MLPy Workshop 7

Hariaksh Pandya - s2692608, Alexandru Girban - s2148980

March 7, 2025

1 Week 7 - Support Vector Machines

1.0.1 Aims

The main concepts covered in this notebook are:

- understanding separable vs non-separable data
- implementing SVMs
- use of different kernels and parameter tuning in SVMs

1. Setup
2. SVC
3. Model assessment
4. Default Data

This week, we will be exploring the basics of support vector machine models. We will be focusing on support vector machines for classification, which is provided by sklearn in the `SVC` model. For more details, please see https://scikit-learn.org/stable/modules/svm.html

The main class that we are using is sklearn.svm.SVC

As usual, during workshops, you will complete the worksheets together in teams of 2-3, using **pair programming**. When completing worksheets:

- You will have tasks tagged by (CORE) and (EXTRA).
- Your primary aim is to complete the (CORE) components during the WS session, afterwards you can try to complete the (EXTRA) tasks for your self-learning process.

Instructions for submitting your workshops can be found at the end of worksheet. As a reminder, you must submit a pdf of your notebook on Learn by 16:00 PM on the Friday of the week the workshop was given.

2 Setup

2.1 Packages

Let's load the some of the packages needed for this workshop.

[1]:

```python
# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# sklearn modules
import sklearn
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC         # SVM
from sklearn.preprocessing import StandardScaler # scaling features
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV, KFold, StratifiedKFold
```

2.2 Helper Functions

This helper function plots the data and visualized the decision boundary and margin.

[2]:

```python
from sklearn.inspection import DecisionBoundaryDisplay
import sklearn.preprocessing

# Visualize the decision boundary and margin
# For D=2 inputs and binary classification
def plot_margin(model, X, y, figsize=(8,7)):

    fig, ax = plt.subplots(1,1,figsize=figsize)

    # Scatter plot of the inputs colored by class
    ax.scatter(X[:,0], X[:,1], c=y, s=30)

    # Show decsision boundary
    DecisionBoundaryDisplay.from_estimator(
        model,
        X,
        plot_method="contour",
        colors="k",
        levels=[-1, 0, 1],
        linestyles=["--", "-", "--"],
        ax=ax,
    )

    # Highlight support vectors
    # If pipeline with StandardScalar, inverse transform the support vectors
    if (isinstance(model, sklearn.pipeline.Pipeline)):
        if (isinstance(model[0], sklearn.preprocessing.StandardScaler)):
            support_vectors = model[0].inverse_transform(model[-1].support_vectors_)
        else:
            support_vectors = model[-1].support_vectors_
    else:
        support_vectors = model.support_vectors_
    ax.scatter(
        support_vectors[:, 0],
        support_vectors[:, 1],
        s=100,
        linewidth=1,
        facecolors="none",
        edgecolors="k",
    )
    plt.show()
```

3 Support Vector Classifier

The class SVC implements support vector classifiers and support vector machines. When creating an SVC object, various options are available, including:

- C: the inverse regularizaton parameter. NOTE: this defaults to C=1 but should always be tuned.
- kernel: options include the **linear** kernel (linear), **polynomial** kernel (poly), **radial basis function** kernel (rbf), **sigmoid** kernel (sigmoid), or a user-defined kernel.
- degree: degree if using the **polynomial** kernel
- gamma: kernel coefficient for **rbf**, **polynomial**, or **sigmoid** kernels.
- coef0: additional coefficient term for the **polynomial** or **sigmoid** kernels.

After calling .fit(), the SVC object will have a number of attributes including:

- support_vectors_: containing the support vectors.

To predict the class labels from the fitted SVC model, we can call .predict(). And although SVMs only provide class labels and not the corresponding class probabilities, SVC provides a method to estimate the class probablities by calling .predict_proba() (or .predict_log_proba() on the log scale) using Platt scaling. This uses cross-validation and makes the implementation slower, thus to turn on this option of estimating the class probabilities, you must first set probability=True when creating the SVC object. Also note that these probability estimates are unreliable on small datasets.

More details on **kernels** are available here: https://scikit-learn.org/stable/modules/svm.html#svm-kernels

3.0.1 Difference between SVC and LinearSVC

Note the sklearn implements two linear support vector classification models LinearSVC() and SVC(kernel='linear'), which yield slightly different decision boundaries, due to the following differences:

- `LinearSVC` (based on LIBLINEAR) is **faster** than `SVC` (based on LIBSVM)
- `LinearSVC` minimizes the squared hinge loss while SVC minimizes the regular hinge loss.
- `LinearSVC` uses the One-vs-Rest scheme for multiclass classification while `SVC` uses the One-vs-One scheme for multiclass classification.
- `LinearSVC` does not provide some of the attributes of `SVC`, such as the support vectors.

For further details, see the documentations of the two classes

- `SVC` with `kernel=linear`: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- `LinearSVC`: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

In the following, we will focus on the `SVC` class.

3.1 Linearly Separable Data

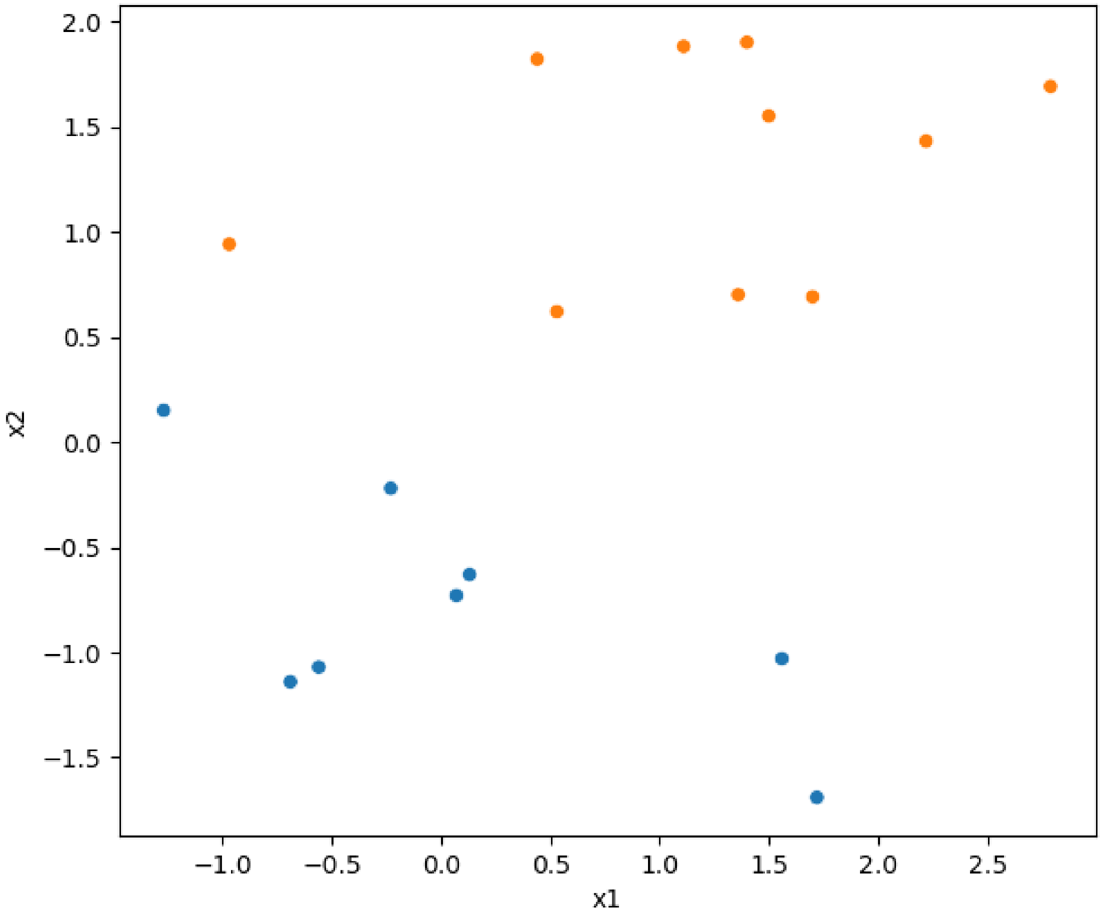
We will begin by examining various toy data sets to explore the basics of these models. For the first example, we will read in data from `ex1.csv`.

[3]:
```
ex1 = pd.read_csv("ex1.csv")
ex1.head()
```

[3]:
```
     x1    x2  y
0 -0.56 -1.07  A
1 -0.23 -0.22  A
2  1.56 -1.03  A
3  0.07 -0.73  A
4  0.13 -0.63  A
```

Plotting the data below, we can see the that data is composed of two classes in two dimensions, and it is clear that these two classes are linearly separable.

[4]:
```
plt.figure(figsize=(7,6))
sns.scatterplot(x='x1', y='x2', hue='y', data=ex1, legend=False)
plt.show()
```



Now, let separate the features and outcome and encode the outcome to a binary vector.

[5]:

```
# Extracting the features and output and encoding y
X_ex1 = np.array(ex1.drop('y', axis=1))
y_ex1 = LabelEncoder().fit_transform(ex1.y)

print(X_ex1.shape)
print(y_ex1.shape)
```

```
(18, 2)
(18,)
```

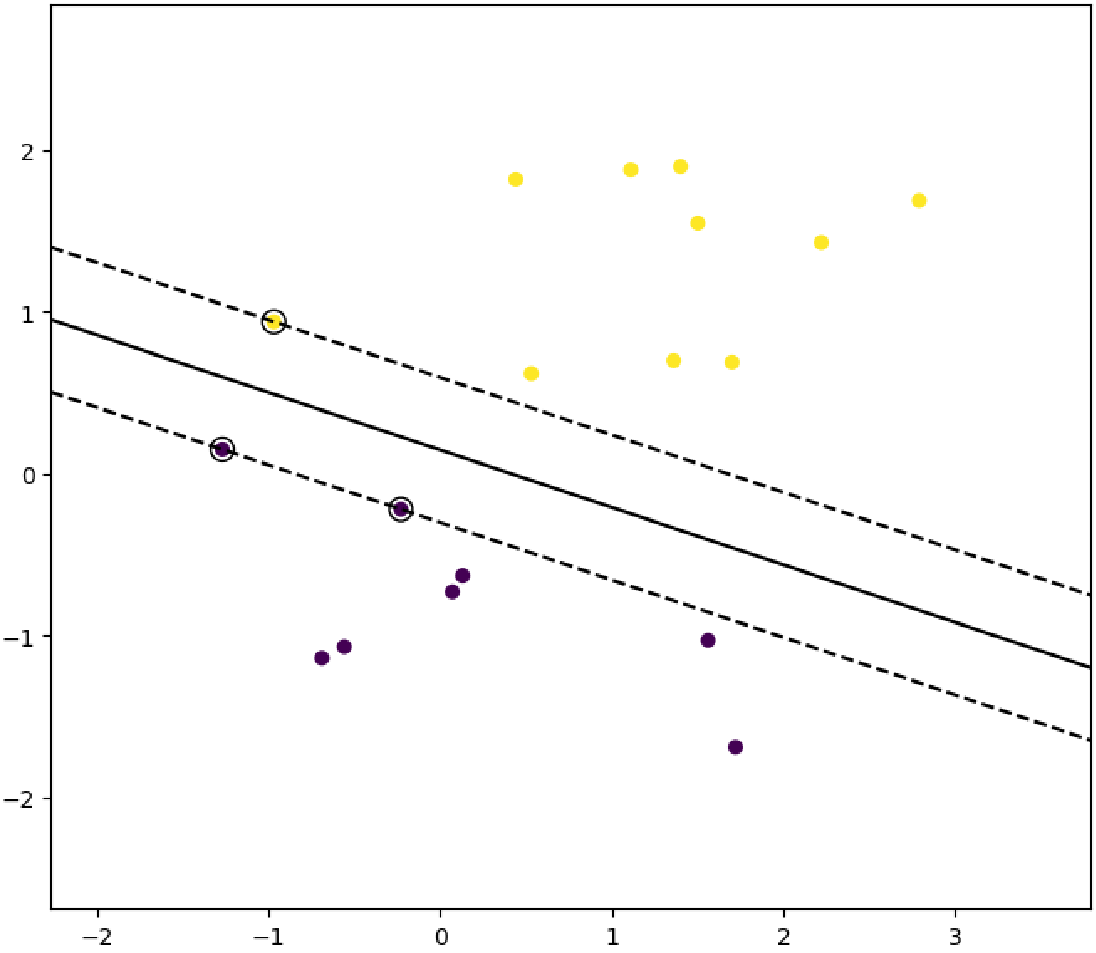
3.1.1 Exercise 1 (CORE)

a. Create and fit an SVC model using a **linear kernel** and `C=100`.

b. Visualize the decision boundary, margin, and support vectors using the function `plot_margin` defined above. How many support vectors are there for each class? Are they on right side of the margin? hyperplane?

[6]:

```
#making the svc model and training it for c = 100
svc_model1 = SVC(kernel='linear', C=100).fit(X_ex1, y_ex1)

# Visualize the decision boundary, margin and support vectors
plot_margin(svc_model1, X_ex1,y_ex1)
```



We can observe 3 support vectors, 2 of which are purple while one of it is yellow, they are one the right side of the hyperplane, essentially the dots on the margin are the support vectors.

3.1.2 Exercise 2 (CORE)

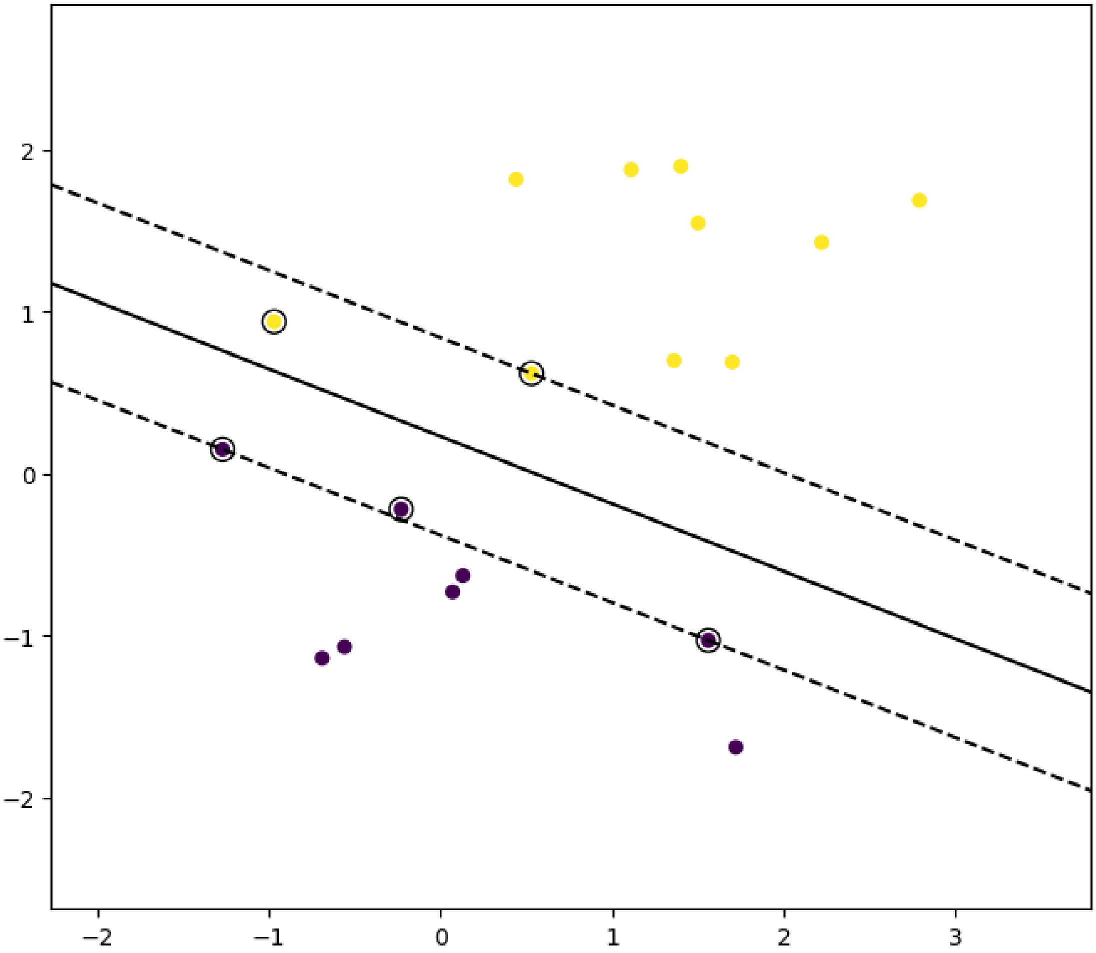
Now, let's see how the results change with a small value of `C`.

a. Create and fit an SVC model using a **linear kernel** and `C=1`.

b. Visualize the decision boundary, margin, and support vectors. Now, how many support vectors are there for each class? Are they on right side of the margin? hyperplane? How has the margin changed?

[7]:

```
#making the svc model and training it for c = 1
svc_model2 = SVC(kernel='linear', C=1).fit(X_ex1, y_ex1)

# Visualize the decision boundary, margin and support vectors
plot_margin(svc_model2, X_ex1,y_ex1)
```

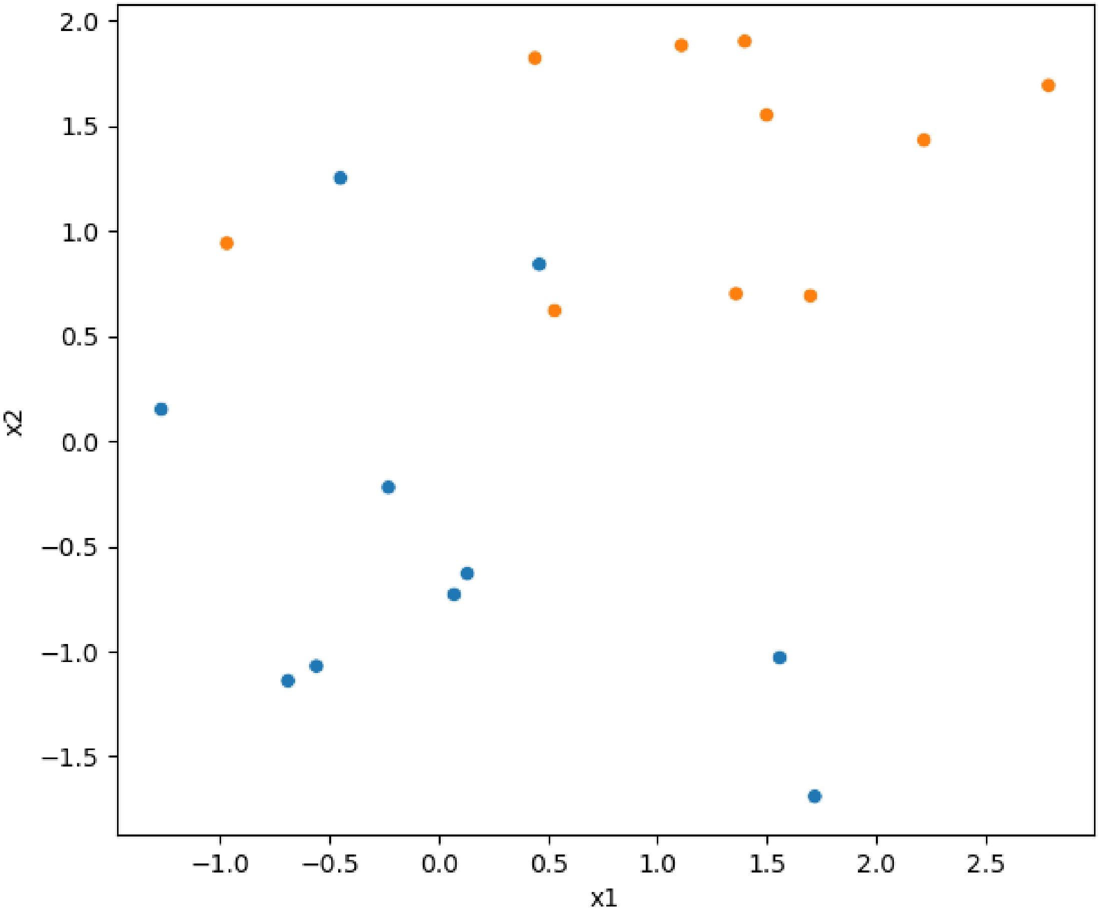


We can observe that as the value of C reduces the margins are more relaxed i.e they have widen, now here we have 5 support vectors, 3 from purple class and 2 from yellow class, it seems that reducing the C relaxes our model, this might help with overfitting since at c=100 we can see that the model is trained quite strictly

3.2 Non-Separable Data

Next, we complicate our previous example by adding two additional points from class A to our data, which result in data that are no longer linearly separable.

[8]:
```
# Read in the data
ex2 = pd.read_csv("ex2.csv")

# Visualize the data
plt.figure(figsize=(7,6))
sns.scatterplot(x='x1', y='x2', hue='y', data=ex2, legend=False)
plt.show()

# Extracting the features and output and encoding y
X_ex2 = np.array(ex2.drop('y', axis=1))
y_ex2 = LabelEncoder().fit_transform(ex2.y)
```

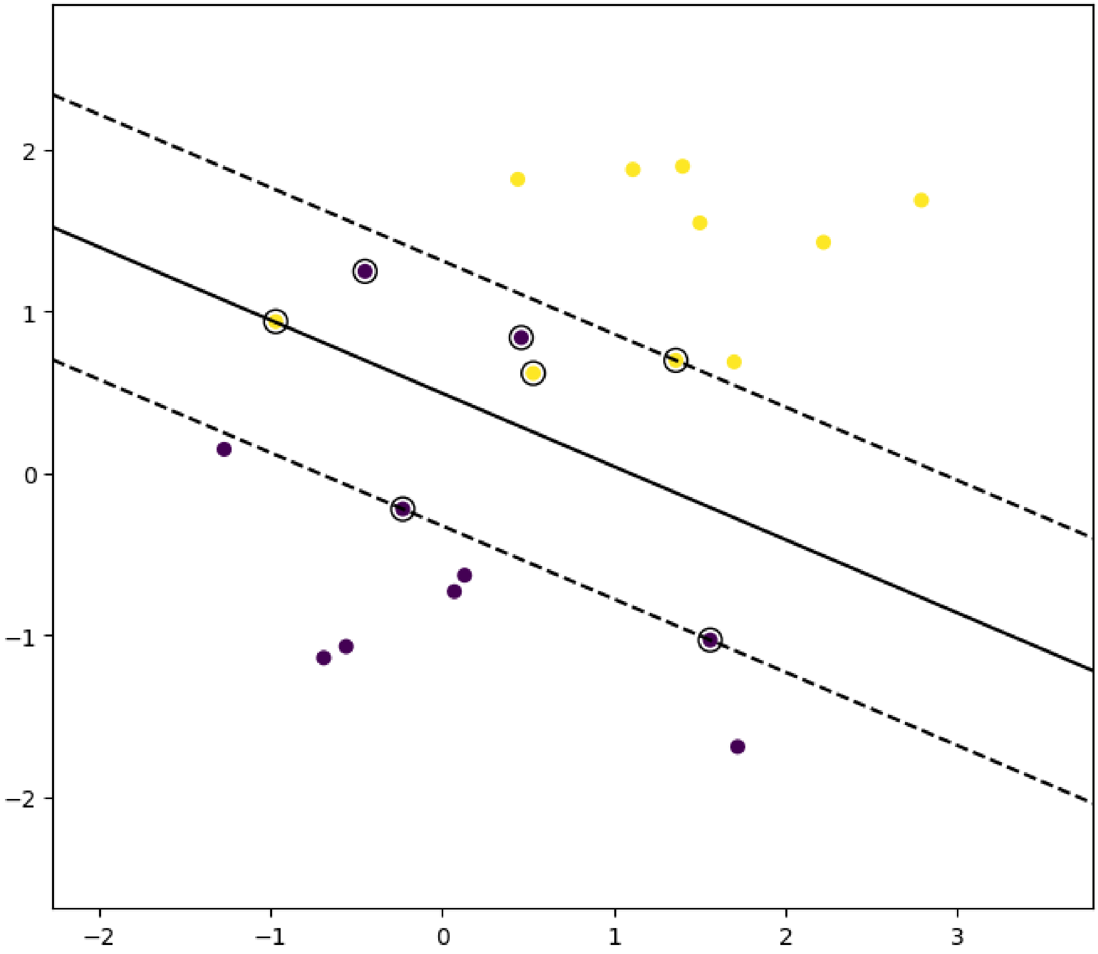


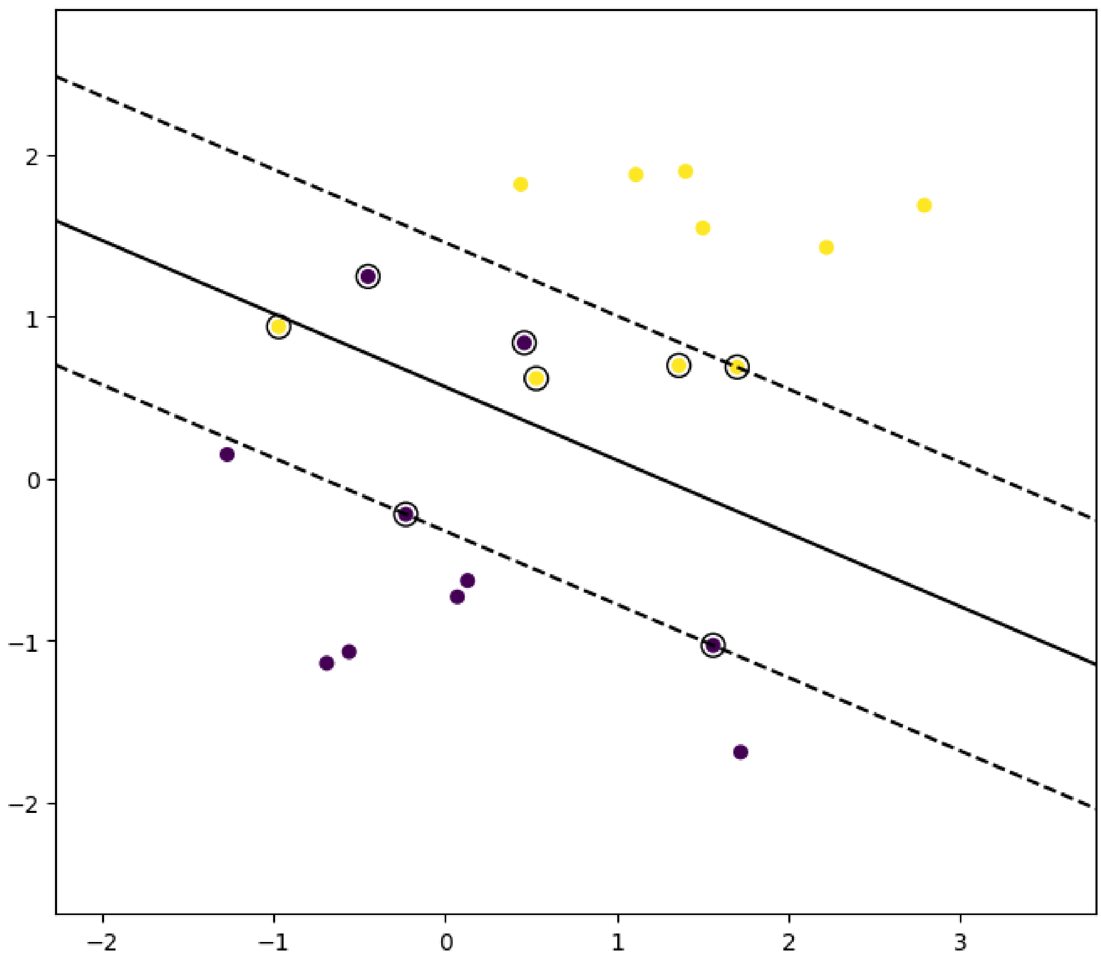
3.2.1 Exercise 3 (CORE)

Fit the SVC model using the **linear kernel** and **same choices of `C=100` and `C=1`**. How have the results changed? Comment on changes related to the margin and the number of support vectors for each class and their location relative to the margin and hyperplane.

[9]:

```
#making the svc model and training it for c = 100
svc_nonsep1 = SVC(kernel='linear', C=100).fit(X_ex2, y_ex2)

# Visualize the decision boundary, margin and support vectors
plot_margin(svc_nonsep1, X_ex2,y_ex2)

#making the svc model and training it for c = 1
svc_nonsep2 = SVC(kernel='linear', C=1).fit(X_ex2, y_ex2)

# Visualize the decision boundary, margin and support vectors
plot_margin(svc_nonsep2, X_ex2,y_ex2)
```





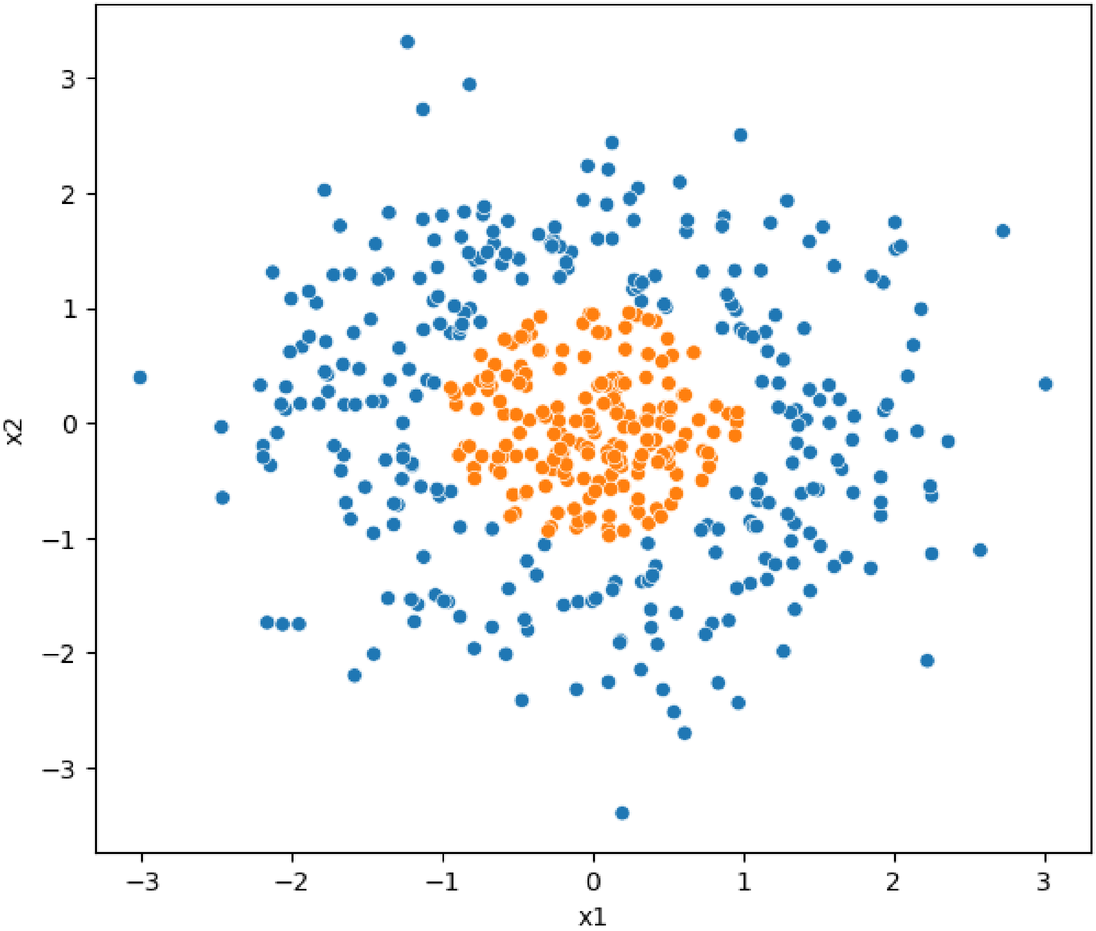
- For c=100
  - we have 7 support vectors, 4 of purple and 3 of yellow
  - The 2 purple support vectors are misaligned i.e on the different side of the hyperplane or the decision boundary, while one of the yellow dots is exactly on the boundary
- For c=100
  - we can notice that the margins have further increase in width and the number of support vectors have increased to 8, 4 purple and 4 yellow
  - Here we have 1 misplaced yellow dot while 2 misplaced purple dot
- we can see that linear kernal does not work well with non seperable dataset, The kernal tries and maintains a straight line but it doesnt fit well, This can be seen by increased number of support vectors and misplaced classes.

3.3 Nonlinear Separation

Next, we will look at a new data set that is not linearly separable, but can be perfectly separated by a nonlinear decision boundary. We will start by loading and visualizing the data.

[10]:

```
# Load the third data set
ex3 = pd.read_csv("ex3.csv")

# Visualize the data
plt.figure(figsize=(7,6))
sns.scatterplot(x='x1', y='x2', hue='y', data=ex3, legend=False)
plt.show()

# Extracting the features and output and encoding y
X_ex3 = np.array(ex3.drop('y', axis=1))
y_ex3 = LabelEncoder().fit_transform(ex3.y)
```

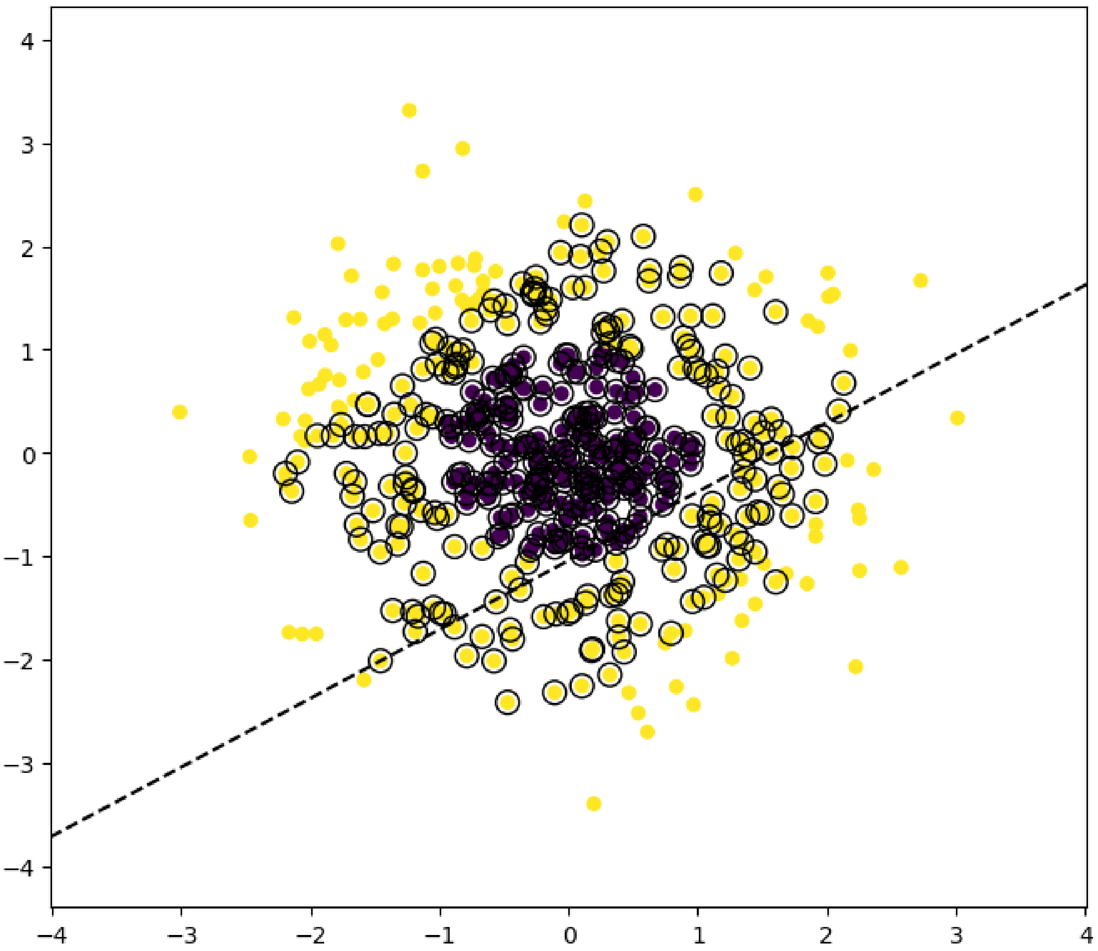


Let's start by trying to fit a linear SVC. As expected, the results are terrible and the number of support vectors is very large.

[11]:

```
# Fit the SVC with a linear kernel on the new data
svm_lin = SVC(kernel='linear', C=100).fit(X_ex3, y_ex3)
```

```
# Visualize the decision boundary, margin and support vectors
plot_margin(svm_lin, X_ex3,y_ex3)
```

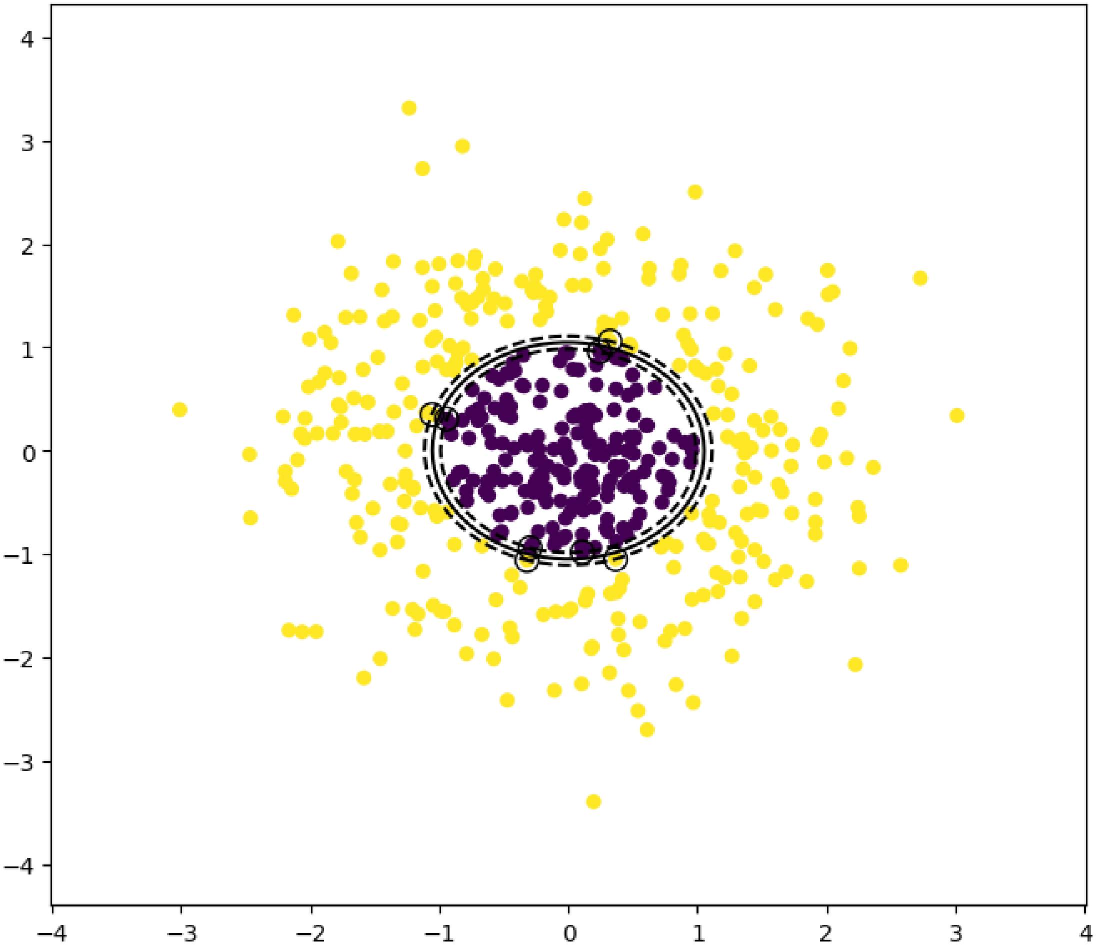


3.3.1 Exercise 4 (CORE)

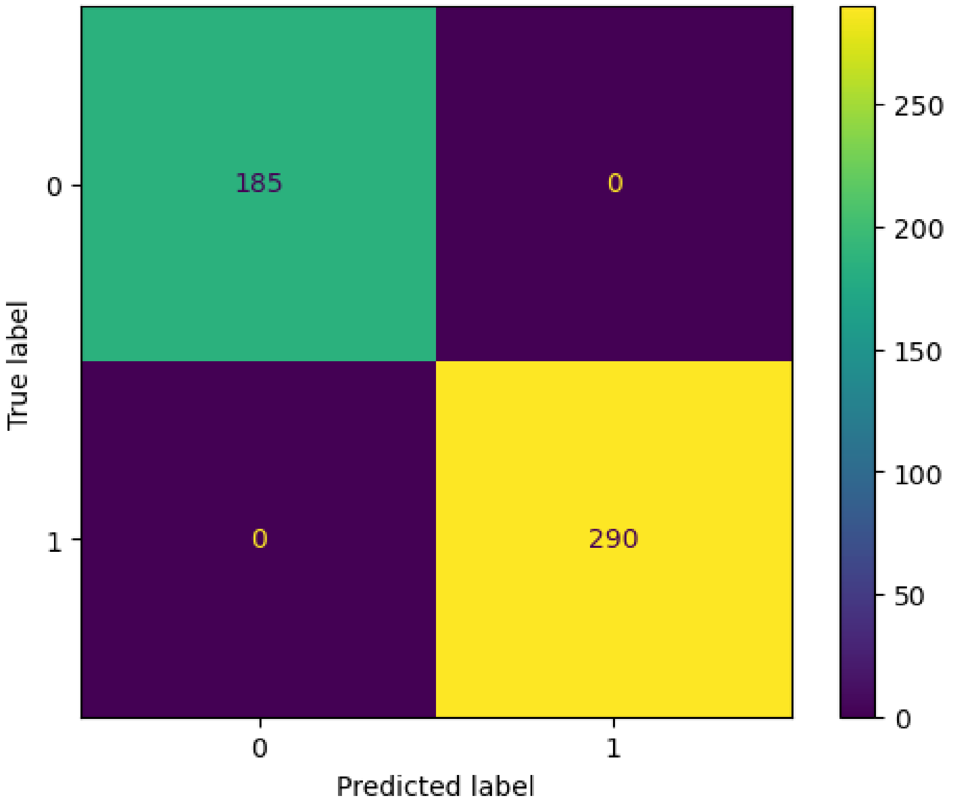
- Fit an SVM with a **polynomial kernel with degree 2** and `C=100`.
- Visualize the margin and decision boundary and comment on the number of support vectors compared to the linear model.
- Compute and visualize the confusion matrix using `ConfusionMatrixDisplay.from_estimator`. Is the fitted SVM able to perfectly separate the classes?

[12]:
```
# Fit the SVC with a linear kernel on the new data
svm_poly = SVC(kernel='poly',degree = 2, C=100).fit(X_ex3, y_ex3)

# Visualize the decision boundary, margin and support vectors
plot_margin(svm_poly, X_ex3,y_ex3)
```

```
num_sv_poly = svm_poly.support_vectors_.shape[0]
print(f"Number of support vectors (Polynomial Kernel, degree=2): {num_sv_poly}")


ConfusionMatrixDisplay.from_estimator(svm_poly, X_ex3, y_ex3)
plt.show()
```



```
Number of support vectors (Polynomial Kernel, degree=2): 8
```

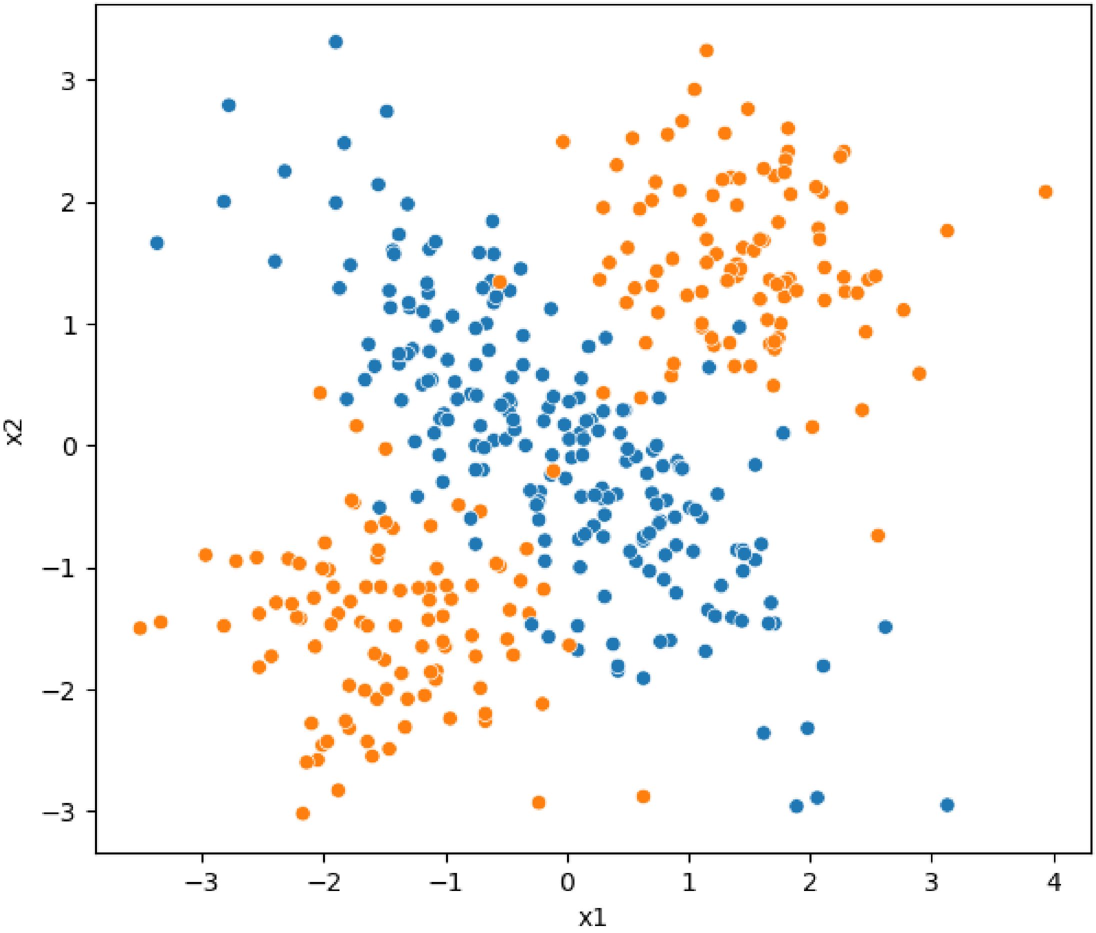


- The linear model had alot of support vectors yet it failed due to the nature of nonlinear dataset, while a polynomial kernel with degree 2 was able to completely seperate/classify the dataset with 8 support vectors.
- This can be observed by the confusion matrix, since the off-diagonal entries are 0 it means we dont have any misplaced classes, the polynomial kernel works for the given dataset.

3.4 Kernels for SVMs and Parameter-Tuning

Next, we will consider an more complicated data, and explore using grid search to tune the model parameters and the effect of different kernels.

[14]:
```
# Load the fourth data set
ex4 = pd.read_csv("ex4.csv")

# Visualize the data
plt.figure(figsize=(7,6))
sns.scatterplot(x='x1', y='x2', hue='y', data=ex4, legend=False)
plt.show()

# Extracting the features and output and encoding y
X_ex4 = np.array(ex4.drop('y', axis=1))
y_ex4 = LabelEncoder().fit_transform(ex4.y)
```



Let's start with the polynomial kernel that we saw in the previous exercise and use cross-validation to tune both the degree of polynomial and the penalty parameter.

[15]:
```
# SVM with polynomial kernel
svm = SVC(kernel='poly', coef0=1)

# Grid search over C and the degree of the polynomial
degrees = [1,2,3,4]
C = np.linspace(0.1, 10, 100)
cv = GridSearchCV(
    svm,
    param_grid = {'C': C,
        'degree': degrees},
    cv = KFold(5, shuffle = True, random_state = 0)
)

# Fit and tune the model
cv.fit(X_ex4, y_ex4)
```

```
# Get the best model parameters and the accuracy of the model
print("Params: ", cv.best_params_)
print("Avg Accuracy: ", cv.best_score_)
```

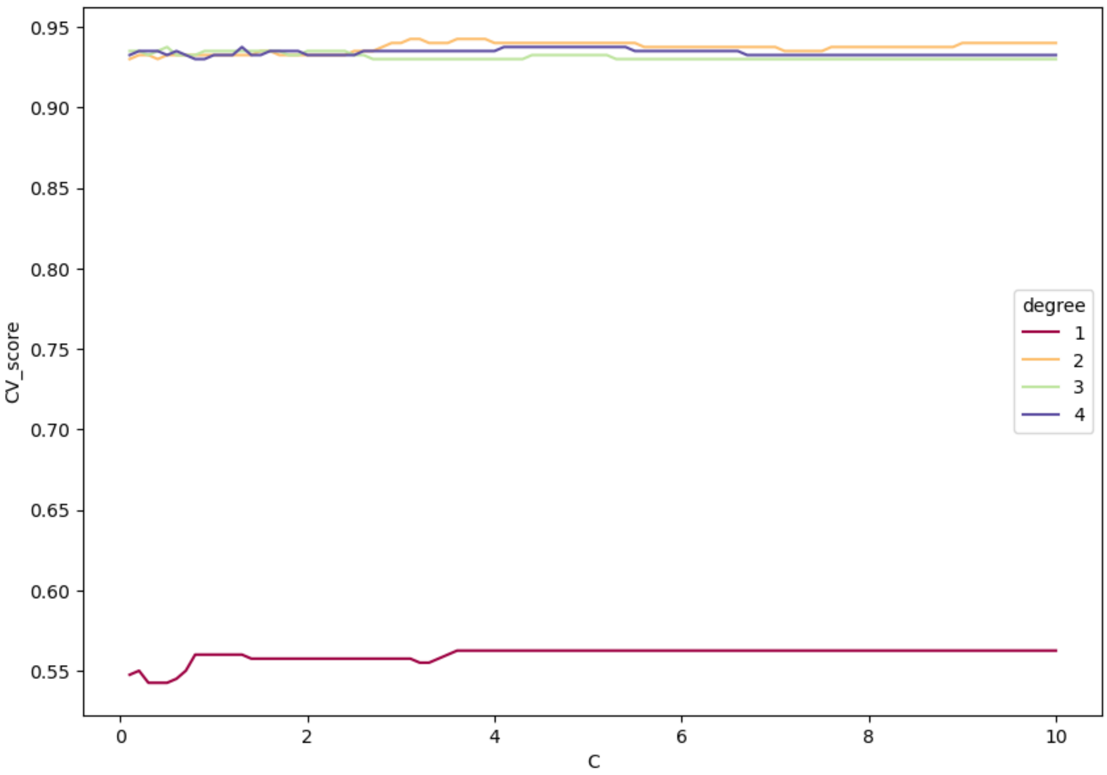
```
Params:  {'C': 3.1, 'degree': 2}
Avg Accuracy:  0.9425000000000001
```

3.4.1 Exercise 5 (CORE)

- Run the following code to plot the CV accuracy. Based on this plot, would you use the best parameter values printed above or choose different values? Why?
- For your selected parameters, fit the svm and plot the decision boundary and margin.

[16]:

```
# Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv.cv_results_
                           ).filter(['param_C',␣
 ↪'param_degree','mean_test_score']
                                    ).rename(columns={'param_C':'C',␣
 ↪'param_degree':'degree','mean_test_score':'CV_score'})

# Plot the CV scores
plt.figure(figsize=(10,7))
sns.lineplot(x='C', y='CV_score', data = cv_accuracy, hue ='degree',␣
 ↪palette="Spectral")
plt.show()
```

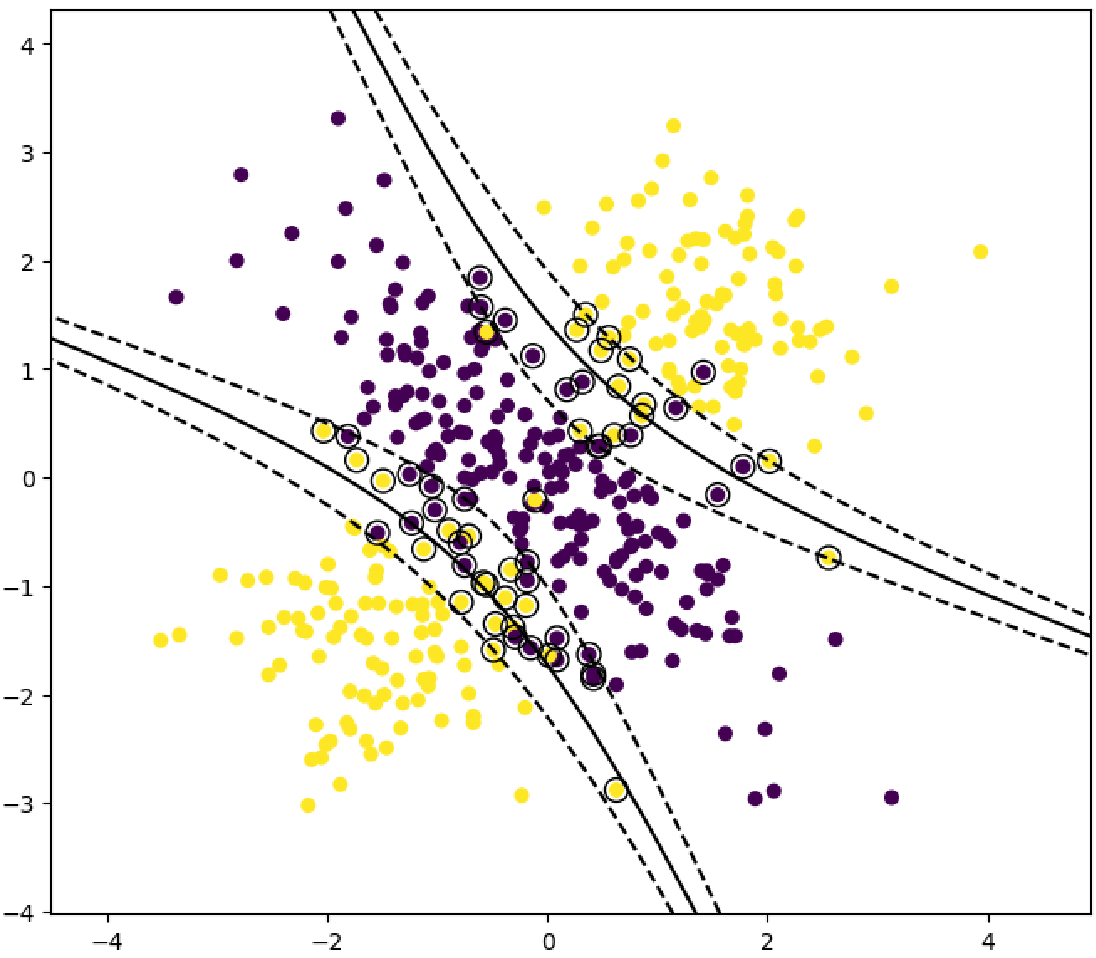


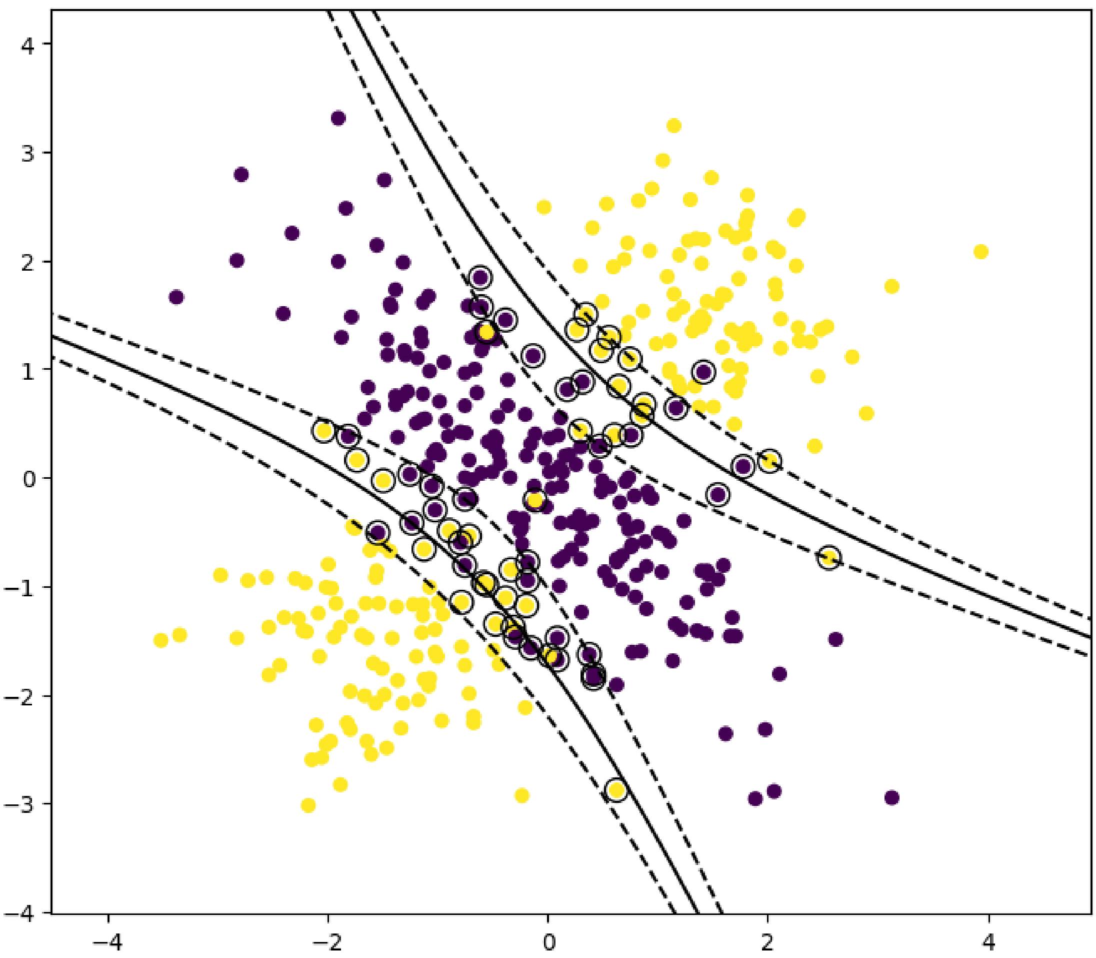
- We can observe that C=3.1, C=4 and C=10 of polynomial degree 2 have similar accuracy
- Furthermore we can observe that linear model does not work for the given dataset
- Additionally polynomial with degree 3 and 4 perform comparatively better compared to linear dataset but they dont perform as well as polynomial of degree 2 when the value of C increases, This could be potentially due to overfitting, which inturn reduces our accuracy in this case.
- One of the ways to confirm which C value would be appropriate would be via confusion matrix
- Furthermore just from glancing the cv_accuracy plot, it seems the C values of 3.1,4,10 would have similar performance despite the difference in the margin fitness/width
- If the conclusion of the confusion matrix is similar, the C=3.1 will be the most appropriate since it would help us deal with overfitting even if it has increased number of support vectors.

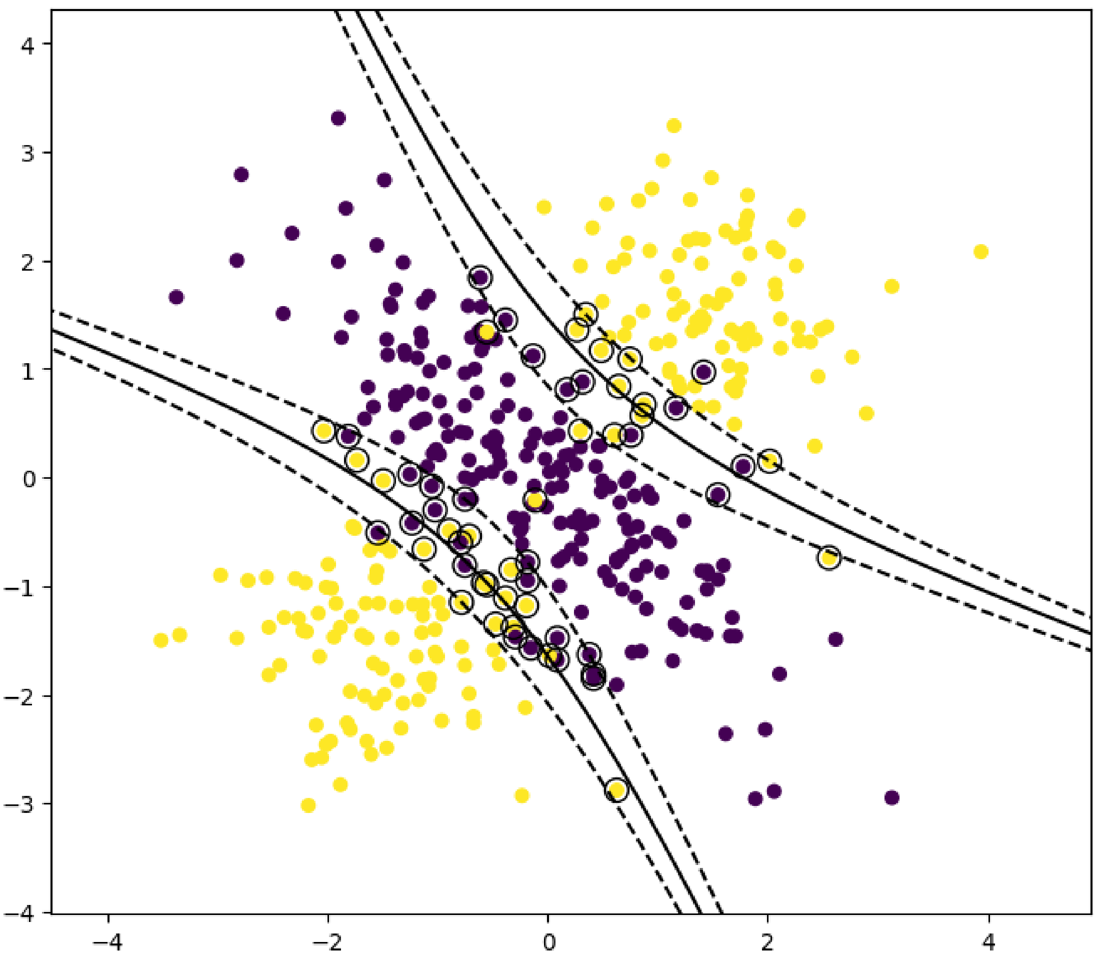
[17]:

```
# Fit the SVC with a linear kernel on the new data
svm_poly31 = SVC(kernel='poly',degree = 2, C=3.1, coef0=1).fit(X_ex4, y_ex4)
svm_poly4 = SVC(kernel='poly',degree = 2, C=4, coef0=1).fit(X_ex4, y_ex4)
svm_poly10 = SVC(kernel='poly',degree = 2, C=10, coef0=1).fit(X_ex4, y_ex4)




# Visualize the decision boundary, margin and support vectors
plot_margin(svm_poly31, X_ex4,y_ex4)
plot_margin(svm_poly4, X_ex4,y_ex4)
plot_margin(svm_poly10, X_ex4,y_ex4)
```

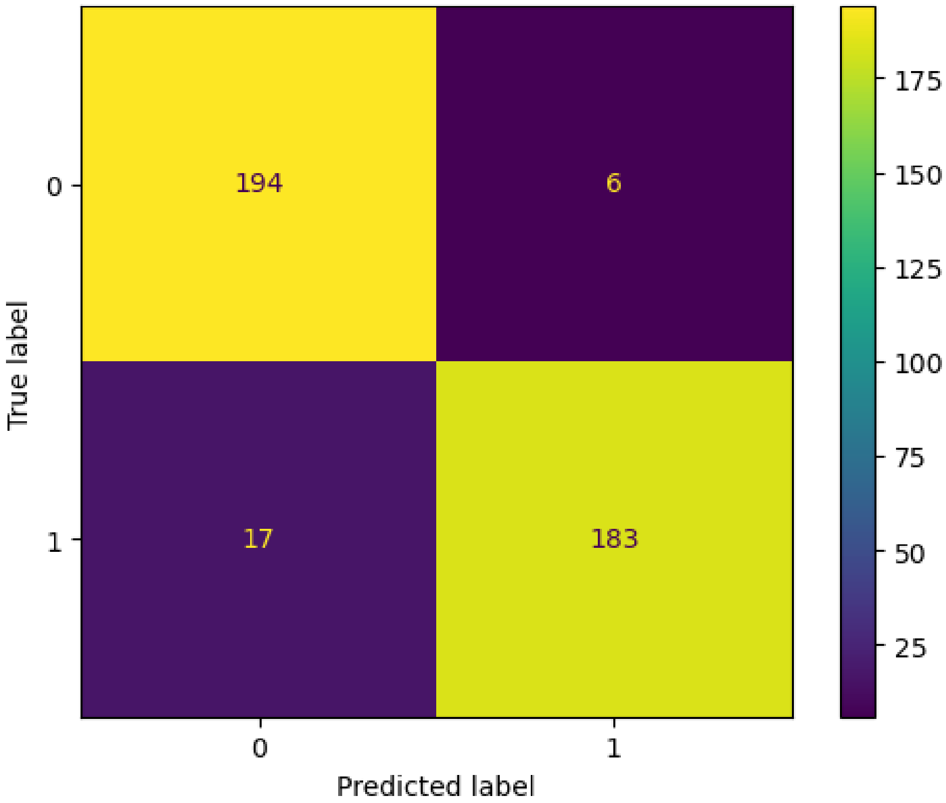
```
num_sv_poly31 = svm_poly31.support_vectors_.shape[0]
print(f"Number of support vectors (Polynomial Kernel, degree=2, c = 3.1):␣
 ↪{num_sv_poly31}")

num_sv_poly4 = svm_poly4.support_vectors_.shape[0]
print(f"Number of support vectors (Polynomial Kernel, degree=2, c = 4):␣
 ↪{num_sv_poly4}")

num_sv_poly10 = svm_poly10.support_vectors_.shape[0]
print(f"Number of support vectors (Polynomial Kernel, degree=2, c = 10):␣
 ↪{num_sv_poly10}")


ConfusionMatrixDisplay.from_estimator(svm_poly31, X_ex4, y_ex4)
plt.show()


ConfusionMatrixDisplay.from_estimator(svm_poly4, X_ex4, y_ex4)
plt.show()


ConfusionMatrixDisplay.from_estimator(svm_poly10, X_ex4, y_ex4)
plt.show()
```

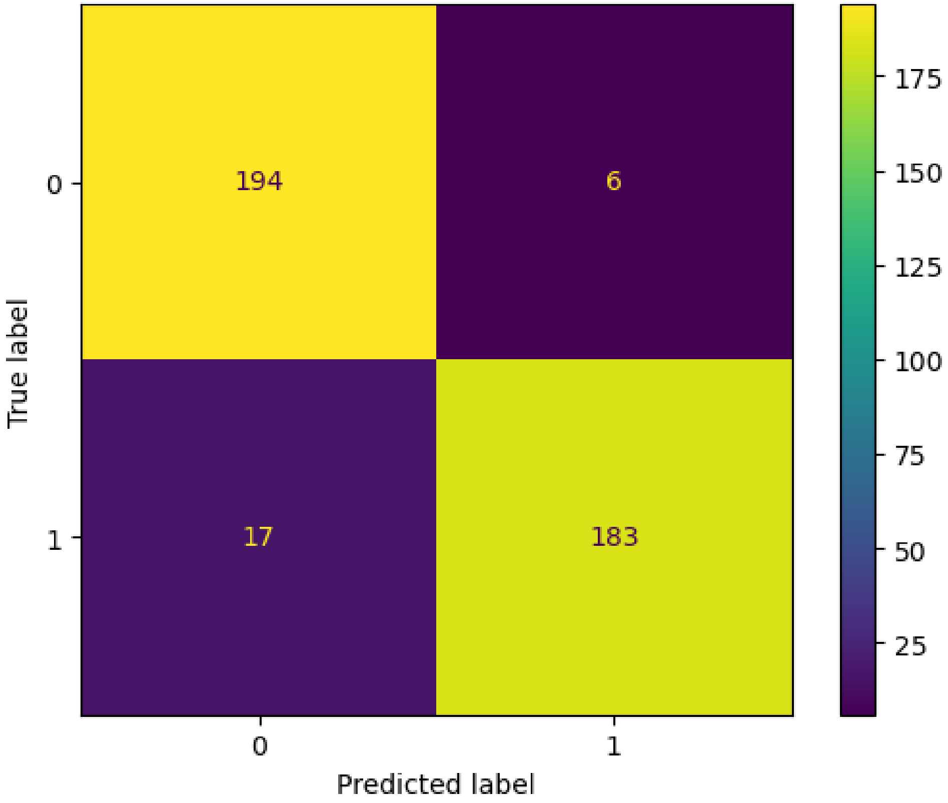

4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

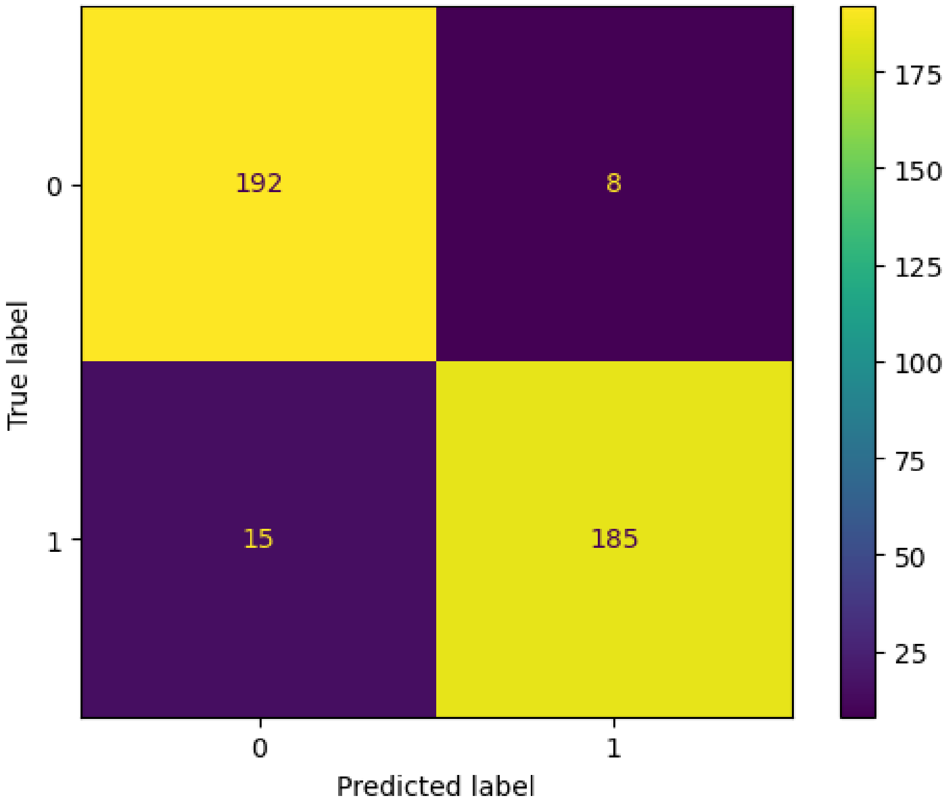

4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4



```
Number of support vectors (Polynomial Kernel, degree=2, c = 3.1): 62
Number of support vectors (Polynomial Kernel, degree=2, c = 4): 60
Number of support vectors (Polynomial Kernel, degree=2, c = 10): 57
```


194
6
17
183
0
1
True label
0
1
Predicted label
175
150
125
100
75
50
25


194
6
17
183
True label
0
1
Predicted label
0
1
175
150
125
100
75
50
25



- The best parameter is indeed C = 3.1, we can see that for different values of C we have the following support vectors:
  - Number of support vectors (Polynomial Kernel, degree=2, c = 3.1): 62
  - Number of support vectors (Polynomial Kernel, degree=2, c = 4): 60
  - Number of support vectors (Polynomial Kernel, degree=2, c = 10): 57
- Furthermore on the basis of the confusion matrix we can observe that they have the same number of misaligned classes.

3.4.2 Exercise 6 (EXTRA)

- Repeat the grid search above with a polynomial kernel but set the coefficent term `coef0=0` (this is the default value). Redraw the plot of the CV accuracy. How have the results changed? Can you explain why?

```
[ ]:
```

3.4.3 Exercise 7 (CORE)

Consider instead the RBF kernel.

- Use grid search to tune both the penalty parameter C and the inverse bandwidth parameter γ.

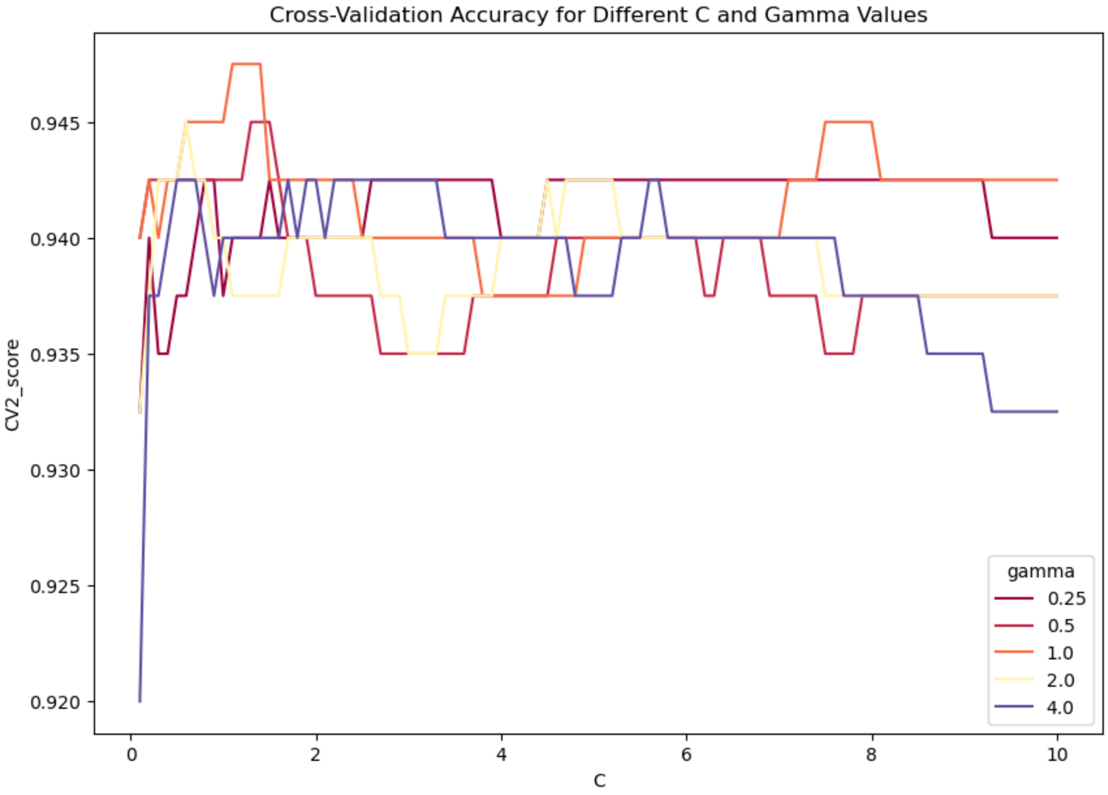
- Plot the decision boundary. How do the results compare to the SVM with a polynomial kernel?

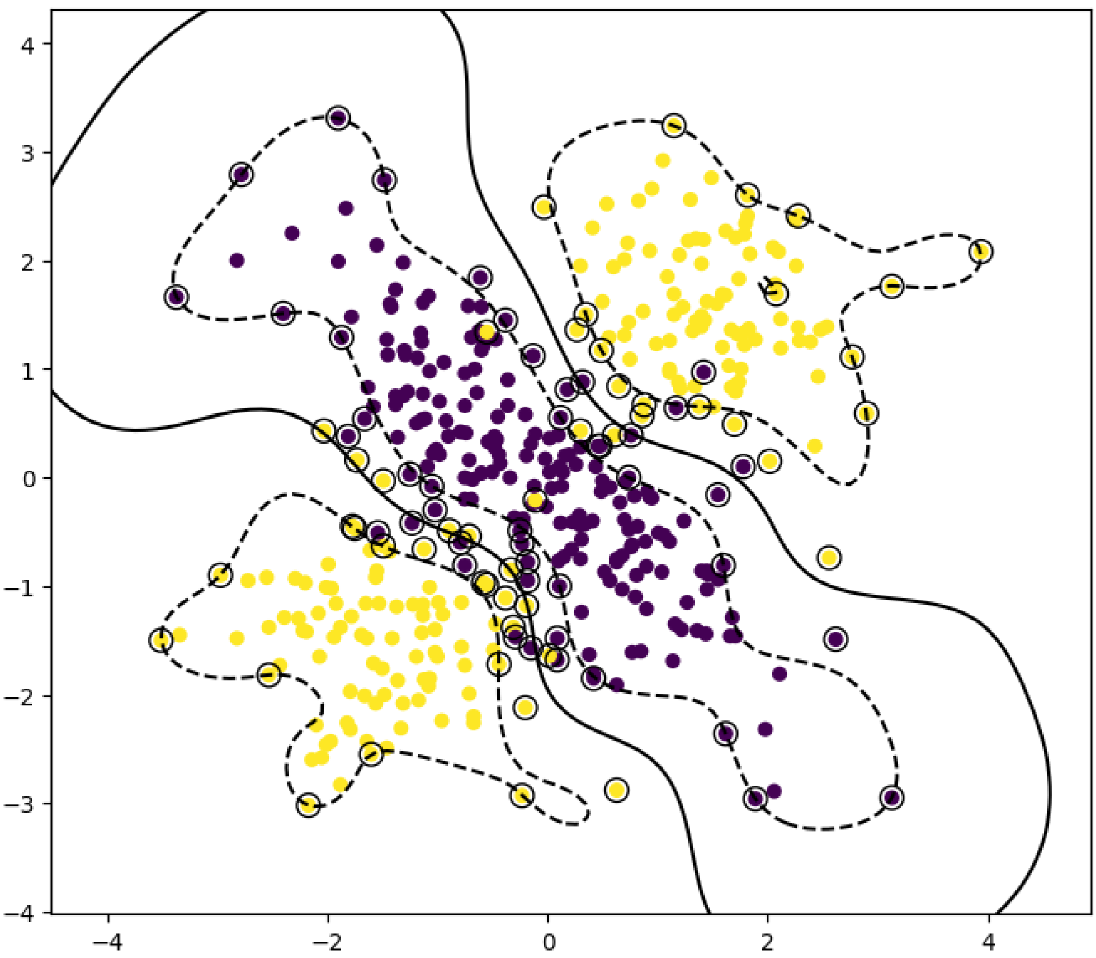
[18]:

```
# SVM with rbf
svm_rbf = SVC(kernel='rbf')

# Grid search over C and gamma (inverse bandwidth parameter)
gamma = [1/4, 1/2, 1, 2, 4]  # Different gamma values
C = np.linspace(0.1, 10, 100)  # Different regularization parameters

# Gridsearch CV wowowow
cv2 = GridSearchCV(
    svm_rbf,
    param_grid={'C': C, 'gamma': gamma},
    cv=KFold(5, shuffle=True, random_state=0)
)

# Fitting and tunning our model
cv2.fit(X_ex4, y_ex4)

# Obtaining best parameters
print("Best Parameters: ", cv2.best_params_)
print("Average Cross-Validation Accuracy: ", cv2.best_score_)

# Preping for data viz
cv2_accuracy = pd.DataFrame(cv2.cv_results_).filter(
    ['param_C', 'param_gamma', 'mean_test_score']
).rename(columns={'param_C': 'C', 'param_gamma': 'gamma', 'mean_test_score':␣
 ↪'CV2_score'})

# Plotting the Cross validation scores
plt.figure(figsize=(10, 7))
sns.lineplot(x='C', y='CV2_score', data=cv2_accuracy, hue='gamma',␣
 ↪palette="Spectral")
plt.title("Cross-Validation Accuracy for Different C and Gamma Values")
plt.show()

# Training new model on the basis of best parameters
best_svm_rbf = SVC(kernel='rbf', C=cv2.best_params_['C'], gamma=cv2.
 ↪best_params_['gamma'])
best_svm_rbf.fit(X_ex4, y_ex4)

# Plotting for best parameters
plot_margin(best_svm_rbf, X_ex4, y_ex4)

print(f"Number of support vectors : {best_svm_rbf.support_vectors_.shape[0]}")
```

```
Best Parameters:  {'C': 1.1, 'gamma': 1}
```

```
Average Cross-Validation Accuracy:  0.9475
```





```
Number of support vectors : 92
```

- The decision boundary of RBF is highly flexible and non-linear, adapting to the complex shape of the data compared to the polynomial despite it fitting the data well enough, but it doesnt do a better job compared to RBF for this dataset.
- The curvy contours suggest that RBF is effectively separating clusters with complex boundaries.
- More support vectors are used at the margins compared to the polynomial SVM, while polynomial had less support vectors it means it that polynomial had stricter margins.
- No. of support vector for RBF = 92 vs polynomial = 62

3.5 Model Assessment

As, we learned last week, there are many metrics to consider beyond accuracy. For example, below we compute and print the classification report, summarizing the results for precision, recall, f1-score, and accuracy.

[19]:
```
from sklearn.metrics import classification_report

# Classification report for the SVM with polynomial kernel
print(classification_report(y_ex4, cv.best_estimator_.predict(X_ex4)))

# Classification report for the SVM with RBF kernel
print(classification_report(y_ex4, cv2.best_estimator_.predict(X_ex4)))
```

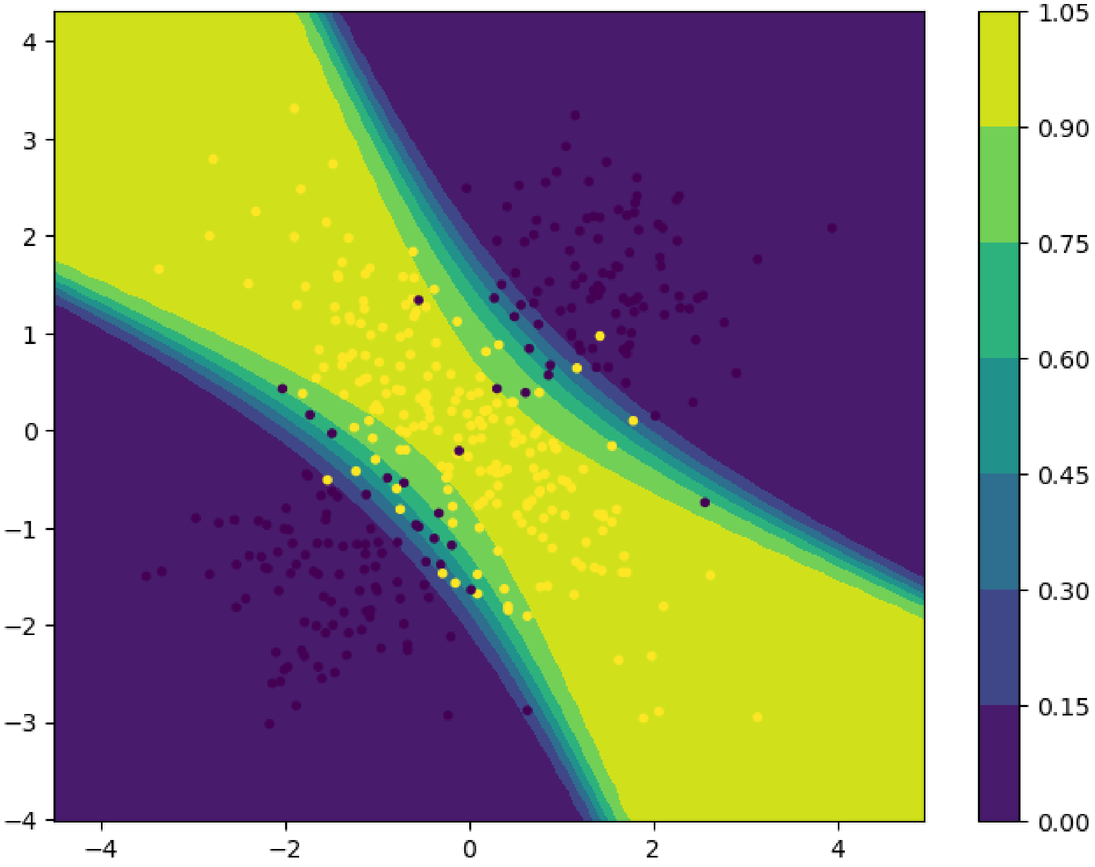
```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       200
           1       0.97      0.92      0.94       200

    accuracy                           0.94       400
   macro avg       0.94      0.94      0.94       400
weighted avg       0.94      0.94      0.94       400

              precision    recall  f1-score   support

           0       0.95      0.95      0.95       200
           1       0.95      0.94      0.95       200

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

We also learned about other tools, such as the ROC curve, AUC, and precision-recall curve. However, since the SVMs are not model-based, **we only obtain hard label assignments when doing predictions**. To overcome this, one heuristic that can be used is **Platt scaling** to convert the SVM output to probabilities. However, these probabilites may not be well calibrated and may be inconsistent with the hard labels.

In the code below, we visualize the probabilities of class assignments across a grid of possible inputs for the tuned SVM polynomial model.

[20]:
```
# Fit the polynomial model with optimal prarmeters and the option to compute␣
 ↪the probabilities
svm_poly = SVC(kernel='poly', C=cv.best_params_["C"], degree=cv.
 ↪best_params_["degree"], probability=True).fit(X_ex4, y_ex4)

# Create a grid of inputs for plotting
x1lim = [X_ex4[:,0].min(),X_ex4[:,0].max()]
x2lim = [X_ex4[:,1].min(),X_ex4[:,1].max()]

xx1 = np.linspace(x1lim[0]-1, x1lim[1]+1, 50)
xx2 = np.linspace(x2lim[0]-1, x2lim[1]+1, 50)
XX2, XX1 = np.meshgrid(xx2, xx1)
```

```
# Calculate the probabiltiy for each point in the grid
xx = np.c_[XX1.ravel(), XX2.ravel()]
xx.shape
P = svm_poly.predict_proba(xx)[:,0].reshape(XX1.shape)

plt.figure(figsize=(8,6))
plt.contourf(XX1, XX2, P)
plt.colorbar()
plt.scatter(x=X_ex4[:,0], y=X_ex4[:,1],c = 1-y_ex4, marker = '.')
plt.show()
```



Notice how the model is quite **confident in the predictions (probabilities close to 0 or 1)** in the corners of the input space, even where we don't have any data. In some cases, this may be undesirable, as we may not want to make such confident assesments in areas where we have little to no data.

3.5.1 Exercise 8 (EXTRA)

- For the SVM with RBF kernel and optimal CV parameters, repeat the plot above to visualize the probabilities of class assignments.
- How do the results compare with the polynomial kernel? Does this impact your choice of kernel?

[ ]:

```

```

4 Default Data

Let's consider the default data that we explored last week. Recall that information is collected on **10000** individuals, recording whether they defaulted on their credit card or not as well as other characteristics. Specifically, the included columns in the data are:

- `default` - whether the individual has defaulted
- `student` - whether the individual is the student
- `balance` - balance in the individual's account
- `income` - income of the individual

[29]:

```
# Load the data
df_default = pd.read_csv("Default.csv", index_col=0)

# For ease of exposition, let's' drop the student varible.
df_default = df_default.drop("student", axis=1)
df_default.head()
```

[29]:

```
  default      balance       income
1      No   729.526495  44361.62507
2      No   817.180407  12106.13470
3      No  1073.549164  31767.13895
4      No   529.250605  35704.49394
5      No   785.655883  38463.49588
```

Next, we define our feature matrix and output vector, and split the data into a train and test set. Recall that due to the class imbalance in the data, we use the option `stratify=y` to maintain the class proportion in the test set.

[30]:

```
from sklearn.model_selection import train_test_split

# Feature matrix and response vector
X, y = df_default.drop(['default'], axis=1), df_default['default']

# Convert to numpy
X = X.values

# Encode default
```

```
y = LabelEncoder().fit_transform(y)

# Stratify split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle= True,␣
 ↪stratify=y,
                                                    test_size = 0.1,␣
 ↪random_state=1112)
```

4.0.1 Exercise 9 (CORE)

a. Run the following code to fit an SVC and tune the penalty parameter `C`.

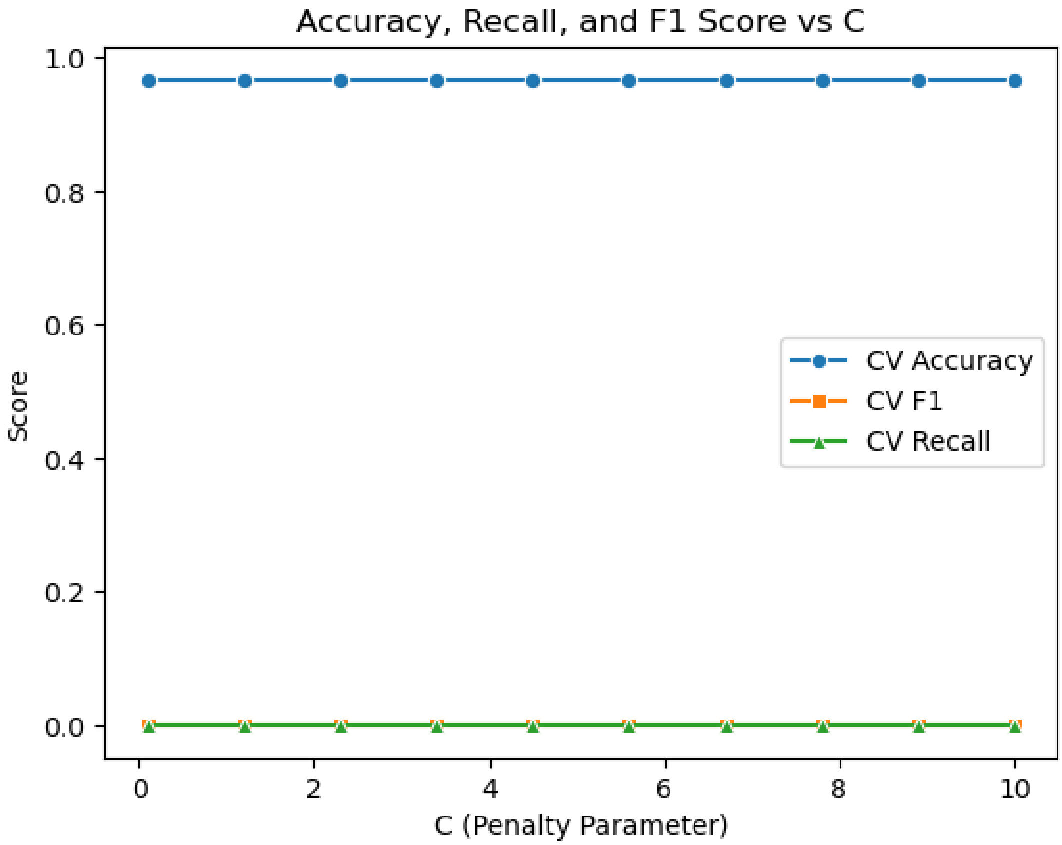
b. Plot the accuracy, recall, and f1 score as function of `C` and visualize the decision boundary. Comment on the results.

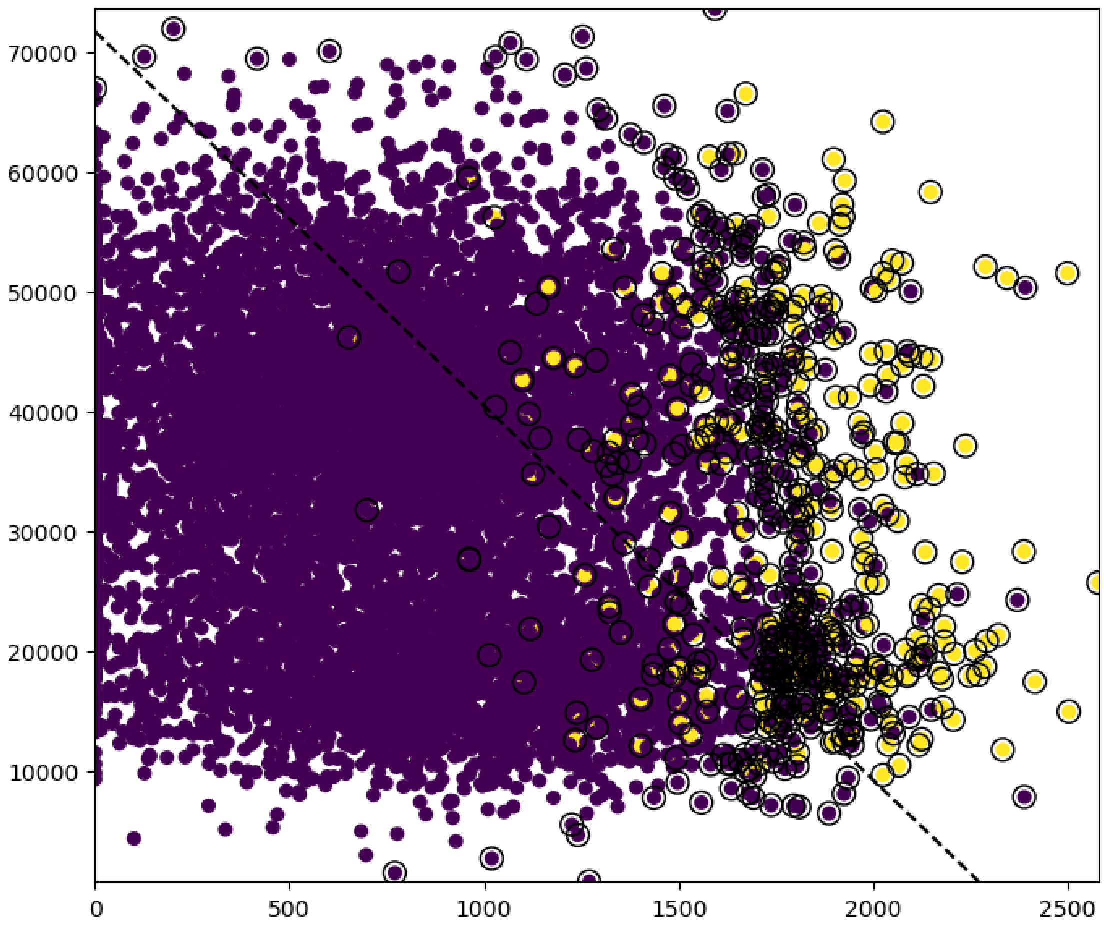
[31]:

```
# SVM with linear kernel
svm_def = make_pipeline(
    StandardScaler(),
    SVC(kernel='linear')
)

# Grid search over C
C = np.linspace(0.1, 10, 10)
cv_def = GridSearchCV(
    svm_def,
    param_grid = {'svc__C': C},
    cv = KFold(5, shuffle = True, random_state = 0),
    scoring = ["accuracy", "f1","recall"],
    refit='recall' #refit based on recall
)

# Fit and tune the model
cv_def.fit(X_train, y_train)

# Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv_def.cv_results_
                           ).
 ↪filter(['param_svc__C','mean_test_accuracy','mean_test_f1',␣
 ↪'mean_test_recall']
                                  ).rename(columns={'param_svc__C':
 ↪'C','mean_test_accuracy':'CV accuracy', 'mean_test_f1':'CV f1',␣
 ↪'mean_test_recall':'CV recall'})
```

[33]:

```
# Plotting stuff
#plt.figure(figsize=(10, 6))
sns.lineplot(x="C", y="CV accuracy", data=cv_accuracy, label="CV Accuracy",␣
 ↪marker="o")
sns.lineplot(x="C", y="CV f1", data=cv_accuracy, label="CV F1", marker="s")
sns.lineplot(x="C", y="CV recall", data=cv_accuracy, label="CV Recall", marker="^")
plt.xlabel("C (Penalty Parameter)")
plt.ylabel("Score")
plt.title("Accuracy, Recall, and F1 Score vs C")
plt.legend()
plt.show()

# Training on the best estimator
best_svm_def = cv_def.best_estimator_

# plotting boundaries
plot_margin(best_svm_def, X_train, y_train)

# Output best parameters and corresponding recall score
cv_def.best_params_, cv_def.best_score_
```





[33]:
```
({'svc__C': 0.1}, 0.0)
```

- Accuracy is high (~99%) but recall is very low also our best score is 0, this shows that there is a class imbalance.
- The decision boundary confirms that the model struggles to classify the default cases.
- To improve recall, we should try class weighting, an RBF kernel, or polynomial models.

4.0.2 Exercise 10 (CORE)

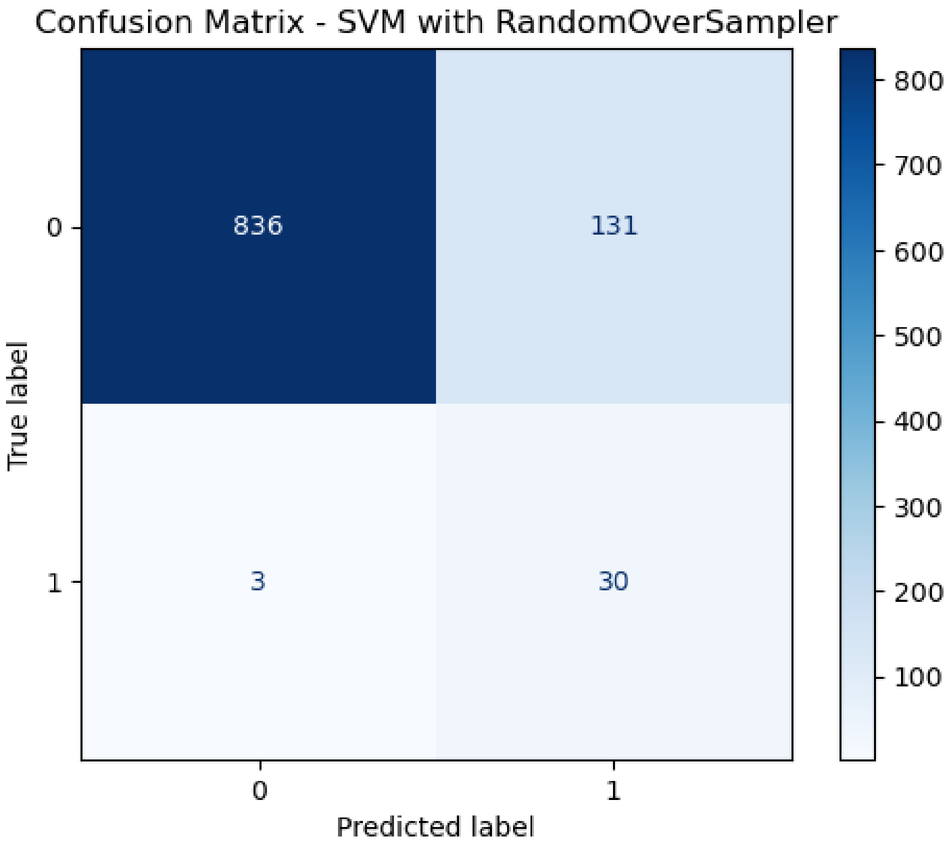
a. To address the imbalance issue, alter the pipeline either using the `RandomOverSampler` or `RandomUnderSampler`.

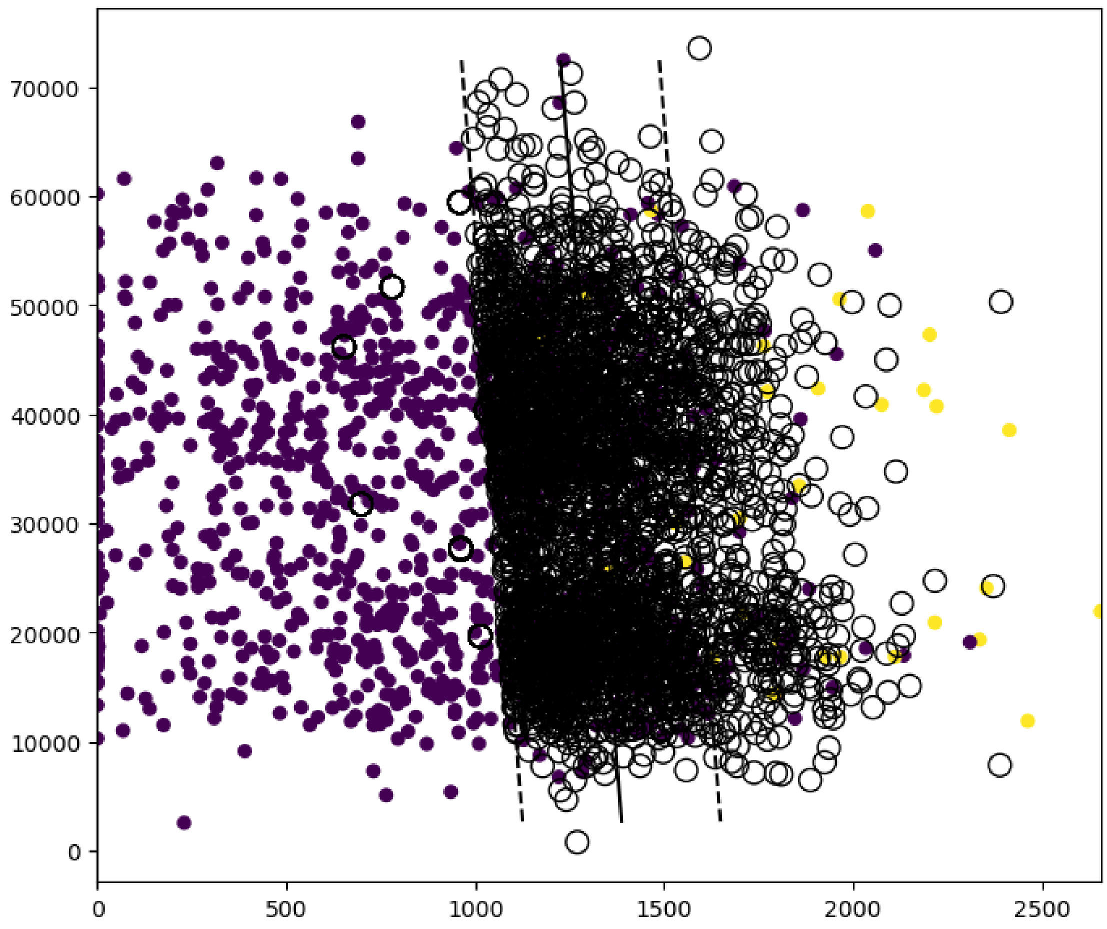
b. Choose a value of `C` and plot the decision boundary and confusion matrix on the test data. How have the results changed?

[34]:
```
# Install the imblearn if necessary
!pip install imblearn

from imblearn.pipeline import make_pipeline as Im_make_pipeline
```

```python
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

# Apply RandomOverSampler to balance the dataset
ros = RandomOverSampler(random_state=1112)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

# SVM pipeline for randomsampler
svm_def_balanced = make_pipeline(
    StandardScaler(),
    SVC(kernel='linear', C=1)  # Choosing C=1 for visualization
)

# Training
svm_def_balanced.fit(X_resampled, y_resampled)

# Predicting
y_pred_test = svm_def_balanced.predict(X_test)

# getting that confusion matrix
cm = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - SVM with RandomOverSampler")
plt.show()

# Plotting decision boundary
plot_margin(svm_def_balanced, X_test, y_test)

# Output for CM
cm
```

```
Requirement already satisfied: imblearn in /opt/conda/lib/python3.9/site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in
/opt/conda/lib/python3.9/site-packages (from imblearn) (0.12.4)
Requirement already satisfied: scikit-learn>=1.0.2 in
/opt/conda/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.3.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (3.2.0)
Requirement already satisfied: joblib>=1.1.1 in /opt/conda/lib/python3.9/site-
packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.9/site-
packages (from imbalanced-learn->imblearn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /opt/conda/lib/python3.9/site-
packages (from imbalanced-learn->imblearn) (1.11.2)
```


Confusion Matrix - SVM with RandomOverSampler
True label
0
1
836
131
3
30
0
1
Predicted label
800
700
600
500
400
300
200
100



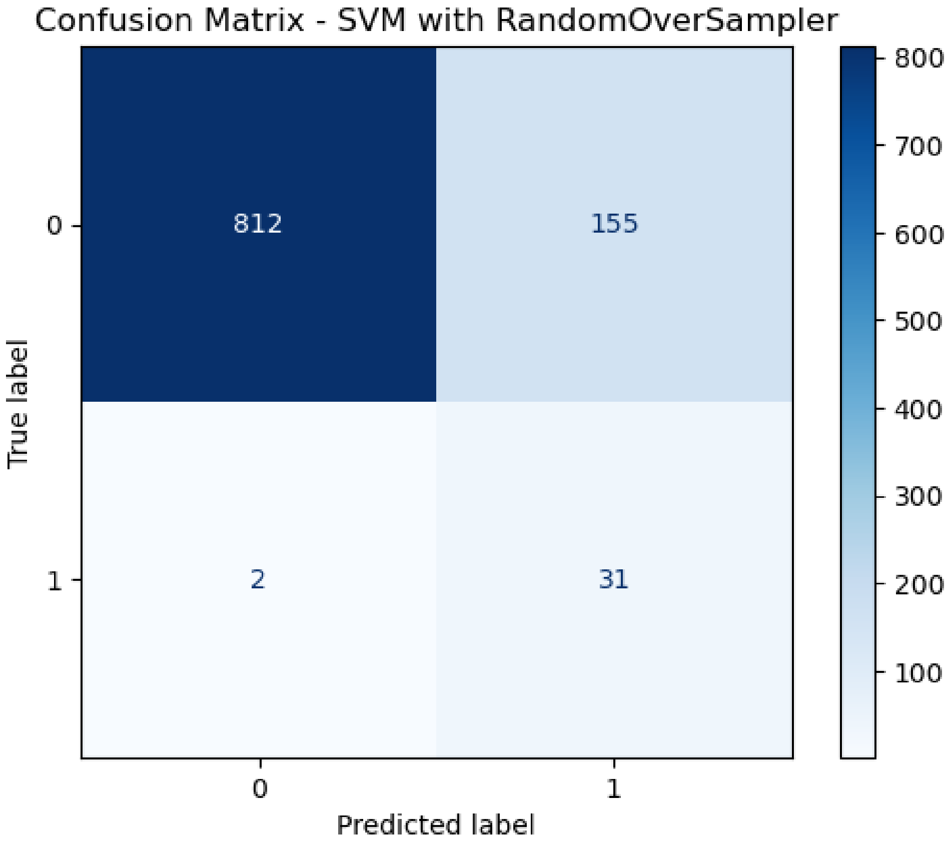
```
[34]: array([[836, 131],
             [  3,  30]])
```

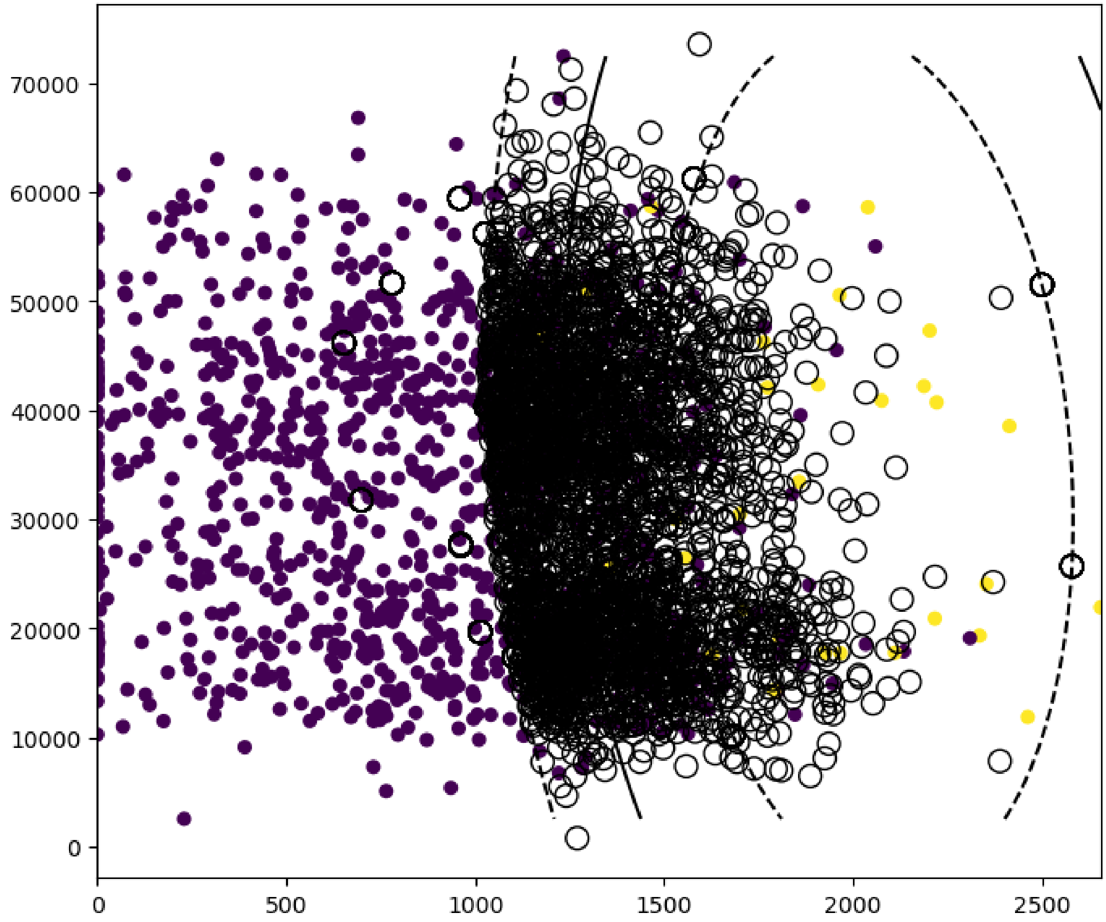
- For confusion Matrix
  - True Negatives (836): Correctly classified non-default cases.
  - False Positives (131): Non-default cases misclassified as defaults.
  - False Negatives (3): Default cases misclassified as non-default.
  - True Positives (30): Correctly classified defaults.
- Previously, recall was close to 0 due to class imbalance. Now, more defaults are correctly classified. issue is that we will have More False Positives
- This model is now more likely to predict defaults, which can increase false alarms.
- Decision Boundary
  - The decision boundary has shifted, and more support vectors are present.
  - Oversampling has allowed the SVM to better separate the classes.
  - However, some misclassifications still occur, particularly on the right side.

4.0.3 Exercise 11 (CORE)

Lastly, let's explore using a nonlinear decision boundary. Alter your pipeline from the previous exercise to use a polynomial kernel of degree 2 with a coefficient of 1. Visualize the decision boundary and confusion matrix on the test data. How do the results compare?

[39]:
```
# Define and Train SVM with Polynomial Kernel
svm_defP = make_pipeline(
    StandardScaler(),
    SVC(kernel='poly', degree=2, coef0=1, C=1)
)

# Training
svm_defP.fit(X_resampled, y_resampled)

# Predicting on the test set
y_pred_test = svm_defP.predict(X_test)

# Computing Confusion Matrix
cm = confusion_matrix(y_test, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - SVM with RandomOverSampler")
plt.show()

# Plotting decision boundary (only if dataset has 2 features)
plot_margin(svm_defP, X_test, y_test)

# Output Confusion Matrix
cm
```


Confusion Matrix - SVM with RandomOverSampler
True label
Predicted label
0
1
0
1
812
155
2
31
800
700
600
500
400
300
200
100



```
[39]: array([[812, 155],
             [  2,  31]])
```

- The polynomial kernel slightly improved recall (from 90.9% to 93.9%).
- False positives increased in the polynomial kernel (155 vs. 131), meaning the model predicts more defaults than necessary.
- False negatives decreased (3 → 2), meaning it correctly identifies more default cases.
- The polynomial kernel creates a more complex, curvy boundary.
- More support vectors are involved, which allows better adaptation to non-linear patterns.
- Compared to the linear model, it captures more complex relationships but at the cost of generalization.
- The polynomial model adapts to the data better, capturing complex structures in the distribution.
- However, more support vectors mean increased model complexity, which could lead to overfitting.

5 Competing the Worksheet

At this point you have hopefully been able to complete all the CORE exercises and attempted the EXTRA ones. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Before generating the PDF, please go to Edit -> Edit Notebook Metadata and change 'Student 1' and 'Student 2' in the **name** attribute to include your name. If you are unable to edit the Notebook Metadata, please add a Markdown cell at the top of the notebook with your name(s).

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF. Once generated, please submit this PDF on Learn page by 16:00 PM on the Friday of the week the workshop was given.

```
[ ]: !jupyter nbconvert --to pdf mlp_week07.ipynb
```