HW4_SVM_Submission

March 1, 2022

1 Support Vector Machines (SVM)

2 HW 4

2.0.1 TEAM 3

- Anjali Sebastian
- Yesha Sharma
- Rupansh Phutela

2.0.2 Set up

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import pandas as pd
import os

np.random.seed(42)

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

2.0.3 utility functions

[2]:
```
def plot_class_regions_for_classifier_subplot(clf, X, y, X_test, y_test, title,␣
↪subplot, target_names = None, plot_decision_regions = True):

    numClasses = np.amax(y) + 1
    color_list_light = ['#FFFFAA', '#EFEFEF', '#AAFFAA', '#AAAAFF']
```

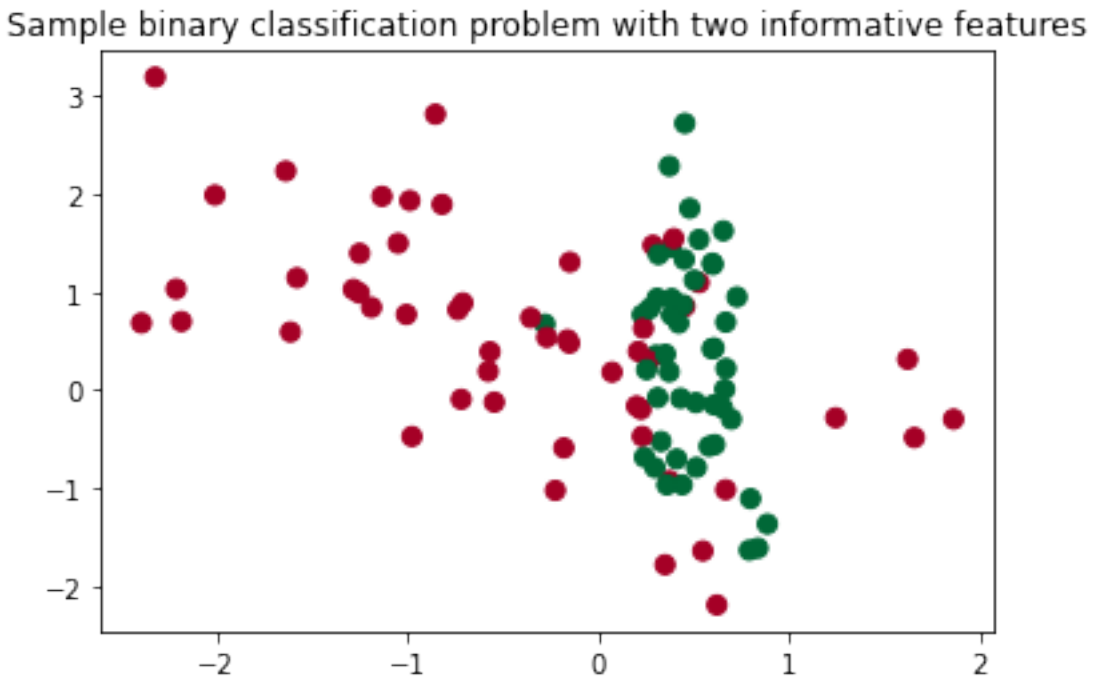
```
    color_list_bold = ['#EEEE00', '#000000', '#00CC00', '#0000CC']
    cmap_light = ListedColormap(color_list_light[0:numClasses])
    cmap_bold  = ListedColormap(color_list_bold[0:numClasses])

    h = 0.03
    k = 0.5
    x_plot_adjust = 0.1
    y_plot_adjust = 0.1
    plot_symbol_size = 50

    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()
    x2, y2 = np.meshgrid(np.arange(x_min-k, x_max+k, h), np.arange(y_min-k,␣
↪y_max+k, h))

    P = clf.predict(np.c_[x2.ravel(), y2.ravel()])
    P = P.reshape(x2.shape)

    if plot_decision_regions:
        subplot.contourf(x2, y2, P, cmap=cmap_light, alpha = 0.8)

    subplot.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, s=plot_symbol_size,␣
↪edgecolor = 'black')
    subplot.set_xlim(x_min - x_plot_adjust, x_max + x_plot_adjust)
    subplot.set_ylim(y_min - y_plot_adjust, y_max + y_plot_adjust)

    if (X_test is not None):
        subplot.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_bold,␣
↪s=plot_symbol_size, marker='^', edgecolor = 'black')
        train_score = clf.score(X, y)
        test_score  = clf.score(X_test, y_test)
        title = title + "\nTrain score = {:.2f}, Test score = {:.2f}".
↪format(train_score, test_score)

    subplot.set_title(title)

    if (target_names is not None):
        legend_handles = []
        for i in range(0, len(target_names)):
            patch = mpatches.Patch(color=color_list_bold[i],␣
↪label=target_names[i])
            legend_handles.append(patch)
        subplot.legend(loc=0, handles=legend_handles)
```

```
def plot_class_regions_for_classifier(clf, X, y, X_test=None, y_test=None,␣
↪title=None, target_names = None, plot_decision_regions = True):

    numClasses = np.amax(y) + 1
    color_list_light = ['#FFFFAA', '#EFEFEF', '#AAFFAA', '#AAAAFF']
    color_list_bold = ['#EEEE00', '#000000', '#00CC00', '#0000CC']
    cmap_light = ListedColormap(color_list_light[0:numClasses])
    cmap_bold  = ListedColormap(color_list_bold[0:numClasses])

    h = 0.03
    k = 0.5
    x_plot_adjust = 0.1
    y_plot_adjust = 0.1
    plot_symbol_size = 50

    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()
    x2, y2 = np.meshgrid(np.arange(x_min-k, x_max+k, h), np.arange(y_min-k,␣
↪y_max+k, h))

    P = clf.predict(np.c_[x2.ravel(), y2.ravel()])
    P = P.reshape(x2.shape)
    plt.figure()
    if plot_decision_regions:
        plt.contourf(x2, y2, P, cmap=cmap_light, alpha = 0.8)

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, s=plot_symbol_size,␣
↪edgecolor = 'black')
    plt.xlim(x_min - x_plot_adjust, x_max + x_plot_adjust)
    plt.ylim(y_min - y_plot_adjust, y_max + y_plot_adjust)

    if (X_test is not None):
        plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_bold,␣
↪s=plot_symbol_size, marker='^', edgecolor = 'black')
        train_score = clf.score(X, y)
        test_score  = clf.score(X_test, y_test)
        title = title + "\nTrain score = {:.2f}, Test score = {:.2f}".
↪format(train_score, test_score)

    if (target_names is not None):
        legend_handles = []
        for i in range(0, len(target_names)):
            patch = mpatches.Patch(color=color_list_bold[i],␣
↪label=target_names[i])
```

```
            legend_handles.append(patch)
        plt.legend(loc=0, handles=legend_handles)

    if (title is not None):
        plt.title(title)
    plt.show()
```

2.0.4 Synthetic dataset

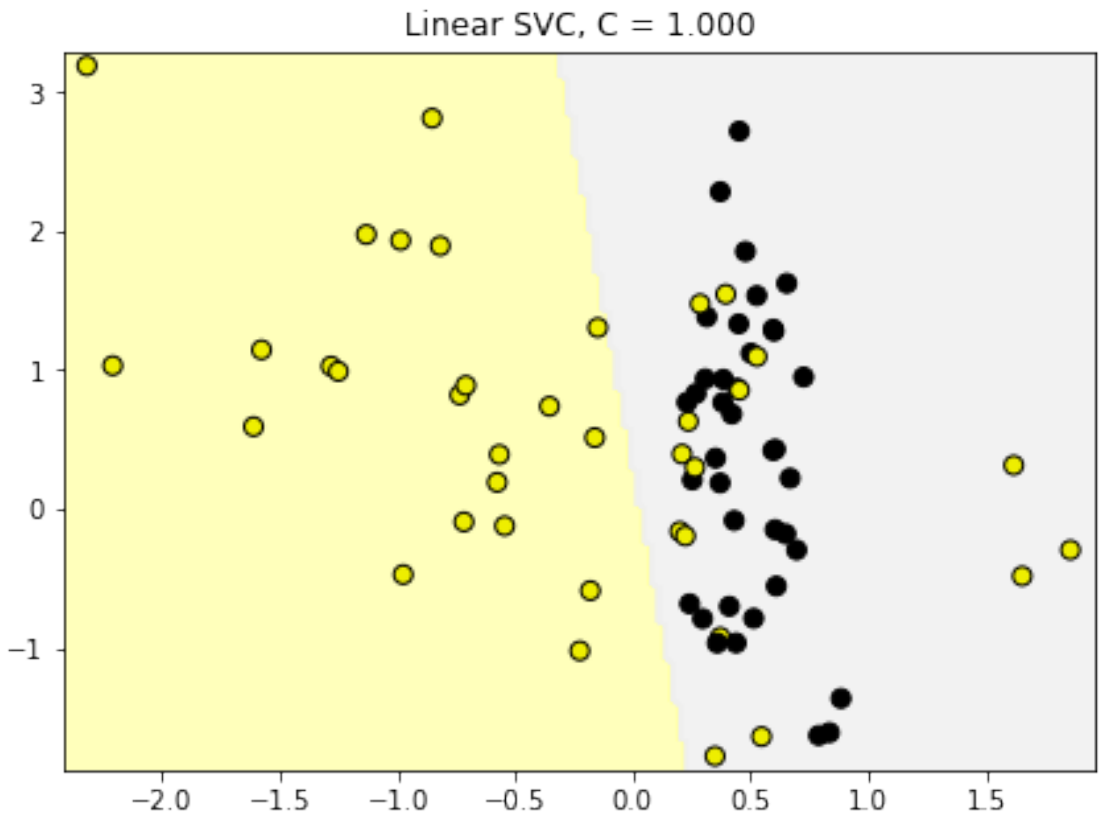
[3]:
```
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(['#FFFF00', '#00FF00', '#0000FF','#000000'])

# synthetic dataset for classification (binary)
plt.figure()
plt.title('Sample binary classification problem with two informative features')

X, y = make_classification(n_samples = 100, n_features=2,
                                n_redundant=0, n_informative=2,
                                n_clusters_per_class=1, flip_y = 0.1,
                                class_sep = 0.5, random_state=0)

plt.scatter(X[:, 0], X[:, 1], c=y, marker= 'o', s=50, cmap=plt.cm.RdYlGn)
plt.show()
```



2.0.5 Linear Support Vector Machine

[4]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

clf = SVC(kernel = 'linear', C=1.0).fit(X_train, y_train)

y_pred = clf.predict(X_test)

result_metrics = classification_report(y_test, y_pred)
print(result_metrics)

fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))

title = 'Linear SVC, C = {:.3f}'.format(1.0)
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None, None,␣
↪title, subaxes)
```

```
              precision    recall  f1-score   support

           0       0.91      0.67      0.77        15
           1       0.64      0.90      0.75        10

    accuracy                           0.76        25
   macro avg       0.78      0.78      0.76        25
weighted avg       0.80      0.76      0.76        25
```



2.0.6 Linear Support Vector Machine: C parameter

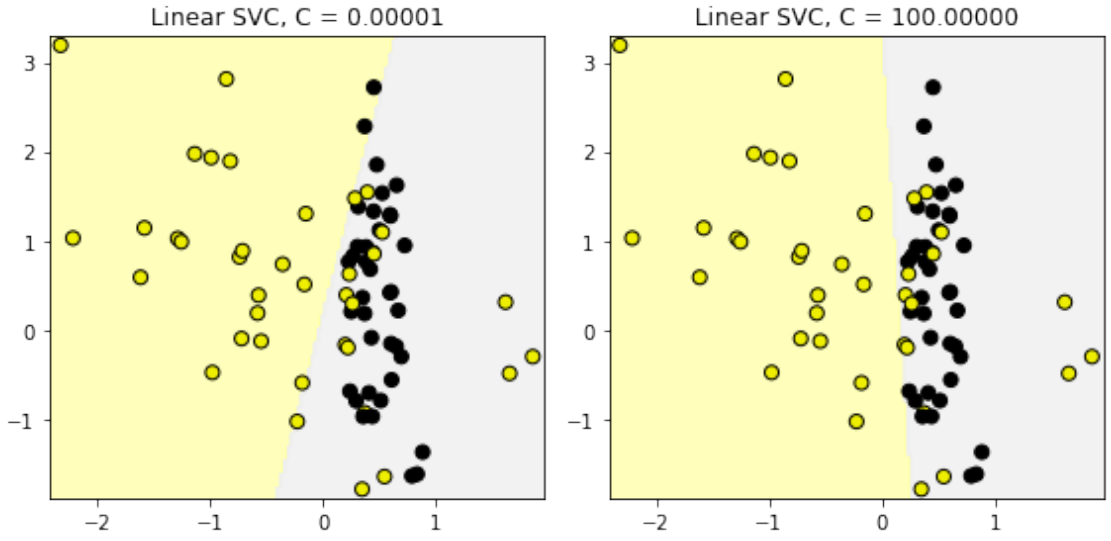
- C is regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

[5]:
```
from sklearn.svm import LinearSVC

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
this_C = 1.0

fig, subaxes = plt.subplots(1, 2, figsize=(8, 4))

for this_C, subplot in zip([0.00001, 100], subaxes):
    clf = LinearSVC(C=this_C, max_iter=1000).fit(X_train, y_train)

    y_pred = clf.predict(X_test)
    result_metrics = classification_report(y_test, y_pred)
    print(result_metrics)

    title = 'Linear SVC, C = {:.5f}'.format(this_C)
    plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
```

```
                                          None, None, title, subplot)
plt.tight_layout()
```

```
              precision    recall  f1-score   support

           0       0.91      0.67      0.77        15
           1       0.64      0.90      0.75        10

    accuracy                           0.76        25
   macro avg       0.78      0.78      0.76        25
weighted avg       0.80      0.76      0.76        25

              precision    recall  f1-score   support

           0       0.92      0.73      0.81        15
           1       0.69      0.90      0.78        10

    accuracy                           0.80        25
   macro avg       0.80      0.82      0.80        25
weighted avg       0.83      0.80      0.80        25
```

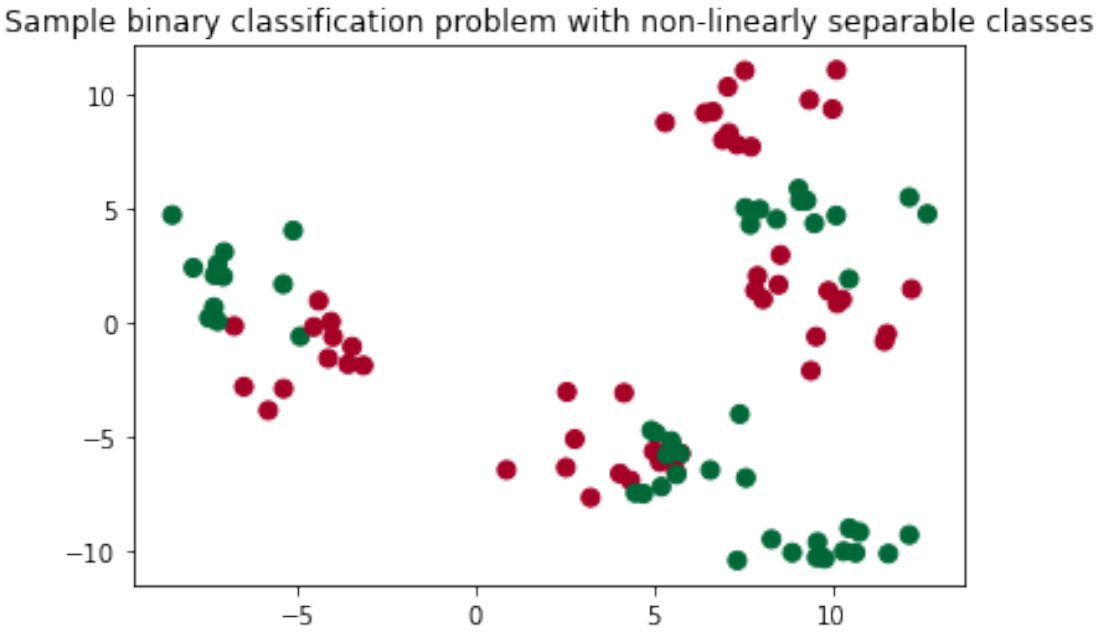
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:985:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
```



2.0.7 Kernelized Support Vector Machines

- More complex synthetic dataset

[6]:
```
# more difficult synthetic dataset for classification (binary)
# with classes that are not linearly separable
X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2, centers = 8,
                        cluster_std = 1.3, random_state = 4)
y_D2 = y_D2 % 2

plt.figure()
plt.title('Sample binary classification problem with non-linearly separable␣
 ↪classes')
plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
           marker= 'o', s=50, cmap=plt.cm.RdYlGn)
plt.show()
```



2.0.8 Classification using kernels

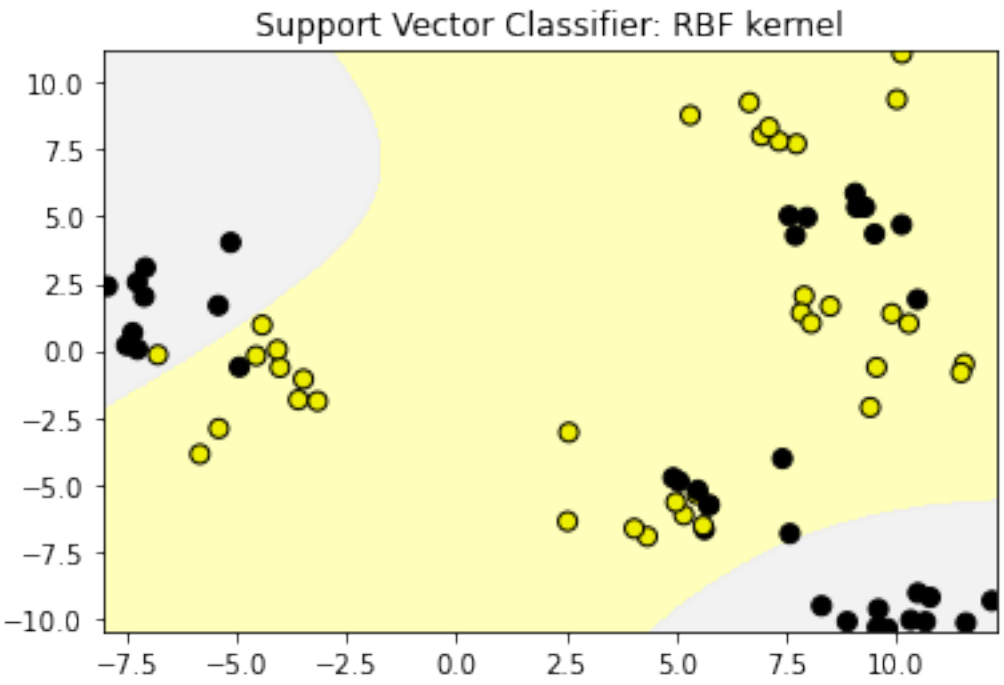
RBF kernel (Gaussian kernel)

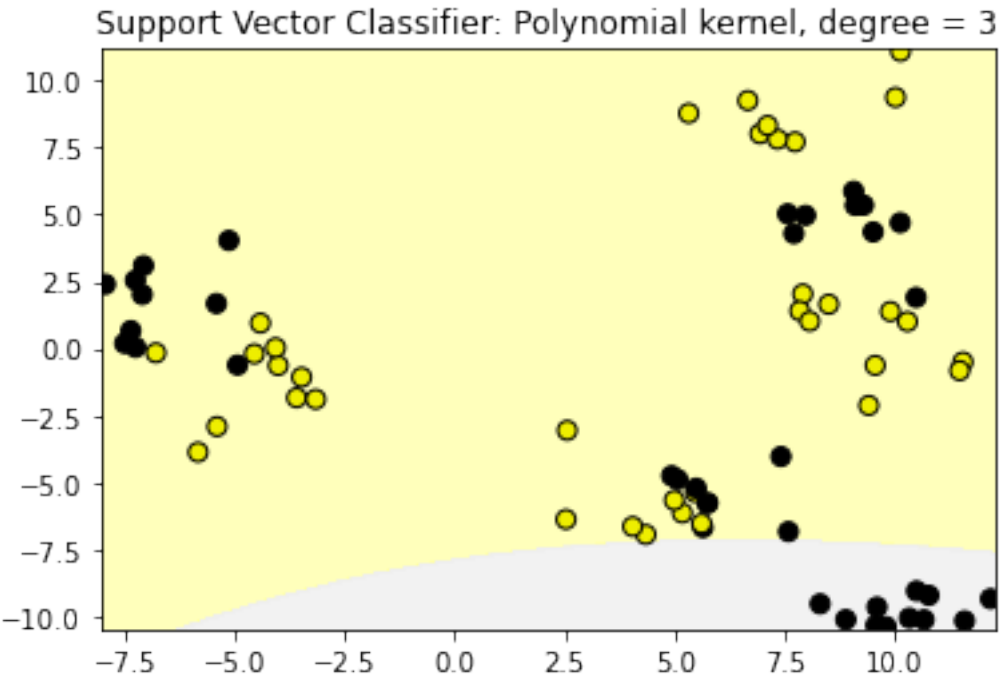
Polynomial kernel

[7]:
```
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state =␣
 ↪0)

clf1 = SVC(max_iter=10000).fit(X_train, y_train)
y_pred = clf1.predict(X_test)
```

```
result_metrics = classification_report(y_test, y_pred)
print('RBF kernel (Gaussian) results\n', result_metrics)


clf2 = SVC(kernel='poly', max_iter=10000).fit(X_train, y_train)
y_pred = clf2.predict(X_test)

result_metrics = classification_report(y_test, y_pred)
print('Polynomial kernel results\n', result_metrics)

# The default SVC kernel is radial basis function (RBF)
plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
                                  X_train, y_train, None, None,
                                  'Support Vector Classifier: RBF kernel')

# Compare decision boundries with polynomial kernel, degree = 3
plot_class_regions_for_classifier(SVC(kernel = 'poly', degree = 3)
                                  .fit(X_train, y_train), X_train,
                                  y_train, None, None,
                                  'Support Vector Classifier: Polynomial kernel,␣
 ↪degree = 3')
```

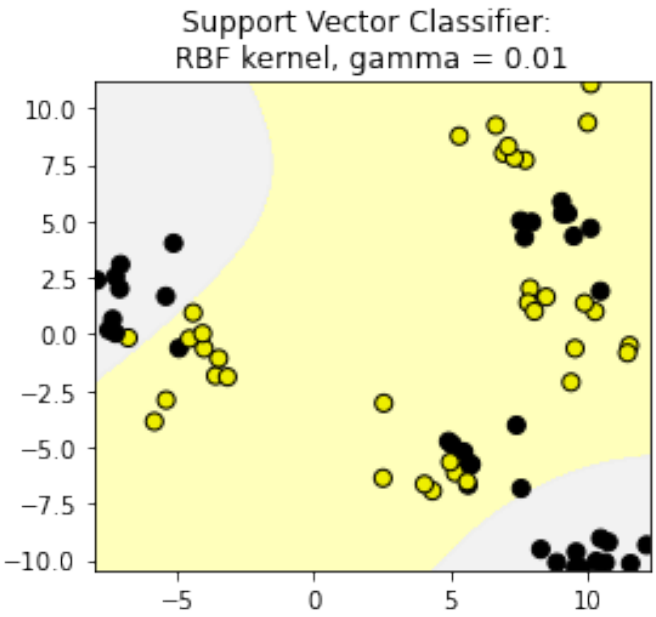
```
RBF kernel (Gaussian) results
              precision    recall  f1-score   support

           0       0.62      1.00      0.76        13
           1       1.00      0.33      0.50        12

    accuracy                           0.68        25
   macro avg       0.81      0.67      0.63        25
weighted avg       0.80      0.68      0.64        25

Polynomial kernel results
              precision    recall  f1-score   support

           0       0.63      0.92      0.75        13
           1       0.83      0.42      0.56        12

    accuracy                           0.68        25
   macro avg       0.73      0.67      0.65        25
weighted avg       0.73      0.68      0.66        25
```

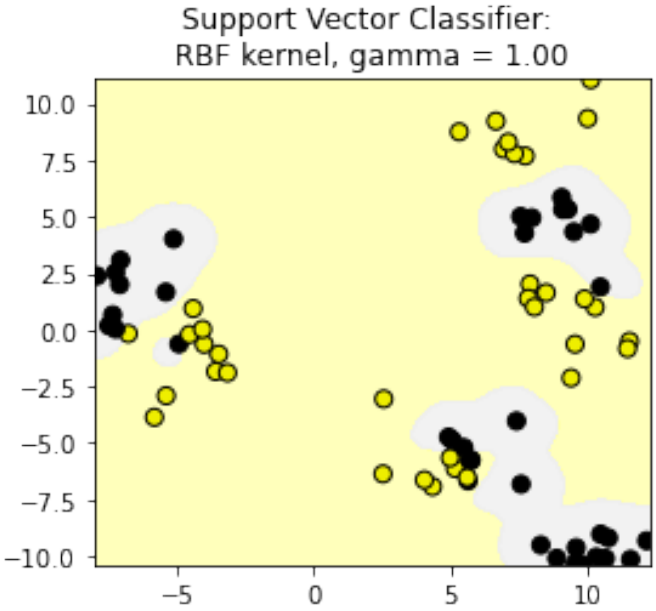

Support Vector Classifier: RBF kernel
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

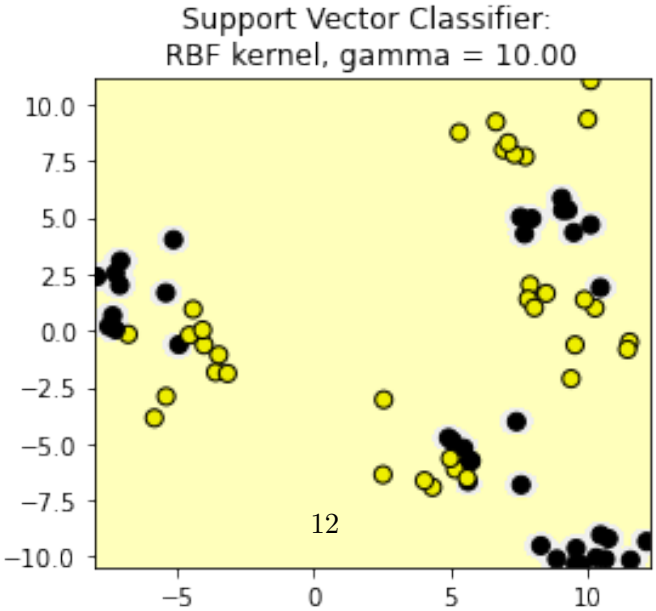

Support Vector Classifier: Polynomial kernel, degree = 3
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

2.0.9 Support Vector Machine with RBF kernel: gamma parameter

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state =␣
↪0)
fig, subaxes = plt.subplots(3, 1, figsize=(4, 11))

for this_gamma, subplot in zip([0.01, 1.0, 10.0], subaxes):
    clf = SVC(kernel = 'rbf', gamma=this_gamma).fit(X_train, y_train)

    title = 'Support Vector Classifier: \nRBF kernel, gamma = {:.2f}'.
↪format(this_gamma)
    plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                              None, None, title, subplot)
    plt.tight_layout()
```

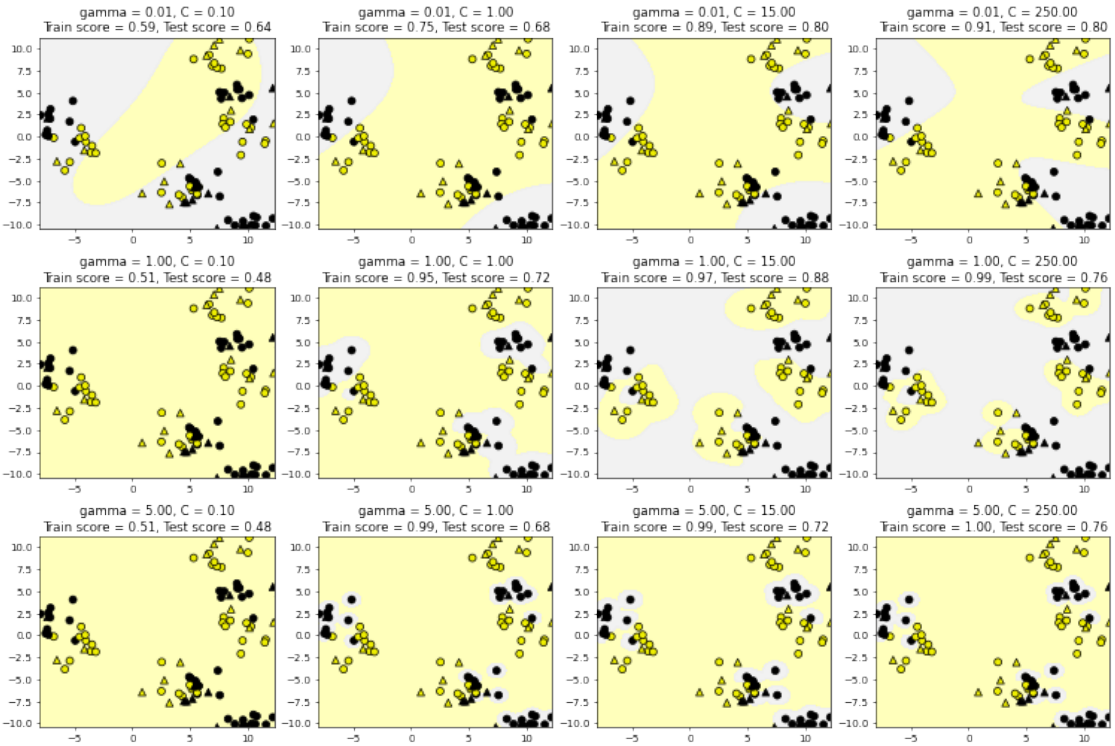

Support Vector Classifier:
RBF kernel, gamma = 0.01
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−5
0
5
10


Support Vector Classifier:
RBF kernel, gamma = 1.00
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−5
0
5
10


Support Vector Classifier:
RBF kernel, gamma = 10.00
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−5
0
5
10

2.0.10 Support Vector Machine with RBF kernel: using both C and gamma parameter

[9]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state =␣
↪0)
fig, subaxes = plt.subplots(3, 4, figsize=(15, 10), dpi=50)

for this_gamma, this_axis in zip([0.01, 1, 5], subaxes):

    for this_C, subplot in zip([0.1, 1, 15, 250], this_axis):
        title = 'gamma = {:.2f}, C = {:.2f}'.format(this_gamma, this_C)
        clf = SVC(kernel = 'rbf', gamma = this_gamma,
                  C = this_C).fit(X_train, y_train)
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                  X_test, y_test, title,
                                                  subplot)
        plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```



[ ]:

3 HW 4 ==========

Part 1:

3.0.1 TEAM 3

- Anjali Sebastian
- Yesha Sharma
- Rupansh Phutela

4 Breast cancer dataset for classification

Apply SVM linear kernel (basically no kernel) and print the performance metrics.

Apply SVM RBF kernel (Gaussian kernel) and print the performance metrics with non-normalized dataset.

Apply SVM RBF kernel with normalized dataset.

Apply SVM RBF kernel using varying C and gamma parameter values. Use C= 0.1, 1, 15, 250. Use gamma= 0.01, 1, 5. Hence, 12 subplots, similar to the above example, should be drawn.

Part 2:

Write a short comparisons of SVM linear kernel and RBF kernel.

Write a short summary of how C and gamma parameters play in SVM RBF kernel.

4.0.1 1. Download Breast Cancer Dataset

[10]:
```
from sklearn.datasets import load_breast_cancer

# Breast cancer dataset for classification
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
```

[11]:
```
X_cancer.shape
```

[11]:
```
(569, 30)
```

[12]:
```
y_cancer.shape
```

[12]:
```
(569,)
```

[13]:
```
cancer.feature_names
```

[13]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
```

```
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

[14]:
```
y_cancer
```

[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

[15]:
```
cancer.target_names
```

[15]:
```
array(['malignant', 'benign'], dtype='<U9')
```

[16]:
```
y_cancer[y_cancer==0].shape
```

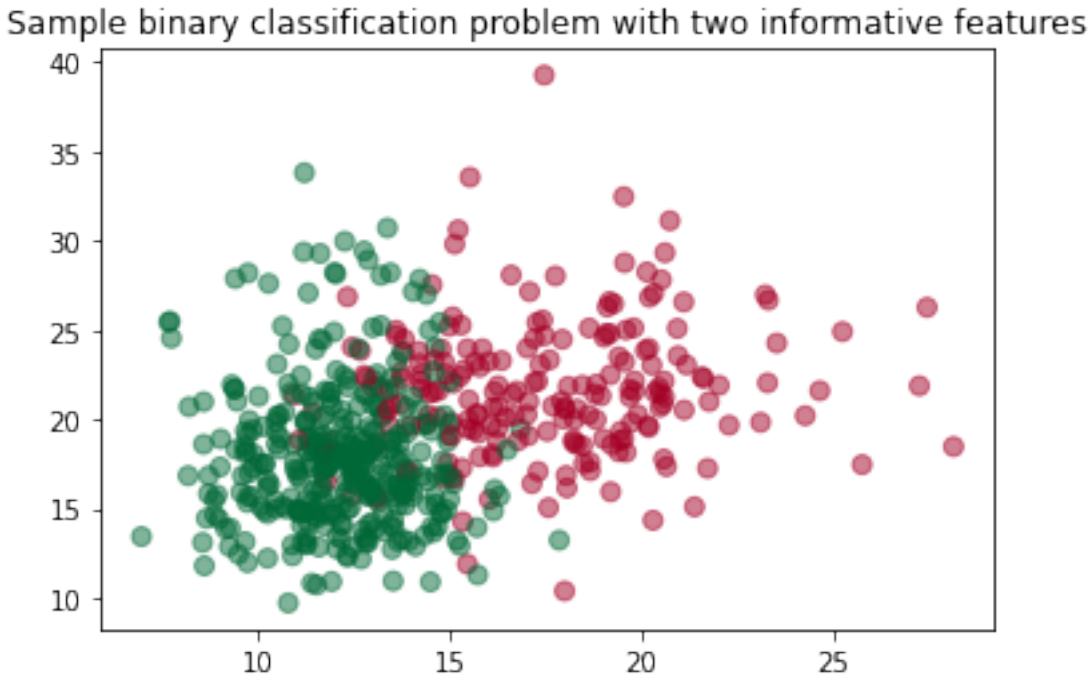
[16]:
```
(212,)
```

Note: The value 0 indicated malignant tumor(212 counts) while 1 indicates benign tumor (357 counts). Data infered from https://scikit-

learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

[17]:
```
# choosing any two attributes
attribute1 = 0
attribute2 = 1
X_cancer = X_cancer[:,[attribute1,attribute2]]
```

[18]:
```
# plotting the 2 of the atrributes of the dataset
plt.figure()
plt.title('Sample binary classification problem with two informative features')

plt.scatter(X_cancer[:, 0], X_cancer[:, 1], c=y_cancer, marker= 'o', s=50,␣
 ↪cmap=plt.cm.RdYlGn, alpha=0.5)
plt.show()
```



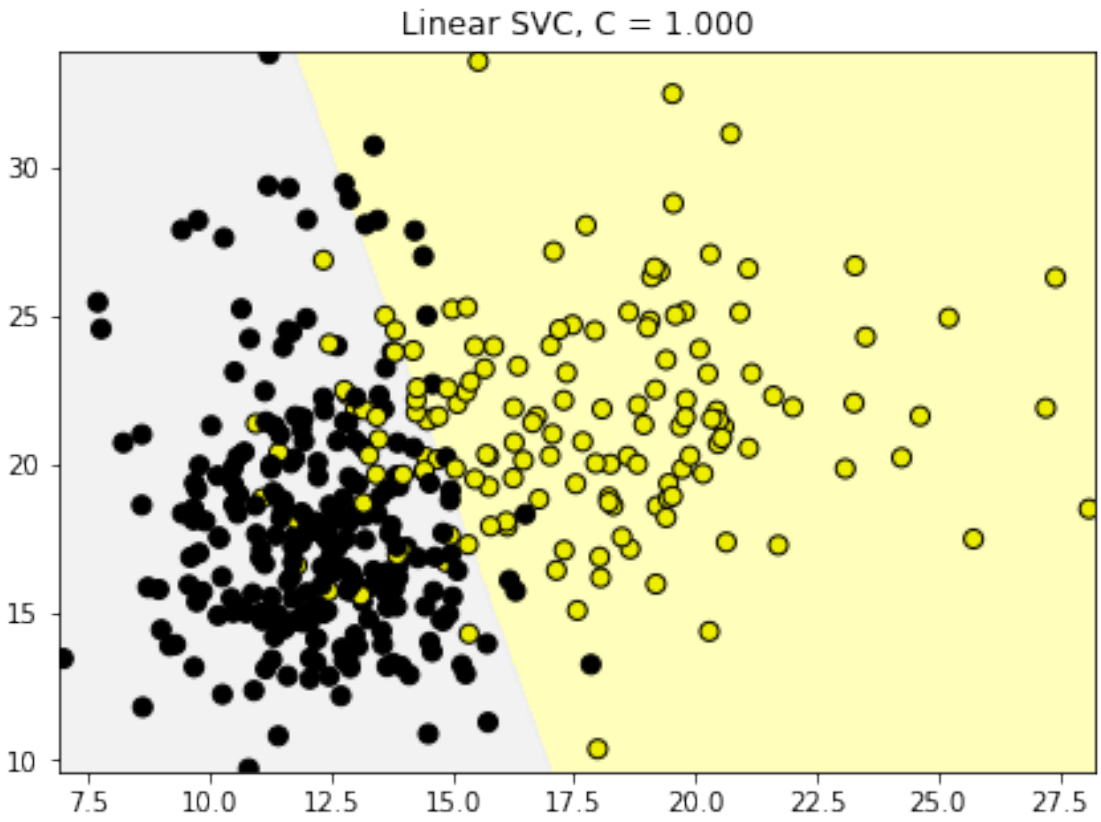
4.0.2 2. Linear Support Vector Machine

- Apply SVM linear kernel (basically no kernel) and print the performance metrics.

[19]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,␣
 ↪test_size=0.3, random_state = 0)
```

```
clf1 = SVC(kernel = 'linear', C=1.0).fit(X_train, y_train)

y_pred = clf1.predict(X_test)

result_metrics = classification_report(y_test, y_pred)
print(result_metrics)

fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))

title = 'Linear SVC, C = {:.3f}'.format(1.0)
plot_class_regions_for_classifier_subplot(clf1, X_train, y_train, None, None,␣
↪title, subaxes)
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.87        63
           1       0.92      0.94      0.93       108

    accuracy                           0.91       171
   macro avg       0.90      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171
```

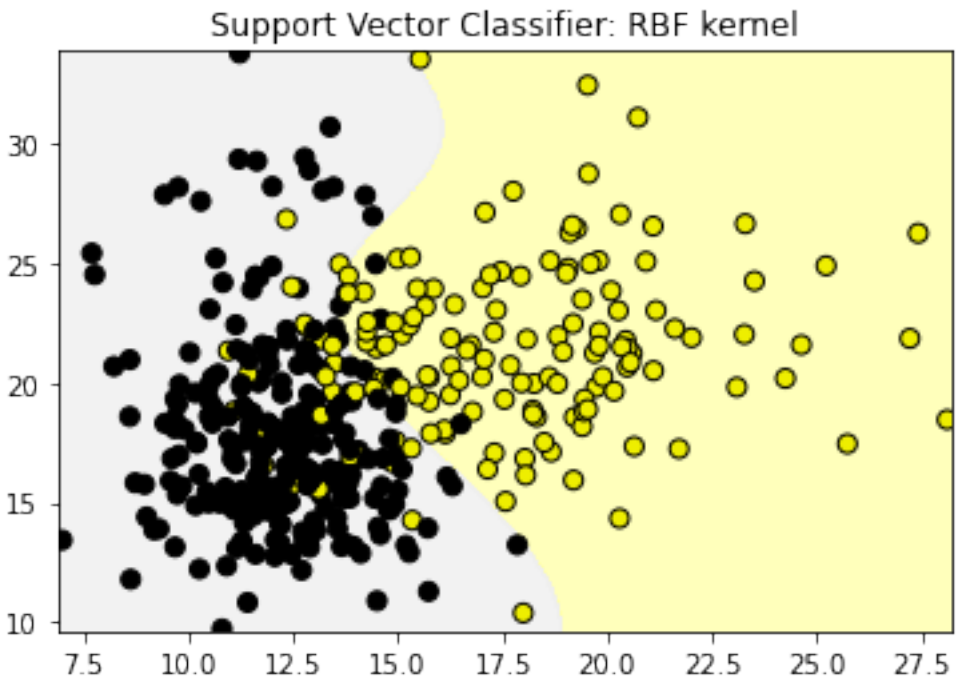


4.0.3 3. SVM with RBF Kernal without Normalization

- Apply SVM RBF kernel (Gaussian kernel) and print the performance metrics with non-normalized dataset.

[20]:
```
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,␣
 ↪test_size=0.3, random_state = 0)

clf2 = SVC(kernel='rbf', max_iter=10000, C=1).fit(X_train, y_train)
y_pred = clf2.predict(X_test)

result_metrics = classification_report(y_test, y_pred)
print('RBF kernel (Gaussian) results\n', result_metrics)


# The default SVC kernel is radial basis function (RBF)
plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
                                  X_train, y_train, None, None,
                                  'Support Vector Classifier: RBF kernel')
```

```
RBF kernel (Gaussian) results
               precision    recall  f1-score   support

           0       0.90      0.83      0.86        63
           1       0.90      0.94      0.92       108

    accuracy                           0.90       171
   macro avg       0.90      0.88      0.89       171
weighted avg       0.90      0.90      0.90       171
```



4.0.4 4. SVM with RBF Kernal with Normalization

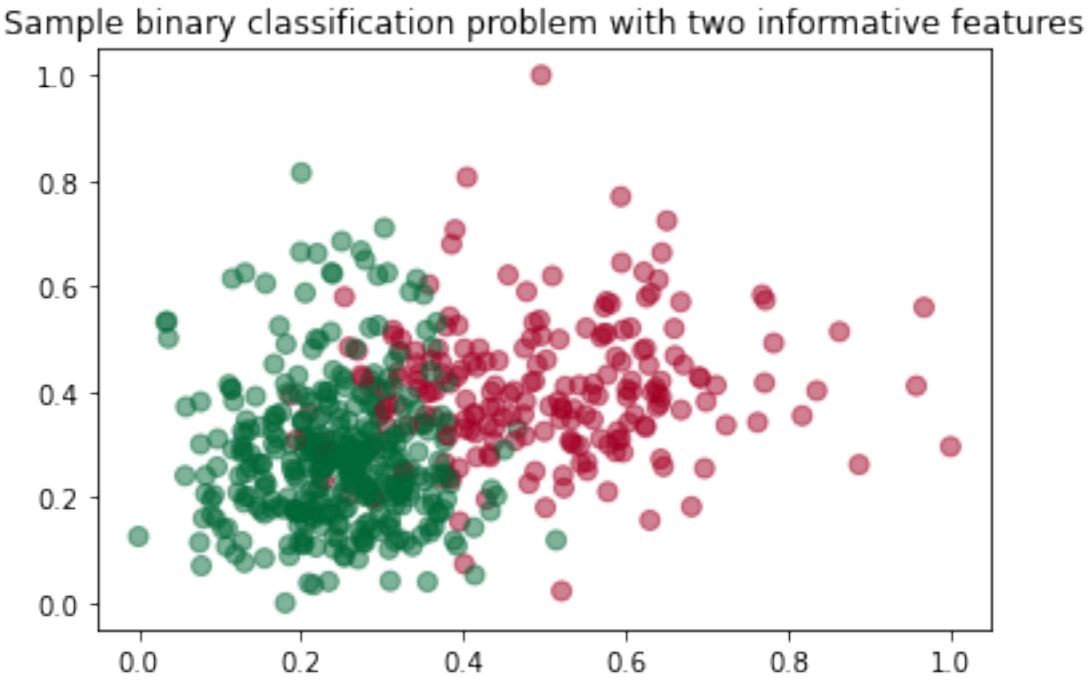
- Apply SVM RBF kernel with the normalized dataset.
- Using MinMaxScaler to normalize

[21]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# normalize the data using MinMAxScaler
X_cancer_normalized = scaler.fit_transform(X_cancer)

X_cancer_normalized
```

[21]:
```
array([[0.52103744, 0.0226581 ],
       [0.64314449, 0.27257355],
       [0.60149557, 0.3902604 ],
       …,
       [0.45525108, 0.62123774],
       [0.64456434, 0.66351031],
       [0.03686876, 0.50152181]])
```

[22]:
```
# plotting the 2 of the atrributes of the dataset
plt.figure()
plt.title('Sample binary classification problem with two informative features')
```

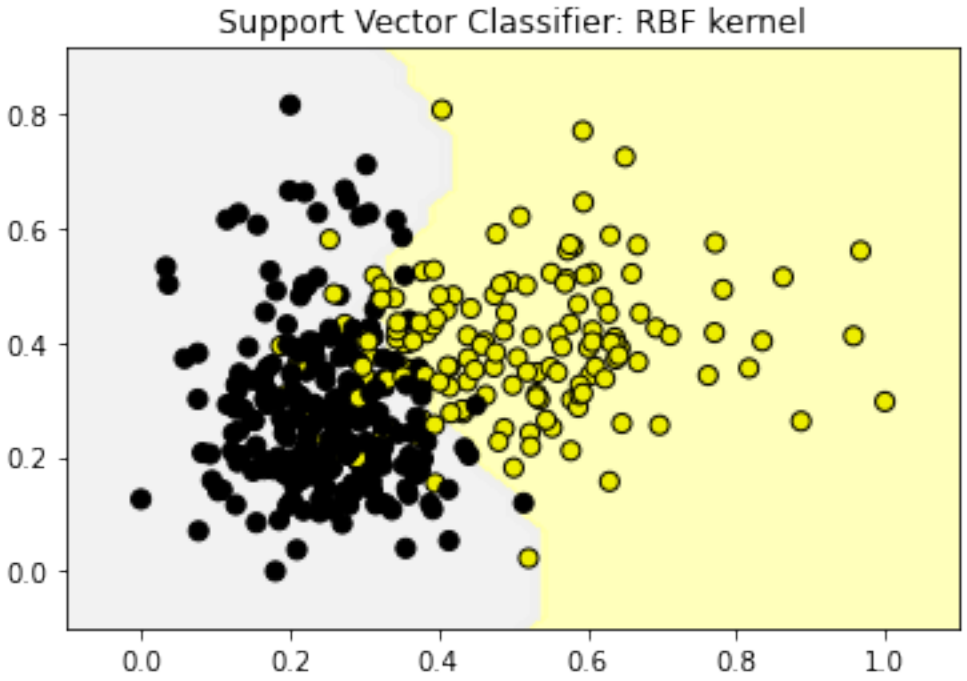
```
plt.scatter(X_cancer_normalized[:, 0], X_cancer_normalized[:, 1], c=y_cancer,␣
 ↪marker= 'o', s=50, cmap=plt.cm.RdYlGn, alpha=0.5)
plt.show()
```



[23]:
```
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(X_cancer_normalized,␣
 ↪y_cancer, test_size=0.3, random_state = 0)

clf3 = SVC(kernel='rbf', max_iter=10000).fit(X_train, y_train)
y_pred = clf3.predict(X_test)

result_metrics = classification_report(y_test, y_pred)
print('RBF kernel (Gaussian) results\n', result_metrics)


# The default SVC kernel is radial basis function (RBF)
plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
                                  X_train, y_train, None, None,
                                  'Support Vector Classifier: RBF kernel')
```

```
RBF kernel (Gaussian) results
              precision    recall  f1-score   support

           0       0.90      0.83      0.86        63
```

```
                1       0.90      0.94      0.92       108

         accuracy                           0.90       171
        macro avg       0.90      0.88      0.89       171
     weighted avg       0.90      0.90      0.90       171
```

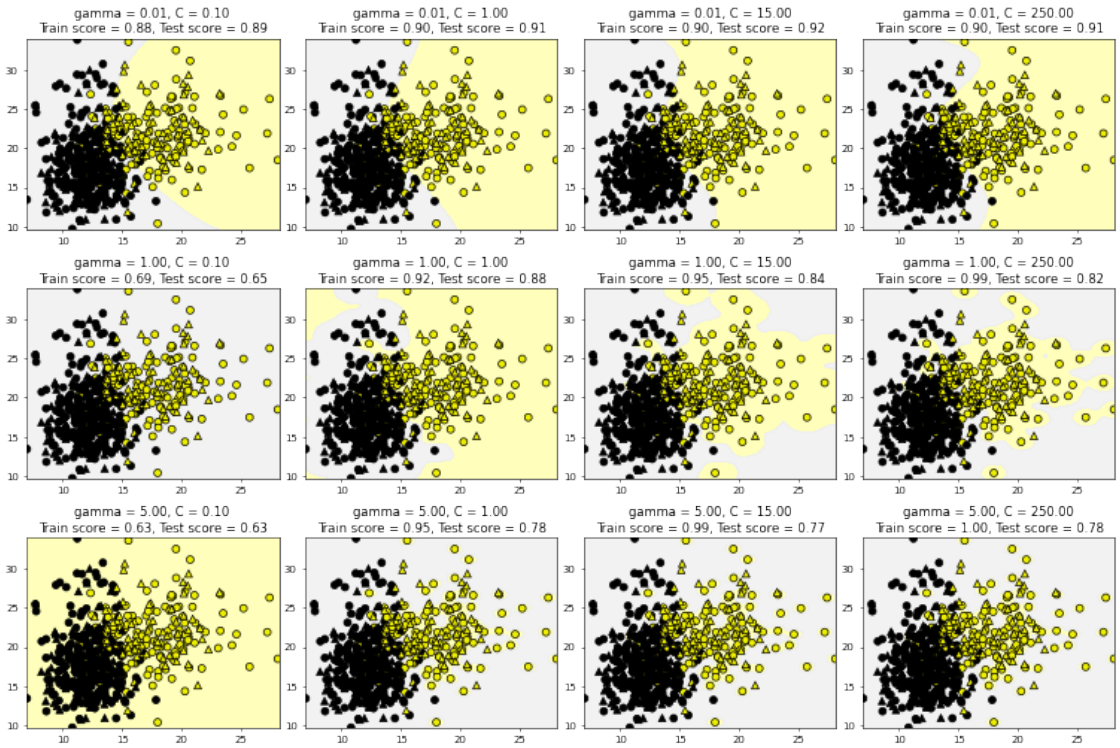


4.0.5 5. SVM RBF kernel using varying C and gamma parameter

- Apply SVM RBF kernel using varying C and gamma parameter values.
- Use C= 0.1, 1, 15, 250. Use gamma= 0.01, 1, 5.
- Hence, 12 subplots, similar to the above example, should be drawn.

[24]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,␣
 ↪test_size=0.3, random_state = 0)
fig, subaxes = plt.subplots(3, 4, figsize=(15, 10), dpi=50)

for this_gamma, this_axis in zip([0.01, 1, 5], subaxes):

    for this_C, subplot in zip([0.1, 1, 15, 250], this_axis):
        title = 'gamma = {:.2f}, C = {:.2f}'.format(this_gamma, this_C)
        clf = SVC(kernel = 'rbf', gamma = this_gamma,
```

```
                C = this_C).fit(X_train, y_train)
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                  X_test, y_test, title,
                                                  subplot)
        plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```



4.0.6 6. Write a short comparisons of SVM linear kernel and RBF kernel.

We took the first two columns from the breast cancer dataset,i.e., "mean radius" & "mean texture" for our analysis. Below is our analysis summary:- - When the kernel is linear, the scores are as below:- precision recall f1-score support

```
        0       0.89      0.86      0.87        63
        1       0.92      0.94      0.93       108

accuracy                            0.91       Data points :171
```

- When the kernel is rbf, the scores are as follows:- precision recall f1-score support

```
        0       0.90      0.83      0.86        63
        1       0.90      0.94      0.92       108
```

accuracy 0.90 Data points :171

The decision boundary for linear kernel is a straight line while it is non-linear for RBF kernel. Linear SVM is producing almost similar results as the RBF kernel. It appears that we have a

linearly separable set of data points where both kernels perform similarly. In this case, linear SVM tries to finds a straight line which maximizes the margin to the closest point to it while the RBF kernel finds a on-linear boundary to do the same. The linear kernel is very fast to operate while the RBF kernel is much slower. There seems to be very small change in model results with normalization

When we make the use of varying C value(regularization parameter) and the Gamma function, RBF kernel outperforms the linear kernel by far in accuracy, precision and recall scores.

In conclusion, the RBF kernel is generally more flexible than the linear kernel as we can model a whole lot more functions with its function space. Also, a non-linear boundary seems to be more flexible

However, if you know (i.e. it is reasonable to assume) that the true function can be well approximated by a linear or polynomial function, then a linear or polynomial kernel will be much better and it will fit the data using a lot less computation then the RBF kernel.

4.0.7 7. Write a short summary of how C and gamma parameters play in SVM RBF kernel.

- Here when gamma is kept as 0.01 and C is increased from 0.1 to 250 progressively, the train score remains consistent at 0.9 while the test score is best at 0.92 for C=15 and then decreases to 0.9 at 250.
- When gamma is 1 and same exercise is repeated, the train score increases from 0.73 to 0.99 while the test score first increases to 0.87 at C=1 and later decreases. This is typical case of overfitting.
- Further increasing gamma to 5 and changing C also shows an overfitting scenario with train score of 1 while test score reduces after C=1
- Lower C value seems to be generalizing better.

By definition, gamma value is a hyper parameter that controls the influence of features on the decision boundary. The gamma parameter is used for the Gaussian kernel function.

The higher the gamma is increased, the more influence of the features it seems to have on the decision boundary. The exercise shows that kernel functions are an efficient way to transform features into another space where the hyperplane in the new feature space does not have to be linear like the original feature space(eg: circular).

Here C is a regularization parameter used to set the tolerance of the model to allow the misclassification of data points. The C value controls the penalty of misclassification. A large value of C is results in a higher penalty for misclassification and a smaller value of C results in a smaller penalty of misclassification. We observe that for large values of C, the optimization is choosing a smaller-margin hyperplane. Conversely, a very small value of C causes the optimizer to look for a larger-margin separating hyperplane, although the hyperplane misclassifies more points. A smaller value of C causes high bias and underfitting, while a larger value of C tends to cause overfitting.

5 –END–