MLPy Workshop 7

Student 1, Student 2

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 decision 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

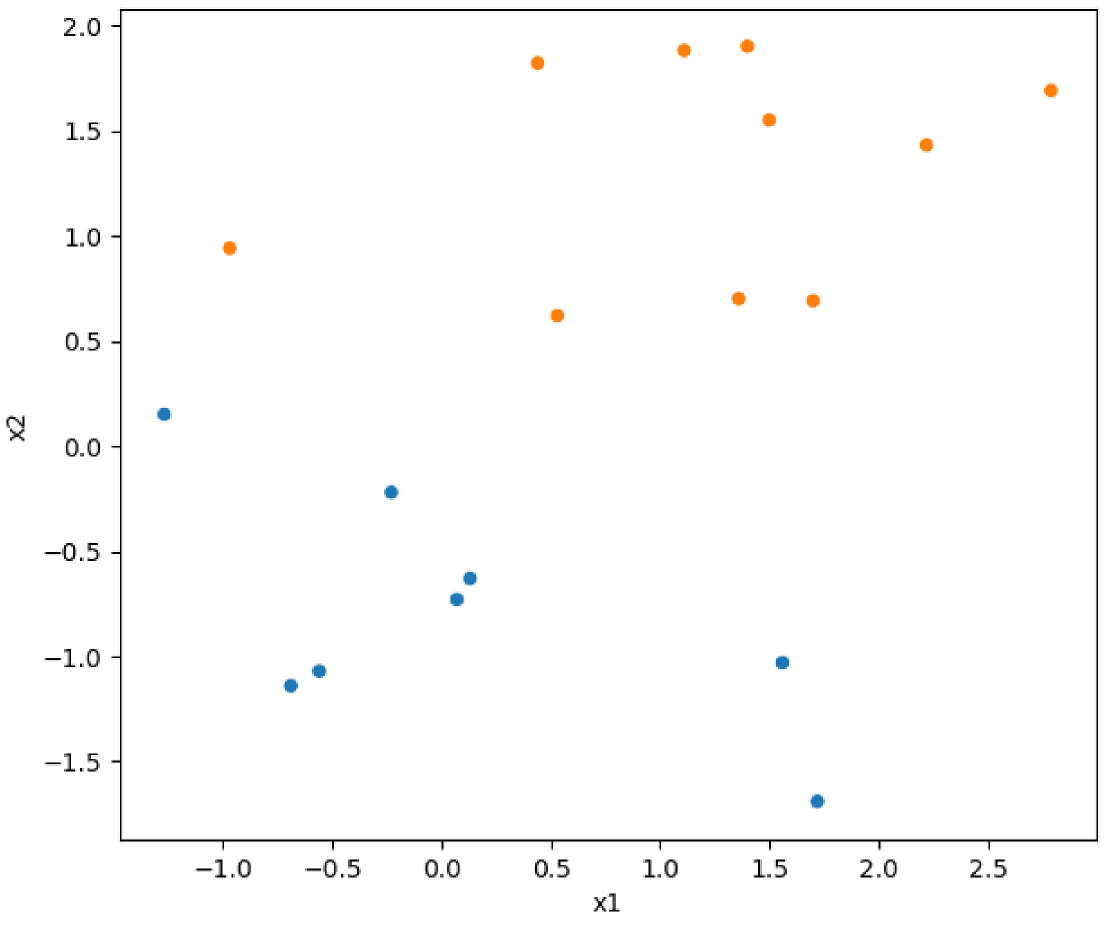
We will begin by examining various toy data sets to explore the basics of these models. For the first example, we will read in data from `ex1.csv`.

[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)
```

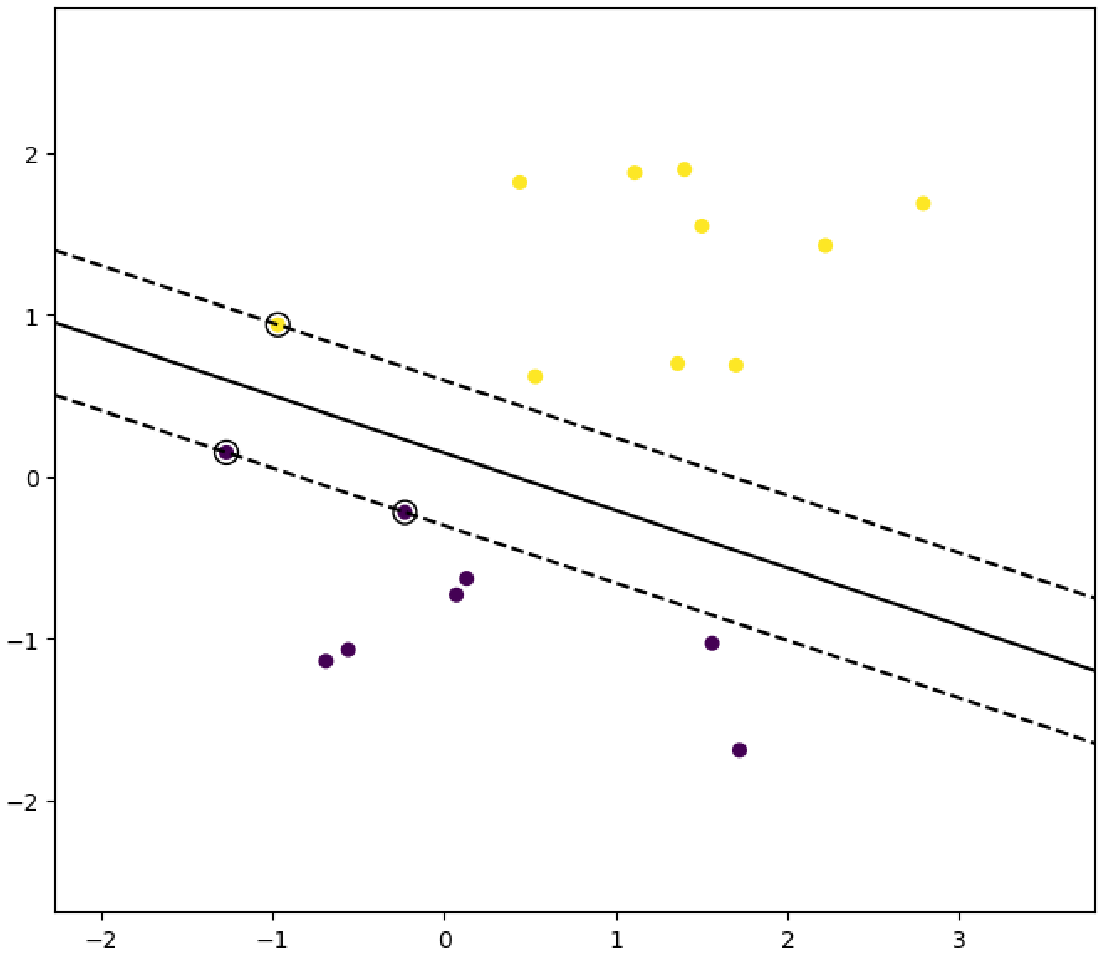
```
(18, 2)
(18,)
```

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]:
```
C = 100
svm = SVC(kernel='linear', C=C)
svm.fit(X_ex1, y_ex1)

plot_margin(svm, X_ex1,y_ex1)
```



There are 2 support vectors for the purple class and they are on the correct side of the margin, there is only 1 support vector for the yellow class and it is on the correct side of the margin.

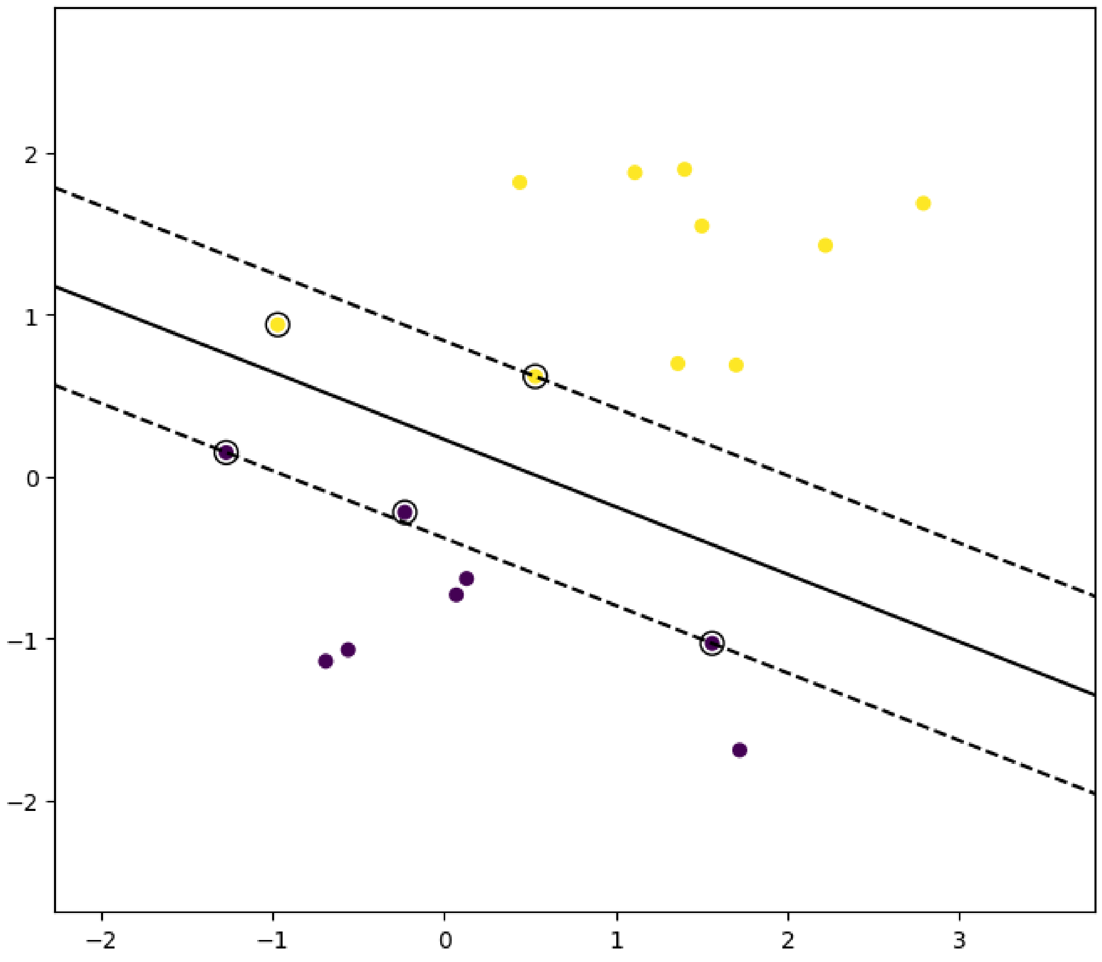
3.1.2 Exercise 2 (CORE)

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]: C = 1
     svm = SVC(kernel='linear', C=C)
     svm.fit(X_ex1, y_ex1)

     plot_margin(svm, X_ex1,y_ex1)
```



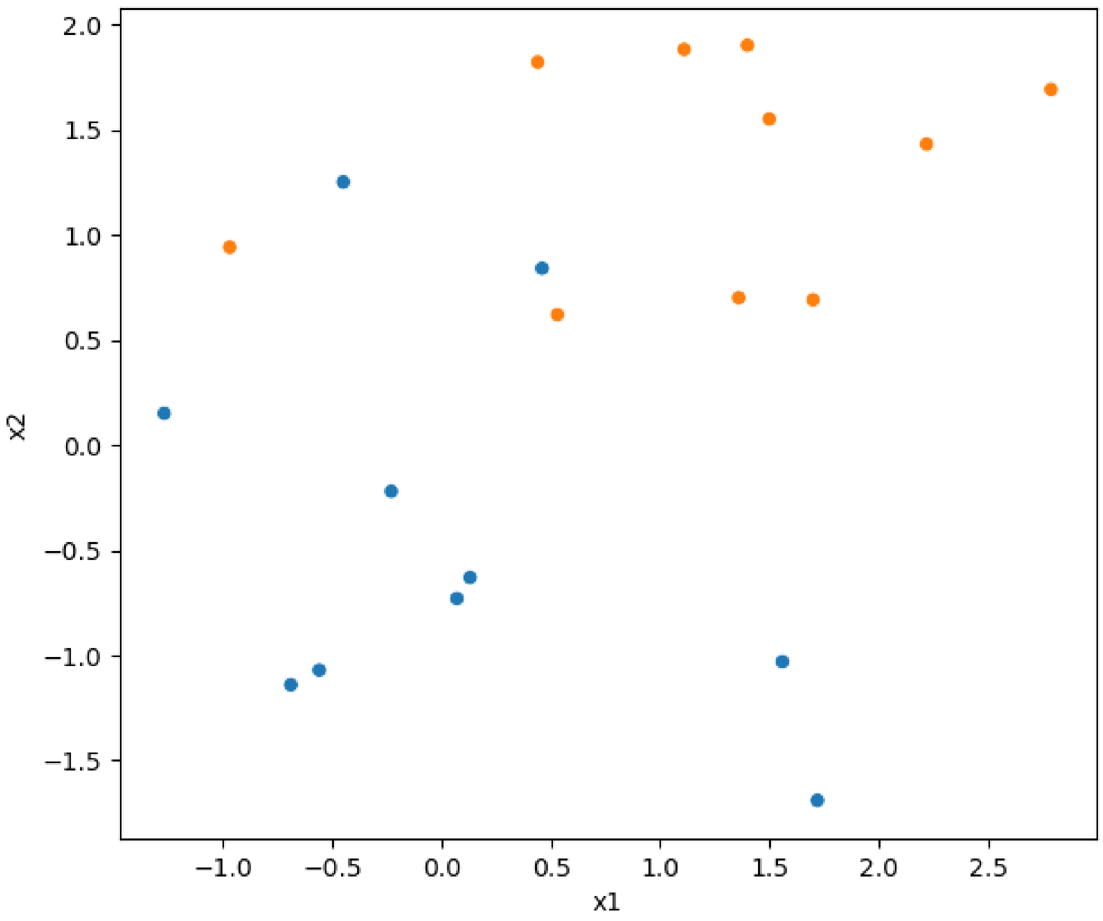
We have relaxed our margin and now the purple class has 3 support vectors, 2 of which are on the margin and 1 of which is beyond the margin but before the decision boundary.

The yellow class now has 2 support vectors with 1 on the margin and 1 beyond the margin but before the decision boundary.

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.

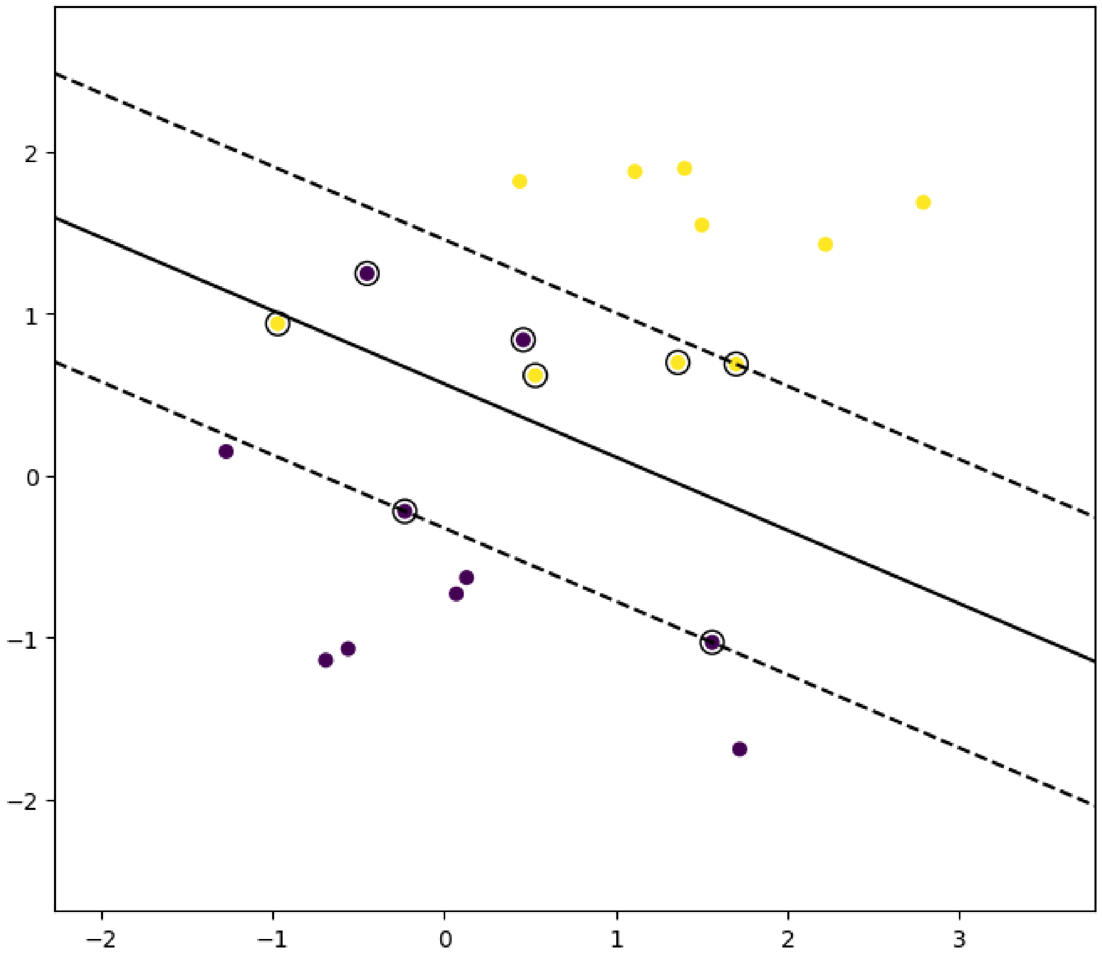
[8]:
```
# Read in the data
ex2 = pd.read_csv("ex2.csv")

# 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)
```

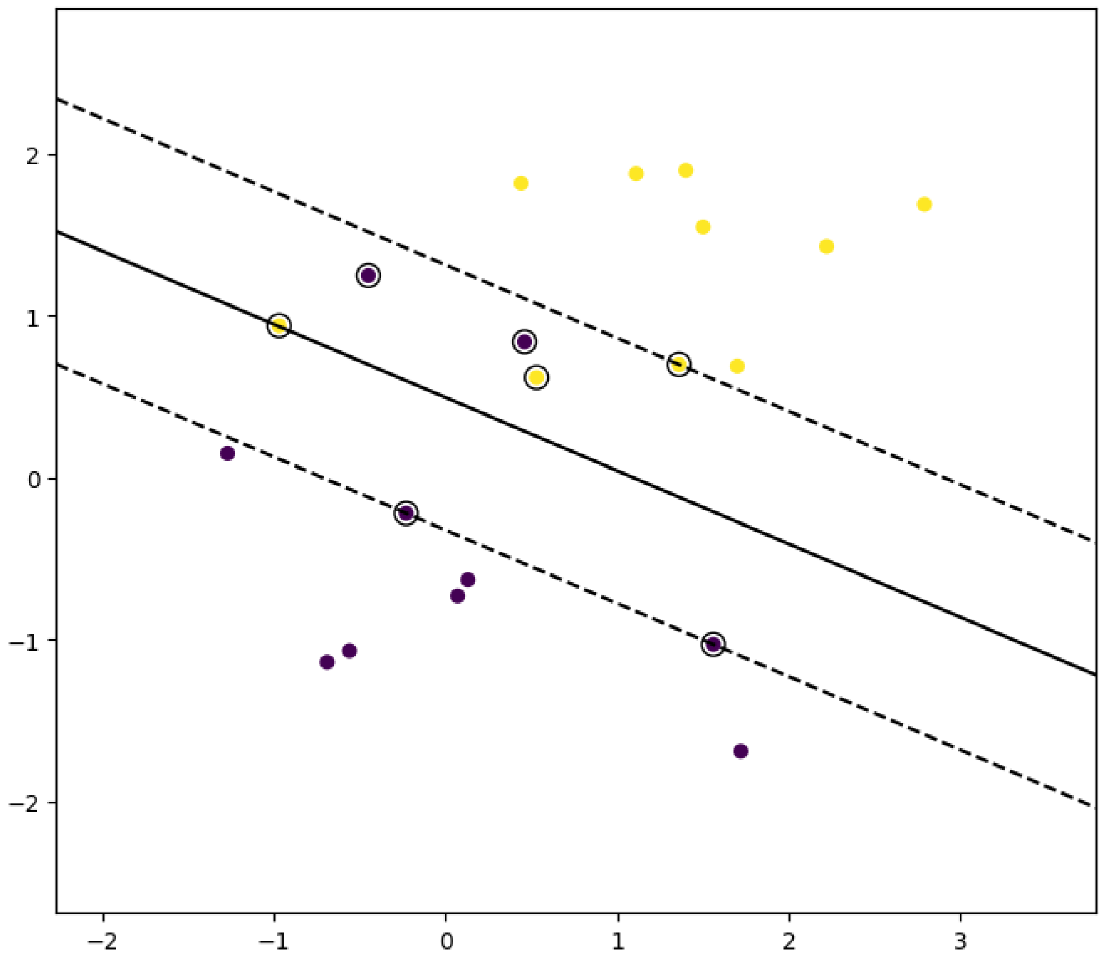


3.2.1 Exercise 3 (CORE)

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.

[10]:
```
svm_one = SVC(kernel='linear', C=1).fit(X_ex2, y_ex2)
svm_hund = SVC(kernel='linear', C=100).fit(X_ex2, y_ex2)

plot_margin(svm_one, X_ex2, y_ex2)
plot_margin(svm_hund, X_ex2, y_ex2)
```





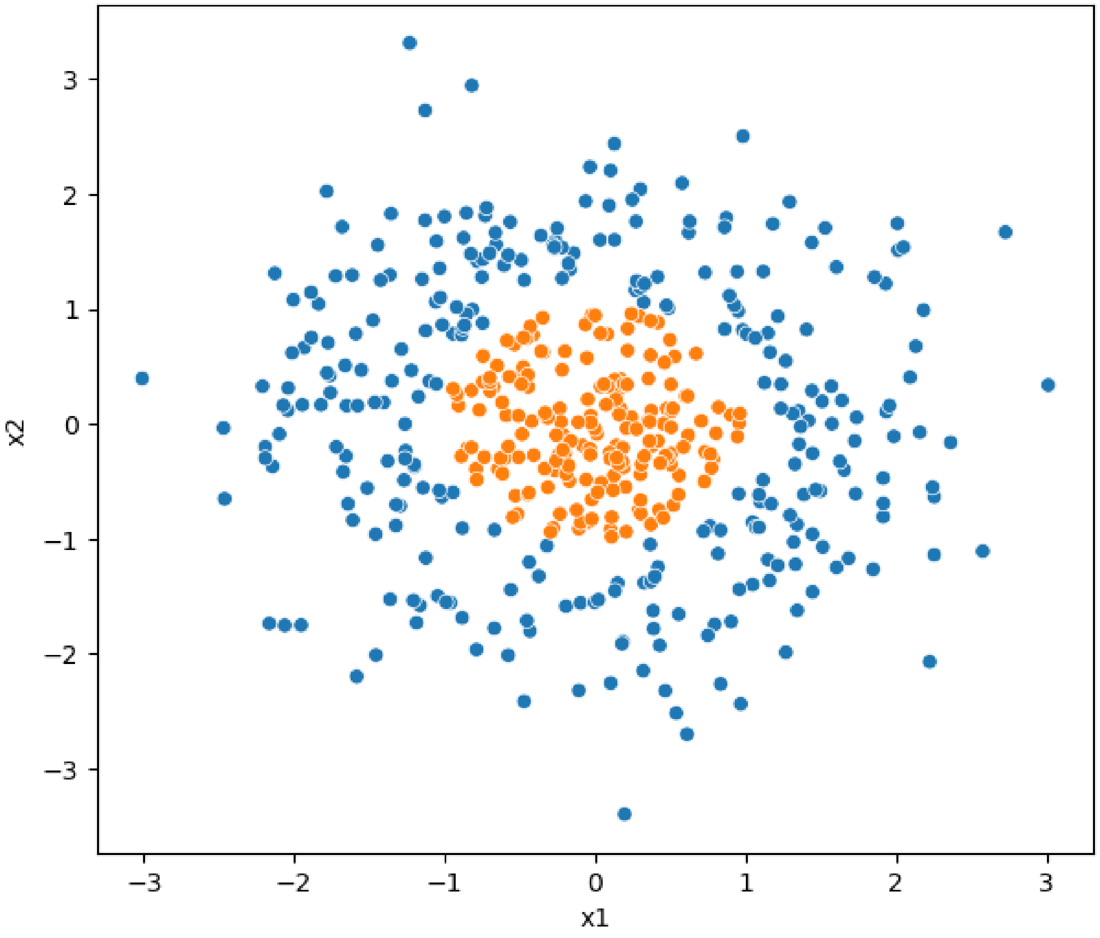
Purple: 4 support vectors, 2 on margin, 2 beyond decision boundary. Yellow: 3 support vectors, 1 on margin, 1 beyon margin but before descision boundary, 1 on decidion boundary.

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.

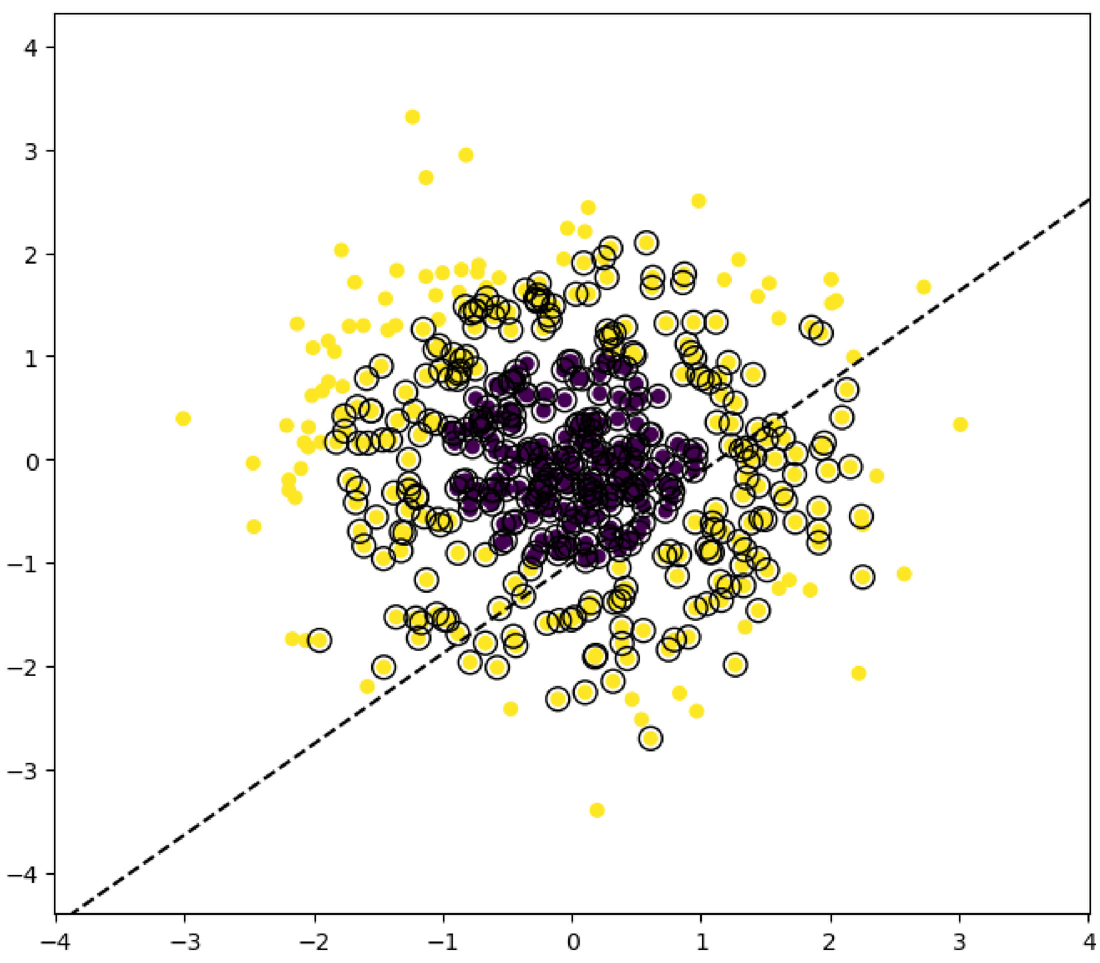
[11]:
```
# Load the third data set
ex3 = pd.read_csv("ex3.csv")

# 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.

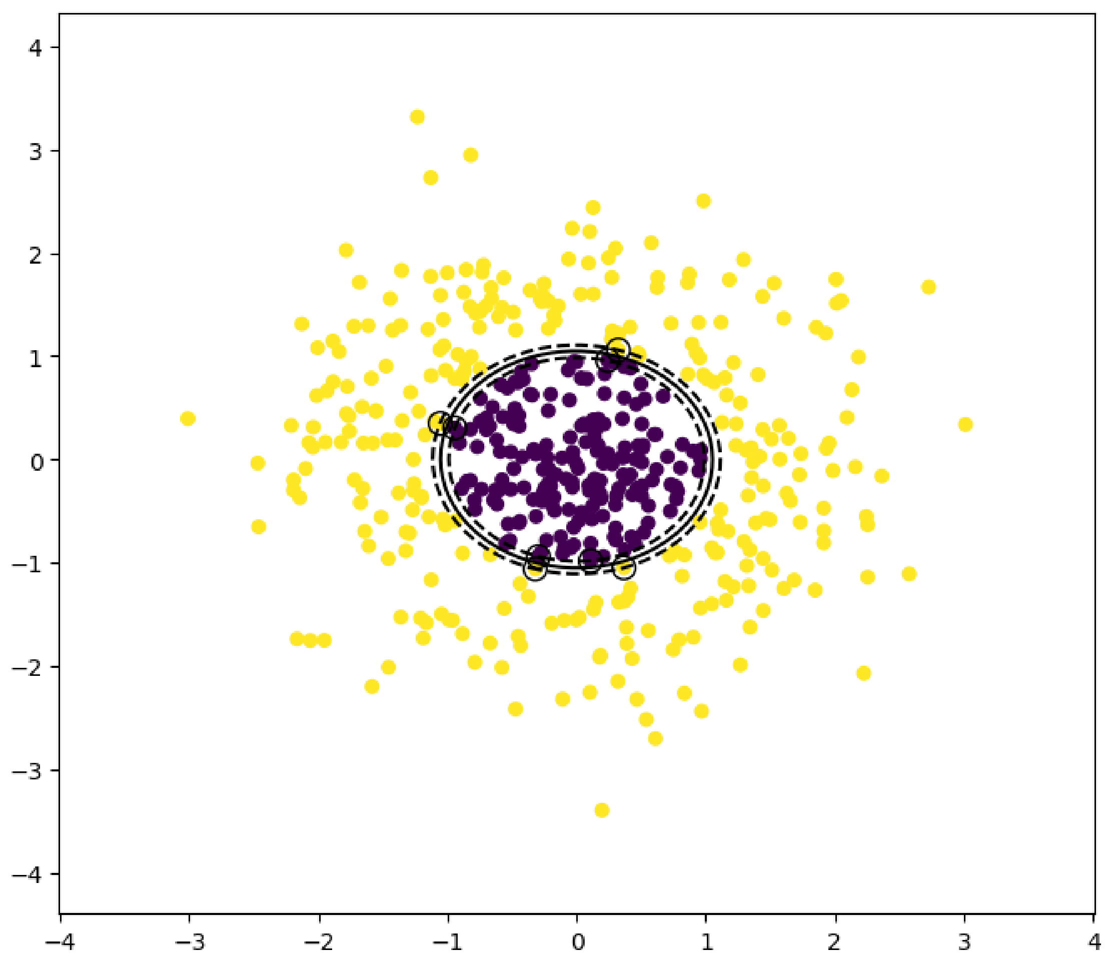
[12]:
```
# Fit the SVC with a linear kernel on the new data
svm_lin = SVC(kernel='linear', C=100).fit(X_ex3, y_ex3)

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



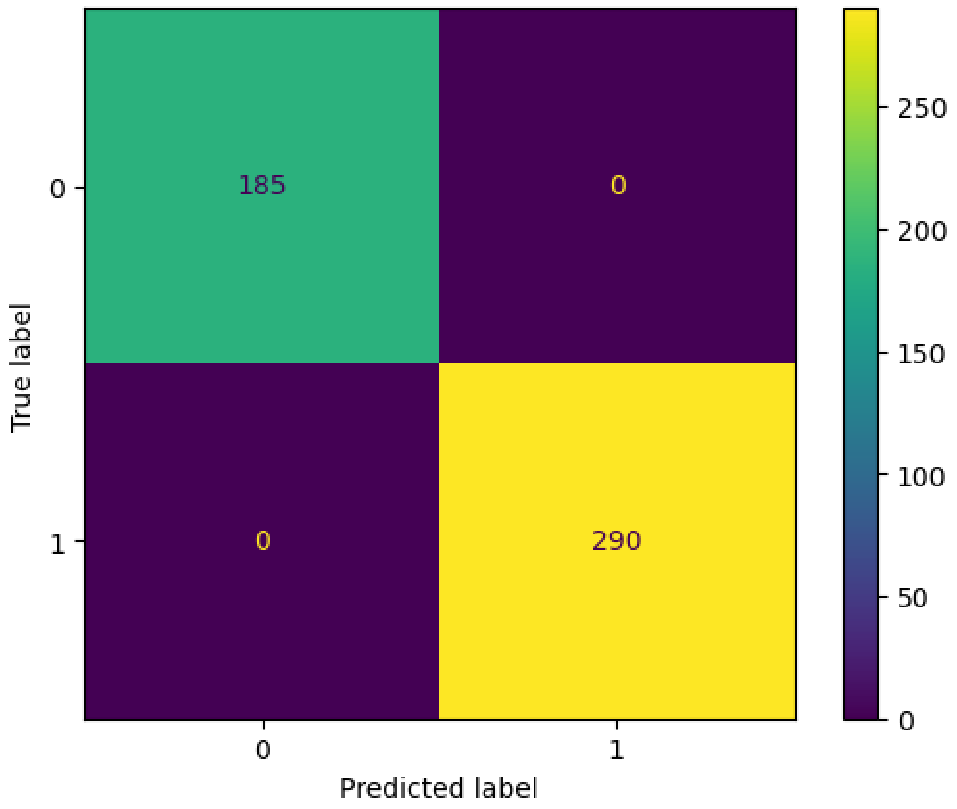
3.3.1 Exercise 4 (CORE)

- 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?

[15]:
```
svm_poly = SVC(kernel='poly', C=100, degree=2).fit(X_ex3, y_ex3)
plot_margin(svm_poly, X_ex3, y_ex3)
ConfusionMatrixDisplay.from_estimator(svm_poly, X_ex3, y_ex3)
```



```
[15]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x178159070>
```

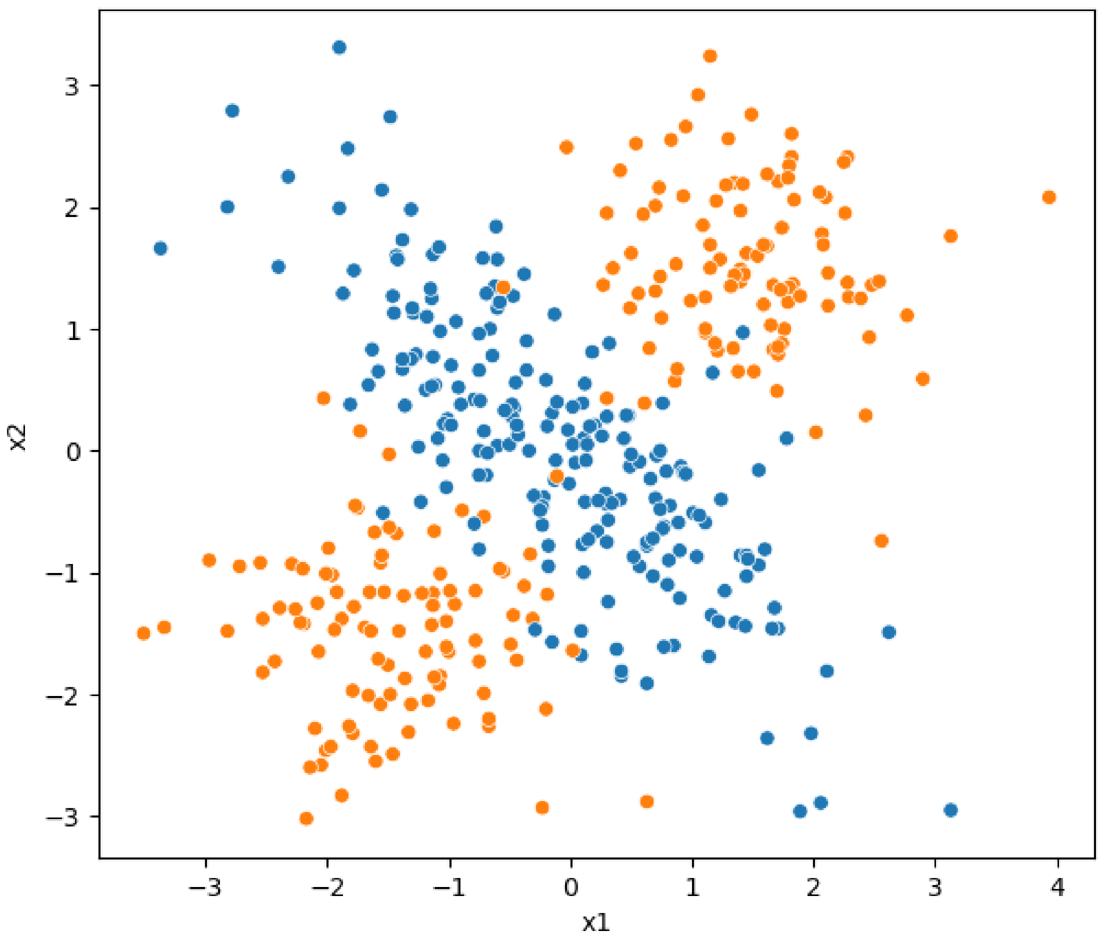


The degree two polynomial creates a circular decision boundary which is able to perfectly seperate the two classes leading to no misclassifications.

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.

[16]:
```python
# 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.

[17]:
```
# 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
```

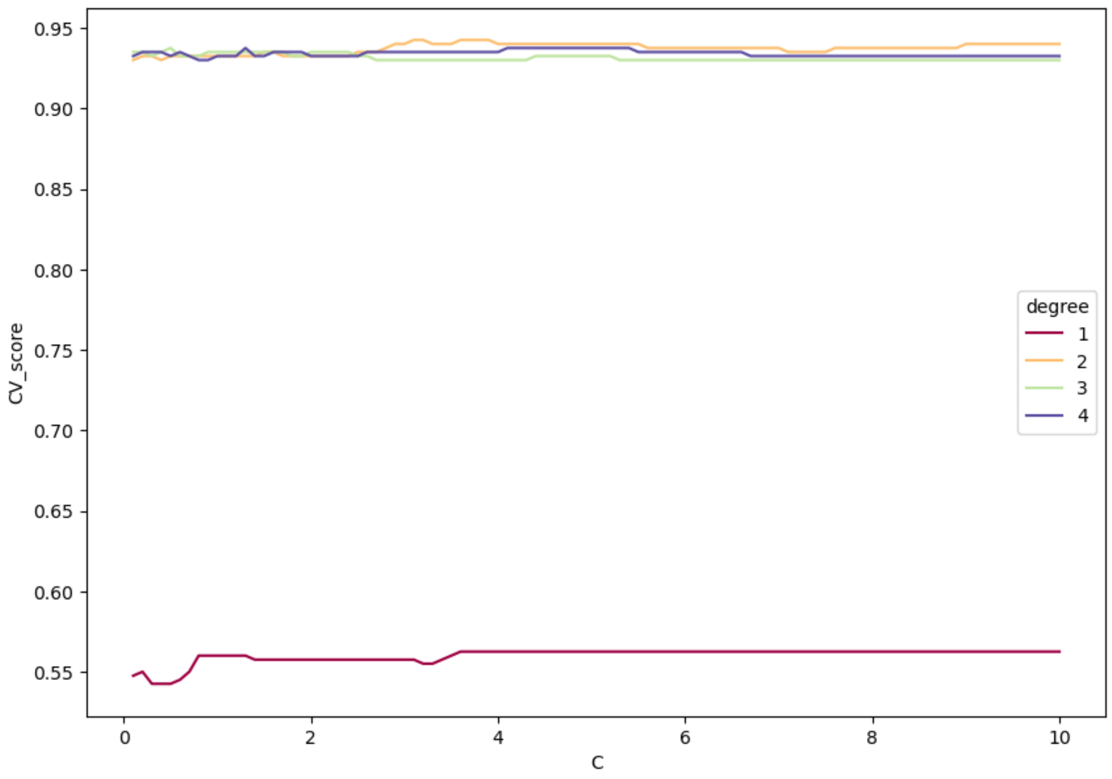
```
cv.fit(X_ex4, y_ex4)

# 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.

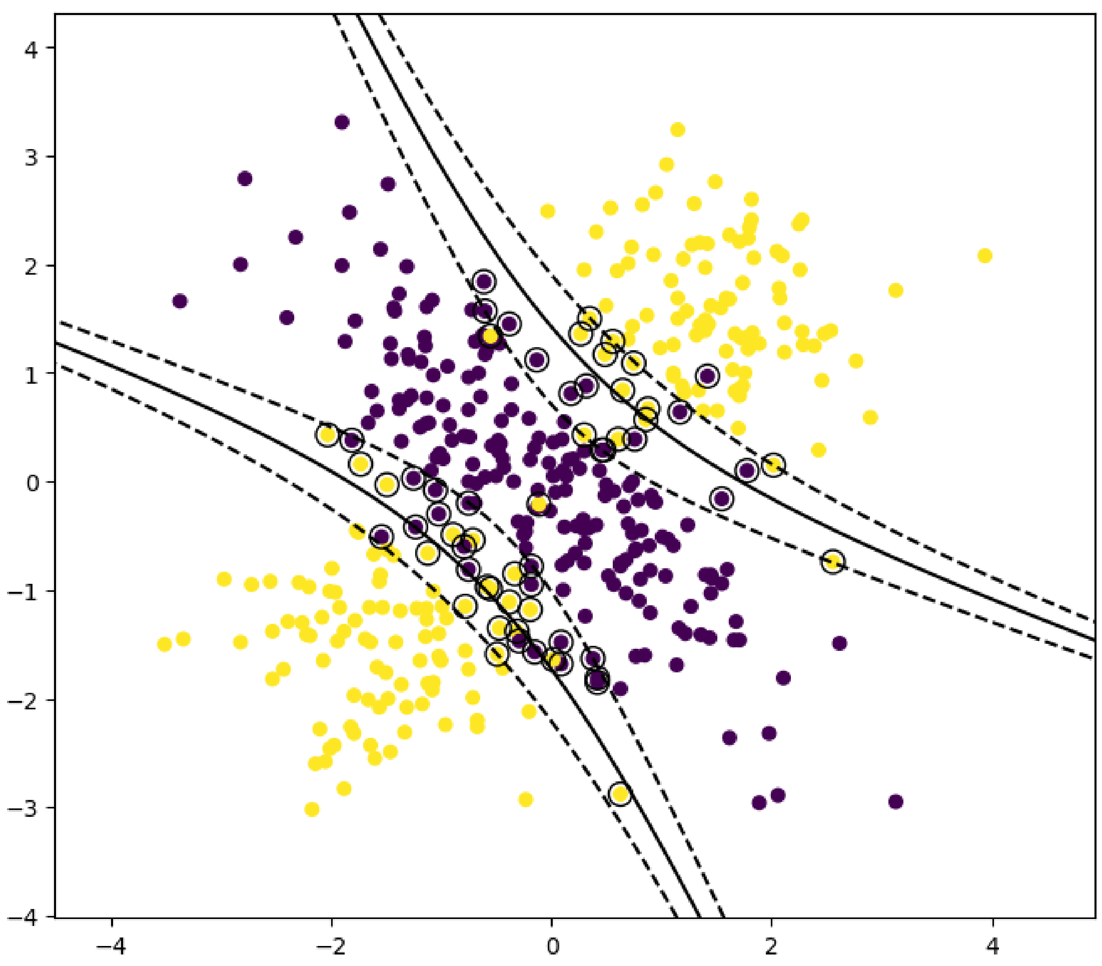
```
[18]: # Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv.cv_results_
                           ).filter(['param_C',␣
 ↪'param_degree','mean_test_score']
                                    ).rename(columns={'param_C':'C',␣
 ↪'param_degree':'degree','mean_test_score':'CV_score'})

# 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()
```



It is clear that we must pick a degree > 1 since this dramatically increases classification accuracy. Since the curves of degree 2, 3, 4 are quite similar we prefer to pick the simplest model which is degree 2. Additionally, the C value does not drastically vary the score on the validation set so we pick the recommnded 3.1.

The CV value recommendations are reasonable.

[19]:
```
plot_margin(cv.best_estimator_, X_ex4, y_ex4)
```



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?

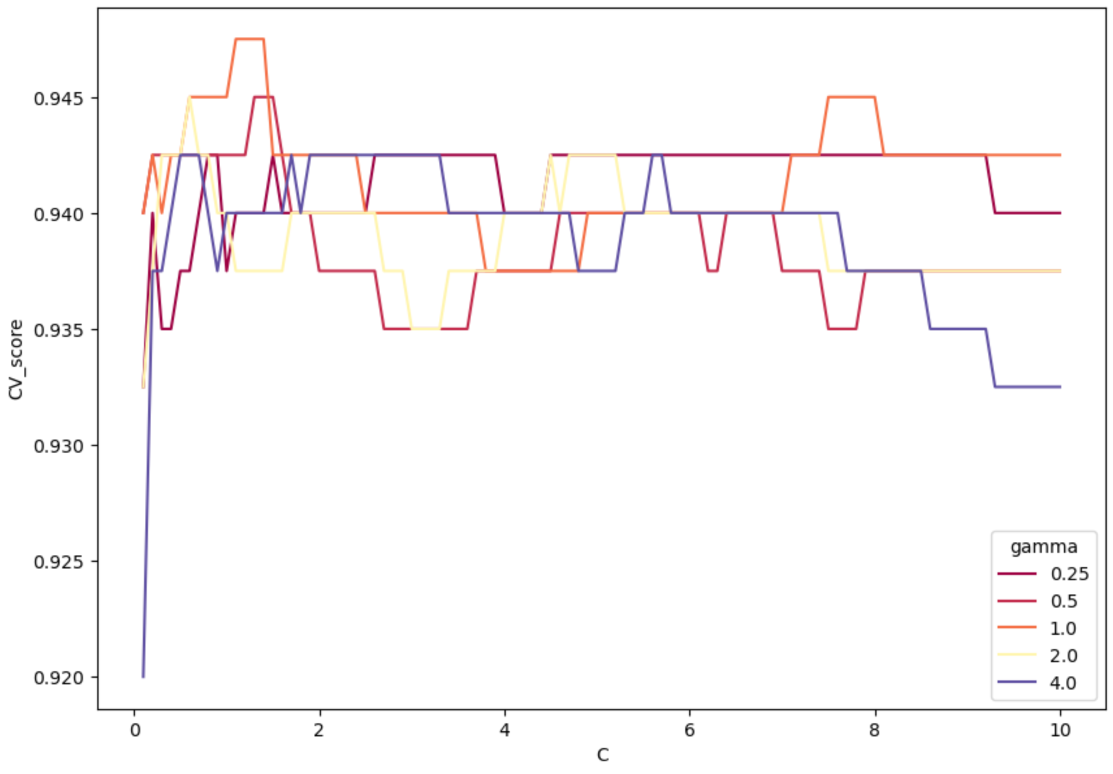
```
[ ]:
```

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?

[21]:
```
# Values of C and the inverse bandwidth
gamma = [1/4,1/2,1,2,4]
C = np.linspace(0.1, 10, 100)

param_grid = {
    "C": C,
    "gamma": gamma
}

svm = SVC(kernel='rbf')

cv2 = GridSearchCV(
    svm,
    param_grid=param_grid,
    cv = KFold(5, shuffle = True, random_state = 0)
)

cv2.fit(X_ex4, y_ex4)

cv_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':'CV_score'})

plt.figure(figsize=(10,7))
sns.lineplot(x='C', y='CV_score', data = cv_accuracy, hue ='gamma',␣
 ↪palette="Spectral")
plt.show()
```



```
[22]: print("Params: ", cv2.best_params_)
```

```
Params:  {'C': 1.1, 'gamma': 1}
```

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.

```
[23]: 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)))
```

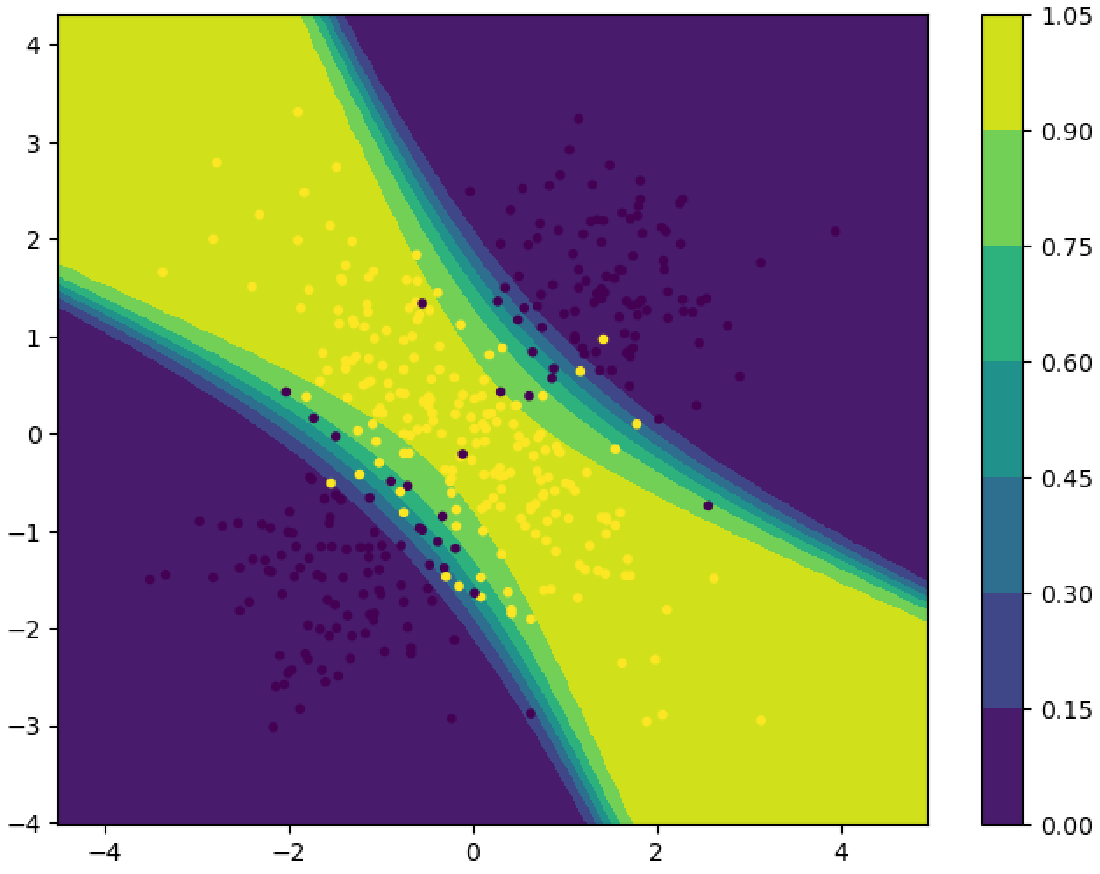
```
              precision    recall  f1-score   support

           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.

[24]:
```python
# 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()
```



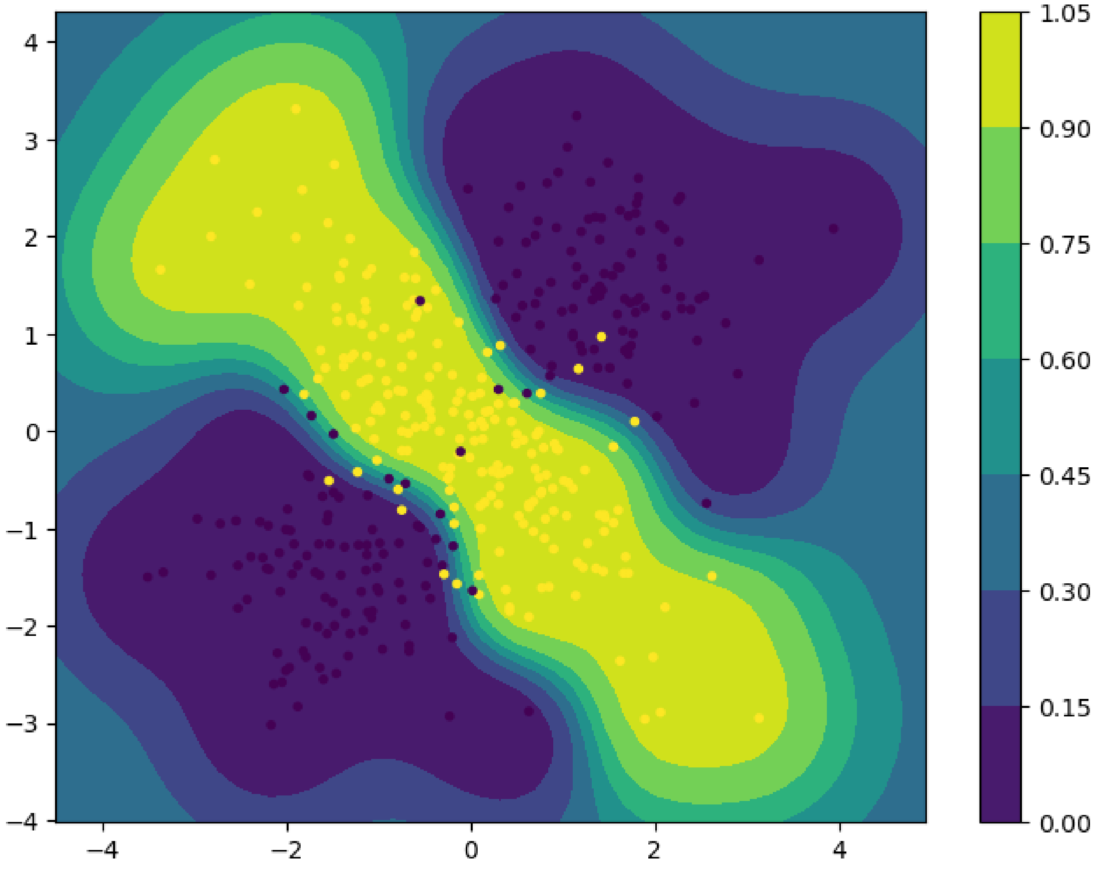
Notice how the model is quite **confident in the predictions (probabilities close to 0 or 1)** in the corners of the input space, even where we don't have any data. In some cases, this may be undesirable, as we may not want to make such confident assesments in areas where we have little to no data.

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?

[26]:
```
svm_rbf_prob = SVC(kernel='rbf', C=cv2.best_params_['C'], gamma=cv2.
 ↪best_params_['gamma'], 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_rbf_prob.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()
```



Observe that in the corners where we had no data, the probabilities are closer to 0.5 than 0.0. This could be more desireable in cases where we have no data.

The borders are wiggly but this can be changed.

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

[27]:
```
# 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()
```

[27]:
```
  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.

[28]:
```
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)

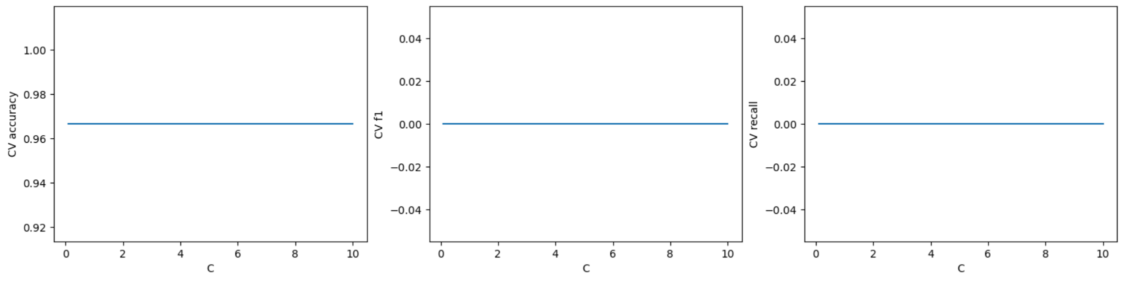
a. Run the following code to fit an SVC and tune the penalty parameter `C`.

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

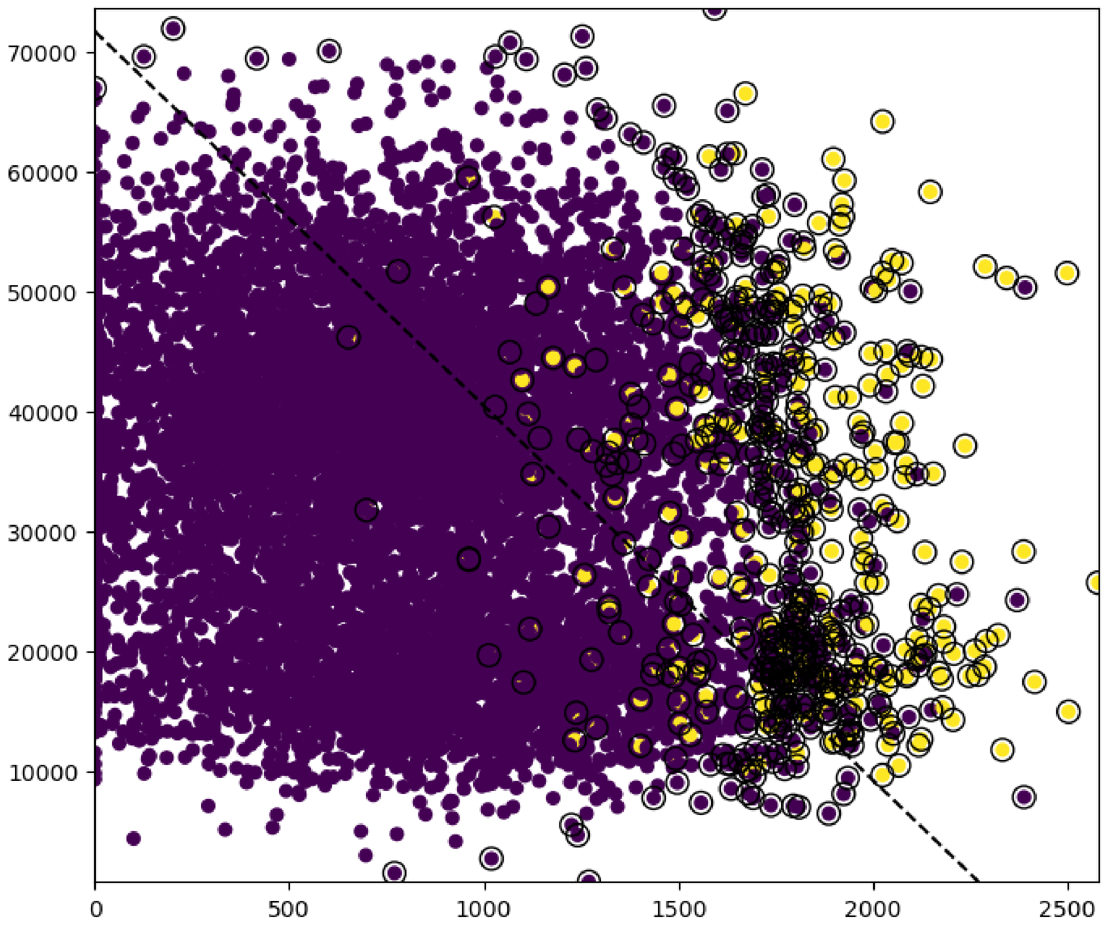
[29]:
```
# SVM with linear kernel
svm_lin = make_pipeline(
    StandardScaler(),
    SVC(kernel='linear')
)

# Grid search over C
C = np.linspace(0.1, 10, 10)
cv_lin = GridSearchCV(
    svm_lin,
    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_lin.fit(X_train, y_train)

# Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv_lin.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'})
```

[30]:
```
print("Params: ", cv_lin.best_params_)
print("Avg Recall: ", cv_lin.best_score_)



fig, ax = plt.subplots(1,3,figsize=(18,4))
sns.lineplot(x='C', y='CV accuracy', data = cv_accuracy,ax=ax[0])
sns.lineplot(x='C', y='CV f1', data = cv_accuracy,ax=ax[1])
sns.lineplot(x='C', y='CV recall', data = cv_accuracy,ax=ax[2])
plt.show()
```

```
Params:  {'svc__C': 0.1}
Avg Recall:  0.0
```



[32]:
```
plot_margin(cv_lin.best_estimator_, X_train,y_train)
```



The performance is poor, since there are so many `non-defaults` the model overfits to the majority class. Predicted that everyone is a `non-default` results in a high accuracy but 0 f1 and recall.

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?

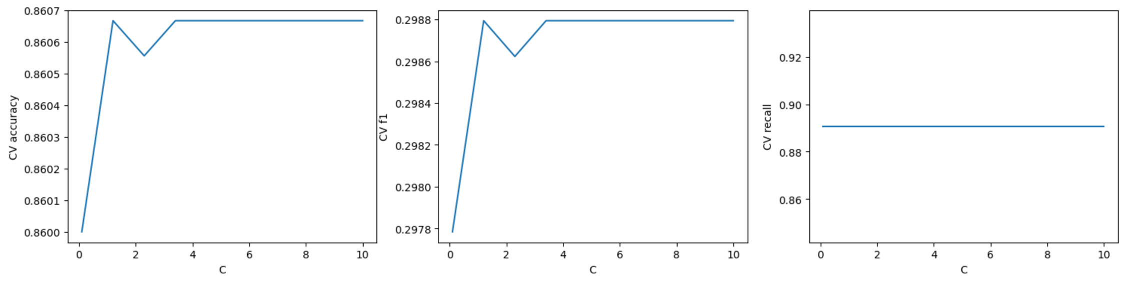
[33]:
```
# Install the imblearn if necessary
# !pip install imblearn

from imblearn.pipeline import make_pipeline as Im_make_pipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

[34]:
```
svm_lin = Im_make_pipeline(
    StandardScaler(),
    RandomOverSampler(random_state = 0),
    SVC(kernel='linear')
)

# Grid search over C
C = np.linspace(0.1, 10, 10)
cv_lin = GridSearchCV(
    svm_lin,
    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_lin.fit(X_train, y_train)

# Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv_lin.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'})
```

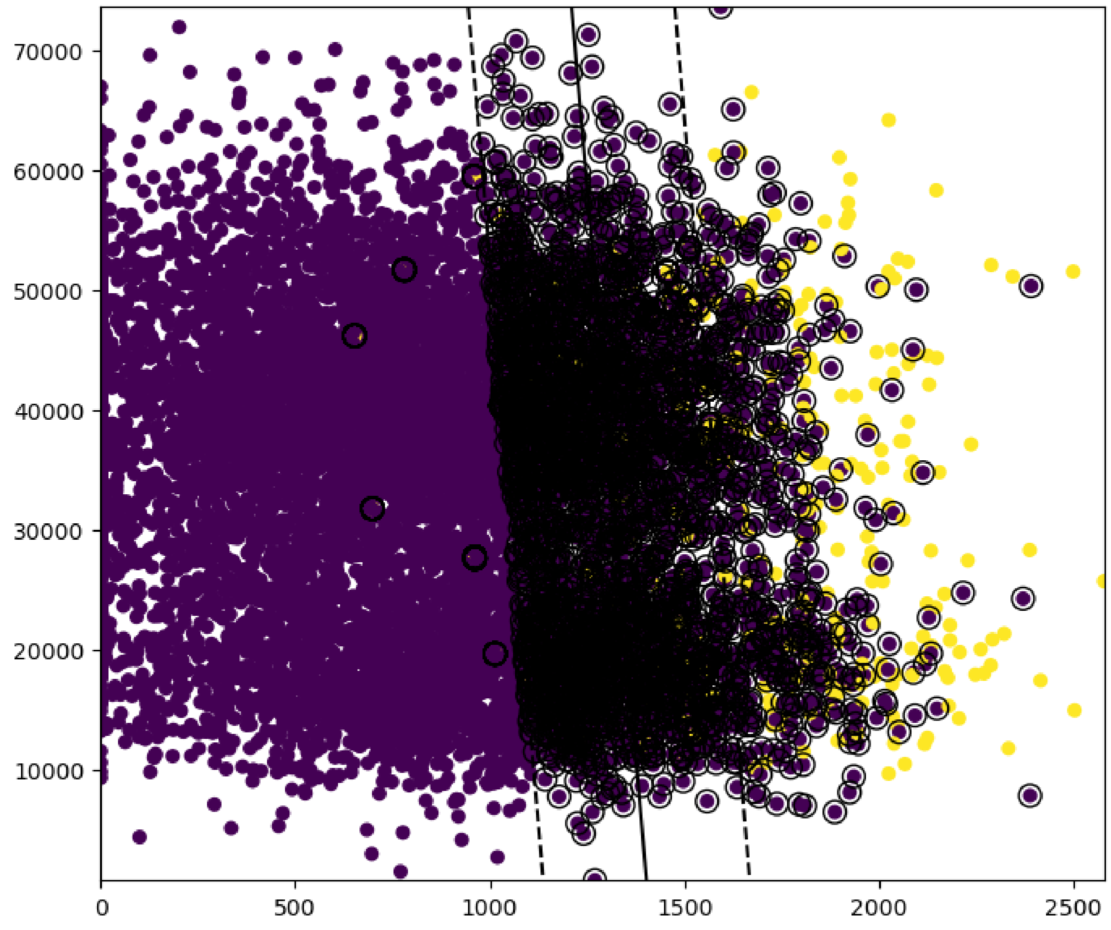
[35]:
```
# Get the best model parameters and the recall of the model
print("Params: ", cv_lin.best_params_)
print("Avg Recall: ", cv_lin.best_score_)

# Plot the CV scores
```

```
fig, ax = plt.subplots(1,3,figsize=(18,4))
sns.lineplot(x='C', y='CV accuracy', data = cv_accuracy,ax=ax[0])
sns.lineplot(x='C', y='CV f1', data = cv_accuracy,ax=ax[1])
sns.lineplot(x='C', y='CV recall', data = cv_accuracy,ax=ax[2])
plt.show()
```

```
Params:  {'svc__C': 0.1}
Avg Recall:  0.8907193412891512
```



[36]:
```
plot_margin(cv_lin.best_estimator_, X_train,y_train)
```



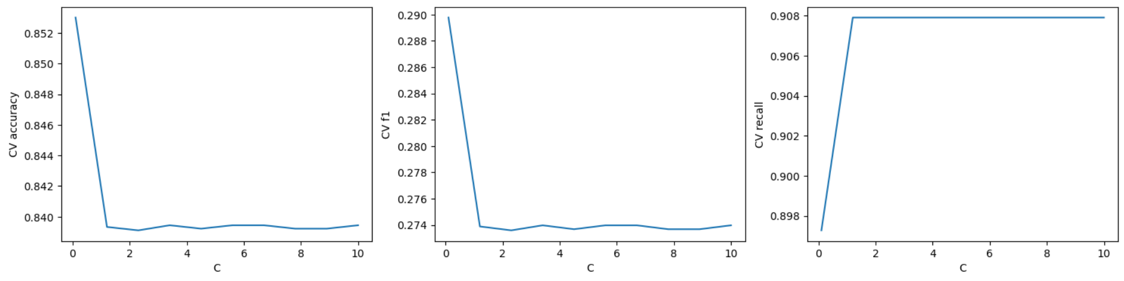
4.0.3 Exercise 11 (CORE)

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?

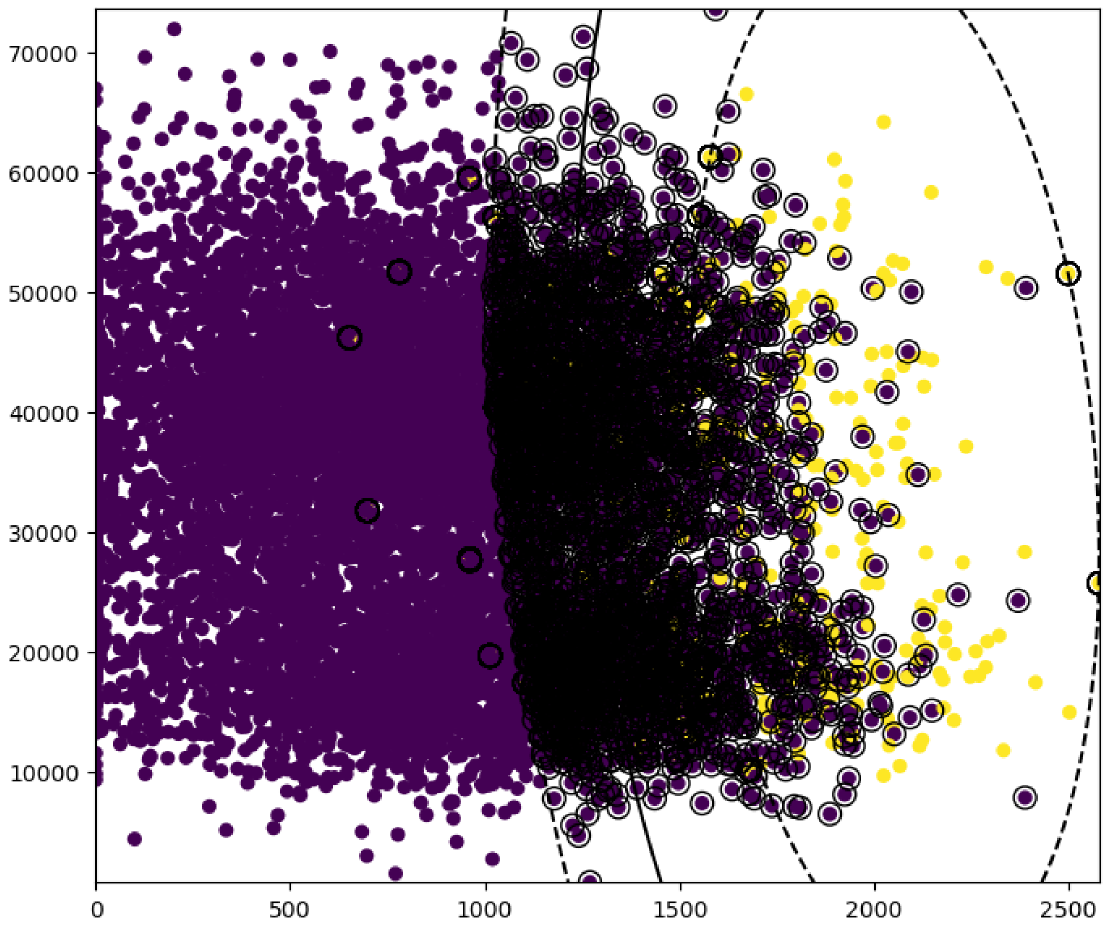
[38]:
```
svm_poly = Im_make_pipeline(
    StandardScaler(),
    RandomOverSampler(random_state = 0),
    SVC(kernel='poly', degree=2, coef0=1)
)

# Grid search over C
C = np.linspace(0.1, 10, 10)
cv_poly = GridSearchCV(
    svm_poly,
    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_poly.fit(X_train, y_train)

# Store cv scores in a data frame
cv_accuracy = pd.DataFrame(cv_poly.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'})
```

[39]:
```
# Get the best model parameters and the recall of the model
print("Params: ", cv_poly.best_params_)
print("Avg Recall: ", cv_poly.best_score_)

# Plot the CV scores
fig, ax = plt.subplots(1,3,figsize=(18,4))
sns.lineplot(x='C', y='CV accuracy', data = cv_accuracy,ax=ax[0])
sns.lineplot(x='C', y='CV f1', data = cv_accuracy,ax=ax[1])
sns.lineplot(x='C', y='CV recall', data = cv_accuracy,ax=ax[2])
plt.show()
```

```
Params:  {'svc__C': 1.2000000000000002}
Avg Recall:  0.9079025332014347
```



[40]:
```
plot_margin(cv_poly.best_estimator_, X_train,y_train)
```



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
```