Pattern Recognition and Machine Learning

Assignment 3
Code (Collection of Jupyter Notebooks)

Group No. 19

Ranjith Tevnan	Sai Bandawar	Jay shah
EE18B146	EE18B150	EE18B158

13 May, 2021

Dataset 1A: Perceptron for every pair of classes

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

import tensorflow as tf
from tensorflow import keras
#tf.config.list_physical_devices('GPU')
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense

#from tensorflow.python.client import device_lib
#print(device_lib.list_local_devices())

from tensorflow.python.client import device_lib
#assert 'GPU' in str(device_lib.list_local_devices())
#print(device_lib.list_local_devices())

# confirm Keras sees the GPU




plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)
```

In [2]:
```
data=pd.read_csv("19/train.csv",header=None)
```

In [3]:
```
Y_train=np.zeros((len(data),4))
data=data.to_numpy()


X_train=data[:,0:2]
labels =data[:,2].astype(int)

for i in range(len(data)):
    Y_train[i,labels[i]]=1
```

In [4]:
```
# define the architecture of the network
np.random.seed(42)
model = Sequential()
model.add(Dense(4, input_dim=2, activation='softmax'))
#model.add(Dense(20, activation='relu', kernel_initializer="uniform"))
#model.add(Dense(4, activation='softmax'))


# train the model using SGD
print("[INFO] compiling model...")

# compile the keras model

# sgd = SGD(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer='sgd', metrics=["accuracy"])


# fit the keras model on the dataset
model.fit(X_train, Y_train, epochs=30, batch_size=1,verbose=1)


# evaluate the keras model
loss, accuracy = model.evaluate(X_train, Y_train,batch_size=1,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
[INFO] compiling model...
```

```
Epoch 1/30
800/800 [==============================] - 0s 360us/step - loss: 0.3767 - accuracy: 0.8875
Epoch 2/30
800/800 [==============================] - 0s 345us/step - loss: 0.1468 - accuracy: 0.9937
Epoch 3/30
800/800 [==============================] - 0s 343us/step - loss: 0.0948 - accuracy: 0.9987
Epoch 4/30
800/800 [==============================] - 0s 347us/step - loss: 0.0701 - accuracy: 0.9987
Epoch 5/30
800/800 [==============================] - 0s 383us/step - loss: 0.0554 - accuracy: 1.00000s - loss: 0.0579 - accuracy: 1.
Epoch 6/30
800/800 [==============================] - 0s 346us/step - loss: 0.0461 - accuracy: 1.0000
Epoch 7/30
800/800 [==============================] - 0s 360us/step - loss: 0.0393 - accuracy: 1.0000
Epoch 8/30
800/800 [==============================] - 0s 374us/step - loss: 0.0347 - accuracy: 1.0000
Epoch 9/30
800/800 [==============================] - 0s 358us/step - loss: 0.0308 - accuracy: 1.0000
Epoch 10/30
800/800 [==============================] - 0s 375us/step - loss: 0.0278 - accuracy: 1.0000
Epoch 11/30
800/800 [==============================] - 0s 348us/step - loss: 0.0256 - accuracy: 1.0000
Epoch 12/30
800/800 [==============================] - 0s 356us/step - loss: 0.0235 - accuracy: 1.0000
Epoch 13/30
800/800 [==============================] - 0s 350us/step - loss: 0.0218 - accuracy: 1.00000s - loss: 0.0208 - accuracy
Epoch 14/30
800/800 [==============================] - 0s 358us/step - loss: 0.0204 - accuracy: 1.0000
Epoch 15/30
800/800 [==============================] - 0s 354us/step - loss: 0.0191 - accuracy: 1.0000
Epoch 16/30
800/800 [==============================] - 0s 353us/step - loss: 0.0180 - accuracy: 1.0000
Epoch 17/30
800/800 [==============================] - 0s 359us/step - loss: 0.0171 - accuracy: 1.0000
Epoch 18/30
800/800 [==============================] - 0s 338us/step - loss: 0.0162 - accuracy: 1.0000
Epoch 19/30
800/800 [==============================] - 0s 352us/step - loss: 0.0153 - accuracy: 1.0000
Epoch 20/30
800/800 [==============================] - 0s 355us/step - loss: 0.0148 - accuracy: 1.0000
Epoch 21/30
800/800 [==============================] - 0s 354us/step - loss: 0.0141 - accuracy: 1.0000
Epoch 22/30
800/800 [==============================] - 0s 348us/step - loss: 0.0135 - accuracy: 1.0000
Epoch 23/30
800/800 [==============================] - 0s 351us/step - loss: 0.0130 - accuracy: 1.0000
Epoch 24/30
800/800 [==============================] - 0s 373us/step - loss: 0.0125 - accuracy: 1.00000s - loss: 0.0145 - accura
Epoch 25/30
800/800 [==============================] - 0s 348us/step - loss: 0.0121 - accuracy: 1.0000
Epoch 26/30
800/800 [==============================] - 0s 359us/step - loss: 0.0116 - accuracy: 1.0000
Epoch 27/30
800/800 [==============================] - 0s 369us/step - loss: 0.0112 - accuracy: 1.0000
Epoch 28/30
800/800 [==============================] - 0s 355us/step - loss: 0.0110 - accuracy: 1.0000
Epoch 29/30
800/800 [==============================] - 0s 353us/step - loss: 0.0106 - accuracy: 1.0000
Epoch 30/30
800/800 [==============================] - 0s 350us/step - loss: 0.0104 - accuracy: 1.0000
800/800 [==============================] - 0s 372us/step - loss: 0.0100 - accuracy: 1.0000


 [INFO] loss=0.0100, accuracy: 100.0000%
```

In [5]:

```
data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()

X_valid=data[0:60,0:2]
labels_valid=data[0:60,2].astype(int)

X_test=data[60:120,0:2]
labels_test=data[60:120,2].astype(int)


Y_valid=np.zeros((len(X_valid),4))
Y_test=np.zeros((len(X_test),4))


for i in range(len(X_valid)):
    Y_valid[i,labels_valid[i]]=1


for i in range(len(X_test)):
    Y_test[i,labels_test[i]]=1
```

In [6]:

```
loss, accuracy = model.evaluate(X_valid, Y_valid,batch_size=10,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
6/6 [==============================] - 0s 786us/step - loss: 0.0070 - accuracy: 1.0000

 [INFO] loss=0.0070, accuracy: 100.0000%
```

In [7]:

```
loss, accuracy = model.evaluate(X_test, Y_test,batch_size=10,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
6/6 [==============================] - 0s 997us/step - loss: 0.0069 - accuracy: 1.0000

 [INFO] loss=0.0069, accuracy: 100.0000%
```

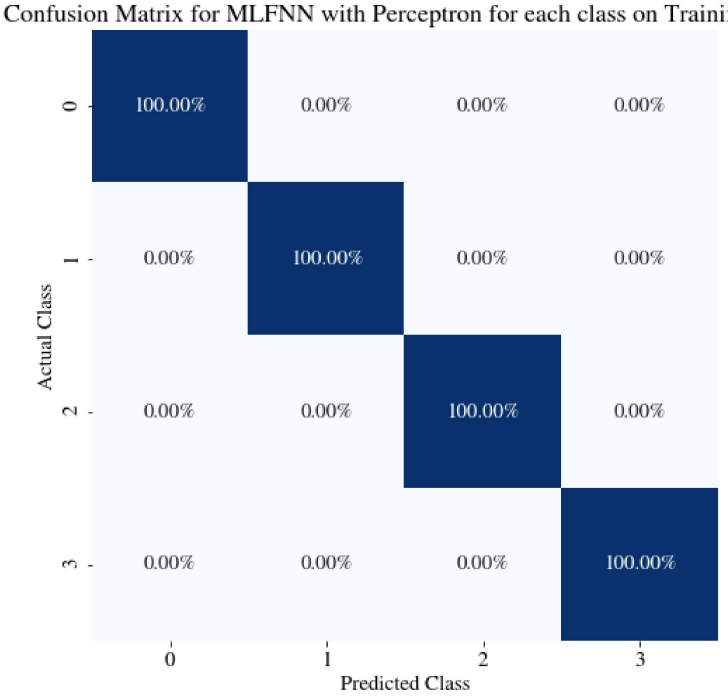
In [8]:

```
model.summary()
```

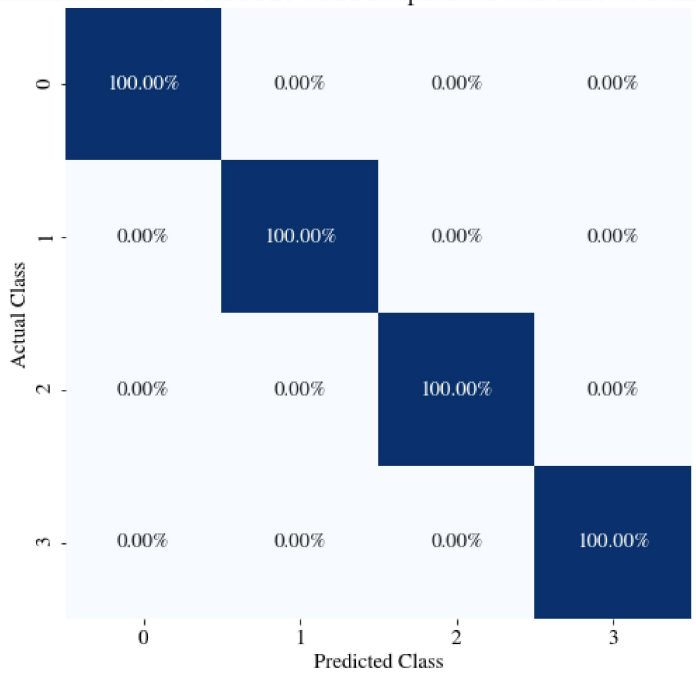
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 12
=================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
predicted_train =np.argmax(model.predict(X_train), axis=-1)
confuse=confusion_matrix(labels,predicted_train)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFNN with Perceptron for each class on Training  data')
#plt.savefig("MLFNN_Confusion_1.png")

plt.show()

predicted_test =np.argmax(model.predict(X_test), axis=-1)
confuse=confusion_matrix(labels_test,predicted_test)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFNN with Perceptron for each class on Testing  data')
#plt.savefig("MLFNN_Confusion_2.png")

plt.show()
```



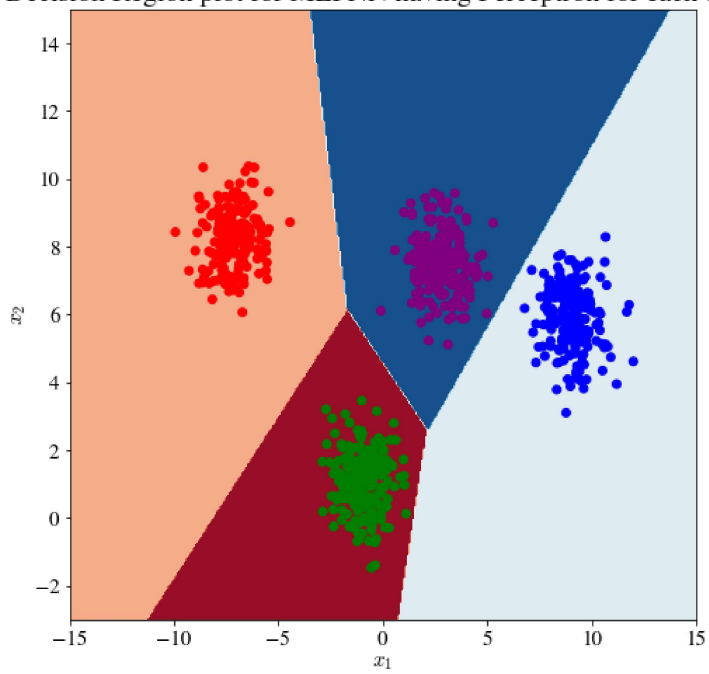
Confusion Matrix for MLFNN with Perceptron for each class on Testing data



In [10]:

```
x1=np.linspace(-15,15,num=350)
x2=np.linspace(-3,15,num=350)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))


predicted=np.argmax(model.predict(grid),axis=1)


predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']
Y_train=pd.read_csv("19/train.csv",header=None)
Y_train=Y_train.to_numpy()
Y_train=Y_train[:,-1]
plt.scatter(X_train[:,0], X_train[:,1], c= Y_train, cmap=matplotlib.colors.ListedColormap(colors))


plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region plot for MLFNN having Perceptron for each class',fontsize=20)

"""recs = []
for i in range(0,len(colors)):
    recs.append(mpatches.Rectangle((0,0),1,1,fc=colors[i]))
plt.legend(recs,unique,loc=4)
"""

#plt.savefig('MLFNN_Decision.png')
plt.show()
#plt.close()
```

Decision Region plot for MLFNN having Perceptron for each class

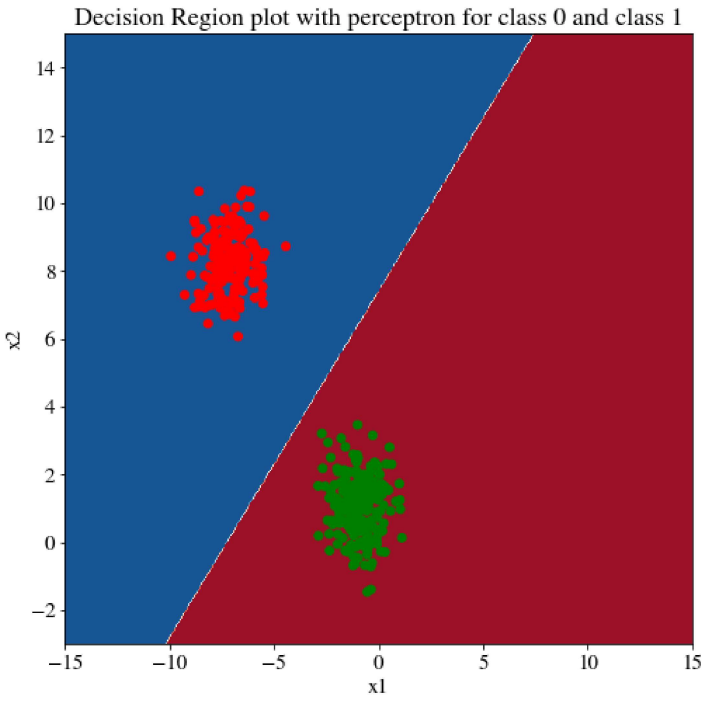


In [11]:

```
(unique, counts) = np.unique(Y_train, return_counts=True)
#print(unique)
model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))



# sgd = SGD(lr=0.01)
model.compile(loss="binary_crossentropy", optimizer='sgd', metrics=["accuracy"])


data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()

X_valid=data[0:60,0:2]
Y_valid=data[0:60,2]

X_test=data[60:120,0:2]
Y_test=data[60:120,2]



for i in range(len(unique)):
    for p in range(i+1,len(unique)):

        X_new_train=X_train[np.where(Y_train==i)]
        X_new_train=np.vstack((X_new_train,X_train[np.where(Y_train==p)]))

        Y_new_train=np.hstack((np.zeros_like(Y_train[np.where(Y_train==i)]),np.ones_like(Y_train[np.where(Y_train==p)])))

        model.fit(X_new_train, Y_new_train, epochs=18, batch_size=1,verbose=0)
        x1=np.linspace(-15,15,num=350)
        x2=np.linspace(-3,15,num=350)
        xx1, xx2 = np.meshgrid(x1, x2)
        r1, r2 = xx1.flatten(), xx2.flatten()
        r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
        grid = np.hstack((r1,r2))

        #predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(grid.shape[0]))
        predicted=model.predict(grid,verbose=1)
        #print(predicted)
        predicted_new=[]
        for a in predicted:
            predicted_new.append(a>=0.5)
        predicted=predicted_new
        pos=np.empty(xx1.shape+(2,))
        pos[:,:,0]=xx1
        pos[:,:,1]=xx2
        #print(predicted)

        predicted=np.array(predicted)
        predicted=predicted.reshape(xx1.shape)
        fig = plt.figure(figsize=(8,8))
        plt.contourf(xx1, xx2, predicted, cmap='RdBu')
```

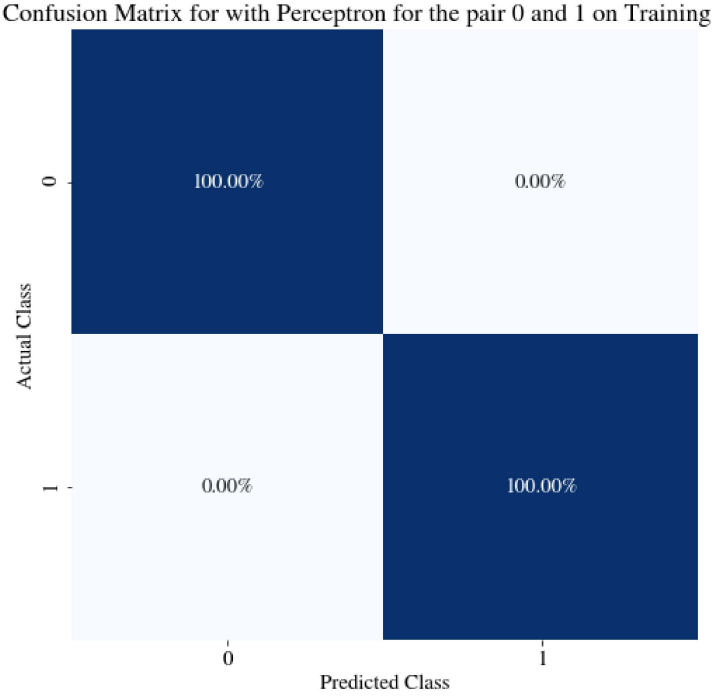
```
colors = ['green','red']
#real=X_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])]
plt.scatter(X_new_train[:,0], X_new_train[:,1], c=Y_new_train, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('x1')
plt.ylabel('x2')
plt.title(f'Decision Region plot with perceptron for class {i} and class {p}')
#plt.legend()
#plt.savefig('plot_Gaussian_2.png')
plt.show()

predicted=model.predict(X_new_train,verbose=1)

predicted_new=[]
for a in predicted:
    predicted_new.append(a>=0.5)
predicted=predicted_new

confuse=confusion_matrix(Y_new_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=[i,p],yticklabels=[i,p])
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for with Perceptron for the pair {i} and {p} on Training  data')
#plt.savefig("MLFNN_Confusion_2.png")

plt.show()

X_new_test=X_test[np.where(Y_test==i)]
X_new_test=np.vstack((X_new_test,X_test[np.where(Y_test==p)]))
#print(X_new_train.shape)
Y_new_test=np.hstack((np.zeros_like(Y_test[np.where(Y_test==i)]),np.ones_like(Y_test[np.where(Y_test==p)])))

predicted=model.predict(X_new_test,verbose=1)

predicted_new=[]
for a in predicted:
    predicted_new.append(a>=0.5)
predicted=predicted_new

confuse=confusion_matrix(Y_new_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=[i,p],yticklabels=[i,p])
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for with Perceptron for the pair {i} and {p} on Testing  data')
#plt.savefig("MLFNN_Confusion_2.png")

plt.show()
```

```
3829/3829 [==============================] - 1s 277us/step
```



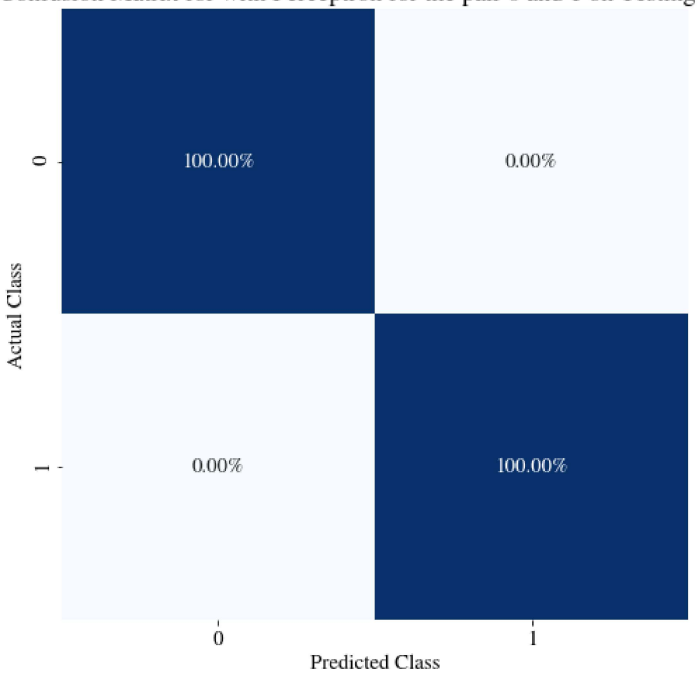
```
13/13 [==============================] - 0s 382us/step
```

Confusion Matrix for with Perceptron for the pair 0 and 1 on Training data

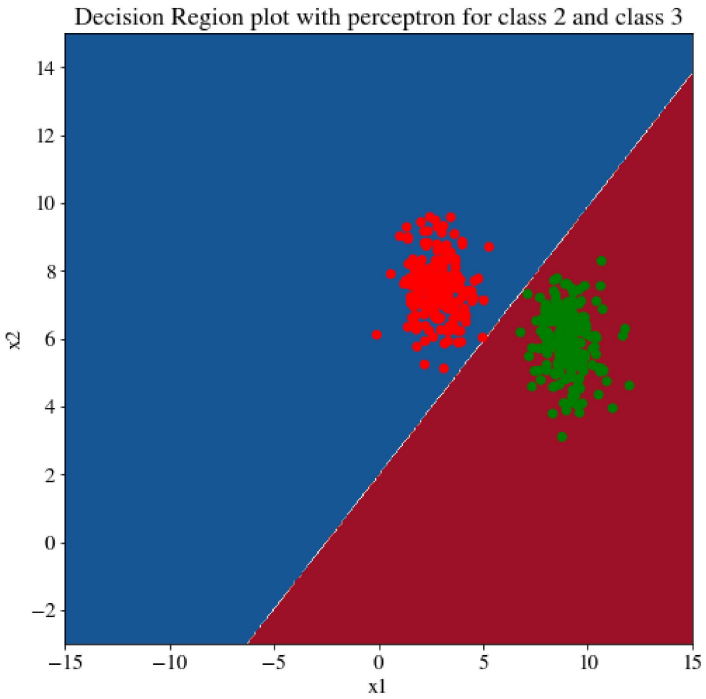



```
1/1 [==============================] - 0s 993us/step
```

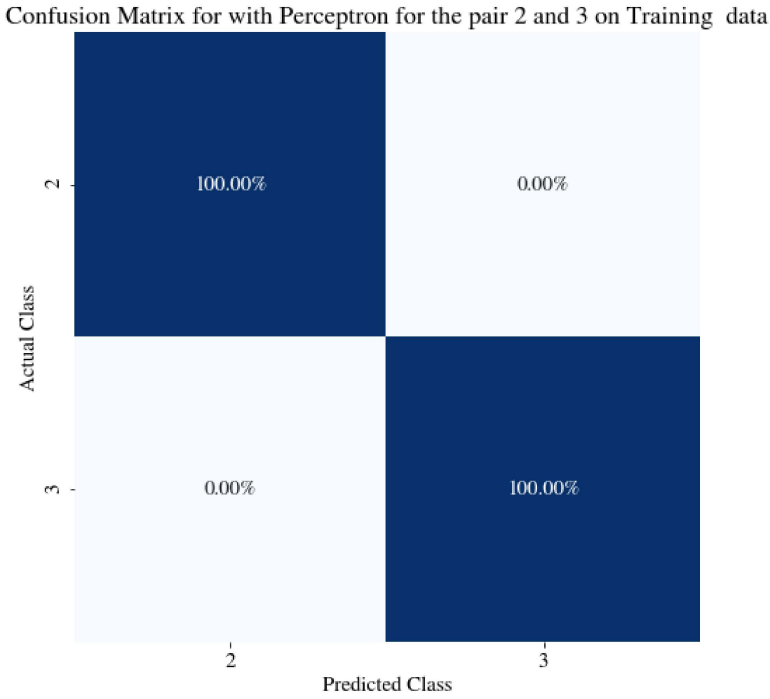
Confusion Matrix for with Perceptron for the pair 0 and 1 on Testing data




```
3829/3829 [==============================] - 1s 295us/step
```




```
13/13 [==============================] - 0s 441us/step
```




```
1/1 [==============================] - 0s 1000us/step
```

Dataset 1A: Multilayer feedforward neural network (MLFFNN) with a single hidden layer for all classes

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.optimizers import SGD
from keras.layers import Dense

%matplotlib inline


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

#Fixing seed value so random numbers generated can be predicted
seed_value = 42
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

In [2]:

```
#Training Data

data=pd.read_csv("19/train.csv",header=None)
data=data.to_numpy()

np.random.shuffle(data)
X_train=data[:,0:2]
labels =data[:,2].astype(int)

Y_train=np.zeros((len(data),4))
for i in range(len(data)):
    Y_train[i,labels[i]]=1


#Validation and Test Data
from sklearn.model_selection import train_test_split

data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()
X=data[:,0:2]
y=data[:,-1].astype(int)


X_valid, X_test, labels_valid, labels_test = train_test_split(X, y,test_size=0.5)

Y_valid=np.zeros((len(X_valid),4))
Y_test=np.zeros((len(X_test),4))


for i in range(len(X_valid)):
    Y_valid[i,labels_valid[i]]=1


for i in range(len(X_test)):
    Y_test[i,labels_test[i]]=1
```

In [3]:

```
nodes=3

# checkpoint= keras.callbacks.ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_or
```

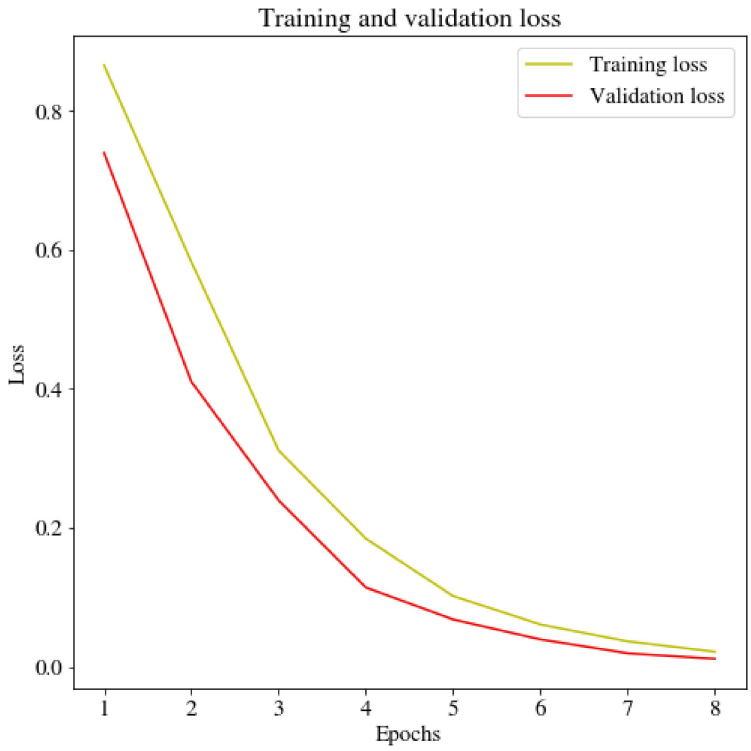
```
es = keras.callbacks.EarlyStopping(
        # Stop training when `val_loss` is no longer improving
        monitor="val_loss",
        # "no longer improving" being defined as "no better than 1e-3 less"
        min_delta=1e-2,
        # "no longer improving" being further defined as "for at least 2 epochs"
        mode='min',
        patience=1,
        verbose=1,
    )


# define the architecture of the network

model = Sequential()
model.add(Dense(nodes, input_dim=2, activation='relu'))
model.add(Dense(4, activation='softmax'))


# train the model using SGD
print("[INFO] compiling model...")

# compile the keras model


model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])


# fit the keras model on the dataset
# fit the keras model on the dataset
history= model.fit(X_train, Y_train,
                        epochs=50,
                        batch_size=1,
                        verbose=1,
                        callbacks=[es],
                        validation_data=(X_valid, Y_valid)
                       )



# evaluate the keras model
loss, accuracy = model.evaluate(X_train, Y_train,batch_size=1,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
[INFO] compiling model...
Epoch 1/50
800/800 [==============================] - 1s 736us/step - loss: 0.8655 - accuracy: 0.4800 - val_loss: 0.7395 - val_accura
cy: 0.5833
Epoch 2/50
800/800 [==============================] - 0s 454us/step - loss: 0.5828 - accuracy: 0.8313 - val_loss: 0.4104 - val_accura
cy: 0.9333
Epoch 3/50
800/800 [==============================] - 0s 454us/step - loss: 0.3115 - accuracy: 0.9538 - val_loss: 0.2397 - val_accura
cy: 1.0000
Epoch 4/50
800/800 [==============================] - 0s 472us/step - loss: 0.1846 - accuracy: 0.9862 - val_loss: 0.1143 - val_accura
cy: 1.0000
Epoch 5/50
800/800 [==============================] - 0s 446us/step - loss: 0.1020 - accuracy: 0.9950 - val_loss: 0.0683 - val_accura
cy: 1.0000
Epoch 6/50
800/800 [==============================] - 0s 448us/step - loss: 0.0611 - accuracy: 0.9975 - val_loss: 0.0395 - val_accura
cy: 1.0000
Epoch 7/50
800/800 [==============================] - 0s 480us/step - loss: 0.0367 - accuracy: 0.9987 - val_loss: 0.0196 - val_accura
cy: 1.0000
Epoch 8/50
800/800 [==============================] - 0s 477us/step - loss: 0.0218 - accuracy: 1.0000 - val_loss: 0.0116 - val_accura
cy: 1.0000
Epoch 00008: early stopping
800/800 [==============================] - 0s 399us/step - loss: 0.0168 - accuracy: 1.0000


 [INFO] loss=0.0168, accuracy: 100.0000%
```

In [4]:

```
#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


acc = history.history['accuracy']  #Use accuracy if acc doesn't work
val_acc = history.history['val_accuracy']  #Use val_accuracy if acc doesn't work
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [5]:
```
loss, accuracy = model.evaluate(X_train, Y_train,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
25/25 [==============================] - 0s 687us/step - loss: 0.0168 - accuracy: 1.0000


 [INFO] loss=0.0168, accuracy: 100.0000%
```

In [6]:
```
loss, accuracy = model.evaluate(X_valid, Y_valid,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
2/2 [==============================] - 0s 999us/step - loss: 0.0116 - accuracy: 1.0000


 [INFO] loss=0.0116, accuracy: 100.0000%
```

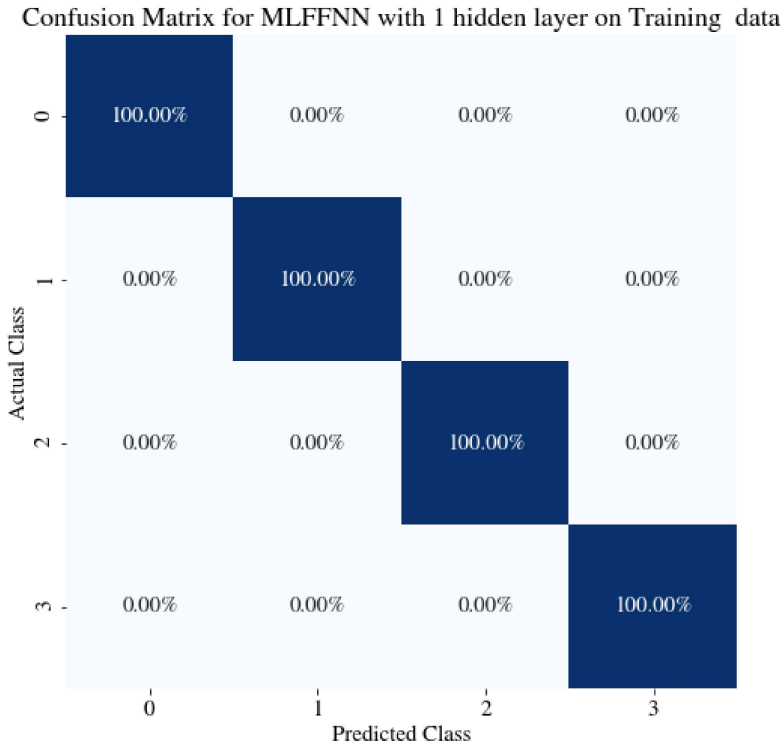
In [7]:
```
loss, accuracy = model.evaluate(X_test, Y_test,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

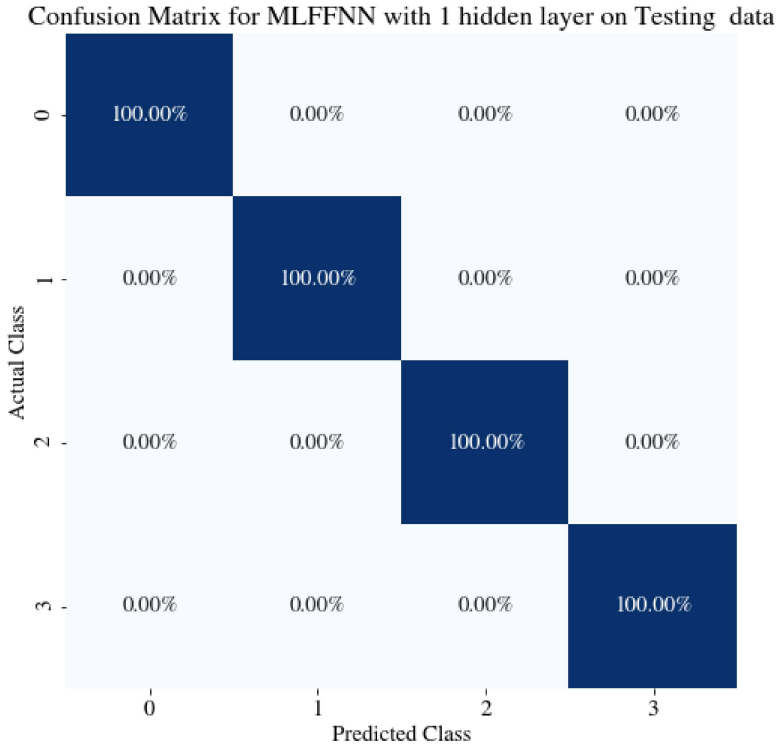
```
2/2 [==============================] - 0s 980us/step - loss: 0.0119 - accuracy: 1.0000


 [INFO] loss=0.0119, accuracy: 100.0000%
```

In [8]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 3)                 9
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 16
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
predicted_train =np.argmax(model.predict(X_train), axis=-1)
confuse=confusion_matrix(labels,predicted_train)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFFNN with 1 hidden layer on Training  data')
# plt.savefig("MLFNN_Confusion_train.png")

plt.show()

predicted_test =np.argmax(model.predict(X_test), axis=-1)
confuse=confusion_matrix(labels_test,predicted_test)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFFNN with 1 hidden layer on Testing  data')
# plt.savefig("MLFNN_Confusion_test.png")

plt.show()
```

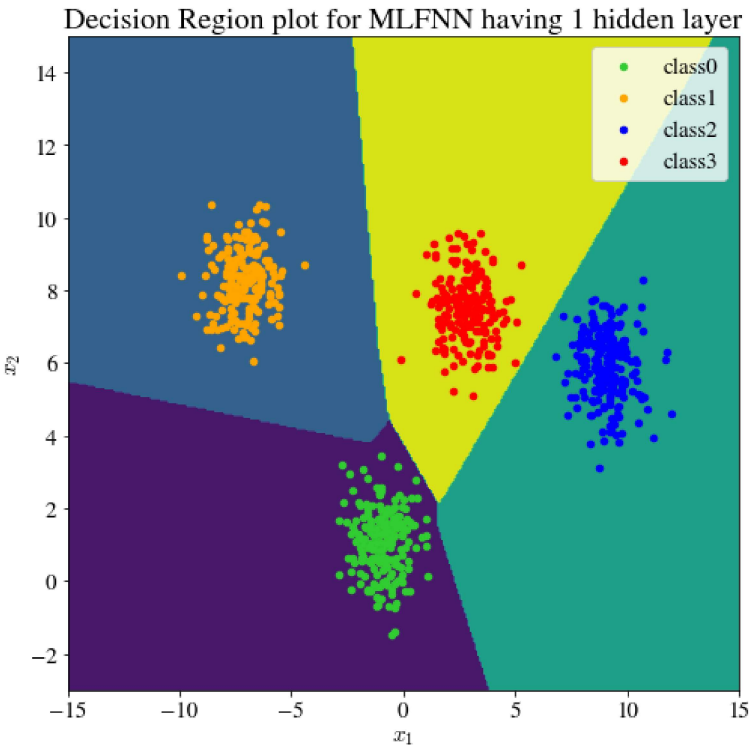




In [10]:

```
x1=np.linspace(-15,15,num=350)
x2=np.linspace(-3,15,num=350)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))


predicted=np.argmax(model.predict(grid),axis=1)


num_cores = multiprocessing.cpu_count()

predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='viridis')
```

```
# colors = ['green','red','blue','purple']
# plt.scatter(X_train[:,0], X_train[:,1], c= labels, cmap=matplotlib.colors.ListedColormap(colors))


data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==0], [data[m][1] for m in range(len(data))
                                    if data[m][2]==0], c ="limegreen",label="class0",s=20)
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==1], [data[m][1] for m in range(len(data))
                                                                      if data[m][2]==1], c ="orange",label="class1",s=20)
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==2], [data[m][1] for m in range(len(data))
                                                                      if data[m][2]==2], c ="blue",label="class2",s=20)
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==3], [data[m][1] for m in range(len(data))
                                                                      if data[m][2]==3], c ="red",label="class3",s=20)

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region plot for MLFNN having 1 hidden layer',fontsize=20)
plt.legend()



# plt.savefig('MLFNN_Decision.png')
plt.show()
```



Dataset 1A: Linear SVM classifier for every pair of classes

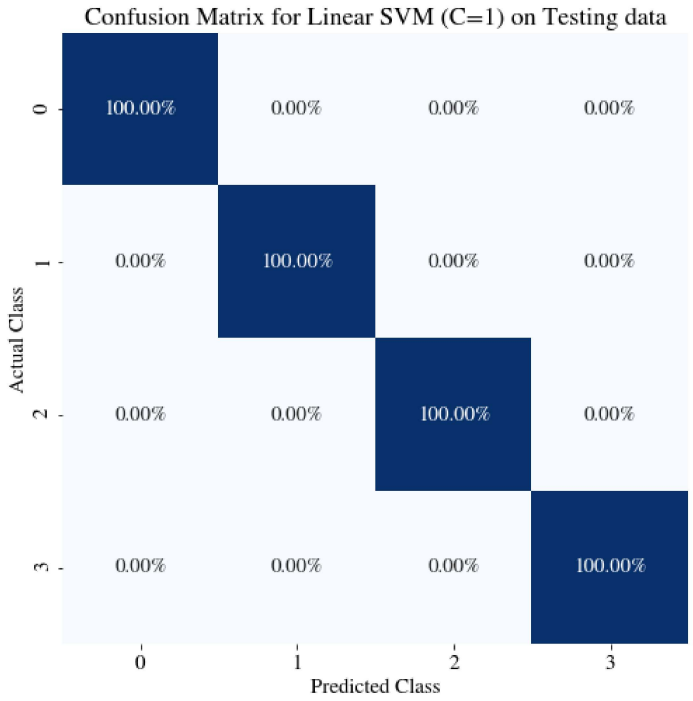
In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn
from sklearn import svm

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

#Taking input from csv file and taking x and y out
data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()


X_train=data[:,0:2]
Y_train=data[:,2]


data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()

X_valid=data[0:60,0:2]
Y_valid=data[0:60,2]

X_test=data[60:120,0:2]
Y_test=data[60:120,2]

C=[0.0001,0.001,0.01,0.1,1,2,4]
#print(X_train.shape)

for c in C:
    clf = svm.SVC(C=c,kernel='linear',decision_function_shape='ovo')
    clf.fit(X_train, Y_train)
    #print(clf.predict(X_valid))
    predicted=clf.predict(X_valid)
    print("accuracy for C="+str(c)+" on validation set is "+str(accuracy_score(Y_valid,predicted)*100))
    predicted=clf.predict(X_train)
    print("accuracy for C="+str(c)+" on training set is "+str(accuracy_score(Y_train,predicted)*100))

clf = svm.SVC(C=1,kernel='linear',decision_function_shape='ovo')
clf.fit(X_train, Y_train)
predicted=clf.predict(X_test)
print("accuracy for C="+str(1)+" on testing set is "+str(accuracy_score(Y_test,predicted)*100))
```

```
accuracy for C=0.0001 on validation set is 100.0
accuracy for C=0.0001 on training set is 100.0
accuracy for C=0.001 on validation set is 100.0
accuracy for C=0.001 on training set is 100.0
accuracy for C=0.01 on validation set is 100.0
accuracy for C=0.01 on training set is 100.0
accuracy for C=0.1 on validation set is 100.0
accuracy for C=0.1 on training set is 100.0
accuracy for C=1 on validation set is 100.0
accuracy for C=1 on training set is 100.0
accuracy for C=2 on validation set is 100.0
accuracy for C=2 on training set is 100.0
accuracy for C=4 on validation set is 100.0
accuracy for C=4 on training set is 100.0
accuracy for C=1 on testing set is 100.0
```

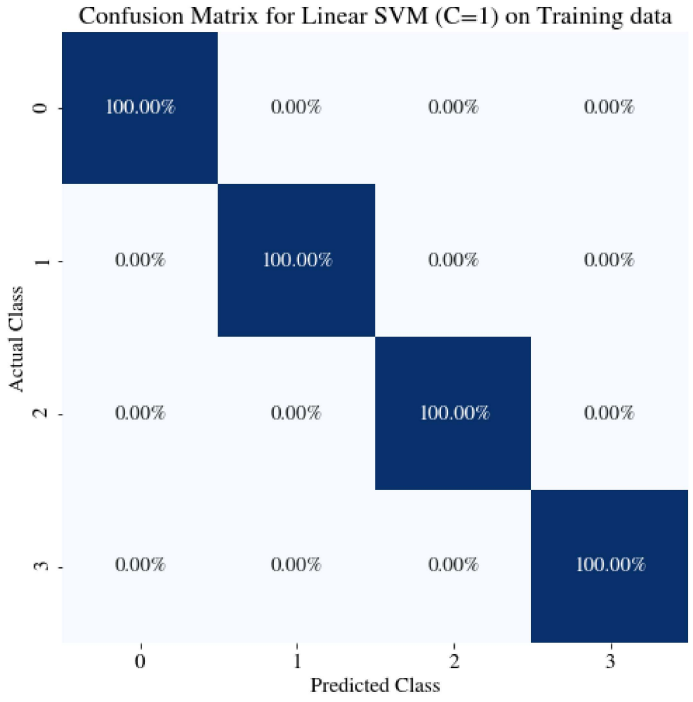
In [2]:

```
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Linear SVM (C=1) on Testing data')
# plt.savefig('SVM_Confusion_test.png')

plt.show()
```



In [3]:

```
predicted=clf.predict(X_train)
confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Linear SVM (C=1) on Training data')
# plt.savefig('SVM_Confusion_train.png')

plt.show()
```



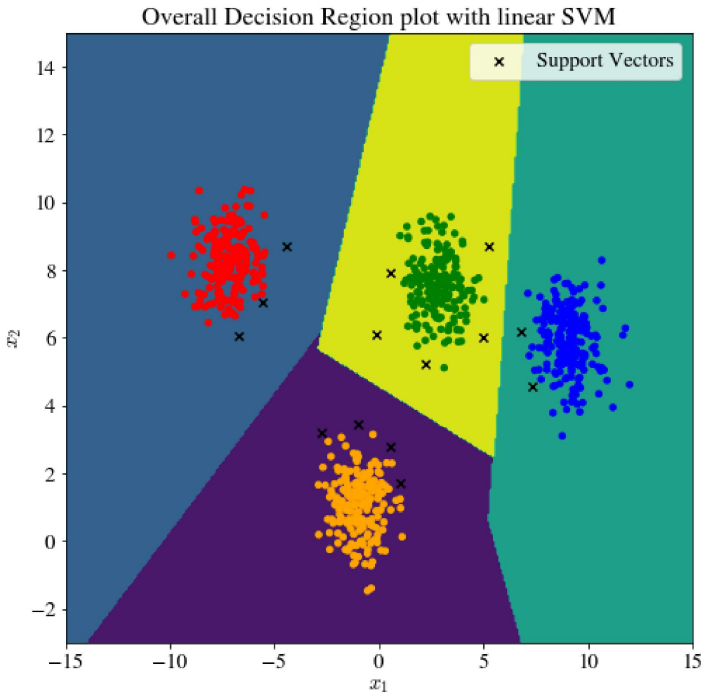
In [4]:

```
#print(clf.support_vectors_)
#print(np.where(X_train == clf.support_vectors_[0]))

#tp=np.delete(X_train[:,0],np.where(X_train[:,0] == clf.support_vectors_[:,0]))
#print(X_train.shape)
#print(X_train[:,0])
tp=np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0])
print(X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])].shape)
```

```
(786, 2)
```

In [5]:

```
x1=np.linspace(-15,15,num=350)
x2=np.linspace(-3,15,num=350)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
#predicted.clear()
num_cores = multiprocessing.cpu_count()

#predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(grid.shape[0]))
predicted=clf.predict(grid)
pos=np.empty(xx1.shape+(2,))
pos[:,:,0]=xx1
pos[:,:,1]=xx2


predicted=np.array(predicted)
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='viridis')
colors = ['orange','red','blue','green']
real=X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])]
plt.scatter(real[:,0], real[:,1], c=Y_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])], cmap=matplotlib.colors.ListedColormap

plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='black')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Overall Decision Region plot with linear SVM')
plt.legend()
# plt.savefig('SVM_Decision_plot.png')
plt.show()
```

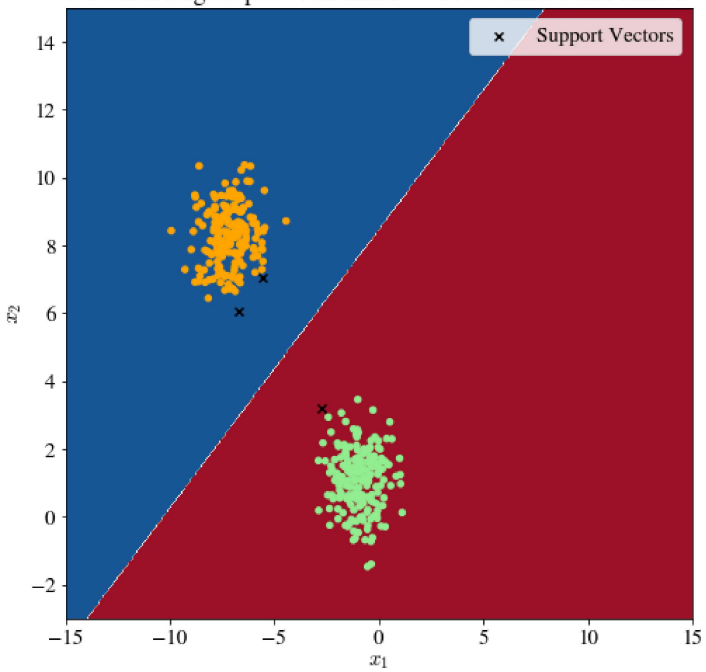


In [6]:

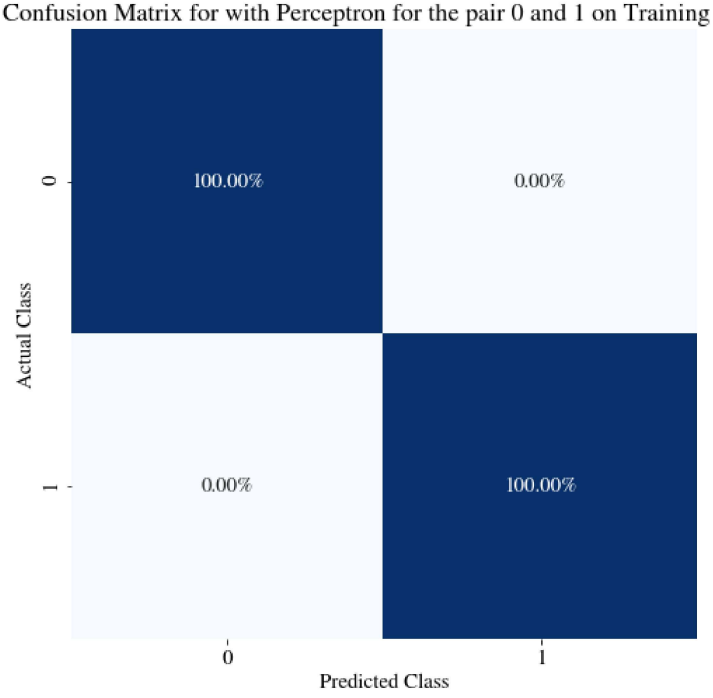
```
(unique, counts) = np.unique(Y_train, return_counts=True)
#print(unique)
for i in range(len(unique)):
    for p in range(i+1,len(unique)):
        X_new_train=X_train[np.where(Y_train==i)]
        X_new_train=np.vstack((X_new_train,X_train[np.where(Y_train==p)]))
        Y_new_train=np.hstack((Y_train[np.where(Y_train==i)],Y_train[np.where(Y_train==p)]))
        clf.fit(X_new_train, Y_new_train)
        x1=np.linspace(-15,15,num=350)
        x2=np.linspace(-3,15,num=350)
        xx1, xx2 = np.meshgrid(x1, x2)
        r1, r2 = xx1.flatten(), xx2.flatten()
        r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
        grid = np.hstack((r1,r2))

        num_cores = multiprocessing.cpu_count()
```

```
        predicted=clf.predict(grid)
        pos=np.empty(xx1.shape+(2,))
        pos[:,:,0]=xx1
        pos[:,:,1]=xx2


        predicted=np.array(predicted)
        predicted=predicted.reshape(xx1.shape)
        fig = plt.figure(figsize=(8,8))
        plt.contourf(xx1, xx2, predicted, cmap='RdBu')
        colors = ['lightgreen','orange']
        real=X_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])]
        plt.scatter(real[:,0], real[:,1], c=Y_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])], cmap=matplotlib.color

        plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='black')
        plt.xlabel('$x_1$')
        plt.ylabel('$x_2$')
        plt.title(f'Decision Region plot with linear SVM for class {i} and class {p}')
        plt.legend()
#         plt.savefig(f'Decision_plot_{i}and{p}.png')
        plt.show()

        predicted=clf.predict(X_new_train)



        confuse=confusion_matrix(Y_new_train,predicted)

        sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
            fmt='.2%', cmap='Blues',cbar=False,xticklabels=[i,p],yticklabels=[i,p])
        plt.xlabel('Predicted Class')
        plt.ylabel("Actual Class")
        plt.title(f'Confusion Matrix for with Perceptron for the pair {i} and {p} on Training  data')
#         plt.savefig(f"Confusion_train_{i}and{p}.png")

        plt.show()

        X_new_test=X_test[np.where(Y_test==i)]
        X_new_test=np.vstack((X_new_test,X_test[np.where(Y_test==p)]))
        #print(X_new_train.shape)
        Y_new_test=np.hstack((Y_test[np.where(Y_test==i)],Y_test[np.where(Y_test==p)]))

        predicted=clf.predict(X_new_test)



        confuse=confusion_matrix(Y_new_test,predicted)

        sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
            fmt='.2%', cmap='Blues',cbar=False,xticklabels=[i,p],yticklabels=[i,p])
        plt.xlabel('Predicted Class')
        plt.ylabel("Actual Class")
        plt.title(f'Confusion Matrix for with Perceptron for the pair {i} and {p} on Testing  data')
#         plt.savefig(f"Confusion_test_{i}and{p}.png")

        plt.show()
```

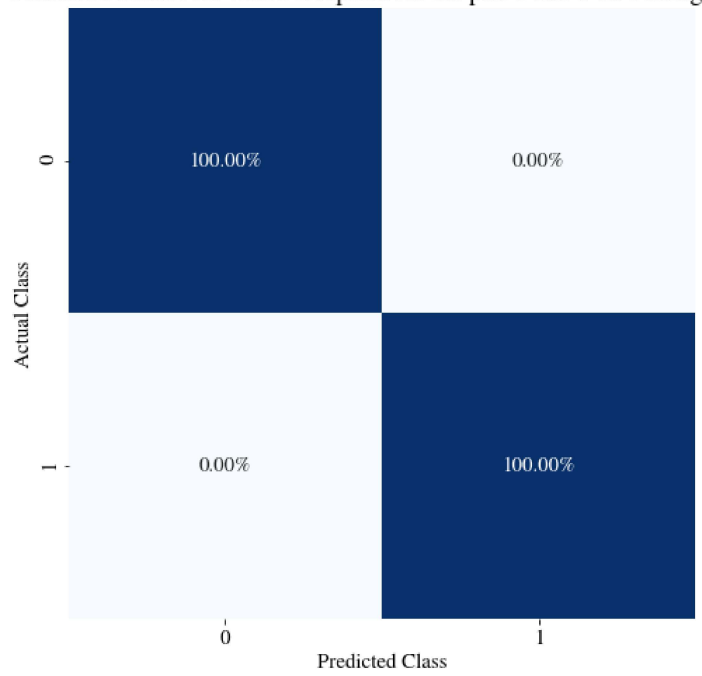
Decision Region plot with linear SVM for class 0 and class 1



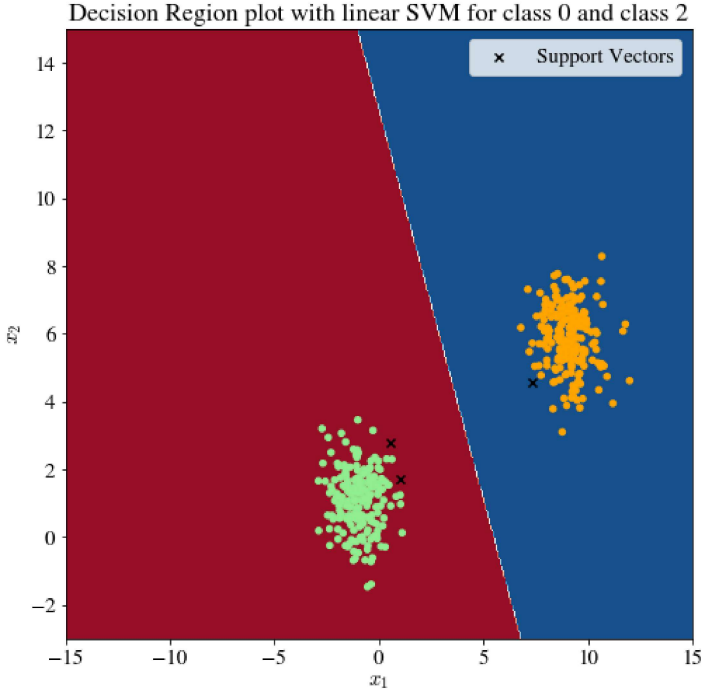
Confusion Matrix for with Perceptron for the pair 0 and 1 on Training data



Confusion Matrix for with Perceptron for the pair 0 and 1 on Testing data



Decision Region plot with linear SVM for class 0 and class 2



Dataset 1(B): MLFFNN with 2 hidden layers

Model specification:

1. 6 nodes each in both hidden layers

2. Early stopping based on validation loss below a certain thershold

3. 112 epochs for convergence observed

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.optimizers import SGD
from keras.layers import Dense

%matplotlib inline


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8.5,8.5)


#Fixing seed value so random numbers generated can be predicted
seed_value = 42
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

In [2]:
```
data=pd.read_csv("19/train.csv",header=None)

Y_train=np.zeros((len(data),3))
data=data.to_numpy()

np.random.shuffle(data)
X_train=data[:,0:2]
labels =data[:,2].astype(int)

for i in range(len(data)):
    Y_train[i,labels[i]]=1
```

In [3]:
```
data=pd.read_csv("19/dev.csv",header=None)

from sklearn.model_selection import train_test_split

data=data.to_numpy()
X=data[:,0:2]
y=data[:,-1].astype(int)


X_valid, X_test, labels_valid, labels_test = train_test_split(X, y,test_size=0.5)
```

```
##Pre-processing normalization of data

mean = np.mean(X_train, axis=0)
X_train -= mean
X_valid -= mean
X_test-=mean

std = np.std(X_train, axis=0)
X_train /= std
X_valid /= std
X_test/=std



Y_valid=np.zeros((len(X_valid),3))
Y_test=np.zeros((len(X_test),3))


for i in range(len(X_valid)):
    Y_valid[i,labels_valid[i]]=1


for i in range(len(X_test)):
    Y_test[i,labels_test[i]]=1
```

In [4]:

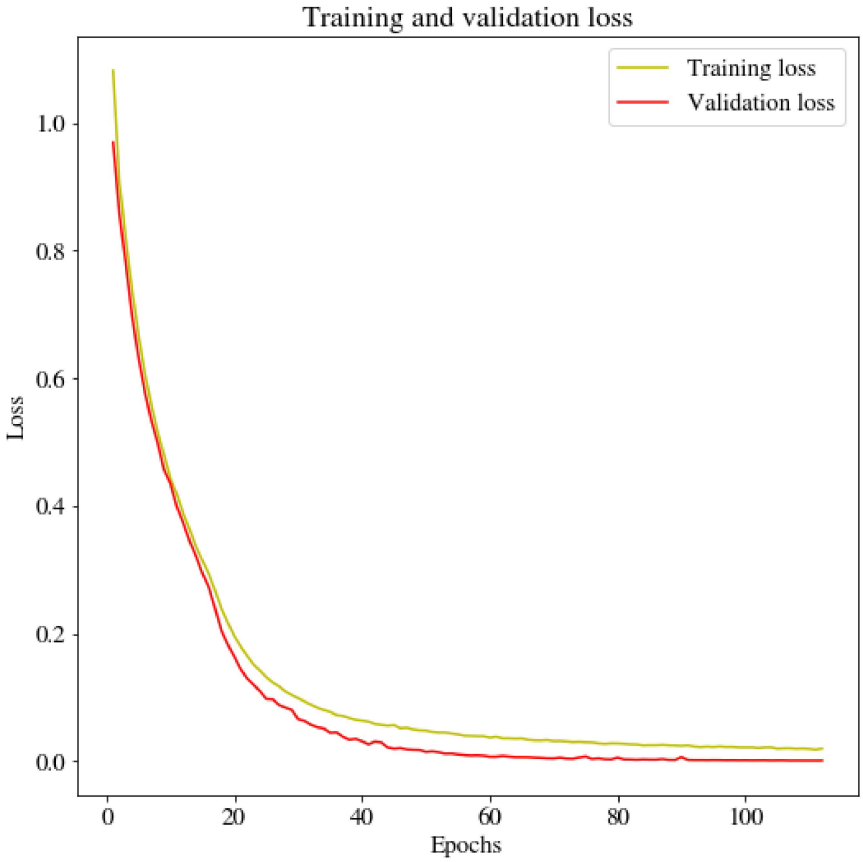
```
# define the architecture of the network

nodes=6

# checkpoint= keras.callbacks.ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1,
es = keras.callbacks.EarlyStopping(
        # Stop training when `val_loss` is no longer improving
        monitor="val_loss",
        # "no longer improving" being defined as "no better than 1e-3 less"
        min_delta=1e-4,
        # "no longer improving" being further defined as "for at least 2 epochs"
        mode='min',
        patience=5,
        verbose=1,
    )
model = Sequential()
model.add(Dense(nodes, input_dim=2, activation='relu'))
model.add(Dense(nodes, activation='relu'))
model.add(Dense(3, activation='softmax'))


# train the model using SGD
print("[INFO] compiling model...")

# compile the keras model


model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])


# fit the keras model on the dataset
history= model.fit(X_train, Y_train,
                        epochs=200,
                        batch_size=1,
                        verbose=1,
                        callbacks=[es],
                        validation_data=(X_valid, Y_valid)
                       )



# evaluate the keras model
loss, accuracy = model.evaluate(X_train, Y_train,batch_size=1,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

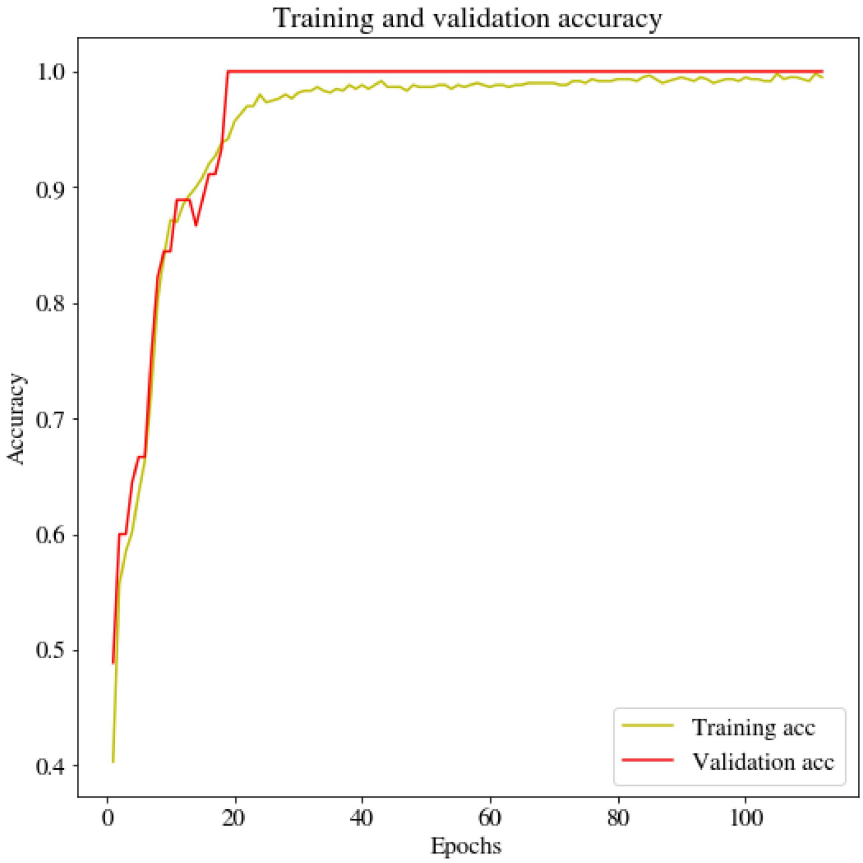
```
[INFO] compiling model...
Epoch 1/200
600/600 [==============================] - 1s 848us/step - loss: 1.0812 - accuracy: 0.4033 - val_loss: 0.9691
- val_accuracy: 0.4889
Epoch 2/200
600/600 [==============================] - 0s 543us/step - loss: 0.9076 - accuracy: 0.5567 - val_loss: 0.8575
- val_accuracy: 0.6000
Epoch 3/200
600/600 [==============================] - 0s 498us/step - loss: 0.8169 - accuracy: 0.5850 - val_loss: 0.7779
- val_accuracy: 0.6000
Epoch 4/200
600/600 [==============================] - 0s 528us/step - loss: 0.7377 - accuracy: 0.6017 - val_loss: 0.6966
- val_accuracy: 0.6444
Epoch 5/200
600/600 [==============================] - 0s 474us/step - loss: 0.6664 - accuracy: 0.6350 - val_loss: 0.6313
- val_accuracy: 0.6667
Epoch 6/200
600/600 [==============================] - 0s 485us/step - loss: 0.6069 - accuracy: 0.6633 - val_loss: 0.5778
- val_accuracy: 0.6667
Epoch 7/200
600/600 [==============================] - 0s 488us/step - loss: 0.5577 - accuracy: 0.7250 - val_loss: 0.5331
- val_accuracy: 0.7556
Epoch 8/200
600/600 [==============================] - 0s 495us/step - loss: 0.5154 - accuracy: 0.8017 - val_loss: 0.4980
- val_accuracy: 0.8222
Epoch 9/200
600/600 [==============================] - 0s 506us/step - loss: 0.4790 - accuracy: 0.8417 - val_loss: 0.4561
- val_accuracy: 0.8444
Epoch 10/200
600/600 [==============================] - 0s 496us/step - loss: 0.4426 - accuracy: 0.8717 - val_loss: 0.4343
- val_accuracy: 0.8444
Epoch 11/200
600/600 [==============================] - 0s 509us/step - loss: 0.4180 - accuracy: 0.8700 - val_loss: 0.3982
- val_accuracy: 0.8889
Epoch 12/200
600/600 [==============================] - 0s 496us/step - loss: 0.3879 - accuracy: 0.8850 - val_loss: 0.3729
- val_accuracy: 0.8889
Epoch 13/200
600/600 [==============================] - 0s 518us/step - loss: 0.3628 - accuracy: 0.8933 - val_loss: 0.3445
- val_accuracy: 0.8889
Epoch 14/200
600/600 [==============================] - 0s 494us/step - loss: 0.3359 - accuracy: 0.9000 - val_loss: 0.3209
- val_accuracy: 0.8667
Epoch 15/200
600/600 [==============================] - 0s 507us/step - loss: 0.3144 - accuracy: 0.9083 - val_loss: 0.2942
- val_accuracy: 0.8889
Epoch 16/200
600/600 [==============================] - 0s 501us/step - loss: 0.2933 - accuracy: 0.9200 - val_loss: 0.2723
- val_accuracy: 0.9111
Epoch 17/200
600/600 [==============================] - 0s 482us/step - loss: 0.2667 - accuracy: 0.9267 - val_loss: 0.2384
- val_accuracy: 0.9111
Epoch 18/200
600/600 [==============================] - 0s 508us/step - loss: 0.2383 - accuracy: 0.9383 - val_loss: 0.2037
- val_accuracy: 0.9333
Epoch 19/200
600/600 [==============================] - 0s 505us/step - loss: 0.2164 - accuracy: 0.9417 - val_loss: 0.1822
- val_accuracy: 1.0000
Epoch 20/200
600/600 [==============================] - 0s 501us/step - loss: 0.1958 - accuracy: 0.9567 - val_loss: 0.1636
- val_accuracy: 1.0000
Epoch 21/200
600/600 [==============================] - 0s 502us/step - loss: 0.1799 - accuracy: 0.9633 - val_loss: 0.1441
- val_accuracy: 1.0000
Epoch 22/200
600/600 [==============================] - 0s 530us/step - loss: 0.1648 - accuracy: 0.9700 - val_loss: 0.1295
- val_accuracy: 1.0000
Epoch 23/200
600/600 [==============================] - 0s 529us/step - loss: 0.1518 - accuracy: 0.9700 - val_loss: 0.1199
- val_accuracy: 1.0000
Epoch 24/200
600/600 [==============================] - 0s 570us/step - loss: 0.1418 - accuracy: 0.9800 - val_loss: 0.1098
- val_accuracy: 1.0000
Epoch 25/200
600/600 [==============================] - 0s 621us/step - loss: 0.1319 - accuracy: 0.9733 - val_loss: 0.0976
- val_accuracy: 1.0000
Epoch 26/200
600/600 [==============================] - 0s 559us/step - loss: 0.1232 - accuracy: 0.9750 - val_loss: 0.0970
- val_accuracy: 1.0000
```

```
Epoch 106/200
600/600 [==============================] - 0s 603us/step - loss: 0.0201 - accuracy: 0.9933 - val_loss: 0.0013
- val_accuracy: 1.0000
Epoch 107/200
600/600 [==============================] - 0s 626us/step - loss: 0.0203 - accuracy: 0.9950 - val_loss: 0.0010
- val_accuracy: 1.0000
Epoch 108/200
600/600 [==============================] - 0s 624us/step - loss: 0.0196 - accuracy: 0.9950 - val_loss: 0.0011
- val_accuracy: 1.0000
Epoch 109/200
600/600 [==============================] - 0s 601us/step - loss: 0.0199 - accuracy: 0.9933 - val_loss: 0.0010
- val_accuracy: 1.0000
Epoch 110/200
600/600 [==============================] - 0s 604us/step - loss: 0.0193 - accuracy: 0.9917 - val_loss: 0.0010
- val_accuracy: 1.0000
Epoch 111/200
600/600 [==============================] - 0s 612us/step - loss: 0.0181 - accuracy: 0.9983 - val_loss: 9.4191e
-04 - val_accuracy: 1.0000
Epoch 112/200
600/600 [==============================] - 0s 621us/step - loss: 0.0198 - accuracy: 0.9950 - val_loss: 0.0010
- val_accuracy: 1.0000
Epoch 00112: early stopping
600/600 [==============================] - 0s 433us/step - loss: 0.0170 - accuracy: 0.9983

[INFO] loss=0.0170, accuracy: 99.8333%
```

In [5]:

```python
#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


acc = history.history['accuracy']  #Use accuracy if acc doesn't work
val_acc = history.history['val_accuracy']  #Use val_accuracy if acc doesn't work
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





In [6]:

```
loss, accuracy = model.evaluate(X_train, Y_train,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
19/19 [==============================] - 0s 731us/step - loss: 0.0170 - accuracy: 0.9983
```

```
[INFO] loss=0.0170, accuracy: 99.8333%
```

In [7]:
```
loss, accuracy = model.evaluate(X_valid, Y_valid,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

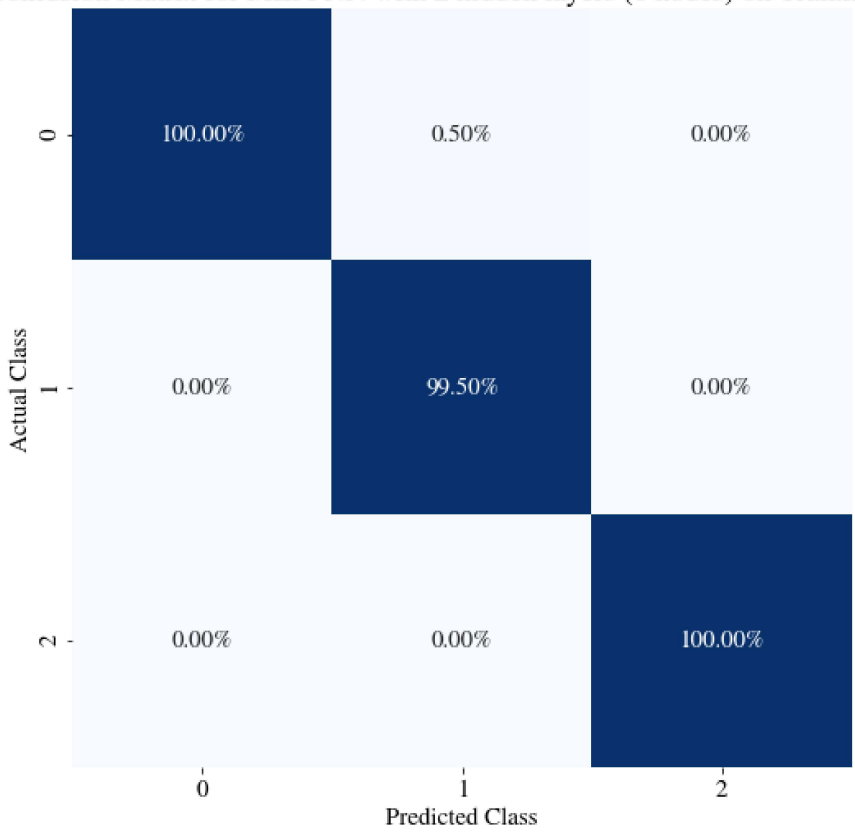
```
2/2 [==============================] - 0s 999us/step - loss: 0.0010 - accuracy: 1.0000


 [INFO] loss=0.0010, accuracy: 100.0000%
```

In [8]:
```
loss, accuracy = model.evaluate(X_test, Y_test,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

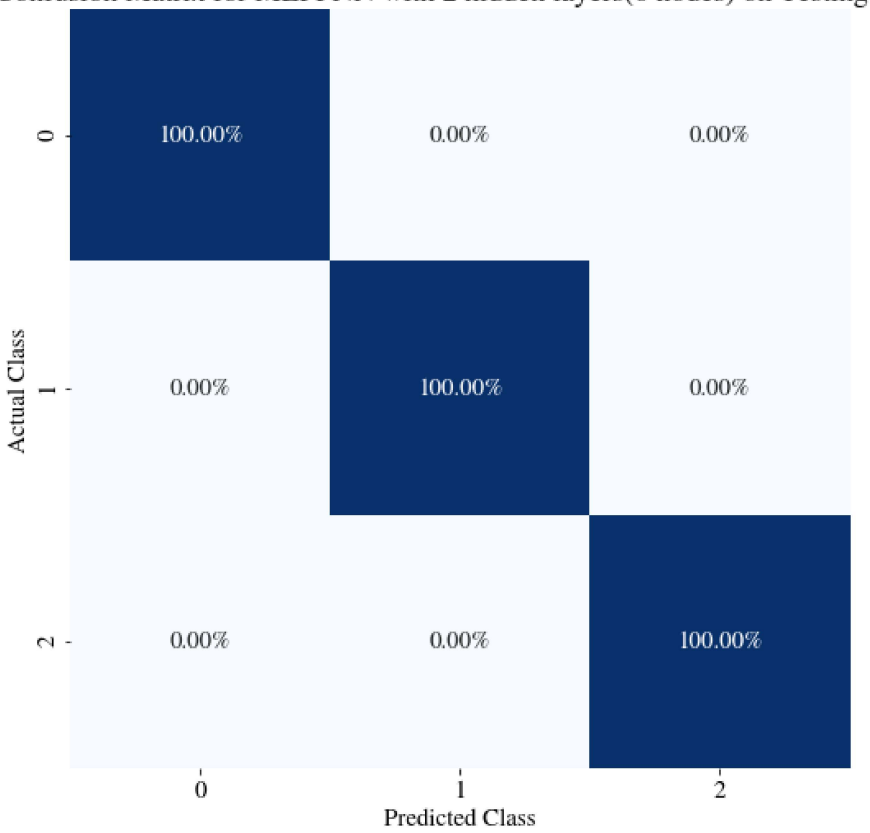
```
1/2 [==============>..............] - ETA: 0s - loss: 0.0096 - accuracy: 1.0000WARNING:tensorflow:Callbacks m
ethod `on_test_batch_end` is slow compared to the batch time (batch time: 0.0000s vs `on_test_batch_end` time:
0.0010s). Check your callbacks.
2/2 [==============================] - 0s 998us/step - loss: 0.0222 - accuracy: 1.0000


 [INFO] loss=0.0222, accuracy: 100.0000%
```

In [9]:
```
predicted_train =np.argmax(model.predict(X_train), axis=-1)
confuse=confusion_matrix(labels,predicted_train)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFFNN with 2 hidden layers (6 nodes) on Training data')
plt.savefig("MLFNN_Confusion_train.png")

plt.show()

predicted_test =np.argmax(model.predict(X_test), axis=-1)
confuse=confusion_matrix(labels_test,predicted_test)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFFNN with 2 hidden layers(6 nodes) on Testing data')
plt.savefig("MLFNN_Confusion_test.png")

plt.show()
```

Confusion Matrix for MLFFNN with 2 hidden layers (6 nodes) on Training data



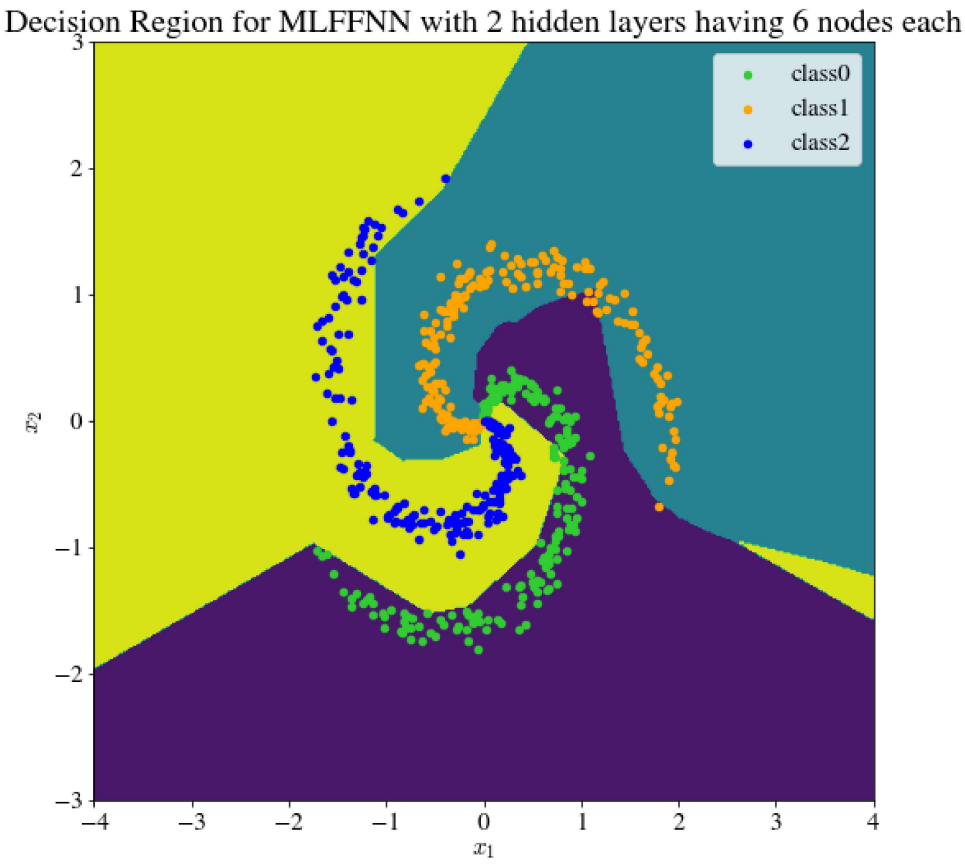
Confusion Matrix for MLFFNN with 2 hidden layers(6 nodes) on Testing data



In [13]:

```
x1=np.linspace(-4,4,num=400)
x2=np.linspace(-3,3,num=400)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
```

```
predicted=np.argmax(model.predict(grid),axis=1)


# predicted=predicted.reshape(xx1.shape)
# fig = plt.figure(figsize=(8,8))
# plt.contourf(xx1, xx2, predicted, cmap='RdBu')
# colors = ['red','red','red']
# plt.scatter(X_train[:,0], X_train[:,1], c= Y_train, cmap=matplotlib.colors.ListedColormap(colors))

num_cores = multiprocessing.cpu_count()
# plt.xlabel('$x_1$')
# plt.ylabel('$x_2$')
# plt.title('Decision Region plot for MLFNN having 2 hidden layers',fontsize=20)

predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8.5,8.5))
plt.contourf(xx1, xx2, predicted, cmap='viridis')
# colors = ['red','red','red']
# plt.scatter(X_train[:,0], X_train[:,1], c= Y_train, cmap=matplotlib.colors.ListedColormap(colors))


data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==0], [data[m][1] for m in range(len(data))
                                    if data[m][2]==0], c ="limegreen",label="class0",s=20)
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==1], [data[m][1] for m in range(len(data))
                                                                      if data[m][2]==1], c ="orange",label="cl
plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==2], [data[m][1] for m in range(len(data))
                                                                      if data[m][2]==2], c ="blue",label="clas


plt.legend()

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region for MLFFNN with 2 hidden layers having 6 nodes each',fontsize=20)
plt.savefig('MLFNN_Decision.png')
plt.show()
plt.close()
```



In [14]:

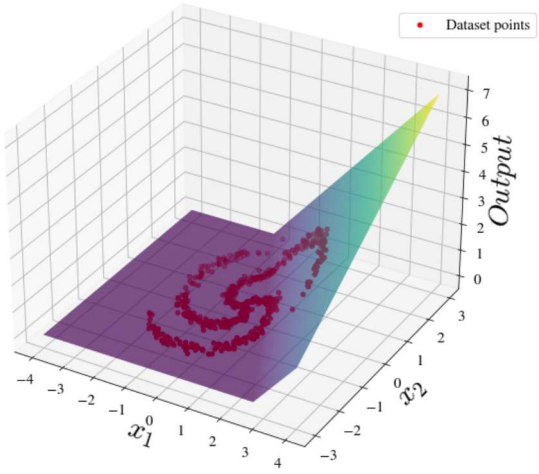
```
import keras.backend as K


outputs_train = []
for layer in model.layers:
    keras_function = K.function([model.input], [layer.output])
    outputs_train.append(keras_function([X_train, 1]))


outputs = []
for layer in model.layers:
    keras_function = K.function([model.input], [layer.output])
    outputs.append(keras_function([grid, 1]))
```

In [15]:

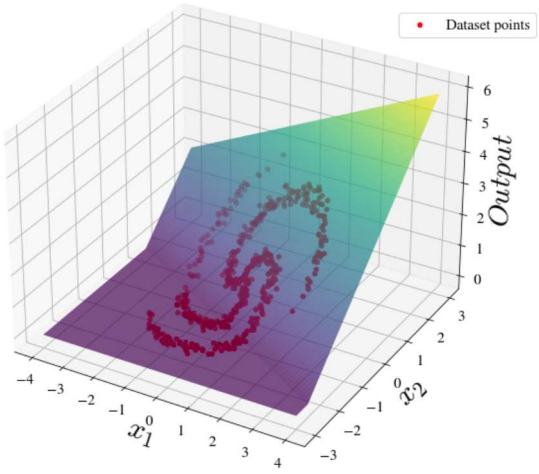
```
for layer in range(2):
    fig = plt.figure(figsize=(20,30))
    for i in range(nodes):
        ax = fig.add_subplot(3, 2 , i+1, projection='3d')
        temp=outputs[layer][0][:,i]
        out=temp.reshape(xx1.shape)
        ax.plot_surface(xx1, xx2, out,rstride=1, cstride=1,
                        cmap='viridis', edgecolor='none')
        ax.scatter3D(X_train[:,0],X_train[:,1], outputs_train[layer][0][:,i], color = "red",label="Dataset poi
        ax.set_title(f" Node {i +1}", fontsize=30)
        ax.set_xlabel(r"$x_1$",fontsize=30)
        ax.set_ylabel(r"$x_2$", fontsize=30)
        ax.set_zlabel(r"$Output$" ,fontsize=30)
        ax.legend(bbox_to_anchor=(1.1, 0.92), bbox_transform=ax.transAxes)
    fig.suptitle(f'Outputs of nodes in Hidden layer {layer +1}(After convergence 112 epochs)', fontsize=45)
    plt.tight_layout()
    plt.subplots_adjust(top=0.92)
    plt.savefig(f'hidden_layer_{layer + 1}.png')
```

Outputs of nodes in Hidden layer 1(After convergence 112 epochs)

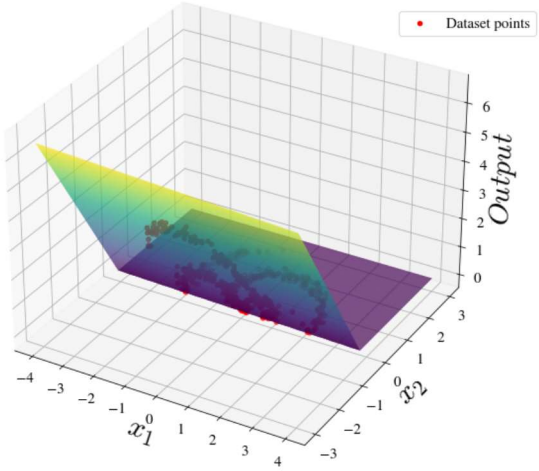
Node 1



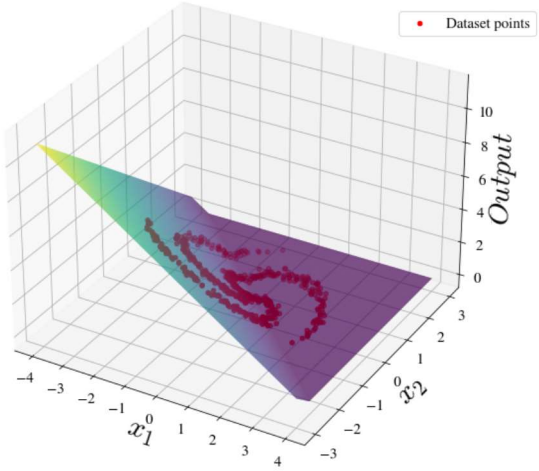
Node 2



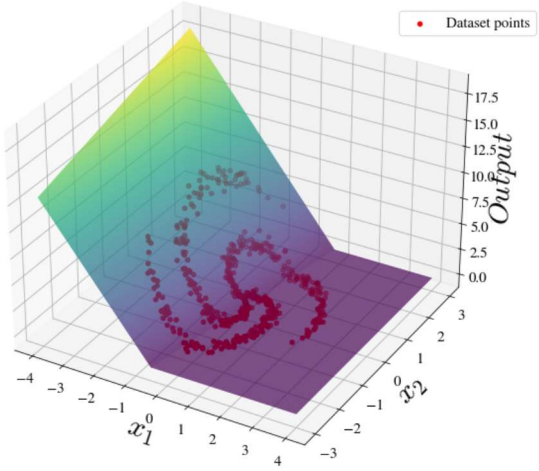
Node 3



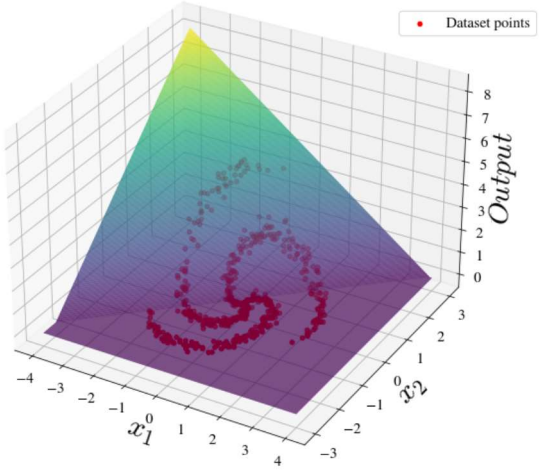
Node 4



Node 5

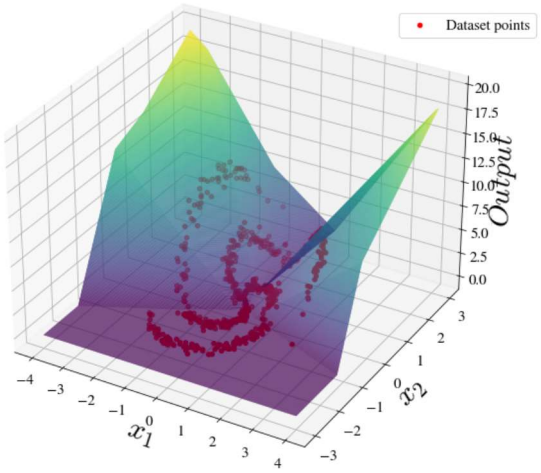


Node 6

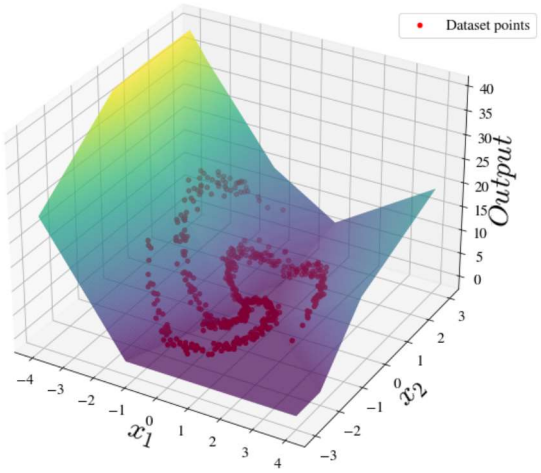


Outputs of nodes in Hidden layer 2(After convergence 112 epochs)

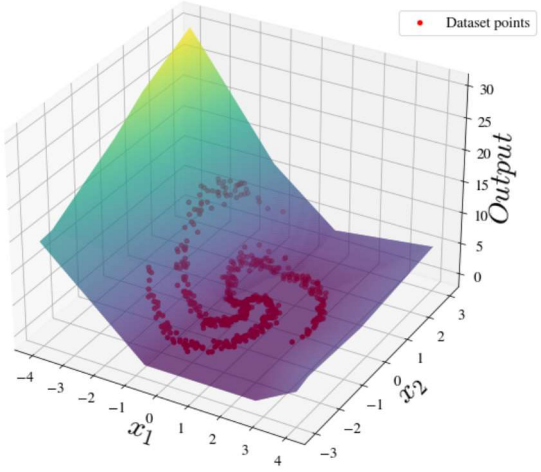
Node 1



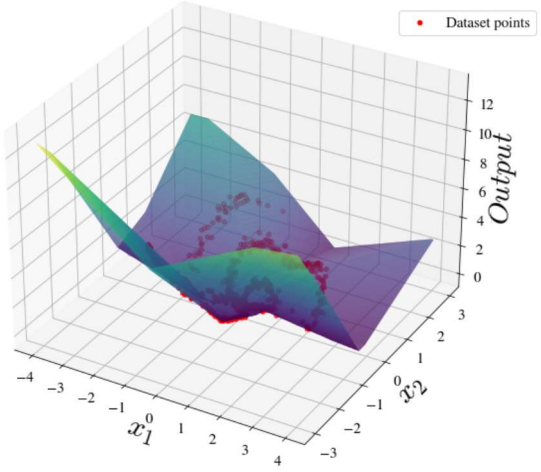
Node 2



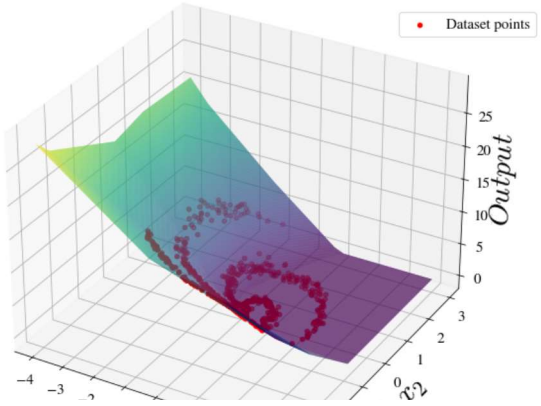
Node 3



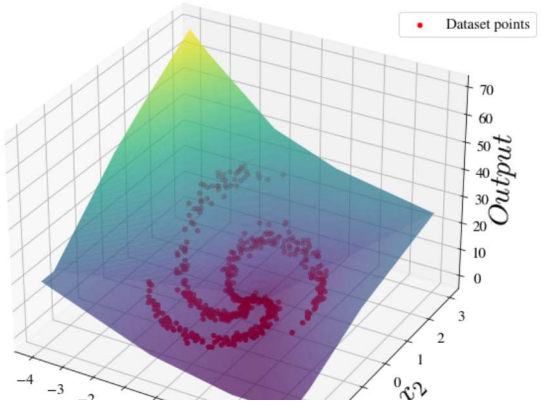
Node 4

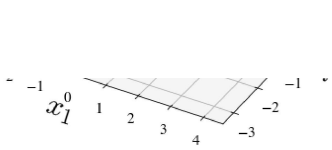


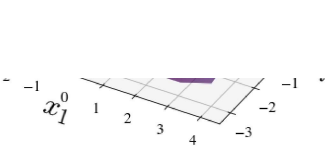
Node 5



Node 6

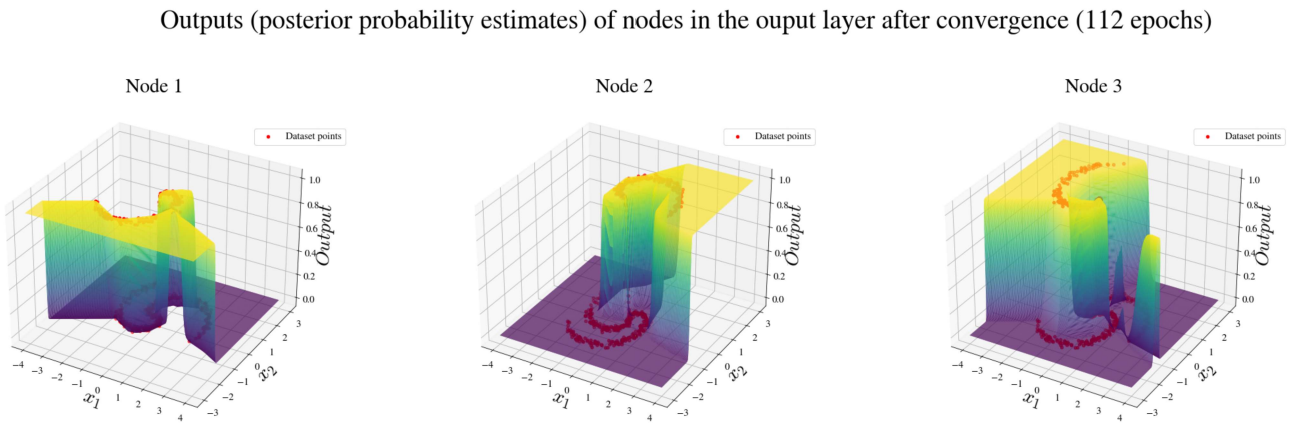






In [16]:

```
fig = plt.figure(figsize=(30,10))

for i in range(3):
    ax = fig.add_subplot(1, 3 , i+1, projection='3d')
    temp=outputs[2][0][:,i]
    out=temp.reshape(xx1.shape)
    ax.plot_surface(xx1, xx2, out,rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
    ax.scatter3D(X_train[:,0],X_train[:,1], outputs_train[2][0][:,i], color = "red",label="Dataset points")
    ax.set_title(f" Node {i +1}", fontsize=30)
    ax.set_xlabel(r"$x_1$",fontsize=30)
    ax.set_ylabel(r"$x_2$", fontsize=30)
    ax.set_zlabel(r"$Output$" ,fontsize=30)
    ax.legend(bbox_to_anchor=(1.1, 0.92), bbox_transform=ax.transAxes)
fig.suptitle(f'Outputs (posterior probability estimates) of nodes in the ouput layer after convergence (112 ep

plt.tight_layout()
plt.subplots_adjust(top=0.87)
plt.savefig('output_layer.png')
plt.show()
```



Dataset1B: Nonlinear SVM using one-against-the-rest approach : (a) Polynomial kernel

In [1]:

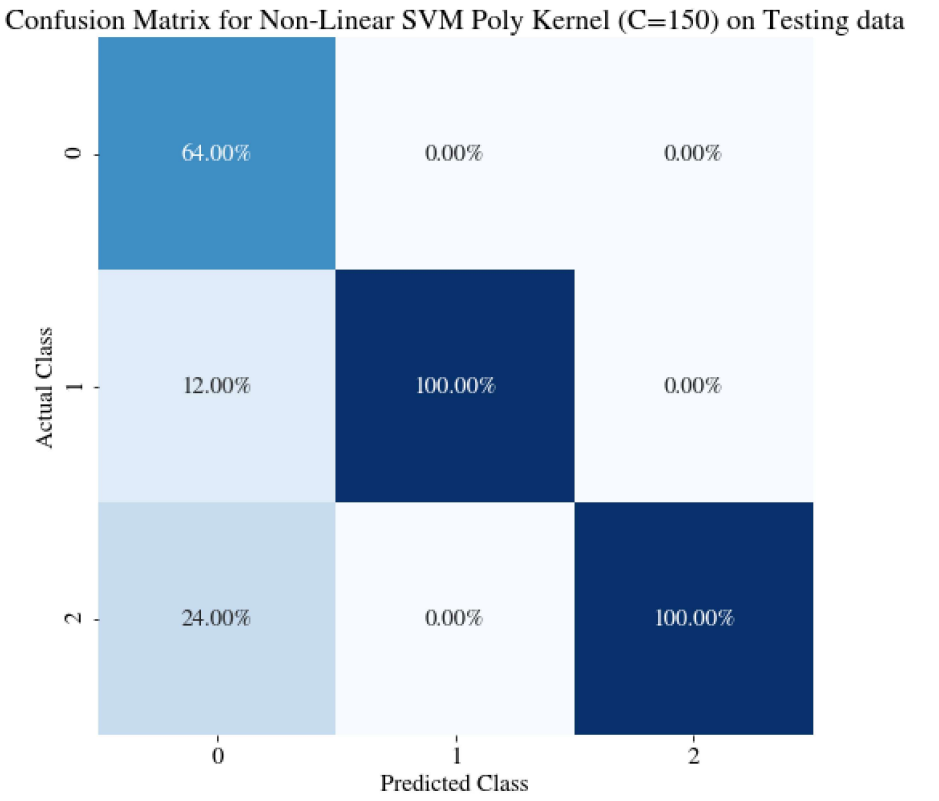
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn
from sklearn import svm


%matplotlib inline


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

np.random.seed(81)

data=pd.read_csv("19/train.csv",header=None)
data=data.to_numpy()

np.random.shuffle(data)
X_train=data[:,0:2]
Y_train =data[:,2].astype(int)

from sklearn.model_selection import train_test_split
data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()
X=data[:,0:2]
y=data[:,-1].astype(int)


X_valid, X_test, Y_valid, Y_test = train_test_split(X, y,test_size=0.5)

# data=pd.read_csv("19/dev.csv",header=None)
# #data.columns =['x1', 'x2','Class']
# X_valid=data.loc[np.r_[0:15, 30:45, 60:75],:]
# X_valid=X_valid.to_numpy()
# Y_valid=X_valid[:,-1]
# X_valid=X_valid[:,:-1]
# data=pd.read_csv("19/dev.csv",header=None)
# X_test=data.loc[np.r_[15:30, 45:60, 75:90],:]
# X_test=X_test.to_numpy()
# Y_test=X_test[:,-1]
# X_test=X_test[:,:-1]


C=[0.0001,0.001,0.01,0.1,4,16,100,150,500,1e3]
#print(X_train.shape)

for c in C:
    clf = svm.SVC(C=c,kernel='poly',degree=3,decision_function_shape='ovr')
    clf.fit(X_train, Y_train)
    #print(clf.predict(X_valid))
    predicted=clf.predict(X_valid)
    print("accuracy for C="+str(c)+" on validation set is "+str(accuracy_score(Y_valid,predicted)*100))
    predicted=clf.predict(X_train)
    print("accuracy for C="+str(c)+" on training set is "+str(accuracy_score(Y_train,predicted)*100))

clf = svm.SVC(C=150,kernel='poly',degree=3,decision_function_shape='ovr')
clf.fit(X_train, Y_train)
predicted=clf.predict(X_test)
print("accuracy for C="+str(150)+" (Best Model) on testing set is "+str(accuracy_score(Y_test,predicted)*100))
```

```
accuracy for C=0.0001 on validation set is 51.11111111111111
```

```
accuracy for C=0.0001 on training set is 51.0
accuracy for C=0.001 on validation set is 51.11111111111111
accuracy for C=0.001 on training set is 51.83333333333333
accuracy for C=0.01 on validation set is 60.0
accuracy for C=0.01 on training set is 59.166666666666664
accuracy for C=0.1 on validation set is 62.22222222222222
accuracy for C=0.1 on training set is 71.5
accuracy for C=4 on validation set is 71.11111111111111
accuracy for C=4 on training set is 73.33333333333333
accuracy for C=16 on validation set is 73.33333333333333
accuracy for C=16 on training set is 73.5
accuracy for C=100 on validation set is 77.77777777777779
accuracy for C=100 on training set is 74.83333333333333
accuracy for C=150 on validation set is 77.77777777777779
accuracy for C=150 on training set is 75.5
accuracy for C=500 on validation set is 77.77777777777779
accuracy for C=500 on training set is 75.5
accuracy for C=1000.0 on validation set is 77.77777777777779
accuracy for C=1000.0 on training set is 75.5
accuracy for C=150 (Best Model) on testing set is 80.0
```

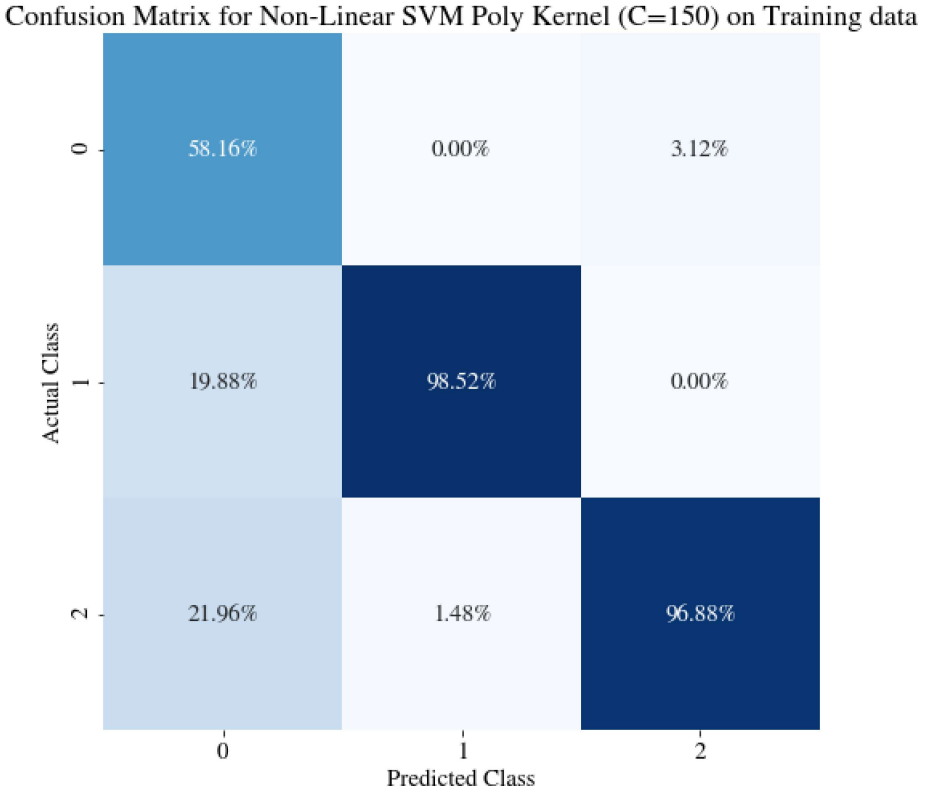
In [2]:

```
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for Non-Linear SVM Poly Kernel (C={150}) on Testing data')
# plt.savefig('poly_Confusion_test.png')

plt.show()
```



In [3]:

```
predicted=clf.predict(X_train)
confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for Non-Linear SVM Poly Kernel (C=150) on Training data')
# plt.savefig('poly_Confusion_train.png')

plt.show()
```



In [4]:

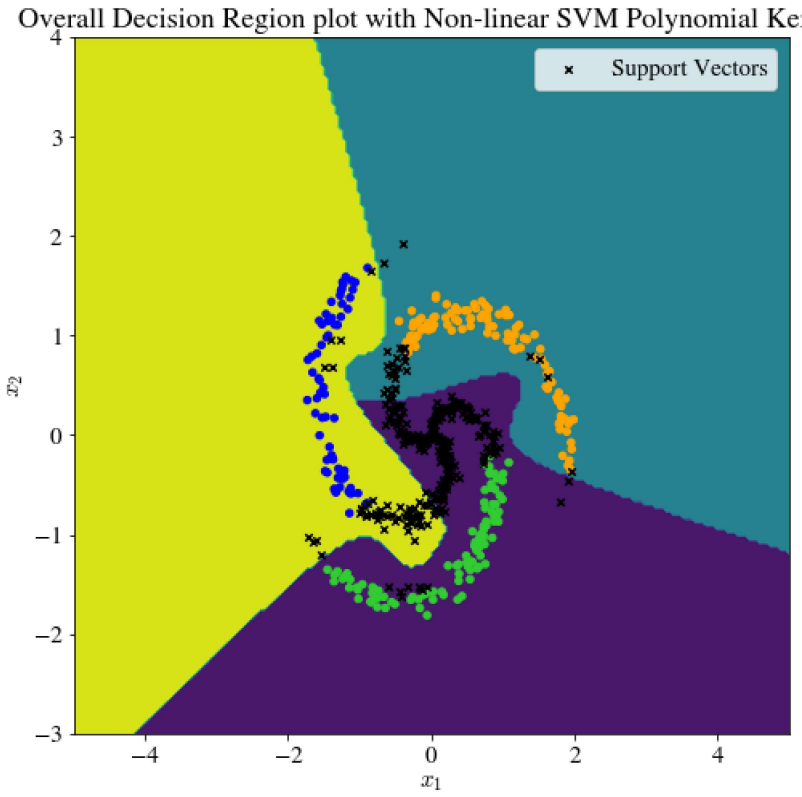
```
#print(clf.support_vectors_)
#print(np.where(X_train == clf.support_vectors_[0]))

#tp=np.delete(X_train[:,0],np.where(X_train[:,0] == clf.support_vectors_[:,0]))
#print(X_train.shape)
#print(X_train[:,0])
tp=np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0])
print(X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])].shape)
```

```
(276, 2)
```

In [5]:

```
x1=np.linspace(-5,5,num=200)
x2=np.linspace(-3,4,num=200)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
#predicted.clear()
num_cores = multiprocessing.cpu_count()

#predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(grid.
predicted=clf.predict(grid)
pos=np.empty(xx1.shape+(2,))
pos[:,:,0]=xx1
pos[:,:,1]=xx2


predicted=np.array(predicted)
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='viridis')
colors = ['limegreen','orange','blue']
real=X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])]


# data=pd.read_csv("19/train.csv",header=None)

# data=data.to_numpy()
# plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==0], [data[m][1] for m in range(len(data))
#                                         if data[m][2]==0], c ="limegreen",label="class0",s=20)
# plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==1], [data[m][1] for m in range(len(data))
```

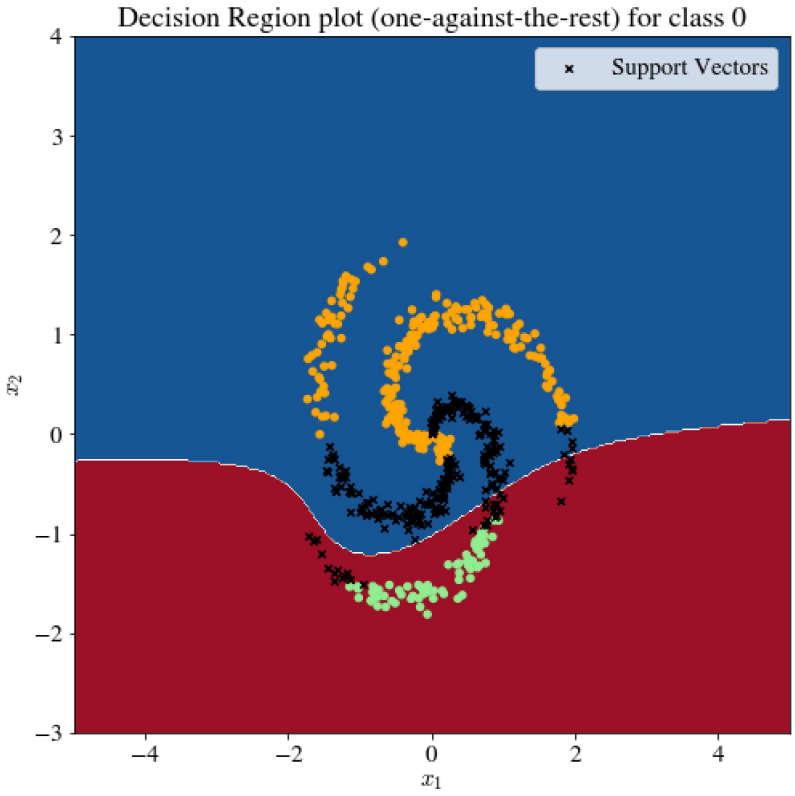
```
#                                                                 if data[m][2]==1], c ="orange",label="cla
# plt.scatter([data[m][0] for m in range(len(data)) if data[m][2]==2], [data[m][1] for m in range(len(data))
#                                                                 if data[m][2]==2], c ="blue",label="class

plt.scatter(real[:,0], real[:,1], c=Y_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])], cmap=matplotlib.co

plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='black'
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Overall Decision Region plot with Non-linear SVM Polynomial Kernel')
plt.legend()
# plt.savefig('Overall_Decision_plot.png')
plt.show()
```

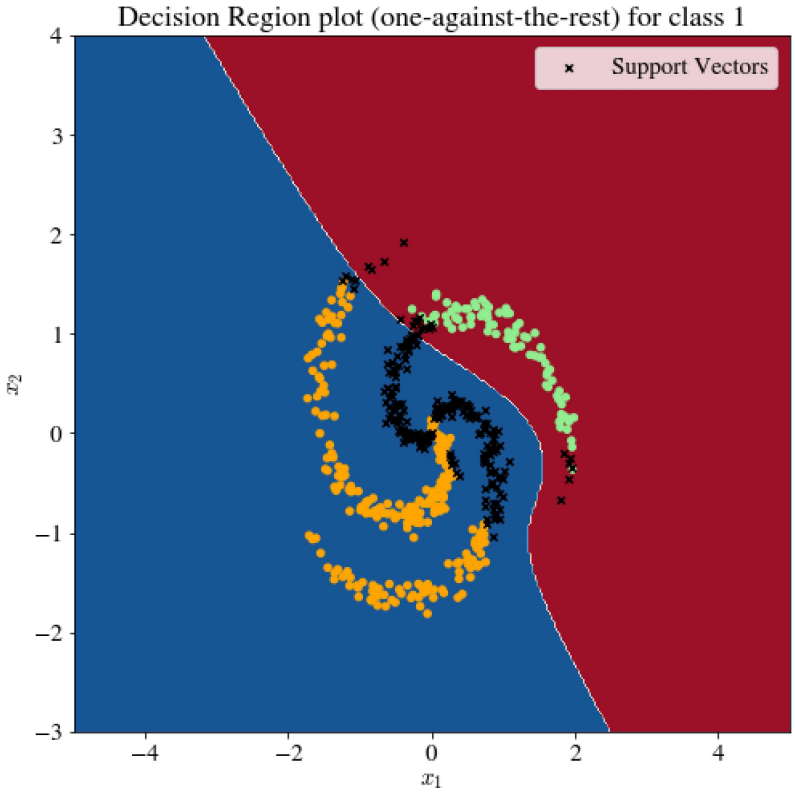


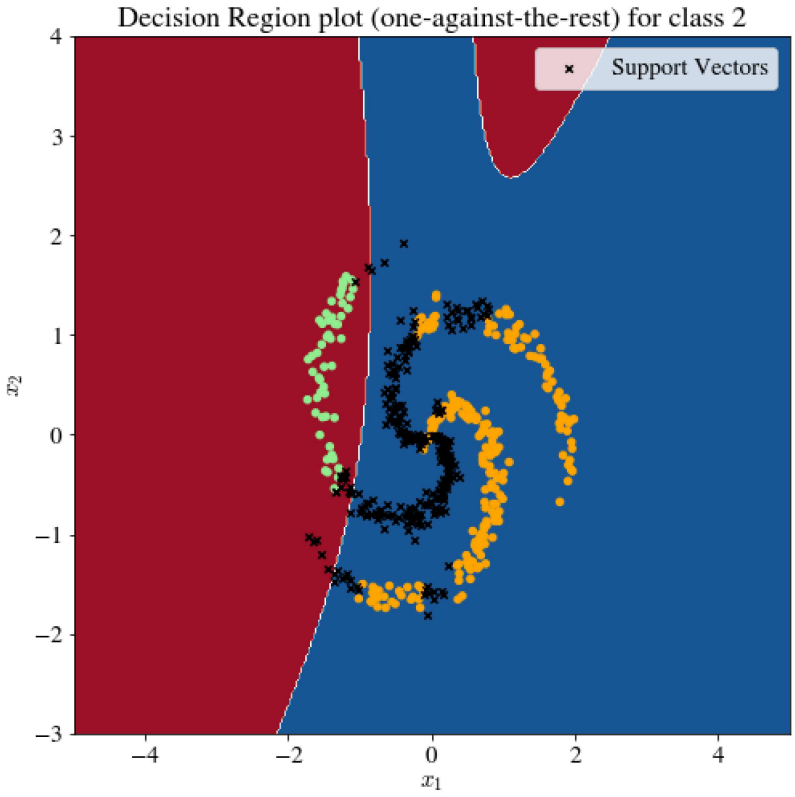
In [6]:

```
(unique, counts) = np.unique(Y_train, return_counts=True)
#print(unique)
for i in range(len(unique)):

    X_new_train=X_train[np.where(Y_train==i)]
    X_new_train=np.vstack((X_new_train,X_train[np.where(Y_train!=i)]))
    #print(X_new_train.shape)
    #Y_new_train=np.hstack((Y_train[np.where(Y_train==i)],Y_train[np.where(Y_train==p)]))
    Y_new_train=np.zeros(Y_train[np.where(Y_train==i)].shape)
    Y_new_train=np.hstack((Y_new_train,np.ones(Y_train[np.where(Y_train!=i)].shape)))

    #print(Y_new_train)
    clf.fit(X_new_train, Y_new_train)
    x1=np.linspace(-5,5,num=300)
    x2=np.linspace(-3,4,num=300)
    xx1, xx2 = np.meshgrid(x1, x2)
    r1, r2 = xx1.flatten(), xx2.flatten()
    r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
    grid = np.hstack((r1,r2))
    #print(grid)
    #predicted.clear()
    num_cores = multiprocessing.cpu_count()

    #predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(g
    predicted=clf.predict(grid)
    pos=np.empty(xx1.shape+(2,))
    pos[:,:,0]=xx1
    pos[:,:,1]=xx2
```

```
    predicted=np.array(predicted)
    predicted=predicted.reshape(xx1.shape)
    fig = plt.figure(figsize=(8,8))
    plt.contourf(xx1, xx2, predicted, cmap='RdBu')
    colors = ['lightgreen','orange']
    real=X_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])]
    plt.scatter(real[:,0], real[:,1], c=Y_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])], cmap=m
    #plt.scatter(np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0]), np.setdiff1d(X_train[:,1],clf.support_vect
    #plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')
    plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='bl
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title(f'Decision Region plot (one-against-the-rest) for class {i}')
    plt.legend()
#     plt.savefig(f'Decision_plot_class{i}.png')
    plt.show()
```




Decision Region plot (one-against-the-rest) for class 1
Support Vectors
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
x_1
x_2


Decision Region plot (one-against-the-rest) for class 2
Support Vectors
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
x_1
x_2

Dataset 1B Nonlinear SVM using one-against-the-rest approach : (b) Gaussian Kernel

In [1]:

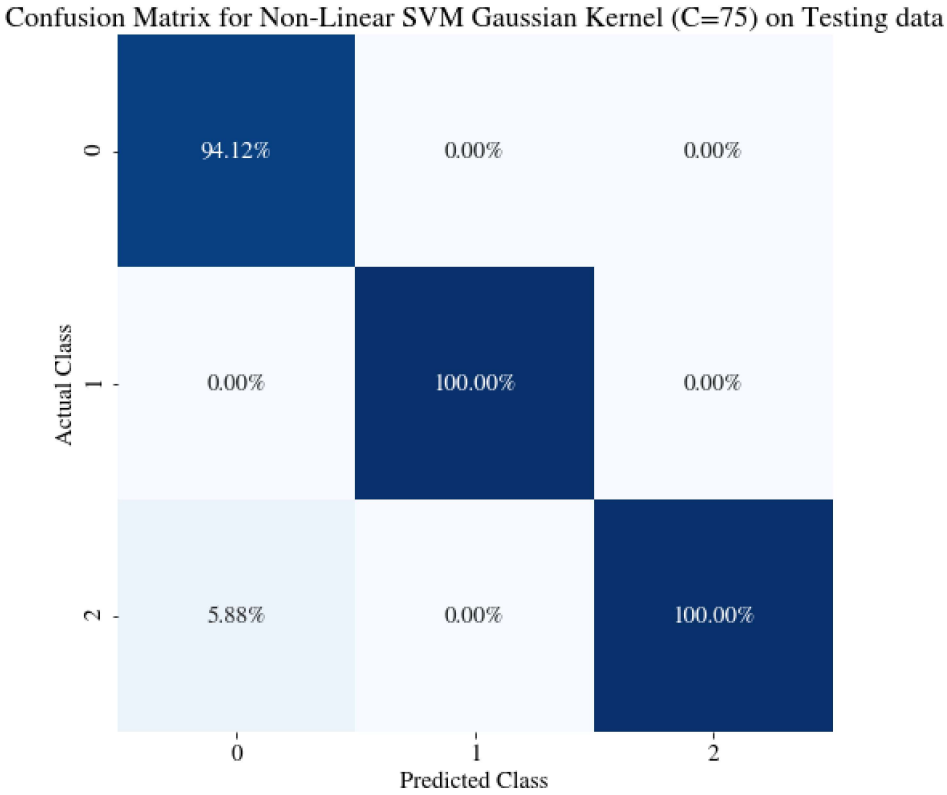
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn
from sklearn import svm


%matplotlib inline


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

np.random.seed(81)

data=pd.read_csv("19/train.csv",header=None)
data=data.to_numpy()

np.random.shuffle(data)
X_train=data[:,0:2]
Y_train =data[:,2].astype(int)

from sklearn.model_selection import train_test_split
data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()
X=data[:,0:2]
y=data[:,-1].astype(int)


X_valid, X_test, Y_valid, Y_test = train_test_split(X, y,test_size=0.5)
# #Taking input from csv file and taking x and y out
# data=pd.read_csv("19/train.csv",header=None)
# data=data.to_numpy()
# X_train=data[:,0:2]
# Y_train=data[:,2]
# data=pd.read_csv("19/dev.csv",header=None)
# #data.columns =['x1', 'x2','Class']
# X_valid=data.loc[np.r_[0:15, 30:45, 60:75],:]
# X_valid=X_valid.to_numpy()
# Y_valid=X_valid[:,-1]
# X_valid=X_valid[:,:-1]
# data=pd.read_csv("19/dev.csv",header=None)
# X_test=data.loc[np.r_[15:30, 45:60, 75:90],:]
# X_test=X_test.to_numpy()
# Y_test=X_test[:,-1]
# X_test=X_test[:,:-1]


C=[0.0001,0.001,0.01,0.1,4,16,50,75,100,150,200,500]
#print(X_train.shape)

for c in C:
    clf = svm.SVC(C=c,kernel='rbf',decision_function_shape='ovr')
    clf.fit(X_train, Y_train)
    #print(clf.predict(X_valid))
    predicted=clf.predict(X_valid)
    print("accuracy for C="+str(c)+" on validation set is "+str(accuracy_score(Y_valid,predicted)*100))
    predicted=clf.predict(X_train)
    print("accuracy for C="+str(c)+" on training set is "+str(accuracy_score(Y_train,predicted)*100))

clf = svm.SVC(C=75,kernel='rbf',decision_function_shape='ovr')
clf.fit(X_train, Y_train)
```

```
predicted=clf.predict(X_test)
print("accuracy for C="+str(75)+" (Best Model) on testing set is "+str(accuracy_score(Y_test,predicted)*100))
```

```
accuracy for C=0.0001 on validation set is 71.11111111111111
accuracy for C=0.0001 on training set is 70.83333333333334
accuracy for C=0.001 on validation set is 71.11111111111111
accuracy for C=0.001 on training set is 70.83333333333334
accuracy for C=0.01 on validation set is 71.11111111111111
accuracy for C=0.01 on training set is 70.83333333333334
accuracy for C=0.1 on validation set is 97.77777777777777
accuracy for C=0.1 on training set is 96.16666666666667
accuracy for C=4 on validation set is 100.0
accuracy for C=4 on training set is 99.33333333333333
accuracy for C=16 on validation set is 100.0
accuracy for C=16 on training set is 99.5
accuracy for C=50 on validation set is 100.0
accuracy for C=50 on training set is 99.5
accuracy for C=75 on validation set is 100.0
accuracy for C=75 on training set is 99.66666666666667
accuracy for C=100 on validation set is 100.0
accuracy for C=100 on training set is 99.66666666666667
accuracy for C=150 on validation set is 100.0
accuracy for C=150 on training set is 99.66666666666667
accuracy for C=200 on validation set is 100.0
accuracy for C=200 on training set is 99.66666666666667
accuracy for C=500 on validation set is 100.0
accuracy for C=500 on training set is 99.66666666666667
accuracy for C=75 (Best Model) on testing set is 97.77777777777777
```

In [2]:

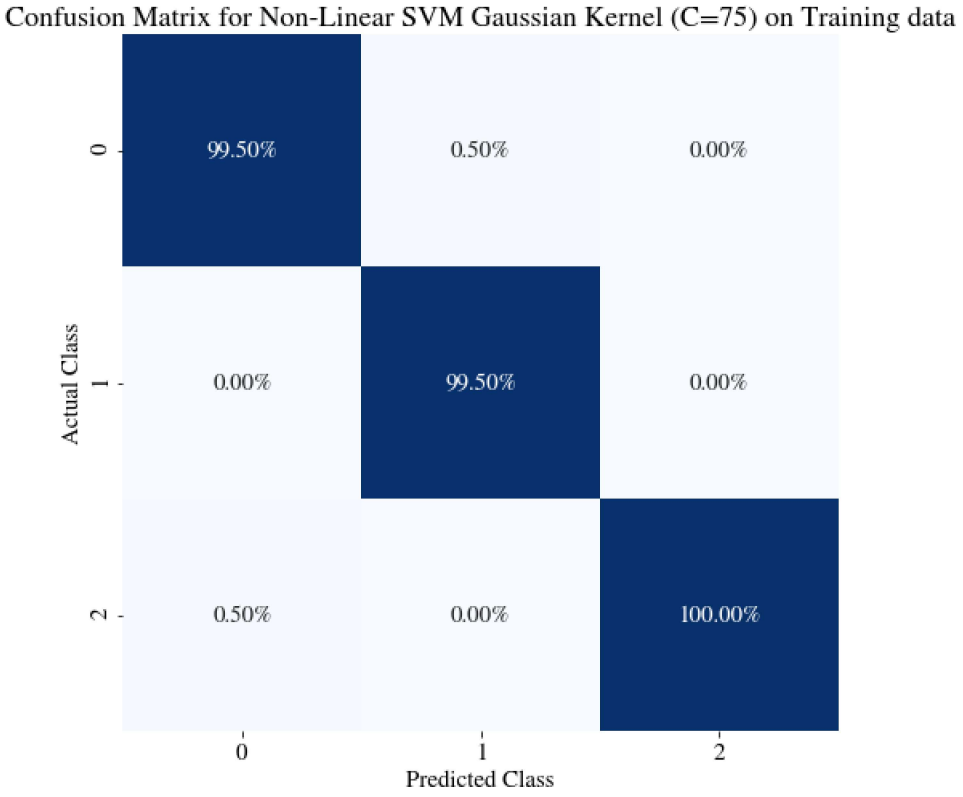
```
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
# plt.title(f'Confusion Matrix for Linear SVM (C={75}) on Testing data')
# #plt.savefig('Confusion_test_2.png')
plt.title(f'Confusion Matrix for Non-Linear SVM Gaussian Kernel (C={75}) on Testing data')
# plt.savefig('G_Confusion_test.png')

plt.show()
```



In [3]:

```
predicted=clf.predict(X_train)
```

```
confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for Non-Linear SVM Gaussian Kernel (C=75) on Training data')
# plt.savefig('G_Confusion_train.png')

plt.show()
```



In [4]:

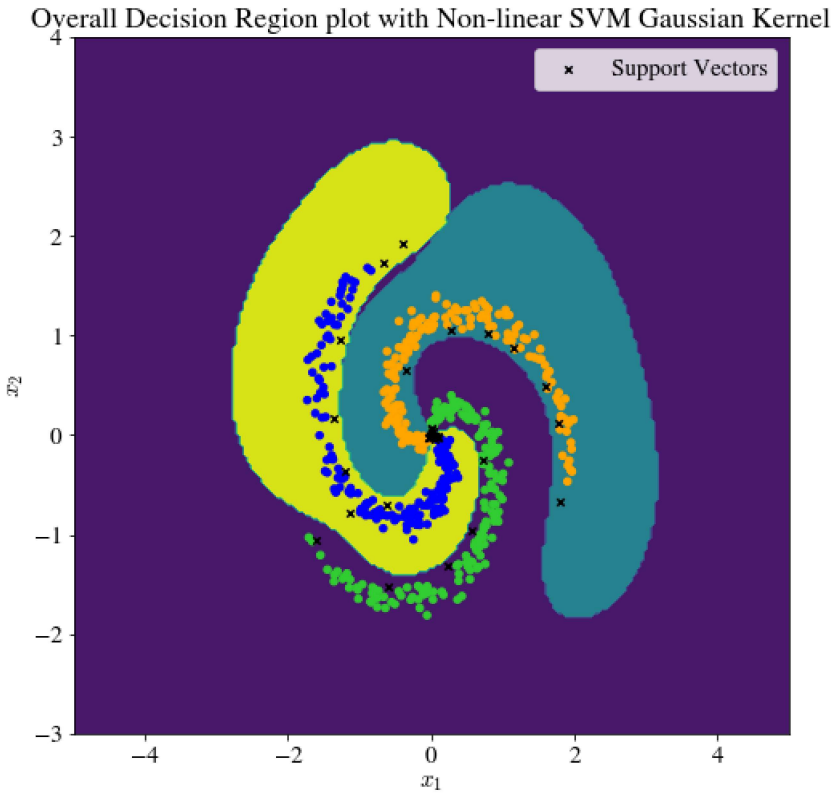
```
#print(clf.support_vectors_)
#print(np.where(X_train == clf.support_vectors_[0]))

#tp=np.delete(X_train[:,0],np.where(X_train[:,0] == clf.support_vectors_[:,0]))
#print(X_train.shape)
#print(X_train[:,0])
tp=np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0])
print(X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])].shape)
```

```
(556, 2)
```

In [5]:

```
x1=np.linspace(-5,5,num=200)
x2=np.linspace(-3,4,num=200)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
#predicted.clear()
num_cores = multiprocessing.cpu_count()

#predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(grid.
predicted=clf.predict(grid)
pos=np.empty(xx1.shape+(2,))
pos[:,:,0]=xx1
pos[:,:,1]=xx2


predicted=np.array(predicted)
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
```

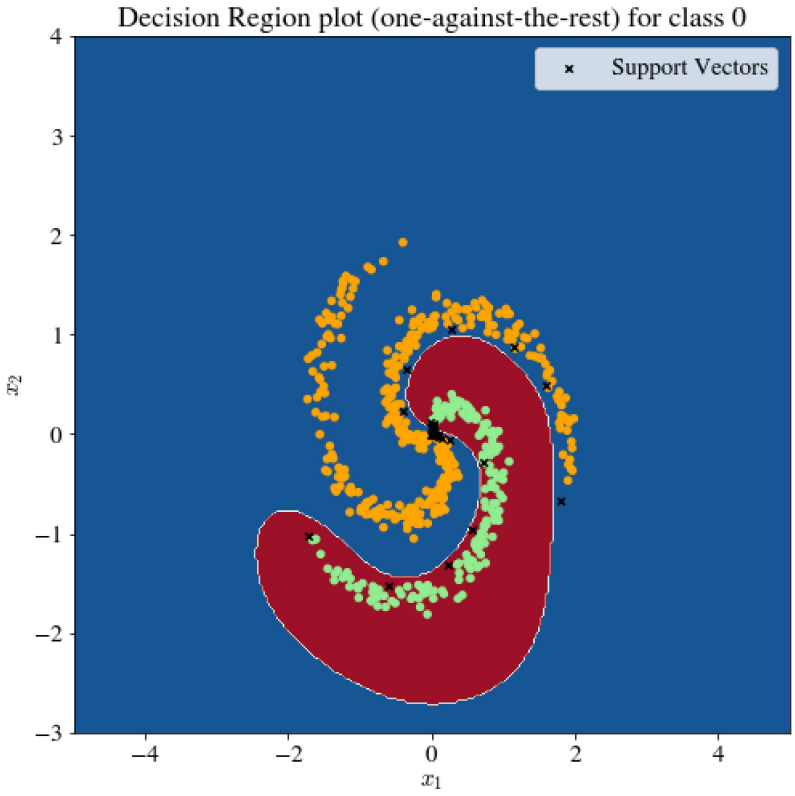
```
plt.contourf(xx1, xx2, predicted, cmap='viridis')
colors = ['limegreen','orange','blue']
real=X_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])]
plt.scatter(real[:,0], real[:,1], c=Y_train[~np.in1d(X_train[:,0],clf.support_vectors_[:,0])], cmap=matplotlib.co
#plt.scatter(np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0]), np.setdiff1d(X_train[:,1],clf.support_vectors_
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')
plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='black'
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Overall Decision Region plot with Non-linear SVM Gaussian Kernel')
plt.legend()
# plt.savefig('Overall_Decision_plot.png')
plt.show()
```

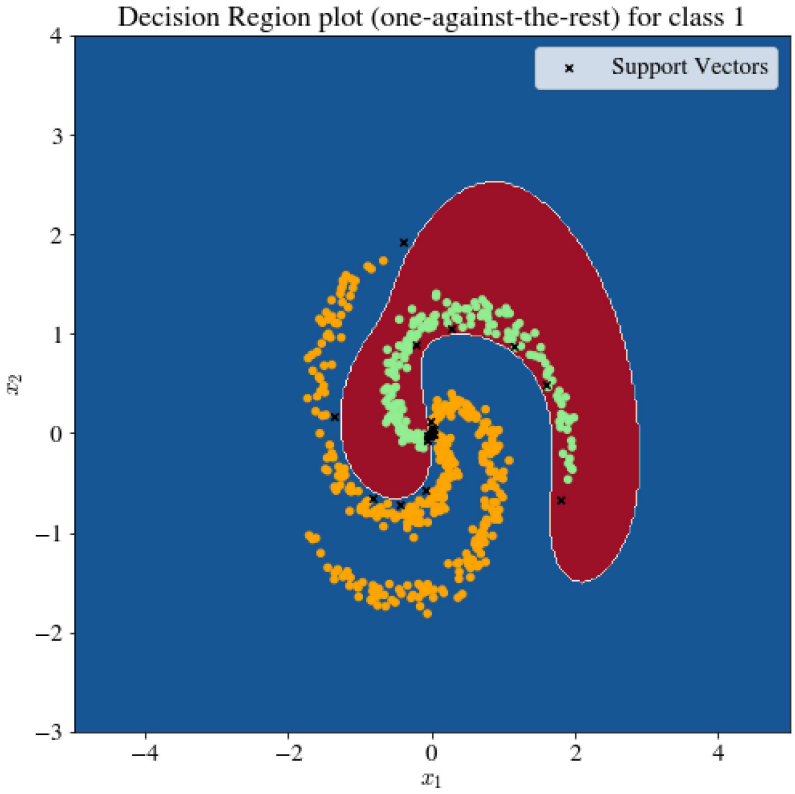


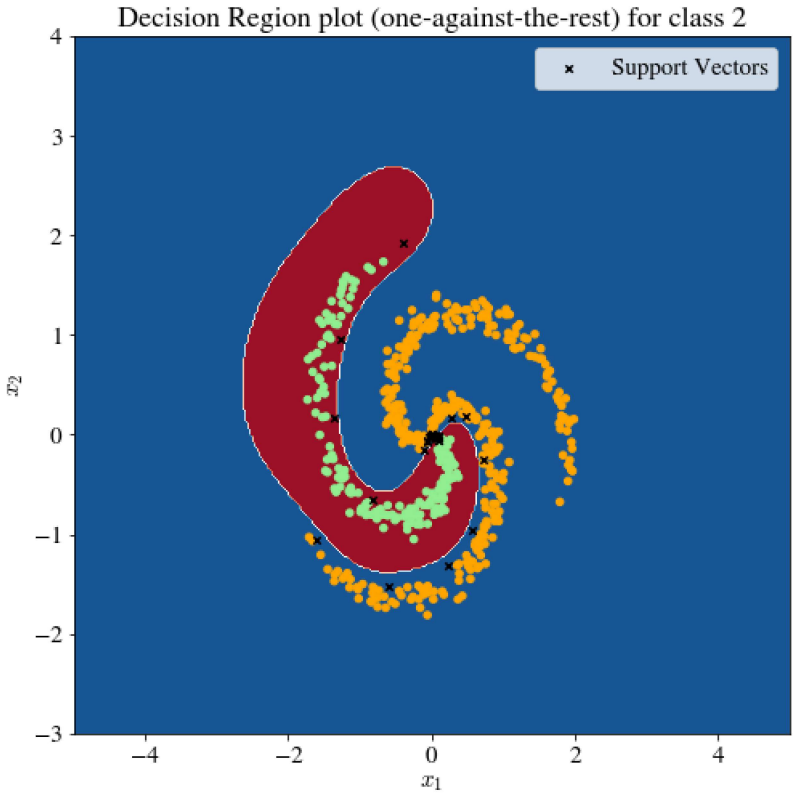
In [6]:

```
(unique, counts) = np.unique(Y_train, return_counts=True)
#print(unique)
for i in range(len(unique)):

    X_new_train=X_train[np.where(Y_train==i)]
    X_new_train=np.vstack((X_new_train,X_train[np.where(Y_train!=i)]))
    #print(X_new_train.shape)
    #Y_new_train=np.hstack((Y_train[np.where(Y_train==i)],Y_train[np.where(Y_train==p)]))
    Y_new_train=np.zeros(Y_train[np.where(Y_train==i)].shape)
    Y_new_train=np.hstack((Y_new_train,np.ones(Y_train[np.where(Y_train!=i)].shape)))
    #print(Y_new_train.shape)
    #print(Y_new_train)
    clf.fit(X_new_train, Y_new_train)
    x1=np.linspace(-5,5,num=300)
    x2=np.linspace(-3,4,num=300)
    xx1, xx2 = np.meshgrid(x1, x2)
    r1, r2 = xx1.flatten(), xx2.flatten()
    r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
    grid = np.hstack((r1,r2))
    #print(grid)
    #predicted.clear()
    num_cores = multiprocessing.cpu_count()

    #predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,counts) for i in range(g
    predicted=clf.predict(grid)
    pos=np.empty(xx1.shape+(2,))
    pos[:,:,0]=xx1
    pos[:,:,1]=xx2
```

```
    predicted=np.array(predicted)
    predicted=predicted.reshape(xx1.shape)
    fig = plt.figure(figsize=(8,8))
    plt.contourf(xx1, xx2, predicted, cmap='RdBu')
    colors = ['lightgreen','orange']
    real=X_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])]
    plt.scatter(real[:,0], real[:,1], c=Y_new_train[~np.in1d(X_new_train[:,0],clf.support_vectors_[:,0])], cmap=m
    #plt.scatter(np.setdiff1d(X_train[:,0],clf.support_vectors_[:,0]), np.setdiff1d(X_train[:,1],clf.support_vect
    #plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')
    plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1], marker='x',label="Support Vectors",color='bl
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title(f'Decision Region plot (one-against-the-rest) for class {i}')
    plt.legend()
#     plt.savefig(f'Decision_plot_class{i}.png')
    plt.show()
```




Decision Region plot (one-against-the-rest) for class 1
Support Vectors
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
x_1
x_2


Decision Region plot (one-against-the-rest) for class 2
Support Vectors
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
x_1
x_2

Dataset 2 Multilayer Feed-Forward Neural Network (MLFFNN) with two hidden layers

After hyperparameter tuning - the optimum number of nodes in hidden layers were found out to be:

1) H1: 55

2) H2: 51

With cross-entropy loss function and 'adam' optimzer (for faster convergence)

In [75]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn

import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.optimizers import SGD
from keras.layers import Dense

import kerastuner
from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
import time
import pickle




%matplotlib inline



plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)
```

In [76]:

```
Y_train=[]
X_train=pd.DataFrame()
train=['coast','forest','opencountry','street','tallbuilding']

for c, train_file in enumerate(train):
    data=pd.read_csv('dataset/'+train_file+'/train.csv')
    X_train=pd.concat([X_train,data],ignore_index=True)


    label_template=[0]*5
    label_template[c]=1

    for i in range(len(data)):
        Y_train.append(label_template)

X_train=X_train.to_numpy()
X_train=X_train[:,1:].astype('float64')
```

```
Y_train=np.array(Y_train)




valid_data=pd.DataFrame()
test_data=pd.DataFrame()


Y_test=[]
Y_valid=[]


for ind, train_file in enumerate(train):
    data=pd.read_csv('dataset/'+train_file+'/dev.csv')

    msk = np.random.rand(len(data)) < 0.5  #50-50 splits
    msk_valid=data[msk]
    msk_test=data[~msk]
    valid_data=pd.concat([valid_data,msk_valid],ignore_index=True)
    test_data=pd.concat([test_data,msk_test],ignore_index=True)


    label_template=[0]*5
    label_template[ind]=1

    for i in range(len(msk_valid)):
        Y_valid.append(label_template)

    for i in range(len(msk_test)):
        Y_test.append(label_template)

valid_data=valid_data.to_numpy()
X_valid=valid_data[:,1:].astype('float64')


test_data=test_data.to_numpy()
X_test=test_data[:,1:].astype('float64')


Y_valid=np.array(Y_valid)
Y_test=np.array(Y_test)
```

In [77]:

```
##Normalization

mean = np.mean(X_train, axis=0)
X_train -= mean
X_valid -= mean
X_test-=mean

std = np.std(X_train, axis=0)
X_train /= std
X_valid /= std
X_test/=std


##Shuffling

sep=X_train.shape[1]
aug=np.hstack((X_train,Y_train))
np.random.shuffle(aug)
X_train=aug[:,:sep]
Y_train=aug[:,sep:]



sep=X_valid.shape[1]
aug=np.hstack((X_valid,Y_valid))
```

```
np.random.shuffle(aug)
X_valid=aug[:,:sep]
Y_valid=aug[:,sep:]


sep=X_test.shape[1]
aug=np.hstack((X_test,Y_test))
np.random.shuffle(aug)
X_test=aug[:,:sep]
Y_test=aug[:,sep:]
```

In [78]:

```
# Model checkpoint- will save the model configuration having the max validation accuracy
checkpoint= keras.callbacks.ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose
```

In [79]:

```
# Using Keras tuner Random Search for hyper-parameter tuning

def build_model(hp):

    model = Sequential()
    model.add(Dense(units=hp.Int('units_1', min_value=25, max_value=70, step=2),input_dim=X_train.shape[
    model.add(Dense(units=hp.Int('units_2', min_value=25, max_value=70, step=2), activation='relu'))
    model.add(Dense(5, activation='softmax'))

    model.compile(optimizer="adam",
                  loss="categorical_crossentropy",
                  metrics=["accuracy"])

    return model


tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=20,              # how many variations on model?
    executions_per_trial=2 ,    # how many trials per variation? (same model could perform differently)
    directory="MLFFNN_project_12th_May",
    project_name="MLFFNN_4")     # I had ran and saved multiple models so just loading it

tuner.search_space_summary()
```

```
INFO:tensorflow:Reloading Oracle from existing project MLFFNN_project_12th_May\MLFFNN_4\oracle.json
INFO:tensorflow:Reloading Tuner from MLFFNN_project_12th_May\MLFFNN_4\tuner0.json
Search space summary
Default search space size: 2
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 25, 'max_value': 70, 'step': 2, 'sampling': None}
units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 25, 'max_value': 70, 'step': 2, 'sampling': None}
```

In [80]:

```
tuner.search(X_train,
             Y_train,
             epochs=50,
             batch_size=20,
             callbacks=[checkpoint],
             validation_data=(X_valid, Y_valid))
```

```
INFO:tensorflow:Oracle triggered exit
```

In [81]:

```
tuner.results_summary(20)
```

```
Results summary
Results in MLFFNN_project_12th_May\MLFFNN_4
Showing 20 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
units_1: 55
units_2: 51
```

```
Score: 0.7228571474552155
Trial summary
Hyperparameters:
units_1: 67
units_2: 67
Score: 0.7142857313156128
Trial summary
Hyperparameters:
units_1: 57
units_2: 43
Score: 0.7028571367263794
Trial summary
Hyperparameters:
units_1: 67
units_2: 41
Score: 0.699999988079071
Trial summary
Hyperparameters:
units_1: 57
units_2: 29
Score: 0.699999988079071
Trial summary
Hyperparameters:
units_1: 35
units_2: 39
Score: 0.6971428394317627
Trial summary
Hyperparameters:
units_1: 65
units_2: 61
Score: 0.6971428394317627
Trial summary
Hyperparameters:
units_1: 53
units_2: 49
Score: 0.691428542137146
Trial summary
Hyperparameters:
units_1: 57
units_2: 49
Score: 0.6857142746448517
Trial summary
Hyperparameters:
units_1: 29
units_2: 59
Score: 0.6857142746448517
Trial summary
Hyperparameters:
units_1: 41
units_2: 49
Score: 0.6857142746448517
Trial summary
Hyperparameters:
units_1: 63
units_2: 37
Score: 0.6857142746448517
Trial summary
Hyperparameters:
units_1: 67
units_2: 63
Score: 0.6828571259975433
Trial summary
Hyperparameters:
units_1: 67
units_2: 27
Score: 0.6828571259975433
Trial summary
Hyperparameters:
units_1: 43
units_2: 41
Score: 0.6828571259975433
Trial summary
Hyperparameters:
units_1: 63
units_2: 53
Score: 0.677142858505249
```

```
Trial summary
Hyperparameters:
units_1: 37
units_2: 27
Score: 0.6714285612106323
Trial summary
Hyperparameters:
units_1: 39
units_2: 31
Score: 0.668571412563324
Trial summary
Hyperparameters:
units_1: 47
units_2: 69
Score: 0.665714293718338
Trial summary
Hyperparameters:
units_1: 25
units_2: 59
Score: 0.665714293718338
```

In [88]:

```
models=tuner.get_best_models(1)
best_model= models[0]

checkpoint= keras.callbacks.ModelCheckpoint('best_model_.h5', monitor='val_accuracy', mode='max', verbos
# Fit data to model
history= best_model.fit(X_train, Y_train,
                        epochs=50,
                        batch_size=20,
                        verbose=1,
                        callbacks=[checkpoint],
                        validation_data=(X_valid, Y_valid)
                       )


print("\nEvaluating the training model...\n ")

# evaluate the keras model
loss, accuracy = best_model.evaluate(X_train, Y_train)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
Epoch 1/50
 1/61 [..............................] - ETA: 0s - loss: 0.5896 - accuracy: 0.8000
Epoch 00001: val_accuracy improved from -inf to 0.72571, saving model to best_model_.h5
61/61 [==============================] - 0s 3ms/step - loss: 0.5991 - accuracy: 0.7861 - val_loss: 0.896
4 - val_accuracy: 0.7257
Epoch 2/50
 1/61 [..............................] - ETA: 0s - loss: 0.5442 - accuracy: 0.8000
Epoch 00002: val_accuracy did not improve from 0.72571
61/61 [==============================] - 0s 914us/step - loss: 0.5848 - accuracy: 0.7967 - val_loss: 0.9
226 - val_accuracy: 0.6857
Epoch 3/50
 1/61 [..............................] - ETA: 0s - loss: 0.4693 - accuracy: 0.9000
Epoch 00003: val_accuracy did not improve from 0.72571
61/61 [==============================] - 0s 815us/step - loss: 0.5672 - accuracy: 0.7984 - val_loss: 0.9
294 - val_accuracy: 0.6571
Epoch 4/50
 1/61 [..............................] - ETA: 0s - loss: 0.3845 - accuracy: 0.8000
Epoch 00004: val_accuracy did not improve from 0.72571
61/61 [==============================] - 0s 825us/step - loss: 0.5621 - accuracy: 0.7975 - val_loss: 0.9
439 - val_accuracy: 0.6857
Epoch 5/50
 1/61 [..............................] - ETA: 0s - loss: 0.8687 - accuracy: 0.6000
Epoch 00005: val_accuracy did not improve from 0.72571
61/61 [==============================] - 0s 864us/step - loss: 0.5415 - accuracy: 0.8115 - val_loss: 0.9
366 - val_accuracy: 0.6800
Epoch 6/50
 1/61 [..............................] - ETA: 0s - loss: 0.5739 - accuracy: 0.9000
Epoch 00006: val_accuracy did not improve from 0.72571
61/61 [==============================] - 0s 891us/step - loss: 0.5272 - accuracy: 0.8164 - val_loss: 0.9
718 - val_accuracy: 0.6743
Epoch 7/50
 1/61 [..............................] - ETA: 0s - loss: 0.3190 - accuracy: 0.9500
```

```
saved_model = load_model('best_model_.h5')
saved_model.summary()
```

```
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting with
a freshly initialized optimizer.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 55)                1375
_________________________________________________________________
dense_1 (Dense)              (None, 51)                2856
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 260
=================================================================
Total params: 4,491
Trainable params: 4,491
Non-trainable params: 0
_________________________________________________________________
```

In [90]:

```
loss, accuracy = saved_model.evaluate(X_valid, Y_valid,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
6/6 [==============================] - 0s 567us/step - loss: 0.8964 - accuracy: 0.7257


 [INFO] loss=0.8964, accuracy: 72.5714%
```

In [91]:

```
loss, accuracy = saved_model.evaluate(X_test, Y_test,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
6/6 [==============================] - 0s 595us/step - loss: 0.9813 - accuracy: 0.6629


 [INFO] loss=0.9813, accuracy: 66.2857%
```

In [92]:

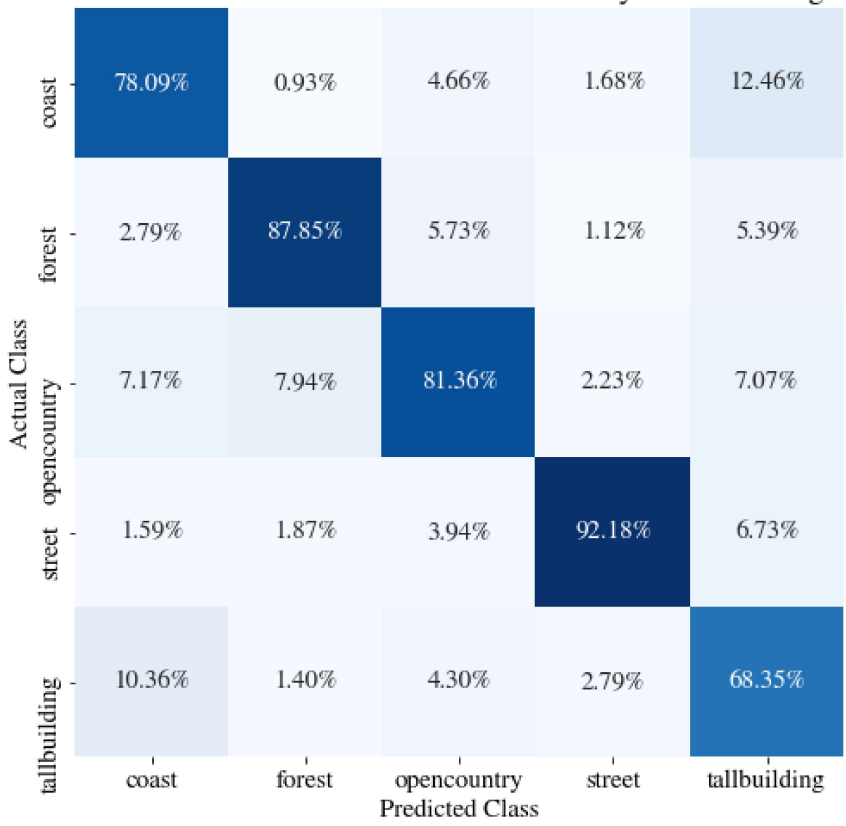
```
loss, accuracy = saved_model.evaluate(X_train, Y_train,verbose=1)
print("\n\n [INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
39/39 [==============================] - 0s 553us/step - loss: 0.5768 - accuracy: 0.8025


 [INFO] loss=0.5768, accuracy: 80.2459%
```

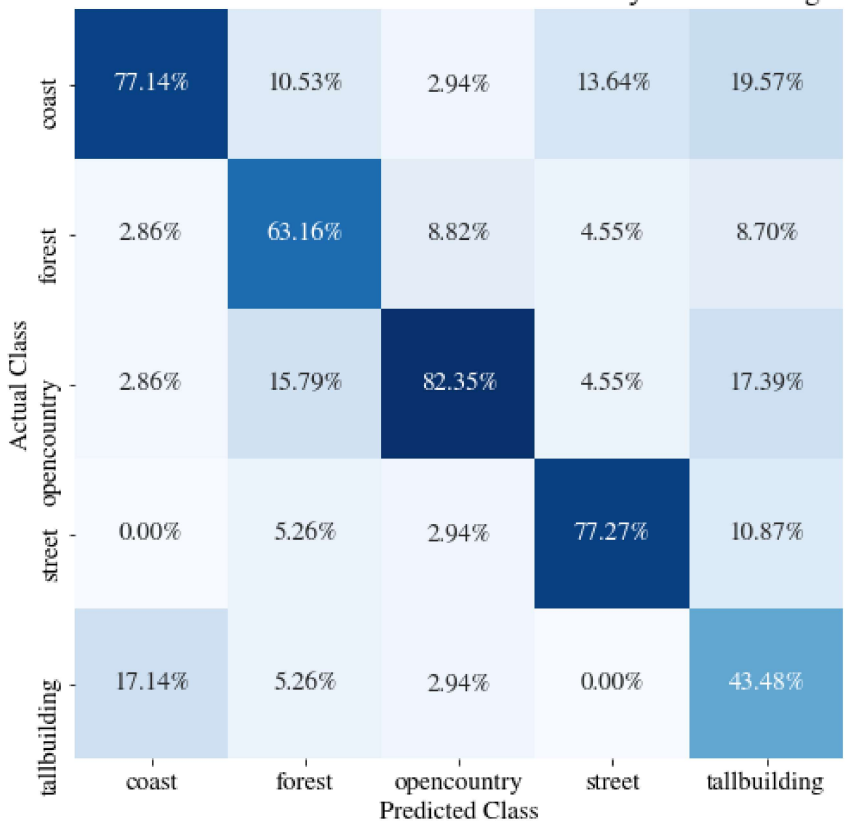
In [94]:

```
predicted_train =np.argmax(saved_model.predict(X_train), axis=-1)
labels=np.where(Y_train==1)[1]
confuse=confusion_matrix(labels,predicted_train)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=train,yticklabels=train)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFNN with 2 hidden layers on Training  data')
plt.savefig('MLFFNN_Confusion_train_final.png')
plt.show()

predicted_test =np.argmax(saved_model.predict(X_test), axis=-1)
labels_test=np.where(Y_test==1)[1]
confuse=confusion_matrix(labels_test,predicted_test)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=train,yticklabels=train)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for MLFNN with 2 hidden layers on Testing data')
plt.savefig('MLFFNN_Confusion_test_final.png')
plt.show()
```

Confusion Matrix for MLFNN with 2 hidden layers on Training data



Confusion Matrix for MLFNN with 2 hidden layers on Testing data



Dataset 2: Gaussian kernel based SVM using one-against-the-rest approach

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn
from sklearn import svm



%matplotlib inline



plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)




np.random.seed(42)

#Taking input from csv file and taking x and y out
train=['coast','forest','opencountry','street','tallbuilding']
data=pd.DataFrame()
for c, train_file in enumerate(train):
    data_2=pd.read_csv('dataset/'+train_file+'/train.csv')
    #data_2=data_2.to_numpy()
    #print(data_2)
    data_2['Y']=c
    X=data_2.iloc[:,1:]
    #print(X)
    data=pd.concat([data,X],ignore_index=True)

#print(data)

X_train=data.iloc[:,:-1]
Y_train=data.iloc[:,-1]

X_train=X_train.to_numpy()
Y_train=Y_train.to_numpy()
#print(X_train)

X_valid=pd.DataFrame()
X_test=pd.DataFrame()
for c, train_file in enumerate(train):
    data_2=pd.read_csv('dataset/'+train_file+'/dev.csv')
    #data_2=data_2.to_numpy()
    #print(data_2)
    data_2['Y']=c
    X=data_2.iloc[:,1:]
    #print(X)
    X_valid=pd.concat([X_valid,X.iloc[0:int(data_2.shape[0]/2),:]],ignore_index=True)
    X_test=pd.concat([X_test,X.iloc[int(data_2.shape[0]/2):,:]],ignore_index=True)

Y_valid=X_valid.iloc[:,-1]
Y_test=X_test.iloc[:,-1]

X_valid=X_valid.iloc[:,:-1]

X_test=X_test.iloc[:,:-1]

X_valid=X_valid.to_numpy()
Y_valid=Y_valid.to_numpy()

X_test=X_test.to_numpy()
Y_test=Y_test.to_numpy()
```

In [2]:

```
sep=X_train.shape[1]
aug=np.c_[(X_train,Y_train)]
np.random.shuffle(aug)
X_train=aug[:,:sep]
Y_train=aug[:,sep:]
```

```
sep=X_valid.shape[1]
aug=np.c_[(X_valid,Y_valid)]
np.random.shuffle(aug)
X_valid=aug[:,:sep]
Y_valid=aug[:,sep:]


sep=X_test.shape[1]
aug=np.c_[(X_test,Y_test)]
np.random.shuffle(aug)
X_test=aug[:,:sep]
Y_test=aug[:,sep:]


Y_train= np.ravel(Y_train)
Y_valid= np.ravel(Y_valid)
Y_test= np.ravel(Y_test)
```

In [3]:

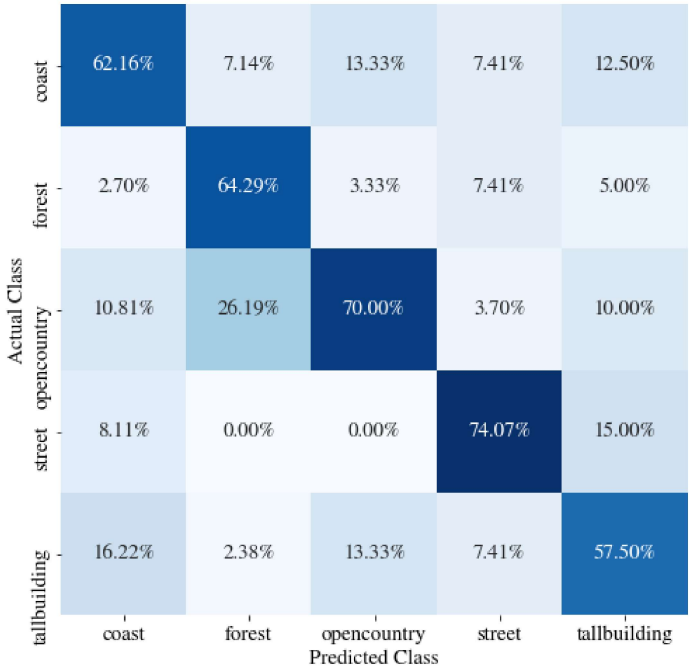
```
C=[0.0001,0.001,0.01,0.1,2.5,3,3.1,4,5,7,16,50,75,100,150]


for c in C:
    clf = svm.SVC(C=c,kernel='rbf',decision_function_shape='ovr')
    clf.fit(X_train, Y_train)
    #print(clf.predict(X_valid))
    predicted=clf.predict(X_valid)
    print("accuracy for C="+str(c)+" on validation set is "+str(accuracy_score(Y_valid,predicted)*100))
    predicted=clf.predict(X_train)
    print("accuracy for C="+str(c)+" on training set is "+str(accuracy_score(Y_train,predicted)*100))

clf = svm.SVC(C=3.1,kernel='rbf',decision_function_shape='ovr')
clf.fit(X_train, Y_train)
predicted=clf.predict(X_test)
print("accuracy for C="+str(3.1)+" (Best Model) on testing set is "+str(accuracy_score(Y_test,predicted)*100))
```

```
accuracy for C=0.0001 on validation set is 23.563218390804597
accuracy for C=0.0001 on training set is 23.524590163934427
accuracy for C=0.001 on validation set is 23.563218390804597
accuracy for C=0.001 on training set is 23.524590163934427
accuracy for C=0.01 on validation set is 23.563218390804597
accuracy for C=0.01 on training set is 23.524590163934427
accuracy for C=0.1 on validation set is 58.04597701149425
accuracy for C=0.1 on training set is 62.704918032786885
accuracy for C=2.5 on validation set is 62.06896551724138
accuracy for C=2.5 on training set is 84.75409836065573
accuracy for C=3 on validation set is 62.06896551724138
accuracy for C=3 on training set is 85.90163934426229
accuracy for C=3.1 on validation set is 62.06896551724138
accuracy for C=3.1 on training set is 86.31147540983606
accuracy for C=4 on validation set is 60.3448275862069
accuracy for C=4 on training set is 88.11475409836066
accuracy for C=5 on validation set is 59.77011494252874
accuracy for C=5 on training set is 90.0
accuracy for C=7 on validation set is 58.620689655172406
accuracy for C=7 on training set is 91.9672131147541
accuracy for C=16 on validation set is 59.195402298850574
accuracy for C=16 on training set is 96.06557377049181
accuracy for C=50 on validation set is 58.04597701149425
accuracy for C=50 on training set is 99.01639344262296
accuracy for C=75 on validation set is 57.47126436781609
accuracy for C=75 on training set is 99.8360655737705
accuracy for C=100 on validation set is 58.620689655172406
accuracy for C=100 on training set is 100.0
accuracy for C=150 on validation set is 57.47126436781609
accuracy for C=150 on training set is 100.0
accuracy for C=3.1 (Best Model) on testing set is 64.77272727272727
```

In [4]:

```
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=train,yticklabels=train)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f'Confusion Matrix for Gaussian kernel based SVM (C={3.1}) on Testing data')
# plt.savefig('SVM_Confusion_test.png')

plt.show()
```

Confusion Matrix for Gaussian kernel based SVM (C=3.1) on Testing data



In [5]:

```
predicted=clf.predict(X_train)
confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=train,yticklabels=train)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title(f" Confusion Matrix Gaussian kernel based SVM (C=3.1) on Training data")
# plt.savefig('SVM_Confusion_train.png')

plt.show()
```

Confusion Matrix Gaussian kernel based SVM (C=3.1) on Training data