REPORT

SUPPORT VECTOR MACHINES

By : MANIK MARWAHA (a1797063)

Submitted for **COMP SCI 7314 Introduction to Statistical Machine Learning** at the School of Computer Science, University of Adelaide towards the Master of Data Science

DEFINITIONS OF VARIABLES USED :

X_I - any data point I in the dataset

Y_i - class of data point i. In case of binary classification like ours it takes value +1 or -1.

W - decision hyper plane normal vector. The vector has values of coefficients of different features which may affect the predicted variable.

B - It is also known as bias. It is defined as the distance of the origin from the hyperplane solution. Without it the classifier will always pass through origin. Hence to find a hyperplane with maximum margin bias is very important.

C (HYPER PARAMETER)- adds penalty to every misclassified data point. For a small C, this penalty is low so we choose a hyperplane with large margin at the expense of more misclassifications. For a large C, misclassified points are minimised but margin is alse very small resulting due to high penalty.

ξ (SLACK VARIABLE) - they are introduced to allow certain constraints to be violated. It is used in our optimization problem in 2 ways. First, it catches the extent of violation for every misclassified point. Second, by adding slack variable to loss function we simultaneously try to minimise its own use.

α (LAGRANGE MULTIPLIER) – it is used for finding the minima of the convex function which is subject to constraints. The number of langrange multipliers we use for optimization is directly proportional to the number of constraints for the given function.

$f(x_i)= w^T x_i + b$: This is the classifier

$y_i(w^T x_i + b)$: Functional margin of x_i

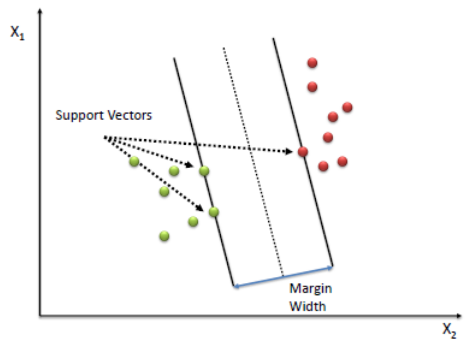
CONCEPT OF SUPPORT VECTOR

Support Vector Machine (SVM) is a supervised machine learning classification algorithm that uses features to predict a target variable. The mapping function for SVM is a decision boundary which makes the distinction between two or more classes.

The basic principle behind SVM is to find the decision boundary such that the margin is maximized keeping the positive and negative class points as wide as possible. The dimension of this decision boundary is (n minus 1)subspace for an n-dimensional space. The dimension of the space depends upon the number of features which affect the results. For a 2-dimension space, the decision boundary is a line which is 1 dimensional.

Margin is defined as the distance of nearest data point from the hyperplane

Support vectors are defined as the data points which are the least distance from the decision boundary (or hyperplane) Hence it becomes increasingly difficult to classify them. These vectors are the training points that lie directly on the margin. There are 3 support vectors in the example shown below as they directly lie on the margin.



Source- medium.com

Support vector directly impact the location of the decision boundary. They are the critical parts of the training dataset which can impact the orientation and position of the decision boundary if removed.

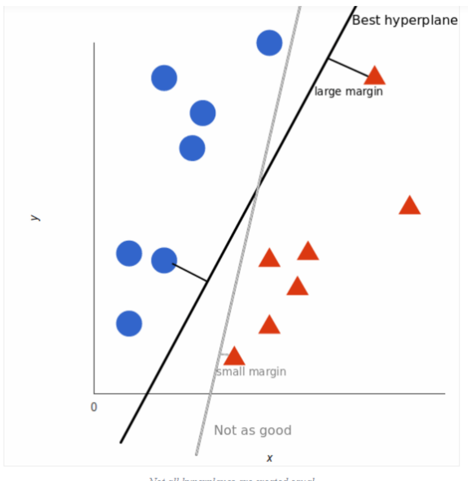
We can use optimization techniques to find this hyperplane. Only the support vectors have an impact on the decision or loss function values. In most cases support vectors form a very small part of the dataset. The optimization problem is a quadratic equation which can be solved using standard methods.

CONCEPT OF MAXIMUM MARGIN

For separation of the 2 classes of data points, our objective is to choose a hyperplane or a decision boundary where the margin is maximum i.e. the distance is maximum between data points of 2 classes .

The smaller the margin we have, the greater is the chance for the data points to be misclassified. On maximising the margin and keeping it wide, the chances of points being misclassified is getting reduced. Hence the first step of solving the optimisation problem is maximization of margin.

For a linearly separable data as shown below we can see that while both the lines are able to classify the data linearly, the better hyperplane is the one in which the margin is large.



Not all hyperplanes are created equal

Hyperplane equation: w`x + b = 0, is denoted as π. Plane w`x + b = 1 positive points lie above this plane.(π(+))and plane w`x + b = -1 negative points lie below this plane( π(-)). For any point x_0 on the plane π(-) we have the equation w`x_0 + b = -1. 'r' is the distance between the 2 planes or the margin.

Note that $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is a unit normal vector of the hyperplane $\mathbf{wx} - b = 1$. We have

$$\mathbf{w}(\mathbf{x}_0 + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) - b = 1$$

since $\mathbf{x}_0 + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$ should be a point in hyperplane $\mathbf{wx} - b = 1$ according to our definition of r.

Expanding this equation, we have

$$\begin{aligned}
&\mathbf{wx}_0 + r\frac{\mathbf{ww}}{\|\mathbf{w}\|} - b = 1 \\
\implies &\mathbf{wx}_0 + r\frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} - b = 1 \\
\implies &\mathbf{wx}_0 + r\|\mathbf{w}\| - b = 1 \\
\implies &\mathbf{wx}_0 - b = 1 - r\|\mathbf{w}\| \\
\implies &-1 = 1 - r\|\mathbf{w}\| \\
\implies &r = \frac{2}{\|\mathbf{w}\|}
\end{aligned}$$

Hence when we minimize vector w we maximise the margin

PRIMAL AND DUAL OF HARD MARGIN SVM

Hard margin SVM are used when the given data points are separable linearly and there are no data points which lie within the margin i.e no misclassified points.

In order to convert into Dual Form, we make use of Lagrange Multipliers. The number of constraints for optimization problem determines the number of Lagrange multipliers. In this case there is only 1 (α).

Primal form:

$$\begin{aligned}
&\min \frac{1}{2}\|w\|^2 \\
&\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1
\end{aligned}$$

Dual form:

$$\text{Maximize } \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$\text{subject to } \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0 \ \forall i$$

PRIMAL AND DUAL OF SOFT MARGIN SVM

In case of soft margin SVM there can be some misclassified points or points which lie on the opposite side of the decision boundary. Hence a more generalised model is formed by soft margin. There are 2 main goals of soft margin SVM:

1. Maximise the margin
2. Maximise the number of correctly classified data points.

There is usually some trade off between these 2 goals which is contrlooed by the hyper parameter C. there are 2 types of misclassified points under soft margin:

- The data point is on the correct side of the margin but on the wrong side of the hyperplane

- The data point is on the wrong side of both the margin and the hyperplane.

Primal form:

$$\text{Minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i, \xi_i \geq 0$$

Dual form:

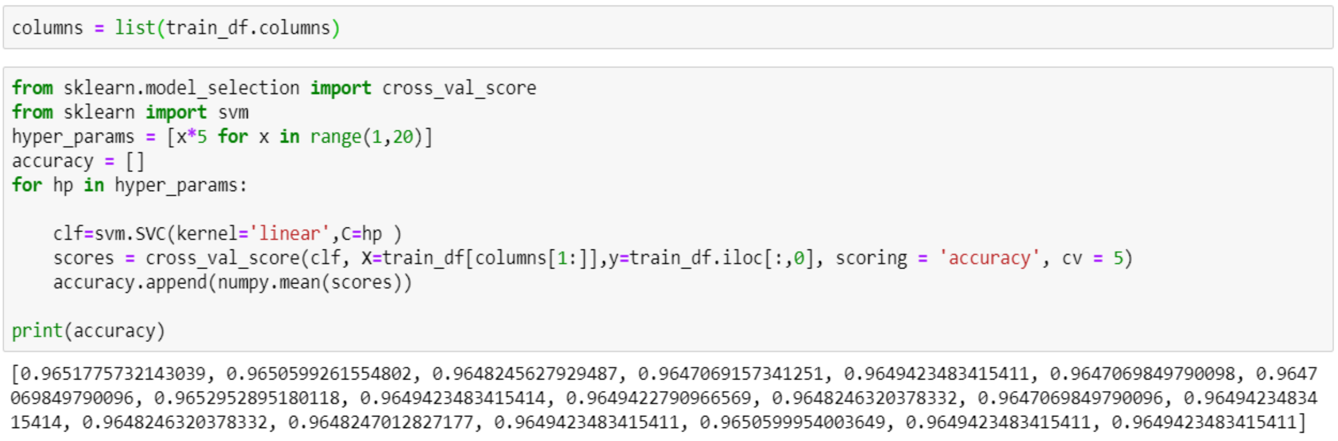
$$L(\alpha) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m\alpha_n y_m y_n \mathbf{k}(\mathbf{x}_m, \mathbf{x}_n)$$

$$\text{subject to} \quad 0 \leq \alpha_n \leq C, \qquad \sum_{n=1}^{N}\alpha_n y_n = 0$$

FINDING HYPER PARAMETER C

The hyper parameter ‘c’ which adds penalty to every misclassified point is found using grid search cross validation. Kfold cross validation technique is used estimation with k = 5. For grid search parameter values from 1 to 100 are chosen as multiples of 5 and then accuracy is calculated for different values. As seen in the image below the accuracy did not change much and was consistently the same. For my code I chose C=10 for finding the accuracy. This grid search cross validation took nearly 5 hours to show the results as it was run on a very extensive data. Hence, I avoid running it every time I run my code.

Below is the code and result as displayed



```
columns = list(train_df.columns)
```

```
from sklearn.model_selection import cross_val_score
from sklearn import svm
hyper_params = [x*5 for x in range(1,20)]
accuracy = []
for hp in hyper_params:

    clf=svm.SVC(kernel='linear',C=hp )
    scores = cross_val_score(clf, X=train_df[columns[1:]],y=train_df.iloc[:,0], scoring = 'accuracy', cv = 5)
    accuracy.append(numpy.mean(scores))

print(accuracy)
```

```
[0.9651775732143039, 0.9650599261554802, 0.9648245627929487, 0.9647069157341251, 0.9649423483415411, 0.9647069849790098, 0.9647
069849790096, 0.9652952895180118, 0.9649423483415414, 0.9649422790966569, 0.9648246320378332, 0.9647069849790096, 0.96494234834
15414, 0.9648246320378332, 0.9648247012827177, 0.9649423483415411, 0.9650599954003649, 0.9649423483415411, 0.9649423483415411]
```

COMPARISON OF ACCURACY OF RESULTS FOR TRAINING DATA SETS

The training data set has been split into small training and validations subsets with training subset size being equal to 0.8 (80% of total dataset). The initial 80% of the data is training and remaining is the validation subset. This is done by using test_train_split from scikit library with shuffle parameter = ‘False’. Then the accuracy results are compared for primal form, dual form and SVM implementation of scikit library.

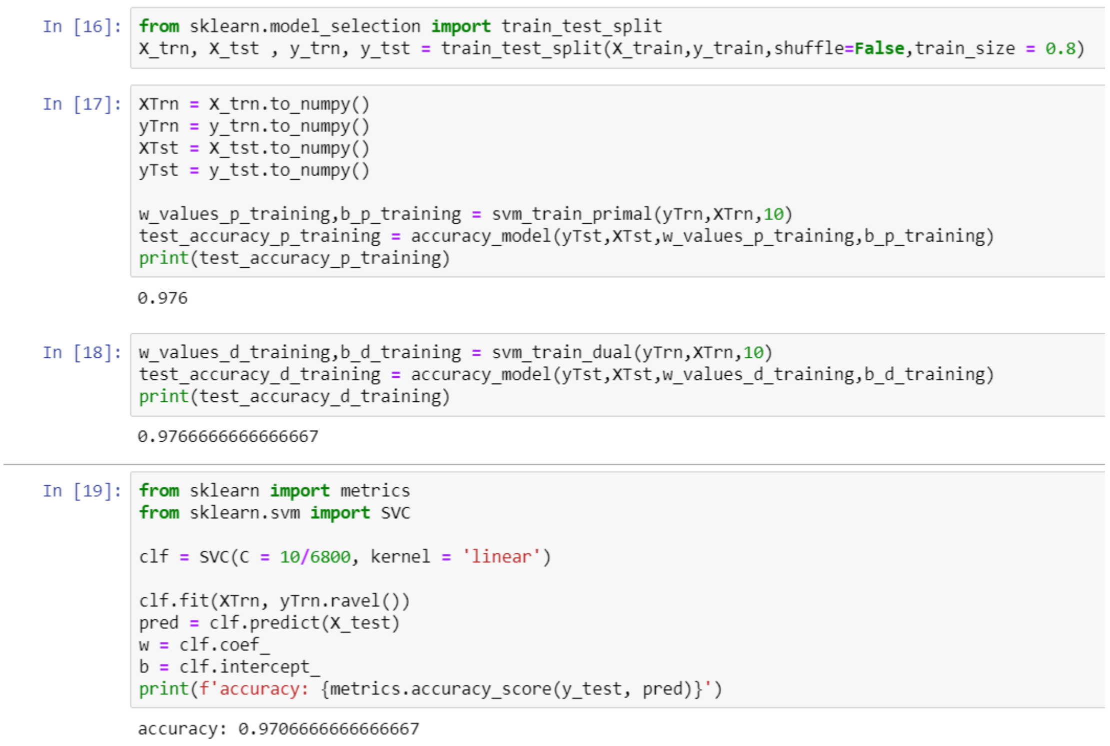
The accuracy score in all the 3 cases is the same for up to 3 decimal places.

The accuracy score of the primal form is 0.976

The accuracy score of the dual form is 0.9766667

The accuracy score obtained by scikit library implementation of SVM is 0.9760667

The training accuracy has not been shown in the code attached so an image of how it was calculated has been attached below.



```
In [16]: from sklearn.model_selection import train_test_split
         X_trn, X_tst , y_trn, y_tst = train_test_split(X_train,y_train,shuffle=False,train_size = 0.8)

In [17]: XTrn = X_trn.to_numpy()
         yTrn = y_trn.to_numpy()
         XTst = X_tst.to_numpy()
         yTst = y_tst.to_numpy()

         w_values_p_training,b_p_training = svm_train_primal(yTrn,XTrn,10)
         test_accuracy_p_training = accuracy_model(yTst,XTst,w_values_p_training,b_p_training)
         print(test_accuracy_p_training)

         0.976

In [18]: w_values_d_training,b_d_training = svm_train_dual(yTrn,XTrn,10)
         test_accuracy_d_training = accuracy_model(yTst,XTst,w_values_d_training,b_d_training)
         print(test_accuracy_d_training)

         0.9766666666666667

In [19]: from sklearn import metrics
         from sklearn.svm import SVC

         clf = SVC(C = 10/6800, kernel = 'linear')

         clf.fit(XTrn, yTrn.ravel())
         pred = clf.predict(X_test)
         w = clf.coef_
         b = clf.intercept_
         print(f'accuracy: {metrics.accuracy_score(y_test, pred)}')

         accuracy: 0.9706666666666667
```

<u>COMPARISON OF ACCURACY OF RESULTS FOR TESTING DATA SETS</u>

To find the accuracy of the primal and the dual I have made a method which compares the predicted and actual result of test set by calculating the correct prediction and then dividing by total number of samples. In the end this result has been compared by an inbuilt 3rd party SVM implementation – implemented using scikit learn.

The accuracy score in all the 3 cases is the same for up to 3 decimal places.

The accuracy score of the primal form is 0.974

The accuracy score of the dual form is 0.9746667

The accuracy score obtained by scikit library implementation of SVM is 0.974.

As the accuracy scores are same in all the 3 cases it shows the accuracy and correctness of the algorithm developed using cvxopt (convex optimisation) in python

THE TABLE BELOW ILLUSTRATES ACCURACY FOR TRAINING AND TESTING SETS AS DISCUSSED ABOVE :

Training accuracy (in percentage)			Testing accuracy (in percentage)		
PRIMAL	DUAL	SCIKIT	PRIMAL	DUAL	SCIKIT
97.6	97.66667	97.60667	97.4	97.46667	97.4

COMPARING W AND B VALUES

(defination of w and b have been given above in the first page of report)

The result as calculated for w and b values is shown below. The w values can be visualized using scatter plots.

The w values as obtained in the code are same for all 3 cases till up to 1 decimal place. After the first decimal place the values show some variation in all the 3 cases.

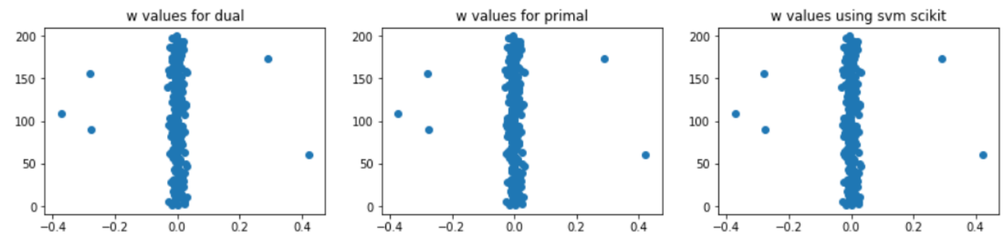
The b values are as follows

Primal : 1.173797

Dual: 1.10861916

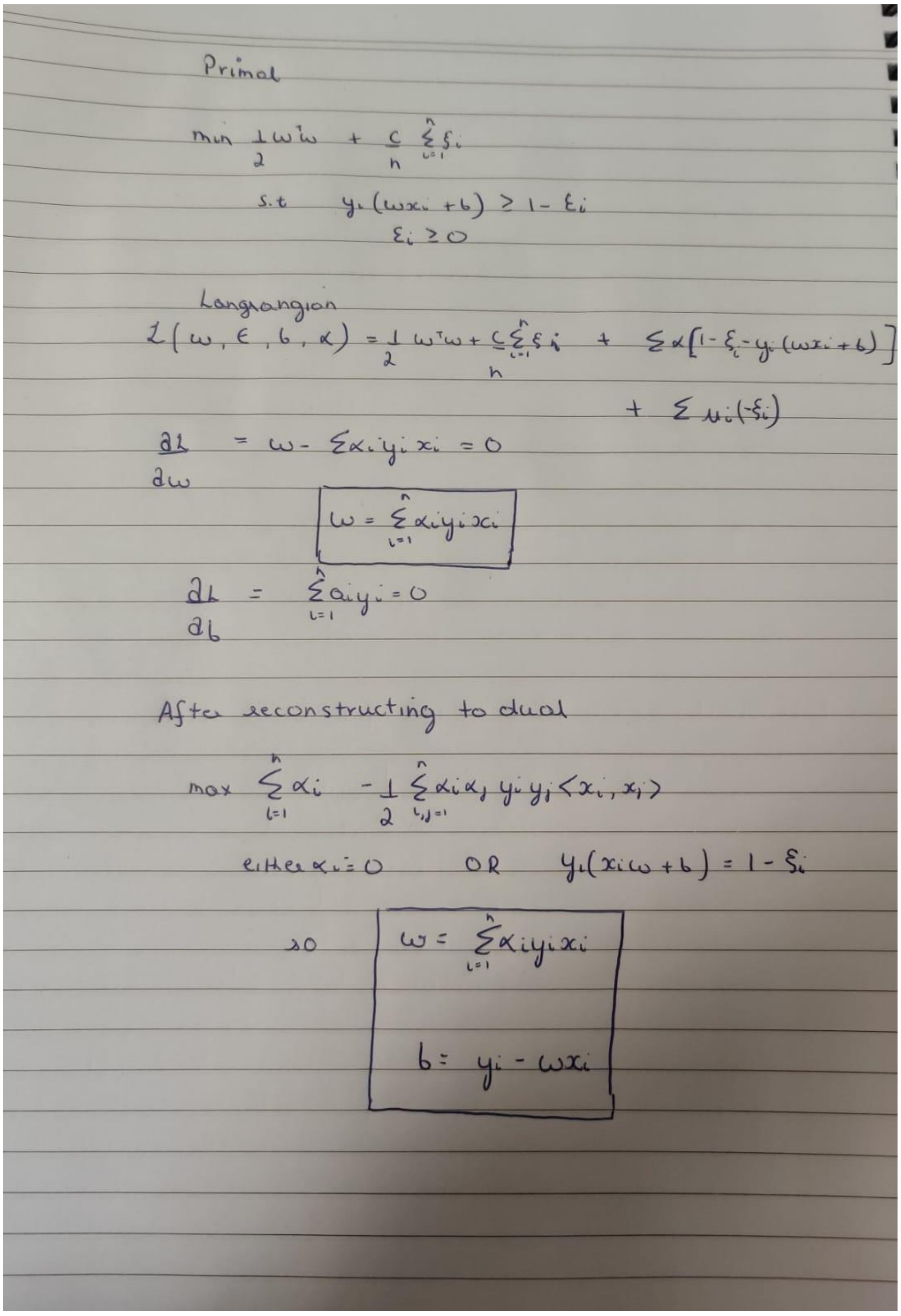
Svm implementation using scikit : 1.15600784

```
In [14]: y = np.array([x for x in range(1,201)]) ## feature number for eavh w value
         fig=plt.figure(figsize=(15,3))

         ax1=plt.subplot(1,3,1)
         ax2=plt.subplot(1,3,2)
         ax3=plt.subplot(1,3,3)
         ax1.scatter(w_values_dual,y)
         ax2.scatter(w_values_primal,y)
         ax3.scatter(w,y)

         ax1.set_title("w values for dual")
         ax2.set_title("w values for primal")
         ax3.set_title("w values using svm scikit")

         plt.show()
```



```
In [15]: print("b value using svm scikit = ",b)
         print("b value using primal method = ",b_primal)
         print("b value using dual method = ",b_dual)

         b value using svm scikit =  [1.15600784]
         b value using primal method =  1.1737968845982747
         b value using dual method =  1.1086191636480154
```

FORMULAE FOR CALCULATING w AND b IS GIVEN IN THE IMAGE BELOW



Primal

$$\min \frac{1}{2} w^T w + \frac{C}{n} \sum_{i=1}^{n} \xi_i$$

s.t $y_i(w x_i + b) \geq 1 - \varepsilon_i$

$\varepsilon_i \geq 0$

Langrangian

$$\mathcal{L}(w, \epsilon, b, \alpha) = \frac{1}{2} w^T w + \frac{C}{n} \sum_{i=1}^{n} \xi_i + \sum \alpha \left[1 - \xi_i - y_i(w x_i + b)\right]$$

$$+ \sum \mu_i(-\xi_i)$$

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum \alpha_i y_i x_i = 0$$

$$\boxed{w = \sum_{i=1}^{n} \alpha_i y_i x_i}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} a_i y_i = 0$$

After reconstructing to dual

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

either $\alpha_i = 0$ OR $y_i(x_i w + b) = 1 - \xi_i$

so
$$\boxed{\begin{array}{l} w = \sum_{i=1}^{n} \alpha_i y_i x_i \\ \\ b = y_i - w x_i \end{array}}$$