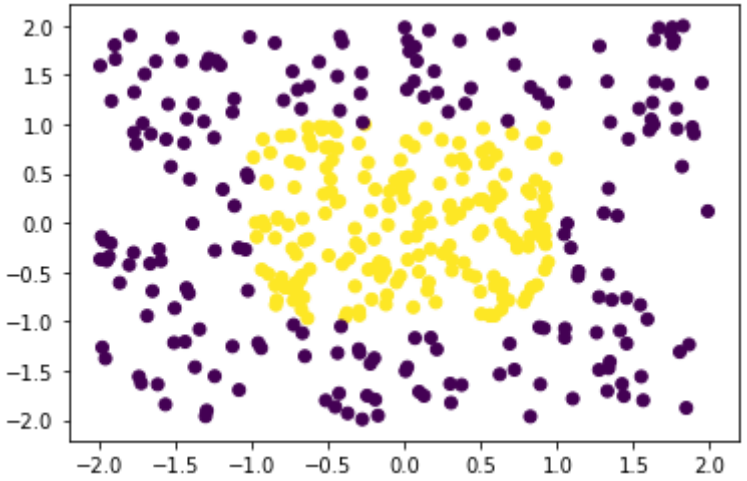
LAB 2 - Assignment 1.1

In [44]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plot
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import KFold
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from numpy.random import random
from sklearn import tree
```

DATASET 1

In [21]:
```
df = pd.read_csv('exp1a.csv')
x = df.iloc[:,0];
y = df.iloc[:,1];
plot.scatter(x,y, c= df["class"]);
```



In [22]:
```
df = pd.read_csv('exp1a.csv')
X = df.iloc[:,0:2];
y = df['class'];

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state = 40)

print('\033[1m' + 'Normal')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i)
    classifier.fit(X_train,y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred,)
    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)

    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)

#model with distance as weights

print('\033[1m' + 'Distance as weights')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i,weights = 'distance')
    classifier.fit(X_train,y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred,)
    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)
    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)
```

```
print('\033[1m')


#Decision Trees
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('decision trees:',scores)

#logistic regression
classifier = LogisticRegression(random_state=0, solver='lbfgs')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('Logisitic regression:',scores)
```

Normal

```
k =  1
accuracy: 0.9833
Cross Validation scores : 0.9775
confusion matrix: [[55  0]
 [ 2 63]]
k =  5
accuracy: 0.9667
Cross Validation scores : 0.9525
confusion matrix: [[51  4]
 [ 0 65]]
k =  11
accuracy: 0.95
Cross Validation scores : 0.9425
confusion matrix: [[49  6]
 [ 0 65]]
k =  21
accuracy: 0.9583
Cross Validation scores : 0.9275
confusion matrix: [[50  5]
 [ 0 65]]
```

Distance as weights

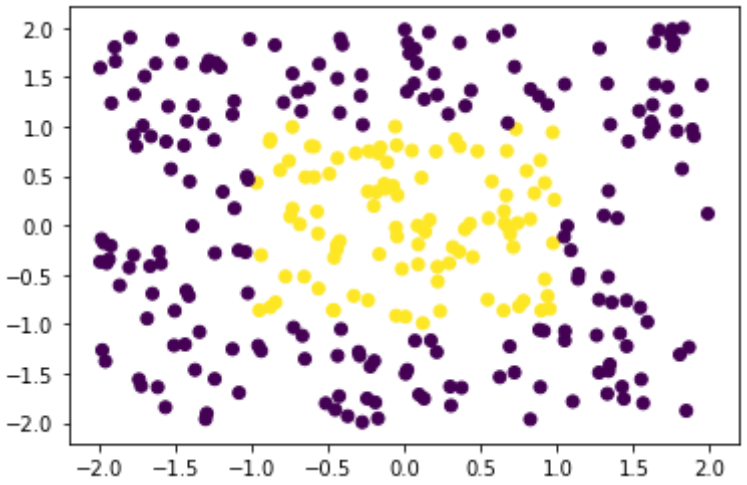
```
k =  1
accuracy: 0.9833
Cross Validation scores : 0.9775
confusion matrix: [[55  0]
 [ 2 63]]
k =  5
accuracy: 0.975
Cross Validation scores : 0.9675
confusion matrix: [[52  3]
 [ 0 65]]
k =  11
accuracy: 0.9667
Cross Validation scores : 0.9625
confusion matrix: [[51  4]
 [ 0 65]]
k =  21
accuracy: 0.9667
```

```
Cross Validation scores : 0.9475
confusion matrix: [[51  4]
 [ 0 65]]
```

decision trees: 99.16666666666667
Logisitic regression: 42.5

DATASET 2

In [23]:
```
df = pd.read_csv('exp1b.csv')
x = df.iloc[:,0];
y = df.iloc[:,1];
plot.scatter(x,y, c= df["class"]);
```



In [24]:
```
df = pd.read_csv('exp1b.csv')
X = df.iloc[:,0:2];
y = df['class'];

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state = 40)

print('\033[1m' + 'Normal')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i)

    classifier.fit(X_train,y_train)

    y_pred = classifier.predict(X_test)


    accuracy = accuracy_score(y_test,y_pred,)


    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)

    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)

#model with distance as weights

print('\033[1m' + 'Distance as weights')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i,weights = 'distance')

    classifier.fit(X_train,y_train)

    y_pred = classifier.predict(X_test)
```

```
    accuracy = accuracy_score(y_test,y_pred,)


    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)

    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)


print('\033[1m')


#Decision Trees
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('decision trees:',scores)

#Logistic regression
classifier = LogisticRegression(random_state=0, solver='lbfgs')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('Logisitic regression:',scores)
```

Normal

```
k =  1
accuracy: 0.9667
Cross Validation scores : 0.96
confusion matrix: [[59  1]
 [ 2 28]]
k =  5
accuracy: 0.9444
Cross Validation scores : 0.9667
confusion matrix: [[57  3]
 [ 2 28]]
k =  11
accuracy: 0.9667
Cross Validation scores : 0.9467
confusion matrix: [[58  2]
 [ 1 29]]
k =  21
accuracy: 0.9333
Cross Validation scores : 0.94
confusion matrix: [[56  4]
 [ 2 28]]
```

Distance as weights

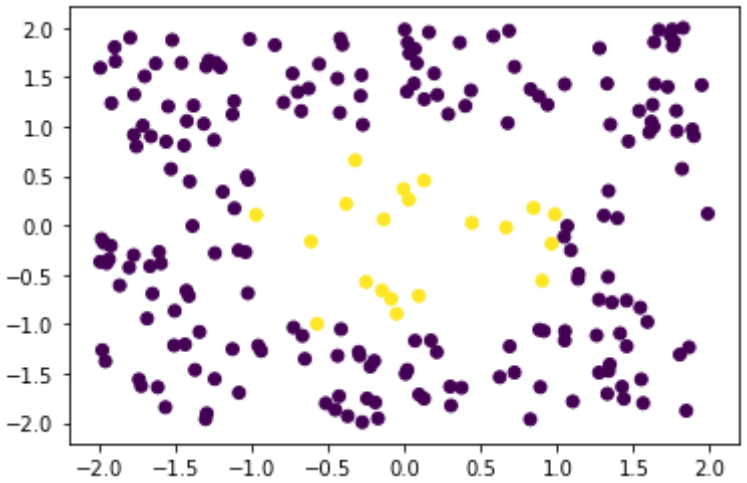
```
k =  1
accuracy: 0.9667
Cross Validation scores : 0.96
confusion matrix: [[59  1]
 [ 2 28]]
k =  5
accuracy: 0.9333
Cross Validation scores : 0.97
confusion matrix: [[58  2]
 [ 4 26]]
k =  11
accuracy: 0.9556
Cross Validation scores : 0.96
confusion matrix: [[58  2]
 [ 2 28]]
k =  21
accuracy: 0.9556
```

```
Cross Validation scores : 0.9633
confusion matrix: [[58  2]
 [ 2 28]]
```

decision trees: 96.66666666666667
Logisitic regression: 65.55555555555556

DATASET 3

In [25]:
```
df = pd.read_csv('exp1c.csv')
x = df.iloc[:,0];
y = df.iloc[:,1];
plot.scatter(x,y, c= df["class"]);
```



In [26]:
```
df = pd.read_csv('exp1c.csv')
X = df.iloc[:,0:2];
y = df['class'];

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state = 40)

print('\033[1m' + 'Normal')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i)

    classifier.fit(X_train,y_train)

    y_pred = classifier.predict(X_test)


    accuracy = accuracy_score(y_test,y_pred,)


    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)

    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)

#model with distance as weights

print('\033[1m' + 'Distance as weights')
print ('\033[0m')

k_value =  [1,5,11,21]
for i in k_value:
    classifier = KNeighborsClassifier(n_neighbors=i,weights = 'distance')

    classifier.fit(X_train,y_train)

    y_pred = classifier.predict(X_test)
```

```
    accuracy = accuracy_score(y_test,y_pred,)


    cm = confusion_matrix(y_test,y_pred)
    scores = cross_val_score(classifier,X,y,cv = 10)

    print("k = ", i)
    print("accuracy: {}".format(round(accuracy,4)))
    print("Cross Validation scores : {}".format(round(np.mean(scores),4)))
    print("confusion matrix:",cm)


print('\033[1m')


#Decision Trees
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('decision trees:',scores)

#Logistic regression
classifier = LogisticRegression(random_state=0, solver='lbfgs')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = accuracy_score(y_test,y_pred).mean()*100
print('Logisitic regression:',scores)
```

Normal

```
k =  1
accuracy: 0.9394
Cross Validation scores : 0.9409
confusion matrix: [[61  2]
 [ 2  1]]
k =  5
accuracy: 0.9848
Cross Validation scores : 0.9773
confusion matrix: [[63  0]
 [ 1  2]]
k =  11
accuracy: 0.9848
Cross Validation scores : 0.9455
confusion matrix: [[63  0]
 [ 1  2]]
k =  21
accuracy: 0.9697
Cross Validation scores : 0.9364
confusion matrix: [[63  0]
 [ 2  1]]
```

Distance as weights

```
k =  1
accuracy: 0.9394
Cross Validation scores : 0.9409
confusion matrix: [[61  2]
 [ 2  1]]
k =  5
accuracy: 0.9848
Cross Validation scores : 0.9773
confusion matrix: [[63  0]
 [ 1  2]]
k =  11
accuracy: 0.9848
Cross Validation scores : 0.9682
confusion matrix: [[63  0]
 [ 1  2]]
k =  21
accuracy: 0.9848
```

```
Cross Validation scores : 0.9682
confusion matrix: [[63  0]
 [ 1  2]]
```

decision trees: 98.48484848484848
Logisitic regression: 95.45454545454545

Q. For each of the data sets, how does the 10-fold cross-validation accuracy rates and confusion matrices vary as k increases?

In the first dataset,performance increases as K increases. However, in the last two datasets, at first it increases and then decreases as K gets larger. This is due to the fact that last two datasets are less balanced that the first.

Q.For each k Nearest Neighbor classifier (k in {1, 5, 11, 21}), how does the 10-fold cross-validation accuracy rates and confusion matrices vary over the three data sets?

Performance of the classifier is best for the first dataset, whereas it is worst for the last dataset. This again is due to the imabalance in the 2 last datasets.

Q.Repeat (b) and (c) for the Nearest Neighbor classifier that employs distance weighting by setting option weights ='distance'. Comment on the change in the generalization performance of the classifiers from the unweighted version for the same values of k and the reason for this change?

When K=1, giving distance weighting option doesn't affect the model. But for K=5,11,21. outputs of distance weighting performed better than the unweighted classifiers. This is because distance weights option gives different weights to the neigbours according to the distances between the neighbour and the sample. The closer the neighbour is, the larger weight it will get. This method proves useful for classification especially when there is a class which has plenty of instances and other classes just have small amounts of instances.

Q. Run decision trees and logistic regression classifiers on the three data sets from the Experiment 1 cluster, and compare their generalization performance with that of the various k-NN classifiers.

A KNN classifer with a suitable K is the best choice to classify these datasets. It is know that logistic regression classifier is not good for non linear data Looking into the results,for other datasets (exp1b and exp1c) performance of logistic regression classifier would improve because for example in exp1c dataset we have 200 instances of class 1 and only 20 instances of class 2. That's why it will classify every new input instance as class 1 and increase its accuracy to 90 % the same as the accuracy of the tree for dataset exp1c.

Assignment 1.2

In [27]:
```
df = pd.read_csv('exp2a.csv')
X = df.iloc[:,0:4];
y = df['class'];
#Decision trees
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)

#KNN
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('KNN:',scores)

#Gaussian Naive bayes
classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('Naive bayes:',scores)

#Logistic Regression
classifier = LogisticRegression(random_state=1, solver='lbfgs',multi_class='multinomial')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('Logisitic regression:',scores)

#SVM
classifier = svm.SVC(gamma='scale')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('SVM:',scores)
```

```
Decision Tree: 100.0
KNN: 100.0
Naive bayes: 44.59131859131859
Logisitic regression: 44.59131859131859
SVM: 100.0
```

Q. What classifiers have a bad generalization performance and what classifiers have a good generalization performance?

SVM, decision trees and KNN have the best performance of the lot with cross validation score of 100%. After playing with the dataset with different attributes it was ovbserved that they can be divided into two pairs,

1. pair 1 with a0003 and a0007
2. pair 2 with a0001 and a0005

If both the attributes in the pair are removed, the performance of the classifier is decreasing, whereas if only one of the is removed, the performance is unaffected.

Q. Can you improve the classifiers from (a) that have a bad generalization?

Naive bayes and regression model has a cv scores of 44% and 44% respectively, this is bad because of the correlation of the attribute pairs. In order to get better results, we would need to have conditionally independent attributes.This can be done by having a parameter for variable smoothing that can be set to a small value.

In [28]:
```
#improving the performance of naive bayes and regression model

#Gaussian Naive bayes
classifier = GaussianNB(var_smoothing=-0.5)
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('Naive bayes:',scores)
```

```
Naive bayes: 54.5913185913186

C:\Users\Balag\Anaconda3\lib\site-packages\sklearn\naive_bayes.py:436: RuntimeWarning: invalid value encountered in l
og
  n_ij = - 0.5 * np.sum(np.log(2. * np.pi * self.sigma_[i, :]))
```

C & D

In [29]:
```
df = pd.read_csv('exp2b.csv')
X = df.iloc[:,0:len(df.columns)-1];
y = df['class'];

#Decision trees
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)

#KNN
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('KNN:',scores)

#Gaussian Naive bayes
classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('Naive bayes:',scores)

#Logistic Regression
classifier = LogisticRegression(random_state=1, solver='lbfgs',multi_class='multinomial')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('Logisitic regression:',scores)

#SVM
classifier = svm.SVC(gamma='scale')
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('SVM:',scores)
```

```
Decision Tree: 73.72972972972973
KNN: 50.5896805896806
Naive bayes: 36.32841932841933
Logisitic regression: 42.78050778050777
SVM: 56.66994266994267
```

Q.Which classifier shows the biggest drop of the generalization performance?

Decision tree, KNN and SVM show the biggest drop in generalization performance. As we can infer from dataset 2b which includes 99 attributes, 4 of which are the same as dataset 2a, which means the rest of the attributes are just noise in the data. This could be a reason for big drop in performance of the models.

Q. Which classifier shows the smallest drop of the generalization performance?

Logistic regression performed the same as it did for the last dataset. If we visulize the data with noise and also we know that logistic regression is probabilistic, the noise in data 2 gets nulled out and the accuracy remains the same althoug it not a good performance compared to other better models.

Assignment 1.3

In [30]:
```
df = pd.read_csv('exp3a.csv')
X = df.iloc[:,0:len(df.columns)-1];
y = df['class'];

# Gaussian naive bayes

classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('naive bayes:',scores)

#KNN
k_value = (1,2,3,4,5)
k_scores = []
for k in k_value:
    knn = KNeighborsClassifier(n_neighbors=k,metric='euclidean')
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print('KNN:', k_scores)

#Decision Tree
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =1, test_size=0.2)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)
```

```
naive bayes: 95.89999999999999
KNN: [0.9880000000000001, 0.983, 0.9890000000000001, 0.983, 0.983]
Decision Tree: 96.7
```

In [31]:
```
df = pd.read_csv('exp3b.csv')
X = df.iloc[:,0:len(df.columns)-1];
y = df['class'];

# Gaussian naive bayes

classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('naive bayes:',scores)

#KNN
k_value = (1,2,3,4,5)
k_scores = []
for k in k_value:
    knn = KNeighborsClassifier(n_neighbors=k,metric='euclidean')
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print('KNN:', k_scores)

#Decision Tree
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =1, test_size=0.2)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)
```

```
naive bayes: 95.89999999999999
KNN: [0.808, 0.8019999999999999, 0.835, 0.8320000000000001, 0.8550000000000001]
Decision Tree: 71.89999999999999
```

In [32]:
```
df = pd.read_csv('exp3c.csv')
X = df.iloc[:,0:len(df.columns)-1];
y = df['class'];

# Gaussian naive bayes

classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('naive bayes:',scores)

#KNN
k_value = (1,2,3,4,5)
k_scores = []
for k in k_value:
    knn = KNeighborsClassifier(n_neighbors=k,metric='euclidean')
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print('KNN:', k_scores)

#Decision Tree
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =1, test_size=0.2)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)
```

```
naive bayes: 92.19999999999999
KNN: [0.73, 0.7089999999999999, 0.766, 0.767, 0.799]
Decision Tree: 63.9
```

In [33]:
```
df = pd.read_csv('exp3d.csv')
X = df.iloc[:,0:len(df.columns)-1];
y = df['class'];

# Gaussian naive bayes

classifier = GaussianNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10, scoring='accuracy').mean()*100
print('naive bayes:',scores)

#KNN
k_value = (1,2,3,4,5)
k_scores = []
for k in k_value:
    knn = KNeighborsClassifier(n_neighbors=k,metric='euclidean')
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print('KNN:', k_scores)

#Decision Tree
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =1, test_size=0.2)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores = cross_val_score(classifier,X,y, cv=10).mean()*100
print('Decision Tree:' ,scores)
```

```
naive bayes: 89.0
KNN: [0.6340000000000001, 0.636, 0.674, 0.6689999999999999, 0.7150000000000001]
Decision Tree: 55.3
```

Q.How does the performance of the different classifiers compare for each dataset?

Looking at the results, we can observe that for dataset 'a', KNN classifer outperformed others. Naive bayes was second best and decision tree was the poorest performing model. Dataset 'a' is the simplest one among all datasets. The reason why KNN was better than other is because of the distribution of the instances. Morever the decision tree model was large with lots of branches which causes overfitting which adversley affected it to be the worst performing algorithm.

In the further expirements, naive bayes performed the best overcoming KNN with k=1 and 5 while decision tree suffers from overfitting and poor pruning performance.

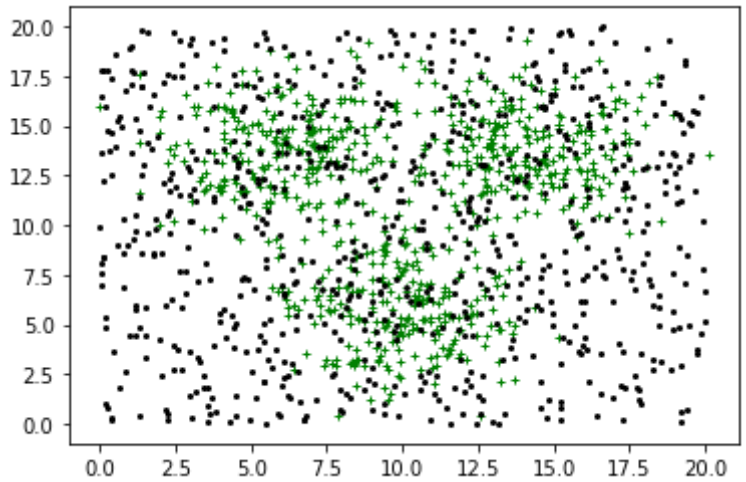
Q.How does the performance of each classifier vary as the number of variables is increased from datasets?

From the results it can be inferred that all models performed worse and worse as variables increased. Naive bayes is based on the assumption that attributes are conditionally independent. In these datasets, naive bayes outperformed others because as attributes increase, the classifier has ability to prevent noise and irrelevant attributes from degrading the performance. As for decision tree it was mentioned previously, as attributes increase the branches become wider and larger and with insuffcient pruning and overfitting, the results are very poor. For KNN, with increasing attributes and noise, small values of K caused instances on boundary to be classified incorrectly. In order to avoid this classification, the value of K should increase until maximum accuracy can be found.

EXTRA - Assignment 2

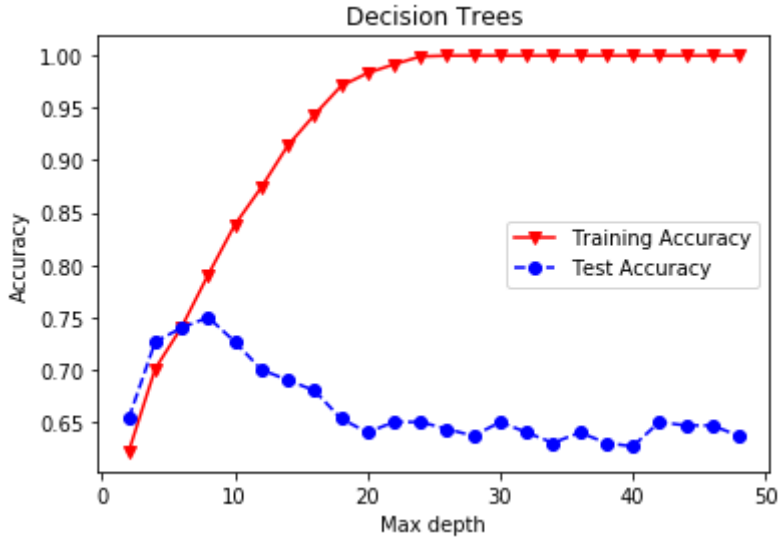
In [61]:
```
N = 1500

G1 = [6, 14]
G2 = [10, 6]
G3 = [14, 14]
covariance = [[4, 0], [0, 4]]  # diagonal covariance

np.random.seed(35)
X = np.random.multivariate_normal(G1, covariance, int(N/6))
X = np.concatenate((X, np.random.multivariate_normal(G2, covariance, int(N/6))))
X = np.concatenate((X, np.random.multivariate_normal(G3, covariance, int(N/6))))
X = np.concatenate((X, 20*np.random.rand(int(N/2),2)))
Y = np.concatenate((np.ones(int(N/2)),np.zeros(int(N/2))))

plot.plot(X[:int(N/2),0],X[:int(N/2),1],'g+',X[int(N/2):,0],X[int(N/2):,1],'k.',ms=4)
```

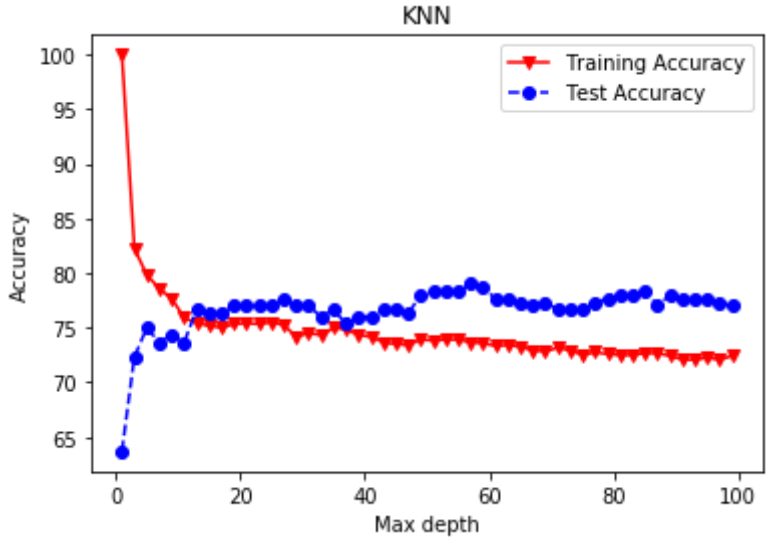
Out[61]:
```
[<matplotlib.lines.Line2D at 0x257ec299278>,
 <matplotlib.lines.Line2D at 0x257ec2997b8>]
```

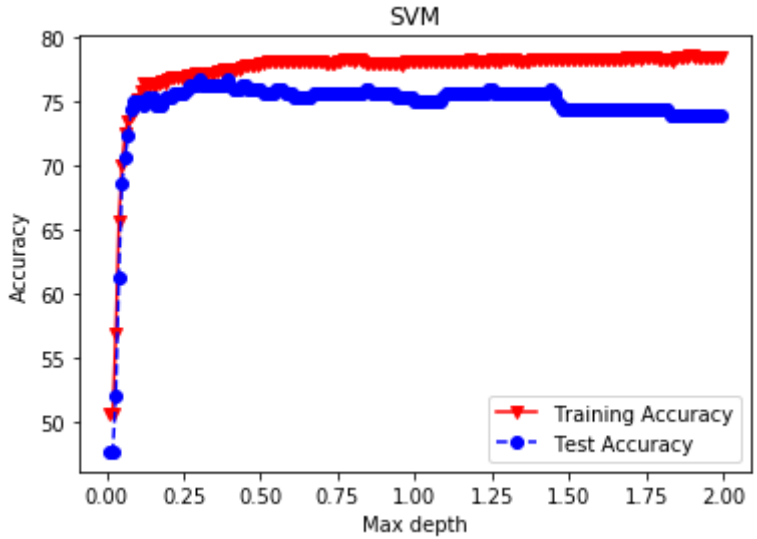


In [62]:
```
import warnings
warnings.filterwarnings("ignore")

#Decision Trees

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=20)
maxdepths = list(range(2,50,2))
acc_train = np.zeros(len(maxdepths))
acc_test = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    classifier = tree.DecisionTreeClassifier(max_depth=depth)
    classifier = classifier.fit(X_train, Y_train)
    Y_pred_train = classifier.predict(X_train)
    Y_pred_test = classifier.predict(X_test)
    acc_train[index] = accuracy_score(Y_train, Y_pred_train)
    acc_test[index] = accuracy_score(Y_test, Y_pred_test)
    index += 1

plot.plot(maxdepths,acc_train,'rv-',maxdepths,acc_test,'bo--')
plot.legend(['Training Accuracy','Test Accuracy'])
plot.xlabel('Max depth')
plot.ylabel('Accuracy')
plot.title('Decision Trees')
plot.show()

#Fitting KNN

maxdepths = list(range(1,101,2))
accuracy_train = list(range(1,101,2))
accuracy_test = list(range(1,101,2))

index = 0
for depth in maxdepths:
    classifier = KNeighborsClassifier(n_neighbors=depth)
    classifier = classifier.fit(X_train,Y_train)
    y_pred_train = classifier.predict(X_train)
    accuracy_score_train = accuracy_score(Y_train,y_pred_train)
    y_pred_test = classifier.predict(X_test)
    accuracy_score_test = accuracy_score(Y_test,y_pred_test)
```

```
    accuracy_train[index] = accuracy_score_train * 100
    accuracy_test[index] = accuracy_score_test * 100
    index = index + 1

accuracy_train = np.round(accuracy_train ,3)
accuracy_test = np.round(accuracy_test ,3)

plot.plot(maxdepths,accuracy_train,'rv-',maxdepths,accuracy_test,'bo--')
plot.legend(['Training Accuracy','Test Accuracy'])
plot.xlabel('Max depth')
plot.ylabel('Accuracy')
plot.title('KNN')
plot.show()

#SVM

maxdepths = list(np.arange(0.01,2,0.01))
accuracy_train = list(np.arange(0.01,2,0.01))
accuracy_test= list(np.arange(0.01,2,0.01))

index = 0
for depth in maxdepths:
    classifier = SVC(C = depth,kernel='rbf',gamma='auto')
    classifier = classifier.fit(X_train,Y_train)
    y_pred_train = classifier.predict(X_train)
    accuracy_score_train = accuracy_score(Y_train,y_pred_train)
    y_pred_test = classifier.predict(X_test)
    accuracy_score_test = accuracy_score(Y_test,y_pred_test)
    accuracy_train[index] = accuracy_score_train * 100
    accuracy_test[index] = accuracy_score_test * 100
    index = index + 1

accuracy_train = np.round(accuracy_train ,3)
accuracy_test = np.round(accuracy_test ,3)

plot.plot(maxdepths,accuracy_train,'rv-',maxdepths,accuracy_test,'bo--')
plot.legend(['Training Accuracy','Test Accuracy'])
plot.xlabel('Max depth')
plot.ylabel('Accuracy')
plot.title('SVM')
plot.show()
```


Decision Trees
Accuracy
Max depth
Training Accuracy
Test Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
10
20
30
40
50


KNN
Accuracy
Max depth
Training Accuracy
Test Accuracy
100
95
90
85
80
75
70
65
0
20
40
60
80
100



The plot above shows that training accuracy will continue to improve as the maximum depth of the tree increases (i.e., as the model becomes more complex). However, the test accuracy initially improves up to a maximum depth of 5, before it gradually decreases due to model overfitting.

As for KNN, the performance on the train data is decreasing sharply with increasing value of depth to a certain extent, and then the decrease more in slower form. Test Data accuracy is increasing to a certain point. Area of underfitting (1,7), Area of overfitting (7,100) and optimal k = 7

In [ ]: