2.1 (a)

2.1

1ⓐ if $y \neq sign(f(x))$

then $1(y \neq sign(f(x))) = 1$

$y \times f(x) < 0$

$1 - y f(x) > 1 > 0$

$1 - y f(x) > 1(y \neq sign(f(x)))$

$\therefore 1(y \neq sign\ f(x)) \leq \max\{0, 1 - y f(x)\}$.

if $y = sign(f(x))$

then $1(y \neq sign(f(x))) = 0$.

$y . f(x) \geq 0$.

$1 - y f(x) \leq 1$

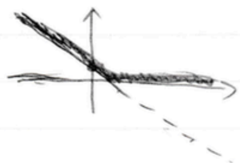
$\therefore \max\{0, 1 - y f(x)\} \in [0, 1]$.

$\therefore 0 \leq [0, 1]$

$\Rightarrow 1(y \neq sign\ f(x)) \leq \max\{0, 1 - y f(x)\}$.

2.1 (b) & (c)

(b)



0 is a constant, $1-m$ is affine $\Rightarrow$ both are convex and $\max\{0, 1-m\}$ is also a convex function

(c) $1-yw^Tx$

w^Tx is R $\quad$ $1-y\,w^Tx$ is an affine function

So $\max\{0, 1-yw^Tx\}$ is also a convex function

2.2 (1)

2.2

1. $h(x, f(x)) = \max_{y \in Y} (h(x, y))$.

and $\max_{y \in Y} (h(x, y)) \geq h(x, y)$

$\therefore h(x, f(x)) \geq h(x, y)$.

2.2 (2)

2 By Q1. $h(x, f(x)) \geq h(x, y)$

$h(x, f(x)) - h(x, y) \geq 0$

$\therefore \Delta(y, f(x)) + h(x, f(x)) - h(x, y) \geq \Delta(y, f(x))$

$\Delta(y, f(x)) + h(x, f(x)) - h(x, y) \leq \underset{\cancel{f \in} F'}{\operatorname{argmax}} [\Delta(y, f(x)) + h(x, f(x)) - h(x, y)]$

$\leq \underset{y' \in Y}{\operatorname{argmax}} [\Delta(y, y') + h(x, y') - h(x, y)]$

2.2 (3)

3. $l(h_w(x_i, y_i)) = \max_{y \in Y} [\Delta(y_i, y) + h(x_i, y) - h(x_i, y_i)]$

$= \max_{y \in Y} [\Delta(y_i, y) + \langle w, \psi(x_i, y)\rangle - \langle w, \varphi(x_i, y_i)\rangle$

Since it is in hilbert space

$\langle w, \varphi(x_i, y)\rangle - \langle w, \varphi(x_i, y_i)\rangle = \langle w, \varphi(x_i, y) - \varphi(x_i, y_i)\rangle$

$\therefore l(h_w, (x_i, y_i)) = \max_{y \in Y} [\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i)\rangle]$

2.2 (4)

4. ⓐ $\underbrace{\Delta(y_i, y)}_{\text{constant Scalar}} + \langle w, \underbrace{\varphi(x_i, y) - \varphi(x_i, y_i)}_{\text{constant vector}} \rangle$.

$\therefore$ if $a = \Delta(y_i, y)$. $b = \varphi(x_i, y) - \varphi(x_i, y_i) \rangle$.
it can be expressed as $wb + a$ $\therefore$ it is an affine function

ⓑ for any $y \in Y$, $\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i) \rangle$ is affine fuction, which is convex

$\therefore \max_{y \in Y} [\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i) \rangle]$ is convex.

2.2 (5)

3. $l(h_w, (x_i, y_i)) \geq \Delta(y, f(x))$
and $l(h_w, (x_i, y_i))$ is convex
$\therefore l(h_w, (x_i, y_i))$ is the convex surrogate for $\Delta(y, f(x))$

3.1 (a) & (b) & (c)

3.1

(a) proven by Q2.4 (b). $\max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i) \rangle]$ is convex.

$\therefore$ nonnegative combination of convex functions is convex as well

$\therefore \frac{1}{n}\sum_{i=1}^{n} \max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i) \rangle]$ is convex

(b) l_2 norm is convex since every norm is always a confex fuction

(c) sum of 2 convex fuctions is still convex

$\therefore J(w) = \underbrace{\lambda \|w\|^2}_{\text{convex}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} \max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i) \rangle]}_{\text{convex}}$

3.2

3.2 $J(w) = \lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max[\Delta(y_i, y) + \langle w, \varphi(x_i, y) - \varphi(x_i, y_i)\rangle]$

$= \lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} [\Delta(y_i, \hat{y}) + \langle w, \varphi(x_i, \hat{y}) - \varphi(x_i, y_i)\rangle]$

$\partial J(w) = 2\lambda w + \partial\left(\frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i, \hat{y})]\right) + \partial\left(\frac{1}{n}\sum_{i=1}^{n} \langle w, \varphi(x_i, \hat{y}) - \varphi(x_i, y_i)\rangle\right)$

$= 2\lambda w + 0 + \frac{1}{n}\sum_{i=1}^{n}(\varphi(x_i, \hat{y}) - \varphi(x_i, y_i))$

$= 2\lambda w + \frac{1}{n}\sum_{i=1}^{n}(\varphi(x_i, \hat{y}) - \varphi(x_i, y_i))$

3.3 & 3.4

3.3: $2\cdot\lambda W + (\varphi(x_i, \hat{y}) - \varphi(x_i, y_i))$

3.4. $2\lambda W + \frac{1}{m} \sum_{i=I}^{I+m-1} (\varphi(x_i, \hat{y}) - \varphi(x_i, y_i))$.

4.1 & 4.2

4.1 $l(h,(x_i,y_i)) = \max_{y \in \mathcal{Y}} [\Delta(y_i, y) - (h(x_i, y_i) - h(x_i, y))]$.

and $m_{i,y}(h) = h(x_i, y_i) - h(x_i, y)$.

$\therefore l(h,(x_i,y_i)) = \max_{y \in \mathcal{Y}} [\Delta(y_i, y) - m_{i,y}(h)]$.

4.2 By Q2 $\therefore$ $\Delta(y_i, y) - (h(x_i,y_i) - h(x_i, y)) \geq \Delta(y_i, y) \geq 0$.

$\therefore \Delta(y_i, y) - m_{i,y}(h) \geq 0$.

$\therefore [\Delta(y_i,y) - m_{i,y}(h)]_+ = \Delta(y_i, y) - m_{i,y}(h)$.

4.3 – Blank

5

5. $l(h,(x,y)) = \max_{y' \in y} [\Delta(y,y') + h(x,y') - h(x,y)]$

if $y = y'$ $\quad \Downarrow \quad \Delta(y,y) + \cancel{h(x,y)} - \cancel{h(x,y)}$

since $\Delta(y,y') = 1(y \neq y')$.

$\therefore = 0$.

if $y \neq y'$. $\quad [1 + h(x,y') - h(x,y)]$

if $y=1$, $y'=-1$. $\Rightarrow [1 + -\frac{g(x)}{2} - \frac{g(x)}{2}]$

$= 1 - g(x) = 1 - y\,g(x)$.

if $y=-1$ $y'=1$ $\Rightarrow [1 + \frac{g(x)}{2} + \frac{g(x)}{2}]$.

$= 1 + g(x) = 1 - y\,g(x)$.

$\therefore$ if $y \neq y'$. $[\Delta(y,y') + h(x,y') - h(x,y)] = 1 - y(gx)$.

$\therefore$ this can be reduced to $\max\{0, 1 - y\,g(x)\}$.

In [1]:

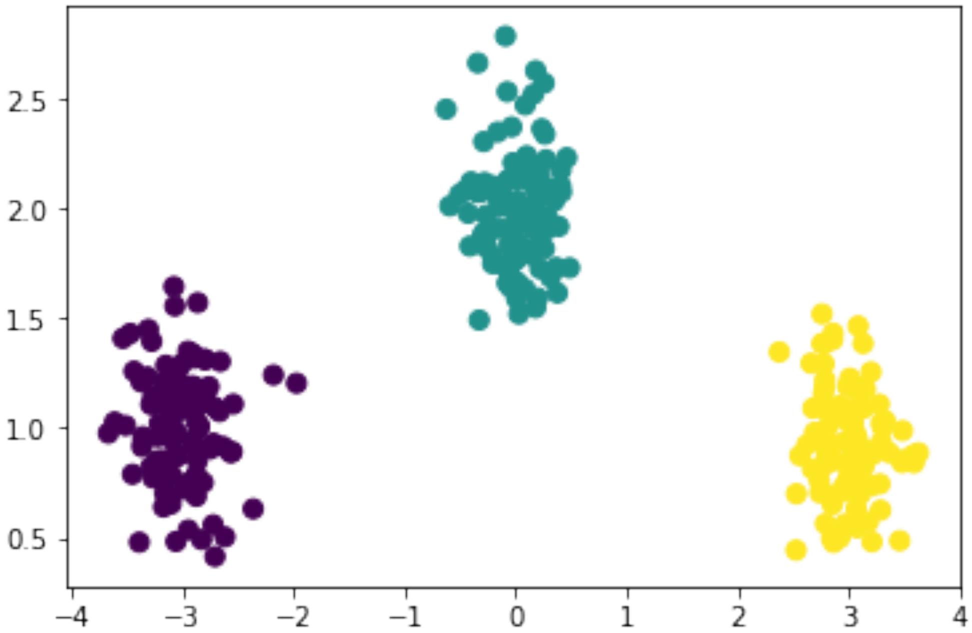
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
import copy
%matplotlib inline
import random
```

In [2]:

```
# Create the  training data
np.random.seed(2)
X, y = make_blobs(n_samples=300,cluster_std=.25, centers=np.array([(-3,1),(0,2),
(3,1)]))
plt.scatter(X[:, 0], X[:, 1], c=y, s=50)
```

Out[2]:

```
<matplotlib.collections.PathCollection at 0x114d99780>
```



In [3]:

```
from sklearn.base import BaseEstimator, ClassifierMixin, clone

class OneVsAllClassifier(BaseEstimator, ClassifierMixin):
    """
    One-vs-all classifier
    We assume that the classes will be the integers 0,..,(n_classes-1).
    We assume that the estimator provided to the class, after fitting, has a "de
cision_function" that
    returns the score for the positive class.
    """
    def __init__(self, estimator, n_classes):
        """
        Constructed with the number of classes and an estimator (e.g. an
        SVM estimator from sklearn)
```

```
SVM estimator from sklearn)
        @param estimator : binary base classifier used
        @param n_classes : number of classes
        """
        self.n_classes = n_classes
        self.estimators = [clone(estimator) for _ in range(n_classes)]
        self.fitted = False

    def fit(self, X, y=None):
        """
        This should fit one classifier for each class.
        self.estimators[i] should be fit on class i vs rest
        @param X: array-like, shape = [n_samples,n_features], input data
        @param y: array-like, shape = [n_samples,] class labels
        @return returns self
        """
        #Your code goes here
        self.X = X
        self.y = y
        self.ylablelist = np.unique(self.y)
        for i in range(self.n_classes):
            self.ylable = self.ylablelist[i] # the class working on
            self.y_inuse = copy.deepcopy(self.y) # make a deepcopy of the orgina
l y
            self.y_inuse[self.y_inuse!=self.ylable] = -1  # change the rest to l
able -1
            self.estimators[i].fit(self.X, self.y_inuse) # fit the one class vs
the rest


        self.fitted = True
        return self

    def decision_function(self, X):
        """
        Returns the score of each input for each class. Assumes
        that the given estimator also implements the decision_function method (w
hich sklearn SVMs do),
        and that fit has been called.
        @param X : array-like, shape = [n_samples, n_features] input data
        @return array-like, shape = [n_samples, n_classes]
        """
        if not self.fitted:
            raise RuntimeError("You must train classifer before predicting data.
")

        if not hasattr(self.estimators[0], "decision_function"):
            raise AttributeError(
                "Base estimator doesn't have a decision_function attribute.")

        #Replace the following return statement with your code
        self.X = X
        self.result = np.zeros((X.shape[0],self.n_classes))
```

```
        for i in range(self.n_classes):

            self.result[:,i] = self.estimators[i].decision_function(self.X)


        return self.result

    def predict(self, X):
        """
        Predict the class with the highest score.
        @param X: array-like, shape = [n_samples,n_features] input data
        @returns array-like, shape = [n_samples,] the predicted classes for each
input
        """
        #Replace the following return statement with your code

        self.X = X
        self.predict_class =  np.zeros(X.shape[0])
        self.decision = self.decision_function(self.X)

        for i in range(self.decision.shape[0]):
            self.predict_class[i] = str(self.ylablelist[np.argmax(self.decision[
i])])



        return self.predict_class

    def norm_predict(self, X):
        """
        Predict the class with the highest score.
        @param X: array-like, shape = [n_samples,n_features] input data
        @returns array-like, shape = [n_samples,] the predicted classes for each
input
        """
        #Replace the following return statement with your code

        self.X = X
        self.predict_class_n =  np.zeros(X.shape[0])

        self.decision = self.decision_function(self.X)

        self.normlized_decision = np.zeros((X.shape[0],self.n_classes))

        for i in range(self.decision.shape[1]):

            self.normlized_decision[:,i] = (self.decision[:,i] / np.linalg.norm(
self.decision[:,i] ))

        for j in range(self.normlized_decision.shape[0]):
            self.predict_class_n[j] = str(self.ylablelist[np.argmax(self.normliz
ed_decision[j])])
```

```
        return self.predict_class_n
```

In [4]:

```
#Here we test the OneVsAllClassifier
from sklearn import svm
svm_estimator = svm.LinearSVC(loss='hinge', fit_intercept=False, C=200)
clf_onevsall = OneVsAllClassifier(svm_estimator, n_classes=3)
clf_onevsall.fit(X,y)

for i in range(3) :
    print("Coeffs %d"%i)
    print(clf_onevsall.estimators[i].coef_) #Will fail if you haven't implemente
d fit yet
```

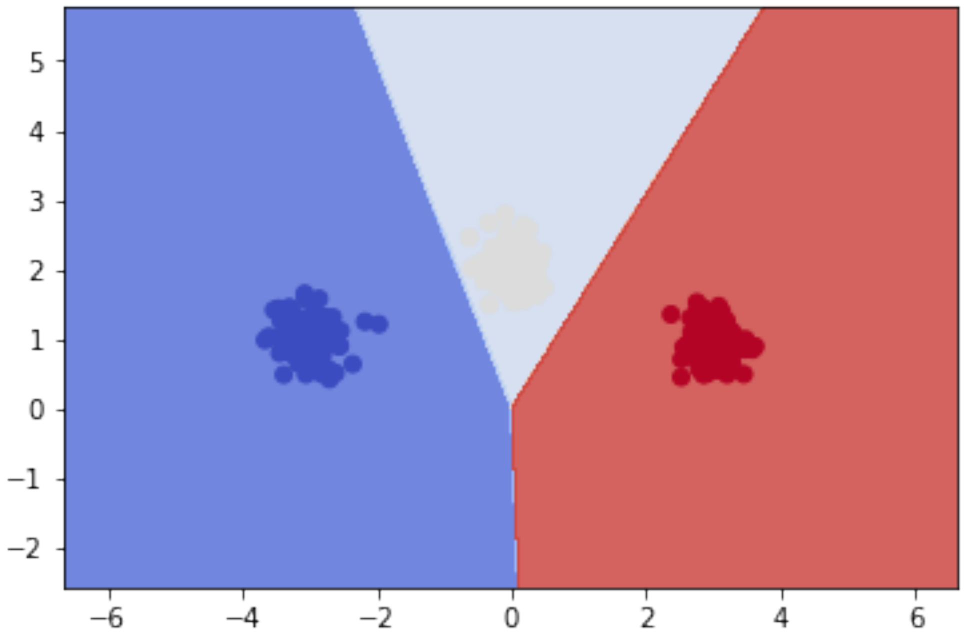
```
Coeffs 0
[[-1.05852747 -0.90296521]]
Coeffs 1
[[ 0.22117096 -0.38900908]]
Coeffs 2
[[ 0.89162796 -0.82467394]]
```

In [5]:

```
# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = min(X[:,0])-3,max(X[:,0])+3
y_min, y_max = min(X[:,1])-3,max(X[:,1])+3
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
mesh_input = np.c_[xx.ravel(), yy.ravel()]

Z = clf_onevsall.predict(mesh_input)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)


from sklearn import metrics
metrics.confusion_matrix(y, clf_onevsall.predict(X))
```

Out[5]:

```
array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```

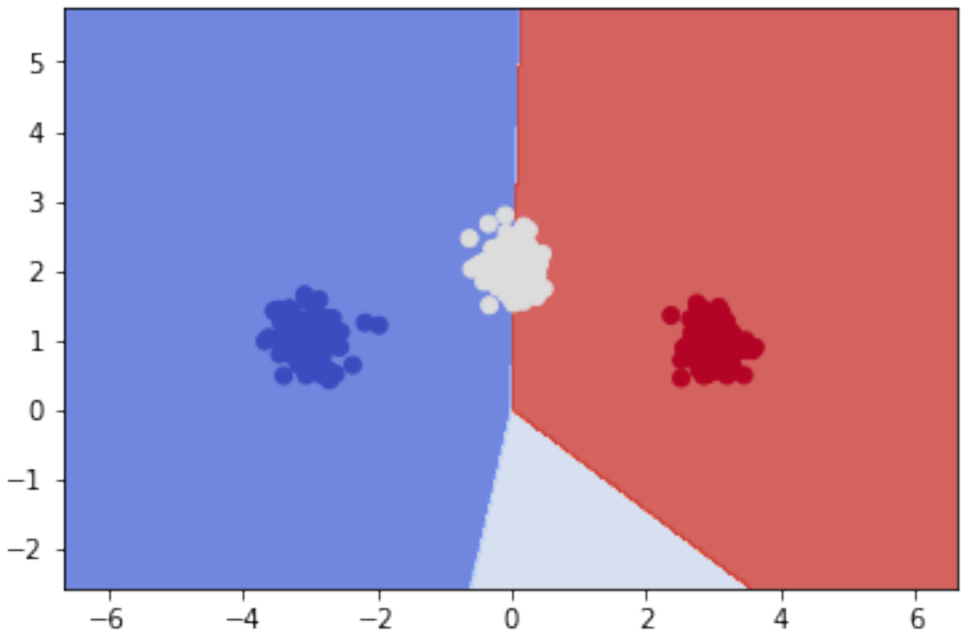


In [6]:

```
# 6.1.2 (Normalized)
# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = min(X[:,0])-3,max(X[:,0])+3
y_min, y_max = min(X[:,1])-3,max(X[:,1])+3
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
mesh_input = np.c_[xx.ravel(), yy.ravel()]

Z = clf_onevsall.norm_predict(mesh_input)

Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)


from sklearn import metrics
metrics.confusion_matrix(y, clf_onevsall.predict(X))
```

Out[6]:

```
array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



Multiclass SVM

In [7]:

```
def zeroOne(y,a) :
    '''
    Computes the zero-one loss.
    @param y: output class
    @param a: predicted class
    @return 1 if different, 0 if same
    '''
    return int(y != a)

def featureMap(X,y,num_classes) :
    '''
    Computes the class-sensitive features.
    @param X: array-like, shape = [n_samples,n_inFeatures] or [n_inFeatures,], i
nput features for input data
    @param y: a target class (in range 0,..,num_classes-1)
    @return array-like, shape = [n_samples,n_outFeatures], the class sensitive f
eatures for class y
    '''
    #The following line handles X being a 1d-array or a 2d-array
    num_samples, num_inFeatures = (1,X.shape[0]) if len(X.shape) == 1 else (X.sh
ape[0],X.shape[1])

    #your code goes here, and replaces following return




    if num_samples >1:
        outFeatures = np.zeros((num_samples,num_inFeatures * num_classes))
        for i in range(num_samples):


            # find out the index in the unique ylablelist to know where to put i
n the feature map
            for j in range (num_inFeatures):
                outFeatures[i,j+y[i]*num_inFeatures] = X[i][j]
    else:
        outFeatures = np.zeros(num_inFeatures * num_classes)
        for j in range(num_inFeatures):
            outFeatures[y*num_inFeatures+j] = X[j]

    return outFeatures
```

In [8]:

```
def sgd(X, y, num_outFeatures, subgd, eta = 0.1, T = 10000):
    '''
    Runs subgradient descent, and outputs resulting parameter vector.
    @param X: array-like, shape = [n_samples,n_features], input training data
    @param y: array-like, shape = [n_samples,], class labels
    @param num_outFeatures: number of class-sensitive features
    @param subgd: function taking x,y and giving subgradient of objective
    @param eta: learning rate for SGD
    @param T: maximum number of iterations
    @return: vector of weights
    '''
    num_samples = X.shape[0]

    t = 0

    w = np.zeros(num_outFeatures)

    index_list = list(range(num_samples))

    while t < T:
        random.shuffle(index_list)
        for i in index_list:
            w = w - subgd(X[i],y[i],w) * eta
            t += 1

    return w
```

In [9]:

```
class MulticlassSVM(BaseEstimator, ClassifierMixin):
    '''
    Implements a Multiclass SVM estimator.
    '''
    def __init__(self, num_outFeatures, lam=1.0, num_classes=3, Delta=zeroOne, P
si=featureMap):
        '''
        Creates a MulticlassSVM estimator.
        @param num_outFeatures: number of class-sensitive features produced by P
si
        @param lam: l2 regularization parameter
        @param num_classes: number of classes (assumed numbered 0,..,num_classes
-1)
        @param Delta: class-sensitive loss function taking two arguments (i.e.,
target margin)
        @param Psi: class-sensitive feature map taking two arguments
        '''
        self.num_outFeatures = num_outFeatures
        self.lam = lam
        self.num_classes = num_classes
        self.Delta = Delta
        self.Psi = lambda X,y : Psi(X,y,num_classes)
        self.fitted = False
```

```
        self.fitted = False

    def subgradient(self,x,y,w):
        '''
        Computes the subgradient at a given data point x,y
        @param x: sample input
        @param y: sample class
        @param w: parameter vector
        @return returns subgradient vector at given x,y,w
        '''
        #Your code goes here and replaces the following return statement
        yy = np.zeros(self.num_classes)
        index = np.where(X==x)[0][0]
        for i in range(self.num_classes):
            yy[i] = self.Delta(i,y) + np.dot(w,(self.Psi(X[index],i)-self.Psi(X[
index],y)))

        subgrad = 2 * self.lam*w + (self.Psi(X[index],np.argmax(yy))-self.Psi(X[
index],y))

        return subgrad

    def fit(self,X,y,eta=0.1,T=10000):
        '''
        Fits multiclass SVM
        @param X: array-like, shape = [num_samples,num_inFeatures], input data
        @param y: array-like, shape = [num_samples,], input classes
        @param eta: learning rate for SGD
        @param T: maximum number of iterations
        @return returns self
        '''
        self.coef_ = sgd(X,y,self.num_outFeatures,self.subgradient,eta,T)
        self.fitted = True
        return self

    def decision_function(self, X):
        '''
        Returns the score on each input for each class. Assumes
        that fit has been called.
        @param X : array-like, shape = [n_samples, n_inFeatures]
        @return array-like, shape = [n_samples, n_classes] giving scores for eac
h sample,class pairing
        '''
        if not self.fitted:
            raise RuntimeError("You must train classifer before predicting data.
")

        #Your code goes here and replaces following return statement


        result = np.zeros((X.shape[0],self.num_classes))


        for i in range(self.num_classes):
```

```
            y_inuse = np.array([i]*(X.shape[0]))

            x_inuse = self.Psi(X,y_inuse)

            result[:,i] = np.dot(x_inuse,self.coef_)


        return result


    def predict(self, X):
        '''
        Predict the class with the highest score.
        @param X: array-like, shape = [n_samples, n_inFeatures], input data to p
redict
        @return array-like, shape = [n_samples,], class labels predicted for eac
h data point
        '''




        predict_class = np.zeros(X.shape[0])
        decision = self.decision_function(X)

        for i in range(X.shape[0]):
            predict_class[i] = np.argmax(decision[i])




        return predict_class
```

In [12]:

```
#the following code tests the MulticlassSVM and sgd
#will fail if MulticlassSVM is not implemented yet
est = MulticlassSVM(6,lam=0.1)
est.fit(X,y)
print("w:")
print(est.coef_)
```

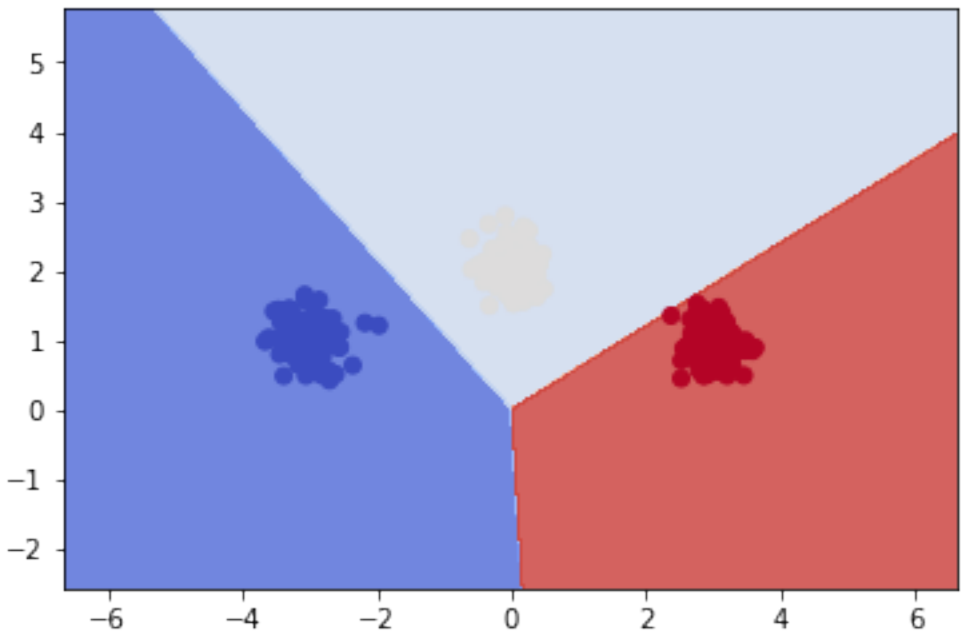
```
w:
[-0.82479706 -0.33408951  0.16611738  0.57775621  0.65867968 -0.2436
667 ]
```

In [13]:

```
Z = est.predict(mesh_input)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)


from sklearn import metrics
metrics.confusion_matrix(y, est.predict(X))
```

Out[13]:

```
array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



In [ ]:

```

```

Optional Question