Data Analysis and Machine-Learning

Chapter 6.1:

Support Vector Machine (1)

Concepts and Calculation



1. Introduction

Support Vector Machine (SVM), a representative machine-learning model based on statistical learning theory, is known to be particularly strong (in terms of its generalization performance) for classifications of high-dimensional datasets. The formulation of SVM can be framed as solving a quadratic programming (QP) problem.

As covered in the earlier chapters, there is generally an unavoidable **tradeoff** between the generalization ability and fitting to the training data (underfitting vs. overfitting problems). In this context, one advantage of SVM is that its direct decision function allows us to naturally decrease testing errors while fitting the model to the training set.

2. Concepts of SVM

Suppose a two-class classification problem, with data scattered in a 3-dimensional area. The hyperplane that separates these classes can be defined as:

where ‘W’ is the normal vector of the hyperplane (i.e.., slope of the function) and ‘b’ is the bias. Our objective would be to find the value of W and b, i.e., a function that separates the two-class dataset.

Now, consider following figures.

plt.figure(figsize=(5,10))

plt.subplot(3,1,1)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.plot(xx, xx2, linewidth=2.5)

plt.plot(zz5, zz6, c='r', linestyle=':', linewidth=2)

plt.plot(zz7, zz8, c='b', linestyle=':', linewidth=2)

plt.grid()

plt.subplot(3,1,2)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.plot(xx3, xx4, linewidth=2.5)

plt.plot(zz9, zz10, c='r', linestyle=':', linewidth=2)

plt.plot(zz11, zz12, c='b', linestyle=':', linewidth=2)

plt.grid()

plt.subplot(3,1,3)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.axvline(x=0, color='g', linewidth=2.5)

plt.axvline(x=-1, color='r', linestyle='--')

plt.axvline(x=1, color='b', linestyle='--')

plt.plot(zz1, zz2, c='black', linestyle=':', linewidth=2)

plt.plot(zz3, zz4, c='black', linestyle=':', linewidth=2)

plt.annotate('maximum margin', (0.4,1.6), (2,3), arrowprops={'width':1}, fontsize=15)

plt.grid()

plt.figure(figsize=(5,10))

plt.subplot(3,1,1)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.plot(xx, xx2, linewidth=2.5)

plt.plot(zz5, zz6, c='r', linestyle=':', linewidth=2)

plt.plot(zz7, zz8, c='b', linestyle=':', linewidth=2)

plt.title('figure 1')

plt.grid()

plt.subplot(3,1,2)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.plot(xx3, xx4, linewidth=2.5)

plt.plot(zz9, zz10, c='r', linestyle=':', linewidth=2)

plt.plot(zz11, zz12, c='b', linestyle=':', linewidth=2)

plt.title('figure 2')

plt.grid()

plt.subplot(3,1,3)

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.axvline(x=0, color='g', linewidth=2.5)

plt.axvline(x=-1, color='r', linestyle='--')

plt.axvline(x=1, color='b', linestyle='--')

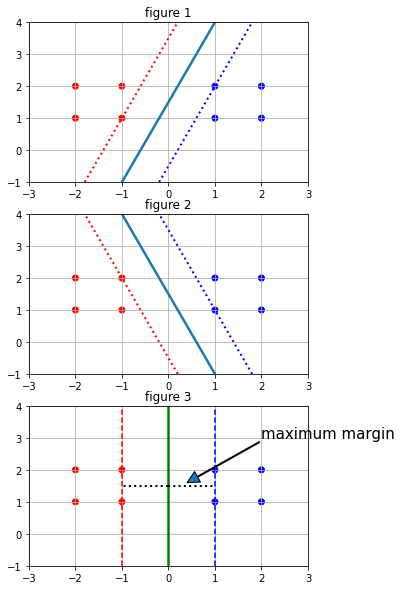
plt.plot(zz1, zz2, c='black', linestyle=':', linewidth=2)

plt.plot(zz3, zz4, c='black', linestyle=':', linewidth=2)

plt.annotate('maximum margin', (0.4,1.6), (2,3), arrowprops={'width':1}, fontsize=15)

plt.title('figure 3')

plt.grid()



There can be numerous numbers of lines (or, hyperplanes) that correctly separate the blue dots and red dots, including the ones shown in the figures. What we want to find out is the “best” hyperplane among the numerous possible hyperplanes that separates the datasets.

This is why we need the concept of “Margin”, in order to find out the hyperplane that “minimizes” generalization errors over the training set so that we can generate a model with a decent prediction performance. The basic mechanism of SVM is composed of this very concept of margins.

Applying the concepts of margins, we can speculate following relationships between the “plus-plane” and “minus-plane” respectively separated by maximum margins from the hyperplane.

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.axvline(x=0, color='g', linewidth=2.5)

plt.axvline(x=-1, color='r', linestyle='--')

plt.axvline(x=1, color='b', linestyle='--')

plt.plot(zz1, zz2, c='black', linestyle=':', linewidth=2)

plt.plot(zz3, zz4, c='black', linestyle=':', linewidth=2)

plt.annotate('maximum margin', (0.4,1.6), (2,3), arrowprops={'width':1}, fontsize=15)

plt.annotate('Wx+b=0', (-0.5,0), fontsize=13)

plt.annotate('Wx+b=-1', (-1.8,0), fontsize=13)

plt.annotate('Wx+b=+1', (1,0), fontsize=13)

plt.annotate('Wx+b<=-1', (-2.5,1.5), fontsize=13)

plt.annotate('Wx+b>=+1', (2,1.5), fontsize=13)

plt.title('figure 3')

plt.grid()

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.xlim(-3,3)

plt.ylim(-1,4)

plt.axvline(x=0, color='g', linewidth=2.5)

plt.axvline(x=-1, color='r', linestyle='--')

plt.axvline(x=1, color='b', linestyle='--')

plt.plot(zz1, zz2, c='black', linestyle=':', linewidth=2)

plt.plot(zz3, zz4, c='black', linestyle=':', linewidth=2)

plt.annotate('maximum margin', (0.4,1.6), (2,3), arrowprops={'width':1}, fontsize=15)

plt.annotate('Wx+b=0', (-0.5,0), fontsize=13)

plt.annotate('Wx+b=-1', (-1.8,0), fontsize=13)

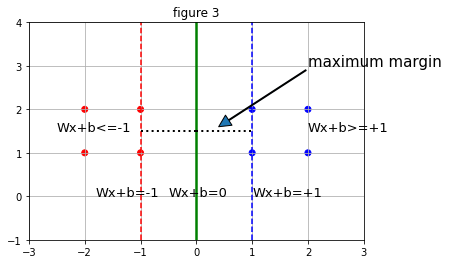
plt.annotate('Wx+b=+1', (1,0), fontsize=13)

plt.annotate('Wx+b<=-1', (-2.5,1.5), fontsize=13)

plt.annotate('Wx+b>=+1', (2,1.5), fontsize=13)

plt.title('figure 3')

plt.grid()



As in the figure,

when plus-plan is defined as:

and minus-plan is defined as:

the points in the plus plane are ***parallel transference*** of the points in the minus plane to the direction of W by lambda:

Solving this altogether,

Applying distributive law,

Recall that our goal here is to find out the value of w.

The L2 vector norm of w is defined as:

As Margin is defined as the difference between the points in plus plan and minus plane [i.e., margin = distance(x+, x-)],

Thus, when margin is organized with W, the margin can be defined as 2 divided by the L2 norm of W. With equation defined as such, our objective is to find the value of W that maximizes the margin:

Since L2 norm of w contains square roots in its equation, it would be more convenient to calculate via squaring the equation as the following:

Now that we can define objective and constraint equations as:

What we want to find out are the values of the decision variables (w and b) that minimizes the value of the objective function, subject to the margin being at least equal to or larger than 1.

Notice that the objective function is quadratic (squared) while the constraint equation is linear (linear constraint). Optimization for such formulation can be solved via ***quadratic programming***, which has the attributes of ***convex*** optimization, such as the one I have covered before in the chapters on gradient descent. This implies that there is only ***one*** global optimum, which enables us to find ***max*** or ***min*** value.

Using Lagrangian multiplier to convert and solve the max/min problem, it is possible to convert the original objective and constraint equations as:

Since the function is convex, the minimum value is the point where the differential coefficient equals to 0.

Conducting derivatives over w and b, respectively,

Applying the resulting equation to the primal objective equation:

For :

Solving for the first part:

and second part:

As a result, we can get:

Now that we got rid of w in the equation, all we have to figure out now is the maximizing value of for the above equation. As in the case for the primal formulation, notice that the objective function is quadratic (), while the constraint equation is linear (). Thus, it is possible to identify that there is ***one*** global optimum (i.e., convex optimization for ) when using quadratic programming to solve the formulation. In other words, there should be only one value of that maximizes the objective function above.

Solving Lagrangian dual problem under KKT conditions,

For

1. if

This becomes the case for when x is on the ***support vector***, i.e., exactly on the line of the plus and minus planes

2. if

This becomes the case for when x is not on the ***support vector***, i.e., outside of the margin boundary lines.

In the end, the x (observation data) that we use to calculate margins are the ones on the plus/minus planes, i.e., the ones on the support vector.

In other words,

Thus, optimal hyperplane can be derived by using only the x data on the support vector, instead of using all x data for its derivation, which is why SVM is known as the ***sparse modelling*** method.

Now that we have derived the value of ‘w’, the only remaining value that we need to know in order to figure out the optimal separating hyperplane,

is ‘b’, the bias of the equation.

Substituting w for what we have calculated earlier:

Throughout the process of calculating weights and bias for the optimal separating hyperplane in SVM, we have so far figured out the calculation mechanism via sparse modelling, using support vector data instead of the whole datasets. Indeed, this characteristic of SVM becomes the main advantage in terms of machine-learning, because generalization ability and classification accuracy can be maximized while being relatively immune to overfitting problems.