Project to implement mCLESS algorithm and experiment on various datasets

Creating the mcless module

Information matrix

Create a module named Infomatrix which contains 2 files:

1. __init__.py
2. Information_matrix.py

In Information_matrix we will write the code to augument the data matrix by columns of 1s in the left side of the matrix

Source Matrix

Create a module named srcmatrix which contains 2 files:

1. __init__.py
2. Source_matrix.py

In the Source_matrix we will write the code to get the source matrix from the labels of the data.

mcless

Create a module named mcless which contains 3 files:

1. __init__.py
2. Calculate_W.py
3. mCLESS.py

Calculate_W.py

The least squares method is formulated as

$$\widehat{W} = \underset{W}{\operatorname{argmin}} ||AW - B||$$

I wrote 2 functions in Calculate_W.py

1. calculate_W_by_svd()

Using Singular Value Decomposition, the least sqaures solution is

$$x = \sum_{i=1}^{r} \frac{u_i b}{\sigma_i} v_i$$

In Calculate_w.py , we calculte the weight matrix using SVD for least squares method which is formulated as

We can decompose $A = U\Sigma V^T$. So, we get W using the above method as,

$$W = \sum_{i=1}^{r} \frac{u_i b}{\sigma_i} v_i$$

2. calculate_W_by_normal()

This function calculates W by method of normal equations

$$(A^T A)\widehat{W} = A^T B$$

So

$$\widehat{W} = (A^T A)^{-1} A^T B$$

Since the solutions by both methods were the same for all the datasets in this experiments, only calculate_W_by_normal() is used for all experiments.

mCLESS.py

This file contains the fit(), predict(), and score() methods for the mCLESS class. We give our train set to fit() method to calculte the $\widehat{W}$. Then the classes are predicted using the predict() method for the test set. score()

Following Documents are the attached codes for the above mentioned files :

In [ ]:

```python
import numpy as np

def Information_matrix(X):
    N = len(X)
    A=np.column_stack((np.ones([N,]),X))
    return A
```

```python
import numpy as np

def Source_matrix(y):
    N = len(y)
    allclasses = np.unique(y)
    B = np.zeros((N,len(allclasses)))
    for i in range(N):
        class_value = int(y[i])
        B[i][class_value] = 1
    return B
```

```python
import numpy as np

def calculate_W_by_svd(A,B):
    # Perform SVD of A
    u,s,vh = np.linalg.svd(A)

    # Find rank of A
    K = max(A.shape[0], A.shape[1])
    r = 0;
    while( r < A.shape[1] and abs(s[r]) >= abs(K*1e-6*s[0]) ):
        r = r+1;

    # Find least square solution
    v = vh.transpose()
    W = np.zeros((A.shape[1], B.shape[1]))
    for i in range(r):
        tmp = ((u[:,i].transpose() @ B)/s[i])
        for j in range(B.shape[1]):
            W[:,j] += tmp[j] * v[:,i]

    return W

def calculate_W_by_normal(A,B):
    W = np.linalg.inv(A.transpose() @ A) @ A.transpose() @ B
    return W
```

```python
import numpy as np
import mcless

def predict_y(X,W):
    A = Information_matrix(X)
    B_pred = A @ W
    print(B_pred.shape)
    print(B_pred[0])
    N = len(B_pred)
    y_pred = np.zeros(N)
    for i in range(N):
        c = np.argmax(B_pred[i])
        y_pred[i] = c
    return y_pred
```

```python
import numpy as np
import srcmatrix as src
import infomatrix as info

#from .Calculate_W import calculate_W_by_svd
from .Calculate_W import *

class mCLESS:
    def __init__(self):
        self.W = np.empty((1,1), dtype='double')

    def fit(self, X, y):
        A = info.Information_matrix(X)
        B = src.Source_matrix(y)
        #self.W = calculate_W_by_svd(A,B)
        self.W = calculate_W_by_normal(A,B)
        return self

    def predict(self,X):
        A = info.Information_matrix(X)
        B_pred = A @ self.W
        N = len(B_pred)
        y_pred = np.zeros(N)
        for i in range(N):
            c = np.argmax(B_pred[i])
            y_pred[i] = c
        return y_pred

    def score(self, X, y):
        y_pred = self.predict(X)
        confusion_matrix = np.zeros((len(np.unique(y)),len(np.unique(y))))
        for i in zip(y,y_pred):
            confusion_matrix[int(i[0]),int(i[1])]+=1
        accuracy = np.trace(confusion_matrix)/np.sum(confusion_matrix)
        return accuracy

```

Comparison of mCLESS with different classifiers

In [102]:
```
# Import the required libraries

from mcless import *
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import time
```

In [103]:
```
#Import the datasets

data1 = np.loadtxt('data/synthetic1.data', delimiter=',')
X1= data1[:,0:2]
y1=data1[:,2]

data2= np.loadtxt('data/synthetic2.data', delimiter=',')
X2= data2[:,0:2]
y2=data2[:,2]

data3 = load_iris()
X3 = data3.data
y3 = data3.target

data4 = load_wine()
X4 = data4.data
y4 = data4.target

data_X = [X1,X2,X3,X4]
data_y = [y1,y2,y3,y4]

data_names = ["Synthetic Data 1","Synthetic Data 2","Iris Data","Wine Data"]
```

In [104]:
```
# Create a list of classifiers

classifiers = [mCLESS(),
               LogisticRegression(max_iter = 1000),
               KNeighborsClassifier(5),
               SVC(kernel="rbf",gamma=2, C=1),
               RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1)]

names = [
        "mCLESS",
        "Logistic Regr",
        "KNeighbors-5 ",
        "RBF SVM ",
        "Random Forest"]
```

In [105]:
```
# Define functions to scale the data column-wisely

def scale_factor(X):
    X = X.transpose()
    scale = np.zeros((len(X)))
    for i in range(len(X)):
        row = X[i]
        scale[i] = np.amax(np.abs(row))
    return scale

def scale_data(X,scale):
    X= X.transpose()
    scaled_X = np.zeros((X.shape))

    for i in range(len(X)):
        row = X[i]
        scaled_X[i] = X[i]/scale[i]

    return scaled_X.transpose()
```

In [106]:
```
# Define a function to run experiments

def run_experiments(data_X,data_y,data_names,classifiers):
    rtrain = 0.7e0
    run = 100
    rtest = 1-rtrain


    for X,y,dataname in zip(data_X,data_y,data_names):
        acc_max = 0
        for name, clf in zip(names, classifiers):
            Acc = np.zeros([run,1])
            mclessAcc = 0.0
            btime = time.time()

            for it in range(run):
                Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=rtest,
                                                                random_state=it,
                                                                stratify = y)
                scale = scale_factor(Xtrain)
                Xtrain_scaled = scale_data(Xtrain,scale)
                Xtest_scaled = scale_data(Xtest,scale)

                clf.fit(Xtrain_scaled, ytrain);
                Acc[it] = clf.score(Xtest_scaled, ytest)

                if(name=='mCLESS' and dataname == "Synthetic Data 1"):
                    if(mclessAcc < Acc[it]):
                        mclessAcc = Acc[it]
                        W1 = clf.W

            etime = time.time()-btime
            accmean = np.mean(Acc)*100
            print('%s: %s: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'%
                  (dataname,name,accmean,np.std(Acc)*100,etime/run))

            if accmean>acc_max:
                acc_max= accmean; algname = name

        print('Classifiers max: %s= %.2f\n' %(algname,acc_max))
```

In [107]:

```
# Define function to plot the decision boundries
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    Wtmp = classifier.W
    # plot class examples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],y=X[y == cl, 1],alpha=0.8,
                    c=colors[idx],marker=markers[idx],label=cl,edgecolor='black')
        p = np.where(y1 == i)
        X_value = X[p[0],:]

        px = np.array([-10.0, 15.0])
        py = (0.0-Wtmp[0,idx]-Wtmp[1,idx]*px)/Wtmp[2,idx]
        py1 = (1.0-Wtmp[0,idx]-Wtmp[1,idx]*px)/Wtmp[2,idx]
        plt.plot(px,py,linestyle='dashed',color=COLOR[idx],label="L%d(X1,x2)=0"%idx)
        plt.plot(px,py1,color=COLOR[idx],label="L%d(X1,X2)=1"%idx)
```

In [108]:
```
# Run the experiments for the above data to compare with different classifiers

run_experiments(data_X,data_y,data_names,classifiers)
```

```
Synthetic Data 1: mCLESS: Acc.(mean,std) = (96.89,1.66)%; E-time= 0.00312
Synthetic Data 1: Logistic Regr: Acc.(mean,std) = (96.99,1.71)%; E-time= 0.01216
Synthetic Data 1: KNeighbors-5 : Acc.(mean,std) = (95.81,1.96)%; E-time= 0.00922
Synthetic Data 1: RBF SVM : Acc.(mean,std) = (96.64,1.69)%; E-time= 0.00623
Synthetic Data 1: Random Forest: Acc.(mean,std) = (95.63,2.12)%; E-time= 0.13492
Classifiers max: Logistic Regr= 96.99

Synthetic Data 2: mCLESS: Acc.(mean,std) = (71.04,2.51)%; E-time= 0.00220
Synthetic Data 2: Logistic Regr: Acc.(mean,std) = (94.12,2.07)%; E-time= 0.00990
Synthetic Data 2: KNeighbors-5 : Acc.(mean,std) = (93.96,2.24)%; E-time= 0.00770
Synthetic Data 2: RBF SVM : Acc.(mean,std) = (94.64,2.18)%; E-time= 0.00723
Synthetic Data 2: Random Forest: Acc.(mean,std) = (95.11,2.00)%; E-time= 0.13372
Classifiers max: Random Forest= 95.11

Iris Data: mCLESS: Acc.(mean,std) = (82.96,4.59)%; E-time= 0.00178
Iris Data: Logistic Regr: Acc.(mean,std) = (93.48,3.01)%; E-time= 0.01266
Iris Data: KNeighbors-5 : Acc.(mean,std) = (95.76,2.31)%; E-time= 0.00566
Iris Data: RBF SVM : Acc.(mean,std) = (95.87,2.37)%; E-time= 0.00424
Iris Data: Random Forest: Acc.(mean,std) = (94.76,2.77)%; E-time= 0.12380
Classifiers max: RBF SVM = 95.87

Wine Data: mCLESS: Acc.(mean,std) = (98.61,1.32)%; E-time= 0.00190
Wine Data: Logistic Regr: Acc.(mean,std) = (95.98,2.39)%; E-time= 0.02090
Wine Data: KNeighbors-5 : Acc.(mean,std) = (95.07,2.50)%; E-time= 0.00662
Wine Data: RBF SVM : Acc.(mean,std) = (98.37,1.70)%; E-time= 0.00535
Wine Data: Random Forest: Acc.(mean,std) = (98.35,1.79)%; E-time= 0.12681
Classifiers max: mCLESS= 98.61
```

From the above experiments we can see that-

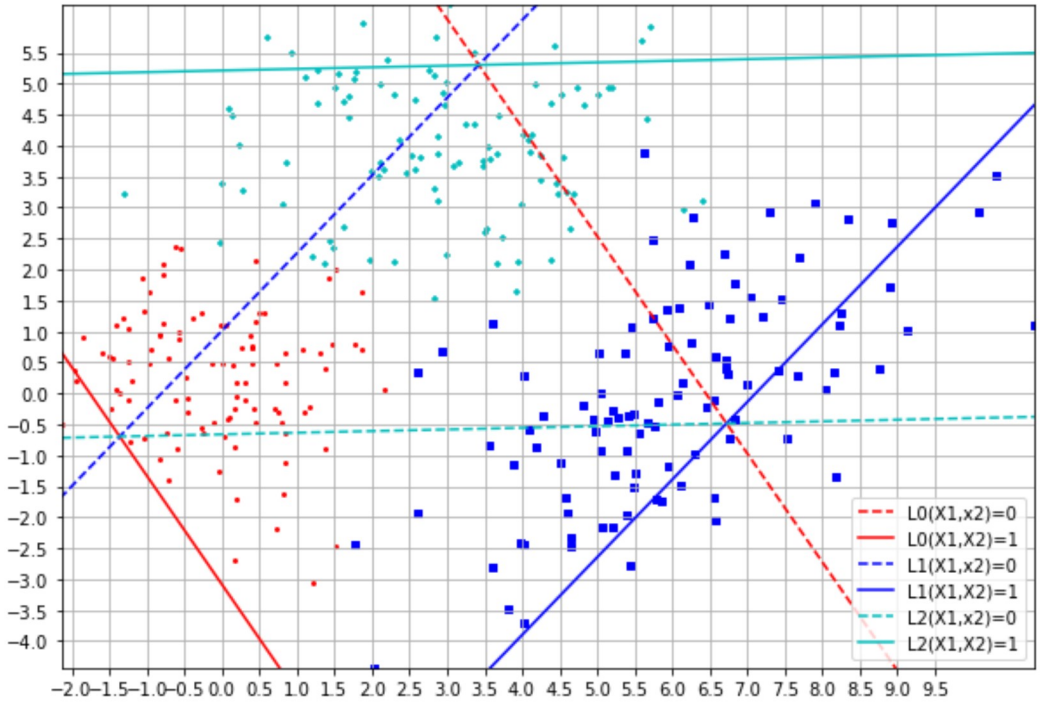
For Synthetic data1 , mCLESS gives accuracy very close to the best accuracy among the other classifiers.

For Synthetic data2 and Iris data , the accuracy of mCLESS is not good.

For Wine data, mCLESS gives the best accuracy among the classifiers.

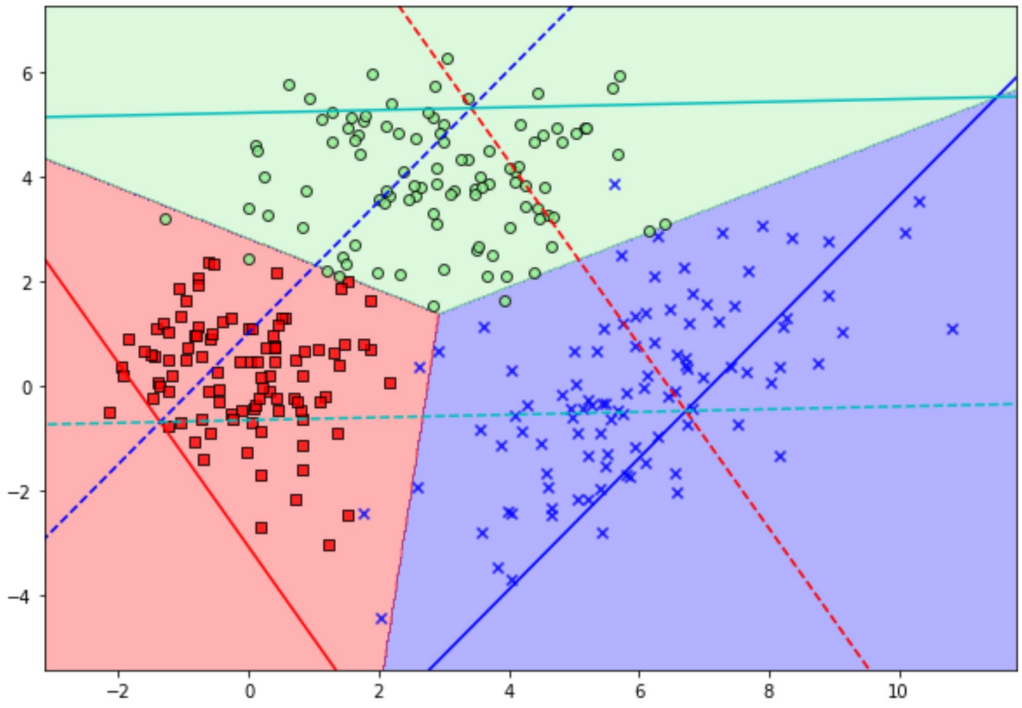
Lets plot the lines represented by weight vectors of mCLESS for Synthetic data 1

In [109]:
```
plt.rcParams["figure.figsize"] = (10,7)
COLOR = ['r','b','c']
MARKER = ['.','s','+','*']
# Get W by running the whole dataset on classifier
mclassifier = mCLESS()
mclassifier.fit(X1, y1)
Wtmp = mclassifier.W

plt.figure(figsize=(10,7))
# Limits of the plot
x1max = np.max(X1[:,0])
x1min = np.min(X1[:,0])
x2max = np.max(X1[:,1])
x2min = np.min(X1[:,1])
plt.xlim(x1min, x1max)
plt.ylim(x2min, x2max)

for i in [0,1,2]:
    p = np.where(y1 == i)
    X_value = X1[p[0],:]
    plt.scatter(X_value[:,0],X_value[:,1],s=15,color=COLOR[i],marker=MARKER[i])

    px = np.array([-10.0, 15.0])
    py = (0.0-Wtmp[0,i]-Wtmp[1,i]*px)/Wtmp[2,i]
    py1 = (1.0-Wtmp[0,i]-Wtmp[1,i]*px)/Wtmp[2,i]
    plt.plot(px,py,linestyle='dashed',color=COLOR[i],label="L%d(X1,x2)=0"%i)
    plt.plot(px,py1,color=COLOR[i],label="L%d(X1,X2)=1"%i)
plt.legend()
plt.grid()
plt.xticks(np.arange(-2, 10, step=0.5))
plt.yticks(np.arange(-4, 6, step=0.5))

plt.show()
```



In [110]:
```
plt.figure(figsize=(10,7))
plot_decision_regions(X1,y1,mclassifier)
```

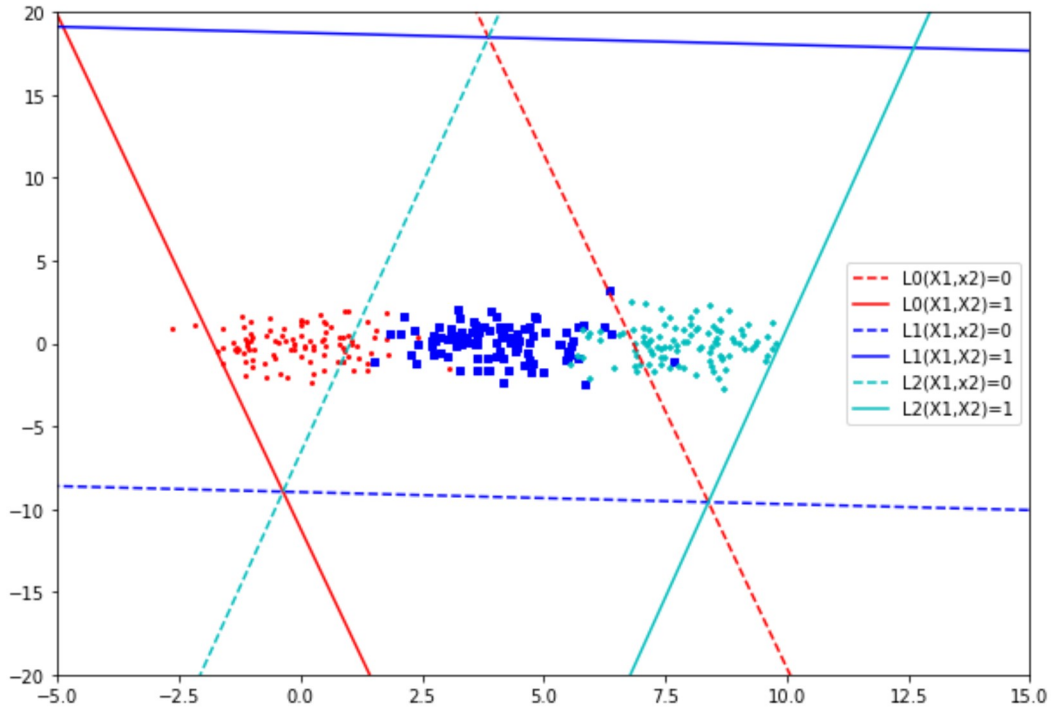
```
C:\Users\anupz\AppData\Local\Temp/ipykernel_1744/1492262336.py:21: UserWarning: You passed
a edgecolor/edgecolors ('black') for an unfilled marker ('x').  Matplotlib is ignoring the
edgecolor in favor of the facecolor.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],y=X[y == cl, 1],alpha=0.8,
```



From the above plot, we can visualize the lines represented by the weight matrix. It shows that for a class j, $L_j(x_1, x_2) = 0$ line passes through the clusters of the samples belonging to the classes other than j, but the line $L_j(x_1, x_2) = 1$ is parallel to the line $L_j(x_1, x_2) = 0$ and passes through the cluster of the samples of class j. In other words, for a sample (x_1, x_2), value of $L_j(x_1, x_2)$ is the signed distance of the sample from $L_j = 0$ line (positive value if the point is on the same side as the class cluster, neagative value otherwise). The classification criterion of $\text{argmax}_j L_j(x_1, x_2)$ for the test sample is trying to identify the class for which the signed distance is maximum.

Lets plot the lines represented by weight vectors of mCLESS for Synthetic data 2

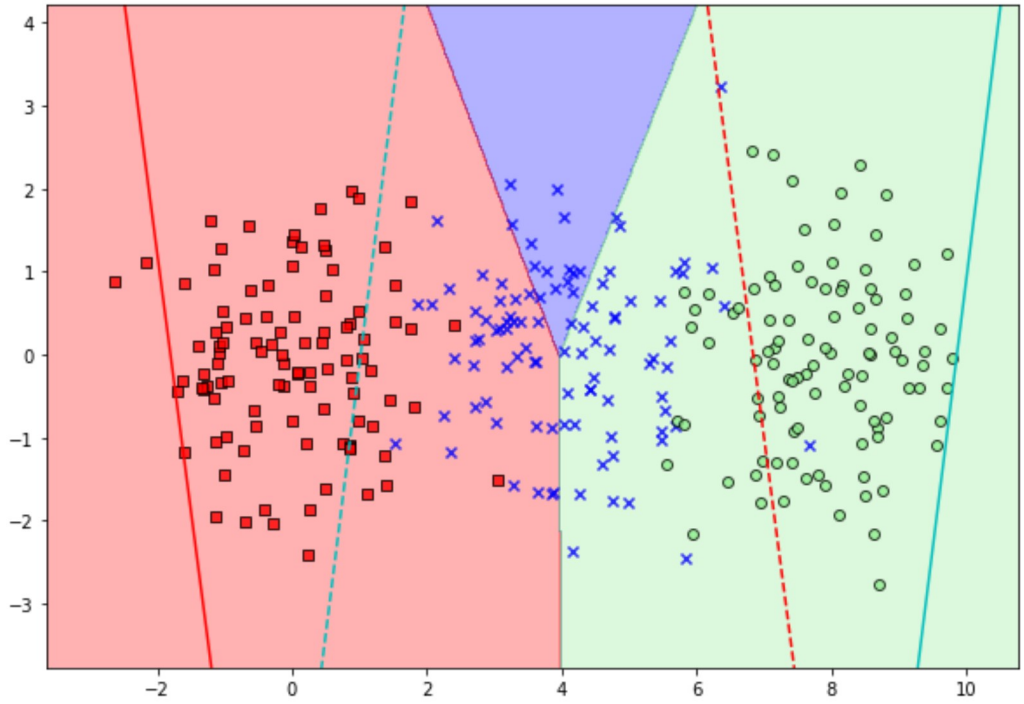
In [294]:
```
mclassifier = mCLESS()
mclassifier2.fit(X2, y2)
Wtmp2 = mclassifier2.W

plt.figure(figsize=(10,7))
# Limits of the plot
x1max = np.max(X2[:,0])
x1min = np.min(X2[:,0])
x2max = np.max(X2[:,1])
x2min = np.min(X2[:,1])
#plt.xlim(x1min, x1max)
#plt.ylim(x2min, x2max)
plt.xlim(-5,15)
plt.ylim(-20,20)

for i in [0,1,2]:
    p = np.where(y1 == i)
    X_value = X2[p[0],:]
    plt.scatter(X_value[:,0],X_value[:,1],s=15,
                color=COLOR[i],marker=MARKER[i])

    px = np.array([-10.0, 15.0])
    py = (0.0-Wtmp2[0,i]-Wtmp2[1,i]*px)/Wtmp2[2,i]
    py1 = (1.0-Wtmp2[0,i]-Wtmp2[1,i]*px)/Wtmp2[2,i]
    plt.plot(px,py,linestyle='dashed',color=COLOR[i],label="L%d(X1,x2)=0"%i)
    plt.plot(px,py1,color=COLOR[i],label="L%d(X1,X2)=1"%i)
plt.legend()
plt.show()
```



Here we can see that one versus all technique would not work for the classes that are collinier. The class 1 is not distinguishing other classes.

In [112]:
```
plot_decision_regions(X2,y2,mclassifier2)
```

```
C:\Users\anupz\AppData\Local\Temp/ipykernel_1744/1492262336.py:21: UserWarning: You passed
a edgecolor/edgecolors ('black') for an unfilled marker ('x').  Matplotlib is ignoring the
edgecolor in favor of the facecolor.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],y=X[y == cl, 1],alpha=0.8,
```



Feature expansion

In Feature expansion, we add a dimension to the dataset to make the data more separable.

For adding the new dimension we are using the dimension as $\sigma(x)$ such that ,

$\sigma(x) = ||x - p||$

which is the distance between x and p.

Strategy 1. Chosing the point p visually for synthetic data sets

In Synthetic data 1 , Lets chose the center of the cluster of class 1 so that the other two classes will be far enough.

In Synthetic data 2 the classes are collinear. Hence we cannot use one versus all technique for classification. Hence we add one dimension to the data so that the classes will be separable by one versus all technique in higher dimension.

If we chose the center of the middle cluster as point p , then both clusters on side will raise in the magnitude of the distance from point p in the higher dimension while the cluster in which p belongs will stay closer to the point. This breaks the linearity of the dataset.

In [295]:
```
# Find the center of the cluster in the middle for Synthetic data 1
tmp1 = np.where(y1 == 1)
tmp1_value = X1[tmp1[0],:]
X1_tmp1 = np.sum(tmp1_value[:,0])/len(tmp1_value[:,0])
X2_tmp1 = np.sum(tmp1_value[:,1])/len(tmp1_value[:,1])
print(X1_tmp1,X2_tmp1)
```

```
5.942200000000001 -0.08425999999999999
```

For synthetic data 1 , p can be chosen as [5.95,-0.08]

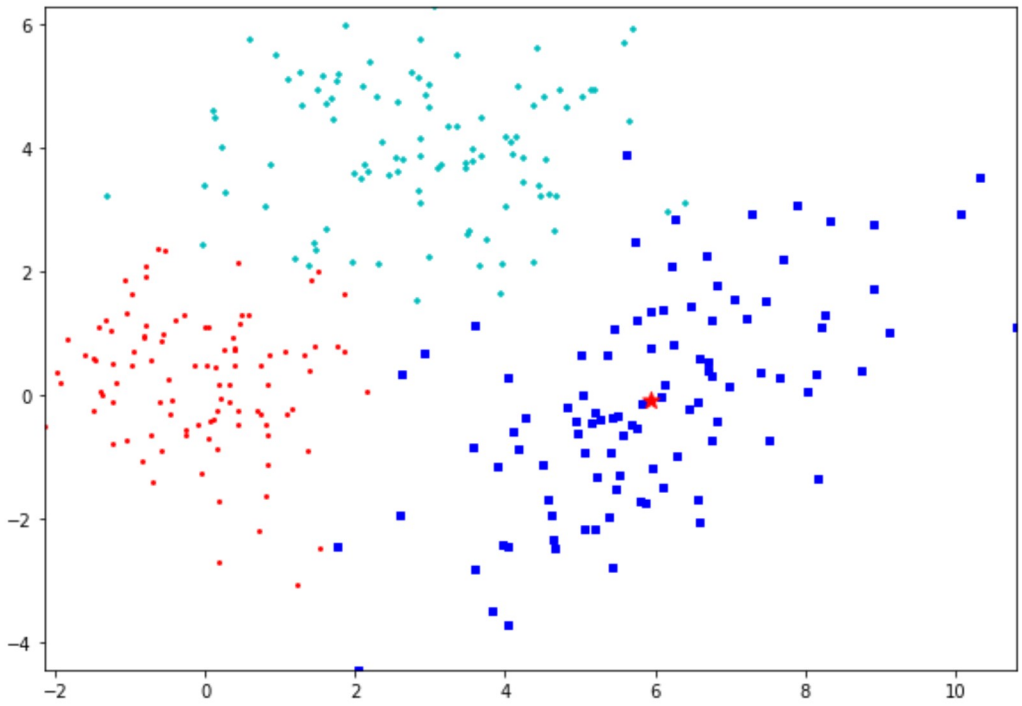
In [296]:
```
# Find the center of the cluster in the middle for Synthetic data 2
tmp2 = np.where(y1 == 1)
tmp2_value = X2[tmp[0],:]
X1_tmp2 = np.sum(tmp2_value[:,0])/len(tmp2_value[:,0])
X2_tmp2 = np.sum(tmp2_value[:,1])/len(tmp2_value[:,1])
print(X1_tmp2,X2_tmp2)
```

```
4.05115 0.09334000000000003
```

For synthetic data 2 , p can be chosen as [4.05,0.09]

In [115]:
```
# Lets plot the dataset along with point p
x121max = np.max(X1[:,0])
x121min = np.min(X1[:,0])
x221max = np.max(X1[:,1])
x221min = np.min(X1[:,1])
plt.xlim(x121min, x121max)
plt.ylim(x221min, x221max)
for i in [0,1,2]:
    p2 = np.where(y2 == i)
    X2_value = X1[p2[0],:]
    plt.scatter(X2_value[:,0],X2_value[:,1],
                s=15,color=COLOR[i],marker=MARKER[i])
plt.scatter(X1_tmp1,X2_tmp1,s=100,color='red',marker='*')
```

Out[115]:
```
<matplotlib.collections.PathCollection at 0x2200d6aaaf0>
```



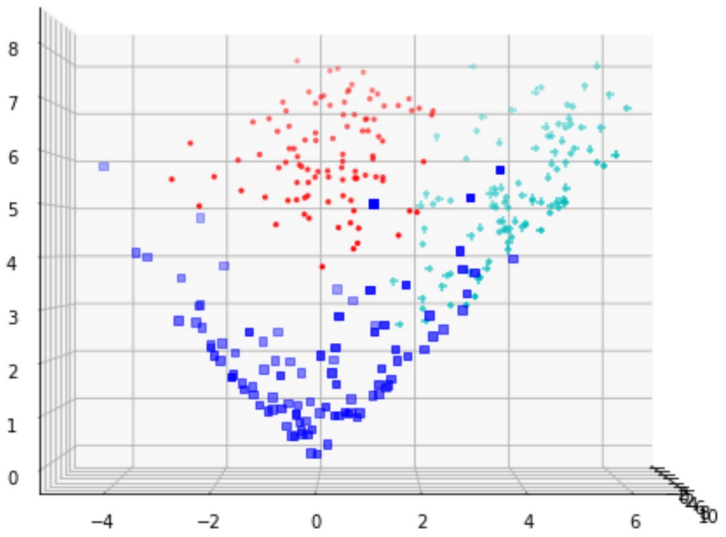
The red star is our chosen point p for synthetic data 1

In [297]:
```
# Calculate the distance of every point from the point p for synthetic data 1
pnorm1 = np.zeros(len(X1))

for i in range(len(X1)):
    pnorm1[i] = np.linalg.norm( np.array([X1[i][0]-X1_tmp1 , X1[i][1]-X2_tmp1]))
```

In [117]:
```
reshaped_pnorm1 = pnorm1.reshape((300,1))
```

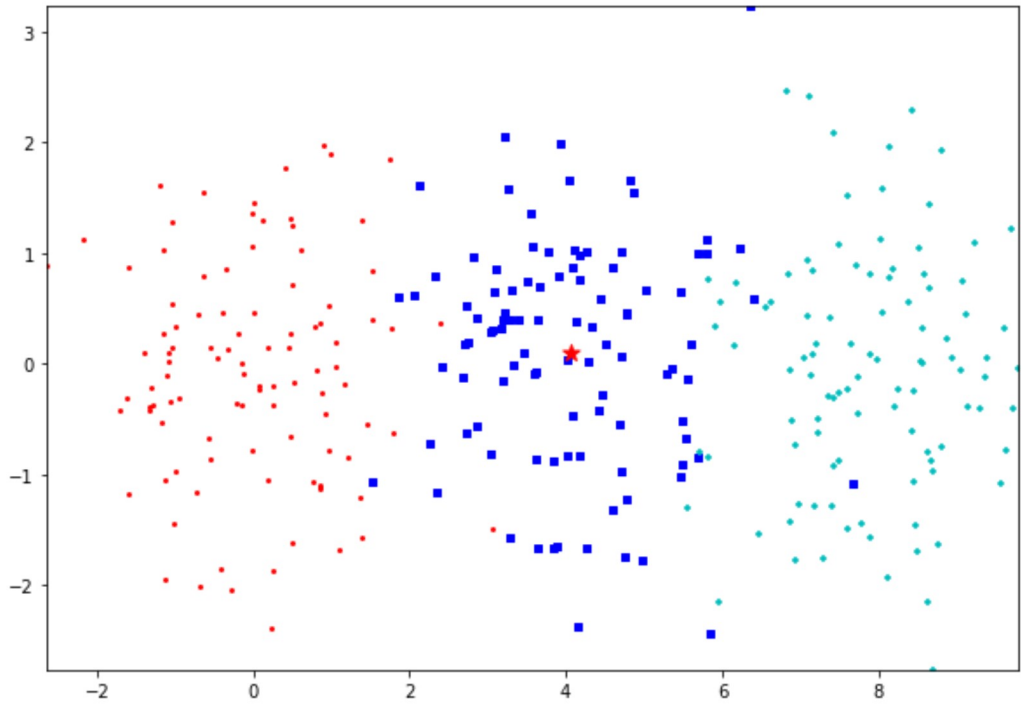
In [118]:
```
# Add the created dimension to dataset
X1_new = np.append(X1,reshaped_pnorm1,axis=1)
```

In [119]:
```
# Plot the data set with new dimension

plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')

for i in range(len(np.unique(y1))):
    temp = np.where(y2==i)
    X_values = X1_new[temp[0],:]
    ax.scatter3D(X_values[:,0],X_values[:,1],
                 X_values[:,2],color=COLOR[i],marker=MARKER[i])
ax.view_init(0,0)
```



In [298]:
```
# Lets plot the dataset along with point p for synthetic data 2
x12max = np.max(X2[:,0])
x12min = np.min(X2[:,0])
x22max = np.max(X2[:,1])
x22min = np.min(X2[:,1])
plt.xlim(x12min, x12max)
plt.ylim(x22min, x22max)
for i in [0,1,2]:
    p2 = np.where(y2 == i)
    X2_value = X2[p2[0],:]
    plt.scatter(X2_value[:,0],X2_value[:,1],s=15,color=COLOR[i],marker=MARKER[i])
plt.scatter(X1_tmp2,X2_tmp2,s=100,color='red',marker='*')
```

Out[298]:
```
<matplotlib.collections.PathCollection at 0x22036260ca0>
```



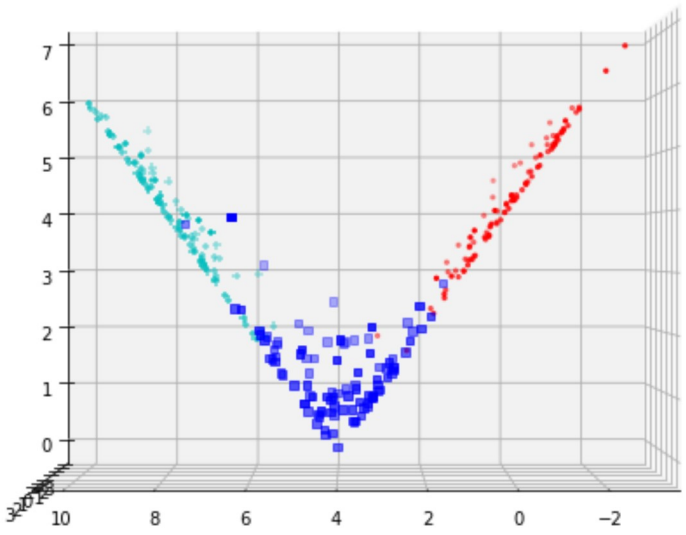
The red star in the middle of the cluster is the point p

In [121]:
```
# Calculate the distance of every point from the point p
pnorm2 = np.zeros(len(X2))

for i in range(len(X2)):
    pnorm2[i] = np.linalg.norm( np.array([X2[i][0]-X1_tmp2 , X2[i][1]-X2_tmp2]))
```

In [122]:
```
reshaped_pnorm2 = pnorm2.reshape((300,1))
```

In [123]:
```
# Add the created dimension to dataset
X2_new = np.append(X2,reshaped_pnorm2,axis=1)
```

In [124]:
```
# Plot the data set with new dimension

plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')

for i in range(len(np.unique(y2))):
    temp = np.where(y2==i)
    X_values = X2_new[temp[0],:]
    ax.scatter3D(X_values[:,0],X_values[:,1],X_values[:,2],
                 color=COLOR[i],marker=MARKER[i])
ax.view_init(0,90)
```



Through this plot we can clearly see that the cluster could be separated using one versus all technique

In [299]:
```
# Run the exoeriments for these expanded datasets
dataX_new1 = [X1_new,X2_new]
datay_new1 = [y1,y2]
data_names_new1 = ['Synthetic data 1 ',
                   'Synthetic data 2']
```

In [126]:
```
run_experiments(dataX_new1,datay_new1,data_names_new1, classifiers)
```

```
Synthetic data 1 : mCLESS: Acc.(mean,std) = (96.45,1.78)%; E-time= 0.00258
Synthetic data 1 : Logistic Regr: Acc.(mean,std) = (96.15,1.86)%; E-time= 0.01212
Synthetic data 1 : KNeighbors-5 : Acc.(mean,std) = (95.38,1.92)%; E-time= 0.00803
Synthetic data 1 : RBF SVM : Acc.(mean,std) = (96.76,1.88)%; E-time= 0.00598
Synthetic data 1 : Random Forest: Acc.(mean,std) = (95.19,2.54)%; E-time= 0.13319
Classifiers max: RBF SVM = 96.76

Synthetic data 2: mCLESS: Acc.(mean,std) = (94.52,2.20)%; E-time= 0.00253
Synthetic data 2: Logistic Regr: Acc.(mean,std) = (94.59,2.24)%; E-time= 0.01102
Synthetic data 2: KNeighbors-5 : Acc.(mean,std) = (94.26,2.07)%; E-time= 0.00771
Synthetic data 2: RBF SVM : Acc.(mean,std) = (94.49,2.14)%; E-time= 0.00670
Synthetic data 2: Random Forest: Acc.(mean,std) = (94.82,2.22)%; E-time= 0.13341
Classifiers max: Random Forest= 94.82
```

From the above results we can see that the mCLESS classifier's accuracy increased from 71.04% to 94.52% just by adding one dimension to Synthetic data 2. Comparing with the accuracies of other classifiers, mCLESS is giving the same accuracy. Also it can be observed that on both of the datasets all classifiers are giving almost same accuracies.

Strategy 2. Choose center of the data as point p

If point p is the center of the dataset , then all the classes are pushed away in higher dimension with the magnitude of the distance of points from point p. This could break linearity in the higher dimension.

In [127]:
```
# Find the center are add the dimension to the data sets

centerX1 = np.mean(X1,axis=0)
X1mcenterX1 = X1-centerX1
normX1mcenterX1 = np.linalg.norm(X1mcenterX1, axis=1)
newX1 = np.append(X1, normX1mcenterX1.reshape((300, 1)), axis=1)

centerX2 = np.mean(X2,axis=0)
X2mcenterX2 = X2-centerX2
normX2mcenterX2 = np.linalg.norm(X2mcenterX2, axis=1)
newX2 = np.append(X2, normX2mcenterX2.reshape((300, 1)), axis=1)

centerX3 = np.mean(X3,axis=0)
X3mcenterX3 = X3-centerX3
normX3mcenterX3 = np.linalg.norm(X3mcenterX3, axis=1)
newX3 = np.append(X3, normX3mcenterX3.reshape((150, 1)), axis=1)

centerX4 = np.mean(X4,axis=0)
X4mcenterX4 = X4-centerX4
normX4mcenterX4 = np.linalg.norm(X4mcenterX4, axis=1)
newX4 = np.append(X4, normX4mcenterX4.reshape((178, 1)), axis=1)
```

In [128]:
```
# Creating new the data parameter for run_experminents function for the expanded dataset
dataX_new2 = [newX1,newX2,newX3,newX4]
```

In [129]:
```
run_experiments(dataX_new2,data_y,data_names, classifiers)
```

```
Synthetic Data 1: mCLESS: Acc.(mean,std) = (95.59,2.22)%; E-time= 0.00262
Synthetic Data 1: Logistic Regr: Acc.(mean,std) = (96.22,1.97)%; E-time= 0.01122
Synthetic Data 1: KNeighbors-5 : Acc.(mean,std) = (95.38,1.84)%; E-time= 0.00819
Synthetic Data 1: RBF SVM : Acc.(mean,std) = (95.89,1.98)%; E-time= 0.00610
Synthetic Data 1: Random Forest: Acc.(mean,std) = (95.93,2.00)%; E-time= 0.13624
Classifiers max: Logistic Regr= 96.22

Synthetic Data 2: mCLESS: Acc.(mean,std) = (94.42,2.26)%; E-time= 0.00415
Synthetic Data 2: Logistic Regr: Acc.(mean,std) = (94.47,2.32)%; E-time= 0.01711
Synthetic Data 2: KNeighbors-5 : Acc.(mean,std) = (94.24,2.06)%; E-time= 0.01050
Synthetic Data 2: RBF SVM : Acc.(mean,std) = (94.48,2.18)%; E-time= 0.00803
Synthetic Data 2: Random Forest: Acc.(mean,std) = (94.77,2.28)%; E-time= 0.14318
Classifiers max: Random Forest= 94.77

Iris Data: mCLESS: Acc.(mean,std) = (95.41,2.44)%; E-time= 0.00173
Iris Data: Logistic Regr: Acc.(mean,std) = (94.46,2.45)%; E-time= 0.01351
Iris Data: KNeighbors-5 : Acc.(mean,std) = (96.78,2.14)%; E-time= 0.00583
Iris Data: RBF SVM : Acc.(mean,std) = (97.11,1.95)%; E-time= 0.00442
Iris Data: Random Forest: Acc.(mean,std) = (95.00,2.64)%; E-time= 0.12386
Classifiers max: RBF SVM = 97.11

Wine Data: mCLESS: Acc.(mean,std) = (98.54,1.34)%; E-time= 0.00221
Wine Data: Logistic Regr: Acc.(mean,std) = (95.63,2.57)%; E-time= 0.02162
Wine Data: KNeighbors-5 : Acc.(mean,std) = (94.44,2.61)%; E-time= 0.00653
Wine Data: RBF SVM : Acc.(mean,std) = (98.33,1.54)%; E-time= 0.00552
Wine Data: Random Forest: Acc.(mean,std) = (98.48,1.60)%; E-time= 0.12758
Classifiers max: mCLESS= 98.54
```

Here we can see that the accuracy of Iris data set improved from 82.96% to 95.41%. Most of the other datasets accuracy remained the same.

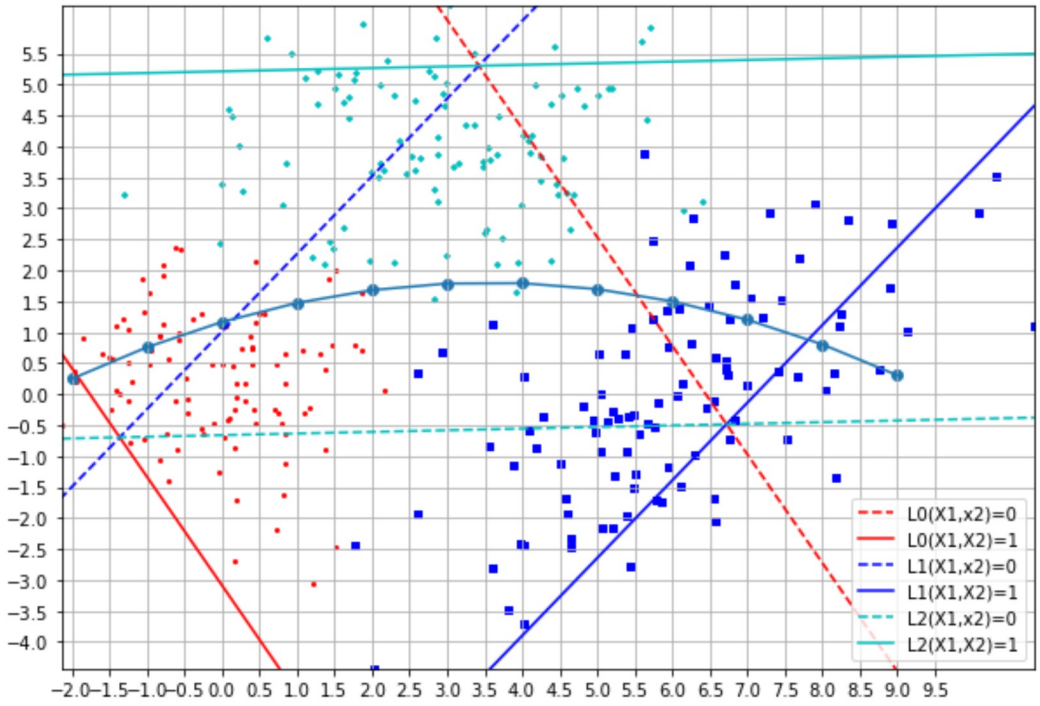
Strategy 3. Linesearch along the line of least squares to find the best p

A degree 2 polynomial is calculated and linesearch is performed on the line represented by this polynomial. The polynomial degree of 2 is chosen because it can provide better fit over a dataset that has three classes (clusters) compared to a linear fit. On the other hand, higher degree polynomial would cause overfitting.

In [301]:
```
# Polyfit for Synthetic data 1
polyfit_coeff1 = np.polyfit(X1[:,0], X1[:,1], 2)
```

In [300]:
```
fitLine_X1_values = np.arange(-2.0, 10.0, 1.0)
fitLine_y1_values = np.polyval(polyfit_coeff1,fitLine_X1_values)
fitLine_X1_values = fitLine_X1_values.reshape((fitLine_X1_values.shape[0], 1))
fitLine_y1_values = fitLine_y1_values.reshape((fitLine_y1_values.shape[0], 1))
fitLine_p1_values = np.hstack((fitLine_X1_values, fitLine_y1_values))
```

In [302]:
```
COLOR = ['r','b','c']
MARKER = ['.','s','+','*']
# Get W by running the whole dataset on classifier
mclassifier = mCLESS()
mclassifier.fit(X1, y1)
Wtmp = mclassifier.W

plt.figure(figsize=(10,7))
# Limits of the plot
x1max = np.max(X1[:,0])
x1min = np.min(X1[:,0])
x2max = np.max(X1[:,1])
x2min = np.min(X1[:,1])
plt.xlim(x1min, x1max)
plt.ylim(x2min, x2max)

for i in [0,1,2]:
    p = np.where(y1 == i)
    X_value = X1[p[0],:]
    plt.scatter(X_value[:,0],X_value[:,1],s=15,color=COLOR[i],marker=MARKER[i])

    px = np.array([-10.0, 15.0])
    py = (0.0-Wtmp[0,i]-Wtmp[1,i]*px)/Wtmp[2,i]
    py1 = (1.0-Wtmp[0,i]-Wtmp[1,i]*px)/Wtmp[2,i]
    plt.plot(px,py,linestyle='dashed',color=COLOR[i],label="L%d(X1,x2)=0"%i)
    plt.plot(px,py1,color=COLOR[i],label="L%d(X1,X2)=1"%i)
plt.plot(fitLine_X1_values,fitLine_y1_values)
plt.scatter(fitLine_X1_values,fitLine_y1_values)
plt.legend()
plt.grid()
plt.xticks(np.arange(-2, 10, step=0.5))
plt.yticks(np.arange(-4, 6, step=0.5))

plt.show()
```



In [246]:
```
def lineSearch(X, y, pvalues, pvalue_indices, name,classifiers):
    for i in range(pvalues.shape[0]):
        print('Testing p =', pvalues[i,:])
        Xmp = X.copy()
        Xmp[:,pvalue_indices[0]] -= pvalues[i,0]
        Xmp[:,pvalue_indices[1]] -= pvalues[i,1]
        normXmp = np.linalg.norm(Xmp, axis=1)
        newX = np.append(X, normXmp.reshape((X.shape[0], 1)), axis=1)
        run_experiments([newX], [y], name, classifiers)
```

In [303]:
```
lineSearch(X1, y1, fitLine_p1_values, [0, 1], ['Data1'], [mCLESS()])
```

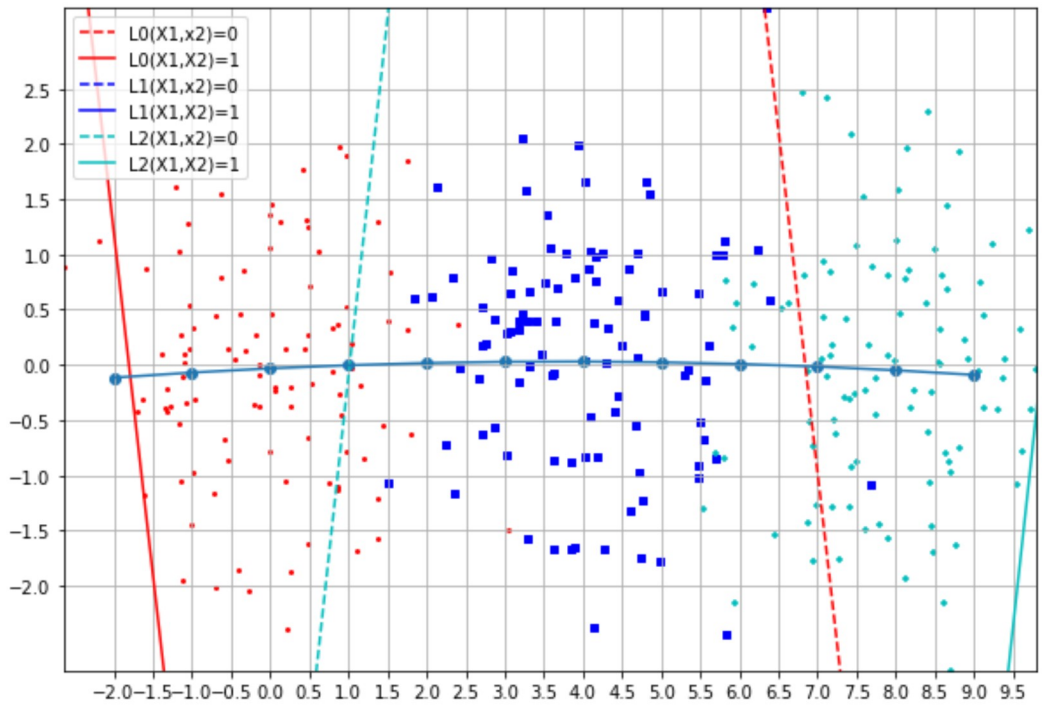
```
Testing p = [-2.          0.24671932]
Data1: mCLESS: Acc.(mean,std) = (93.67,2.39)%; E-time= 0.00126
Classifiers max: mCLESS= 93.67

Testing p = [-1.          0.75649821]
Data1: mCLESS: Acc.(mean,std) = (93.98,2.33)%; E-time= 0.00110
Classifiers max: mCLESS= 93.98

Testing p = [0.         1.16535816]
Data1: mCLESS: Acc.(mean,std) = (94.98,2.03)%; E-time= 0.00120
Classifiers max: mCLESS= 94.98

Testing p = [1.         1.47329916]
Data1: mCLESS: Acc.(mean,std) = (96.56,1.95)%; E-time= 0.00101
Classifiers max: mCLESS= 96.56

Testing p = [2.        1.6803212]
Data1: mCLESS: Acc.(mean,std) = (95.78,2.13)%; E-time= 0.00110
Classifiers max: mCLESS= 95.78

Testing p = [3.        1.7864243]
Data1: mCLESS: Acc.(mean,std) = (95.53,2.30)%; E-time= 0.00095
Classifiers max: mCLESS= 95.53

Testing p = [4.         1.79160845]
Data1: mCLESS: Acc.(mean,std) = (95.90,2.10)%; E-time= 0.00128
Classifiers max: mCLESS= 95.90

Testing p = [5.         1.69587364]
Data1: mCLESS: Acc.(mean,std) = (96.42,1.93)%; E-time= 0.00108
Classifiers max: mCLESS= 96.42

Testing p = [6.         1.49921989]
Data1: mCLESS: Acc.(mean,std) = (96.82,1.69)%; E-time= 0.00137
Classifiers max: mCLESS= 96.82

Testing p = [7.         1.20164719]
Data1: mCLESS: Acc.(mean,std) = (96.34,1.96)%; E-time= 0.00139
Classifiers max: mCLESS= 96.34

Testing p = [8.         0.80315554]
Data1: mCLESS: Acc.(mean,std) = (95.73,2.03)%; E-time= 0.00145
Classifiers max: mCLESS= 95.73

Testing p = [9.         0.30374493]
Data1: mCLESS: Acc.(mean,std) = (95.38,2.12)%; E-time= 0.00104
Classifiers max: mCLESS= 95.38
```

The best accuracy among these is 96.82% which is almost same as the accuracy befoew the feature expansion(96.89%)

In [304]:
```
# Polyfit for Synthetic data 2
polyfit_coeff2 = np.polyfit(X2[:,0], X2[:,1], 2)
```

In [249]:
```
fitLine_X2_values = np.arange(-2.0, 10.0, 1.0)
fitLine_y2_values = np.polyval(polyfit_coeff2,fitLine_X2_values)
fitLine_X2_values = fitLine_X2_values.reshape((fitLine_X2_values.shape[0], 1))
fitLine_y2_values = fitLine_y2_values.reshape((fitLine_y2_values.shape[0], 1))
fitLine_p2_values = np.hstack((fitLine_X2_values, fitLine_y2_values))
```

In [250]:
```
COLOR = ['r','b','c']
MARKER = ['.','s','+','*']
# Get W by running the whole dataset on classifier
mclassifier2 = mCLESS()
mclassifier2.fit(X2, y2)
Wtmp2 = mclassifier2.W

plt.figure(figsize=(10,7))
# Limits of the plot
x1max = np.max(X2[:,0])
x1min = np.min(X2[:,0])
x2max = np.max(X2[:,1])
x2min = np.min(X2[:,1])
plt.xlim(x1min, x1max)
plt.ylim(x2min, x2max)

for i in [0,1,2]:
    p = np.where(y1 == i)
    X_value = X2[p[0],:]
    plt.scatter(X_value[:,0],X_value[:,1],s=15,color=COLOR[i],marker=MARKER[i])

    px = np.array([-10.0, 15.0])
    py = (0.0-Wtmp2[0,i]-Wtmp2[1,i]*px)/Wtmp2[2,i]
    py1 = (1.0-Wtmp2[0,i]-Wtmp2[1,i]*px)/Wtmp2[2,i]
    plt.plot(px,py,linestyle='dashed',color=COLOR[i],label="L%d(X1,x2)=0"%i)
    plt.plot(px,py1,color=COLOR[i],label="L%d(X1,X2)=1"%i)
plt.plot(fitLine_X2_values,fitLine_y2_values)
plt.scatter(fitLine_X2_values,fitLine_y2_values)
plt.legend()
plt.grid()
plt.xticks(np.arange(-2, 10, step=0.5))
plt.yticks(np.arange(-2, 3, step=0.5))

plt.show()
```



In [251]:
```
lineSearch(X2, y2, fitLine_p2_values, [0, 1], ['Data2'], [mCLESS()])
```

```
Testing p = [-2.         -0.11692225]
Data2: mCLESS: Acc.(mean,std) = (76.90,2.78)%; E-time= 0.00126
Classifiers max: mCLESS= 76.90

Testing p = [-1.         -0.07058173]
Data2: mCLESS: Acc.(mean,std) = (82.40,2.70)%; E-time= 0.00139
Classifiers max: mCLESS= 82.40

Testing p = [ 0.         -0.03305021]
Data2: mCLESS: Acc.(mean,std) = (86.19,4.00)%; E-time= 0.00151
Classifiers max: mCLESS= 86.19

Testing p = [ 1.         -0.00432769]
Data2: mCLESS: Acc.(mean,std) = (87.71,3.54)%; E-time= 0.00126
Classifiers max: mCLESS= 87.71

Testing p = [2.         0.01558582]
Data2: mCLESS: Acc.(mean,std) = (91.64,2.73)%; E-time= 0.00120
Classifiers max: mCLESS= 91.64

Testing p = [3.         0.02669033]
Data2: mCLESS: Acc.(mean,std) = (93.87,2.42)%; E-time= 0.00138
Classifiers max: mCLESS= 93.87

Testing p = [4.         0.02898584]
Data2: mCLESS: Acc.(mean,std) = (94.48,2.20)%; E-time= 0.00136
Classifiers max: mCLESS= 94.48

Testing p = [5.         0.02247235]
Data2: mCLESS: Acc.(mean,std) = (94.31,2.11)%; E-time= 0.00142
Classifiers max: mCLESS= 94.31

Testing p = [6.         0.00714985]
Data2: mCLESS: Acc.(mean,std) = (91.20,2.67)%; E-time= 0.00143
Classifiers max: mCLESS= 91.20

Testing p = [ 7.         -0.01698165]
Data2: mCLESS: Acc.(mean,std) = (87.25,3.64)%; E-time= 0.00159
Classifiers max: mCLESS= 87.25

Testing p = [ 8.         -0.04992215]
Data2: mCLESS: Acc.(mean,std) = (83.75,4.27)%; E-time= 0.00143
Classifiers max: mCLESS= 83.75

Testing p = [ 9.         -0.09167165]
Data2: mCLESS: Acc.(mean,std) = (82.33,2.70)%; E-time= 0.00126
Classifiers max: mCLESS= 82.33
```

The best accuracy among all is at the point [4.0,0.02] which is almost same as the accuracy atthe center of the middle cluster [4.05,0.09] which we chose by using strategy 1.

Polyfit for Iris Data and Wine Data

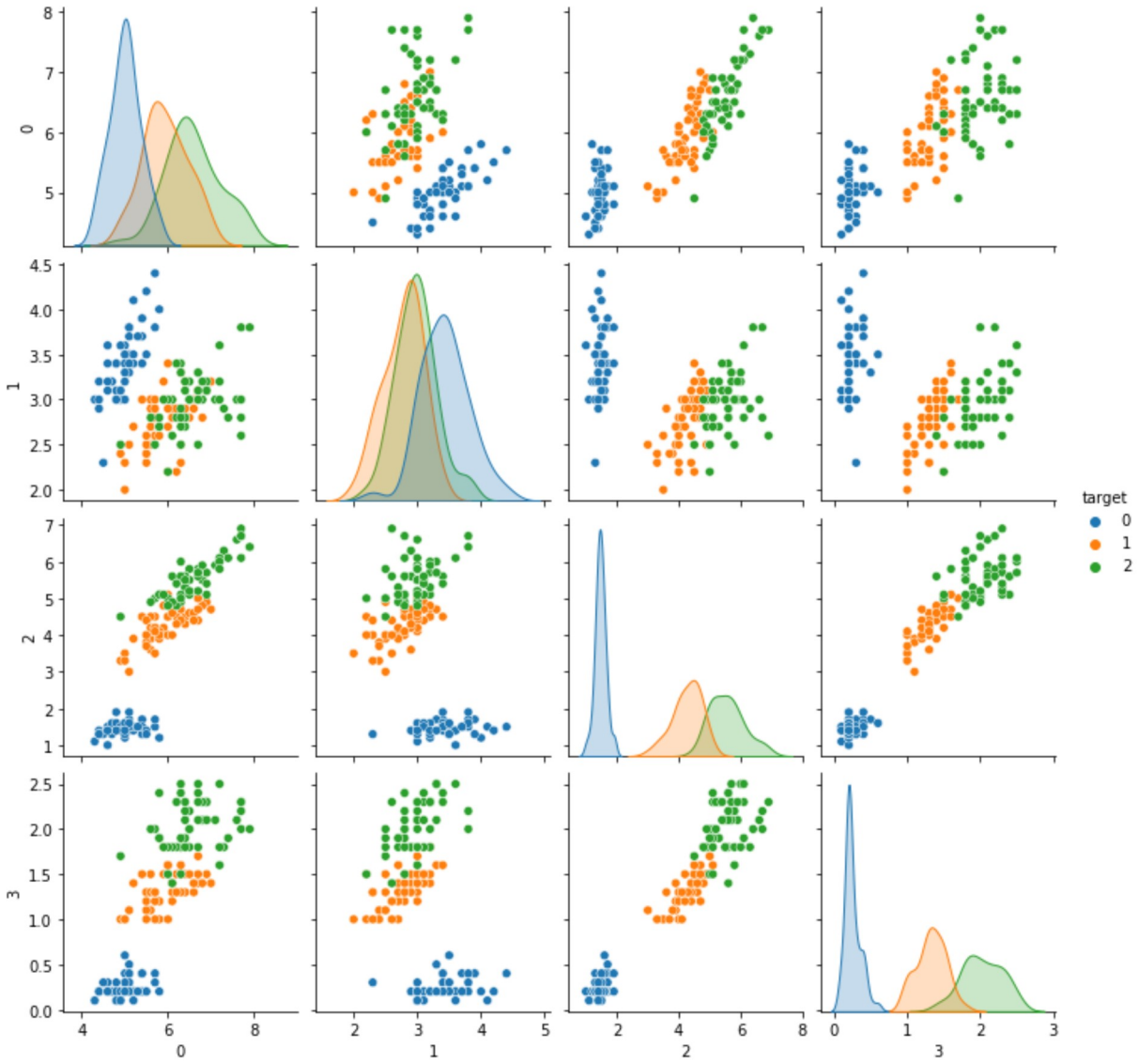
Since these datasets are higher dimensional datasets, we cannot really visualize the data sets. For the purpose of better understanding the data, consider the pairplots of features to experiment on these datasets

In [252]:
```
import seaborn as sns
import pandas as pd
```

In [253]:
```
df3 = pd.DataFrame(X3)
df3['target'] = y3
```

In [254]:
```
sns.pairplot(df3, hue='target', palette='tab10')
```

Out[254]: <seaborn.axisgrid.PairGrid at 0x22026836d30>



In the Iris data, the classes can be viewed with less overlap by selecting the features at index 2 and 3. So lets consider feature pair 2-3 to polyfit the data

In [255]:
```
# Polyfit
polyfit_coeff3 = np.polyfit(X3[:,2], X3[:,3], 2)
```

In [256]:
```
fitLine_X3_values = np.arange(-2.0, 10.0, 1.0)
fitLine_y3_values = np.polyval(polyfit_coeff3,fitLine_X3_values)
fitLine_X3_values = fitLine_X3_values.reshape((fitLine_X3_values.shape[0], 1))
fitLine_y3_values = fitLine_y3_values.reshape((fitLine_y3_values.shape[0], 1))
fitLine_p3_values = np.hstack((fitLine_X3_values, fitLine_y3_values))
```

In [257]:
```
lineSearch(X3, y3, fitLine_p3_values, [2, 3], ['Data3'],[mCLESS()])
```

```
Testing p = [-2.         -1.26472372]
Data3: mCLESS: Acc.(mean,std) = (94.76,2.79)%; E-time= 0.00164
Classifiers max: mCLESS= 94.76

Testing p = [-1.         -0.82318345]
Data3: mCLESS: Acc.(mean,std) = (95.48,2.55)%; E-time= 0.00139
Classifiers max: mCLESS= 95.48

Testing p = [ 0.         -0.38678103]
Data3: mCLESS: Acc.(mean,std) = (96.72,2.25)%; E-time= 0.00108
Classifiers max: mCLESS= 96.72

Testing p = [1.         0.04448354]
Data3: mCLESS: Acc.(mean,std) = (97.43,2.07)%; E-time= 0.00110
Classifiers max: mCLESS= 97.43

Testing p = [2.         0.47061026]
Data3: mCLESS: Acc.(mean,std) = (97.20,1.90)%; E-time= 0.00097
Classifiers max: mCLESS= 97.20

Testing p = [3.         0.89159913]
Data3: mCLESS: Acc.(mean,std) = (96.22,2.47)%; E-time= 0.00107
Classifiers max: mCLESS= 96.22

Testing p = [4.         1.30745016]
Data3: mCLESS: Acc.(mean,std) = (95.13,2.68)%; E-time= 0.00113
Classifiers max: mCLESS= 95.13

Testing p = [5.         1.71816334]
Data3: mCLESS: Acc.(mean,std) = (94.78,2.66)%; E-time= 0.00101
Classifiers max: mCLESS= 94.78

Testing p = [6.         2.12373867]
Data3: mCLESS: Acc.(mean,std) = (94.74,2.84)%; E-time= 0.00113
Classifiers max: mCLESS= 94.74

Testing p = [7.         2.52417615]
Data3: mCLESS: Acc.(mean,std) = (94.63,2.94)%; E-time= 0.00117
Classifiers max: mCLESS= 94.63

Testing p = [8.         2.91947578]
Data3: mCLESS: Acc.(mean,std) = (94.50,3.16)%; E-time= 0.00111
Classifiers max: mCLESS= 94.50

Testing p = [9.         3.30963756]
Data3: mCLESS: Acc.(mean,std) = (94.26,3.36)%; E-time= 0.00119
Classifiers max: mCLESS= 94.26
```

Accuracy is little bit improved if the point p is chosen as [1.0,0.04]

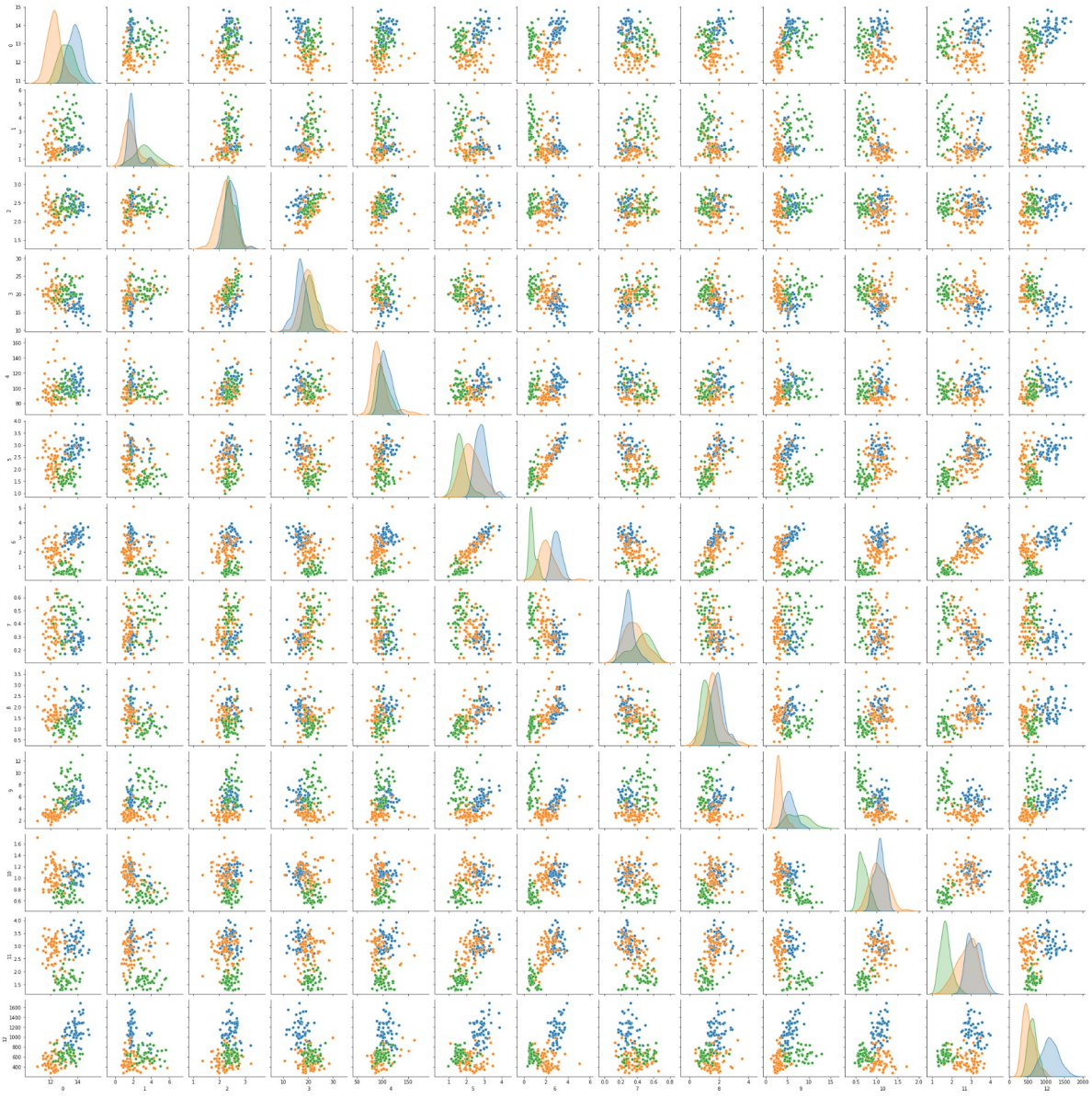
In [258]:
```
# For Wine  data set
```

In [259]:
```
df4 = pd.DataFrame(X4)
```

In [260]:
```
df4['target'] = y4
```

In [261]:
```
sns.pairplot(df4, hue='target', palette='tab10')
```

Out[261]:
```
<seaborn.axisgrid.PairGrid at 0x22029b836a0>
```



All classes can be seen with less overlapping in the feature pairs 0-11 amd 6-12. So lets zoom in to see which pair gives a better view of classes.

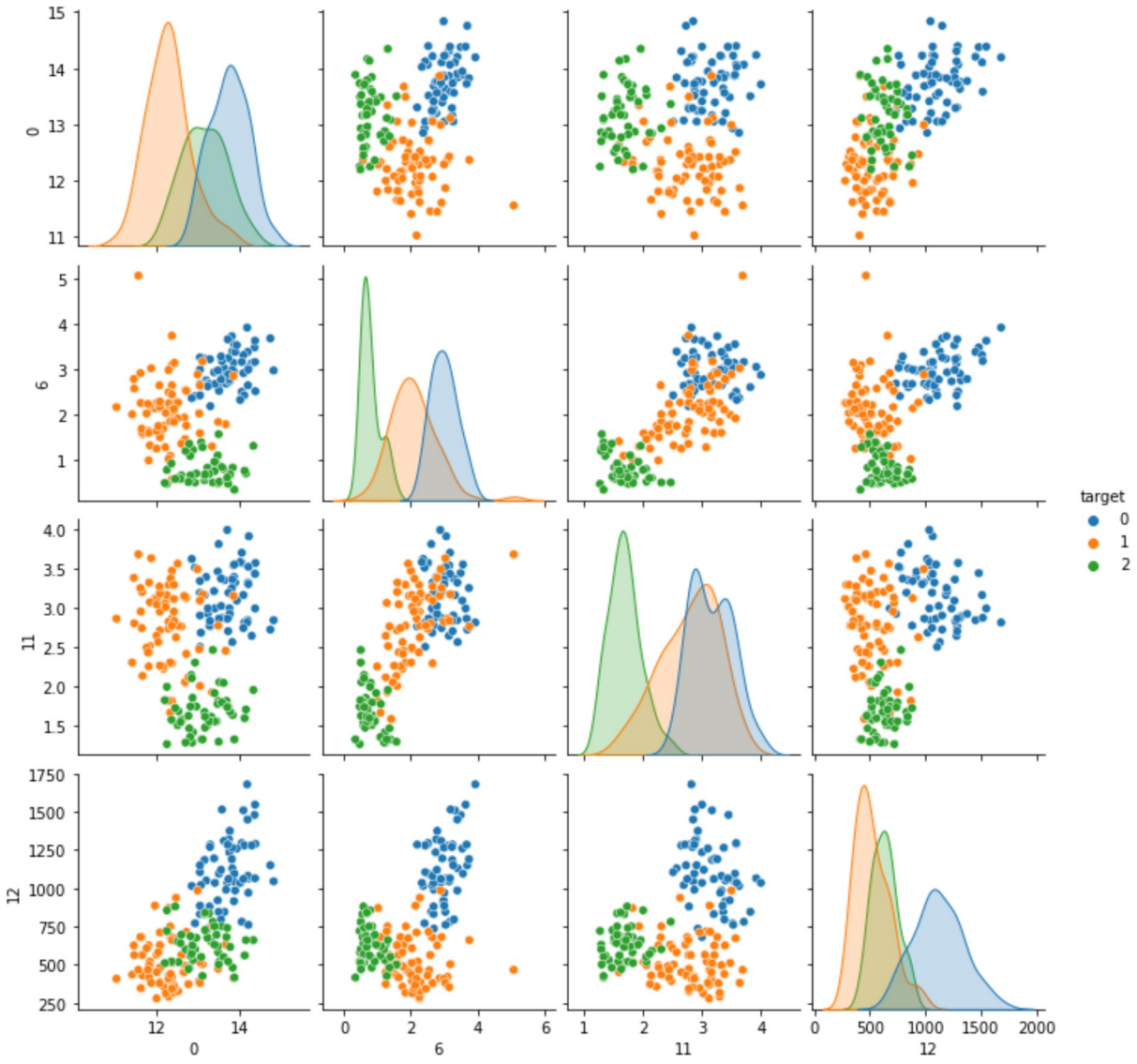
In [262]:
```
df4_select_features = df4.iloc[:, [0,6,11,12]]
```

In [263]:
```
df4_select_features['target'] = df4['target']
```

```
C:\Users\anupz\AppData\Local\Temp/ipykernel_1744/345901122.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df4_select_features['target'] = df4['target']
```

In [264]:
```
sns.pairplot(df4_select_features,hue='target',palette='tab10')
```

Out[264]:
```
<seaborn.axisgrid.PairGrid at 0x220327afbb0>
```



If we consider the pairplot plot of feature at index 6 and 12 then we can see the separability of classes in a better way.

Lets try to take features at the index 6 and 12 to polyfit the data set 4

In [265]:
```
polyfit_coeff4 = np.polyfit(X4[:,6], X4[:,12], 2)
```

In [266]:
```
fitLine_X4_values = np.arange(-2.0, 10.0, 1.0)
fitLine_y4_values = np.polyval(polyfit_coeff4,fitLine_X4_values)
fitLine_X4_values = fitLine_X4_values.reshape((fitLine_X4_values.shape[0], 1))
fitLine_y4_values = fitLine_y4_values.reshape((fitLine_y4_values.shape[0], 1))
fitLine_p4_values = np.hstack((fitLine_X4_values, fitLine_y4_values))
```

In [267]:
```
lineSearch(X4, y4, fitLine_p4_values, [6, 12], ['Data4'],[mCLESS()])
```

```
Testing p = [ -2.         985.65509507]
Data4: mCLESS: Acc.(mean,std) = (98.35,1.33)%; E-time= 0.00126
Classifiers max: mCLESS= 98.35

Testing p = [ -1.         741.41558156]
Data4: mCLESS: Acc.(mean,std) = (98.56,1.32)%; E-time= 0.00120
Classifiers max: mCLESS= 98.56

Testing p = [  0.         610.12546229]
Data4: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00101
Classifiers max: mCLESS= 98.56

Testing p = [  1.         591.78473728]
Data4: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00094
Classifiers max: mCLESS= 98.56

Testing p = [  2.         686.39340652]
Data4: mCLESS: Acc.(mean,std) = (98.57,1.38)%; E-time= 0.00111
Classifiers max: mCLESS= 98.57

Testing p = [  3.         893.95147002]
Data4: mCLESS: Acc.(mean,std) = (98.39,1.38)%; E-time= 0.00111
Classifiers max: mCLESS= 98.39

Testing p = [   4.         1214.45892776]
Data4: mCLESS: Acc.(mean,std) = (98.50,1.33)%; E-time= 0.00094
Classifiers max: mCLESS= 98.50

Testing p = [   5.         1647.91577976]
Data4: mCLESS: Acc.(mean,std) = (98.00,1.63)%; E-time= 0.00111
Classifiers max: mCLESS= 98.00

Testing p = [   6.         2194.32202601]
Data4: mCLESS: Acc.(mean,std) = (98.56,1.45)%; E-time= 0.00094
Classifiers max: mCLESS= 98.56

Testing p = [   7.         2853.67766652]
Data4: mCLESS: Acc.(mean,std) = (98.54,1.44)%; E-time= 0.00112
Classifiers max: mCLESS= 98.54

Testing p = [   8.         3625.98270127]
Data4: mCLESS: Acc.(mean,std) = (98.63,1.47)%; E-time= 0.00093
Classifiers max: mCLESS= 98.63

Testing p = [   9.         4511.23713028]
Data4: mCLESS: Acc.(mean,std) = (98.63,1.47)%; E-time= 0.00118
Classifiers max: mCLESS= 98.63
```

Accuracy of mCLESS on wine data set remains almost the same through various points

Strategy 4. Linesearch along the edges triangle formed by the centers of the cluster

In [268]:
```
# 3 clusters center
def find_centers(X,y):
    centers= np.zeros((len(np.unique(y)),X.shape[1]))
    for i in range(len(np.unique(y))):
        Xtmp_idx = np.where(y==i)
        xCenter = np.mean(X[Xtmp_idx[0]],axis=0)
        centers[i] = xCenter
    return centers
```

In [278]:
```
def plot_triangle(X,y,p_indices=[0,1]):

    centers = np.zeros((len(np.unique(y)),2))

    x12max = np.max(X[:,p_indices[0]])
    x12min = np.min(X[:,p_indices[0]])
    x22max = np.max(X[:,p_indices[1]])
    x22min = np.min(X[:,p_indices[1]])
    plt.xlim(x12min, x12max)
    plt.ylim(x22min, x22max)

    for i in [0,1,2]:
        p = np.where(y == i)
        X_value = X[p[0],:]
        xCenter = np.mean(X[p[0]],axis=0)

        centers[i] = np.array(xCenter[p_indices[0]],xCenter[p_indices[1]])
        plt.scatter(X_value[:,p_indices[0]],X_value[:,p_indices[1]],
                    s=15,color=COLOR[i],marker=MARKER[i])
        centers[i] = xCenter[p_indices]

    # since all the datasets have 3 classes we can customize the function for 3 classes
    plt.plot([centers[0][0],centers[1][0]],[centers[0][1],centers[1][1]])
    plt.plot([centers[0][0],centers[2][0]],[centers[0][1],centers[2][1]])
    plt.plot([centers[1][0],centers[2][0]],[centers[1][1],centers[2][1]])
```

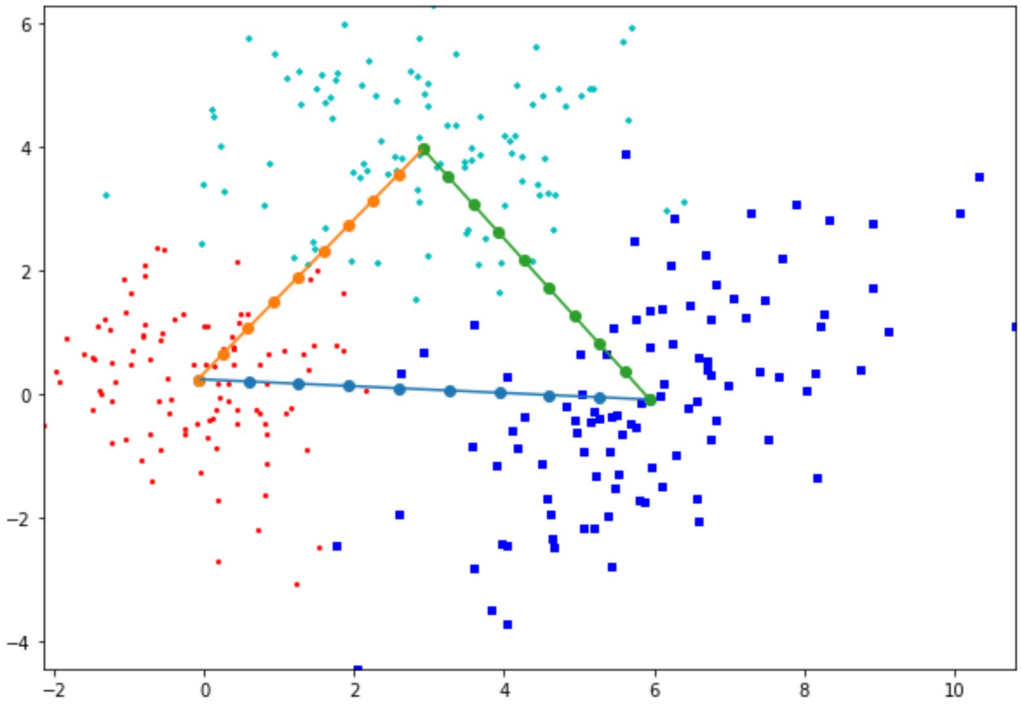
In [316]:

```
def linesearch_traingle(X,y,p_indices,num_intervals=10):
    all_centers = find_centers(X,y)
    centers = all_centers[:,p_indices]

    # For line 1
    xv1 = np.linspace(centers[0][0],centers[1][0],num_intervals)
    m1 = (centers[1][1] - centers[0][1] ) /  (centers[1][0] - centers[0][0] )
    b1 = centers[0][1] -  m1*centers[0][0]
    fx1 = m1*xv1 + b1
    xv1 = xv1.reshape((xv1.shape[0],1))
    fx1 = fx1.reshape((fx1.shape[0],1))
    p1_values = np.hstack((xv1,fx1))

    # For line 2
    xv2 = np.linspace(centers[2][0],centers[1][0],num_intervals)
    m2 = (centers[1][1] - centers[2][1] ) /  (centers[1][0] - centers[2][0] )
    b2 = centers[2][1] -  m2*centers[2][0]
    fx2 = m2*xv2 + b2
    xv2 = xv2.reshape((xv2.shape[0],1))
    fx2 = fx2.reshape((fx2.shape[0],1))
    p2_values = np.hstack((xv2,fx2))

    # For line 3
    xv3 = np.linspace(centers[0][0],centers[2][0],num_intervals)
    m3 = (centers[2][1] - centers[0][1] ) /  (centers[2][0] - centers[0][0] )
    b3 = centers[2][1] -  m3*centers[2][0]
    fx3 = m3*xv3 + b3
    xv3 = xv3.reshape((xv3.shape[0],1))
    fx3 = fx3.reshape((fx3.shape[0],1))
    p3_values = np.hstack((xv3,fx3))

    p_values = np.vstack((p1_values,p2_values,p3_values))
    lineSearch(X,y,p_values,p_indices, ["Triangle search"], [mCLESS()])

    print('\n\n\n\n\n\n\n')
    plot_triangle(X,y,p_indices)
    plt.scatter(xv1, fx1)
    plt.scatter(xv3, fx3)
    plt.scatter(xv2, fx2)
```

In [317]:
```
linesearch_traingle(X1,y1,[0,1])
```

```
Testing p = [-0.07773  0.24373]
Triangle search: mCLESS: Acc.(mean,std) = (94.59,2.07)%; E-time= 0.00122
Classifiers max: mCLESS= 94.59

Testing p = [0.59115111 0.20728667]
Triangle search: mCLESS: Acc.(mean,std) = (95.05,2.15)%; E-time= 0.00111
Classifiers max: mCLESS= 95.05

Testing p = [1.26003222 0.17084333]
Triangle search: mCLESS: Acc.(mean,std) = (96.15,2.11)%; E-time= 0.00126
Classifiers max: mCLESS= 96.15

Testing p = [1.92891333 0.1344    ]
Triangle search: mCLESS: Acc.(mean,std) = (96.34,1.90)%; E-time= 0.00141
Classifiers max: mCLESS= 96.34

Testing p = [2.59779444 0.09795667]
Triangle search: mCLESS: Acc.(mean,std) = (95.99,2.08)%; E-time= 0.00111
Classifiers max: mCLESS= 95.99

Testing p = [3.26667556 0.06151333]
Triangle search: mCLESS: Acc.(mean,std) = (96.20,2.21)%; E-time= 0.00115
Classifiers max: mCLESS= 96.20

Testing p = [3.93555667 0.02507   ]
Triangle search: mCLESS: Acc.(mean,std) = (96.58,1.84)%; E-time= 0.00092
Classifiers max: mCLESS= 96.58

Testing p = [ 4.60443778 -0.01137333]
Triangle search: mCLESS: Acc.(mean,std) = (96.73,1.70)%; E-time= 0.00108
Classifiers max: mCLESS= 96.73

Testing p = [ 5.27331889 -0.04781667]
Triangle search: mCLESS: Acc.(mean,std) = (96.62,1.72)%; E-time= 0.00099
Classifiers max: mCLESS= 96.62

Testing p = [ 5.9422  -0.08426]
Triangle search: mCLESS: Acc.(mean,std) = (96.45,1.78)%; E-time= 0.00125
Classifiers max: mCLESS= 96.45

Testing p = [2.921   3.95976]
Triangle search: mCLESS: Acc.(mean,std) = (95.68,2.12)%; E-time= 0.00111
Classifiers max: mCLESS= 95.68

Testing p = [3.25668889 3.51042444]
Triangle search: mCLESS: Acc.(mean,std) = (95.48,2.15)%; E-time= 0.00110
Classifiers max: mCLESS= 95.48

Testing p = [3.59237778 3.06108889]
Triangle search: mCLESS: Acc.(mean,std) = (95.60,2.16)%; E-time= 0.00111
Classifiers max: mCLESS= 95.60

Testing p = [3.92806667 2.61175333]
Triangle search: mCLESS: Acc.(mean,std) = (95.62,2.06)%; E-time= 0.00111
Classifiers max: mCLESS= 95.62

Testing p = [4.26375556 2.16241778]
Triangle search: mCLESS: Acc.(mean,std) = (95.89,2.05)%; E-time= 0.00097
Classifiers max: mCLESS= 95.89

Testing p = [4.59944444 1.71308222]
Triangle search: mCLESS: Acc.(mean,std) = (96.22,1.99)%; E-time= 0.00113
```

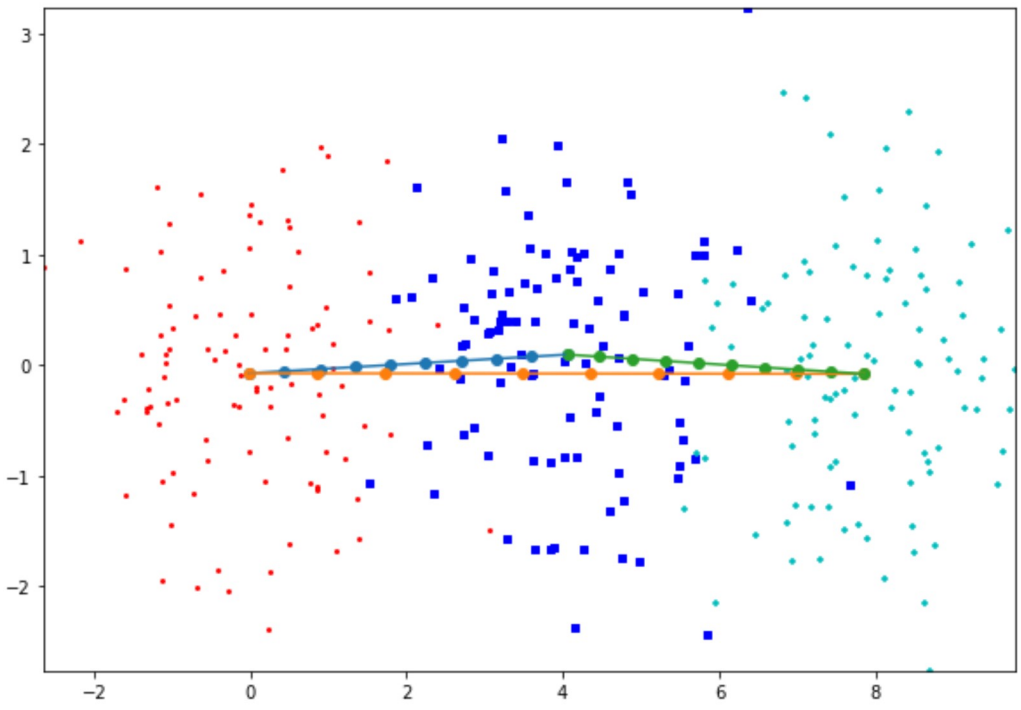
```
Classifiers max: mCLESS= 96.22

Testing p = [4.93513333 1.26374667]
Triangle search: mCLESS: Acc.(mean,std) = (96.53,1.85)%; E-time= 0.00142
Classifiers max: mCLESS= 96.53

Testing p = [5.27082222 0.81441111]
Triangle search: mCLESS: Acc.(mean,std) = (96.64,1.70)%; E-time= 0.00158
Classifiers max: mCLESS= 96.64

Testing p = [5.60651111 0.36507556]
Triangle search: mCLESS: Acc.(mean,std) = (96.53,1.76)%; E-time= 0.00131
Classifiers max: mCLESS= 96.53

Testing p = [ 5.9422  -0.08426]
Triangle search: mCLESS: Acc.(mean,std) = (96.45,1.78)%; E-time= 0.00112
Classifiers max: mCLESS= 96.45

Testing p = [-0.07773  0.24373]
Triangle search: mCLESS: Acc.(mean,std) = (94.59,2.07)%; E-time= 0.00124
Classifiers max: mCLESS= 94.59

Testing p = [0.25546222 0.65662222]
Triangle search: mCLESS: Acc.(mean,std) = (94.78,2.02)%; E-time= 0.00143
Classifiers max: mCLESS= 94.78

Testing p = [0.58865444 1.06951444]
Triangle search: mCLESS: Acc.(mean,std) = (95.57,2.09)%; E-time= 0.00125
Classifiers max: mCLESS= 95.57

Testing p = [0.92184667 1.48240667]
Triangle search: mCLESS: Acc.(mean,std) = (96.48,2.00)%; E-time= 0.00142
Classifiers max: mCLESS= 96.48

Testing p = [1.25503889 1.89529889]
Triangle search: mCLESS: Acc.(mean,std) = (95.97,2.09)%; E-time= 0.00127
Classifiers max: mCLESS= 95.97

Testing p = [1.58823111 2.30819111]
Triangle search: mCLESS: Acc.(mean,std) = (95.62,2.10)%; E-time= 0.00143
Classifiers max: mCLESS= 95.62

Testing p = [1.92142333 2.72108333]
Triangle search: mCLESS: Acc.(mean,std) = (95.27,2.24)%; E-time= 0.00145
Classifiers max: mCLESS= 95.27

Testing p = [2.25461556 3.13397556]
Triangle search: mCLESS: Acc.(mean,std) = (95.23,2.25)%; E-time= 0.00149
Classifiers max: mCLESS= 95.23

Testing p = [2.58780778 3.54686778]
Triangle search: mCLESS: Acc.(mean,std) = (95.67,2.19)%; E-time= 0.00119
Classifiers max: mCLESS= 95.67

Testing p = [2.921   3.95976]
Triangle search: mCLESS: Acc.(mean,std) = (95.68,2.12)%; E-time= 0.00115
Classifiers max: mCLESS= 95.68
```


6
4
2
0
−2
−4
−2
0
2
4
6
8
10

In [318]:
```
linesearch_traingle(X2,y2,[0,1])
```

```
Testing p = [-0.01639 -0.07605]
Triangle search: mCLESS: Acc.(mean,std) = (86.25,4.01)%; E-time= 0.00152
Classifiers max: mCLESS= 86.25

Testing p = [ 0.43555889 -0.05722889]
Triangle search: mCLESS: Acc.(mean,std) = (85.42,3.61)%; E-time= 0.00125
Classifiers max: mCLESS= 85.42

Testing p = [ 0.88750778 -0.03840778]
Triangle search: mCLESS: Acc.(mean,std) = (87.34,3.60)%; E-time= 0.00095
Classifiers max: mCLESS= 87.34

Testing p = [ 1.33945667 -0.01958667]
Triangle search: mCLESS: Acc.(mean,std) = (88.87,3.19)%; E-time= 0.00132
Classifiers max: mCLESS= 88.87

Testing p = [ 1.79140556e+00 -7.65555556e-04]
Triangle search: mCLESS: Acc.(mean,std) = (90.57,2.92)%; E-time= 0.00140
Classifiers max: mCLESS= 90.57

Testing p = [2.24335444 0.01805556]
Triangle search: mCLESS: Acc.(mean,std) = (92.33,2.43)%; E-time= 0.00147
Classifiers max: mCLESS= 92.33

Testing p = [2.69530333 0.03687667]
Triangle search: mCLESS: Acc.(mean,std) = (93.64,2.57)%; E-time= 0.00111
Classifiers max: mCLESS= 93.64

Testing p = [3.14725222 0.05569778]
Triangle search: mCLESS: Acc.(mean,std) = (93.58,2.50)%; E-time= 0.00112
Classifiers max: mCLESS= 93.58

Testing p = [3.59920111 0.07451889]
Triangle search: mCLESS: Acc.(mean,std) = (93.87,2.46)%; E-time= 0.00113
Classifiers max: mCLESS= 93.87

Testing p = [4.05115 0.09334]
Triangle search: mCLESS: Acc.(mean,std) = (94.52,2.20)%; E-time= 0.00128
Classifiers max: mCLESS= 94.52

Testing p = [ 7.85305 -0.08171]
Triangle search: mCLESS: Acc.(mean,std) = (83.42,3.97)%; E-time= 0.00099
Classifiers max: mCLESS= 83.42

Testing p = [ 7.43061667 -0.06226   ]
Triangle search: mCLESS: Acc.(mean,std) = (85.21,3.83)%; E-time= 0.00113
Classifiers max: mCLESS= 85.21

Testing p = [ 7.00818333 -0.04281   ]
Triangle search: mCLESS: Acc.(mean,std) = (87.21,3.62)%; E-time= 0.00145
Classifiers max: mCLESS= 87.21

Testing p = [ 6.58575 -0.02336]
Triangle search: mCLESS: Acc.(mean,std) = (88.93,3.18)%; E-time= 0.00138
Classifiers max: mCLESS= 88.93

Testing p = [ 6.16331667e+00 -3.91000000e-03]
Triangle search: mCLESS: Acc.(mean,std) = (90.36,2.89)%; E-time= 0.00113
Classifiers max: mCLESS= 90.36

Testing p = [5.74088333 0.01554   ]
Triangle search: mCLESS: Acc.(mean,std) = (92.57,2.43)%; E-time= 0.00126
```

```
Classifiers max: mCLESS= 92.57

Testing p = [5.31845 0.03499]
Triangle search: mCLESS: Acc.(mean,std) = (94.00,2.18)%; E-time= 0.00125
Classifiers max: mCLESS= 94.00

Testing p = [4.89601667 0.05444   ]
Triangle search: mCLESS: Acc.(mean,std) = (94.43,2.15)%; E-time= 0.00159
Classifiers max: mCLESS= 94.43

Testing p = [4.47358333 0.07389   ]
Triangle search: mCLESS: Acc.(mean,std) = (94.85,2.17)%; E-time= 0.00139
Classifiers max: mCLESS= 94.85

Testing p = [4.05115 0.09334]
Triangle search: mCLESS: Acc.(mean,std) = (94.52,2.20)%; E-time= 0.00115
Classifiers max: mCLESS= 94.52

Testing p = [-0.01639 -0.07605]
Triangle search: mCLESS: Acc.(mean,std) = (86.25,4.01)%; E-time= 0.00121
Classifiers max: mCLESS= 86.25

Testing p = [ 0.85799222 -0.07667889]
Triangle search: mCLESS: Acc.(mean,std) = (87.30,3.61)%; E-time= 0.00136
Classifiers max: mCLESS= 87.30

Testing p = [ 1.73237444 -0.07730778]
Triangle search: mCLESS: Acc.(mean,std) = (90.23,3.06)%; E-time= 0.00123
Classifiers max: mCLESS= 90.23

Testing p = [ 2.60675667 -0.07793667]
Triangle search: mCLESS: Acc.(mean,std) = (93.36,2.58)%; E-time= 0.00130
Classifiers max: mCLESS= 93.36

Testing p = [ 3.48113889 -0.07856556]
Triangle search: mCLESS: Acc.(mean,std) = (93.68,2.56)%; E-time= 0.00120
Classifiers max: mCLESS= 93.68

Testing p = [ 4.35552111 -0.07919444]
Triangle search: mCLESS: Acc.(mean,std) = (94.86,2.17)%; E-time= 0.00120
Classifiers max: mCLESS= 94.86

Testing p = [ 5.22990333 -0.07982333]
Triangle search: mCLESS: Acc.(mean,std) = (94.12,2.12)%; E-time= 0.00101
Classifiers max: mCLESS= 94.12

Testing p = [ 6.10428556 -0.08045222]
Triangle search: mCLESS: Acc.(mean,std) = (90.65,2.85)%; E-time= 0.00117
Classifiers max: mCLESS= 90.65

Testing p = [ 6.97866778 -0.08108111]
Triangle search: mCLESS: Acc.(mean,std) = (87.38,3.55)%; E-time= 0.00100
Classifiers max: mCLESS= 87.38

Testing p = [ 7.85305 -0.08171]
Triangle search: mCLESS: Acc.(mean,std) = (83.42,3.97)%; E-time= 0.00113
Classifiers max: mCLESS= 83.42
```

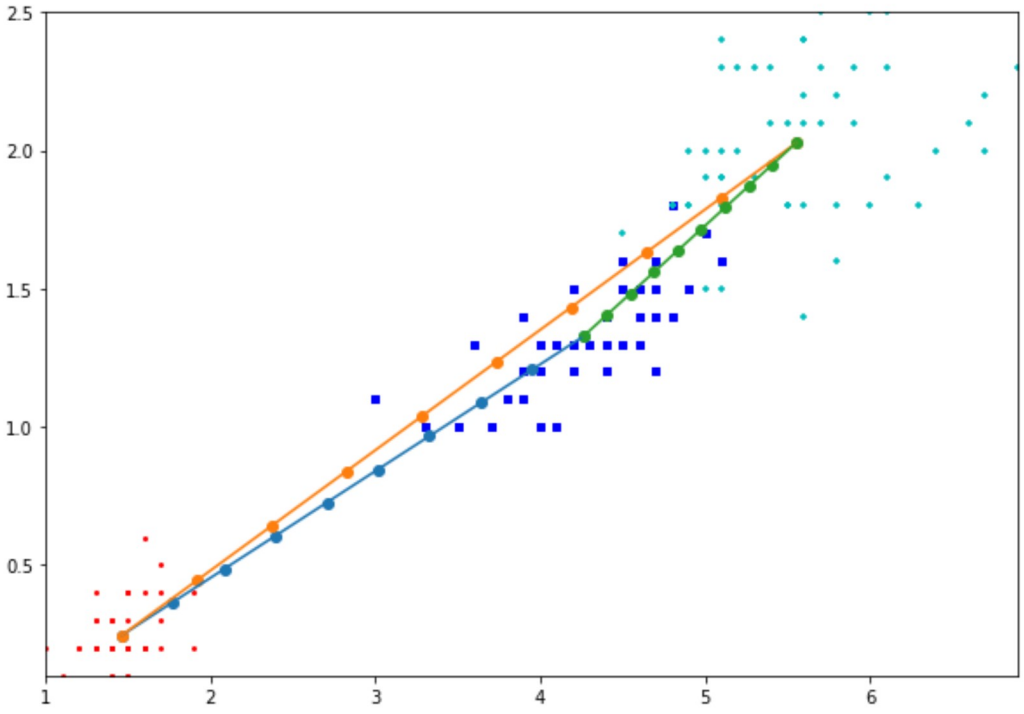


From the above results , we can observe that as $\mathbf{p}$ moves towards the center of the dataset the accuracy tends to get closer to the accuracy of the strategy 1 where $\mathbf{p}$ was chosen to be the center of the cluster of class 1.

In [319]:
```
linesearch_traingle(X3,y3,[2,3])
```

```
Testing p = [1.462 0.246]
Triangle search: mCLESS: Acc.(mean,std) = (97.33,1.87)%; E-time= 0.00137
Classifiers max: mCLESS= 97.33

Testing p = [1.77288889 0.366     ]
Triangle search: mCLESS: Acc.(mean,std) = (97.30,1.87)%; E-time= 0.00136
Classifiers max: mCLESS= 97.30

Testing p = [2.08377778 0.486     ]
Triangle search: mCLESS: Acc.(mean,std) = (97.13,1.89)%; E-time= 0.00156
Classifiers max: mCLESS= 97.13

Testing p = [2.39466667 0.606     ]
Triangle search: mCLESS: Acc.(mean,std) = (96.93,2.06)%; E-time= 0.00098
Classifiers max: mCLESS= 96.93

Testing p = [2.70555556 0.726     ]
Triangle search: mCLESS: Acc.(mean,std) = (96.61,2.33)%; E-time= 0.00091
Classifiers max: mCLESS= 96.61

Testing p = [3.01644444 0.846     ]
Triangle search: mCLESS: Acc.(mean,std) = (96.22,2.47)%; E-time= 0.00113
Classifiers max: mCLESS= 96.22

Testing p = [3.32733333 0.966     ]
Triangle search: mCLESS: Acc.(mean,std) = (95.76,2.47)%; E-time= 0.00116
Classifiers max: mCLESS= 95.76

Testing p = [3.63822222 1.086     ]
Triangle search: mCLESS: Acc.(mean,std) = (95.52,2.78)%; E-time= 0.00116
Classifiers max: mCLESS= 95.52

Testing p = [3.94911111 1.206     ]
Triangle search: mCLESS: Acc.(mean,std) = (95.20,2.67)%; E-time= 0.00098
Classifiers max: mCLESS= 95.20

Testing p = [4.26  1.326]
Triangle search: mCLESS: Acc.(mean,std) = (95.09,2.71)%; E-time= 0.00126
Classifiers max: mCLESS= 95.09

Testing p = [5.552 2.026]
Triangle search: mCLESS: Acc.(mean,std) = (94.65,2.77)%; E-time= 0.00125
Classifiers max: mCLESS= 94.65

Testing p = [5.40844444 1.94822222]
Triangle search: mCLESS: Acc.(mean,std) = (94.63,2.77)%; E-time= 0.00102
Classifiers max: mCLESS= 94.63

Testing p = [5.26488889 1.87044444]
Triangle search: mCLESS: Acc.(mean,std) = (94.65,2.71)%; E-time= 0.00119
Classifiers max: mCLESS= 94.65

Testing p = [5.12133333 1.79266667]
Triangle search: mCLESS: Acc.(mean,std) = (94.74,2.74)%; E-time= 0.00111
Classifiers max: mCLESS= 94.74

Testing p = [4.97777778 1.71488889]
Triangle search: mCLESS: Acc.(mean,std) = (94.83,2.68)%; E-time= 0.00133
Classifiers max: mCLESS= 94.83

Testing p = [4.83422222 1.63711111]
Triangle search: mCLESS: Acc.(mean,std) = (94.83,2.68)%; E-time= 0.00118
```

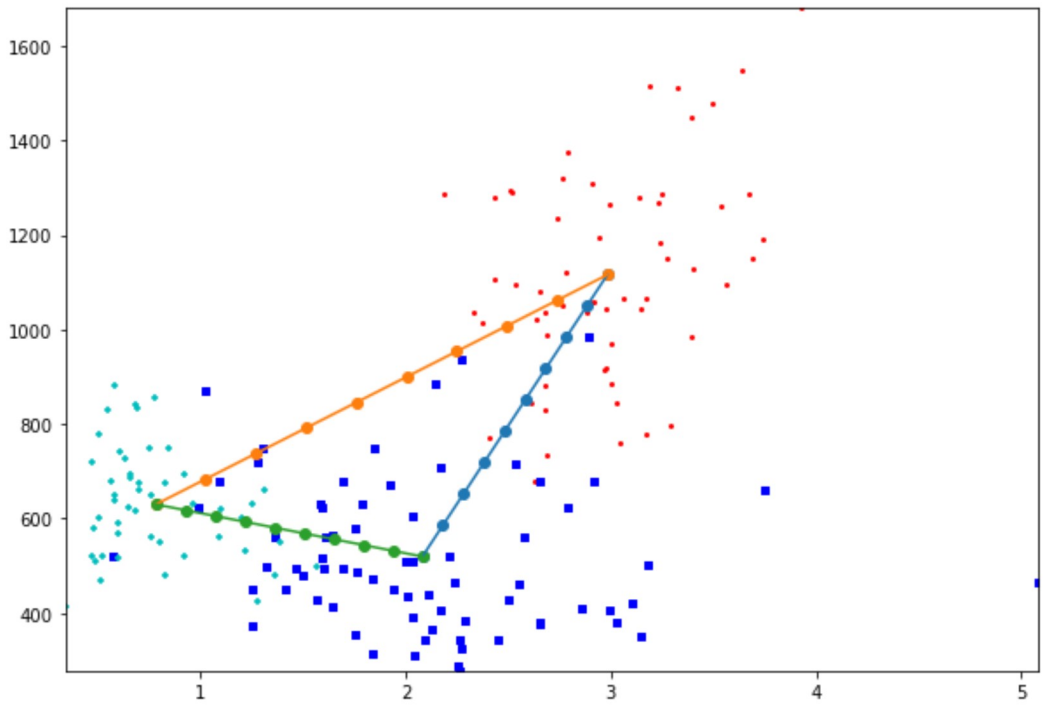
```
Classifiers max: mCLESS= 94.83

Testing p = [4.69066667 1.55933333]
Triangle search: mCLESS: Acc.(mean,std) = (94.83,2.66)%; E-time= 0.00101
Classifiers max: mCLESS= 94.83

Testing p = [4.54711111 1.48155556]
Triangle search: mCLESS: Acc.(mean,std) = (94.89,2.67)%; E-time= 0.00117
Classifiers max: mCLESS= 94.89

Testing p = [4.40355556 1.40377778]
Triangle search: mCLESS: Acc.(mean,std) = (95.04,2.75)%; E-time= 0.00118
Classifiers max: mCLESS= 95.04

Testing p = [4.26  1.326]
Triangle search: mCLESS: Acc.(mean,std) = (95.09,2.71)%; E-time= 0.00119
Classifiers max: mCLESS= 95.09

Testing p = [1.462 0.246]
Triangle search: mCLESS: Acc.(mean,std) = (97.33,1.87)%; E-time= 0.00113
Classifiers max: mCLESS= 97.33

Testing p = [1.91644444 0.44377778]
Triangle search: mCLESS: Acc.(mean,std) = (97.30,1.87)%; E-time= 0.00108
Classifiers max: mCLESS= 97.30

Testing p = [2.37088889 0.64155556]
Triangle search: mCLESS: Acc.(mean,std) = (96.96,2.04)%; E-time= 0.00090
Classifiers max: mCLESS= 96.96

Testing p = [2.82533333 0.83933333]
Triangle search: mCLESS: Acc.(mean,std) = (96.46,2.37)%; E-time= 0.00083
Classifiers max: mCLESS= 96.46

Testing p = [3.27977778 1.03711111]
Triangle search: mCLESS: Acc.(mean,std) = (95.76,2.49)%; E-time= 0.00096
Classifiers max: mCLESS= 95.76

Testing p = [3.73422222 1.23488889]
Triangle search: mCLESS: Acc.(mean,std) = (95.37,2.71)%; E-time= 0.00079
Classifiers max: mCLESS= 95.37

Testing p = [4.18866667 1.43266667]
Triangle search: mCLESS: Acc.(mean,std) = (95.09,2.71)%; E-time= 0.00095
Classifiers max: mCLESS= 95.09

Testing p = [4.64311111 1.63044444]
Triangle search: mCLESS: Acc.(mean,std) = (94.83,2.66)%; E-time= 0.00129
Classifiers max: mCLESS= 94.83

Testing p = [5.09755556 1.82822222]
Triangle search: mCLESS: Acc.(mean,std) = (94.74,2.74)%; E-time= 0.00094
Classifiers max: mCLESS= 94.74

Testing p = [5.552 2.026]
Triangle search: mCLESS: Acc.(mean,std) = (94.65,2.77)%; E-time= 0.00095
Classifiers max: mCLESS= 94.65
```


2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6

In [320]:
```
linesearch_traingle(X4,y4,[6,12])
```

```
Testing p = [   2.98237288 1115.71186441]
Triangle search: mCLESS: Acc.(mean,std) = (98.41,1.36)%; E-time= 0.00114
Classifiers max: mCLESS= 98.41

Testing p = [   2.88220312 1049.46688417]
Triangle search: mCLESS: Acc.(mean,std) = (98.35,1.36)%; E-time= 0.00115
Classifiers max: mCLESS= 98.35

Testing p = [  2.78203337 983.22190393]
Triangle search: mCLESS: Acc.(mean,std) = (98.35,1.33)%; E-time= 0.00112
Classifiers max: mCLESS= 98.35

Testing p = [  2.68186361 916.97692369]
Triangle search: mCLESS: Acc.(mean,std) = (98.37,1.37)%; E-time= 0.00138
Classifiers max: mCLESS= 98.37

Testing p = [  2.58169385 850.73194345]
Triangle search: mCLESS: Acc.(mean,std) = (98.44,1.35)%; E-time= 0.00131
Classifiers max: mCLESS= 98.44

Testing p = [  2.4815241  784.48696321]
Triangle search: mCLESS: Acc.(mean,std) = (98.48,1.35)%; E-time= 0.00118
Classifiers max: mCLESS= 98.48

Testing p = [  2.38135434 718.24198297]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.32)%; E-time= 0.00110
Classifiers max: mCLESS= 98.56

Testing p = [  2.28118458 651.99700273]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00094
Classifiers max: mCLESS= 98.56

Testing p = [  2.18101483 585.75202249]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00112
Classifiers max: mCLESS= 98.56

Testing p = [  2.08084507 519.50704225]
Triangle search: mCLESS: Acc.(mean,std) = (98.59,1.39)%; E-time= 0.00113
Classifiers max: mCLESS= 98.59

Testing p = [  0.78145833 629.89583333]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00100
Classifiers max: mCLESS= 98.56

Testing p = [  0.92583464 617.6304121 ]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00102
Classifiers max: mCLESS= 98.56

Testing p = [  1.07021094 605.36499087]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00127
Classifiers max: mCLESS= 98.56

Testing p = [  1.21458725 593.09956964]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00133
Classifiers max: mCLESS= 98.56

Testing p = [  1.35896355 580.83414841]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00104
Classifiers max: mCLESS= 98.56

Testing p = [  1.50333985 568.56872718]
Triangle search: mCLESS: Acc.(mean,std) = (98.57,1.41)%; E-time= 0.00113
```

```
Classifiers max: mCLESS= 98.57

Testing p = [  1.64771616 556.30330595]
Triangle search: mCLESS: Acc.(mean,std) = (98.57,1.41)%; E-time= 0.00103
Classifiers max: mCLESS= 98.57

Testing p = [  1.79209246 544.03788472]
Triangle search: mCLESS: Acc.(mean,std) = (98.57,1.41)%; E-time= 0.00100
Classifiers max: mCLESS= 98.57

Testing p = [  1.93646877 531.77246348]
Triangle search: mCLESS: Acc.(mean,std) = (98.57,1.41)%; E-time= 0.00100
Classifiers max: mCLESS= 98.57

Testing p = [  2.08084507 519.50704225]
Triangle search: mCLESS: Acc.(mean,std) = (98.59,1.39)%; E-time= 0.00090
Classifiers max: mCLESS= 98.59

Testing p = [   2.98237288 1115.71186441]
Triangle search: mCLESS: Acc.(mean,std) = (98.41,1.36)%; E-time= 0.00116
Classifiers max: mCLESS= 98.41

Testing p = [   2.73782682 1061.7323054 ]
Triangle search: mCLESS: Acc.(mean,std) = (98.35,1.36)%; E-time= 0.00105
Classifiers max: mCLESS= 98.35

Testing p = [   2.49328076 1007.75274639]
Triangle search: mCLESS: Acc.(mean,std) = (98.31,1.34)%; E-time= 0.00112
Classifiers max: mCLESS= 98.31

Testing p = [  2.2487347  953.77318738]
Triangle search: mCLESS: Acc.(mean,std) = (98.37,1.34)%; E-time= 0.00111
Classifiers max: mCLESS= 98.37

Testing p = [  2.00418864 899.79362837]
Triangle search: mCLESS: Acc.(mean,std) = (98.37,1.37)%; E-time= 0.00145
Classifiers max: mCLESS= 98.37

Testing p = [  1.75964258 845.81406937]
Triangle search: mCLESS: Acc.(mean,std) = (98.46,1.36)%; E-time= 0.00106
Classifiers max: mCLESS= 98.46

Testing p = [  1.51509652 791.83451036]
Triangle search: mCLESS: Acc.(mean,std) = (98.50,1.33)%; E-time= 0.00097
Classifiers max: mCLESS= 98.50

Testing p = [  1.27055046 737.85495135]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.32)%; E-time= 0.00098
Classifiers max: mCLESS= 98.56

Testing p = [  1.02600439 683.87539234]
Triangle search: mCLESS: Acc.(mean,std) = (98.57,1.38)%; E-time= 0.00090
Classifiers max: mCLESS= 98.57

Testing p = [  0.78145833 629.89583333]
Triangle search: mCLESS: Acc.(mean,std) = (98.56,1.40)%; E-time= 0.00112
Classifiers max: mCLESS= 98.56
```



The triangle search results are not different from the results of linesearch results which means that both strategies give same results

Project Summary

Many machine learning algorithms are not interpretable. Hence the objective of this project is to create an interpretable algorithm which gives accuracy as good as the accuracy of other complex algorithms.

To achieve this, the algorithm mCLESS is developed using the least squares method.

Experiments are done on four data sets

1. Synthetic Data 1
2. Synthetic Data 2
3. Iris data
4. Wine data

Performances of following classifiers are then compared to the performance of mCLESS.

1. Logistic Regression
2. K-neighboursClassifiers
3. SVM - rbf
4. RandomForestClassifier

The performance meteric used for the experiments is percentage accuracy.

All Algorithms are run 100 times, creating random 70:30 split of the data for training and testing in each run. Then the average accuracy is reported for each algorithm. From these experiments , we found that that mCLESS gives accuracy as good as other classifiers at a much less computational cost.

Feature expansion -

Feature expansion is done by adding a dimension equal to distance of a data sample from a particular point p in the dataset.

The startegies implemented to choose p are as follows:

1. Visually choose the point p for synthetic data1 and synthetic data2.
2. Chose the center of the dataset as point p.
3. Linesearch for point p along the least sqaures quadratic polynomial.
4. Linesearch along the triangle formed by the centers of the cluster.

The center of the dataset can be chosen as p because when we add the dimension, all points would go away from center, thus increasing the spaces between the clusters. It can make classses more separable.

For Synthetic data 1, the exepriments were done by chosing p as the center of the data, by linesearch, and by linesearch of the triangle. It was found that accuracy for this dataset remains almost the same, irrespective of the choice of point p.

For Synthetic data 2, the centers of the class clusters are almost colinear. If the point p is chosen as the center of the class 1 cluster then the distances of the points in other class clusters from p are comparatively larger than the distances of points in class 1 cluster. Thus when we add $||\mathbf{x} - \mathbf{p}||$ as the additional dimension with p chosen to be the class 1 center, the points in other classes move away with respect to the class 1 cluster in the higher dimensional space. This breaks the colinearity of the clusters and makes them more separable.

The experiments validate this point. In all the four strategies mentioned above, the accuracy increased from around 71% to around 95% when the point p is the center of the cluster of class 1 in the Synthetic dataset 2. Coincidently, the overall center of the entire data set is also near the center of the class 1 cluster. Hence, its accuracy also showed similar improvement.

For Iris data, by using p as the overall center of the entire data set, the accuracy improved from 82.96% to 95.41%. For doing linesearch for Iris data, we chose the feature at the index 2 and 3 because the classes looked more separable in the pairplot of features at 2 and 3. The accuracy can be futher improved to 97.43% by the point $p = [1.0, 0.04]$ which was obtained using the linesearch.

For Wine dataset, the accuracy before adding dimension is 98.61%. The accuracy of wine data did not improve much by trying the above mentioned strategies to find p.

Conclusion -

I have successfully implemented and tested the algorithm mCLESS for multi-class classification, which is also interpretable. The parametres of the classifier are visualized and it showcases why the classifier is interpretable. The weight matrix calculated by the least squares method represents the family of lines for each class such that if the distance of a point from the line $L_j = 0$ is maximum, then the point belongs to class j.

For the datasets which cannot be separated using one versus all techniques, it appears that the drop is accuracy is because the decision boundaries (hyperplanes) become almost parallel and collinear to one another. This project shows that a significant improvement in the prediction accuracy can be made by adding an additional dimension to the data by the equation: $||\mathbf{x} - \mathbf{p}||$, where the point $\mathbf{p}$ is chosen such that it breaks collinearity and facilitates clear separation of the classes. Making the data more separable increases the accuracy of any classification algorithm. It was found that $\mathbf{p}$ can be chosen to be in the region near the center of the overall data set. It improves the accuracy for colinear clusters, and also does not significantly affect the accuracy in cases where the classes are not strongly non-colinear.