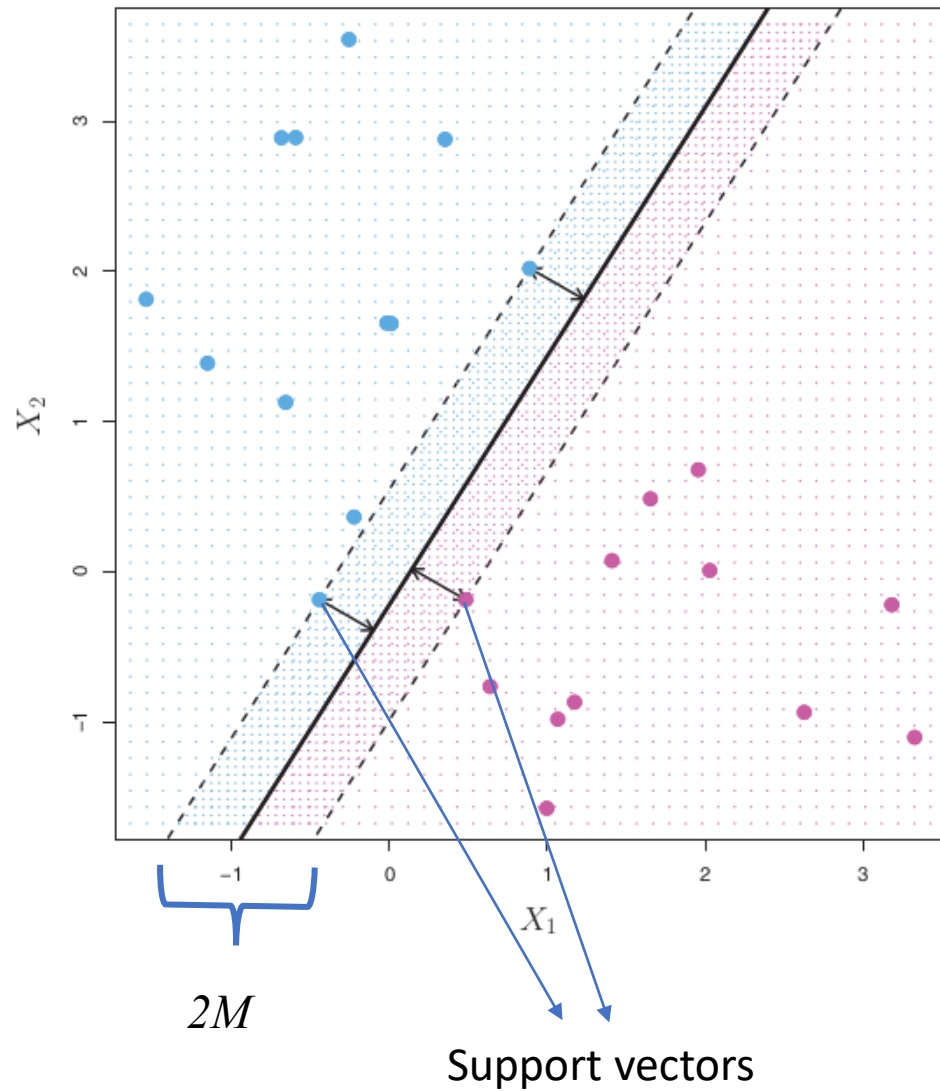
# Hyperplane Classification



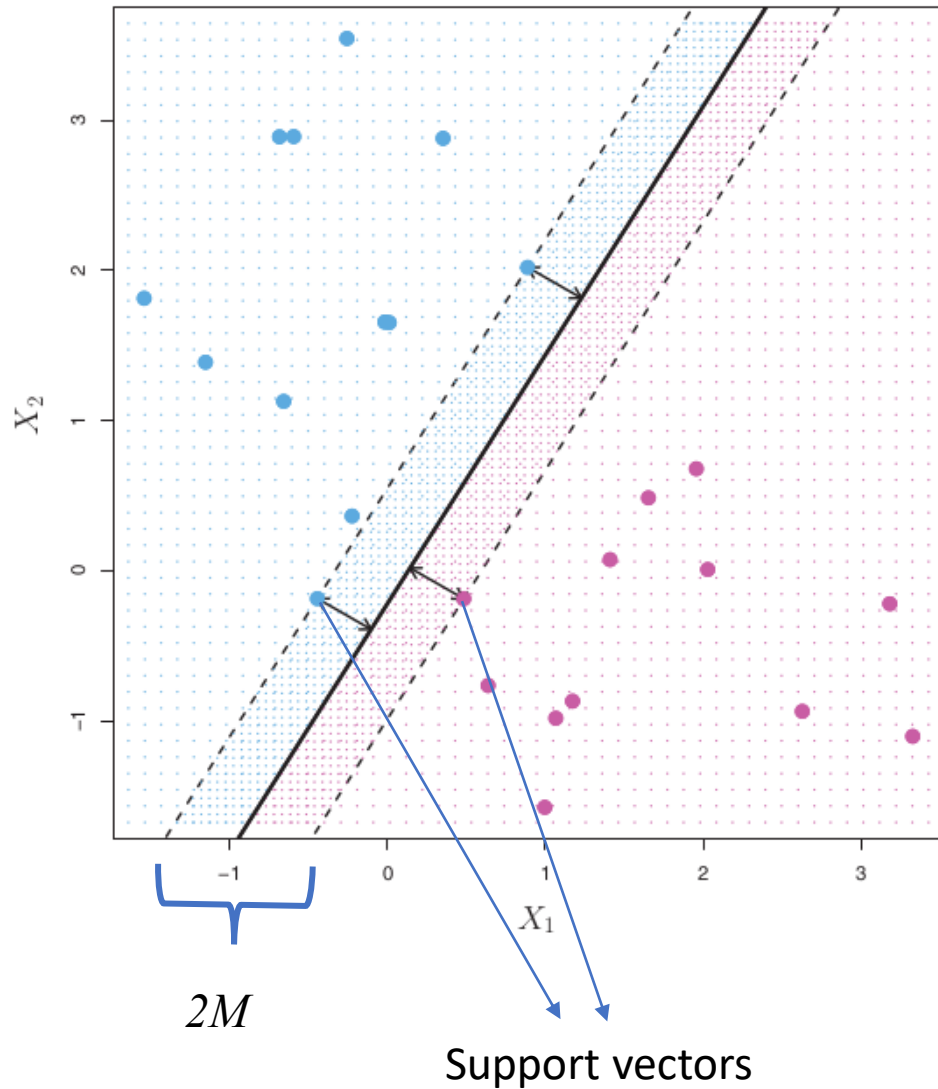
- Infinite numbers of hyperplanes.
  - Hard to predict the best separating plane.
- Does not work well for the test set.
  - Solution only exists for separable case.

$$f(x) = x^T \beta + \beta_0 \text{ with } y_i f(x_i) > 0 \forall i$$

# Maximal Margin Classifier (Hard Margin)



# Maximal Margin Classifier (Hard Margin)

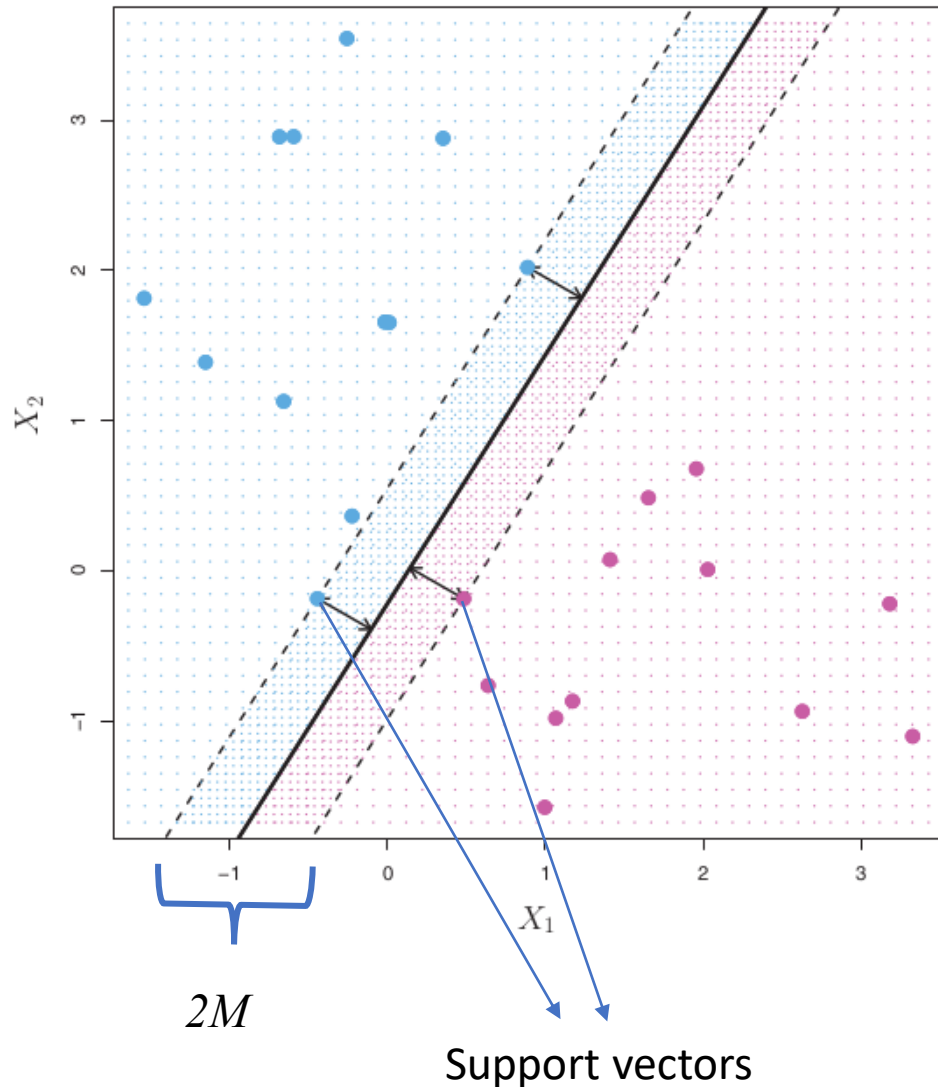


- A special case of SVM
- Separating hyperplane is the farthest from the training set

$$\max_{\beta, \beta_0, \|\beta\|=1} M$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq M, i = 1, \dots, N$$

# Maximal Margin Classifier (Hard Margin)

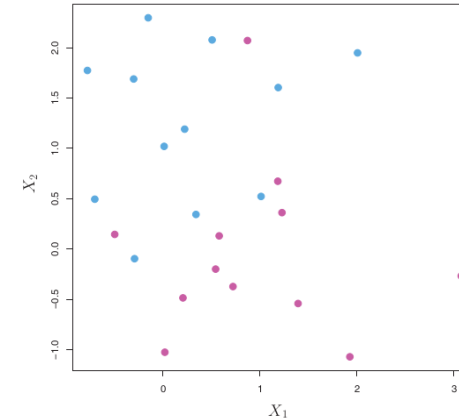


- A special case of SVM
- Separating hyperplane is the farthest from the training set

$$\max_{\beta, \beta_0, \|\beta\|=1} M$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq M, i = 1, \dots, N$$

- Sensitive to individual observations
- No solution for non-separable case.

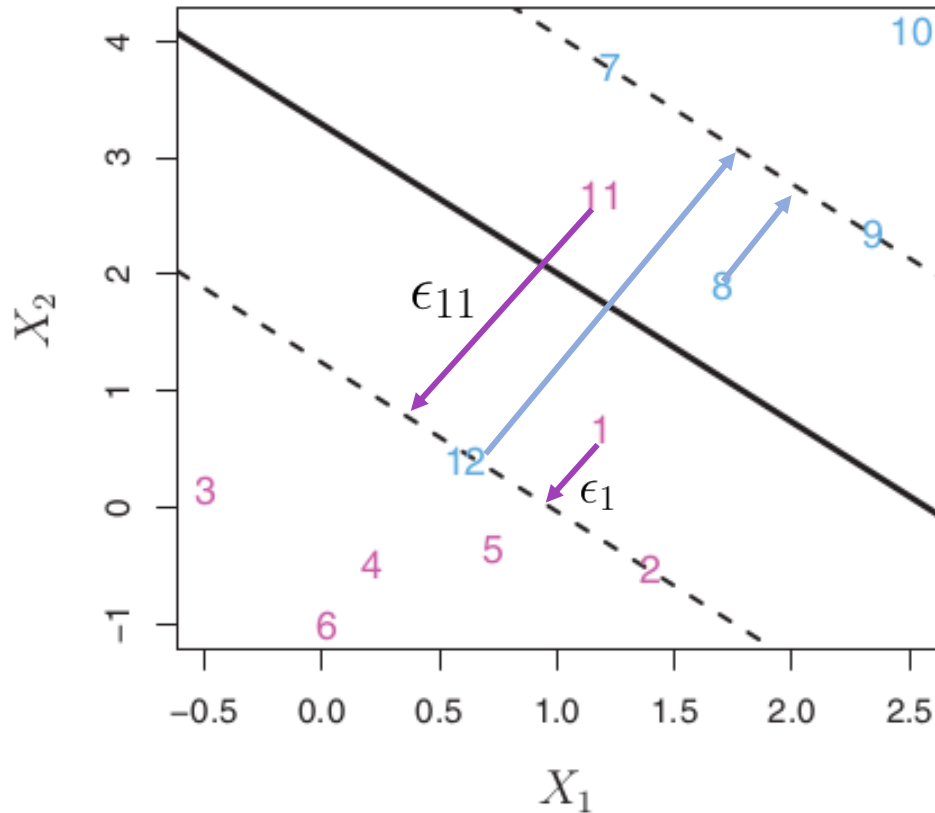


- Need a more robust classifier for each observation
- Better classification of most of the training observations



# Support Vector Classifier (Soft Margin)

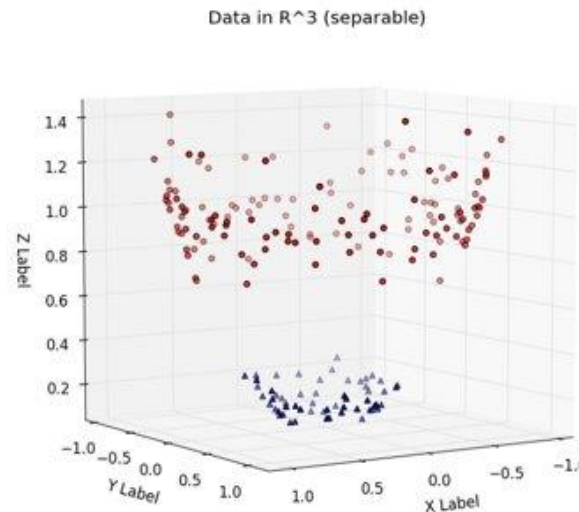
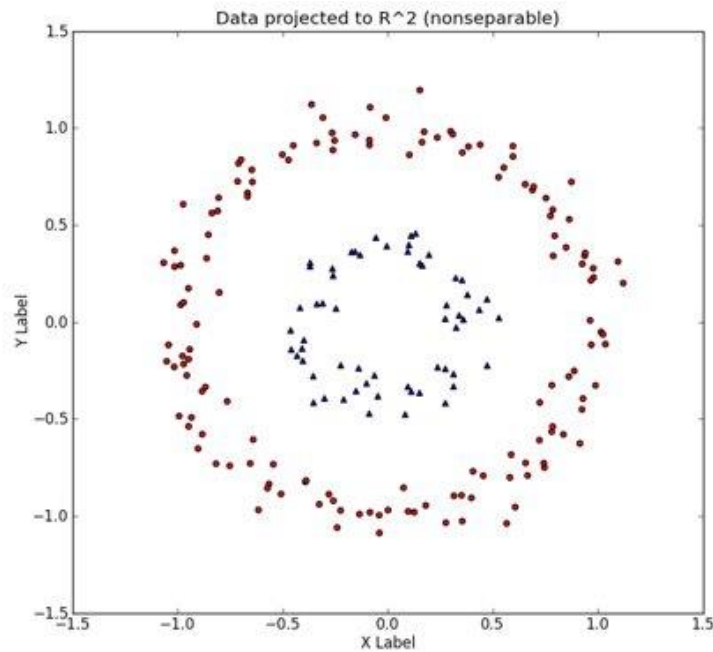
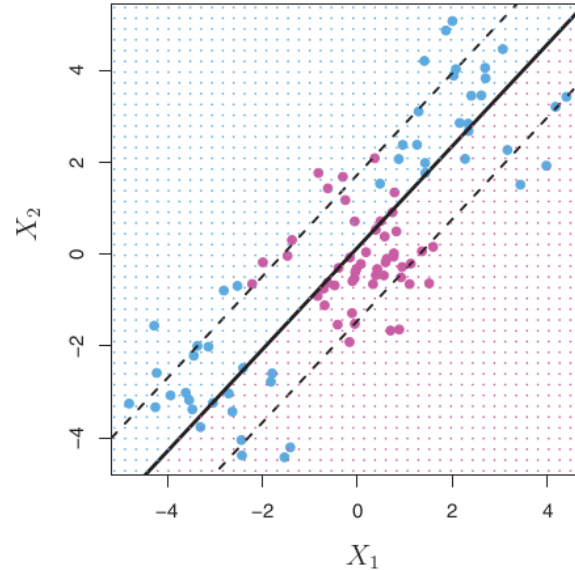
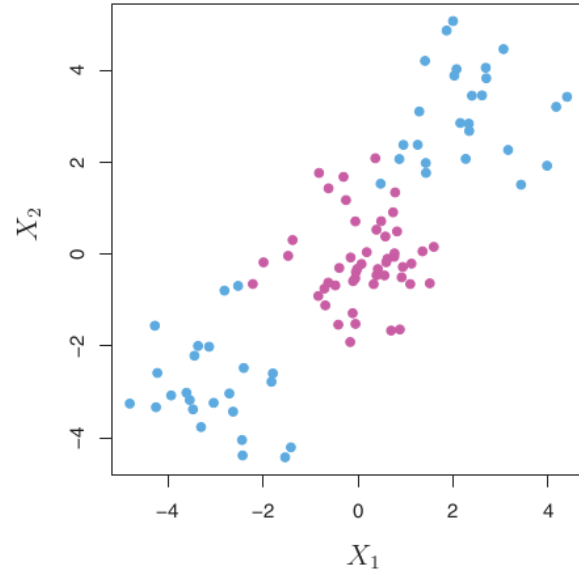
- Allow some points to be classified incorrectly.



$$\begin{aligned}
 & \underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} && M \\
 & \text{subject to} && \sum_{j=1}^p \beta_j^2 = 1 \\
 & && y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \\
 & && \epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C
 \end{aligned}$$

$\epsilon_i$  = Slack variable  
 $C$  = Tuning parameter

# Non Linear Decision Boundary



- Enlarge the feature space

$$X_1, X_2, \dots, X_p \longrightarrow X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$$

- Map all decision boundary to a space where linear separation is possible.
- Computationally expensive (Large feature set)
- Can lead to overfitting
- Not standardized -> Different combinations features

Use a kernel function

- Can be precomputed
- Smaller feature size
- Polynomial kernel

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$

- Gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = \exp\left\{-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2\right\}$$

# Incorporating Kernels into code

- The main difference is that we need to solve for  $\alpha$  from modified Lagrangian dual:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^N \alpha_i = 0 \end{aligned}$$

- This is a quadratic programming which cvxopt solves, except that cvxopt solves equations of form:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2} \alpha^T P \alpha + q^T \alpha \\ \text{subject to} \quad & G \alpha \preceq h, \\ & A \alpha = b \end{aligned}$$

- Need to choose  $P, q, G, h, A, b$  so that Lagrangian dual turns into the cvxopt form.

$$P = \begin{pmatrix} & \vdots & \\ \dots & y_i y_j K(x_i, x_j) & \dots \\ & \vdots & \end{pmatrix}_{N \times N} \quad q = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix}_{N \times 1}$$

$$G = \begin{pmatrix} -I_{N \times N} \\ \hline I_{N \times N} \end{pmatrix}_{2N \times N} \quad h = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \hline C \\ \vdots \\ C \end{pmatrix}_{2N \times 1}$$

$$A = y^T \quad b = 0$$

# Code Walkthrough

```
def compute_cost(ww, xx, yy):  
    # calculate hinge loss  
    distances = 1 - yy * (xx @ ww)  
    distances[distances < 0] = 0 # equivalent to max(0, distance)  
    hinge_loss = C * (distances.sum() / xx.shape[0])  
  
    # calculate cost  
    cost = 0.5 * (ww @ ww) + hinge_loss  
    return cost  
  
def calculate_cost_gradient(ww, xx, yy):  
    dw = np.zeros(len(ww))  
    distance = 1 - (yy * (xx @ ww))  
    if distance <= 0:  
        di = ww  
    else:  
        di = ww - (C * yy * xx)  
    dw += di  
    return dw
```

# Code Walkthrough

```
def gradient_descent(xx, yy, ww, max_iters):  
    costs = []  
    # stochastic gradient descent  
    for epoch in range(1, max_iters):  
        for i in range(xx.shape[0]):  
            ascent = calculate_cost_gradient(ww, xx[i], yy[i])  
            ww = ww - (learning_rate * ascent)  
  
            cost = compute_cost(ww, xx, yy)  
            costs.append(cost)  
  
    return costs, ww
```

# Code Walkthrough

```
start_time = timer()
print('=====')
# Cost function, Gradient calculation and Gradient descent functions defined on the top
# Search for weights start here
Costs, Weight = gradient_descent(X_train, y_train, weights, max_epochs)

# Y prediction from fitted parameter
# numpy sign function gives sign of output result. Since we are interested the sign of the dot product product of
#  $X @ w$ , not the actual value. Remember, +1 is 1 and -1 is 0 in our convention.
y_score_man = (X_test @ Weight)
y_test_pred_man = np.sign(X_test @ Weight)
acc_man = acc(y_test, y_test_pred_man) # Accuracy from confusion matrix
fpr_man, tpr_man, _ = roc_curve(y_test, y_score_man) # false positive rate, true positive rate
auc_man = roc_auc_score(y_test, y_score_man) # ROC area under curve
f1_man = f1_score(y_test, y_test_pred_man)

end_time = timer()
time_manual = end_time - start_time
print('Accuracy from manual computation is %4.3f percent' % (100 * acc_man))
print('F1 score from manual computation is %4.3f.' % f1_man)
print('ROC-AUC from manual computation is %4.3f. \n' % auc_man)
print('Time taken = %f seconds. \n' % time_manual)
# print('Diagnostic check:')
# print('Weights from manual linear SVM are ')
# print(Weight)
print('=====\\n\\n')
```

# Code Walkthrough

```
Anaconda Powershell Prompt (anaconda3)

=====
DataSet split into Training:Test = 70:30 ratio.

Accuracy of TRAINING SET with rbf kernel is 100.000 percent.
Accuracy of TEST SET using rbf kernel is 94.634 percent.
F1 score of TEST SET using rbf kernel is 0.931.
ROC-AUC of TEST SET using rbf kernel is 0.983.

Accuracy of TRAINING SET with linear kernel is 98.117 percent.
Accuracy of TEST SET using linear kernel is 95.122 percent.
F1 score of TEST SET using linear kernel is 0.935.
ROC-AUC of TEST SET using linear kernel is 0.990.

Accuracy of TRAINING SET with poly kernel is 99.372 percent.
Accuracy of TEST SET using poly kernel is 94.634 percent.
F1 score of TEST SET using poly kernel is 0.929.
ROC-AUC of TEST SET using poly kernel is 0.994.

Time taken for processing different in-built functions = 0.514890 seconds.
=====

=====
Accuracy from manual computation is 97.561 percent
F1 score from manual computation is 0.964.
ROC-AUC from manual computation is 0.997.

Time taken = 3.531305 seconds.

Diagnostic check:
Weights from manual linear SVM are
[-2.21305987  0.07344833  0.14475808  0.04075921  0.03963185 -0.03565069
  0.14959991  0.08908605  0.10507681  0.04340588]
=====
```

# Code Walkthrough

## Bootstrapping

```
Anaconda Powershell Prompt (anaconda3)
Shape of X feature vector is (683, 9)
Shape of y vector is (683,)
0.6500732064421669 0.34992679355783307
Shape of X_train is (512, 9)
Shape of X_holdout is (171, 9)
fitting set # 0
Accuracy on first bootstrap 0.99
Accuracy on second bootstrap 0.975
Accuracy on third bootstrap 0.965
fitting set # 1
Accuracy on first bootstrap 0.965
Accuracy on second bootstrap 0.985
Accuracy on third bootstrap 0.97
fitting set # 2
Accuracy on first bootstrap 0.985
Accuracy on second bootstrap 0.98
Accuracy on third bootstrap 0.985
fitting set # 3
Accuracy on first bootstrap 0.985
Accuracy on second bootstrap 0.975
Accuracy on third bootstrap 0.97
fitting set # 4
Accuracy on first bootstrap 0.975
Accuracy on second bootstrap 0.98
Accuracy on third bootstrap 0.965
fitting set # 5
Accuracy on first bootstrap 0.985
Accuracy on second bootstrap 0.975
Accuracy on third bootstrap 0.965
fitting set # 6
Accuracy on first bootstrap 0.975
Accuracy on second bootstrap 0.975
Accuracy on third bootstrap 0.965
fitting set # 7
Accuracy on first bootstrap 0.98
Accuracy on second bootstrap 0.98
Accuracy on third bootstrap 0.975
fitting set # 8
Accuracy on first bootstrap 0.975
Accuracy on second bootstrap 0.975
Accuracy on third bootstrap 0.97
[[111 4]
 [ 0 56]]
Accuracy of holdout 0.977
ROC AUC holdout 0.983
F1 score holdout 0.966
```

## CVXOPT

```
Anaconda Powershell Prompt (anaconda3)

Confusion matrix for our svm with linear kernel:
[[47.  0.]
 [ 2. 20.]]
Confusion matrix for our svm with radial kernel:
[[38.  0.]
 [11. 20.]]
Confusion matrix for our svm with polynomial kernel:
[[46.  3.]
 [ 3. 17.]]
Takes sklearn 0.014 to do the computation with linear kernel
Takes our code 0.709 to do the computation with linear kernel
Takes sklearn 0.016 to do the computation with rbf kernel
Takes our code 2.899 to do the computation with rbf kernel
Takes sklearn 0.020 to do the computation with poly kernel
Takes our code 1.104 to do the computation with poly kernel
```



# Results

## Breast cancer data set (Binary classification)

- 9 attributes
- 700 samples
- Homebrewed code
  - Linear decision boundary
  - Stochastic gradient descent minimization
  - CVXOPT Quadratic Equation Solver
  - 70:30 validation set
- Scikit-learn
  - 3 different kernels -> RBF, Linear and Polynomial

# Confusion Matrices

Linear Kernel

No Cancer	1.2e+02	6
Cancer	4	72
	No Cancer	Cancer
	Predicted label	

Polynomial

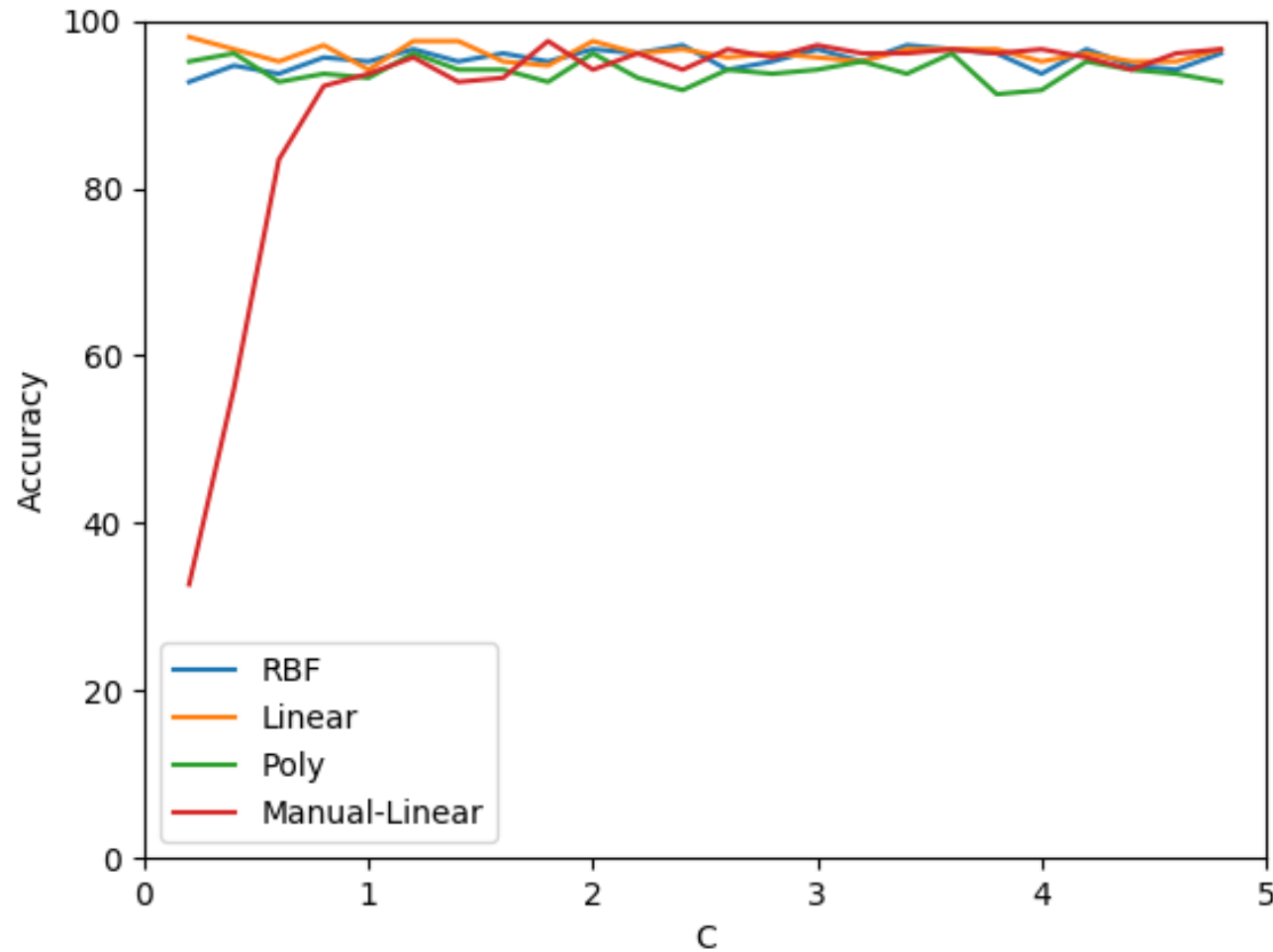
No Cancer	1.2e+02	7
Cancer	4	72
	No Cancer	Cancer
	Predicted label	

RBF

No Cancer	1.2e+02	9
Cancer	2	74
	No Cancer	Cancer
	Predicted label	

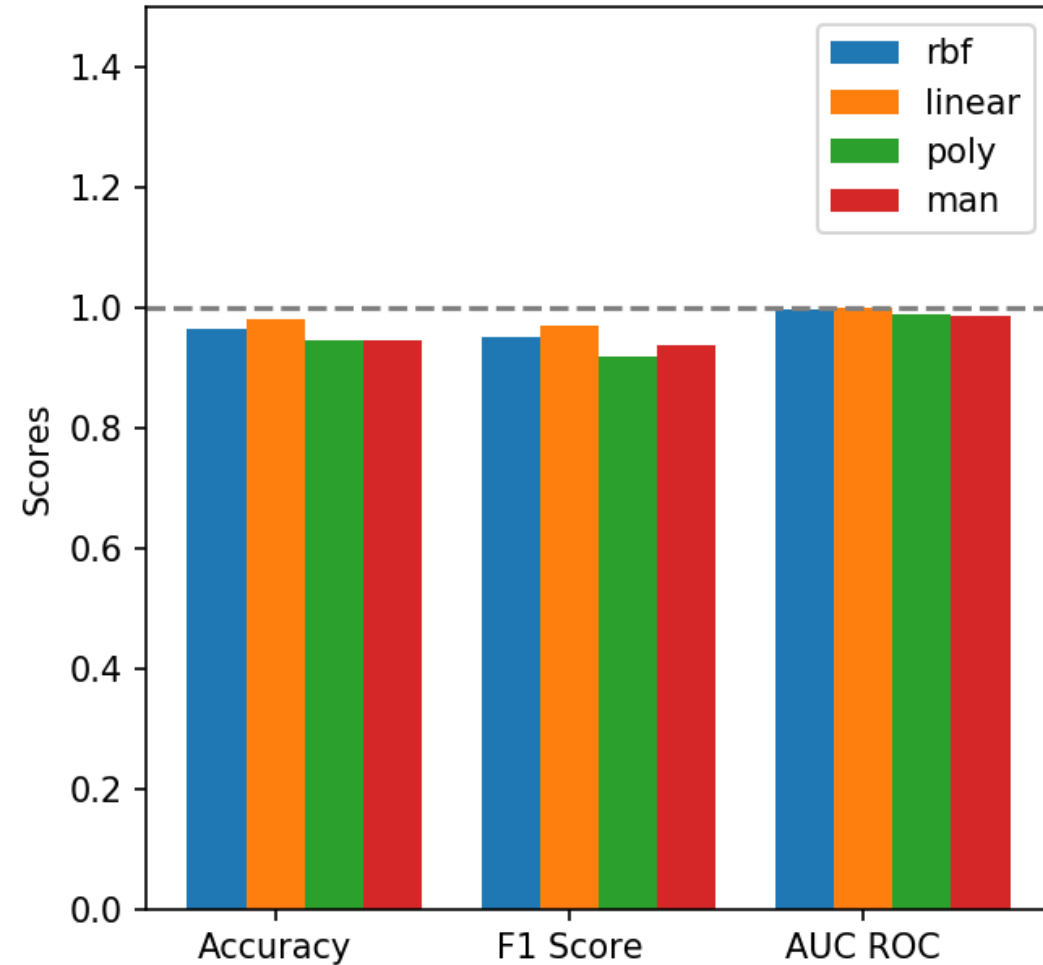
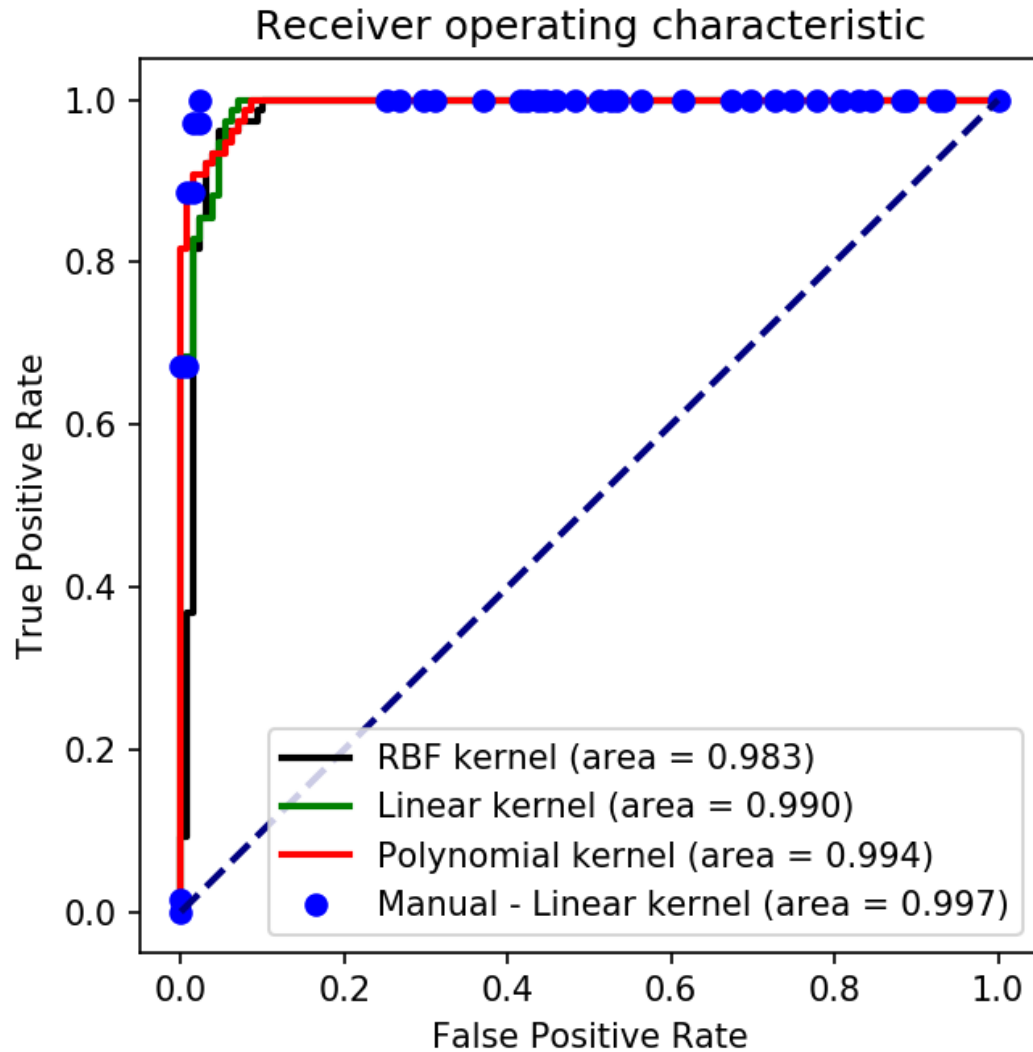
RBF classified two more data points as cancer

# The Effect of Margin Width on Accuracy



Using a very small  $C$  value leads to lower accuracy in the homebrewed code.

# Classifier Output Quality Tests



Accuracy of homebrewed linear SVM classifies as accurate as blackbox algorithms.

# Shortcomings of SVM

- Non probabilistic classification
- Large number of hyperparameters
- Choosing a "good" kernel function
- Not accurate when the signal to noise ratio is low
- Long training time
- Not suitable for multiclass classification
- Extensive memory requirement of quadratic programming solver for large datasets

# Technical Difficulties

- Optimization algorithm requires dual equation solver
- Kernel trick is tricky