Lab 3 part 2: Practical application

Tanvi Mukesh PARMAL

Arnaud MATHEY

Imports:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import sklearn
import sklearn.tree as tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from random import choices
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier , GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

2.1 Regression Trees

In [2]:
```
auto=pd.read_csv('auto.txt', sep=";")
```

In [3]:
```
auto.mpg.describe()
```

Out[3]:
```
count    300.000000
mean      23.699333
std        7.963814
min        9.000000
25%       17.000000
50%       23.600000
75%       29.925000
max       46.600000
Name: mpg, dtype: float64
```

As you can see there is 300 values, mean of 23.59 deviation of 7.96. I don't really see what to add next it is kind of self explanatory with value from [9,46]

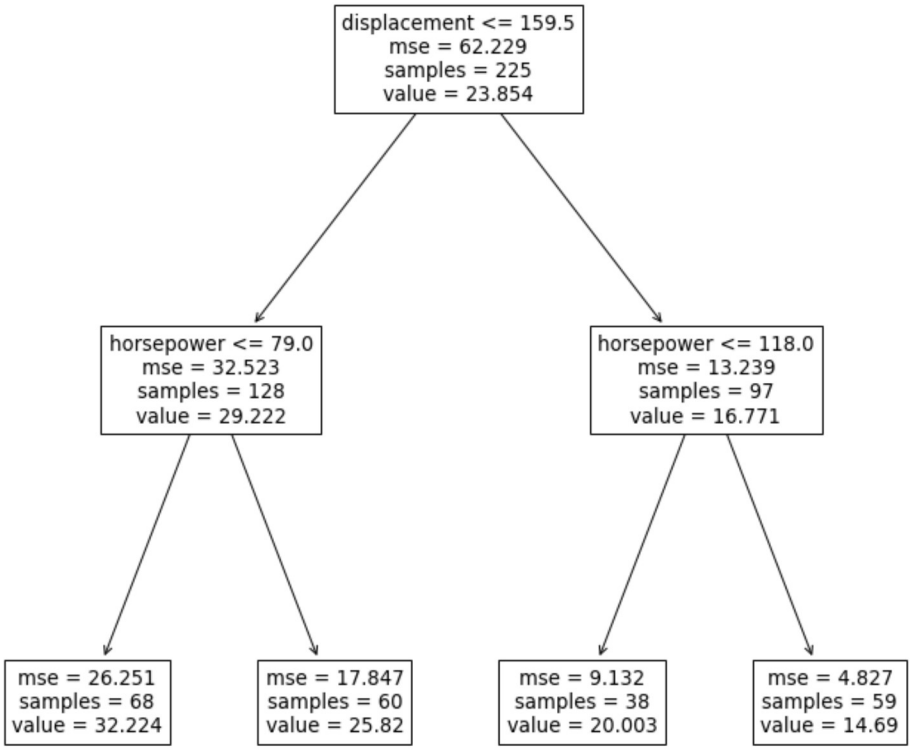
In [4]:
```
x_train = pd.read_csv('x_train.csv',index_col=0)
x_test = pd.read_csv('x_test.csv',index_col=0)
y_train = pd.read_csv('y_train.csv',index_col=0)
y_test = pd.read_csv('y_test.csv',index_col=0)
y_train_reg = pd.read_csv('y_train_reg.csv',index_col=0)
y_test_reg = pd.read_csv('y_test_reg.csv',index_col=0)
```

In [5]:
```
regr_tree_auto = tree.DecisionTreeRegressor ( max_depth =2  )
regr_tree_auto.fit(x_train , y_train_reg )
```

Out[5]:
```
DecisionTreeRegressor(max_depth=2)
```

In [6]:
```
plt.figure ( figsize = ( 12 , 12 ) )
tree.plot_tree(regr_tree_auto , feature_names=x_train.columns )
```

Out[6]:
```
[Text(334.8, 543.6, 'displacement <= 159.5\nmse = 62.229\nsamples = 225\nvalue = 23.854'),
 Text(167.4, 326.16, 'horsepower <= 79.0\nmse = 32.523\nsamples = 128\nvalue = 29.222'),
 Text(83.7, 108.72000000000003, 'mse = 26.251\nsamples = 68\nvalue = 32.224'),
 Text(251.10000000000002, 108.72000000000003, 'mse = 17.847\nsamples = 60\nvalue = 25.82'),
 Text(502.20000000000005, 326.16, 'horsepower <= 118.0\nmse = 13.239\nsamples = 97\nvalue = 16.771'),
 Text(418.5, 108.72000000000003, 'mse = 9.132\nsamples = 38\nvalue = 20.003'),
 Text(585.9, 108.72000000000003, 'mse = 4.827\nsamples = 59\nvalue = 14.69')]
```



In [7]:

```
print(mean_squared_error(y_test_reg, regr_tree_auto.predict(x_test)))
print(mean_squared_error(y_train_reg, regr_tree_auto.predict(x_train)))

df2 = {'cylinders':6.0,'displacement': 120.0, 'horsepower': 100.0,'weight':3425.0}
df3 = x_train.append(df2, ignore_index = True)


print(regr_tree_auto.predict(df3)[-1])
```

```
21.4155894241315
15.500719947059405
25.819999999999997
```

a)The way I understand this coefficient is that value represent the number of information given by a node. the closer to mean value of a coefficient the better it is. so the top node here is really close to 23.7 which is really good. sample is the number of value that goes to a node not really usefull for the first one since all the value will be submitted to this node. mse is the mean square error for training data on a the parametter of the node

b) According to the tree there is 4 terminal node

c) it is 15.500719947059405

d) the gaz value will be around 25.82 according to tree, using the predict function we confirm this result to be good

2) mean square error of the test set is 21.41. If we compare this value to the mean of mgp whitch is 23.7 it seems like there is overfitting however the train value is 15.5 which is also really bad so the model at this point might just be bad. I have also tested with 6 node and I obtain of train error of only 6 but a test error of 25 which defintly indicate overfitting.

The only way to deal with overfitting on this algorithm is by selecting the right number of nodes and doing tree pruning which basicly mean dropping some branches to try to improve the result. In this case If it was asked I would robably choose 4 nodes which give us a train error around 10 and try to to prun the tree but to be honest considering how poorly the algorithm is working I don't think I will gave us a really good model maybe a more powerfull type of algorithm like random forest will just work better.

2.2 Classication Trees

In [8]:

```
class_tree_auto=tree.DecisionTreeClassifier(max_depth=3)
class_tree_auto.fit(x_train,y_train)
```

Out[8]:
```
DecisionTreeClassifier(max_depth=3)
```

In [9]:
```
x_train
```

Out[9]:

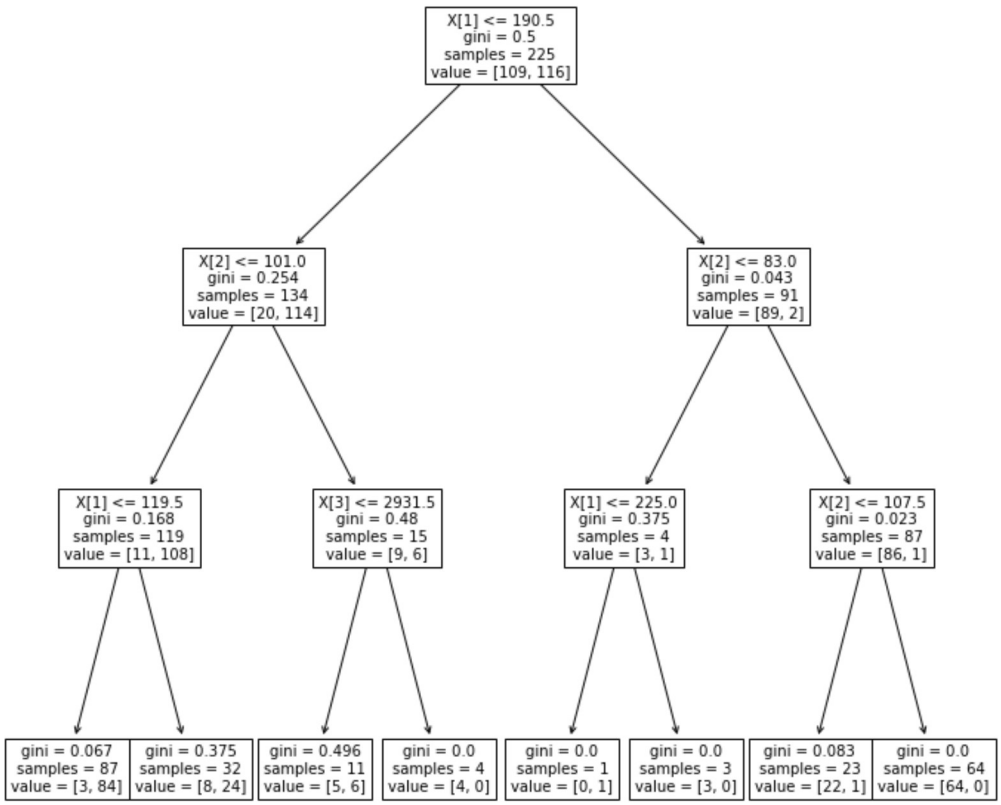
	cylinders	displacement	horsepower	weight
259	4	156.0	92	2620
37	4	88.0	76	2065
97	4	79.0	67	1950
191	4	91.0	60	1800
135	4	120.0	88	2957
...	...	...	...	...
251	4	91.0	67	1850
192	8	260.0	110	3365
117	6	250.0	72	3432
47	8	400.0	175	4385
172	6	231.0	105	3425

225 rows × 4 columns

In [10]:
```
print(y_train_reg)
```

```
      mpg
259  25.8
37   30.0
97   31.0
191  36.1
135  23.0
..    ...
251  44.6
192  19.9
117  15.0
47   14.0
172  20.5

[225 rows x 1 columns]
```

In [11]:
```
plt.figure(figsize=(12,12))
tree.plot_tree(class_tree_auto, fontsize=10)
plt.show()
```



In [12]:
```
df2 = {'cylinders':6.0,'displacement': 120.0, 'horsepower': 100.0,'weight':1425.0}
df3 = x_train.append(df2, ignore_index = True)
print(class_tree_auto.predict(df3)[-1])
print(confusion_matrix(y_test, class_tree_auto.predict(x_test)))
```

```
1
[[34  7]
 [ 2 32]]
```

1) the gini value is 0.5 an indication that the node uses values of multiples class. because it is the first one it does have all the branches and parameters

2) The predict value of a diplacement of 120 and horsepower of 100 is 1 so high consuption for the algorithm.

Using the tree and looking at the node we arrived at values from [8,24] so if the predicted value is 23 or 24 it would be considered as high so a 1. but less than 23 should be a 0. We are going to trust the predict function here and say that it is a 1

In [13]:
```
prun_tree_auto= DecisionTreeClassifier(random_state=1)
path = prun_tree_auto.cost_complexity_pruning_path(x_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [14]:
```
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
```

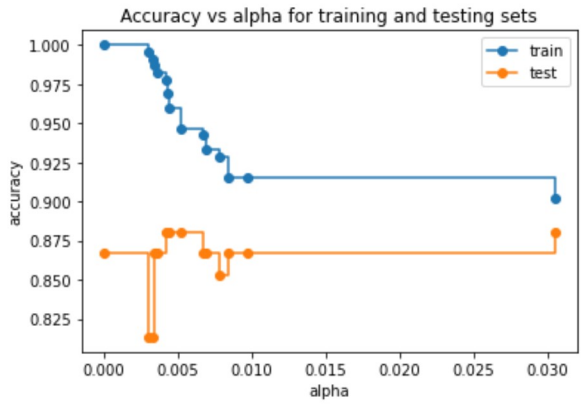
Out[14]:
```
Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')
```

In [15]:
```
clfs = []
best_alpha = 0
best_score = 0
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(x_train, y_train)
    clfs.append(clf)
    score = clf.score(x_test, y_test)
    if(score > best_score):
        best_score=score
        best_alpha=ccp_alpha
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))

print("the best test score is: ")
print(best_score)
print("the best alpha is: ")
print(best_alpha)
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.3308852109011313
the best test score is:
0.88
the best alpha is:
0.004222222222222221
```

In [16]:
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]
```

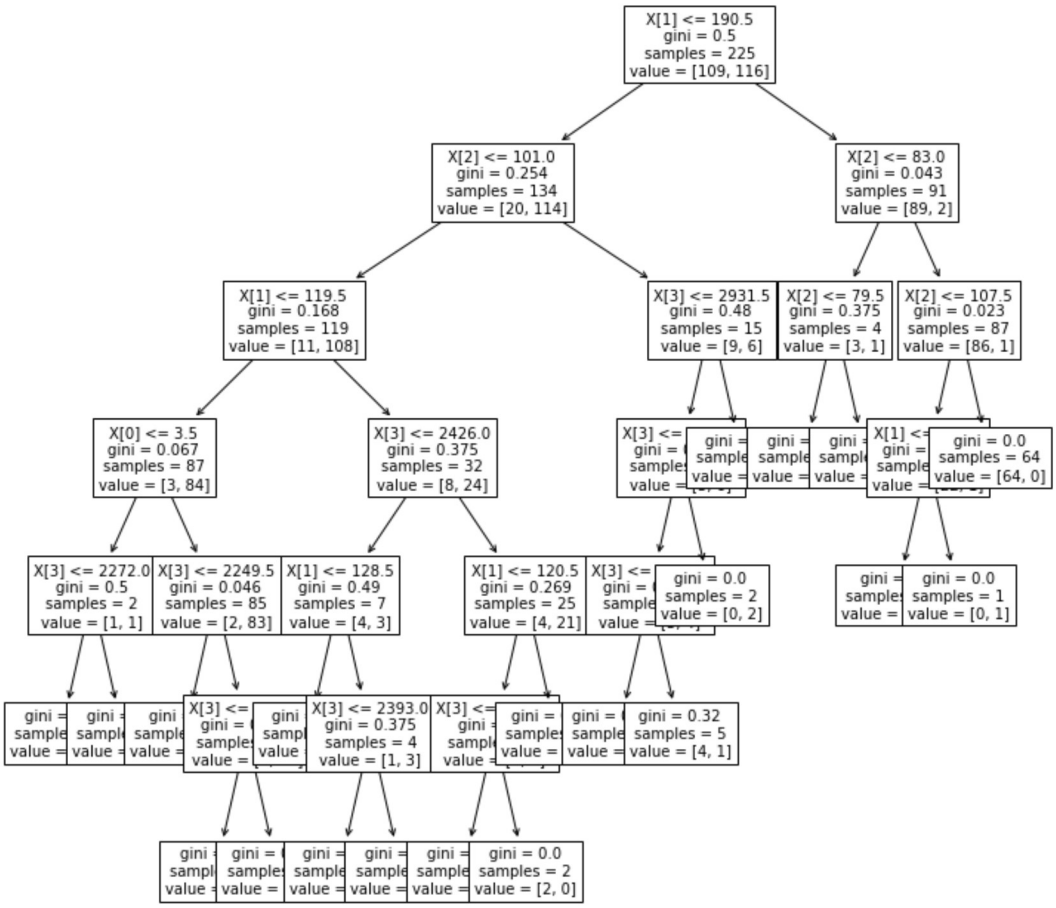
In [17]:
```
train_scores = [clf.score(x_train, y_train) for clf in clfs]
test_scores = [clf.score(x_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



In [18]:
```
clf = DecisionTreeClassifier(random_state=0, ccp_alpha=best_alpha)
clf.fit(x_train, y_train)
print(confusion_matrix( y_test,clf.predict(x_test)))
print(accuracy_score(clf.predict(x_train), y_train))
print(clf.get_n_leaves())
print(clf.get_depth())
print(print(classification_report(clf.predict(x_test), y_test)))
plt.figure(figsize=(12,12))
tree.plot_tree(clf, fontsize=10)
plt.show()
```

```
[[35  6]
 [ 3 31]]
```

```
0.9777777777777777
20
6
              precision    recall  f1-score   support

           0       0.85      0.92      0.89        38
           1       0.91      0.84      0.87        37

    accuracy                           0.88        75
   macro avg       0.88      0.88      0.88        75
weighted avg       0.88      0.88      0.88        75

None
```



The number of leaf is 20, the depth is 6, the confusion matrix tells us that there is 6 false positive for 35 positive and 3 false negative for 31 good negative quitte a good result. train accuracy is 37% and test accuracy is 88%. The report told us 85% precision on positive and 91% on negative.

2.3 Ensemble methods : Bagging, Random forest and Boosting

In [19]:
```
bagging_auto = RandomForestClassifier(max_features = 4 , random_state = 1)
bagging_auto.fit( x_train , y_train.mpg01)
```

Out[19]:
```
RandomForestClassifier(max_features=4, random_state=1)
```

Bagging is around making different tree with bootstrapping on their train dataset whitch mean that each tree is going to have is own trainning data. the idea beetween random forest is to perform the mean of the preediction of each tree

In [20]:
```
print(accuracy_score(bagging_auto.predict(x_test), y_test))
print(accuracy_score(bagging_auto.predict(x_train), y_train))
print(confusion_matrix( y_test,bagging_auto.predict(x_test)))
```

```
0.8533333333333334
1.0
[[35  6]
 [ 5 29]]
```

Train accuracy is 1.0 which is normal it is train data and test accuracy is 85% which seems pretty good The numbers of n_estimators by befault is 100 so for m=sqrt(p) we need 10 predictors

In [21]:
```
bagging_auto = RandomForestClassifier(max_features = 4 , random_state = 1,n_estimators = 10)
```

```
bagging_auto.fit( x_train , y_train.mpg01)
```

Out[21]:
```
RandomForestClassifier(max_features=4, n_estimators=10, random_state=1)
```

In [22]:
```
print(accuracy_score(bagging_auto.predict(x_test), y_test))
print(accuracy_score(bagging_auto.predict(x_train), y_train))
print(confusion_matrix( y_test,bagging_auto.predict(x_test)))
```

```
0.88
0.9955555555555555
[[35  6]
 [ 3 31]]
```

the result have increased accuracy over the previous one maybe a consequence of overfitting with too much tree involved?

In [23]:
```
grad_auto = GradientBoostingClassifier(learning_rate = 1 , random_state = 0,max_depth = 1)
grad_auto.fit( x_train , y_train.mpg01)
```

Out[23]:
```
GradientBoostingClassifier(learning_rate=1, max_depth=1, random_state=0)
```

In [24]:
```
print(accuracy_score(grad_auto.predict(x_test), y_test))
print(accuracy_score(grad_auto.predict(x_train), y_train))
print(confusion_matrix( y_test,grad_auto.predict(x_test)))
```

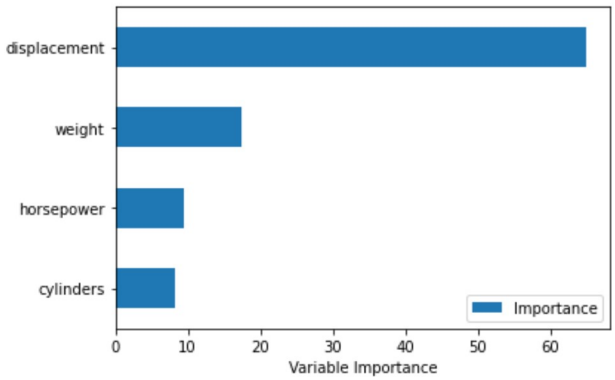
```
0.88
0.92
[[35  6]
 [ 3 31]]
```

We did obtain the same acuracy as the random forest in term of test but not in term of train interesting result. with higher depth we don't optain better result so the algorithm seems similar to the forest in term of accuracy

In [25]:
```
print(bagging_auto.feature_importances_ )
Importance = pd.DataFrame({'Importance' : bagging_auto.feature_importances_*100},index = x_train.columns)
Importance.sort_values(by = 'Importance' , axis = 0 ,ascending = True).plot (kind = 'barh' )
plt.xlabel( 'Variable Importance' )
```

```
[0.08182757 0.65006153 0.09431786 0.17379303]
```

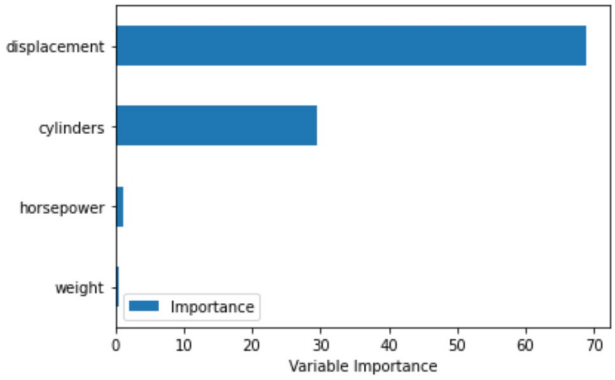
Out[25]:
```
Text(0.5, 0, 'Variable Importance')
```



In [26]:
```
print(grad_auto.feature_importances_ )
Importance = pd.DataFrame({'Importance' : grad_auto.feature_importances_*100},index = x_train.columns)
Importance.sort_values(by = 'Importance' , axis = 0 ,ascending = True).plot (kind = 'barh' )
plt.xlabel( 'Variable Importance' )
```

```
[0.29408813 0.68967345 0.0120484  0.00419002]
```

Out[26]:
```
Text(0.5, 0, 'Variable Importance')
```



This is really interesting, because both algorithm were giving me same accuracy I was thinking that they would be quite similar, however it seems that the way they reconised the important features are quite different. Every algorithm that we have tested so far shows us displacement as being the main factor however they all tells us a different story about the second criteria. The result of boosting is due to boosting tring to get strong factors so the role of the weakest is minimised while random forest is just combining random tree with bootstrap.

2.4 Support Vector Machine (SVM)

In [27]:
```
svm_auto = SVC(C=1,kernel = 'linear', random_state = 0)
svm_auto.fit(x_train, np.ravel(y_train))
```

Out[27]:
```
SVC(C=1, kernel='linear', random_state=0)
```

Calculate the confusion matrix and the accuracy for the test set and comment on the results

In [28]:
```
y_pred = svm_auto.predict(x_test)
```

In [29]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(accuracy_score(y_test, y_pred))
print(len(svm_auto.support_))
```

```
[[34  7]
 [ 3 31]]
0.8666666666666667
73
```

The positive to false positive ratio is quite good but the negative to false negative is bad.

There is 73 support vectors

In [30]:
```
svm_auto.support_
```

Out[30]:
```
array([  4,   5,   6,  11,  16,  19,  20,  21,  24,  40,  60,  73,  75,
        86,  96,  97,  98, 102, 103, 113, 115, 123, 130, 138, 149, 163,
       171, 175, 183, 187, 191, 200, 214, 217, 222,   0,   9,  13,  15,
        17,  25,  26,  30,  39,  46,  47,  69,  71,  78,  95, 101, 110,
       112, 121, 122, 124, 133, 135, 139, 140, 141, 144, 148, 154, 156,
       159, 172, 176, 182, 194, 197, 201, 215])
```

In [31]:
```
tuned_parameters=[{'C':[0.001,0.01,0.1,1,5,10]}]
svm_auto_CV=GridSearchCV(SVC(kernel='linear'), tuned_parameters, cv=10, scoring='accuracy')
svm_auto_CV.fit(x_train,np.ravel(y_train))
```

Out[31]:
```
GridSearchCV(cv=10, estimator=SVC(kernel='linear'),
             param_grid=[{'C': [0.001, 0.01, 0.1, 1, 5, 10]}],
             scoring='accuracy')
```

In [32]:
```
svm_auto_CV.best_params_
```

Out[32]:
```
{'C': 5}
```

The best value for C is 5 in term of acuracy.

In [33]:
```
#Fitting SVM to the Training set

svm_auto= SVC(kernel ='linear', C=5)
svm_auto.fit(x_train,np.ravel(y_train))
print(len(svm_auto.support_))
```

```
47
```

There is 47 support vectors

In [34]:
```
y_pred = svm_auto_CV.predict(x_test)
```

In [35]:
```
accuracy_score(y_test, y_pred)
```

Out[35]:
```
0.8666666666666667
```

In [36]:
```
confusion_matrix(y_test, y_pred)
```

Out[36]:
```
array([[34,  7],
       [ 3, 31]], dtype=int64)
```

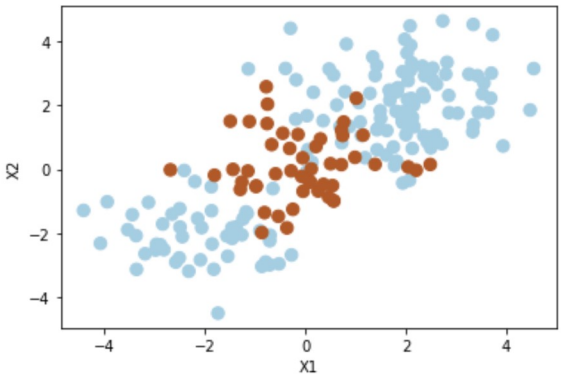
In [37]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87        41
           1       0.82      0.91      0.86        34

    accuracy                           0.87        75
   macro avg       0.87      0.87      0.87        75
weighted avg       0.87      0.87      0.87        75
```

The accuricy is similar than with the previous model. the confusion matrix is identical and we have the same property on good positive detection and bad negative ones

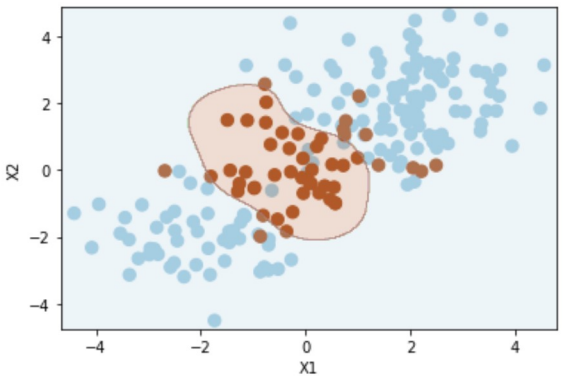
In [38]:
```
np.random.seed(8)
X=np.random.randn(200,2)
X[:100] = X[:100] + 2
X[101:150] = X[101:150]-2
Y=np.concatenate([np.repeat( -1,150),np.repeat(1, 50)])
plt.scatter(X[:,0], X[:,1], s=70, c=Y, cmap=plt.cm.Paired)
plt.xlabel('X1')
plt.ylabel('X2')
```

Out[38]:
```
Text(0, 0.5, 'X2')
```



In [39]:
```
svm_radial=SVC(C=1.0, kernel='rbf', gamma=1)
svm_radial.fit(X,Y)
plt.scatter(X[:,0], X[:,1], s=70, c=Y, cmap=plt.cm.Paired )
plt.xlabel('X1')
plt.ylabel('X2')
h=0.02
pad=0.25
x_min, x_max = X [:, 0].min() - pad, X[:,0].max()+pad
y_min, y_max = X [:, 1].min() - pad, X[:,1].max()+pad
xx,yy=np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,y_max, h))
Z= svm_radial.predict(np.c_[ xx.ravel(), yy.ravel()])
Z=Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.2)
```

Out[39]:
```
<matplotlib.contour.QuadContourSet at 0x1a4a16f9130>
```



4) Fit an SVM with a radial kernel

In [40]:
```
tuned_parameters = [{ 'C': [0.01, 0.1, 1, 10, 100], 'gamma': [0.5, 1,2,3,4]}]
```

```
svm_radial_auto_CV = GridSearchCV(SVC(kernel='rbf'), tuned_parameters, cv=10, scoring='accuracy')
svm_radial_auto_CV.fit(x_train,np.ravel(y_train))
```

Out[40]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid=[{'C': [0.01, 0.1, 1, 10, 100],
                          'gamma': [0.5, 1, 2, 3, 4]}],
             scoring='accuracy')
```

In [41]:
```
svm_radial_auto_CV.best_params_
```

Out[41]:
```
{'C': 1, 'gamma': 0.5}
```

Fit the SVM classifier to the train set with the chosen value of C and γ.

In [42]:
```
svm_radial_auto= SVC(kernel ='rbf', C=1, gamma=0.5)
svm_radial_auto.fit(x_train,np.ravel(y_train))
```

Out[42]:
```
SVC(C=1, gamma=0.5)
```

Calculate performance metrics (accuracy, confusion matrix, etc)

In [43]:
```
y_pred = svm_radial_auto.predict(x_test)
```

In [44]:
```
accuracy_score(y_test, y_pred)
```

Out[44]:
```
0.4533333333333333
```

In [45]:
```
confusion_matrix(y_test, y_pred)
```

Out[45]:
```
array([[ 0, 41],
       [ 0, 34]], dtype=int64)
```

In [46]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        41
           1       0.45      1.00      0.62        34

    accuracy                           0.45        75
   macro avg       0.23      0.50      0.31        75
weighted avg       0.21      0.45      0.28        75

C:\Users\Akiesa\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `
zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

5)Fit an SVM with a polynomial kernel

In [47]:
```
tuned_parameters = [{ 'C': [0.1, 1, 10], 'degree': [3,4,5]}]
svm_polynomial_auto_CV = GridSearchCV(SVC(kernel='poly'), tuned_parameters, cv=10, scoring='accuracy')
svm_polynomial_auto_CV.fit(x_train,np.ravel(y_train))
```

Out[47]:
```
GridSearchCV(cv=10, estimator=SVC(kernel='poly'),
             param_grid=[{'C': [0.1, 1, 10], 'degree': [3, 4, 5]}],
             scoring='accuracy')
```

In [48]:
```
svm_polynomial_auto_CV.best_params_
```

Out[48]:
```
{'C': 1, 'degree': 4}
```

In [49]:
```
svm_polynomial_auto= SVC(kernel ='poly', C=1, degree=4)
svm_polynomial_auto.fit(x_train,np.ravel(y_train))
```

Out[49]:
```
SVC(C=1, degree=4, kernel='poly')
```

Calculate performance metrics(accuracy, confusion matrix, etc)

In [50]:
```
y_pred = svm_polynomial_auto.predict(x_test)
```

In [51]:
```
accuracy_score(y_test, y_pred)
```

Out[51]:
```
0.8533333333333334
```

In [52]:
```
confusion_matrix(y_test, y_pred)
```

Out[52]:
```
array([[32,  9],
       [ 2, 32]], dtype=int64)
```

In [53]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.78      0.85        41
           1       0.78      0.94      0.85        34

    accuracy                           0.85        75
   macro avg       0.86      0.86      0.85        75
weighted avg       0.87      0.85      0.85        75
```

The polimial kernal has less false positive error but higher false negative acuracy based on the metrics.

2.5 Conclusion

The different conclusion matrix that we got during the exercise trying to compare everithing tells us all kind of the same story. pretty good accuracy one positive / false positive value and less good performance on negative value. the polimial and the random forest were the worst and we didn't obtain good data from SVM with radial Kernel. Every method has her way to deal with the importance of the parameters the trees tend to classify the parameters beetween each over for exemple. while the SVM method varies a bit bepending on the type of core

In [ ]:
```
```