Homework3

January 1, 2020

1 Question 4.1

1.1 a.

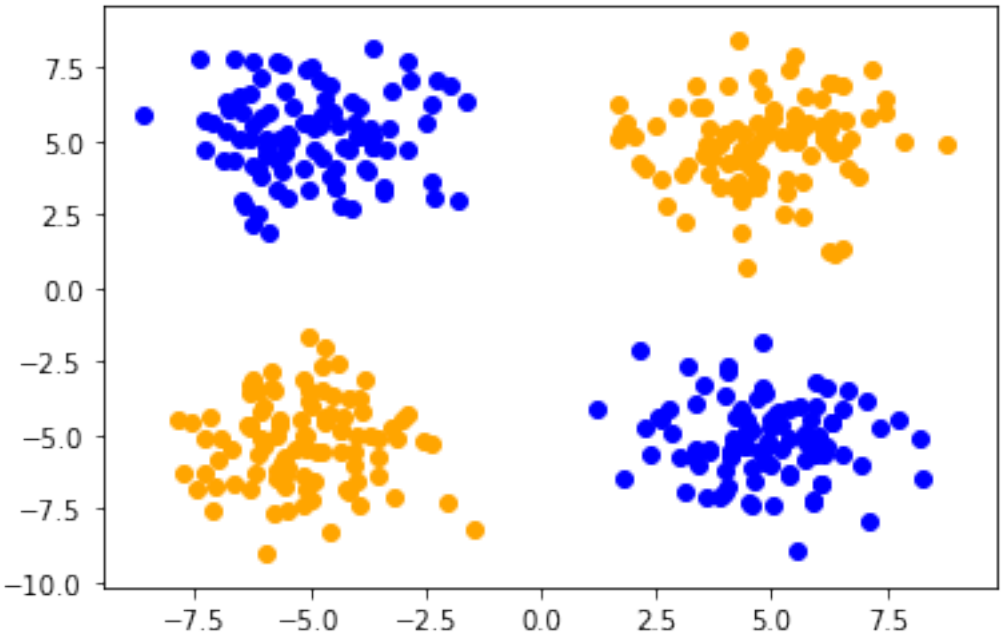
1.1.1 Creating (X1,y1) Dataset

[0]:
```
import numpy as np
np.random.seed(0)
mean = (-5, 5)
cov = [[2, 0], [0, 2]]
X1_1 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, -5)
cov = [[2, 0], [0, 2]]
X1_2 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, 5)
cov = [[2, 0], [0, 2]]
X1_3 = np.random.multivariate_normal(mean, cov, (100))
mean = (-5, -5)
cov = [[2, 0], [0, 2]]
X1_4 = np.random.multivariate_normal(mean, cov, (100))
X1 = np.vstack((X1_1,X1_2,X1_3,X1_4))
y1=np.append(np.zeros(200),np.ones(200))
```

1.1.2 Plotting (X1,y1) Dataset

[2]:
```
import matplotlib.pyplot as plt
plt.scatter(X1[0:200,0],X1[0:200,1],color='blue')
plt.scatter(X1[200:400,0],X1[200:400,1],color='orange')
```

[2]:
```
<matplotlib.collections.PathCollection at 0x7f7434f22d30>
```



1.2 b.

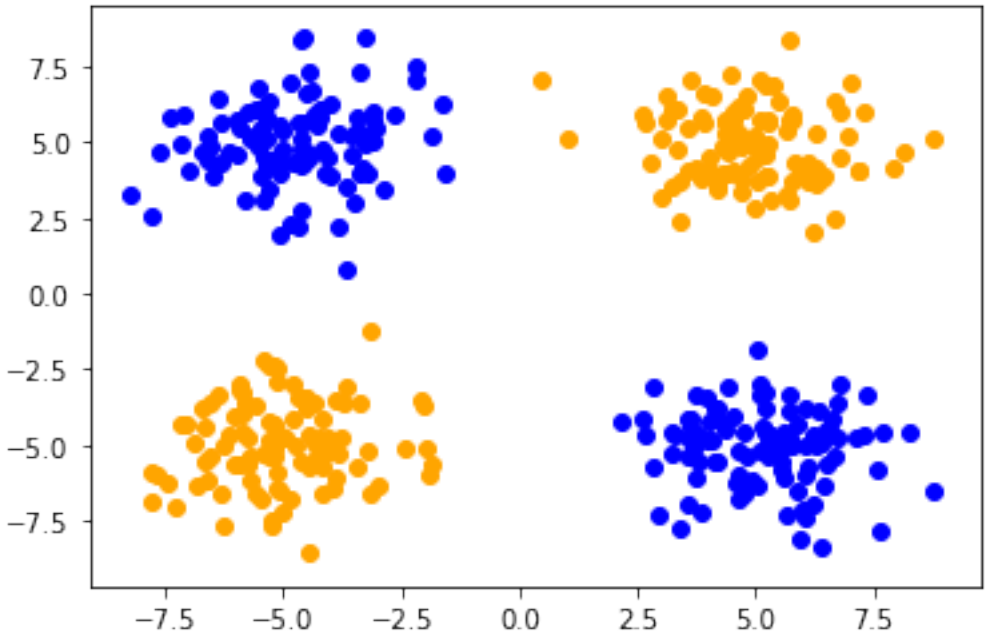
1.2.1 Creating (X2,y2) Dataset

[0]:
```
import numpy as np
np.random.seed(10)
mean = (-5, 5)
cov = [[2, 0], [0, 2]]
X1_1 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, -5)
cov = [[2, 0], [0, 2]]
X1_2 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, 5)
cov = [[2, 0], [0, 2]]
X1_3 = np.random.multivariate_normal(mean, cov, (100))
mean = (-5, -5)
cov = [[2, 0], [0, 2]]
X1_4 = np.random.multivariate_normal(mean, cov, (100))
X2 = np.vstack((X1_1,X1_2,X1_3,X1_4))
y2=np.append(np.zeros(200),np.ones(200))
```

1.2.2 Plotting (X2,y2) Dataset

[4]:
```
import matplotlib.pyplot as plt
plt.scatter(X2[0:200,0],X2[0:200,1],color='blue')
plt.scatter(X2[200:400,0],X2[200:400,1],color='orange')
```

[4]:
```
<matplotlib.collections.PathCollection at 0x7f74322068d0>
```



1.3 c.

1.3.1 Creating (X3,y3) Dataset

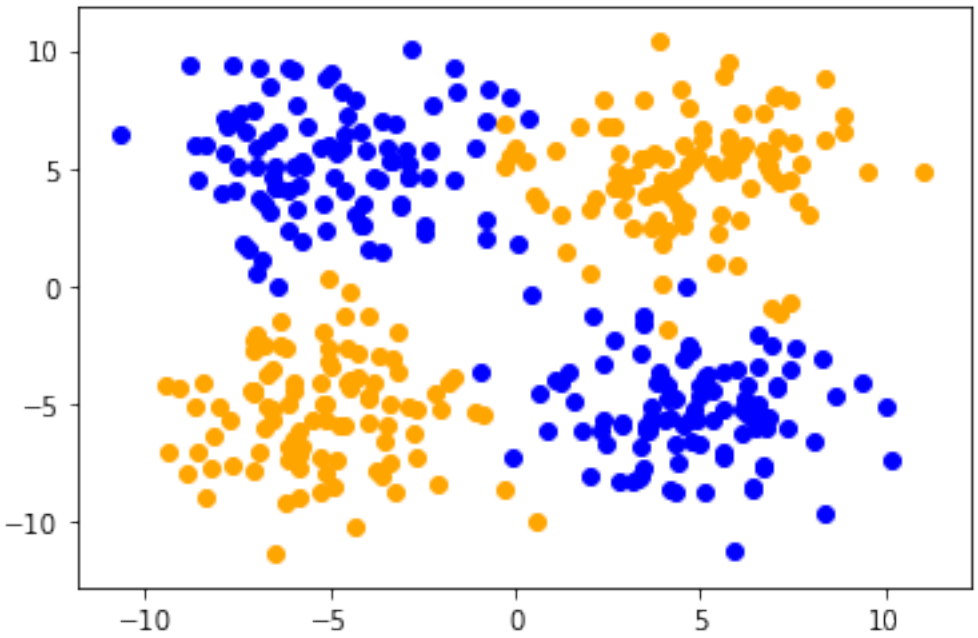
[0]:
```
import numpy as np
np.random.seed(0)
mean = (-5, 5)
cov = [[5, 0], [0, 5]]
X1_1 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, -5)
cov = [[5, 0], [0, 5]]
X1_2 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, 5)
cov = [[5, 0], [0, 5]]
X1_3 = np.random.multivariate_normal(mean, cov, (100))
mean = (-5, -5)
cov = [[5, 0], [0, 5]]
X1_4 = np.random.multivariate_normal(mean, cov, (100))
```

```
X3 = np.vstack((X1_1,X1_2,X1_3,X1_4))
y3=np.append(np.zeros(200),np.ones(200))
```

1.3.2 Plotting (X3,y3) Dataset

[6]:
```
import matplotlib.pyplot as plt
plt.scatter(X3[0:200,0],X3[0:200,1],color='blue')
plt.scatter(X3[200:400,0],X3[200:400,1],color='orange')
```

[6]:
```
<matplotlib.collections.PathCollection at 0x7f7432172be0>
```



1.3.3 Creating (X4,y4) Dataset

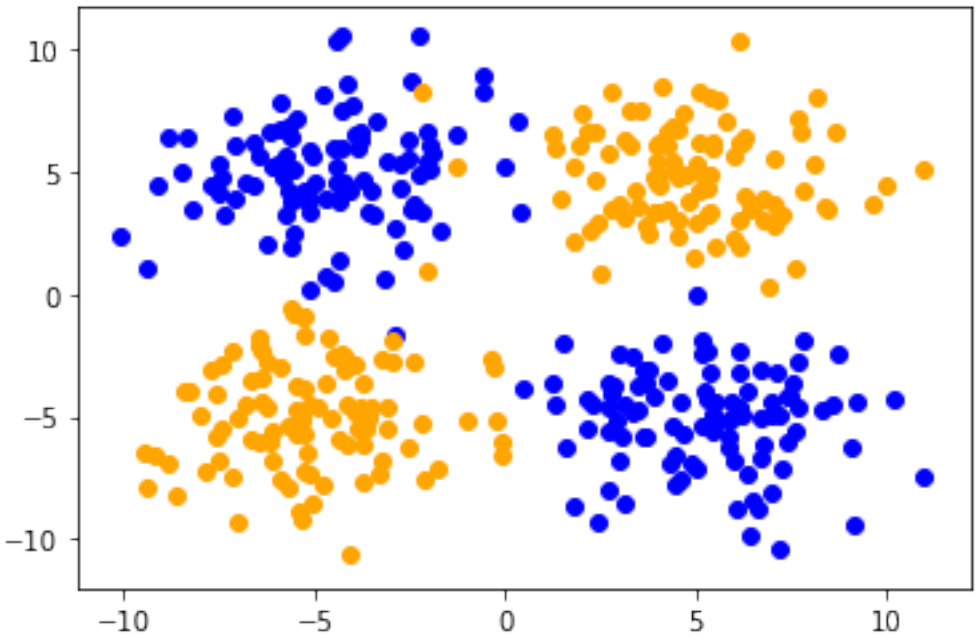
[0]:
```
import numpy as np
np.random.seed(10)
mean = (-5, 5)
cov = [[5, 0], [0, 5]]
X1_1 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, -5)
cov = [[5, 0], [0, 5]]
X1_2 = np.random.multivariate_normal(mean, cov, (100))
mean = (5, 5)
cov = [[5, 0], [0, 5]]
X1_3 = np.random.multivariate_normal(mean, cov, (100))
mean = (-5, -5)
```

```
cov = [[5, 0], [0, 5]]
X1_4 = np.random.multivariate_normal(mean, cov, (100))
X4 = np.vstack((X1_1,X1_2,X1_3,X1_4))
y4=np.append(np.zeros(200),np.ones(200))
```

1.3.4 Plotting (X4,y4) Dataset

[8]:
```
import matplotlib.pyplot as plt
plt.scatter(X4[0:200,0],X4[0:200,1],color='blue')
plt.scatter(X4[200:400,0],X4[200:400,1],color='orange')
```

[8]:
```
<matplotlib.collections.PathCollection at 0x7f7432157550>
```



1.4 d.

Note that plots for each dataset has been drawn below each one of them.

2 Question 4.2

2.0.1 Defining plot_decision_boundary class

[9]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.optimizers import Adam
```

```
import matplotlib.pyplot as plt
%matplotlib inline
def plot_decision_boundary(X, y, model, steps=1000, cmap='Paired'):
    cmap = plt.get_cmap(cmap)

    # Define region of interest by data limits
    xmin, xmax = X[:,0].min() - 1, X[:,0].max() + 1
    ymin, ymax = X[:,1].min() - 1, X[:,1].max() + 1
    steps = 1000
    x_span = np.linspace(xmin, xmax, steps)
    y_span = np.linspace(ymin, ymax, steps)
    xx, yy = np.meshgrid(x_span, y_span)

    # Make predictions across region of interest
    labels = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Plot decision boundary in region of interest
    z = labels.reshape(xx.shape)

    fig, ax = plt.subplots()
    ax.contourf(xx, yy, z, cmap=cmap, alpha=0.5)

    # Get predicted labels on training data and plot
    ax.scatter(X[:,0], X[:,1], c=y, cmap=cmap, lw=0)

    return fig, ax
```

```
Using TensorFlow backend.
```

```
<IPython.core.display.HTML object>
```

2.1 a.

2.1.1 Runinig the back propagation with lr=0.01 and 2 nodes on with 1000 iterations

[10]:
```
model421 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model421.add(Dense(output_dim=2, input_shape=(2, ), **layer_kw))
model421.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model421.compile(optimizer=sgd, loss='binary_crossentropy')
history = model421.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3657: The name tf.log is
deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/ops/nn_impl.py:183: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=2, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
```

```
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is
deprecated. Please use tf.compat.v1.Session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name
tf.get_default_session is deprecated. Please use
tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is
deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name
tf.is_variable_initialized is deprecated. Please use
tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name
tf.variables_initializer is deprecated. Please use
tf.compat.v1.variables_initializer instead.
```

2.1.2 Runinig the back propagation with lr=0.01 and 4 nodes on with 1000 iterations

[11]:
```
model422 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model422.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model422.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model422.compile(optimizer=sgd, loss='binary_crossentropy')
history = model422.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
```

```
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

2.1.3 Runinig the back propagation with lr=0.01 and 15 nodes on with 1000 iterations

[12]:
```
model423 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model423.add(Dense(output_dim=15, input_shape=(2, ), **layer_kw))
model423.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model423.compile(optimizer=sgd, loss='binary_crossentropy')
history = model423.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=15, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

2.2 b.

2.2.1 Evaluating the model421 with lr=0.01 and 2 nodes and 1000 iterationson on (train set)

[13]:
```
model421.evaluate(x=X1,y=y1)
```

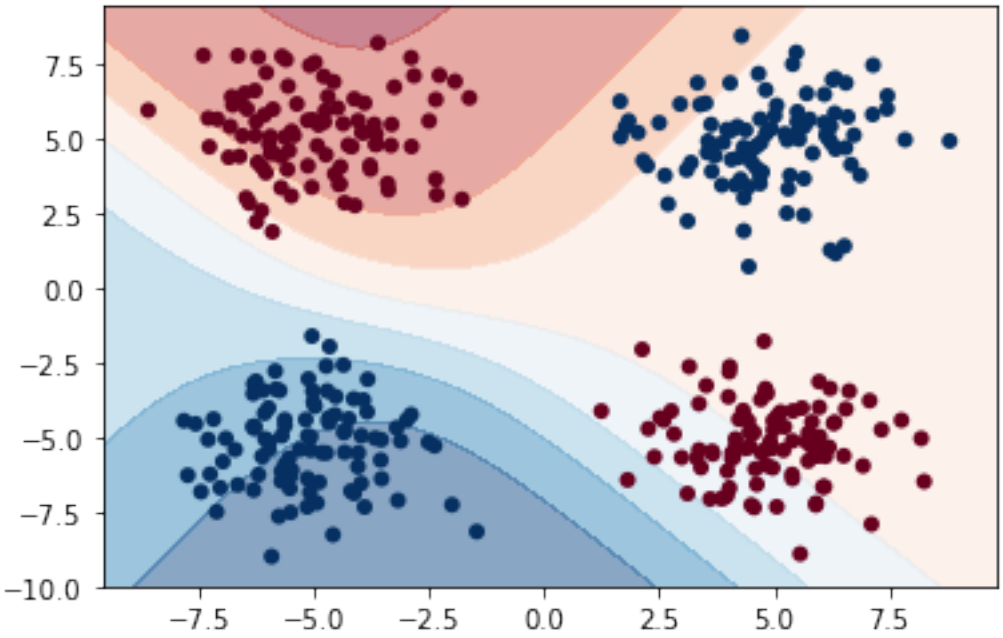
```
400/400 [==============================] - 0s 176us/step
```

[13]:
```
0.5204467594623565
```

2.2.2 Decision boundary for model421 on (train set)

[14]:
```
plot_decision_boundary(X1, y1, model421, cmap='RdBu')
```

[14]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f74320fb898>)
```



2.2.3 Evaluating the model421 with lr=0.01 and 2 nodes and 1000 iterationson on (test set)

[15]:
```
model421.evaluate(x=X2,y=y2)
```

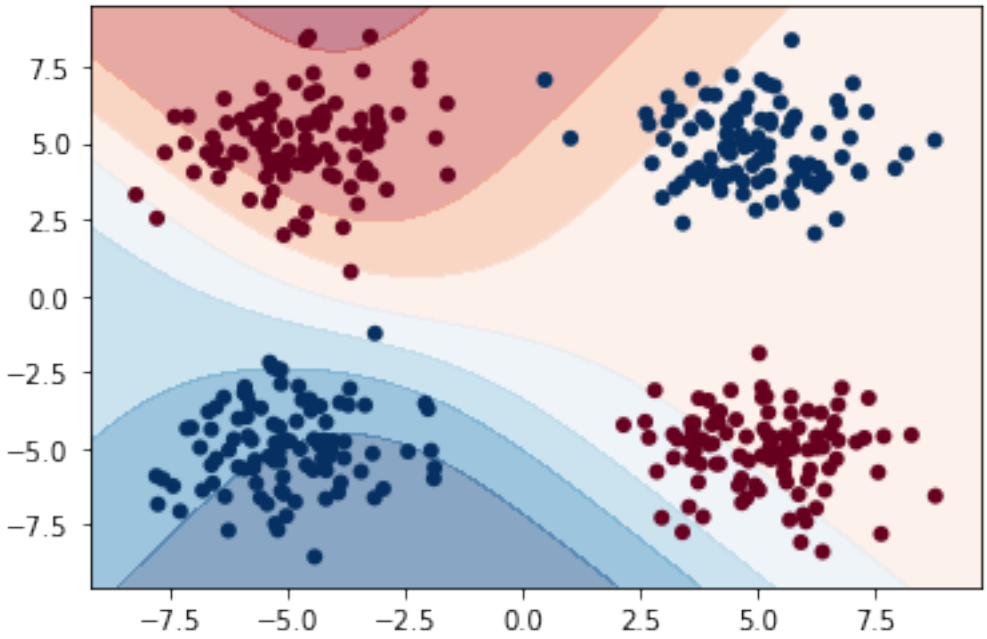
```
400/400 [==============================] - 0s 78us/step
```

[15]:
```
0.5193993538618088
```

2.2.4 Decision boundary for model421 on (test set)

[16]:
```
plot_decision_boundary(X2, y2, model421, cmap='RdBu')
```

[16]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e426fb38>)
```



2.2.5 Evaluating the model422 with lr=0.01 and 4 nodes and 1000 iterationson on (train set)

[17]:
```
model422.evaluate(x=X1,y=y1)
```

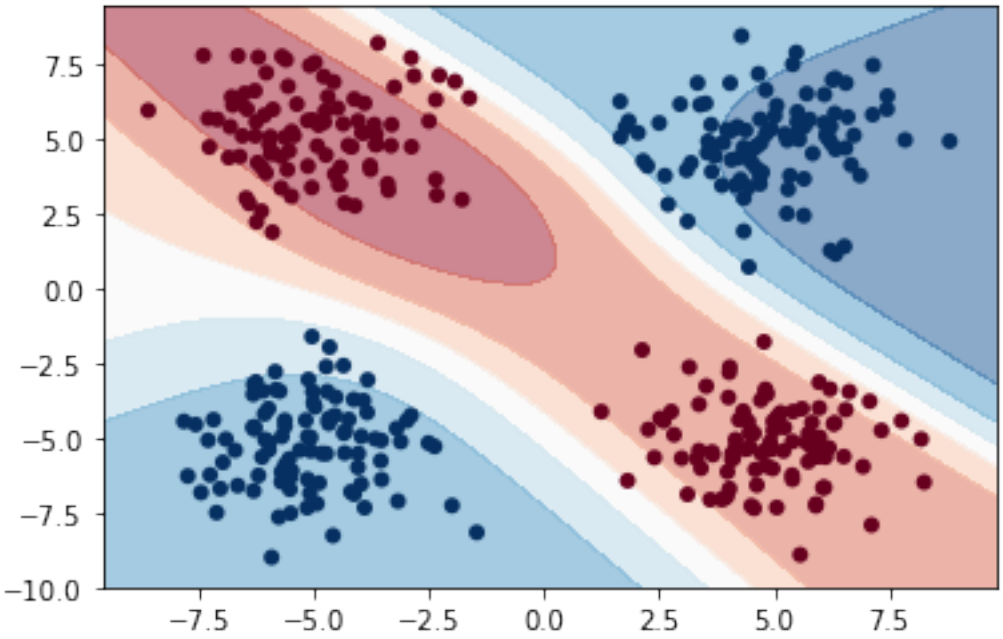
```
400/400 [==============================] - 0s 163us/step
```

[17]:
```
0.1761579179763794
```

2.2.6 Decision boundary for model422 on (train set)

[18]:
```
plot_decision_boundary(X1, y1, model422, cmap='RdBu')
```

[18]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e008b4e0>)
```



2.2.7 Evaluating the model422 with lr=0.01 and 4 nodes and 1000 iterationson on (test set)

[19]:
```
model422.evaluate(x=X2,y=y2)
```

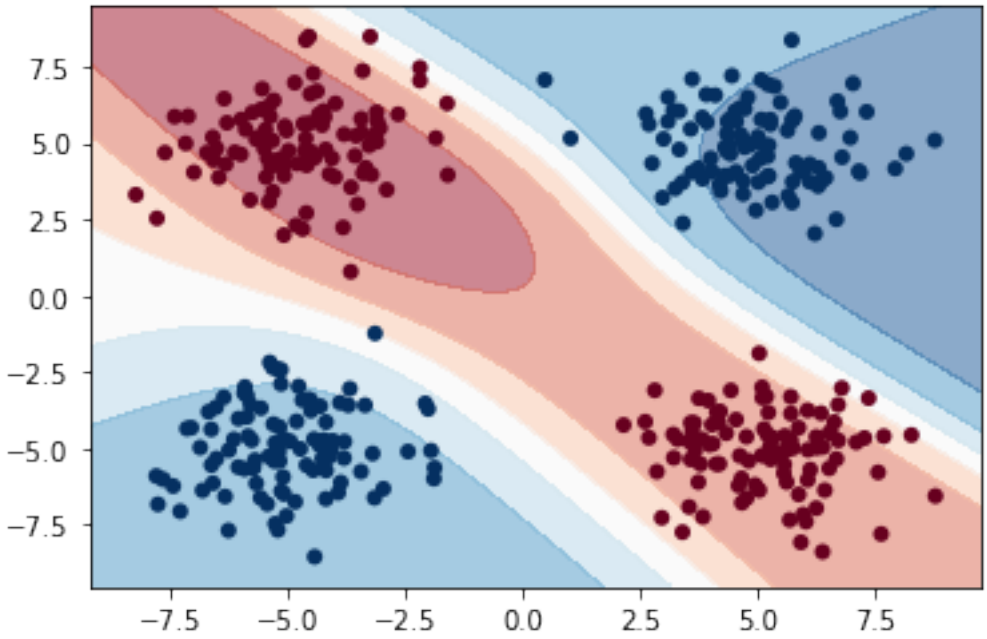
```
400/400 [==============================] - 0s 89us/step
```

[19]:
```
0.17818798661231994
```

2.2.8 Decision boundary for model422 on (test set)

[20]:
```
plot_decision_boundary(X2, y2, model422, cmap='RdBu')
```

[20]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73812b8a58>)
```



2.2.9 Evaluating the model423 with lr=0.01 and 15 nodes and 1000 iterationson on (train set)

[21]:
```
model423.evaluate(x=X1,y=y1)
```

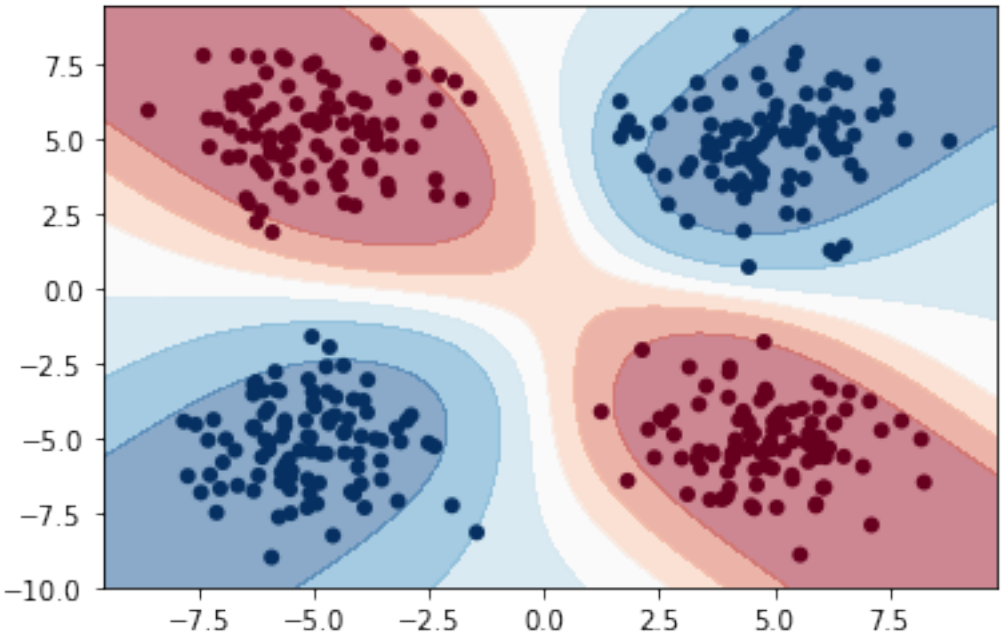
```
400/400 [==============================] - 0s 182us/step
```

[21]:
```
0.056607278883457186
```

2.2.10 Decision boundary for model423 on (train set)

[22]:
```
plot_decision_boundary(X1, y1, model423, cmap='RdBu')
```

[22]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e43b77f0>)
```



2.2.11 Evaluating the model423 with lr=0.01 and 15 nodes and 1000 iterationson on (test set)

[23]:
```
model423.evaluate(x=X2,y=y2)
```

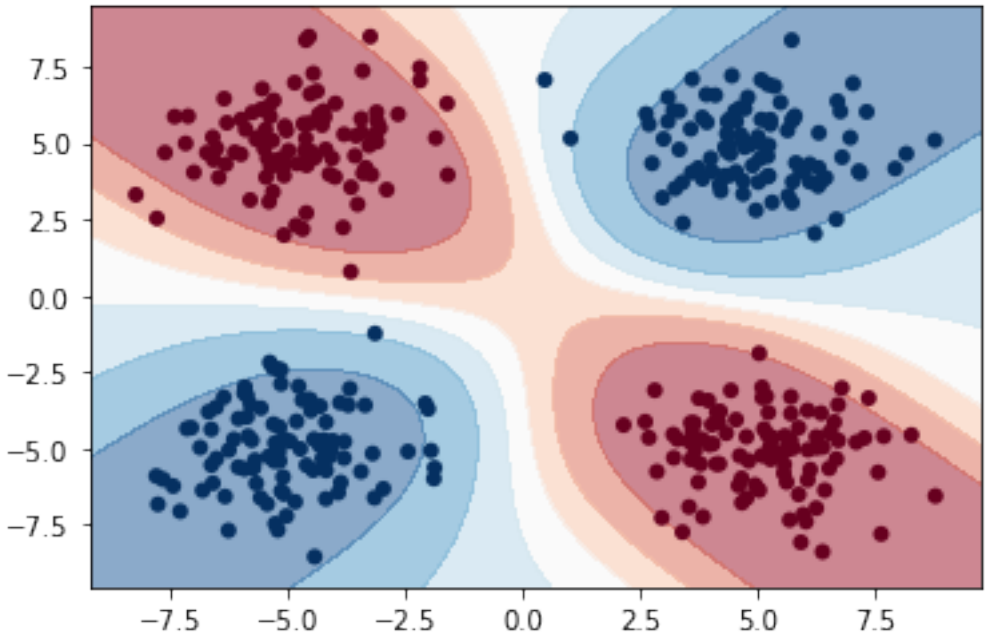
```
400/400 [==============================] - 0s 76us/step
```

[23]:
```
0.056273600310087206
```

2.2.12 Decision boundary for model423 on (test set)

[24]:
```
plot_decision_boundary(X2, y2, model423, cmap='RdBu')
```

[24]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73812365f8>)
```



2.3 c.

It looks pretty obvious that by increasing the number of nodes inside the layer we get better scores per model and also better decision boundaries.

3 Question 4.3

3.1 a.

3.1.1 1. Runing the back propagation algorithm with 4 first layer nodes and lr=0.01 for 300 iterations

[25]:
```
model4311 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4311.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4311.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model4311.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4311.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
```

```
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

3.1.2 2. Runing the back propagation algorithm with 4 first layer nodes and lr=0.001 for 300 iterations

[26]:
```
model4312 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4312.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4312.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.001)
model4312.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4312.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

3.1.3 3. Runing the back propagation algorithm with 4 first layer nodes and lr=0.01 for 1000 iterations

[27]:
```
model4313 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4313.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4313.add(Dense(output_dim=1, **layer_kw))
```

```
sgd = SGD(lr=0.01)
model4313.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4313.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

3.1.4 4. Runing the back propagation algorithm with 4 first layer nodes and lr=0.001 for 1000 iterations

[28]:
```
model4314 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4314.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4314.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.001)
model4314.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4314.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

3.2 b.

3.2.1 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (train set)

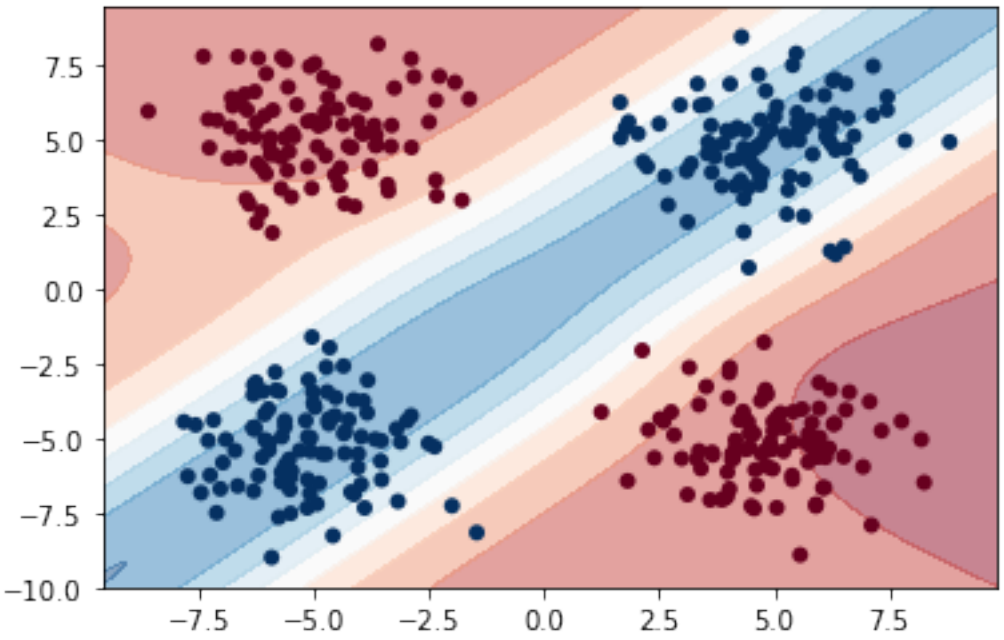
[29]:
```
model4311.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 279us/step
```

[29]:
```
0.5470116567611695
```

[30]:
```
plot_decision_boundary(X1, y1, model4311, cmap='RdBu')
```

[30]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e0105da0>)
```



3.2.2 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (test set)

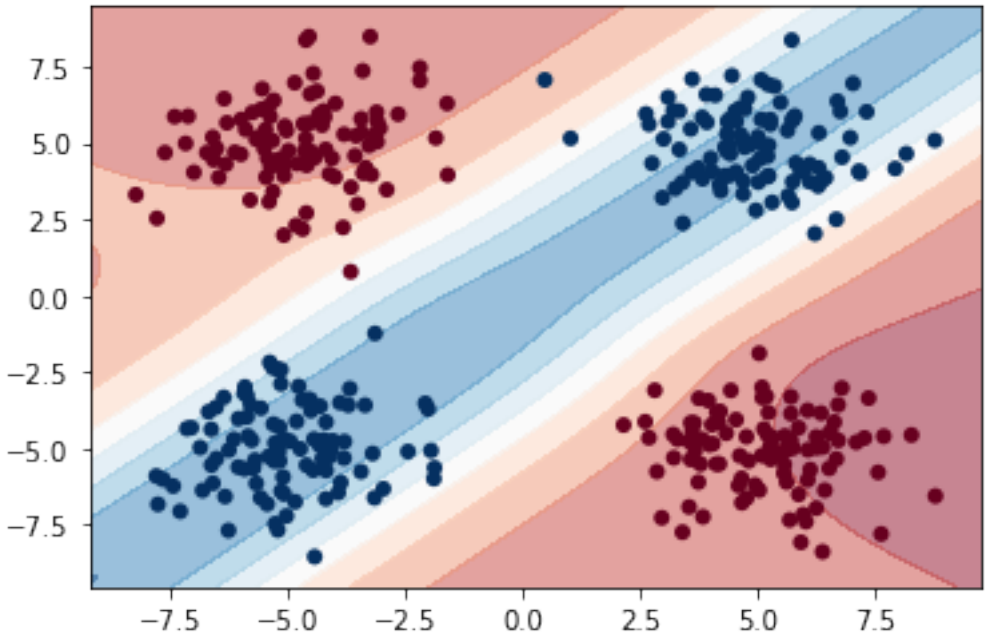
[31]:
```
model4311.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 83us/step
```

[31]:
```
0.5454028224945069
```

[32]:
```
plot_decision_boundary(X2, y2, model4311, cmap='RdBu')
```

[32]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e43b7c88>)
```



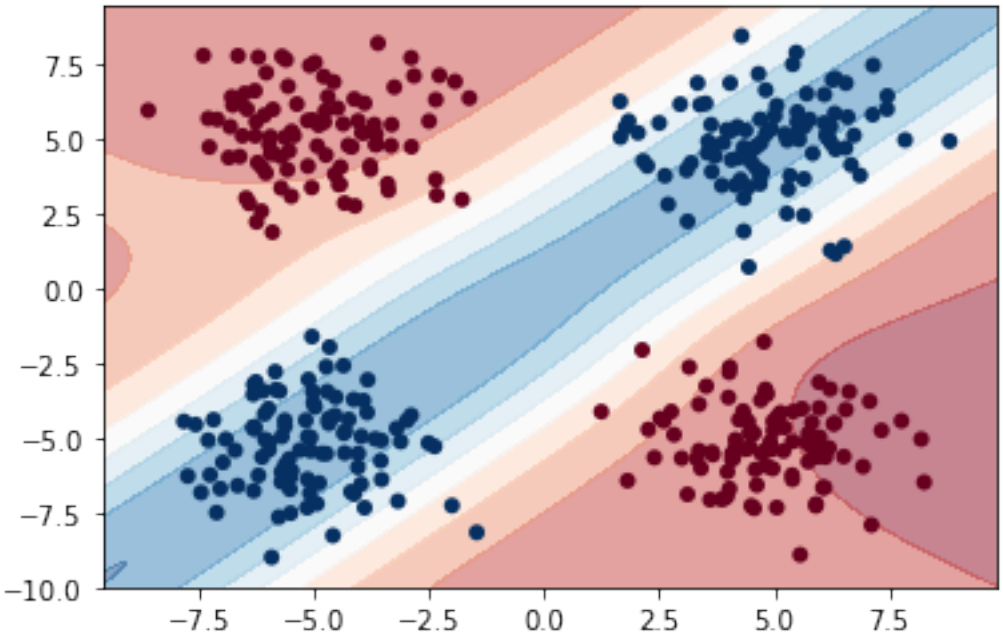
[33]:
```
model4311.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 75us/step
```

[33]:
```
0.5470116567611695
```

[34]:
```
plot_decision_boundary(X1, y1, model4311, cmap='RdBu')
```

[34]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380d626d8>)
```



3.2.3 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (test set)

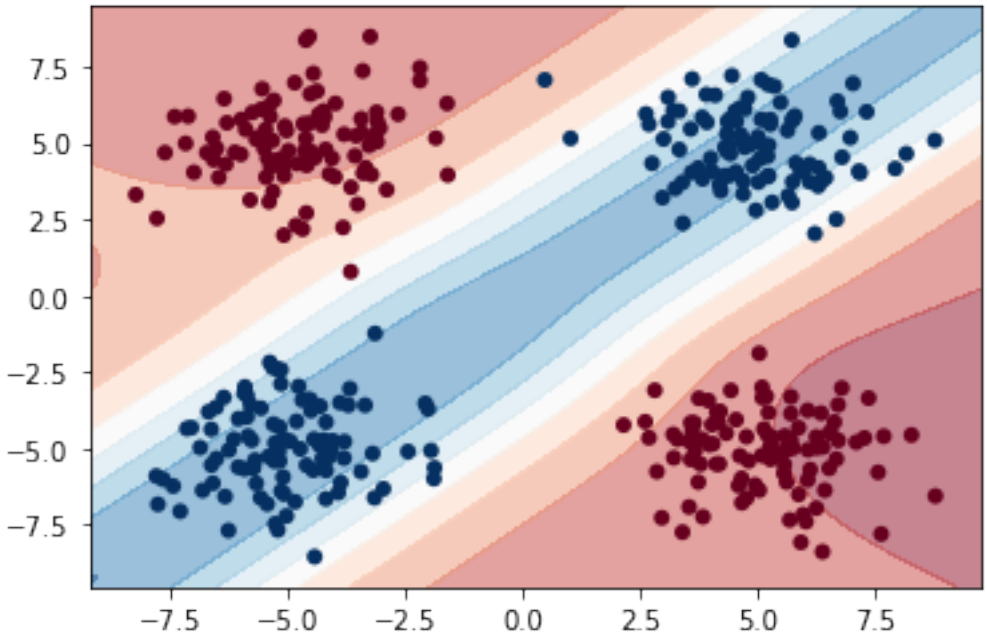
[35]:
```
model4311.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 80us/step
```

[35]:
```
0.5454028224945069
```

[36]:
```
plot_decision_boundary(X2, y2, model4311, cmap='RdBu')
```

[36]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380cbf748>)
```



3.2.4 Evaluating the model4312 with lr=0.001 and 4 nodes and 300 iterationson on (train set)

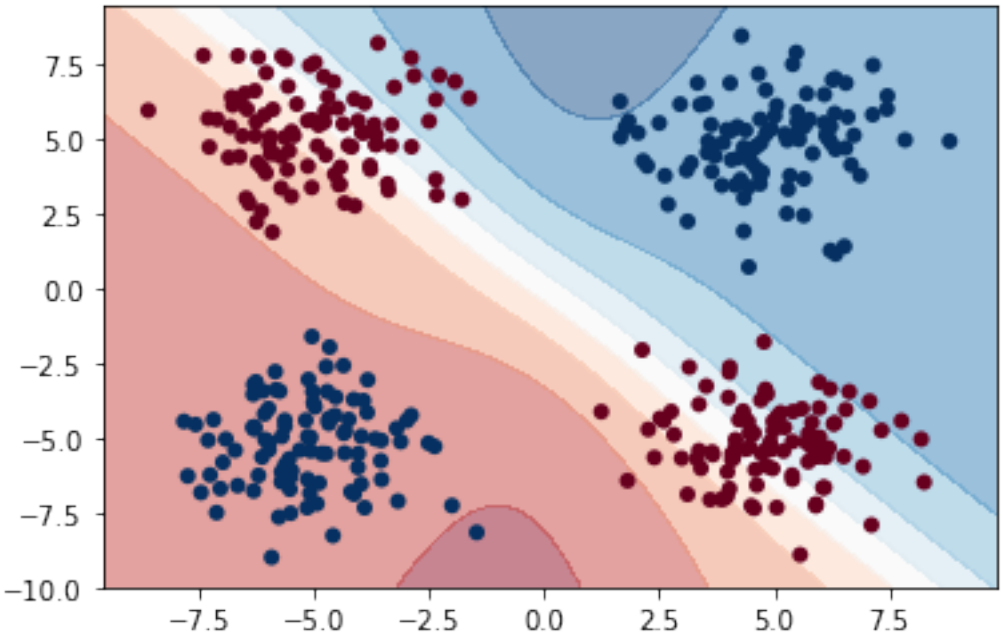
[37]:
```
model4312.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 295us/step
```

[37]:
```
0.6977813148498535
```

[38]:
```
plot_decision_boundary(X1, y1, model4312, cmap='RdBu')
```

[38]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380c39898>)
```



3.2.5 Evaluating the model4312 with lr=0.001 and 4 nodes and 300 iterationson on (test set)

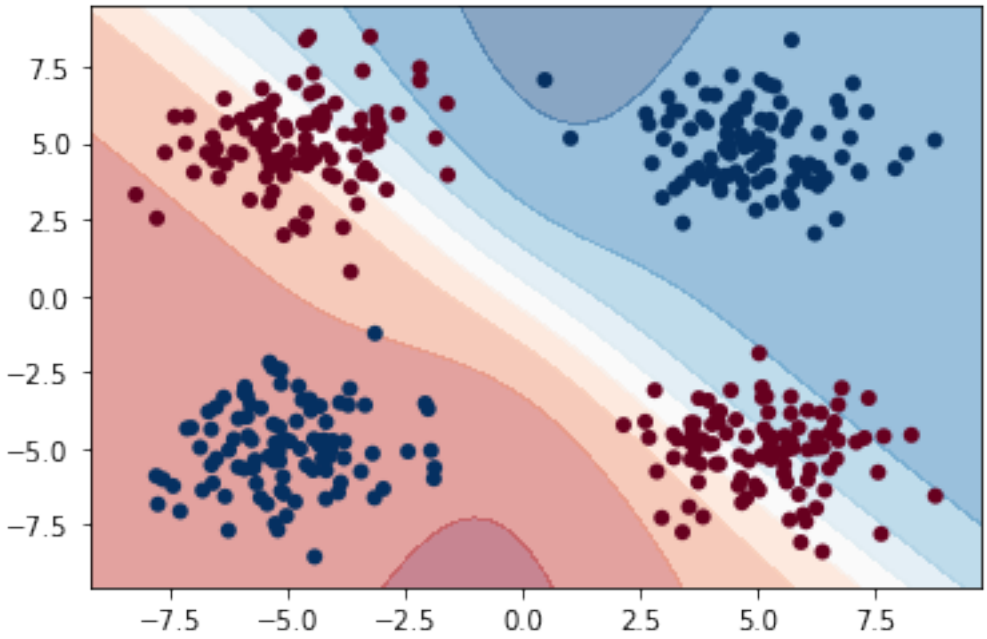
[39]:
```
model4312.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 82us/step
```

[39]:
```
0.7035510158538818
```

[40]:
```
plot_decision_boundary(X2, y2, model4312, cmap='RdBu')
```

[40]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380bb7a90>)
```



3.2.6 Evaluating the model4313 with lr=0.01 and 4 nodes and 1000 iterationson on (train set)

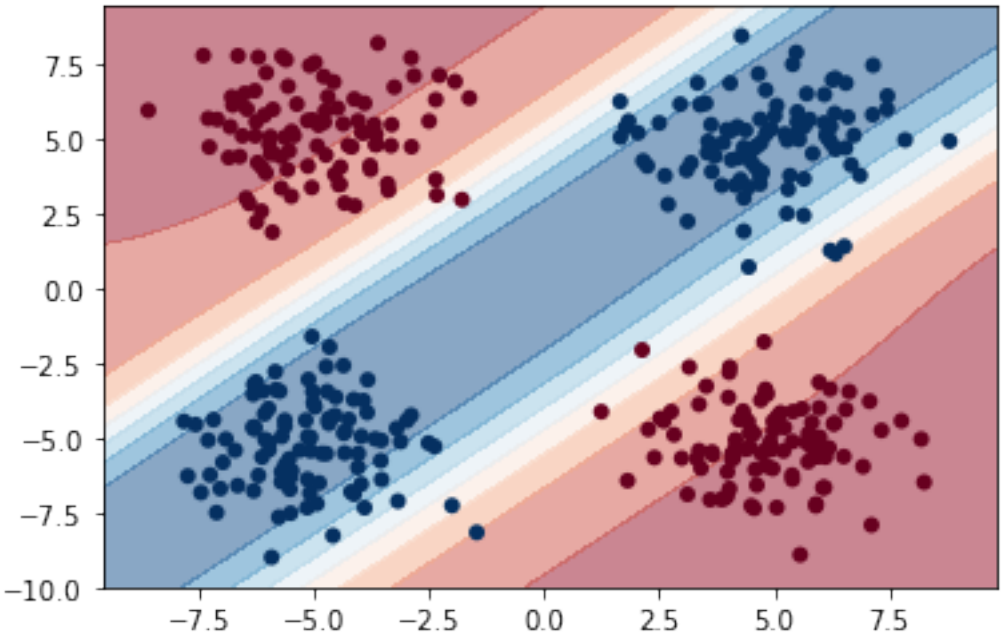
[41]:
```
model4313.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 313us/step
```

[41]:
```
0.2207092797756195
```

[42]:
```
plot_decision_boundary(X1, y1, model4313, cmap='RdBu')
```

[42]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380d3aa90>)
```



3.2.7 Evaluating the model4313 with lr=0.01 and 4 nodes and 1000 iterationson on (test set)

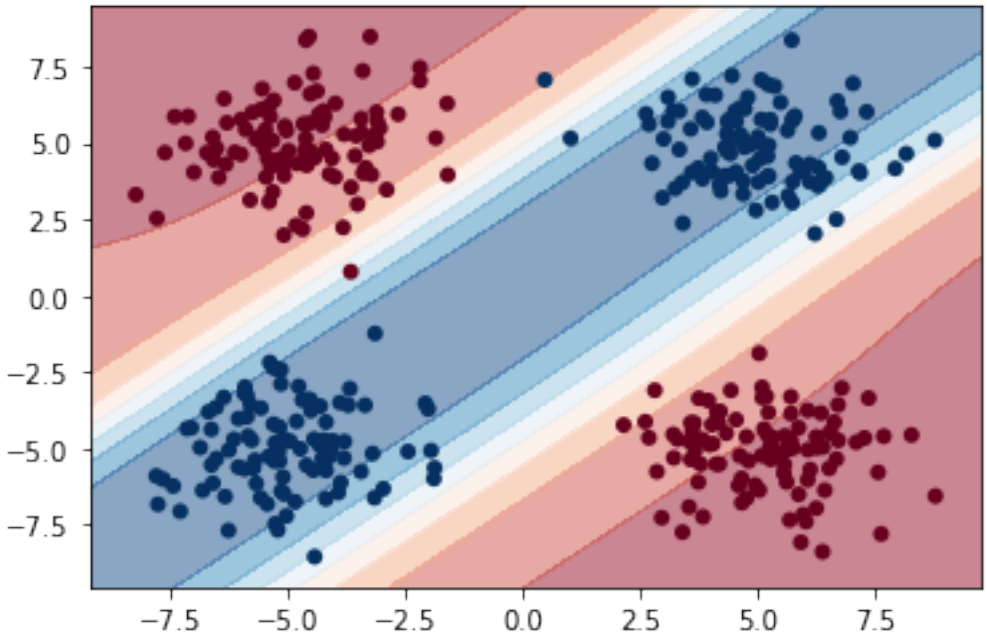
[43]:
```
model4313.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 81us/step
```

[43]:
```
0.21496012032032014
```

[44]:
```
plot_decision_boundary(X2, y2, model4313, cmap='RdBu')
```

[44]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380b365f8>)
```



3.2.8 Evaluating the model4314 with lr=0.001 and 4 nodes and 1000 iterationson on (train set)

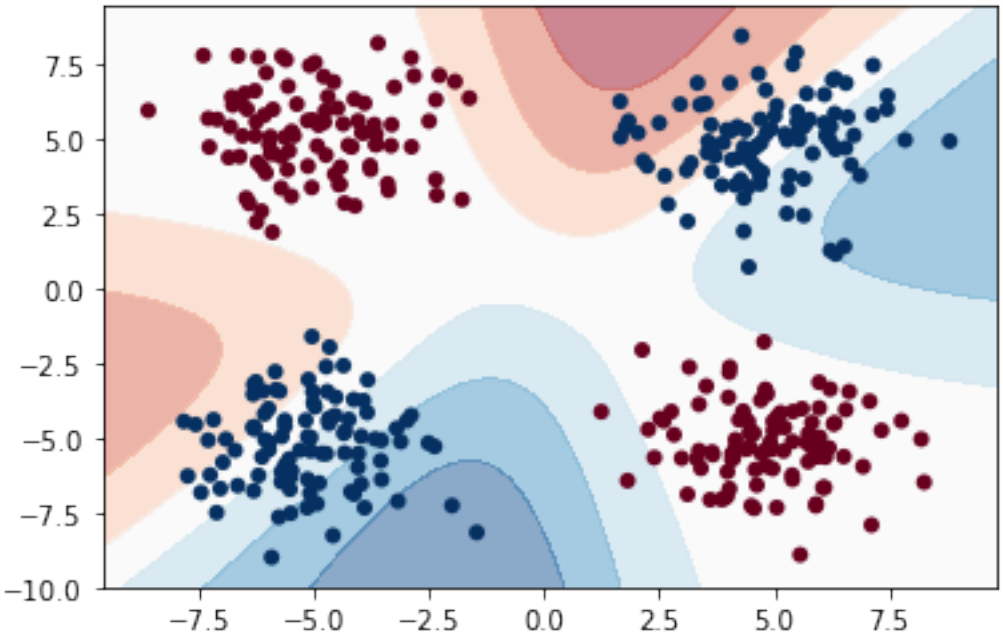
[45]:
```
model4314.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 294us/step
```

[45]:
```
0.6926731514930725
```

[46]:
```
plot_decision_boundary(X1, y1, model4314, cmap='RdBu')
```

[46]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380e35da0>)
```



3.2.9 Evaluating the model4314 with lr=0.001 and 4 nodes and 1000 iterationson on (test set)

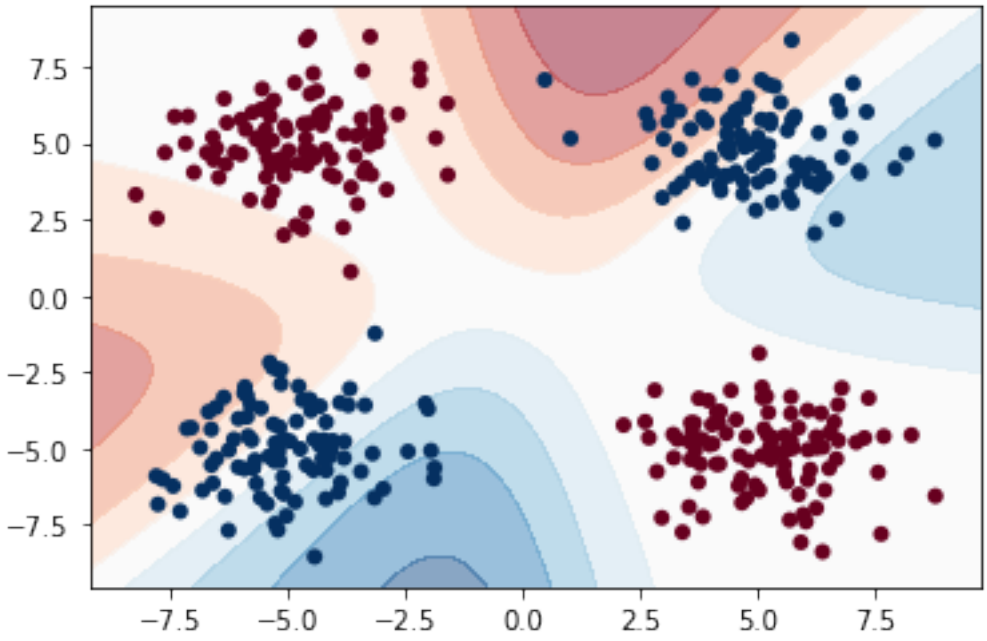
[47]:
```
model4314.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 73us/step
```

[47]:
```
0.6929089498519897
```

[48]:
```
plot_decision_boundary(X2, y2, model4314, cmap='RdBu')
```

[48]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380aae780>)
```



3.3 c.

By looking at the scores achived by different models and also comparing their decision boundaries, we can see the model4313 (lr=0.01 and iterations=1000) was the best among the others. Its loss was much less than the others and also it provides better boundaries.

Surrprisingly the next good model is model4311 (lr=0.01 and iterations=300) which does better job than model4314 with higher iterations and less learning rate.

Obviously with the same learning rates, models with higher itterations provided better results.

Also at the same iterrations, models with less learning rates, provided better outcomes.

Its worthy to note that it looks iterration parameter has more effect on the results than the learning rate.

4 Question 4.4

4.1 a.

4.1.1 Runing adaptive learning rate variation of the back propagation algorithm with lr=0.001 for 300 iterations on 4 nodes

[49]:
```
model441 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model441.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
```

```
model441.add(Dense(output_dim=1, **layer_kw))
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
model441.compile(optimizer=adam, loss='binary_crossentropy')
history = model441.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

4.2 b.

4.2.1 Evaluating the model441 with lr=0.001 and 4 nodes and 300 iterationson on (train set)

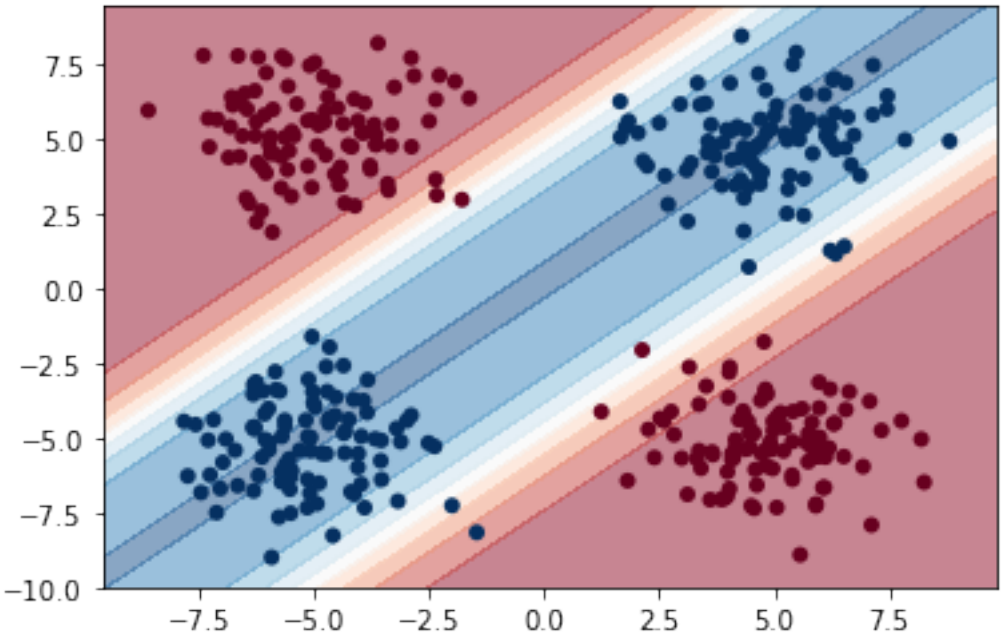
[50]:
```
model441.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 320us/step
```

[50]:
```
0.1576684844493866
```

[51]:
```
plot_decision_boundary(X1, y1, model441, cmap='RdBu')
```

[51]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380e70710>)
```



4.2.2 Evaluating the model441 with lr=0.001 and 4 nodes and 300 iterationson on (test set)

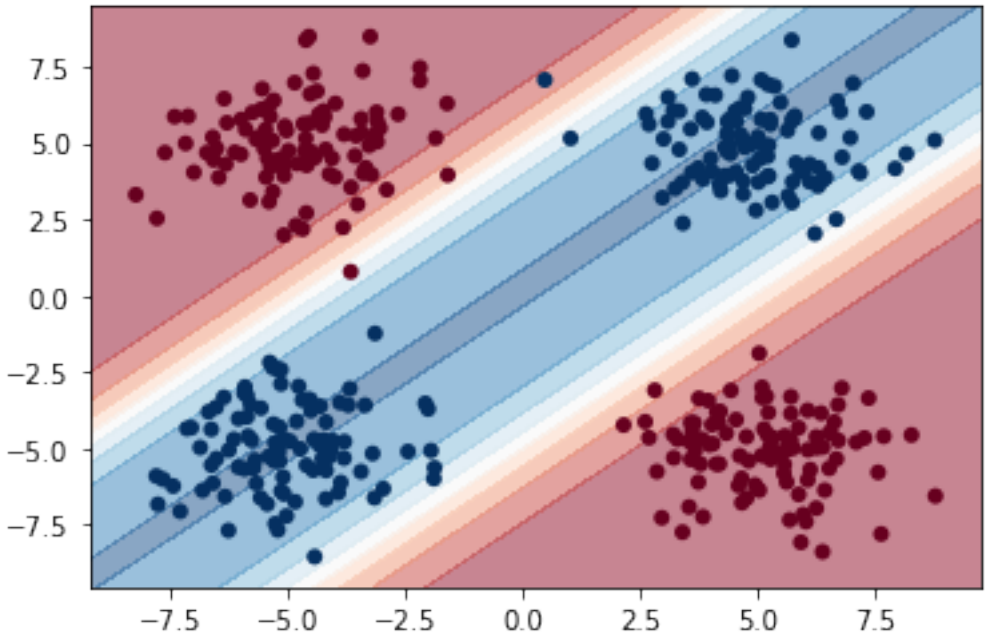
[52]:
```
model441.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 85us/step
```

[52]:
```
0.15056105315685273
```

[53]:
```
plot_decision_boundary(X2, y2, model441, cmap='RdBu')
```

[53]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380839780>)
```



4.3 c.

Results for this model are much better than the standard back propagation algorithm.

For train set, error in the standard model was 0.7189758801460266 and by using adaptive learning rate it decreases to 0.2106320881843567.

For test set, error in the standard model was 0.7277438926696778 and by using adaptive learning rate it decreases to 0.2106320881843567.

Also by looking at the decision boundaries we can see much better performance for adaptive variation.

5 Question 4.5

[0]:
```
X1T=X1
y1T=y1
X2T=X2
y2T=y2
X1=X3
y1=y3
X2=X4
y2=y4
```

5.0.1 Runinig the back propagation with lr=0.01 and 2 nodes on with 1000 iterations

[55]:
```
model421 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model421.add(Dense(output_dim=2, input_shape=(2, ), **layer_kw))
model421.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model421.compile(optimizer=sgd, loss='binary_crossentropy')
history = model421.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=2, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.2 Runinig the back propagation with lr=0.01 and 4 nodes on with 1000 iterations

[56]:
```
model422 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model422.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model422.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model422.compile(optimizer=sgd, loss='binary_crossentropy')
history = model422.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.3 Runinig the back propagation with lr=0.01 and 15 nodes on with 1000 iterations

[57]:
```
model423 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model423.add(Dense(output_dim=15, input_shape=(2, ), **layer_kw))
model423.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model423.compile(optimizer=sgd, loss='binary_crossentropy')
history = model423.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=15, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.4 Evaluating the model421 with lr=0.01 and 2 nodes and 1000 iterationson on (train set)

[58]:
```
model421.evaluate(x=X1,y=y1)
```

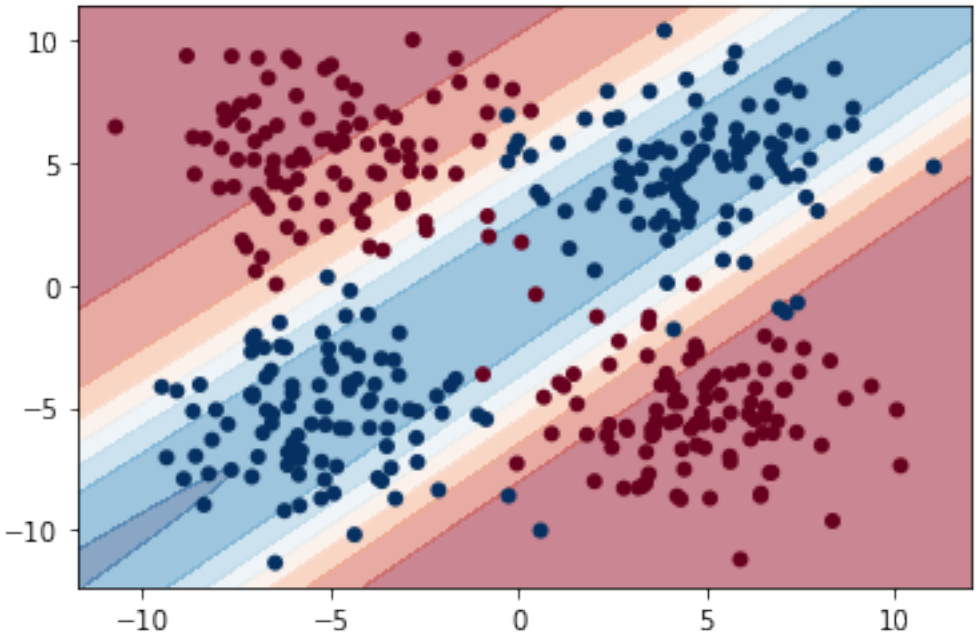
```
400/400 [==============================] - 0s 407us/step
```

[58]:
```
0.3977922785282135
```

5.0.5 Decision boundary for model421 on (train set)

[59]:
```
plot_decision_boundary(X1, y1, model421, cmap='RdBu')
```

[59]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f738073c710>)
```



5.0.6 Evaluating the model421 with lr=0.01 and 2 nodes and 1000 iterationson on (test set)

[60]:
```
model421.evaluate(x=X2,y=y2)
```

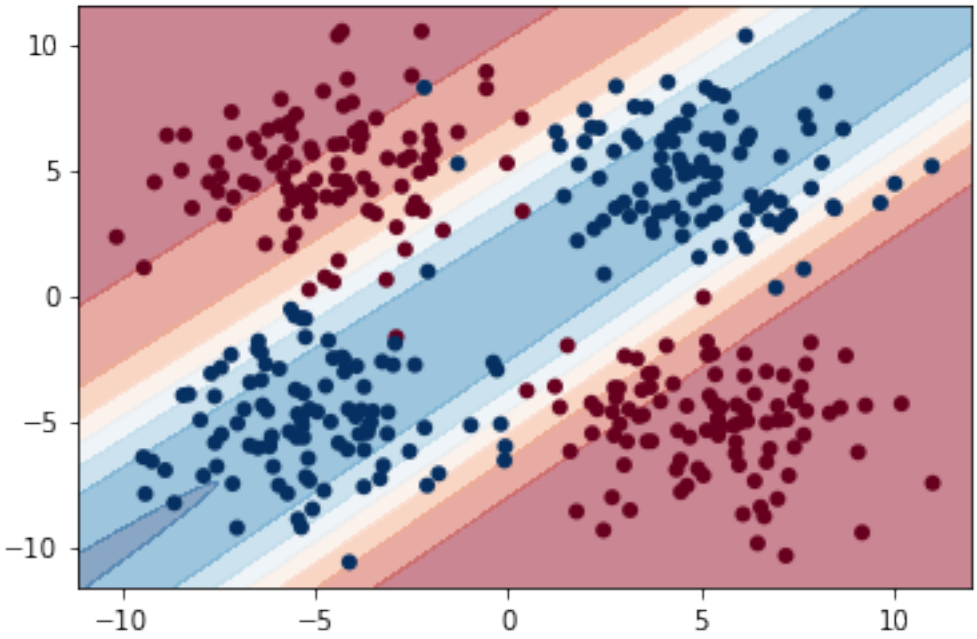
```
400/400 [==============================] - 0s 54us/step
```

[60]:
```
0.39156843185424806
```

5.0.7 Decision boundary for model421 on (test set)

[61]:
```
plot_decision_boundary(X2, y2, model421, cmap='RdBu')
```

[61]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f738048dd68>)
```



5.0.8 Evaluating the model422 with lr=0.01 and 4 nodes and 1000 iterationson on (train set)

[62]:
```
model422.evaluate(x=X1,y=y1)
```

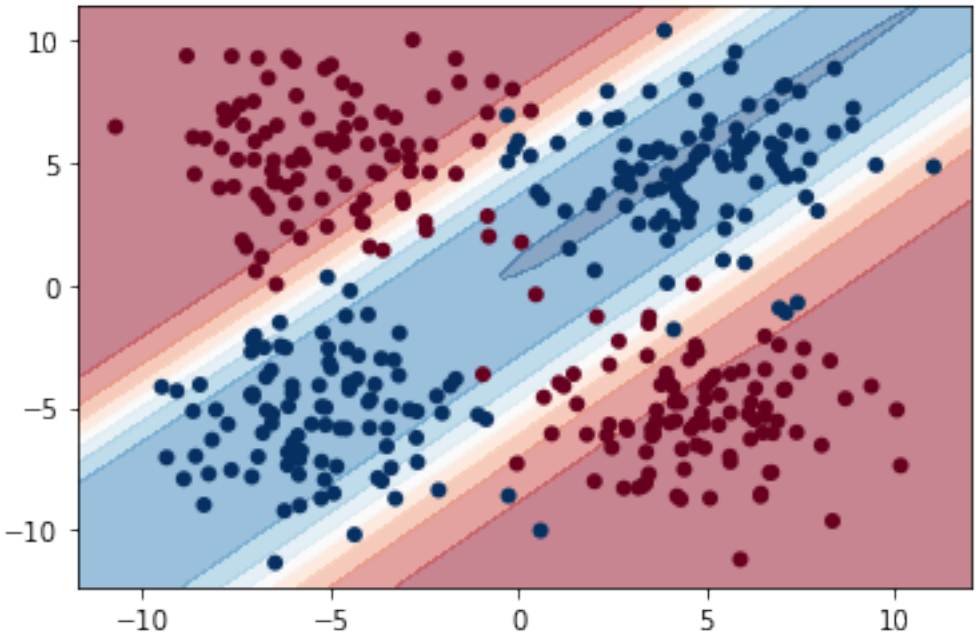
```
400/400 [==============================] - 0s 436us/step
```

[62]:
```
0.2721293413639069
```

5.0.9 Decision boundary for model422 on (train set)

[63]:
```
plot_decision_boundary(X1, y1, model422, cmap='RdBu')
```

[63]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380426a20>)
```



5.0.10 Evaluating the model422 with lr=0.01 and 4 nodes and 1000 iterationson on (test set)

[64]:
```
model422.evaluate(x=X2,y=y2)
```

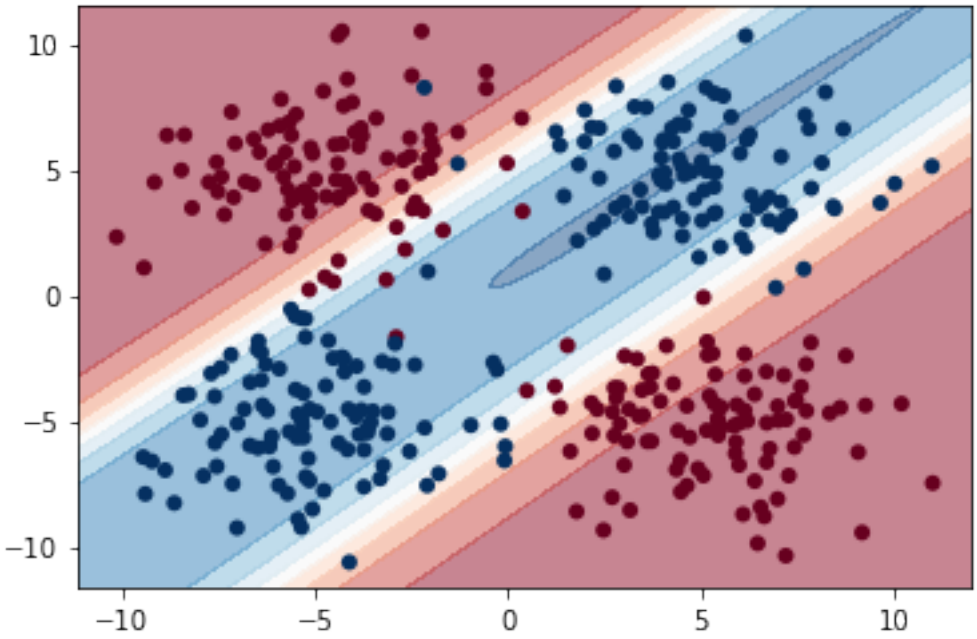
```
400/400 [==============================] - 0s 57us/step
```

[64]:
```
0.26075752258300783
```

5.0.11 Decision boundary for model422 on (test set)

[65]:
```
plot_decision_boundary(X2, y2, model422, cmap='RdBu')
```

[65]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f7380819400>)
```



5.0.12 Evaluating the model423 with lr=0.01 and 15 nodes and 1000 iterationson on (train set)

[66]:
```
model423.evaluate(x=X1,y=y1)
```

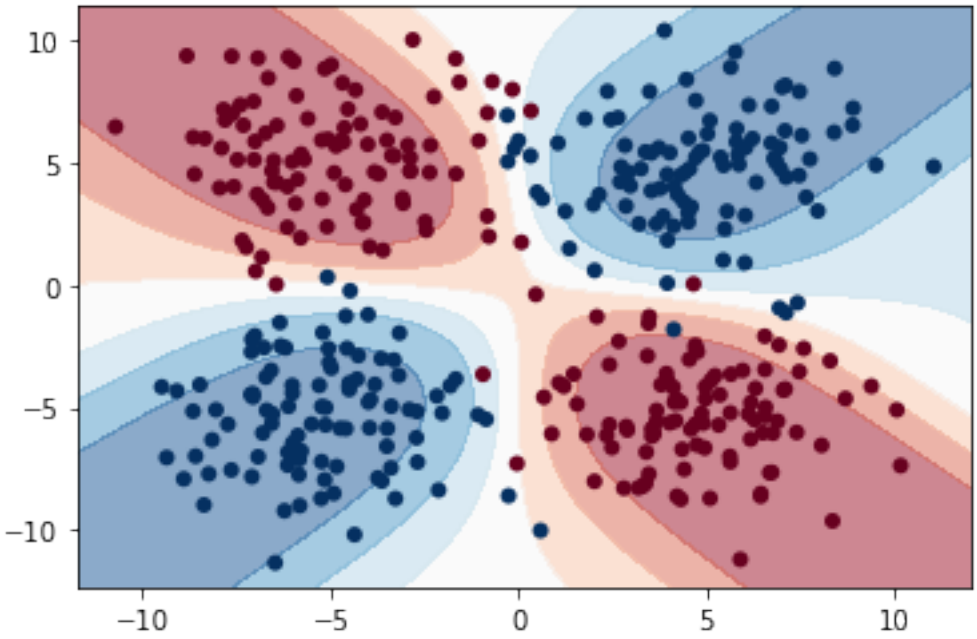
```
400/400 [==============================] - 0s 424us/step
```

[66]:
```
0.13639279574155808
```

5.0.13 Decision boundary for model423 on (train set)

[67]:
```
plot_decision_boundary(X1, y1, model423, cmap='RdBu')
```

[67]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73802cc278>)
```



5.0.14 Evaluating the model423 with lr=0.01 and 15 nodes and 1000 iterationson on (test set)

[68]:
```
model423.evaluate(x=X2,y=y2)
```

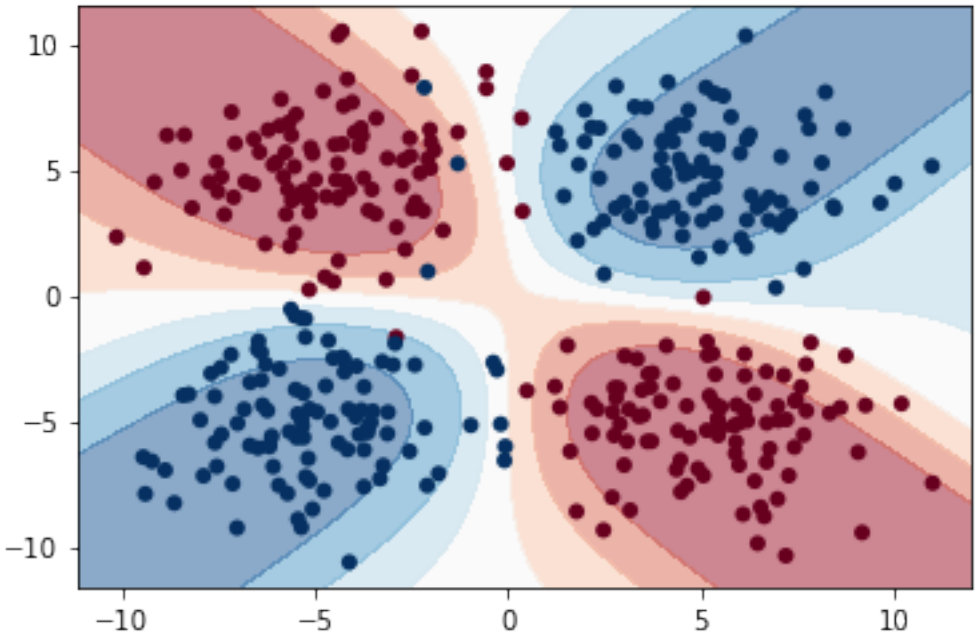
```
400/400 [==============================] - 0s 85us/step
```

[68]:
```
0.1327403622865677
```

5.0.15 Decision boundary for model423 on (test set)

[69]:
```
plot_decision_boundary(X2, y2, model423, cmap='RdBu')
```

[69]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73e43343c8>)
```



It looks pretty obvious that by increasing the number of nodes inside the layer we get better scores per model and also better decision boundaries.

5.0.16 Runing the back propagation algorithm with 4 first layer nodes and lr=0.01 for 300 iterations on

[70]:
```
model4311 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4311.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4311.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model4311.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4311.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.17 Runing the back propagation algorithm with 4 first layer nodes and lr=0.001 for 300 iterations on

[71]:
```
model4312 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4312.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4312.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.001)
model4312.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4312.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.18 Runing the back propagation algorithm with 4 first layer nodes and lr=0.01 for 1000 iterations on

[72]:
```
model4313 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4313.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4313.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.01)
model4313.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4313.fit(X1, y1, verbose=0, nb_epoch=1000)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
```

```
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.19 Runing the back propagation algorithm with 4 first layer nodes and lr=0.001 for 1000 iterations on

[73]:
```
model4314 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model4314.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model4314.add(Dense(output_dim=1, **layer_kw))
sgd = SGD(lr=0.001)
model4314.compile(optimizer=sgd, loss='binary_crossentropy')
history = model4314.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.20 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (train set)

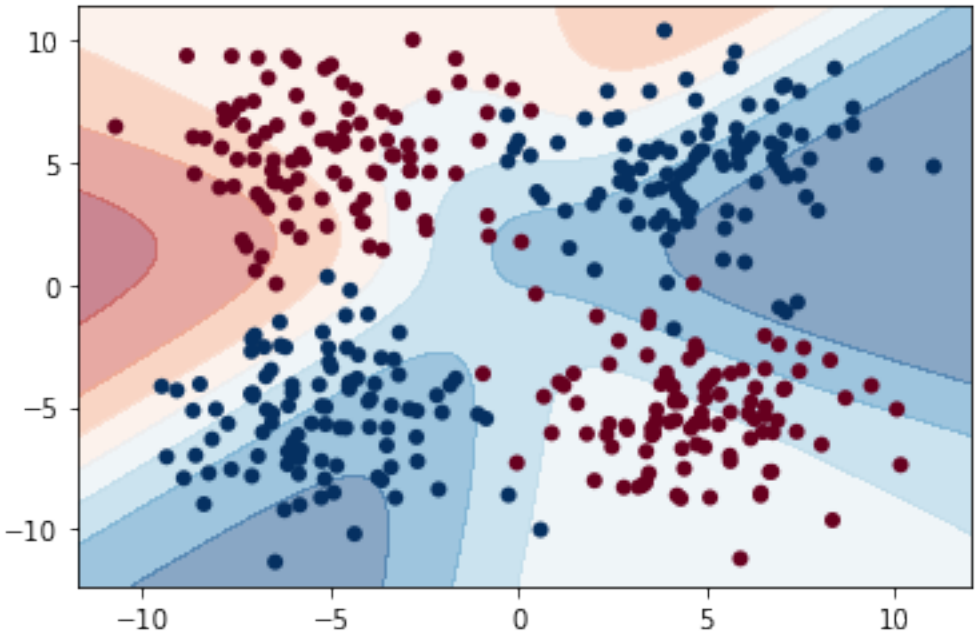
[74]:
```
model4311.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 533us/step
```

[74]:
```
0.6220700716972352
```

[75]:
```
plot_decision_boundary(X1, y1, model4311, cmap='RdBu')
```

[75]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f73804e9f60>)
```



5.0.21 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (test set)

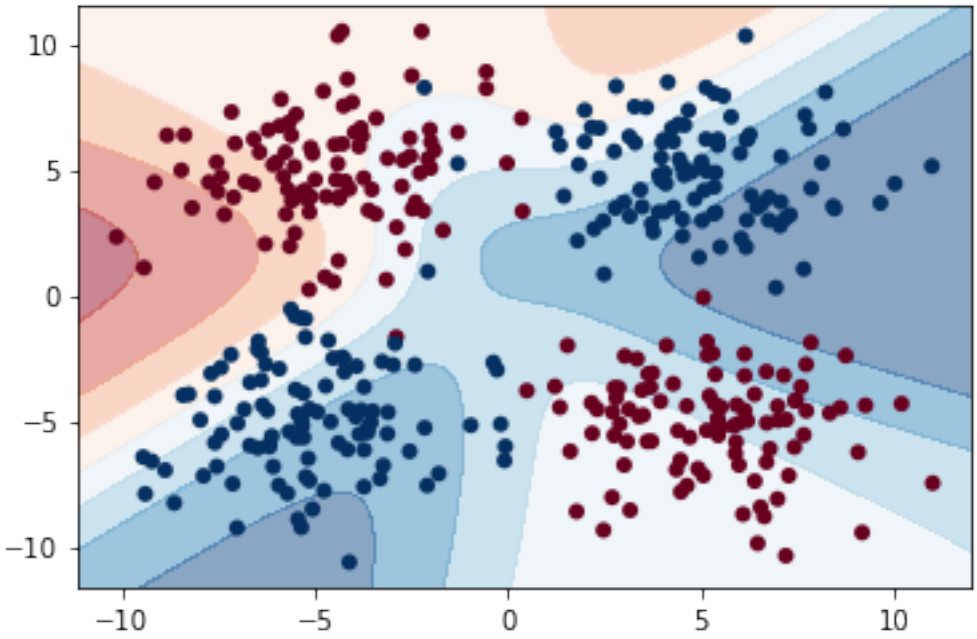
[76]:
```
model4311.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 65us/step
```

[76]:
```
0.6276660132408142
```

[77]:
```
plot_decision_boundary(X2, y2, model4311, cmap='RdBu')
```

[77]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737ff03048>)
```



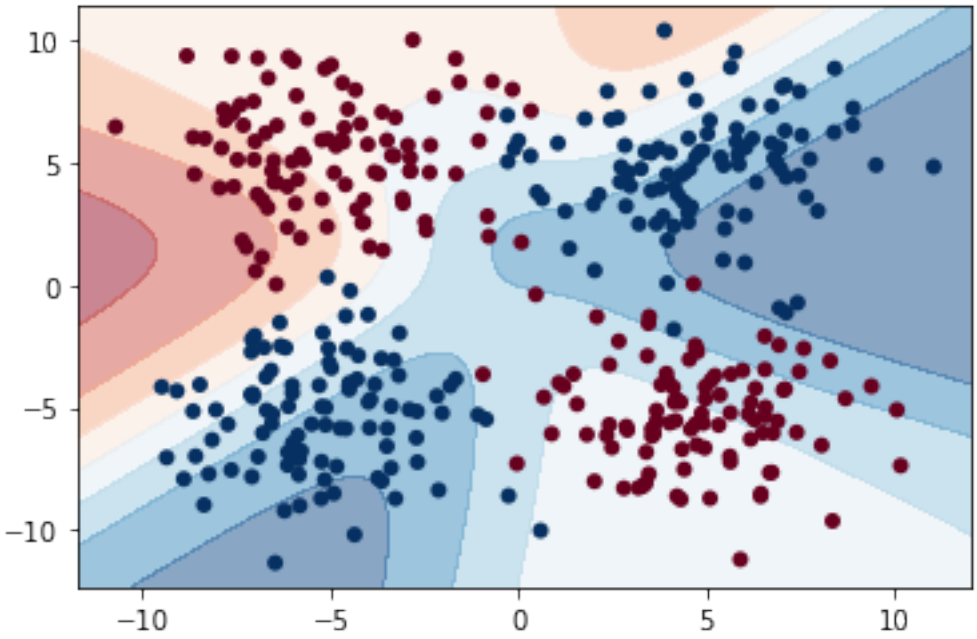
[78]:
```
model4311.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 63us/step
```

[78]:
```
0.6220700716972352
```

[79]:
```
plot_decision_boundary(X1, y1, model4311, cmap='RdBu')
```

[79]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fe77e48>)
```



5.0.22 Evaluating the model4311 with lr=0.01 and 4 nodes and 300 iterationson on (test set)

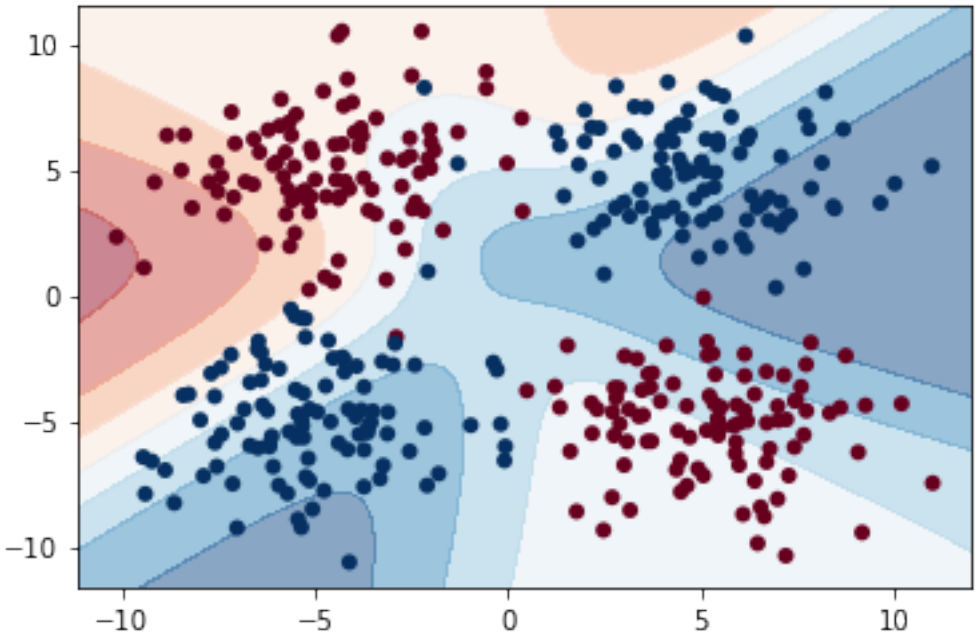
[80]:
```
model4311.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 69us/step
```

[80]:
```
0.6276660132408142
```

[81]:
```
plot_decision_boundary(X2, y2, model4311, cmap='RdBu')
```

[81]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737ff18588>)
```



5.0.23 Evaluating the model4312 with lr=0.001 and 4 nodes and 300 iterationson on (train set)

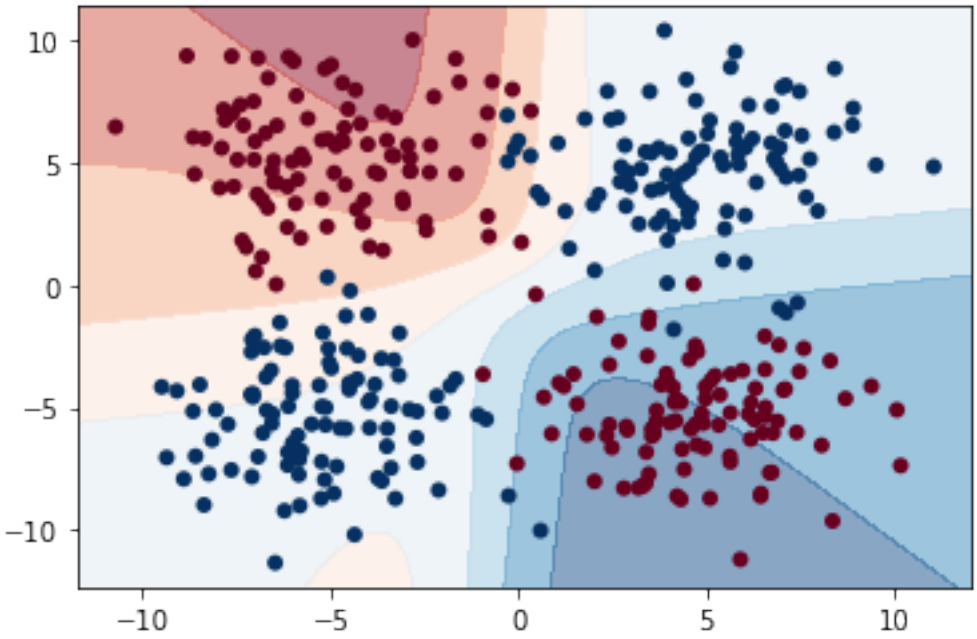
[82]:
```
model4312.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 584us/step
```

[82]:
```
0.7071782326698304
```

[83]:
```
plot_decision_boundary(X1, y1, model4312, cmap='RdBu')
```

[83]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737ff56c50>)
```



5.0.24 Evaluating the model4312 with lr=0.001 and 4 nodes and 300 iterationson on (test set)

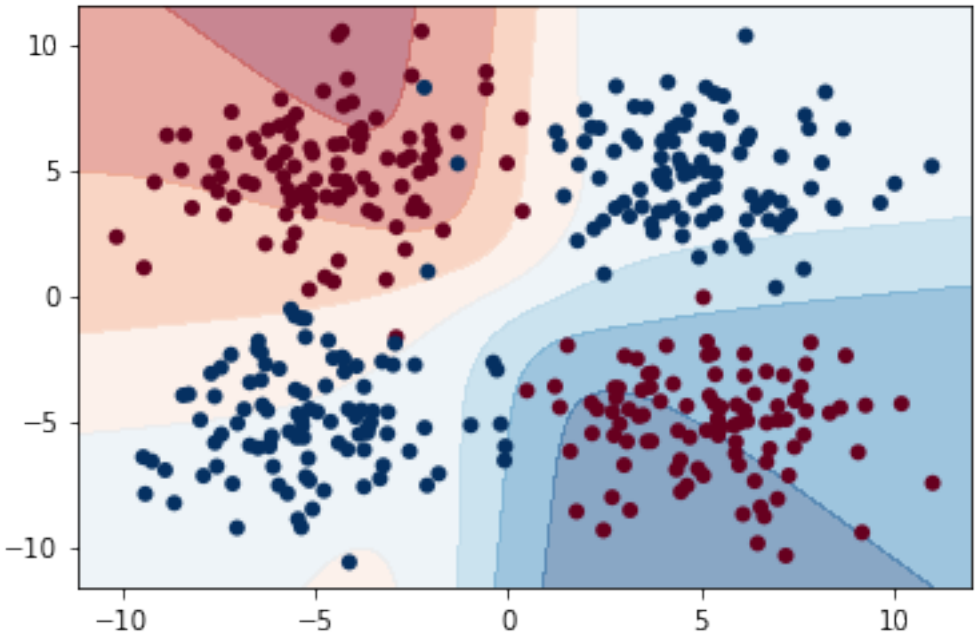
[84]:
```
model4312.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 71us/step
```

[84]:
```
0.706860761642456
```

[85]:
```
plot_decision_boundary(X2, y2, model4312, cmap='RdBu')
```

[85]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fd445c0>)
```



5.0.25 Evaluating the model4313 with lr=0.01 and 4 nodes and 1000 iterationson on (train set)

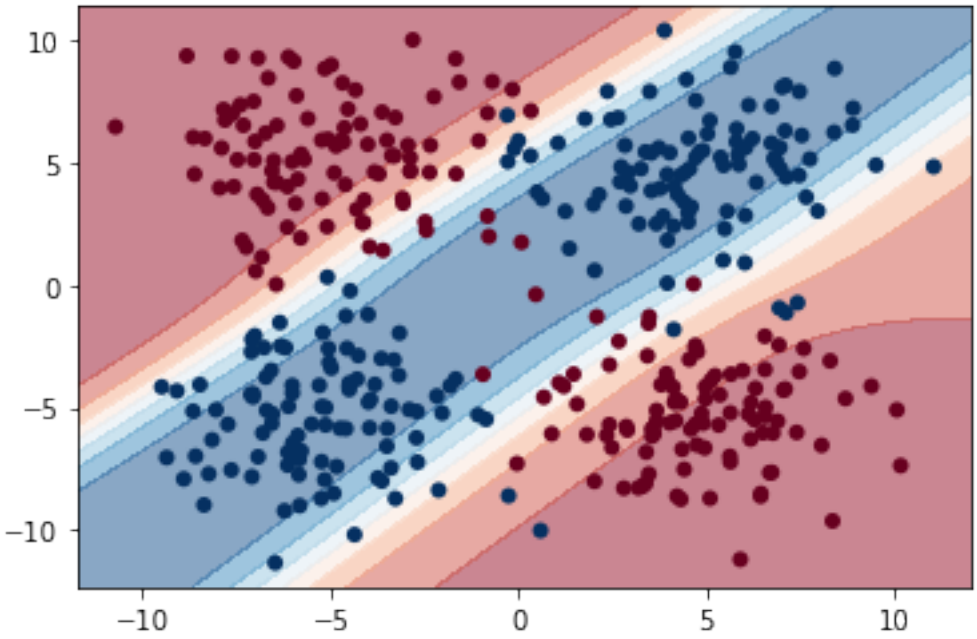
[86]:
```
model4313.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 578us/step
```

[86]:
```
0.28562148451805114
```

[87]:
```
plot_decision_boundary(X1, y1, model4313, cmap='RdBu')
```

[87]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fcb78d0>)
```



5.0.26 Evaluating the model4313 with lr=0.01 and 4 nodes and 1000 iterationson on (test set)

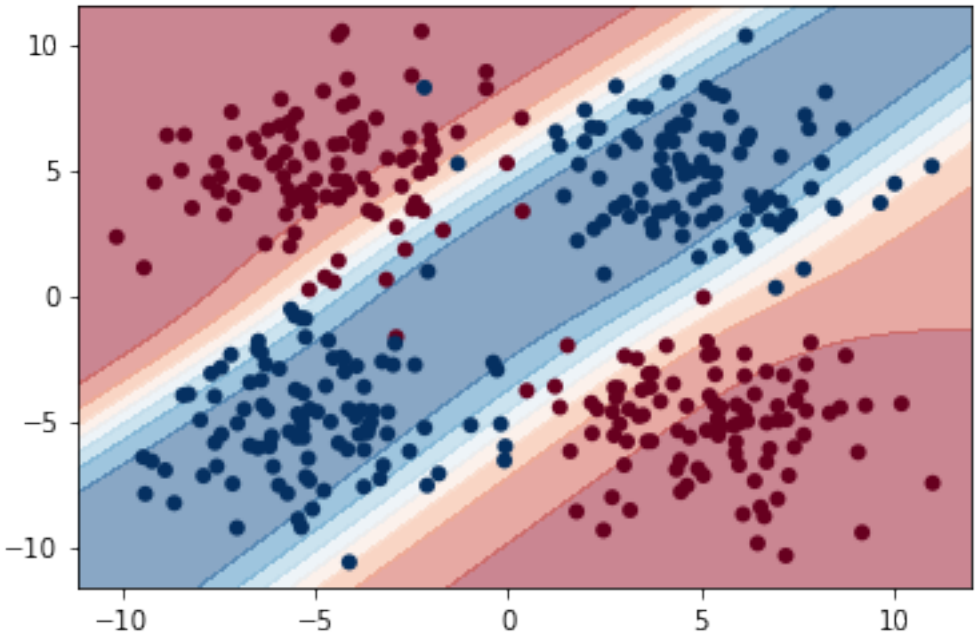
[88]:
```
model4313.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 83us/step
```

[88]:
```
0.27493416786193847
```

[89]:
```
plot_decision_boundary(X2, y2, model4313, cmap='RdBu')
```

[89]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fd64be0>)
```



5.0.27 Evaluating the model4314 with lr=0.001 and 4 nodes and 1000 iterationson on (train set)

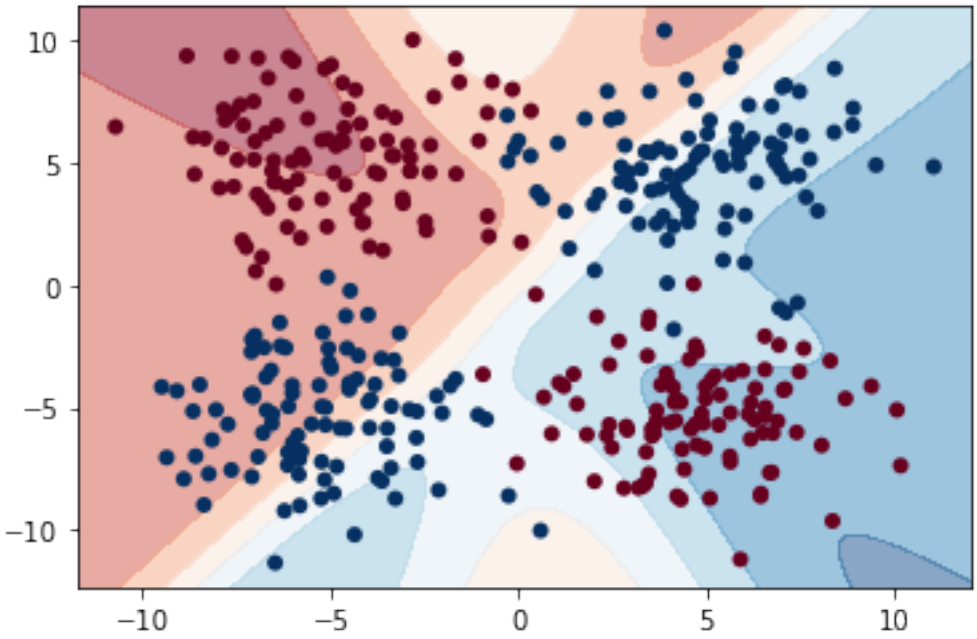
[90]:
```
model4314.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 550us/step
```

[90]:
```
0.6947650074958801
```

[91]:
```
plot_decision_boundary(X1, y1, model4314, cmap='RdBu')
```

[91]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fd44978>)
```



5.0.28 Evaluating the model4314 with lr=0.001 and 4 nodes and 1000 iterationson on (test set)

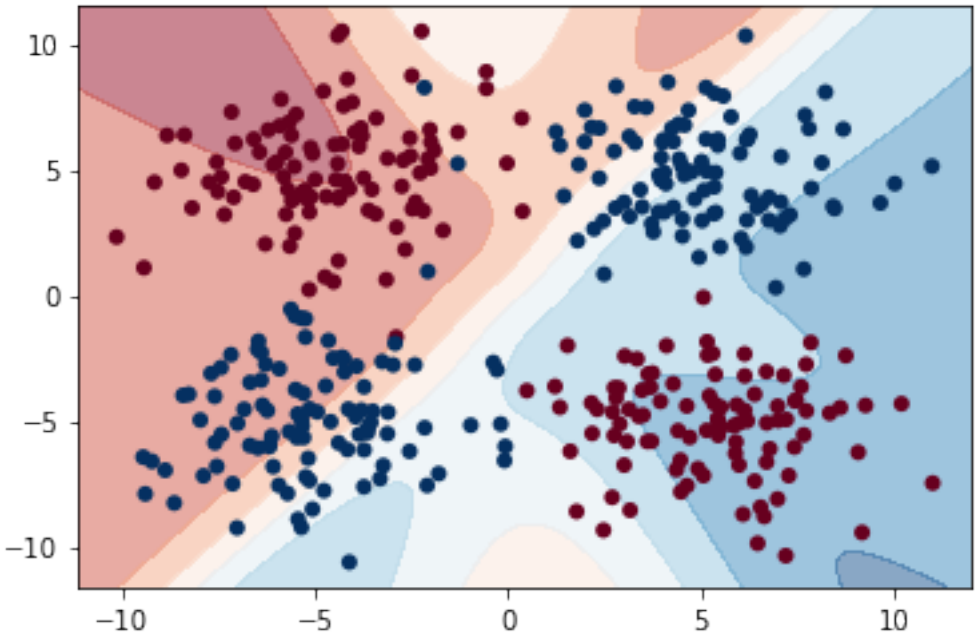
[92]:
```
model4314.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 72us/step
```

[92]:
```
0.6966988658905029
```

[93]:
```
plot_decision_boundary(X2, y2, model4314, cmap='RdBu')
```

[93]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737fb7d6d8>)
```



By looking at the scores achived by different models and also comparing their decision boundaries, we can see the model4313 (lr=0.01 and iterations=1000) was the best among the others. Its loss was much less than the others and also it provides better boundaries.

Surrprisingly the next good model is model4311 (lr=0.01 and iterations=300) which does better job than model4314 with higher iterations and less learning rate.

Obviously with the same learning rates, models with higher itterations provided better results.

Also at the same iterrations, models with less learning rates, provided better outcomes.

Its worthy to note that it looks iterration parameter has more effect on the results than the learning rate.

5.0.29 Runing adaptive learning rate variation of the back propagation algorithm with lr=0.001 for 300 iterations on 4 nodes

[94]:
```
model441 = Sequential()

# kwarg dict for convenience
layer_kw = dict(activation='sigmoid', init='glorot_uniform')

# Add layers to our model
model441.add(Dense(output_dim=4, input_shape=(2, ), **layer_kw))
model441.add(Dense(output_dim=1, **layer_kw))
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
model441.compile(optimizer=adam, loss='binary_crossentropy')
history = model441.fit(X1, y1, verbose=0, nb_epoch=300)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_shape=(2,),
```

```
activation="sigmoid", units=4, kernel_initializer="glorot_uniform")`
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(activation="sigmoid",
units=1, kernel_initializer="glorot_uniform")`

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning:
The `nb_epoch` argument in `fit` has been renamed `epochs`.
  # This is added back by InteractiveShellApp.init_path()
```

5.0.30 Evaluating the model441 with lr=0.001 and 4 nodes and 300 iterationson on (train set)

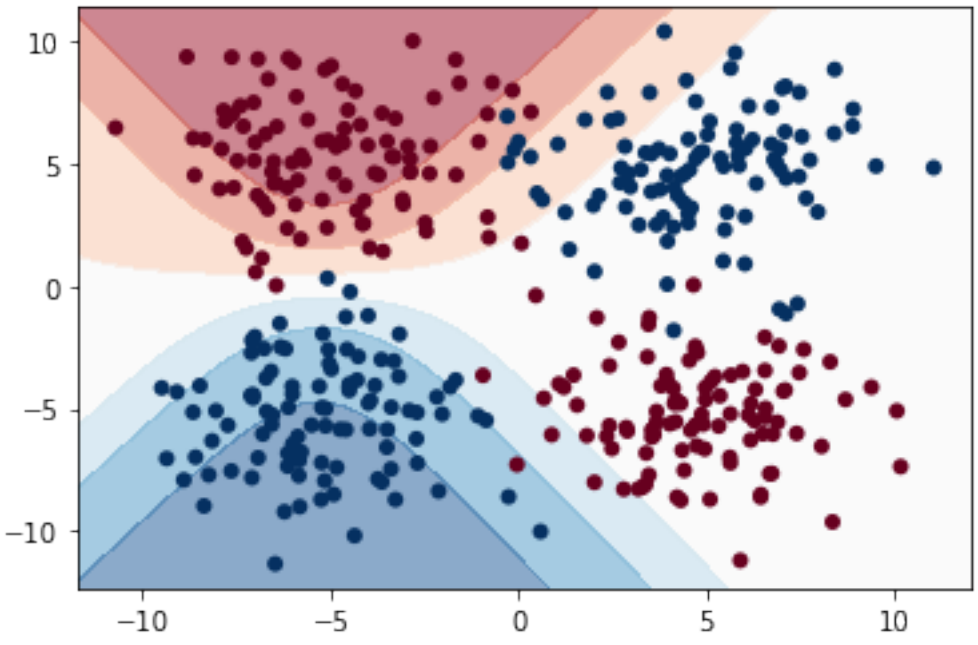
[95]:
```
model441.evaluate(x=X1,y=y1)
```

```
400/400 [==============================] - 0s 565us/step
```

[95]:
```
0.4647888207435608
```

[96]:
```
plot_decision_boundary(X1, y1, model441, cmap='RdBu')
```

[96]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737faf18d0>)
```



5.0.31 Evaluating the model441 with lr=0.001 and 4 nodes and 300 iterationson on (test set)

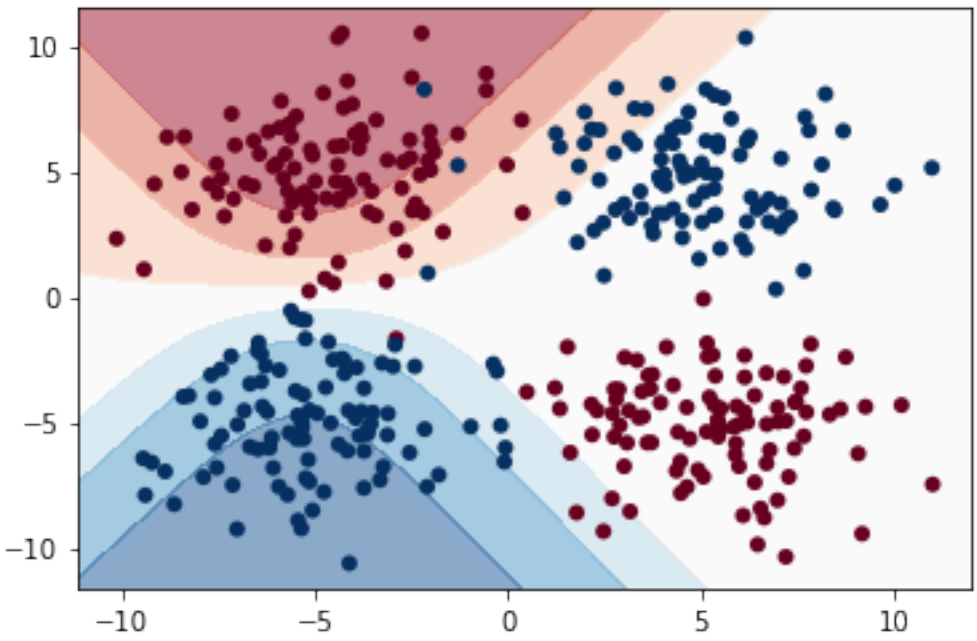
[97]:
```
model441.evaluate(x=X2,y=y2)
```

```
400/400 [==============================] - 0s 90us/step
```

[97]:
```
0.46908896446228027
```

[98]:
```
plot_decision_boundary(X2, y2, model441, cmap='RdBu')
```

[98]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f737f9731d0>)
```



Results for this model are much better than the standard back propagation algorithm.

For train set, error in the standard model was 0.6992755627632141 and by using adaptive learning rate it decreases to 0.25804547011852264.

For test set, error in the standard model was 0.6972246980667114 and by using adaptive learning rate it decreases to 0.25367753744125365.

Also by looking at the decision boundaries we can see much better performance for adaptive variation.

6 Question 4.6

6.1 a.

[0]:
```
X1=X1T
y1=y1T
X2=X2T
y2=y2T
from sklearn import svm
```

6.1.1 SVM classifier with C=1, sigma=0.5 on for train

[0]:
```
SVMC1S05=svm.SVC(kernel='rbf',gamma=0.5,C=1,tol=0.001).fit(X1,y1)
```

6.1.2 SVM classifier with C=100, sigma=0.5 on for train

[0]:
```
SVMC100S05=svm.SVC(kernel='rbf',gamma=0.5,C=100,tol=0.001).fit(X1,y1)
```

6.1.3 SVM classifier with C=1000, sigma=0.5 on for train

[0]:
```
SVMC1000S05=svm.SVC(kernel='rbf',gamma=0.5,C=1000,tol=0.001).fit(X1,y1)
```

6.1.4 SVM classifier with C=1, sigma=1 on for train

[0]:
```
SVMC1S1=svm.SVC(kernel='rbf',gamma=1,C=1,tol=0.001).fit(X1,y1)
```

6.1.5 SVM classifier with C=100, sigma=1 on for train

[0]:
```
SVMC100S1=svm.SVC(kernel='rbf',gamma=1,C=100,tol=0.001).fit(X1,y1)
```

6.1.6 SVM classifier with C=1000, sigma=1 on for train

[0]:
```
SVMC1000S1=svm.SVC(kernel='rbf',gamma=1,C=1000,tol=0.001).fit(X1,y1)
```

6.1.7 SVM classifier with C=1, sigma=2 on for train

[0]:
```
SVMC1S2=svm.SVC(kernel='rbf',gamma=2,C=1,tol=0.001).fit(X1,y1)
```

6.1.8 SVM classifier with C=100, sigma=2 on for train

[0]:
```
SVMC100S2=svm.SVC(kernel='rbf',gamma=2,C=100,tol=0.001).fit(X1,y1)
```

6.1.9 SVM classifier with C=1000, sigma=2 on for train

[0]:
```
SVMC1000S2=svm.SVC(kernel='rbf',gamma=2,C=1000,tol=0.001).fit(X1,y1)
```

6.1.10 SVM classifier with C=1, sigma=4 on for train

[0]:
```
SVMC1S4=svm.SVC(kernel='rbf',gamma=4,C=1,tol=0.001).fit(X1,y1)
```

6.1.11 SVM classifier with C=100, sigma=4 on for train

[0]:
```
SVMC100S4=svm.SVC(kernel='rbf',gamma=4,C=100,tol=0.001).fit(X1,y1)
```

6.1.12 SVM classifier with C=1000, sigma=4 on for train

[0]:
```
SVMC1000S4=svm.SVC(kernel='rbf',gamma=4,C=1000,tol=0.001).fit(X1,y1)
```

6.2 b.

6.2.1 Evaluating SVMC1S05 on (train set)

[112]:
```
SVMC1S05.score(X1,y1)
```

[112]:
```
1.0
```

6.2.2 Evaluating SVMC1S05 on (test set)

[113]:
```
SVMC1S05.score(X2,y2)
```

[113]:
```
1.0
```

6.2.3 Evaluating SVMC100S05 on (train set)

[114]:
```
SVMC100S05.score(X1,y1)
```

[114]:
```
1.0
```

6.2.4 Evaluating SVMC100S05 on (test set)

[115]:
```
SVMC100S05.score(X2,y2)
```

[115]:
```
1.0
```

6.2.5 Evaluating SVMC1000S05 on (train set)

[116]:
```
SVMC1000S05.score(X1,y1)
```

[116]:
```
1.0
```

6.2.6 Evaluating SVMC1000S05 on (test set)

[117]:
```
SVMC1000S05.score(X2,y2)
```

[117]:
```
1.0
```

6.2.7 Evaluating SVMC1S1 on (train set)

[118]:
```
SVMC1S1.score(X1,y1)
```

[118]:
```
1.0
```

6.2.8 Evaluating SVMC1S1 on (test set)

[119]:
```
SVMC1S1.score(X2,y2)
```

[119]:
```
1.0
```

6.2.9 Evaluating SVMC100S1 on (train set)

[120]:
```
SVMC100S1.score(X1,y1)
```

[120]:
```
1.0
```

6.2.10 Evaluating SVMC100S1 on (test set)

[121]:
```
SVMC100S1.score(X2,y2)
```

[121]:
```
1.0
```

6.2.11 Evaluating SVMC1000S1 on (train set)

[122]:
```
SVMC1000S1.score(X1,y1)
```

[122]:
```
1.0
```

6.2.12 Evaluating SVMC1000S1 on (test set)

[123]:
```
SVMC1000S1.score(X2,y2)
```

[123]:
```
1.0
```

6.2.13 Evaluating SVMC1S2 on (train set)

[124]:
```
SVMC1S2.score(X1,y1)
```

[124]:
```
1.0
```

6.2.14 Evaluating SVMC1S2 on (test set)

[125]:
```
SVMC1S2.score(X2,y2)
```

[125]:
```
0.995
```

6.2.15 Evaluating SVMC100S2 on (train set)

[126]:
```
SVMC100S2.score(X1,y1)
```

[126]:
```
1.0
```

6.2.16 Evaluating SVMC100S2 on (test set)

[127]:
```
SVMC100S2.score(X2,y2)
```

[127]:
```
0.995
```

6.2.17 Evaluating SVMC1000S2 on (train set)

[128]:
```
SVMC1000S2.score(X1,y1)
```

[128]:
```
1.0
```

6.2.18 Evaluating SVMC1000S2 on (test set)

[129]:
```
SVMC1000S2.score(X2,y2)
```

[129]:
```
0.995
```

6.2.19 Evaluating SVMC1S4 on (train set)

[130]:
```
SVMC1S4.score(X1,y1)
```

[130]:
```
1.0
```

6.2.20 Evaluating SVMC1S4 on (test set)

[131]:
```
SVMC1S4.score(X2,y2)
```

[131]:
```
0.9925
```

6.2.21 Evaluating SVMC100S4 on (train set)

[132]:
```
SVMC100S4.score(X1,y1)
```

[132]:
```
1.0
```

6.2.22 Evaluating SVMC100S4 on (test set)

[133]:
```
SVMC100S4.score(X2,y2)
```

[133]:
```
0.9925
```

6.2.23 Evaluating SVMC1000S4 on (train set)

[134]:
```
SVMC1000S4.score(X1,y1)
```

[134]:
```
1.0
```

6.2.24 Evaluating SVMC1000S4 on (test set)

[135]:
```
SVMC1000S4.score(X2,y2)
```

[135]:
```
0.9925
```

6.3 c.

6.3.1 Number of support vectors for SVMC1S05

[136]:
```
SVMC1S05.n_support_
```

[136]:
```
array([60, 59], dtype=int32)
```

6.3.2 Number of support vectors for SVMC100S05

[137]:
```
SVMC100S05.n_support_
```

[137]:
```
array([63, 58], dtype=int32)
```

6.3.3 Number of support vectors for SVMC1000S05

[138]:
```
SVMC1000S05.n_support_
```

[138]:
```
array([63, 58], dtype=int32)
```

6.3.4 Number of support vectors for SVMC1S1

[139]:
```
SVMC1S1.n_support_
```

[139]:
```
array([96, 98], dtype=int32)
```

6.3.5 Number of support vectors for SVMC100S1

[140]:
```
SVMC100S1.n_support_
```

[140]:
```
array([ 96, 100], dtype=int32)
```

6.3.6 Number of support vectors for SVMC1000S1

[141]:
```
SVMC1000S1.n_support_
```

[141]:
```
array([ 96, 100], dtype=int32)
```

6.3.7 Number of support vectors for SVMC1S2

[142]:
```
SVMC1S2.n_support_
```

[142]:
```
array([124, 124], dtype=int32)
```

6.3.8 Number of support vectors for SVMC100S2

[143]:
```
SVMC100S2.n_support_
```

[143]:
```
array([124, 125], dtype=int32)
```

6.3.9 Number of support vectors for SVMC1000S2

[144]:
```
SVMC1000S2.n_support_
```

[144]:
```
array([124, 125], dtype=int32)
```

6.3.10 Number of support vectors for SVMC1S4

[145]:
```
SVMC1S4.n_support_
```

[145]:
```
array([152, 156], dtype=int32)
```

6.3.11 Number of support vectors for SVMC100S4

[146]:
```
SVMC100S4.n_support_
```

[146]:
```
array([152, 156], dtype=int32)
```

6.3.12 Number of support vectors for SVMC1000S4

[147]:
```
SVMC1000S4.n_support_
```

[147]:
```
array([152, 156], dtype=int32)
```

It looks like increasing C (with keeping Sigma at a fixed value) increases the support vectors.
Same happens with increasing Sigma (with keeping C at a fixed value). This increase has a higher momentum.

7 Question 4.7

7.1 a.

[0]:
```
X1=X3
y1=y3
X2=X4
y2=y4
```

7.1.1 SVM classifier with C=1, sigma=0.5 on for train

[0]:
```
SVMC1S05=svm.SVC(kernel='rbf',gamma=0.5,C=1,tol=0.001).fit(X1,y1)
```

7.1.2 SVM classifier with C=100, sigma=0.5 on for train

[0]:
```
SVMC100S05=svm.SVC(kernel='rbf',gamma=0.5,C=100,tol=0.001).fit(X1,y1)
```

7.1.3 SVM classifier with C=1000, sigma=0.5 on for train

[0]:
```
SVMC1000S05=svm.SVC(kernel='rbf',gamma=0.5,C=1000,tol=0.001).fit(X1,y1)
```

7.1.4 SVM classifier with C=1, sigma=1 on for train

[0]:
```
SVMC1S1=svm.SVC(kernel='rbf',gamma=1,C=1,tol=0.001).fit(X1,y1)
```

7.1.5 SVM classifier with C=100, sigma=1 on for train

[0]:
```
SVMC100S1=svm.SVC(kernel='rbf',gamma=1,C=100,tol=0.001).fit(X1,y1)
```

7.1.6 SVM classifier with C=1000, sigma=1 on for train

[0]:
```
SVMC1000S1=svm.SVC(kernel='rbf',gamma=1,C=1000,tol=0.001).fit(X1,y1)
```

7.1.7 SVM classifier with C=1, sigma=2 on for train

[0]:
```
SVMC1S2=svm.SVC(kernel='rbf',gamma=2,C=1,tol=0.001).fit(X1,y1)
```

7.1.8 SVM classifier with C=100, sigma=2 on for train

[0]:
```
SVMC100S2=svm.SVC(kernel='rbf',gamma=2,C=100,tol=0.001).fit(X1,y1)
```

7.1.9 SVM classifier with C=1000, sigma=2 on for train

[0]:
```
SVMC1000S2=svm.SVC(kernel='rbf',gamma=2,C=1000,tol=0.001).fit(X1,y1)
```

7.1.10 SVM classifier with C=1, sigma=4 on for train

[0]:
```
SVMC1S4=svm.SVC(kernel='rbf',gamma=4,C=1,tol=0.001).fit(X1,y1)
```

7.1.11 SVM classifier with C=100, sigma=4 on for train

[0]:
```
SVMC100S4=svm.SVC(kernel='rbf',gamma=4,C=100,tol=0.001).fit(X1,y1)
```

7.1.12 SVM classifier with C=1000, sigma=4 on for train

[0]:
```
SVMC1000S4=svm.SVC(kernel='rbf',gamma=4,C=1000,tol=0.001).fit(X1,y1)
```

7.2 b.

7.2.1 Evaluating SVMC1S05 on (train set)

[161]:
```
SVMC1S05.score(X1,y1)
```

[161]:
```
0.99
```

7.2.2 Evaluating SVMC1S05 on (test set)

[162]:
```
SVMC1S05.score(X2,y2)
```

[162]:
```
0.965
```

7.2.3 Evaluating SVMC100S05 on (train set)

[163]:
```
SVMC100S05.score(X1,y1)
```

[163]:
```
1.0
```

7.2.4 Evaluating SVMC100S05 on (test set)

[164]:
```
SVMC100S05.score(X2,y2)
```

[164]:
```
0.965
```

7.2.5 Evaluating SVMC1000S05 on (train set)

[165]:
```
SVMC1000S05.score(X1,y1)
```

[165]:
```
1.0
```

7.2.6 Evaluating SVMC1000S05 on (test set)

[166]:
```
SVMC1000S05.score(X2,y2)
```

[166]:
```
0.965
```

7.2.7 Evaluating SVMC1S1 on (train set)

[167]:
```
SVMC1S1.score(X1,y1)
```

[167]:
```
0.995
```

7.2.8 Evaluating SVMC1S1 on (test set)

[168]:
```
SVMC1S1.score(X2,y2)
```

[168]:
```
0.9675
```

7.2.9 Evaluating SVMC100S1 on (train set)

[169]:
```
SVMC100S1.score(X1,y1)
```

[169]:
```
1.0
```

7.2.10 Evaluating SVMC100S1 on (test set)

[170]:
```
SVMC100S1.score(X2,y2)
```

[170]:
```
0.965
```

7.2.11 Evaluating SVMC1000S1 on (train set)

[171]:
```
SVMC1000S1.score(X1,y1)
```

[171]:
```
1.0
```

7.2.12 Evaluating SVMC1000S1 on (test set)

[172]:
```
SVMC1000S1.score(X2,y2)
```

[172]:
```
0.965
```

7.2.13 Evaluating SVMC1S2 on (train set)

[173]:
```
SVMC1S2.score(X1,y1)
```

[173]:
```
0.9975
```

7.2.14 Evaluating SVMC1S2 on (test set)

[174]:
```
SVMC1S2.score(X2,y2)
```

[174]:
```
0.965
```

7.2.15 Evaluating SVMC100S2 on (train set)

[175]:
```
SVMC100S2.score(X1,y1)
```

[175]:
```
1.0
```

7.2.16 Evaluating SVMC100S2 on (test set)

[176]:
```
SVMC100S2.score(X2,y2)
```

[176]:
```
0.965
```

7.2.17 Evaluating SVMC1000S2 on (train set)

[177]:
```
SVMC1000S2.score(X1,y1)
```

[177]:
```
1.0
```

7.2.18 Evaluating SVMC1000S2 on (test set)

[178]:
```
SVMC1000S2.score(X2,y2)
```

[178]:
```
0.965
```

7.2.19 Evaluating SVMC1S4 on (train set)

[179]:
```
SVMC1S4.score(X1,y1)
```

[179]:
```
1.0
```

7.2.20 Evaluating SVMC1S4 on (test set)

[180]:
```
SVMC1S4.score(X2,y2)
```

[180]:
```
0.9525
```

7.2.21 Evaluating SVMC100S4 on (train set)

[181]:
```
SVMC100S4.score(X1,y1)
```

[181]:
```
1.0
```

7.2.22 Evaluating SVMC100S4 on (test set)

[182]:
```
SVMC100S4.score(X2,y2)
```

[182]:
```
0.9525
```

7.2.23 Evaluating SVMC1000S4 on (train set)

[183]:
```
SVMC1000S4.score(X1,y1)
```

[183]:
```
1.0
```

7.2.24 Evaluating SVMC1000S4 on (test set)

[184]:
```
SVMC1000S4.score(X2,y2)
```

[184]:
```
0.9525
```

7.3 c.

7.3.1 Number of support vectors for SVMC1S05

[185]:
```
SVMC1S05.n_support_
```

[185]:
```
array([ 96, 108], dtype=int32)
```

7.3.2 Number of support vectors for SVMC100S05

[186]:
```
SVMC100S05.n_support_
```

[186]:
```
array([88, 91], dtype=int32)
```

7.3.3 Number of support vectors for SVMC1000S05

[187]:
```
SVMC1000S05.n_support_
```

[187]:
```
array([88, 91], dtype=int32)
```

7.3.4 Number of support vectors for SVMC1S1

[188]:
```
SVMC1S1.n_support_
```

[188]:
```
array([132, 139], dtype=int32)
```

7.3.5 Number of support vectors for SVMC100S1

[189]:
```
SVMC100S1.n_support_
```

[189]:
```
array([128, 132], dtype=int32)
```

7.3.6 Number of support vectors for SVMC1000S1

[190]:
```
SVMC1000S1.n_support_
```

[190]:
```
array([128, 132], dtype=int32)
```

7.3.7 Number of support vectors for SVMC1S2

[191]:
```
SVMC1S2.n_support_
```

[191]:
```
array([161, 161], dtype=int32)
```

7.3.8 Number of support vectors for SVMC100S2

[192]:
```
SVMC100S2.n_support_
```

[192]:
```
array([160, 160], dtype=int32)
```

7.3.9 Number of support vectors for SVMC1000S2

[193]:
```
SVMC1000S2.n_support_
```

[193]:
```
array([160, 160], dtype=int32)
```

7.3.10 Number of support vectors for SVMC1S4

[194]:
```
SVMC1S4.n_support_
```

[194]:
```
array([177, 182], dtype=int32)
```

7.3.11 Number of support vectors for SVMC100S4

[195]:
```
SVMC100S4.n_support_
```

[195]:
```
array([177, 183], dtype=int32)
```

7.3.12 Number of support vectors for SVMC1000S4

[196]:
```
SVMC1000S4.n_support_
```

[196]:
```
array([177, 183], dtype=int32)
```

It looks like increasing C (with keeping Sigma at a fixed value) increases the support vectors.

Same happens with increasing Sigma (with keeping C at a fixed value). This increase has a higher momentum.