Module6_v2

June 1, 2022

0.0.1 Grading

The final score that you will receive for your programming assignment is generated in relation to the total points set in your programming assignment item—not the total point value in the nbgrader notebook. When calculating the final score shown to learners, the programming assignment takes the percentage of earned points vs. the total points provided by nbgrader and returns a score matching the equivalent percentage of the point value for the programming assignment. **DO NOT CHANGE VARIABLE OR METHOD SIGNATURES** The autograder will not work properly if your change the variable or method signatures.

0.0.2 Validate Button

Please note that this assignment uses nbgrader to facilitate grading. You will see a **validate button** at the top of your Jupyter notebook. If you hit this button, it will run tests cases for the lab that aren't hidden. It is good to use the validate button before submitting the lab. Do know that the labs in the course contain hidden test cases. The validate button will not let you know whether these test cases pass. After submitting your lab, you can see more information about these hidden test cases in the Grader Output. ***Cells with longer execution times will cause the validate button to time out and freeze. Please know that if you run into Validate time-outs, it will not affect the final submission grading.***

1 Homework 6: Support Vector Machines

In this assignment we'll explore the details of the Soft-Margin SVM and look at how the choice of tuning parameters affects the learned models. We'll also look at kernel SVMs for non-linearly separable and methods for choosing and visualizing good hyperparameters.

Note: Below are some helper functions which are used throughout this notebook. Execute these before continuing. Do not modify.

[1]:
```
import numpy as np
from sklearn.datasets import make_blobs
from matplotlib.colors import Normalize
import matplotlib.pyplot as plt
%matplotlib inline
```

```
def linear_plot(X, y, w=None, b=None):

    mycolors = {"blue": "steelblue", "red": "#a76c6e", "green": "#6a9373"}
    colors = [mycolors["red"] if yi==1 else mycolors["blue"] for yi in y]

    # Plot data
    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8,8))
    ax.scatter(X[:,0], X[:,1], color=colors, s=150, alpha=0.95, zorder=2)

    # Plot boundaries
    lower_left = np.min([np.min(X[:,0]), np.min(X[:,1])])
    upper_right = np.max([np.max(X[:,0]), np.max(X[:,1])])
    gap = .1*(upper_right-lower_left)
    xplot = np.linspace(lower_left-gap, upper_right+gap, 20)
    if w is not None and b is not None:
        ax.plot(xplot, (-b - w[0]*xplot)/w[1], color="gray", lw=2, zorder=1)
        ax.plot(xplot, ( 1 -b - w[0]*xplot)/w[1], color="gray", lw=2, ls="--",␣
↪zorder=1)
        ax.plot(xplot, (-1 -b - w[0]*xplot)/w[1], color="gray", lw=2, ls="--",␣
↪zorder=1)


    ax.set_xlim([lower_left-gap, upper_right+gap])
    ax.set_ylim([lower_left-gap, upper_right+gap])

    ax.grid(alpha=0.25)

def part2data():

    np.random.seed(1239)

    X = np.zeros((22,2))
    X[0:10,0]  = 1.5*np.random.rand(10)
    X[0:10,1]  = 1.5*np.random.rand(10)
    X[10:20,0] = 1.5*np.random.rand(10) +  1.75
    X[10:20,1] = 1.5*np.random.rand(10) +  1
    X[20,0] = 1.5
    X[20,1] = 2.25
    X[21,0] = 1.6
    X[21,1] = 0.25

    y = np.ones(22)
    y[10:20] = -1
    y[20] = 1
    y[21] = -1
    return X, y

def part3data(N=100, seed=1235):

    np.random.seed(seed)

    X = np.random.uniform(-1,1,(N,2))
    y = np.array([1 if y-x > 0 else -1 for (x,y) in zip(X[:,0]**2 * np.sin(2*np.
 ↪pi*X[:,0]), X[:,1])])
    X = X + np.random.normal(0,.1,(N,2))

    return X, y

def nonlinear_plot(X, y, clf=None):

    mycolors = {"blue": "steelblue", "red": "#a76c6e", "green": "#6a9373"}

    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10,10))

    colors = [mycolors["red"] if yi==1 else mycolors["blue"] for yi in y]
    ax.scatter(X[:,0],X[:,1], marker='o', color=colors, s=100, alpha=0.5)

    ax.arrow(-1.25,0,2.5,0, head_length=0.05, head_width=0.05, fc="gray",␣
 ↪ec="gray", lw=2, alpha=0.25)
    ax.arrow(0,-1.25,0,2.5, head_length=0.05, head_width=0.05, fc="gray",␣
 ↪ec="gray", lw=2, alpha=0.25)
    z = np.linspace(0.25,3.5,10)

    ax.set_xlim([-1.50,1.50])
    ax.set_ylim([-1.50,1.50])

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.xticks([], fontsize=16)
    plt.yticks([], fontsize=16)


    if clf:

        clf.fit(X,y)

        x_min = X[:, 0].min()+.00
        x_max = X[:, 0].max()-.00
        y_min = X[:, 1].min()+.00
        y_max = X[:, 1].max()-.00
        colors = [mycolors["red"] if yi==1 else mycolors["blue"] for yi in y]

        XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
        Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(XX.shape)
        plt.contour(XX, YY, Z, colors=[mycolors["blue"], "gray",␣
 ↪mycolors["red"]], linestyles=['--', '-', '--'],
                    levels=[-1.0, 0, 1.0], linewidths=[2,2,2], alpha=0.9)


class MidpointNormalize(Normalize):

    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))

def plotSearchGrid(grid):

    scores = [x for x in grid.cv_results_["mean_test_score"]]
    scores = np.array(scores).reshape(len(grid.param_grid["C"]), len(grid.
 ↪param_grid["gamma"]))

    plt.figure(figsize=(10, 8))
    plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
    plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,
               norm=MidpointNormalize(vmin=0.2, midpoint=0.92))
    plt.xlabel('gamma')
    plt.ylabel('C')
    plt.colorbar()
    plt.xticks(np.arange(len(grid.param_grid["gamma"])), grid.
 ↪param_grid["gamma"], rotation=45)
    plt.yticks(np.arange(len(grid.param_grid["C"])), grid.param_grid["C"])
    plt.title('Validation accuracy')
    plt.show()

from IPython.core.display import HTML
HTML("""
<style>
.MathJax nobr>span.math>span{border-left-width:0 !important};
</style>
""")
```

[1]: <IPython.core.display.HTML object>

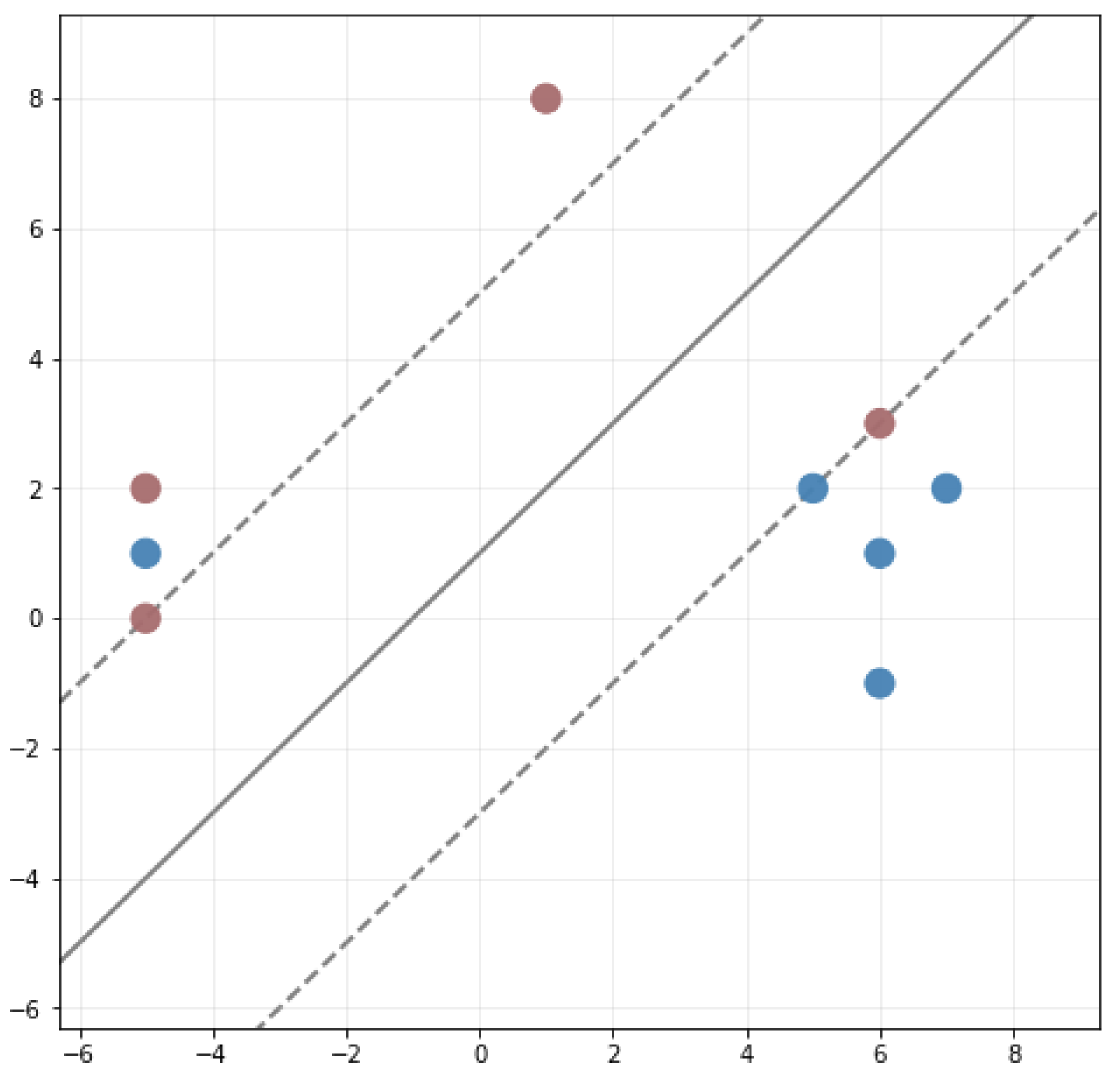
1.0.1 Part 1: SVM [20 pts, Peer Review]

[ ]:

```
```

[2]:

```
import numpy as np

# Data and Labels
X = np.array([[1,8],[7,2],[6,-1],[-5,0], [-5,1], [-5,2],[6,3],[6,1],[5,2]])
y = np.array([1,-1,-1,1,-1,1,1,-1,-1])

# Support vector parameters
w, b = np.array([-1/4, 1/4]), -1/4

# Plot the data and support vector boundaries
linear_plot(X, y, w=w, b=b)
```



Part A [**5 pts**]: What is the margin of this particular SVM?

[3]:
```
margin = 0 # update margin to be a correct number

# your code here
margin = 1/4
margin
```

[3]:
```
0.25
```

[4]:
```
# tests that margin was updated correctly
```

Part B [**5 pts, Peer Review**]: Which training examples are the support vectors? Assign the coordinate in the list. e.g. support_vectors = [(1,0),(0,0)]

[5]:
```
# uncomment and update support_vectors
# your code here

support_vectors=[(-5,0),(6,3),(5,2), (-5,1)]
support_vectors
```

[5]:
```
[(-5, 0), (6, 3), (5, 2), (-5, 1)]
```

[6]:
```
# tests support_vectors
```

Answer the following question in this week's **Peer Review assignment**: Explain your answer for which training examples are support vectors. For the support vectors that you have identified, which are the in-bound support vectors and which are the out-bound support vectors?

(-5,0), (5,2) are in-bound support vectors because they lie on the margin on the appropriate side of the hyperplane. (-5,1) and (6,3) are out-bound support vectors because they lie on the opposite side of the hyperplane than their respective class. All four of these points are considered support vectors because they each influence the orientation of the hyperplane.

[ ]:
```
# instructor testing cell
# your code here
```

Part C [5 pts, Peer Review]: Which training examples have nonzero slack? List their coordinates.

[8]:
```
# list of coordinates with nonzero slack
nonzero_slack=[]
# your code here
nonzero_slack = [(6,3),(-5,1)] #The data points on the wrong side of the␣
 ↪classifier
```

I knew (6,3) and (-5,1) had nonzero slack because they were both on the opposite side of the hyperplane than their respective class.

[9]:
```
# tests nonzero_slack list
```

Answer the following question in this week's **Peer Review assignment**: How did you know which training examples had nonzero slack?

[10]:
```
# instructor testing cell
# your code here
```

Part D [5 pts, Peer Review]: Compute the slack ξ_i associated with the misclassified points.

[11]:
```
# your code here

# compute the slack associated with the misclassified points
# return a list slacks with the slack computations
```

```
slack1 = 1 - ((np.dot(np.transpose(w), np.array([6,3] )) + b) * 1)
slack2 = 1 - ((np.dot(np.transpose(w), np.array([-5,1] )) + b) * -1)

slacks = [slack1, slack2]
print(slacks)
```

```
[2.0, 2.25]
```

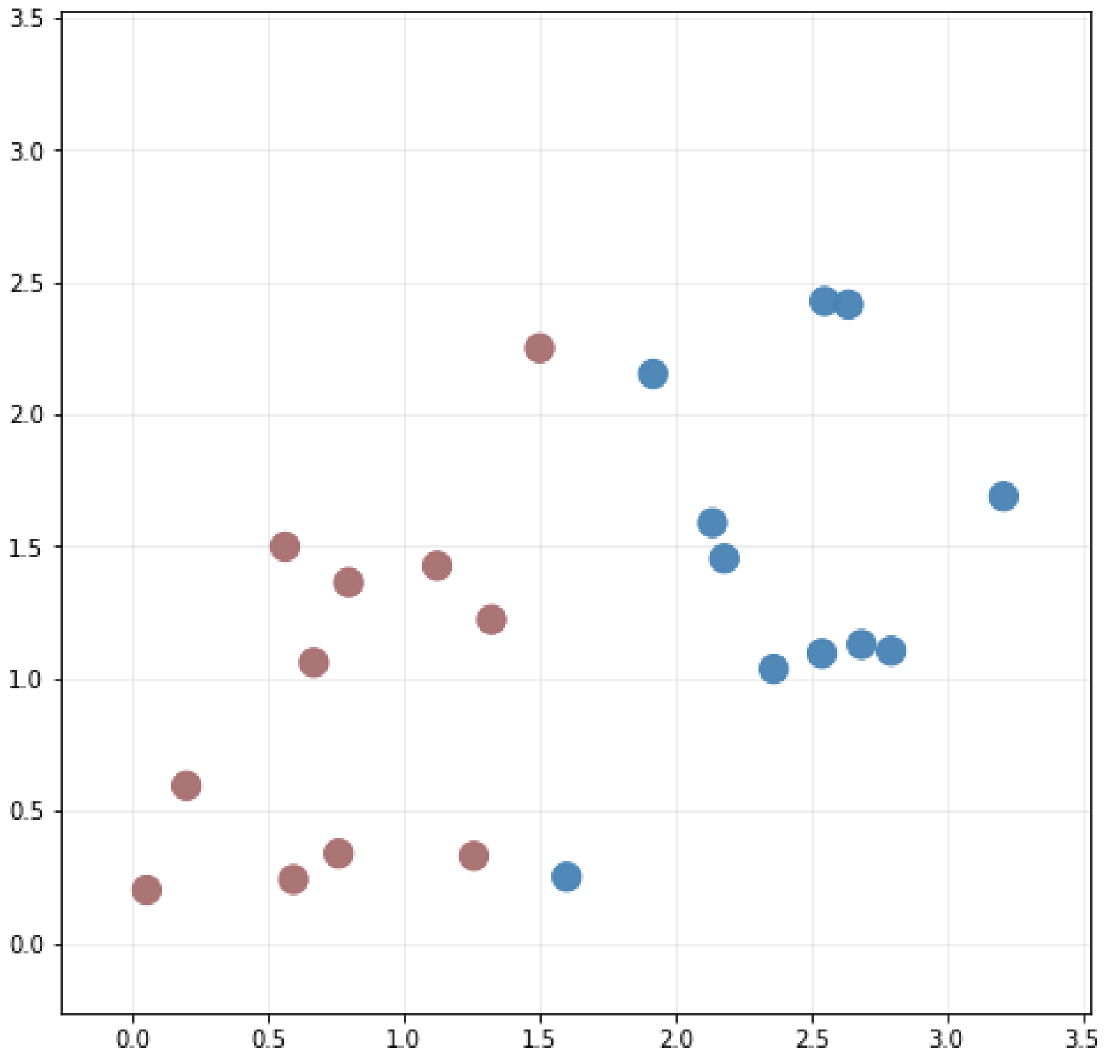
[12]:
```
# tests slacks
```

Answer the following question in your **Peer Review assignment for the week**: Do these slack values agree with the plot of the data and the support vector boundaries?

1.0.2 Part 2: The Margin vs Slack

In this problem we'll figure out how to fit linear SVM models to data using sklearn. Consider the data shown below.

[13]:
```
X, y = part2data()
linear_plot(X, y)
```



Part A [5 pts]: Let's fit a linear Soft-Margin SVM to the data above. For SVMs with a linear kernel we'll use the `LinearSVM` method from sklearn's `svm` module. Go now and look at the documentation.

Recall that the primal objective function for the linear kernel SVM is as follows

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i^p.$$

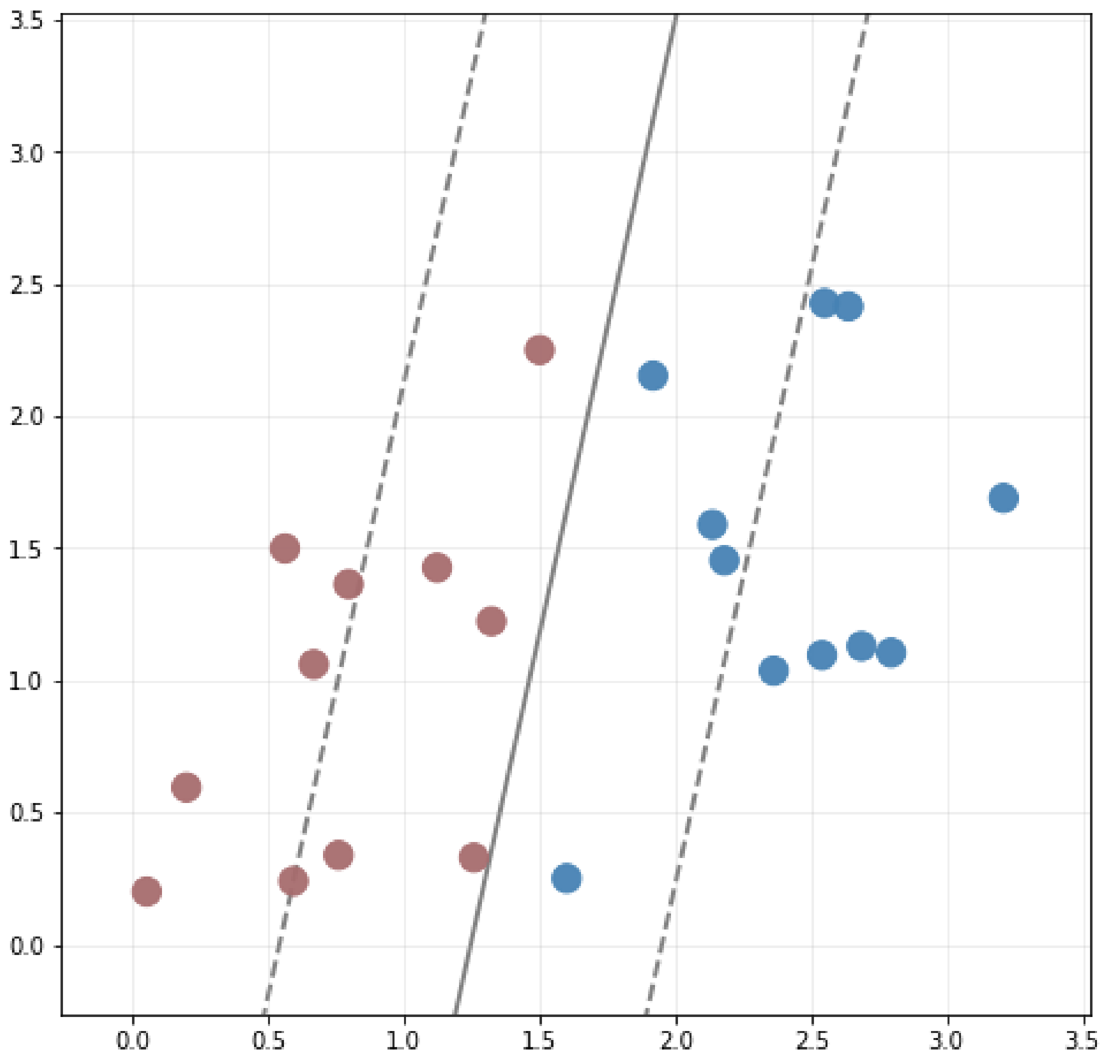
Note that the C parameter definition in sklearn is different from the textbook.

The two optional parameters in `LinearSVM` that we'll be most concerned with are `C`, the hyper-parameter weighting the slackness contribution to the primal objective function, and `loss`, which determines the exponent on the slack variables in the sum. When $p = 1$ from above equation, the loss is hinge loss, whereas is $p = 2$, it's $L2$ form, the loss is called squared-hinge.

Write some code below to train a linear SVM with $C = 1$ and $p = 1$, get the computed weight vector and bias, and the plot the resulting model.

[14]:
```
from sklearn.svm import LinearSVC

# TODO: Build a LinearSVC model called lsvm. Train the model and get the␣
↪parameters, pay attention to the loss parameter
lsvm = None
# your code here
mod_svc = LinearSVC(C=1, loss='hinge')
lsvm = mod_svc.fit(X,y)
# use this code to plot the resulting model
w = lsvm.coef_[0]
b = lsvm.intercept_
print(w,b)
linear_plot(X, y, w=w, b=b)
```

```
[-1.41897528  0.30584252] [1.76759606]
```



[15]:
```
# tests w and b
```

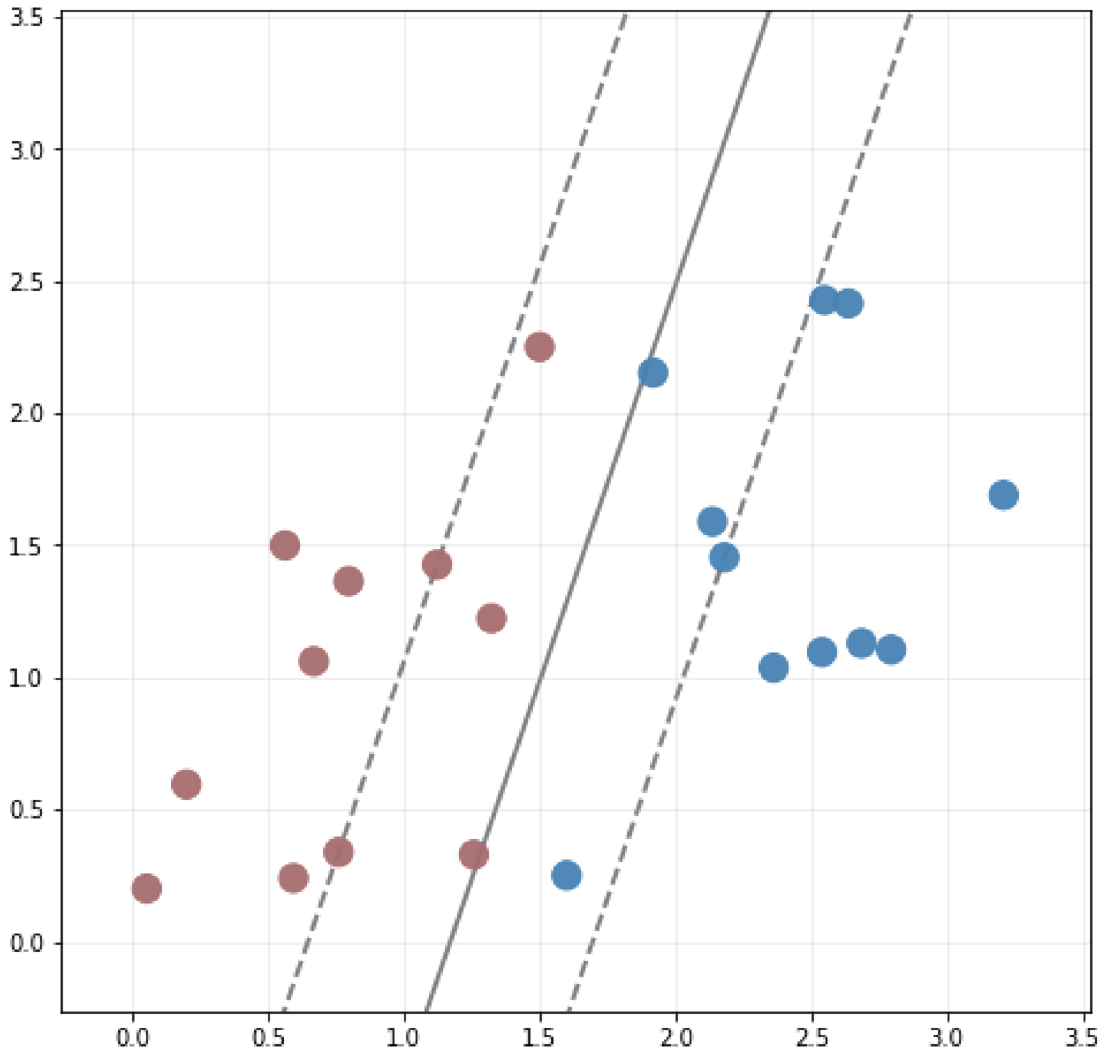
Part B: Experiment with different values of `C`. Answer the following question in your Peer Review for the week: How does the choice of `C` affect the nature of the decision boundary and the associated margin?

With the SVC package, a larger C value increases the decision boundary (makes margins smaller). A smaller C value decreases the decision boundary (makes margins larger).

[16]:
```
from sklearn.svm import LinearSVC

# TODO: Change the svm model parameter and plot the result.

# your code here
```

```
mod_svc = LinearSVC(C=2, loss='hinge')
lsvm = mod_svc.fit(X,y)
# use this code to plot the resulting model
w = lsvm.coef_[0]
b = lsvm.intercept_
print(w,b)
linear_plot(X, y, w=w, b=b)
```

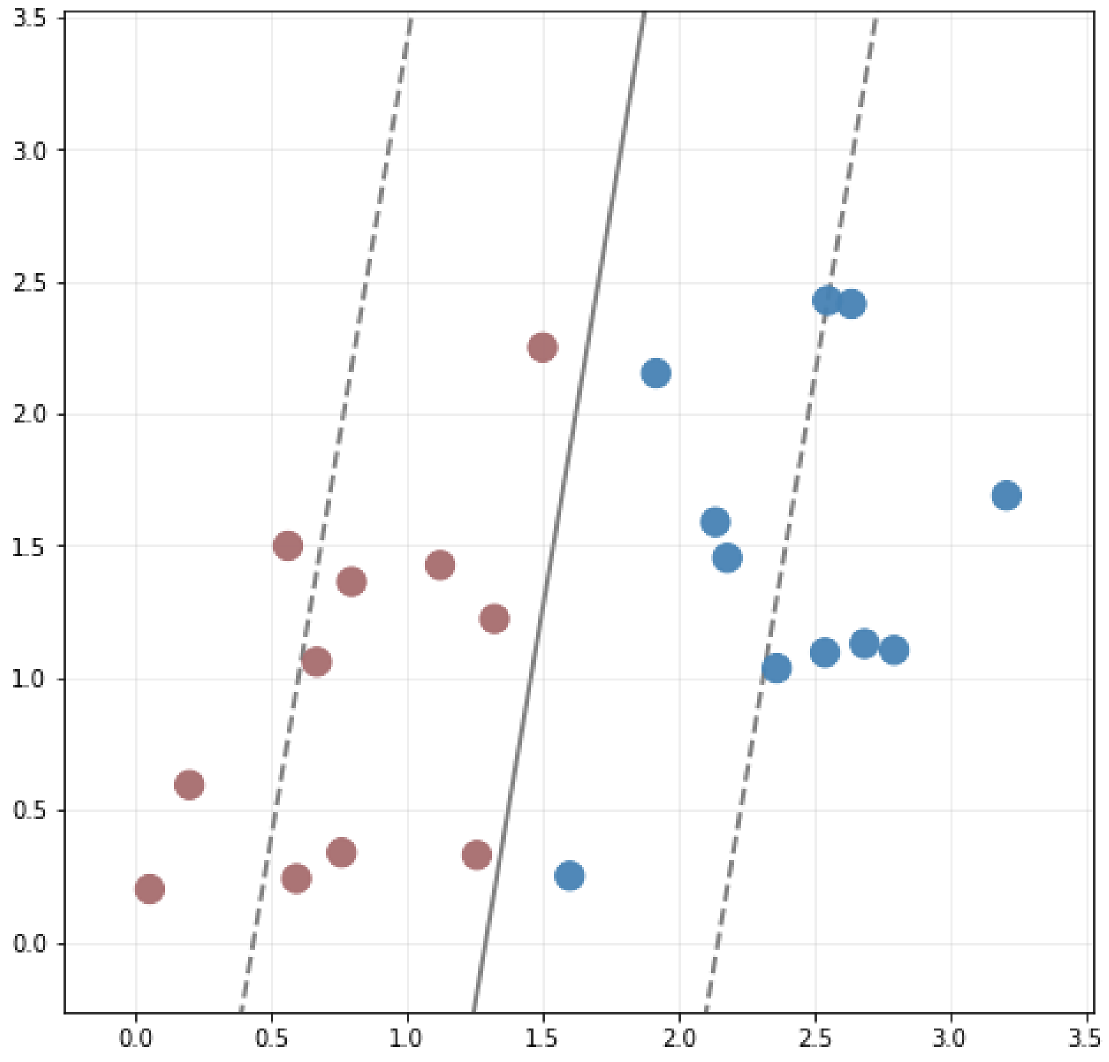
```
[-1.90654962  0.63658974] [2.2323101]
```



[17]:
```
mod_svc = LinearSVC(C=0.5, loss='hinge')
lsvm = mod_svc.fit(X,y)
# use this code to plot the resulting model
w = lsvm.coef_[0]
```

```
b = lsvm.intercept_
print(w,b)
linear_plot(X, y, w=w, b=b)
```

```
[-1.16772693  0.19430092] [1.50634057]
```

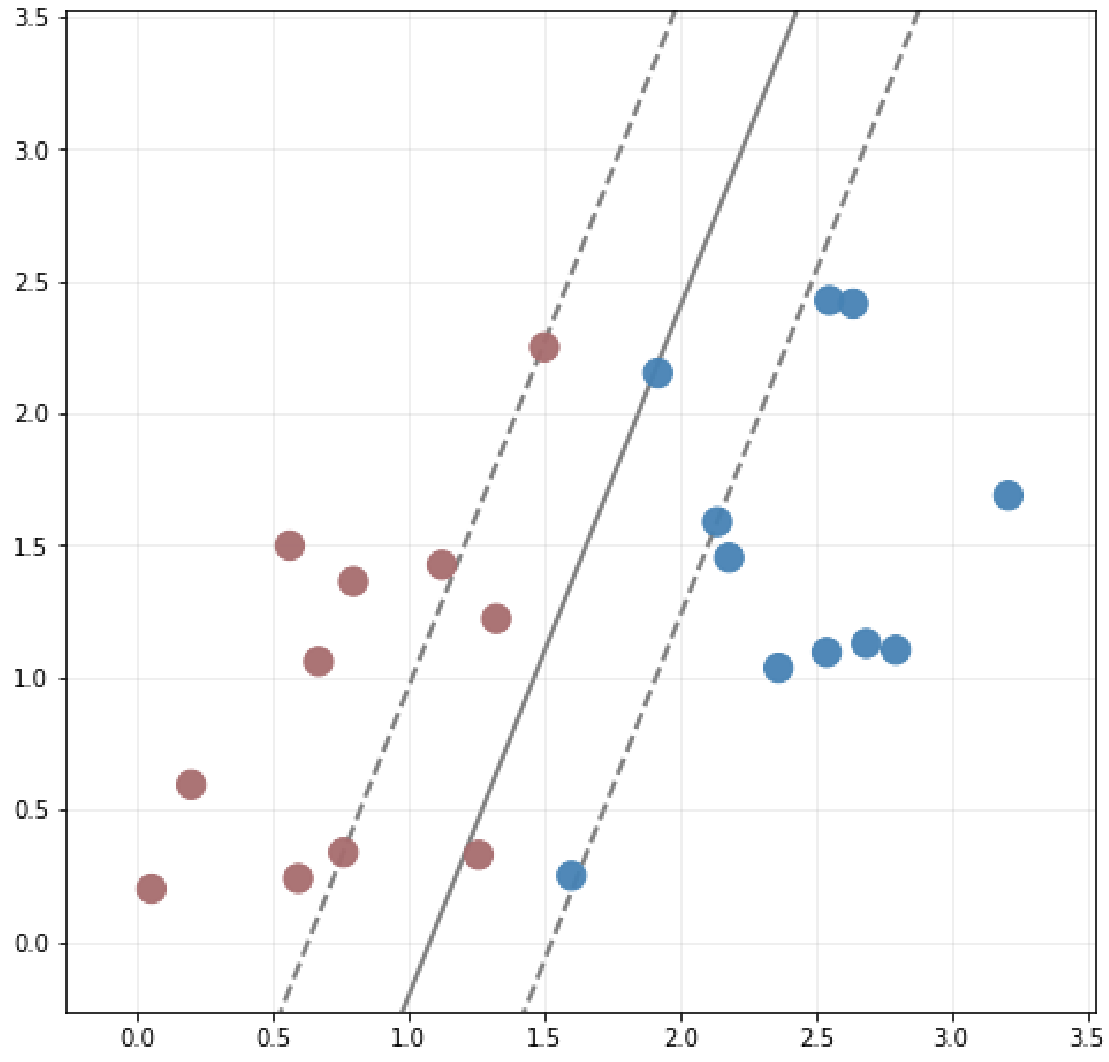


Part C: Set `C=3` and compare the results you get when using the `hinge` vs the `squared_hinge` values for the `loss` parameter. In this week's Peer Review Assignment: Explain your observations. Compare hinge loss vs. squared hinge loss.

'Hinge-loss' seems to classify the points better. 'Squared-hinge' loss results in 8 total support vectors (within the margins). 'Hing-loss' results in 7 total support vectors with 4 points directly on the margins. Hinge loss penalizes support vectors linearly to their distances to the hyperplane, whereas squared hinge loss more heavily penalizes ones that have more violations. Thus, squared hinge loss provides more slack within the dashed lines but allows fewer support vectors on the plane's wrong side than plain hinge loss.

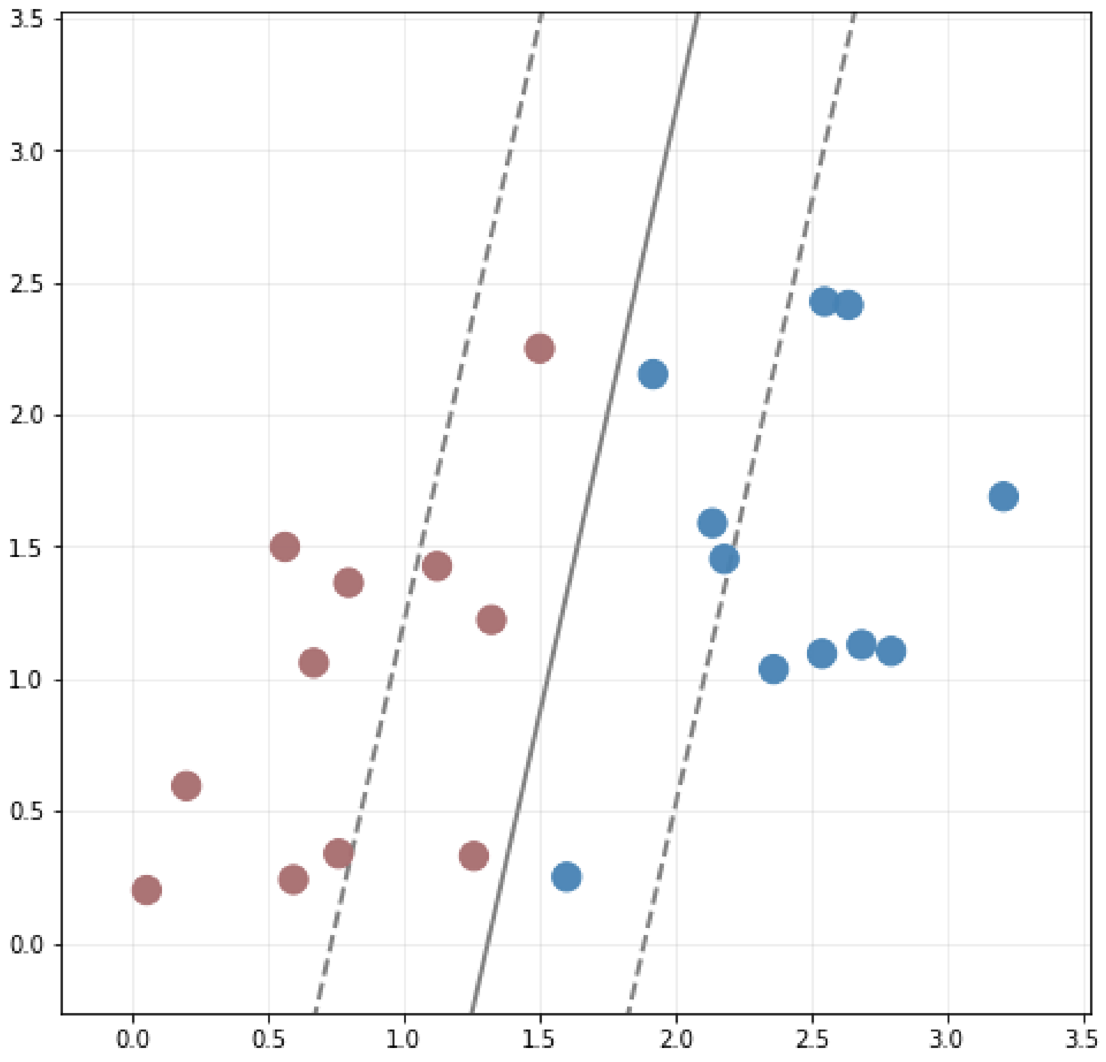
[18]:
```
from sklearn.svm import LinearSVC

# TODO: Train the model and get the parameters, pay attention to the loss␣
 ↪parameter
# your code here
mod_svc = LinearSVC(C=3, loss='hinge')
lsvm = mod_svc.fit(X,y)
# use this code to plot the resulting model
w = lsvm.coef_[0]
b = lsvm.intercept_
print(w,b)
linear_plot(X, y, w=w, b=b)
```

```
[-2.2281118   0.85523586] [2.40253472]
```



[19]:
```
mod_svc = LinearSVC(C=3, loss='squared_hinge')
lsvm = mod_svc.fit(X,y)
# use this code to plot the resulting model
w = lsvm.coef_[0]
b = lsvm.intercept_
print(w,b)
linear_plot(X, y, w=w, b=b)
```

```
[-1.73394899  0.38182408] [2.26942587]
```



Part D: In general, how does the choice of `C` affect the bias and variance of the model? Write your answer in this week's Peer Review assignment.

In this case, you using LinearSVC in sklearn, the larger C results in higher variance and lower bias. A smaller C results in lower variance and higher bias. A small c can allow more slack and is therefore a robust model.

[20]:
```
# testing cell
# your code here
```

1.0.3 Part 3: Nonlinear SVM, Parameter Tuning, Accuracy, and Cross-Validation

---

Any support vector machine classifier will have at least one parameter that needs to be tuned based on the training data. The guaranteed parameter is the C associated with the slack variables in the primal objective function, i.e.

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i$$

If you use a kernel fancier than the linear kernel then you will likely have other parameters as well. For instance in the polynomial kernel $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$ you have to select the shift c and the polynomial degree d. Similarly the rbf kernel

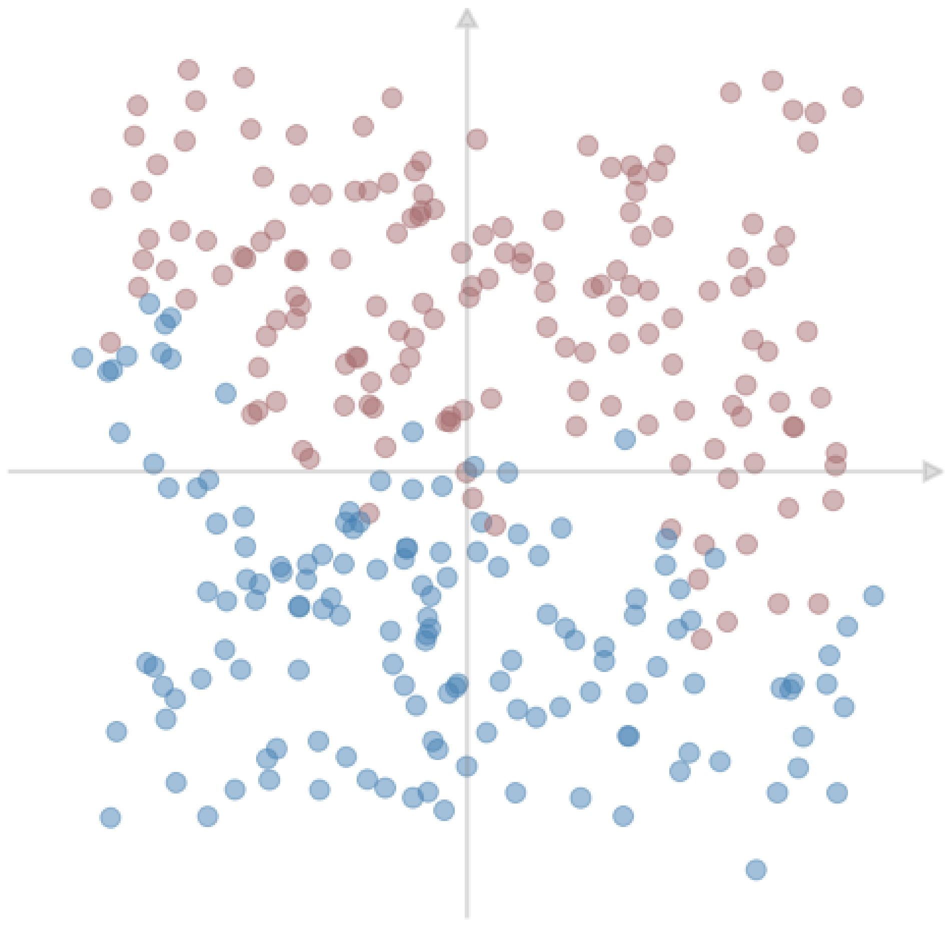
$$K(\mathbf{x},\mathbf{z}) = \exp\left[-\gamma\|\mathbf{x}-\mathbf{z}\|^2\right]$$

has one tuning parameter, namely γ, which controls how fast the similarity measure drops off with distance between $\mathbf{x}$ and $\mathbf{z}$.

For our examples we'll consider the rbf kernel, which gives us two parameters to tune, namely C and γ.

Consider the following two dimensional data

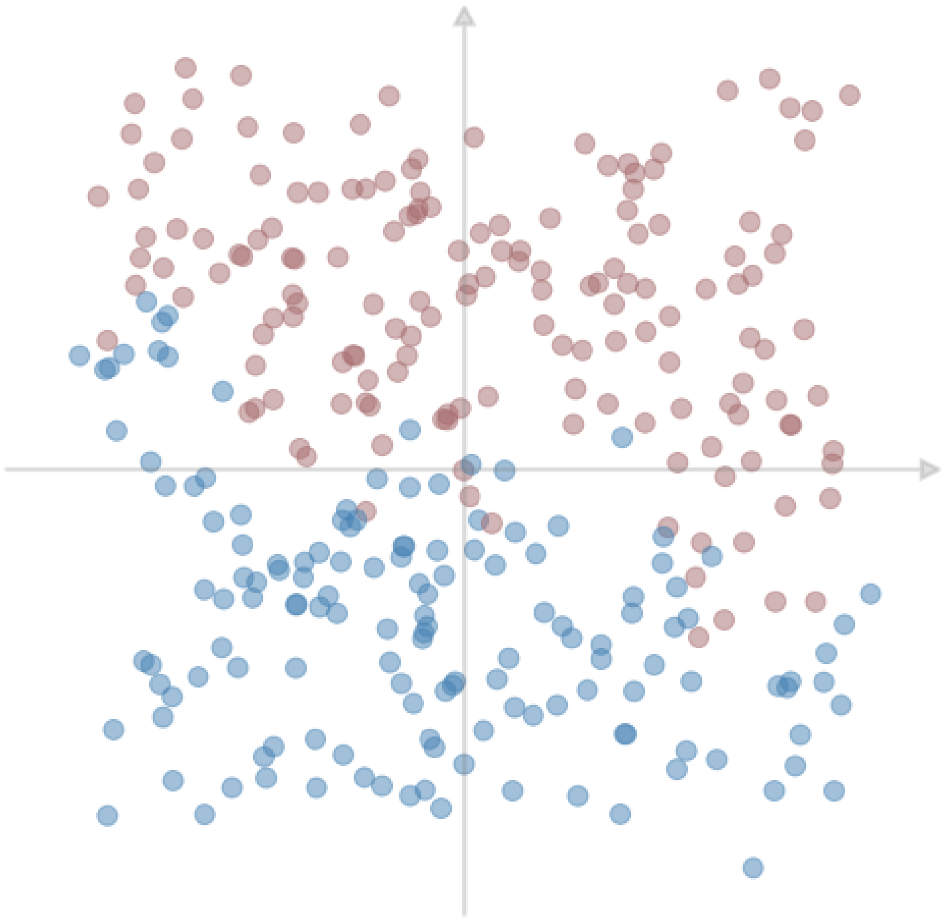
[21]:
```
X, y = part3data(N=300, seed=1235)
nonlinear_plot(X, y)
```



Part A: We can use the method SVC from sklearn's `svm` module to fit an SVM with a nonlinear kernel to the data. Go now and look at the documentation. Note that we pass the `kernel="rbr"` parameter to use the RBF kernel. The other two parameters we'll be concerned with are `C` and the RBF parameter `gamma`.

Write some code to fit an SVM with RBF kernel to the data and plot the results. Use the parameter values `C=1` and `gamma=1`.

[42]:
```
from sklearn.svm import SVC

nlsvm= None
# your code here
nlvsm = SVC(C=1, gamma=1).fit(X,y)
nonlinear_plot(X, y, nlsvm)
```

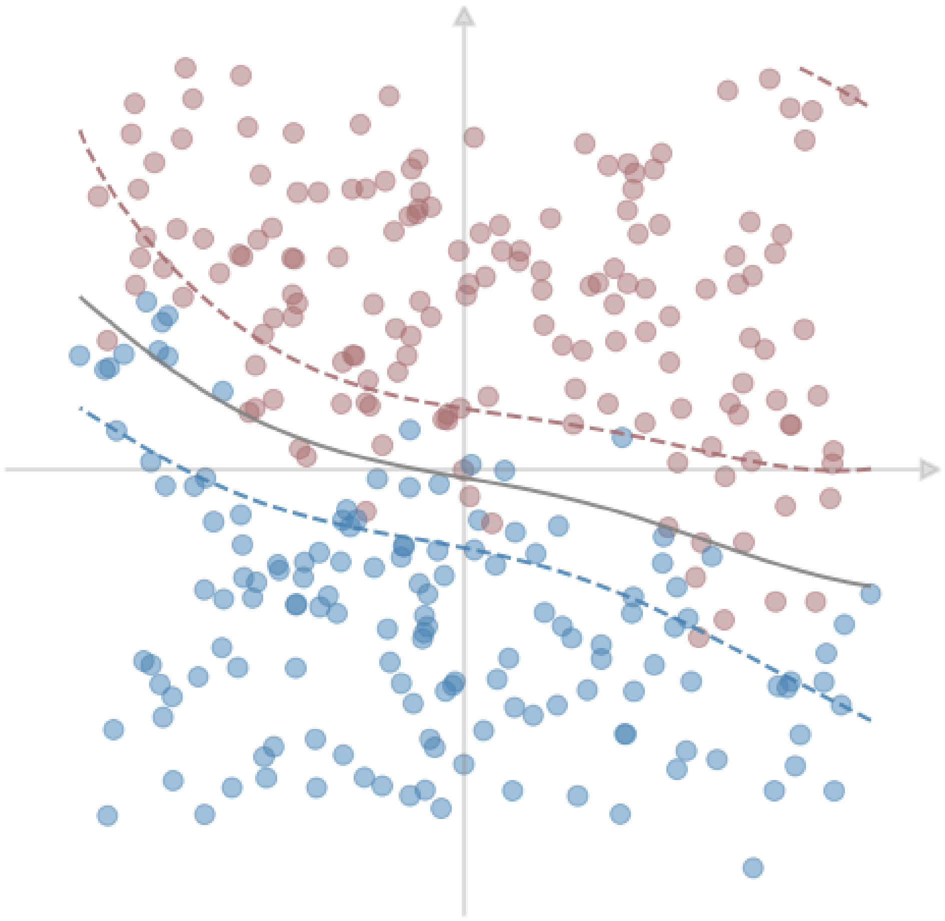


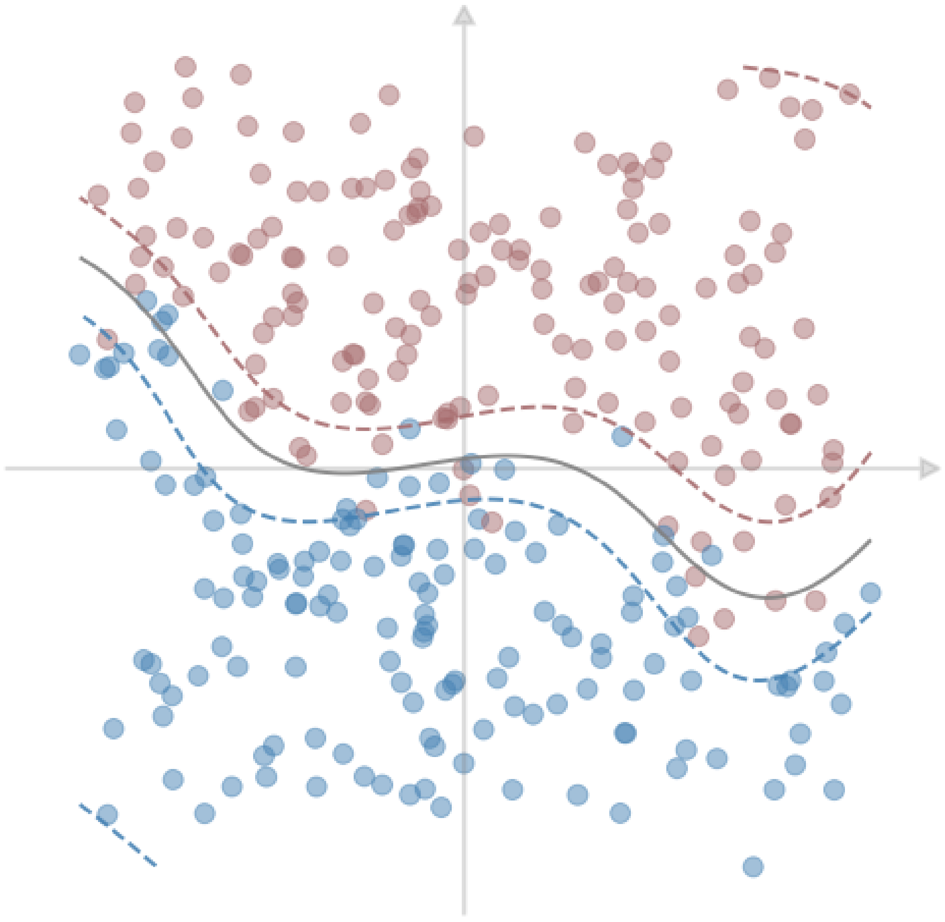
Part B: In this part we'll use cross-validation to estimate the validation accuracy achieved by our model. Experiment with the values of the hyperparameters to see if you can get a good validation accuracy. How do the choice of `C` and `gamma` affect the resulting decision boundary? Write your response in this week's Peer Review assignment.

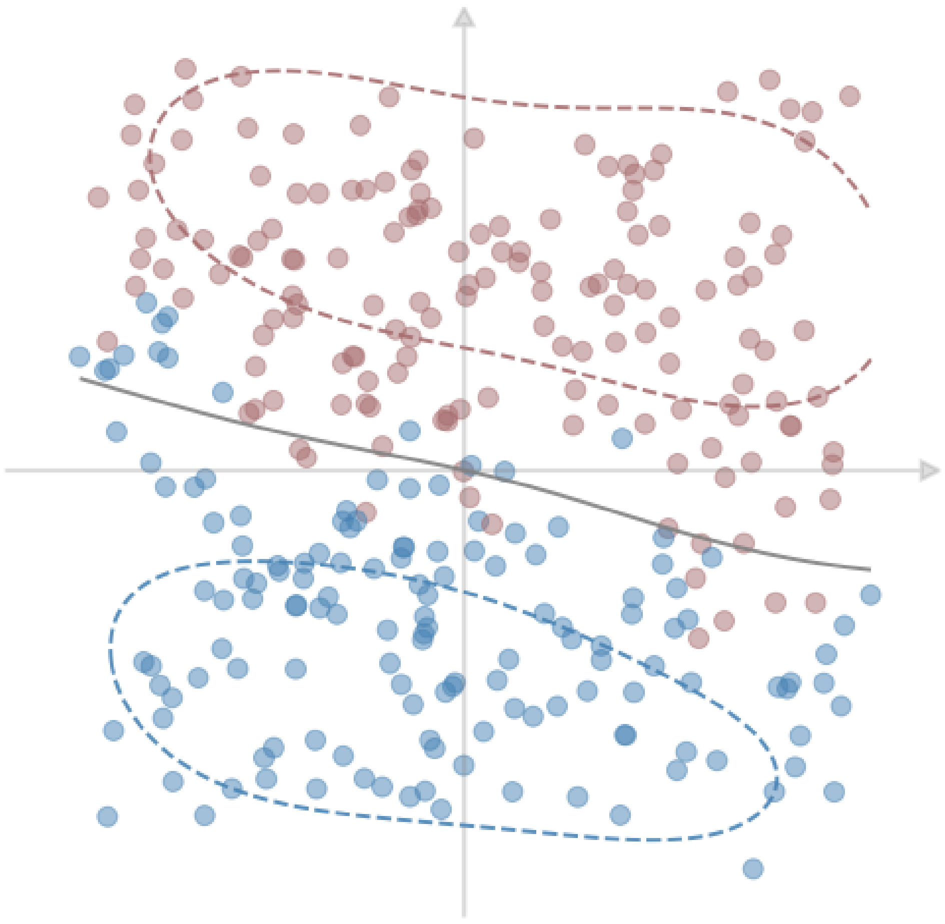
A smaller C value fits more of a smaller kernel to the data points. It results in a small 'circle' decision boundary in each side of the hyperplane. Also, this decision boundary is further away from the hyperplane (larger margins). On the other hand, a larger C value results in a larger kernel(decision boundary) and a smaller distance from the margin. In summary, a smaller C allows more slack (wider soft margin). A larger C has a tighter soft margin. A larger gamma seems to add more curvature to the decision boundary and hyperplane. The margin is smaller and so is the hyperplane. A smaller gamma results in more of a straight line in the decision boundary and the hyperplane. In summary, a larger gamma leads to having more complex decision boundaries.

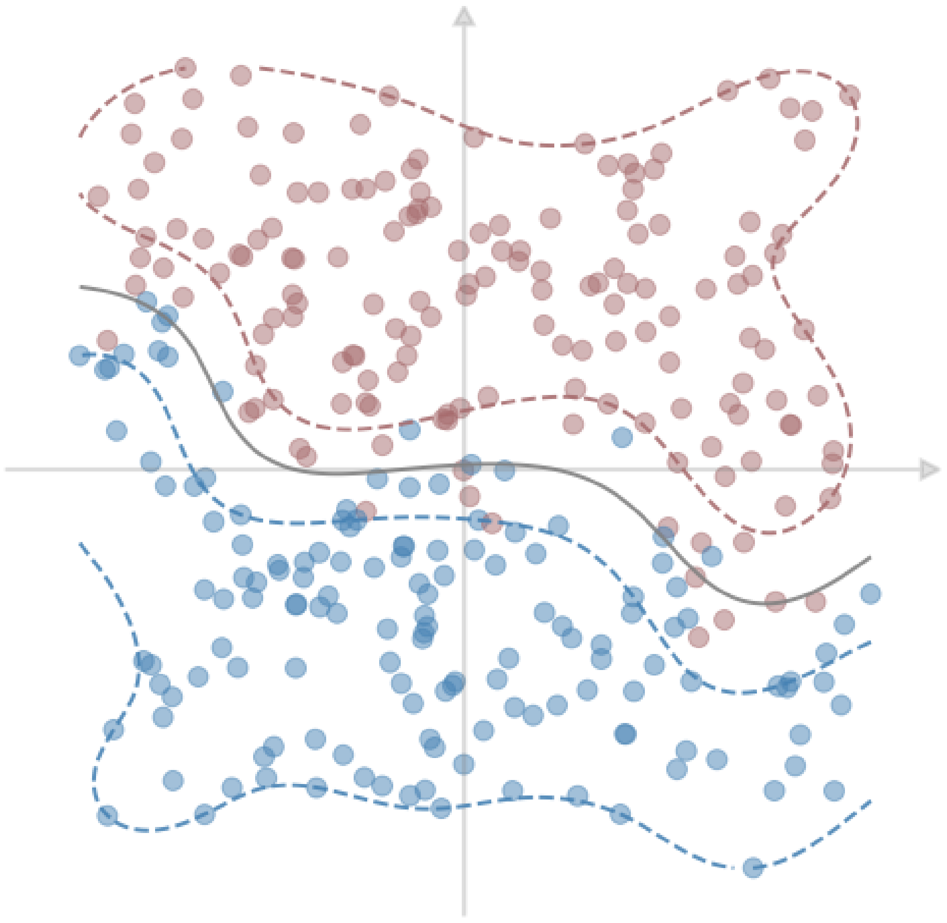
[43]:
```
from sklearn.model_selection import cross_val_score

# your code here

# uncomment and update nlsvm and scores
nlsvm =SVC(C=1, gamma = 1).fit(X,y)
scores= cross_val_score(nlsvm, X, y=y)
print("cross-val mean-accuracy: {:.3f}".format(np.mean(scores)))
nonlinear_plot(X, y, nlsvm)

# C tuning
# your code here
nlsvm =SVC(C=10, gamma = 'scale').fit(X,y)
scores= cross_val_score(nlsvm, X, y=y)
print("cross-val mean-accuracy: {:.3f}".format(np.mean(scores)))
nonlinear_plot(X, y, nlsvm)

nlsvm =SVC(C=0.1, gamma = 'scale').fit(X,y)
scores= cross_val_score(nlsvm, X, y=y)
print("cross-val mean-accuracy: {:.3f}".format(np.mean(scores)))
nonlinear_plot(X, y, nlsvm)

# gamma tuning
# your code here
nlsvm =SVC(C=1, gamma = 5).fit(X,y)
scores= cross_val_score(nlsvm, X, y=y)
print("cross-val mean-accuracy: {:.3f}".format(np.mean(scores)))
nonlinear_plot(X, y, nlsvm)

# your code here
nlsvm =SVC(C=1, gamma = 0.1).fit(X,y)
scores= cross_val_score(nlsvm, X, y=y)
print("cross-val mean-accuracy: {:.3f}".format(np.mean(scores)))
nonlinear_plot(X, y, nlsvm)
```

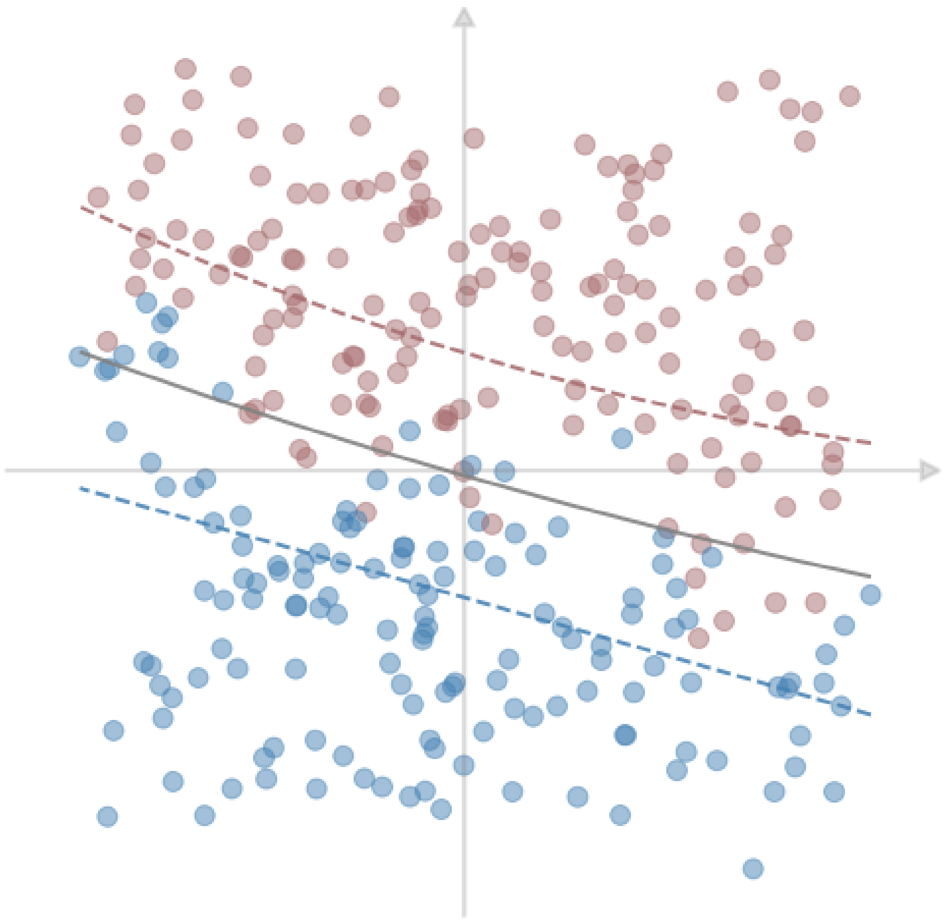
```
cross-val mean-accuracy: 0.920
cross-val mean-accuracy: 0.930
cross-val mean-accuracy: 0.903
cross-val mean-accuracy: 0.933
cross-val mean-accuracy: 0.907
```











Part C: How does the choice of **kernel** function affect the bias/variance of the model? Write your answer in this week's Peer Review assignment.

An expressible (flexible) kernel (such as RBF or polynomial with high order) makes a more complex model- increases variance, reduces bias.

[24]:
```
# instructor testing cell
# your code here
```

1.0.4 Part 4: Automating the Parameter Search

On the previous problem we were able to choose some OK parameters just by hand-tuning. But in real life (where time is money) it would be better to do something a little more automated. One common thing to do is a **grid-search** over a predefined range of the parameters. In this case you will loop over all possible combinations of parameters, estimate the accuracy of your model using K-Folds cross-validation, and then choose the parameter combination that produces the highest validation accuracy.

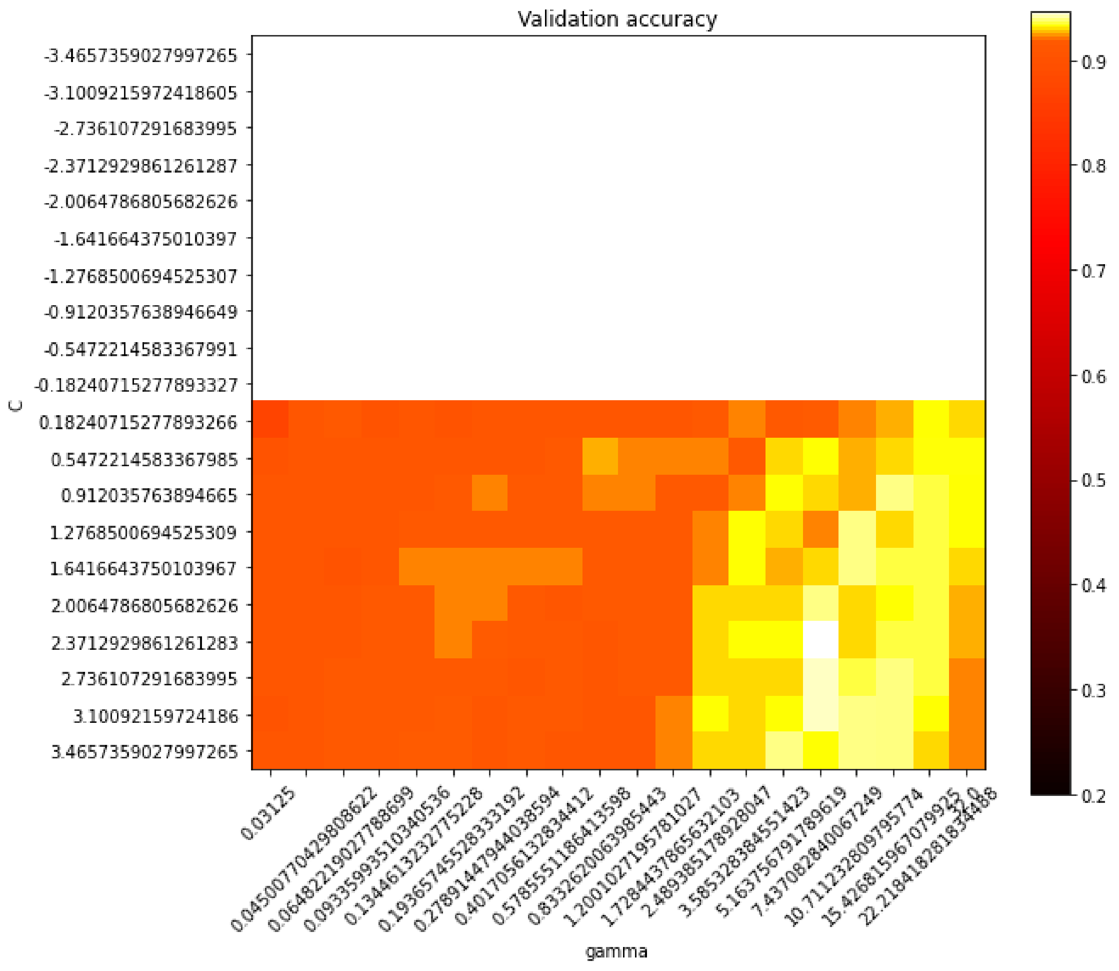
Part A: Below is an experiment where we search over a logarithmic range between 2^{-5} and 2^{5} for C and a range between 2^{-5} and 2^{5} for γ. For the accuracy measure we use K-Folds CV with $K = 3$.

[46]:
```
from sklearn.model_selection import cross_val_score, GridSearchCV
grid=None # ToDo: replace it to proper GridSearchCV object and run the grid␣
↪search with cross validation

# your code here
params = {'C': np.log(np.logspace(-5, 5, base=2, num=20)), 'gamma':np.
↪logspace(-5, 5, base=2, num=20)}
grid = GridSearchCV(SVC(), param_grid=params, cv=3)
grid.fit(X,y)
# )
```

[46]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([-3.4657359 , -3.1009216 , -2.73610…
                          'gamma': array([3.12500000e-02, 4.50077043e-02,
6.48221903e-02, 9.33599351e-02,
       1.34461323e-01, 1.93657455e-01, 2.78914479e-01, 4.01705613e-01,
       5.78555119e-01, 8.33262006e-01, 1.20010272e+00, 1.72844379e+00,
       2.48938518e+00, 3.58532838e+00, 5.16375679e+00, 7.43708284e+00,
       1.07112328e+01, 1.54268160e+01, 2.22184183e+01, 3.20000000e+01])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Part B: The following function will plot a heat-map of the cross-validation accuracies for each combination of parameters. Which combination looks the best? Enter your response in this week's Peer Review assignment.

[47]:
```
plotSearchGrid(grid)
```



```
[ ]:
```

```
[ ]: # instuctor testing cell
     # your code here
```

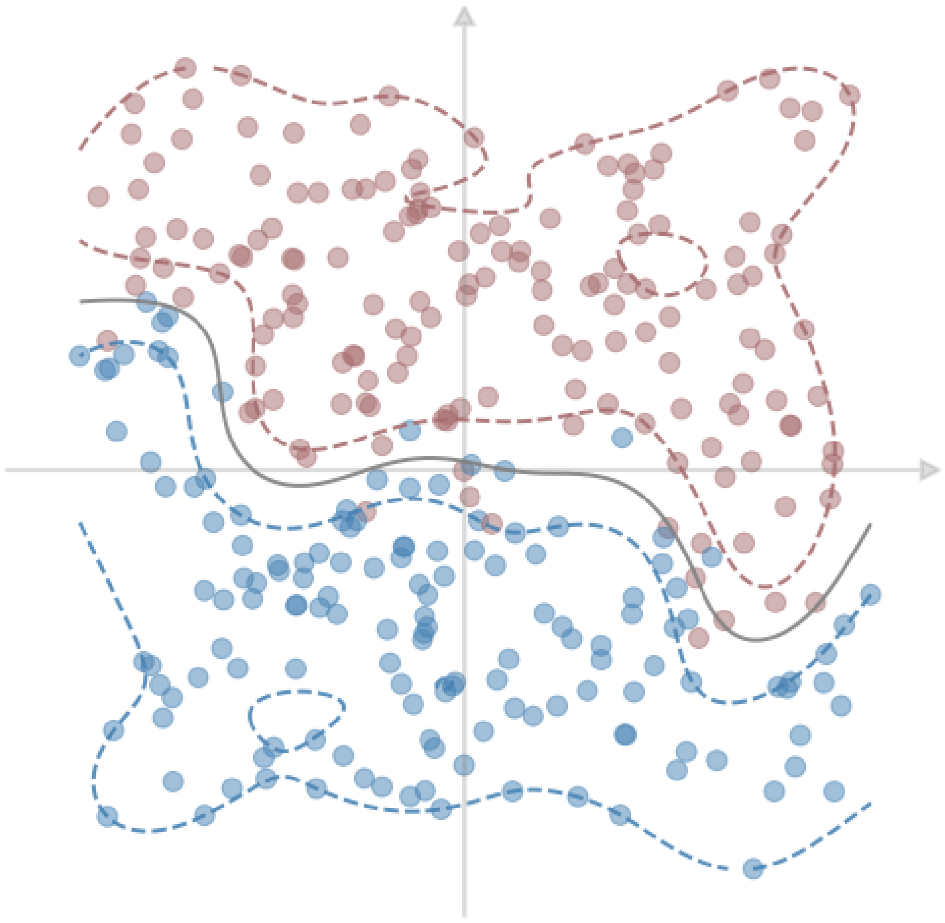
Part C: The GridSearchCV object scores, among other things, the best combination of parameters as well as the cross-validation accuracy achieved with those parameters. Print those quantities for our model.

```
[52]: # print best paramters and best accuracy
      # your code here
      print(grid.best_params_)
      print(grid.best_score_)
```

```
{'C': 2.3712929861261283, 'gamma': 7.437082840067249}
0.9466666666666667
```

Part D [Peer Review]: The GridSearchCV object also stores the classifier trained with the best hyperparameters. Pass this best estimator into the `nonlinear_plot` function to view the best decision boundary. Answer the Peer Review questions for this section. What can you tell about the best decision boundary?

```
[55]: # pass GridSearchCV object best estimator into nonlinear_plot
      # your code here
      nonlinear_plot(X, y, grid.best_estimator_)
```



```
[ ]:
```