In [1]:

```
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
```

In [2]:

```
data = pd.read_csv("F:/assignments/Sem 6 Assignments/ML Assignment 1/Q1_1/Iris.csv")
```

In [3]:

```
data.columns
```

Out[3]:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [4]:

```
data
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

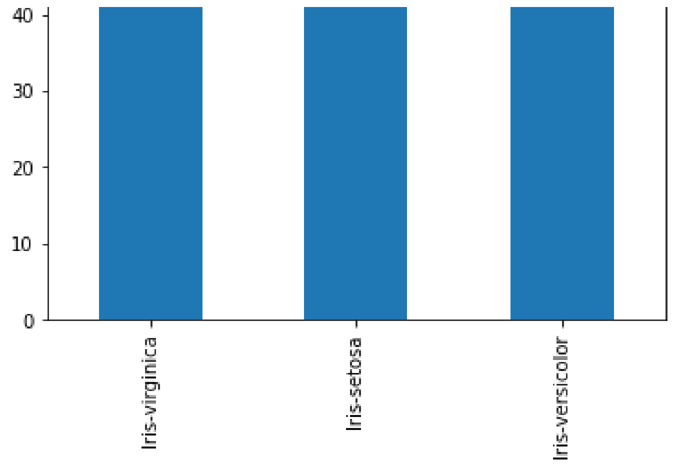
150 rows × 6 columns

In [5]:

```
df2=pd.DataFrame(data)
df2['Species'].value_counts().plot(kind='bar')
plt.figure()
df2.hist(column='SepalLengthCm')
plt.figure()
df2.hist(column='SepalWidthCm')
plt.figure()
df2.hist(column='PetalLengthCm')
plt.figure()
df2.hist(column='PetalWidthCm')
```

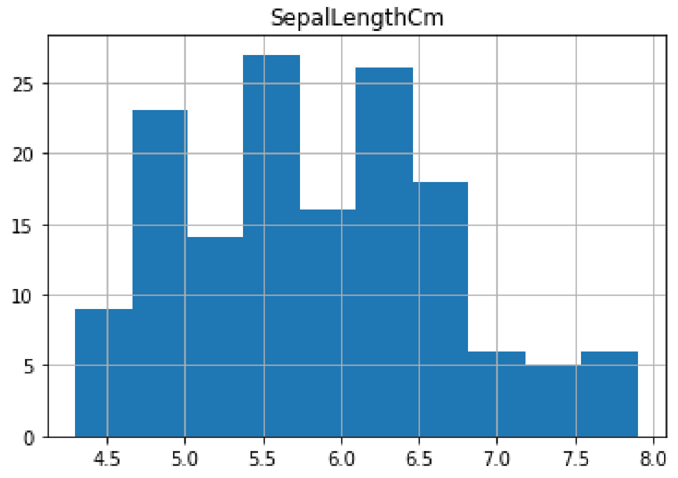
Out[5]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002388CDB1A48>]],
      dtype=object)
```



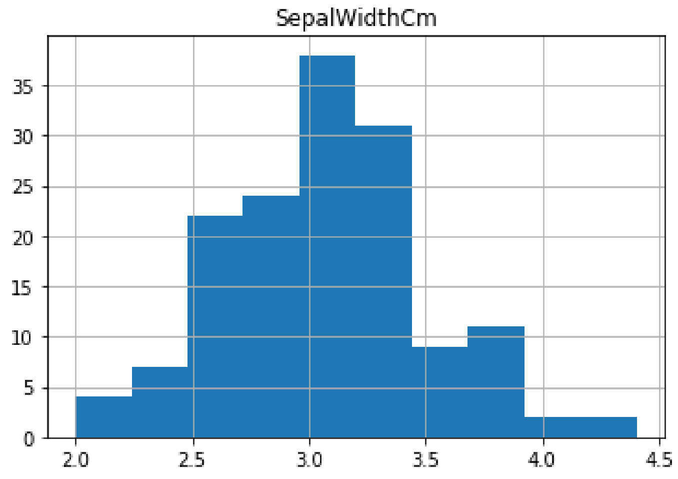
40
30
20
10
0
Iris-virginica
Iris-setosa
Iris-versicolor

```
<Figure size 432x288 with 0 Axes>
```



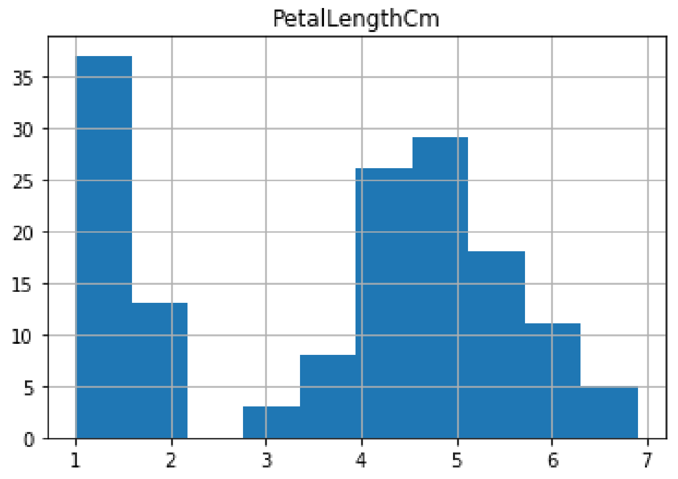
SepalLengthCm
25
20
15
10
5
0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

```
<Figure size 432x288 with 0 Axes>
```



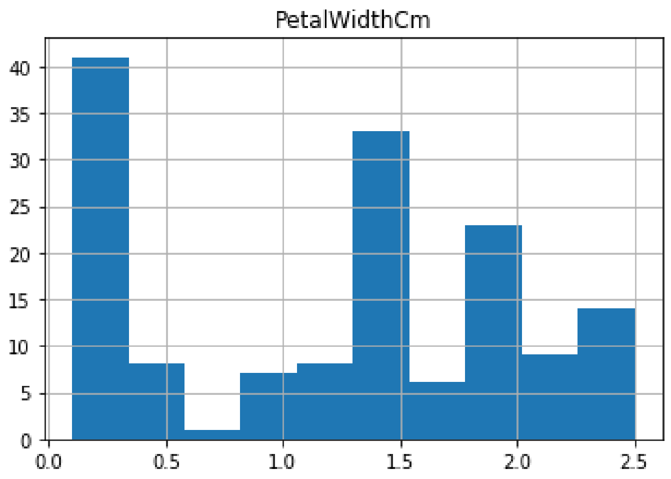
SepalWidthCm
35
30
25
20
15
10
5
0
2.0
2.5
3.0
3.5
4.0
4.5

```
<Figure size 432x288 with 0 Axes>
```



PetalLengthCm
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7

```
<Figure size 432x288 with 0 Axes>
```


PetalWidthCm
40
35
30
25
20
15
10
5
0
0.0
0.5
1.0
1.5
2.0
2.5

In [1]:

```
import numpy as np
import idx2numpy
import random
from matplotlib import pyplot as plt
import cv2
```

In [2]:

```
train_images = idx2numpy.convert_from_file('train-images.idx3-ubyte')
train_labels = idx2numpy.convert_from_file('train-labels.idx1-ubyte')
test_images = idx2numpy.convert_from_file('t10k-images.idx3-ubyte')
test_labels = idx2numpy.convert_from_file('t10k-labels.idx1-ubyte')
```

In [3]:

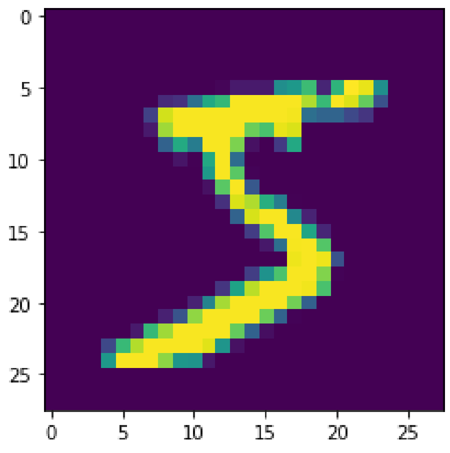
```
im1=train_images[0]
im2=train_images[56]
```

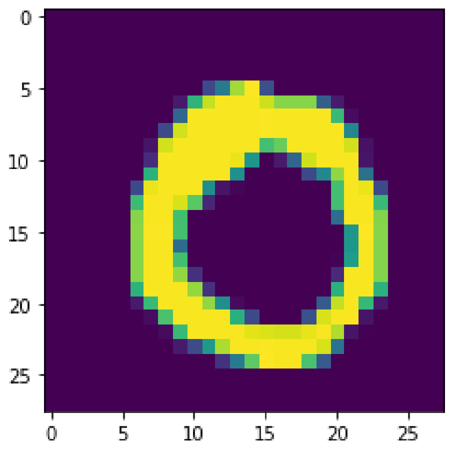
In [4]:

```
cv2.imwrite('color_img.jpg', im1)
plt.imshow(im1)
plt.figure()
cv2.imwrite('color_img.jpg', im2)
plt.imshow(im2)
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x2473c0eaf88>
```





In [5]:

```
new_train=[]
for i in range(0,10):
    c=0
    for j in range(0,len(train_labels)):
        if(train_labels[j]==i and c<=1000):
            new_train.append(train_images[j])
            c+=1
```

In [6]:

```
train_new=[]
#flatten images to a 1d array
for i in new_train:
    i=np.array(i)
    train_new.append(i.flatten())
train_images=train_new
```

Using TSNE for dimensionality reduction

In [7]:

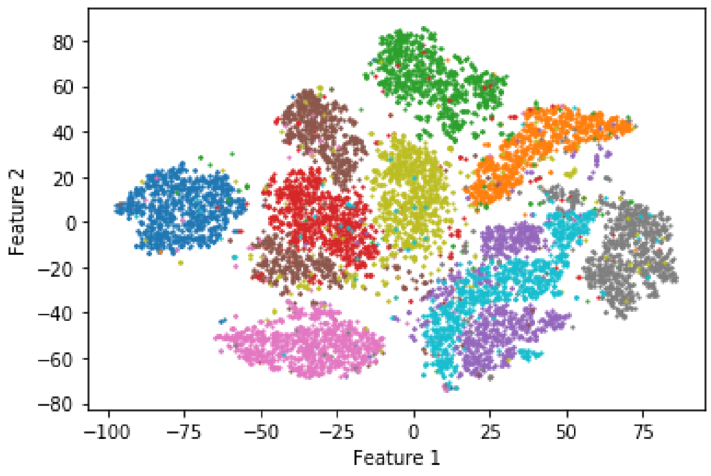
```
from sklearn.manifold import TSNE
embedded=X_embedded = TSNE(n_components=2, init='random').fit_transform(train_images)
```

In [8]:

```
x=[]
y=[]
for i in embedded:
    x.append(i[0])
    y.append(i[1])
```

In [11]:

```
for i in range(0,10):
    x1=[]
    y1=[]
    for j in range(i*1000,(i+1)*1000):
        x1.append(x[j])
        y1.append(y[j])
    plt.scatter(x1,y1,s=1.5)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



The Data is separable barring four classes that have seemed to mix up with each other (blue and purple, red and brown)

Linear Regression

In [1]:

```
import pandas as pd
import sklearn as sk
from sklearn import model_selection
from matplotlib import pyplot as plt
import numpy as np
from sklearn.metrics import r2_score
```

In [2]:

```
data = pd.read_csv("F:/assignments/Sem 6 Assignments/ML Assignment 1/Q2/abalone.data")
```

Raw Data

In [3]:

```
data
```

Out[3]:

	Sex	Length	Diameter	Height	Whole Weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...	...	...	...	...	...	...	...	...	...
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4177 rows × 9 columns

Visualization of dataset

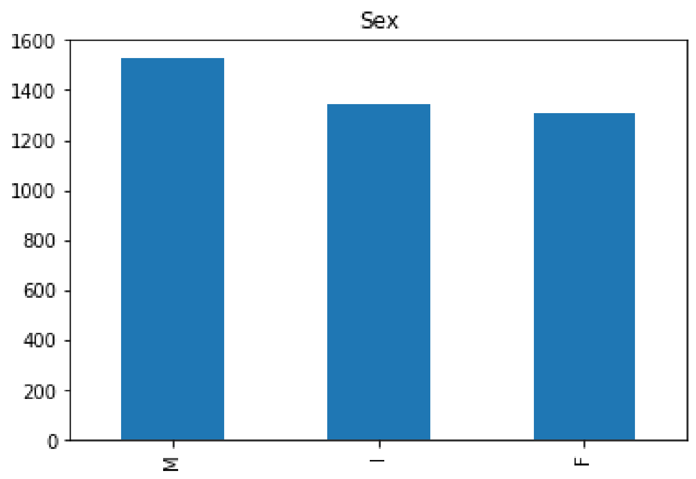
In [4]:

```
data['Sex'].value_counts().plot(kind='bar')
plt.title('Sex')
plt.figure()
data.hist(column='Length')
plt.figure()
data.hist(column='Diameter')
plt.figure()
data.hist(column='Height')
plt.figure()
data.hist(column='Whole Weight')
plt.figure()
data.hist(column='Shucked weight')
plt.figure()
data.hist(column='Viscera weight')
```

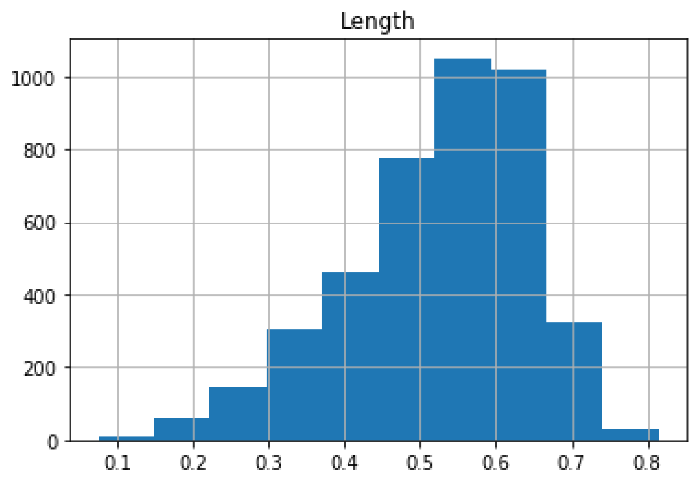
```
plt.figure()
data.hist(column='Shell weight')
```

Out[4]:

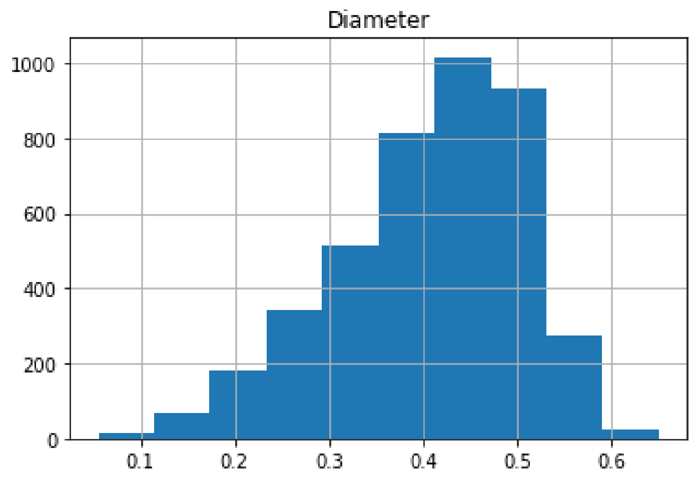
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000267A5FBB948>]],
      dtype=object)
```



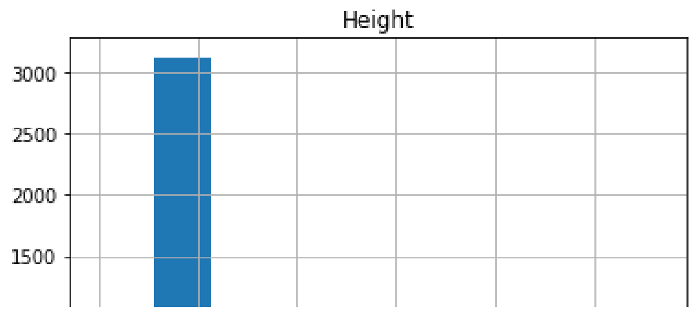
```
<Figure size 432x288 with 0 Axes>
```

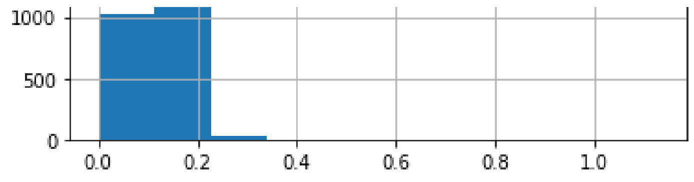


```
<Figure size 432x288 with 0 Axes>
```

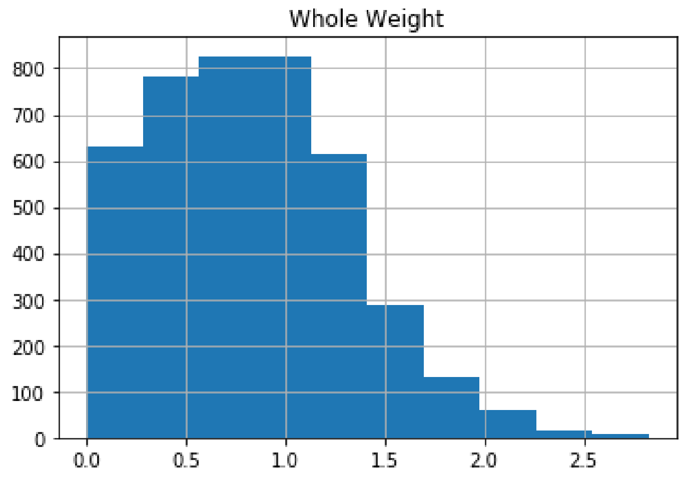


```
<Figure size 432x288 with 0 Axes>
```

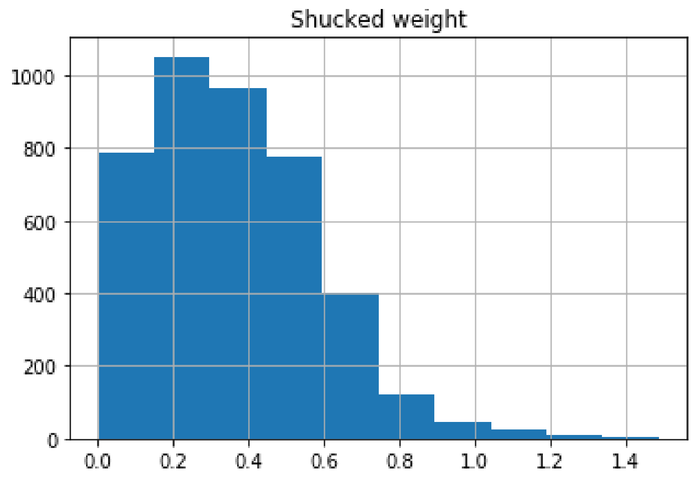



1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

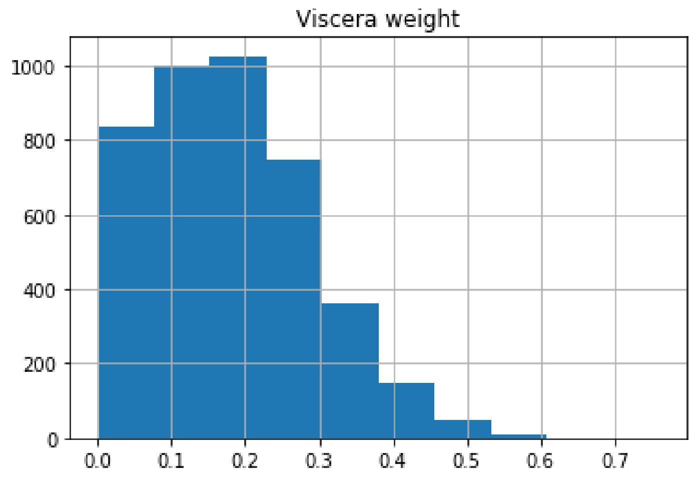
```
<Figure size 432x288 with 0 Axes>
```


Whole Weight
800
700
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5

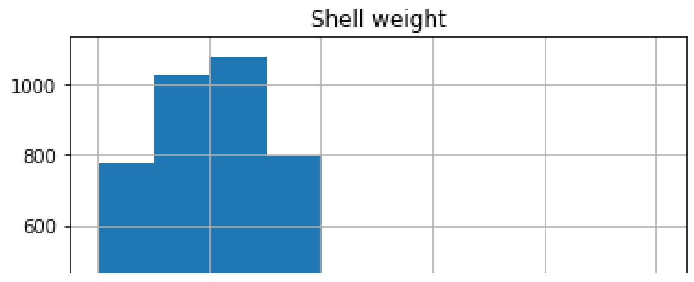
```
<Figure size 432x288 with 0 Axes>
```

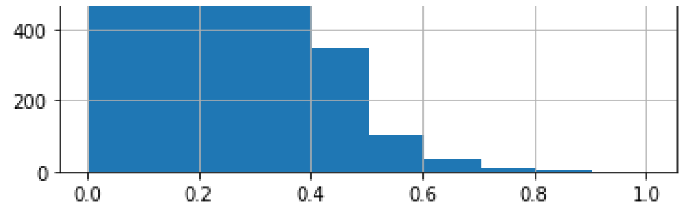

Shucked weight
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

```
<Figure size 432x288 with 0 Axes>
```


Viscera weight
1000
800
600
400
200
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

```
<Figure size 432x288 with 0 Axes>
```


Shell weight
1000
800
600



In [5]:

```
#one hot encoding for Sex
one_hot = pd.get_dummies(data['Sex'])
data = data.drop('Sex',axis = 1)
data = data.join(one_hot)
```

Data after One Hot Encoding for Discrete Values (Sex)

In [6]:

```
data
```

Out[6]:

	Length	Diameter	Height	Whole Weight	Shucked weight	Viscera weight	Shell weight	Rings	F	I	M
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15	0	0	1
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7	0	0	1
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9	1	0	0
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10	0	0	1
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
4172	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11	1	0	0
4173	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10	0	0	1
4174	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9	0	0	1
4175	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10	1	0	0
4176	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12	0	0	1

4177 rows × 11 columns

Min-Max Scaling

In [7]:

```
#min max normalization
for column in data.columns:
    data[column]=(data[column]-data[column].min())/(data[column].max()-data[column].min(
))
```

Splitting Dataset into 90% train 10% test

In [8]:

```
train_,test=model_selection.train_test_split(data, test_size=0.1, train_size=0.9)
```

Note: Training is done using the 5-fold validation. RMSE is reported on the validation set. Best Model from the 5 folds is determined using the RMSE value. Lower the RMSE, better is

the model

Linear Regression w/o Regularization LR=0.00001

In [9]:

```
def linear(w,x_train,y_train,x_test,y_test):
    lr=0.00001
    x_train=np.array(x_train)
    y_train=np.array(y_train)
    x_test=np.array(x_test)
    y_test=np.array(y_test)

    for i in range(x_train.shape[0]): #3007
        grad=0
        for j in range(x_train.shape[1]): #10
            grad=grad+(np.dot(x_train[i],w)-y_train[i])*x_train[i][j]
            w[j]=w[j]-lr*grad

    #validation
    y_pred=np.dot(x_test,w)
    mse=np.sqrt(np.sum(np.square(y_pred-y_test)))/x_test.shape[0]
    return(w,mse)
```

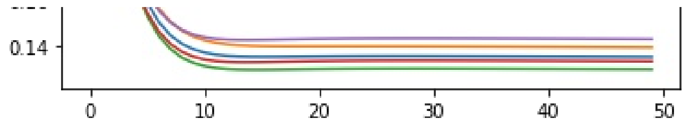
In [10]:

```
best_rmse=10000
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)

y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# print(train_.shape)
# print(y.shape)
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    w=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    for i in range(50):
        w,mse=linear(w,x_train,y_train,x_test,y_test)
        rmse.append(mse)

    plt.plot(rmse)
    if(sum(rmse)<best_rmse):
        w_best=w
        best_rmse=sum(rmse)
plt.figure()
```

Out[10]:

```
<Figure size 432x288 with 0 Axes>
```

0.28
0.26
0.24
0.22
0.20
0.18
0.16



```
<Figure size 432x288 with 0 Axes>
```

RMSE for model without penalty on test set

In [11]:

```
y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
new_test=np.array(new_test)
y_pred=np.dot(new_test,w)
y=np.array(y)
# np.sqrt(np.sum(np.square((y-y_pred))))/y.size
# r2_score(y,y_pred)
np.sqrt(np.sum(np.square(y_pred-y)))/len(y)
```

Out[11]:

```
0.14695609463470916
```

L2 Regularization LR:0.00001 lambda=0.01

Notice the loss function here. The square of the norm of the weights of the model are added to the loss function.

In [12]:

```
def l2(w,x_train,y_train,x_test,y_test):
    lr=0.00001
    x_train=np.array(x_train)
    y_train=np.array(y_train)
    x_test=np.array(x_test)
    y_test=np.array(y_test)
    lamb=0.0001

    for i in range(x_train.shape[0]): #3007
        grad=0
        for j in range(x_train.shape[1]): #10
            grad=grad+(np.dot(x_train[i],w)-y_train[i])*x_train[i][j] + lamb*w[j]
            w[j]=w[j]-lr*grad

    #validation
    y_pred=np.dot(x_test,w)
    mse=(np.sqrt(np.sum(np.square(y_pred-y_test))) + lamb*np.square(np.linalg.norm(w)))
/x_test.shape[0]#notice the change in loss function
    return(w,mse)
```

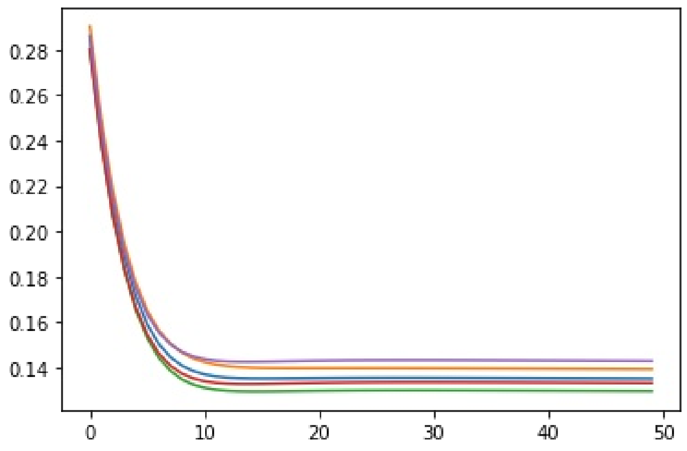
In [13]:

```
best_rmse=10000
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)

y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# print(train_.shape)
# print(y.shape)
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
```

```
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    wl2=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    for i in range(50):
        wl2,mse=l2(wl2,x_train,y_train,x_test,y_test)
        rmse.append(mse)

    plt.plot(rmse)
    if(sum(rmse)<best_rmse):
        w_best_l2=wl2
        best_rmse=sum(rmse)
plt.figure()
```

Out[13]:

```
<Figure size 432x288 with 0 Axes>
```



```
<Figure size 432x288 with 0 Axes>
```

RMSE for L2 on test set

In [14]:

```
y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
new_test=np.array(new_test)
y_pred=np.dot(new_test,w_best_l2)
y=np.array(y)
# np.sqrt(np.sum(np.square((y-y_pred))))/y.size
# r2_score(y,y_pred)
np.sqrt(np.sum(np.square(y_pred-y)))/len(y)
```

Out[14]:

```
0.14705283270813393
```

L1 Regularization LR:0.00001, lambda=0.001

Notice the loss function here. The absolute value of the weights of the model are added to the loss function

In [15]:

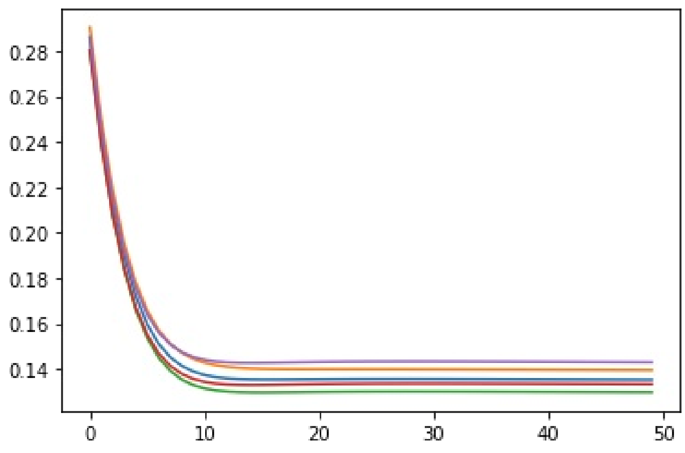
```
def l1(w,x_train,y_train,x_test,y_test):
    lr=0.00001
    x_train=np.array(x_train)
    y_train=np.array(y_train)
    x_test=np.array(x_test)
    y_test=np.array(y_test)
    constant=0.5
    lamb=0.001

    for i in range(x_train.shape[0]): #3007
        grad=0
        for j in range(x_train.shape[1]): #10
            grad=grad+(np.dot(x_train[i],w)-y_train[i])*x_train[i][j] + lamb*constant
            w[j]=w[j]-lr*grad

    #validation
    y_pred=np.dot(x_test,w)
    mse=(np.sqrt(np.sum(np.square(y_pred-y_test))) + lamb*np.sum(np.abs(w)))/x_test.shap
e[0]#notice the change in loss function here
    return(w,mse)
```

In [16]:

```
best_rmse=10000
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)

y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# print(train_.shape)
# print(y.shape)
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    wl1=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    for i in range(50):
        wl1,mse=l1(wl1,x_train,y_train,x_test,y_test)
        rmse.append(mse)

    plt.plot(rmse)
    if(sum(rmse)<best_rmse):
        w_best_l1=wl1
        best_rmse=sum(rmse)
plt.figure()
```

Out[16]:

```
<Figure size 432x288 with 0 Axes>
```



```
<Figure size 432x288 with 0 Axes>
```

RMSE for L1 on test set

In [17]:

```
y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
new_test=np.array(new_test)
y_pred=np.dot(new_test,w_best_l1)
y=np.array(y)
# np.sqrt(np.sum(np.square((y-y_pred))))/y.size
np.sqrt(np.sum(np.square(y_pred-y)))/len(y)
```

Out[17]:

```
0.1471399932030919
```

SkLearn implementation of Linear Regression no Penalty

In [18]:

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)


y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# y1=test['Rings']
# new_test=test.drop(['Rings'],axis=1)
# new_test=np.array(new_test)
best_rmse=10000
# print(train_.shape)
# print(y.shape)

for train_index, test_index in kf.split(train_):
#     rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
#     wl1=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    y_pred=reg.predict(x_test)
    rmse=np.sqrt(np.sum(np.square(y_pred-y_test)))/len(y_pred)
#     print(rmse)
    if(rmse<best_rmse):
        best_rmse=rmse
        best_model=reg
#     print(reg.score(new_test,y1))

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
```

RMSE for inbuilt Linear Regression on test set

In [19]:

```
from sklearn.metrics import mean_squared_error

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
y_pred=best_model.predict(new_test)
rms = mean_squared_error(y, y_pred,squared=False)
rms
```

Out[19]:

```
0.08471284911075605
```

SkLearn Implementation of Ridge (L2)

In [20]:

```python
from sklearn.linear_model import Ridge
reg=Ridge(alpha=0.000001)
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)


y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# y1=test['Rings']
# new_test=test.drop(['Rings'],axis=1)
# new_test=np.array(new_test)
best_rmse=10000
# print(train_.shape)
# print(y.shape)

for train_index, test_index in kf.split(train_):
#     rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
#     wl1=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    y_pred=reg.predict(x_test)
    rmse=np.sqrt(np.sum(np.square(y_pred-y_test)))/len(y_pred)
#     print(rmse)
    if(rmse<best_rmse):
        best_rmse=rmse
        best_model=reg
#     print(reg.score(new_test,y1))

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
```

RMSE for Ridge on test set

In [21]:

```python
from sklearn.metrics import mean_squared_error

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
y_pred=best_model.predict(new_test)
rms = mean_squared_error(y, y_pred,squared=False)
rms
```

Out[21]:

```
0.08471285141421189
```

SkLearn Implementation of Lasso (L1)

In [22]:

```python
from sklearn.linear_model import Lasso
```

```
reg=Lasso(alpha=0.00001)
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)


y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# y1=test['Rings']
# new_test=test.drop(['Rings'],axis=1)
# new_test=np.array(new_test)
best_rmse=10000
# print(train_.shape)
# print(y.shape)

for train_index, test_index in kf.split(train_):
#     rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
#     wl1=np.zeros((x_train.shape[1],1))
#     print(X_train.shape)
#     print(X_test.shape)
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    y_pred=reg.predict(x_test)
    rmse=np.sqrt(np.sum(np.square(y_pred-y_test)))/len(y_pred)
#     print(rmse)
    if(rmse<best_rmse):
        best_rmse=rmse
        best_model=reg
#     print(reg.score(new_test,y1))

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
```

RMSE for Lasso on test set

In [23]:

```
from sklearn.metrics import mean_squared_error

y=test['Rings']
new_test=test.drop(['Rings'],axis=1)
y_pred=best_model.predict(new_test)
rms = mean_squared_error(y, y_pred,squared=False)
rms
```

Out[23]:

```
0.08480590164862717
```

The introduction of penalty (L1 and L2) does not seem to have much effect on the RMSE value. After running multiple runs of this code, no conclusive best model can be found. Sometimes L1 performs the best, sometimes L2 and sometimes no penalty is the best model. The RMSE values are very close to each other.

The inbuilt regression models give slightly lower RMSE (about 0.06)

Closed Form RMSE on validation set

In [24]:

```
kf=model_selection.KFold(n_splits=5)
```

```
y = train_['Rings']
new_train = train_.drop(['Rings'], axis=1)
# y1=test['Rings']
# new_test=test.drop(['Rings'],axis=1)
# new_test=np.array(new_test)

idx=1
for train_index, test_index in kf.split(train_):
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    w=np.dot(np.linalg.inv(np.dot(np.transpose(x_train),x_train)),np.dot(np.transpose(x_
train),y_train))

    y_pred=np.dot(x_test,w)
    y_test=np.array(y_test)
    print("RMSE of fold "+str(idx)+":",np.sqrt(np.sum(np.square(y_pred-y_test)))/len(y_t
est))
    idx+=1
```

```
RMSE of fold 1: 0.0027954060476873955
RMSE of fold 2: 0.003147262374071904
RMSE of fold 3: 0.002681020281461619
RMSE of fold 4: 0.0026932049414070255
RMSE of fold 5: 0.0029880871585471888
```

Logistic Regression on Ionosphere Dataset

In [1]:

```
import pandas as pd
import sklearn as sk
from sklearn import model_selection
from matplotlib import pyplot as plt
import numpy as np
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
data = pd.read_csv("F:/assignments/Sem 6 Assignments/ML Assignment 1/Q3/ionosphere_data_k
aggle.csv")
data
```

Out[2]:

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	...	feature26	feature27
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290
...	...	...	...	...	...	...	...	...	...	...	...	...	...
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	0.93522
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	0.92489
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	0.89147
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	0.81147

351 rows × 35 columns

Replacing g and b with 1 and 0 in the label column

In [3]:

```
data['label'].replace({'g':1,'b':0},inplace=True)
data
```

Out[3]:

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	...	feature26	feature27
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	feature10	...	feature26	feature27
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290
...	...	...	...	...	...	...	...	...	...	...	...	...	...
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	0.93522
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	0.92489
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	0.89147
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	0.81147

351 rows × 35 columns

Splitting Dataset into 90% train 10% test

In [4]:

```
train_,test=model_selection.train_test_split(data, test_size=0.1, train_size=0.9)
```

In [5]:

```
stats=pd.DataFrame()
stats["mean"]=train_.mean()
stats["Var"]=train_.var()
stats
```

Out[5]:

	mean	Var
feature1	0.892063	0.096593
feature2	0.000000	0.000000
feature3	0.642893	0.238592
feature4	0.051843	0.192843
feature5	0.608174	0.254543
feature6	0.123541	0.213319
feature7	0.555684	0.226843
feature8	0.118215	0.278598
feature9	0.501287	0.260881
feature10	0.189137	0.244225
feature11	0.471601	0.309997
feature12	0.171377	0.239788
feature13	0.395721	0.388738
feature14	0.110508	0.245293
feature15	0.333867	0.430329

	mean	Var
feature16	0.075898	0.210811
feature17	0.366644	0.385899
feature18	0.004513	0.249589
feature19	0.348317	0.394385
feature20	-0.008207	0.274522
feature21	0.336343	0.364187
feature22	0.024012	0.267117
feature23	0.361513	0.361545
feature24	-0.051329	0.277417
feature25	0.378971	0.336360
feature26	-0.058471	0.261215
feature27	0.540448	0.263351
feature28	-0.065516	0.304881
feature29	0.370661	0.326723
feature30	-0.005855	0.253571
feature31	0.340722	0.326127
feature32	-0.000377	0.264168
feature33	0.354652	0.264820
feature34	0.014154	0.222551
label	0.638095	0.231665

In [6]:

```
stats['Var'].nlargest(5)
```

Out[6]:

```
feature15    0.430329
feature19    0.394385
feature13    0.388738
feature17    0.385899
feature21    0.364187
Name: Var, dtype: float64
```

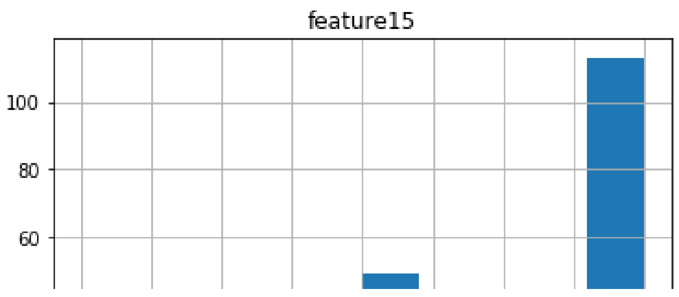
Plotting Histograms of the Features with the highest variances (top 5)

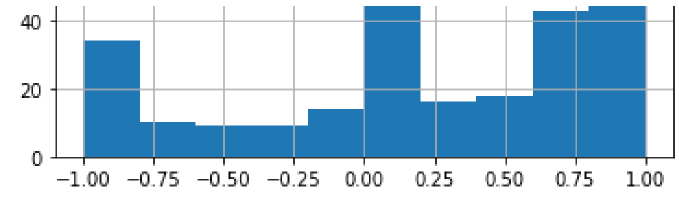
In [7]:

```
train_.hist(column='feature15')
train_.hist(column='feature19')
train_.hist(column='feature13')
train_.hist(column='feature17')
train_.hist(column='feature21')
```

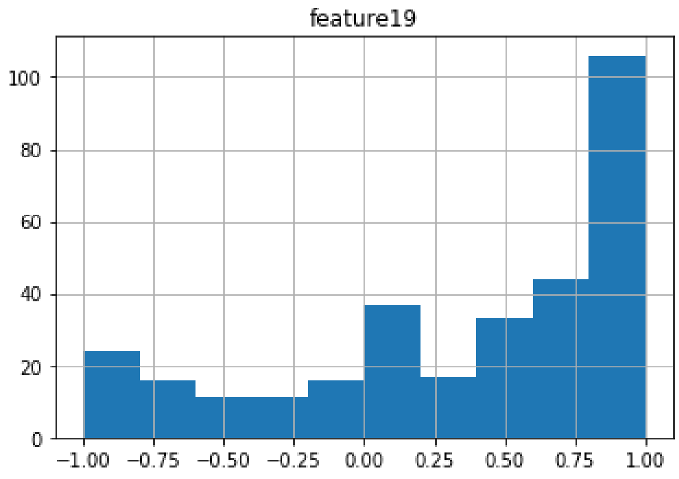
Out[7]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000027AC8F32EC8>]],
      dtype=object)
```

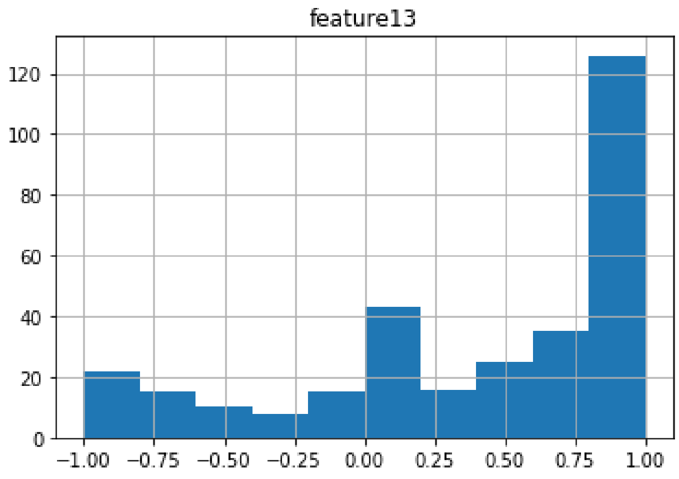




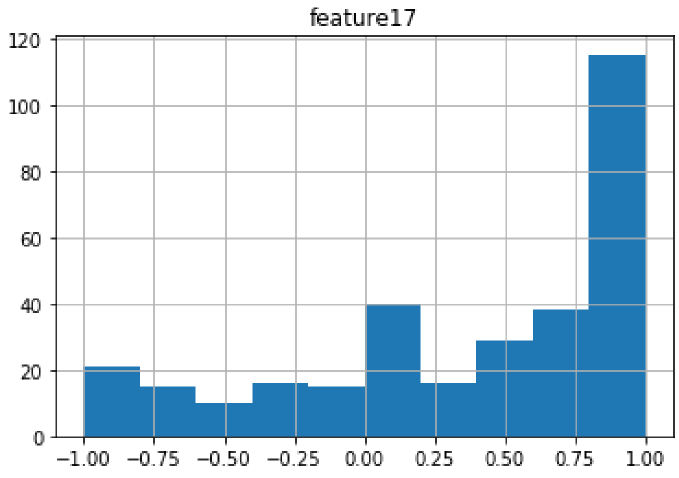
40
20
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



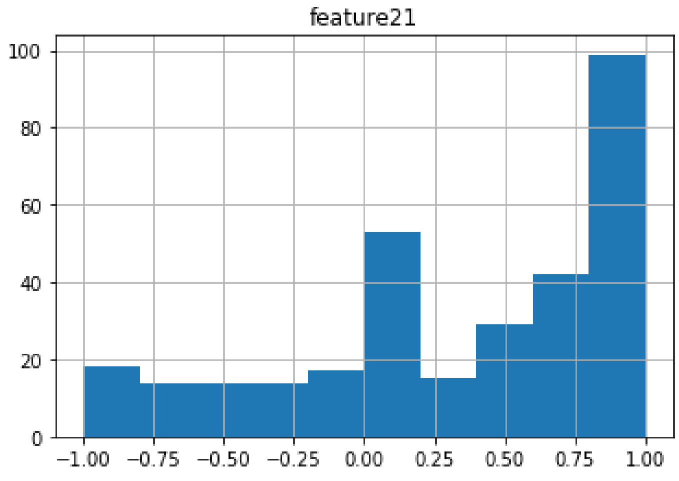
feature19
100
80
60
40
20
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



feature13
120
100
80
60
40
20
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



feature17
120
100
80
60
40
20
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



feature21
100
80
60
40
20
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

In [8]:

```
reg=LogisticRegression(penalty='none',max_iter=1000)
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)
y1=test['label']
new_test=test.drop(['label'],axis=1)
new_test=np.array(new_test)
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

y = train_['label']
new_train = train_.drop(['label'], axis=1)
# print(train_.shape)
# print(y.shape)
stats1=[]
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    ac=accuracy_score(reg.predict(new_test), y1)

    stats1.append(list(precision_recall_fscore_support(reg.predict(new_test), y1, averag
e='micro')))
    stats1[-1].append(ac)

# y=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
# from sklearn.metrics import precision_recall_fscore_support
stats1=pd.DataFrame(stats1)
stats1.columns =['Precision', 'Recall', 'F1-Score', 'Support','Accuracy']
stats1=stats1.drop(['Support'],axis=1)
print("Logistic Regression Stats without PCA")
print(stats1)
```

```
Logistic Regression Stats without PCA
   Precision    Recall  F1-Score  Accuracy
0   0.861111  0.861111  0.861111  0.861111
1   0.861111  0.861111  0.861111  0.861111
2   0.861111  0.861111  0.861111  0.861111
3   0.888889  0.888889  0.888889  0.888889
4   0.861111  0.861111  0.861111  0.861111
```

PCA for dimensionality reduction

In [9]:

```
from sklearn.decomposition import PCA
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)
for val in range(90,100,1):
    pca = PCA(val/100)
    y1=test['label']
    new_test=test.drop(['label'],axis=1)
    new_test=np.array(new_test)
    # print(new_test.shape)
    y = train_['label']
    new_train = train_.drop(['label'], axis=1)
    # print(train_.shape)
    # print(y.shape)
    stats1=[]
    ind=0
    for train_index, test_index in kf.split(train_):
        x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    #     print(y_test.shape)

    #     components=pca.fit_transform(x_train)
    #     pca.fit(x_train)
    #     x_train= pca.transform(x_train)
    #     pca.fit(x_train)
    #     x_train= pca.transform(x_train)
        pca.fit(x_train)
        x_train= pca.transform(x_train)
        new_test1=pca.transform(new_test)
    #     ind=10
    #     y_train=y_train.values.reshape(-1, 1)
    #     y_test=y_test.values.reshape(-1, 1)
    #     print(y_test.shape)
    #     y_train=pca.transform(y_train)
    #     y_test=pca.transform(y_test)
        reg=LogisticRegression(penalty='none',max_iter=1000)
        reg.fit(x_train,y_train)
        ac=accuracy_score(reg.predict(new_test1), y1)

        stats1.append(list(precision_recall_fscore_support(reg.predict(new_test1), y1, a
verage='micro')))
        stats1[-1].append(ac)

    # y=test['label']
    # new_test=test.drop(['label'],axis=1)
    # new_test=np.array(new_test)
    # from sklearn.metrics import precision_recall_fscore_support
    stats1=pd.DataFrame(stats1)
    stats1.columns =['Precision', 'Recall', 'F1-Score', 'Support','Accuracy']
    stats1=stats1.drop(['Support'],axis=1)
    print("Logistic Regression Stats with PCA with variance", val/100)
    print(stats1)
    print()
```

```
Logistic Regression Stats with PCA with variance 0.9
   Precision    Recall  F1-Score  Accuracy
0   0.861111  0.861111  0.861111  0.861111
1   0.861111  0.861111  0.861111  0.861111
2   0.861111  0.861111  0.861111  0.861111
3   0.888889  0.888889  0.888889  0.888889
4   0.888889  0.888889  0.888889  0.888889

Logistic Regression Stats with PCA with variance 0.91
   Precision    Recall  F1-Score  Accuracy
0   0.861111  0.861111  0.861111  0.861111
1   0.861111  0.861111  0.861111  0.861111
2   0.861111  0.861111  0.861111  0.861111
3   0.888889  0.888889  0.888889  0.888889
4   0.888889  0.888889  0.888889  0.888889

Logistic Regression Stats with PCA with variance 0.92
   Precision    Recall  F1-Score  Accuracy
0   0.916667  0.916667  0.916667  0.916667
1   0.861111  0.861111  0.861111  0.861111
2   0.861111  0.861111  0.861111  0.861111
3   0.888889  0.888889  0.888889  0.888889
4   0.916667  0.916667  0.916667  0.916667

Logistic Regression Stats with PCA with variance 0.93
   Precision    Recall  F1-Score  Accuracy
0   0.888889  0.888889  0.888889  0.888889
1   0.861111  0.861111  0.861111  0.861111
2   0.861111  0.861111  0.861111  0.861111
3   0.916667  0.916667  0.916667  0.916667
4   0.916667  0.916667  0.916667  0.916667
```

```
Logistic Regression Stats with PCA with variance 0.94
   Precision    Recall  F1-Score  Accuracy
0   0.916667  0.916667  0.916667  0.916667
1   0.888889  0.888889  0.888889  0.888889
2   0.888889  0.888889  0.888889  0.888889
3   0.916667  0.916667  0.916667  0.916667
4   0.916667  0.916667  0.916667  0.916667

Logistic Regression Stats with PCA with variance 0.95
   Precision    Recall  F1-Score  Accuracy
0   0.916667  0.916667  0.916667  0.916667
1   0.916667  0.916667  0.916667  0.916667
2   0.888889  0.888889  0.888889  0.888889
3   0.944444  0.944444  0.944444  0.944444
4   0.916667  0.916667  0.916667  0.916667

Logistic Regression Stats with PCA with variance 0.96
   Precision    Recall  F1-Score  Accuracy
0   0.916667  0.916667  0.916667  0.916667
1   0.916667  0.916667  0.916667  0.916667
2   0.888889  0.888889  0.888889  0.888889
3   0.916667  0.916667  0.916667  0.916667
4   0.916667  0.916667  0.916667  0.916667

Logistic Regression Stats with PCA with variance 0.97
   Precision    Recall  F1-Score  Accuracy
0   0.916667  0.916667  0.916667  0.916667
1   0.888889  0.888889  0.888889  0.888889
2   0.888889  0.888889  0.888889  0.888889
3   0.916667  0.916667  0.916667  0.916667
4   0.916667  0.916667  0.916667  0.916667

Logistic Regression Stats with PCA with variance 0.98
   Precision    Recall  F1-Score  Accuracy
0   0.888889  0.888889  0.888889  0.888889
1   0.888889  0.888889  0.888889  0.888889
2   0.916667  0.916667  0.916667  0.916667
3   0.916667  0.916667  0.916667  0.916667
4   0.888889  0.888889  0.888889  0.888889

Logistic Regression Stats with PCA with variance 0.99
   Precision    Recall  F1-Score  Accuracy
0   0.888889  0.888889  0.888889  0.888889
1   0.861111  0.861111  0.861111  0.861111
2   0.916667  0.916667  0.916667  0.916667
3   0.888889  0.888889  0.888889  0.888889
4   0.833333  0.833333  0.833333  0.833333
```

The best model among the 5 folds is determined using the accuracy, higher the accuracy, better is the model. Again, F1 Score/Accuracy is first reported on the validation set and testing is done on the 10% test dataset

Logistic Regression w/o penalty

In [10]:

```
reg=LogisticRegression(penalty='none',max_iter=1000)
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)
# y1=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

y = train_['label']
new_train = train_.drop(['label'], axis=1)
# print(train_.shape)
# print(y.shape)
stats1=[]
best_accuracy=0
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    ac=accuracy_score(reg.predict(x_test), y_test)

    stats1.append(list(precision_recall_fscore_support(reg.predict(x_test), y_test, aver
age='micro')))
    stats1[-1].append(ac)
    if(ac>best_accuracy):
        best_model_nopenalty=reg
        best_accuracy=ac

# y=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
# from sklearn.metrics import precision_recall_fscore_support
stats1=pd.DataFrame(stats1)
stats1.columns =['Precision', 'Recall', 'F1-Score', 'Support','Accuracy']
stats1=stats1.drop(['Support'],axis=1)
print("Logistic Regression Stats without Penalty")
print(stats1)
```

```
Logistic Regression Stats without Penalty
   Precision    Recall  F1-Score  Accuracy
0   0.841270  0.841270  0.841270  0.841270
1   0.873016  0.873016  0.873016  0.873016
2   0.888889  0.888889  0.888889  0.888889
3   0.936508  0.936508  0.936508  0.936508
4   0.888889  0.888889  0.888889  0.888889
```

In [11]:

```
y=test['label']
y_numpy=np.array(y)
new_test=test.drop(['label'],axis=1)
new_test=np.array(new_test)
ac=accuracy_score(best_model_nopenalty.predict(new_test), y)
fscore=precision_recall_fscore_support(best_model_nopenalty.predict(new_test), y, averag
e='micro')
print("Testing Best Model with no Penalty")
print("Accuracy=",ac)
print("Precision=",fscore[0])
print("Recall=",fscore[1])
print("F1-Score=",fscore[2])
```

```
Testing Best Model with no Penalty
Accuracy= 0.8611111111111112
Precision= 0.8611111111111112
Recall= 0.8611111111111112
F1-Score= 0.8611111111111112
```

Logistic Regression with L1 penalty

In [12]:

```
reg=LogisticRegression(penalty='l1',max_iter=1000,solver='liblinear')
kf=model_selection.KFold(n_splits=5)
```

```
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)
# y1=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

y = train_['label']
new_train = train_.drop(['label'], axis=1)
# print(train_.shape)
# print(y.shape)
stats1=[]
best_accuracy=0
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    ac=accuracy_score(reg.predict(x_test), y_test)

    stats1.append(list(precision_recall_fscore_support(reg.predict(x_test), y_test, aver
age='micro')))
    stats1[-1].append(ac)
    if(ac>best_accuracy):
        best_model_l1=reg
        best_accuracy=ac

# y=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
# from sklearn.metrics import precision_recall_fscore_support
stats1=pd.DataFrame(stats1)
stats1.columns =['Precision', 'Recall', 'F1-Score', 'Support','Accuracy']
stats1=stats1.drop(['Support'],axis=1)
print("Logistic Regression Stats L1")
print(stats1)
```

```
Logistic Regression Stats L1
   Precision    Recall  F1-Score  Accuracy
0   0.873016  0.873016  0.873016  0.873016
1   0.857143  0.857143  0.857143  0.857143
2   0.888889  0.888889  0.888889  0.888889
3   0.904762  0.904762  0.904762  0.904762
4   0.841270  0.841270  0.841270  0.841270
```

In [13]:

```
y=test['label']
y_numpy=np.array(y)
new_test=test.drop(['label'],axis=1)
new_test=np.array(new_test)
ac=accuracy_score(best_model_l1.predict(new_test), y)
fscore=precision_recall_fscore_support(best_model_l1.predict(new_test), y, average='micr
o')
print("Testing Best Model with L1 Penalty")
print("Accuracy=",ac)
print("Precision=",fscore[0])
print("Recall=",fscore[1])
print("F1-Score=",fscore[2])
```

```
Testing Best Model with L1 Penalty
Accuracy= 0.9166666666666666
Precision= 0.9166666666666666
Recall= 0.9166666666666666
F1-Score= 0.9166666666666666
```

Logistic Regression with L2

In [14]:

```
reg=LogisticRegression(penalty='l2',max_iter=1000,solver='liblinear')
kf=model_selection.KFold(n_splits=5)
# train=kf.get_n_splits(train_)
# print(train_.shape)
# print(y.shape)
# y1=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

y = train_['label']
new_train = train_.drop(['label'], axis=1)
# print(train_.shape)
# print(y.shape)
stats1=[]
best_accuracy=0
for train_index, test_index in kf.split(train_):
    rmse=[]

#     print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = new_train.iloc[train_index], new_train.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#     print(y_test.shape)
    reg.fit(x_train,y_train)
    ac=accuracy_score(reg.predict(x_test), y_test)

    stats1.append(list(precision_recall_fscore_support(reg.predict(x_test), y_test, aver
age='micro')))
    stats1[-1].append(ac)
    if(ac>best_accuracy):
        best_model_l2=reg
        best_accuracy=ac
#         print("YAY")

# y=test['label']
# new_test=test.drop(['label'],axis=1)
# new_test=np.array(new_test)
# from sklearn.metrics import precision_recall_fscore_support
stats1=pd.DataFrame(stats1)
stats1.columns =['Precision', 'Recall', 'F1-Score', 'Support','Accuracy']
stats1=stats1.drop(['Support'],axis=1)
print("Logistic Regression Stats L2")
print(stats1)
```

```
Logistic Regression Stats L2
   Precision    Recall  F1-Score  Accuracy
0   0.841270  0.841270  0.841270  0.841270
1   0.841270  0.841270  0.841270  0.841270
2   0.873016  0.873016  0.873016  0.873016
3   0.888889  0.888889  0.888889  0.888889
4   0.873016  0.873016  0.873016  0.873016
```

In [15]:

```
y=test['label']
y_numpy=np.array(y)
new_test=test.drop(['label'],axis=1)
new_test=np.array(new_test)
ac=accuracy_score(best_model_l2.predict(new_test), y)
fscore=precision_recall_fscore_support(best_model_l2.predict(new_test), y, average='micr
o')
print("Testing Best Model with L2 Penalty")
print("Accuracy=",ac)
print("Precision=",fscore[0])
print("Recall=",fscore[1])
print("F1-Score=",fscore[2])
```

```
Testing Best Model with L2 Penalty
Accuracy= 0.9166666666666666
Precision= 0.9166666666666666
Recall= 0.9166666666666666
F1-Score= 0.9166666666666666
```

Introduction of L1 and L2 penalty has improved the accuracy/F1 Score.

In [16]:

```
print(best_model_nopenalty)
print(best_model_l1)
print(best_model_l2)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

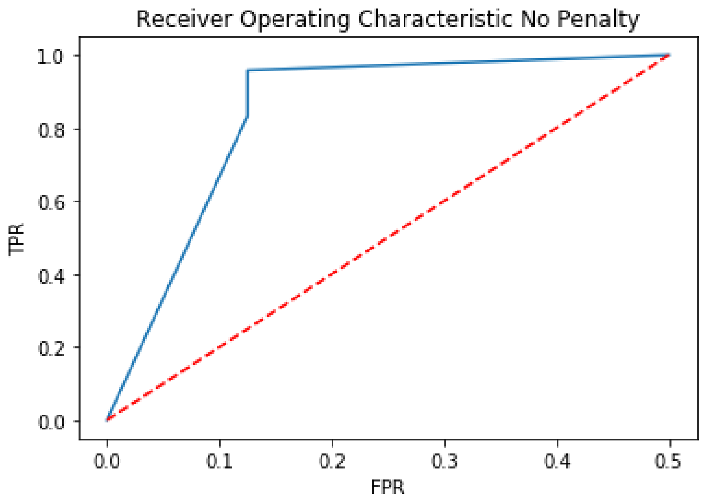
Plotting ROC-AUC curve for the three best models

In [17]:

```
y=test['label']
y_numpy=np.array(y)
new_test=test.drop(['label'],axis=1)
new_test=np.array(new_test)
probab_l2=best_model_l2.predict_proba(new_test)
probab_l1=best_model_l1.predict_proba(new_test)
probab_no=best_model_nopenalty.predict_proba(new_test)

#second column is 1 first column is 0
curve_no_x=[]
curve_no_y=[]
curve_l1_x=[]
curve_l1_y=[]
curve_l2_x=[]
curve_l2_y=[]
# print(y)
for i in range (0,110,10):
    threshold=i/100
    predicted=[]
#     print(threshold)
    for j in probab_no:
        if(j[1]>threshold):
            predicted.append(1)
        else:
            predicted.append(0)
#     print(predicted)
    tpr=0
    fpr=0
    for j in range(len(predicted)):
        if(predicted[j]==y_numpy[j] and y_numpy[j]==1):
            tpr+=1
        if(predicted[j]!=y_numpy[j] and predicted[j]==1):
            fpr+=1
    tpr=tpr/sum(y_numpy)
    fpr=fpr/sum(y_numpy)
    curve_no_x.append(fpr)
    curve_no_y.append(tpr)

plt.plot(curve_no_x,curve_no_y)
plt.plot([0, max(curve_no_x)], [0, max(curve_no_y)],'r--')
plt.title('Receiver Operating Characteristic No Penalty')
plt.ylabel('TPR')
plt.xlabel('FPR')
```

Out[17]:

```
Text(0.5, 0, 'FPR')
```

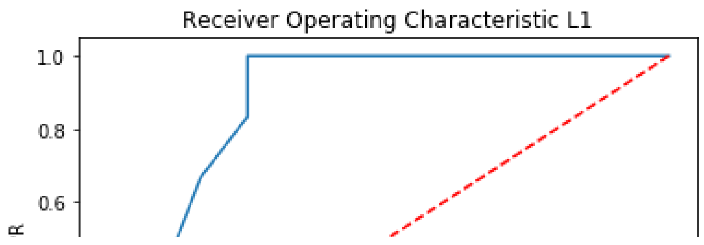


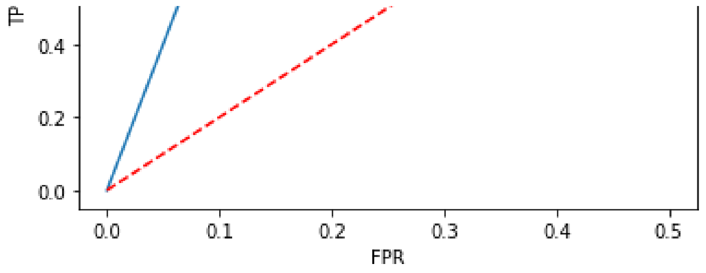
In [18]:

```
for i in range (0,110,10):
    threshold=i/100
    predicted=[]
#     print(threshold)
    for j in probab_l1:
        if(j[1]>threshold):
            predicted.append(1)
        else:
            predicted.append(0)
#     print(predicted)
    tpr=0
    fpr=0
    for j in range(len(predicted)):
        if(predicted[j]==y_numpy[j] and predicted[j]==1):
            tpr+=1
        if(predicted[j]!=y_numpy[j] and predicted[j]==1):
            fpr+=1
    tpr=tpr/sum(y_numpy)
    fpr=fpr/sum(y_numpy)
    curve_l1_x.append(fpr)
    curve_l1_y.append(tpr)

plt.plot(curve_l1_x,curve_l1_y)
plt.plot([0, max(curve_l1_x)], [0, max(curve_l1_y)],'r--')
plt.title('Receiver Operating Characteristic L1')
plt.ylabel('TPR')
plt.xlabel('FPR')
```

Out[18]:

```
Text(0.5, 0, 'FPR')
```



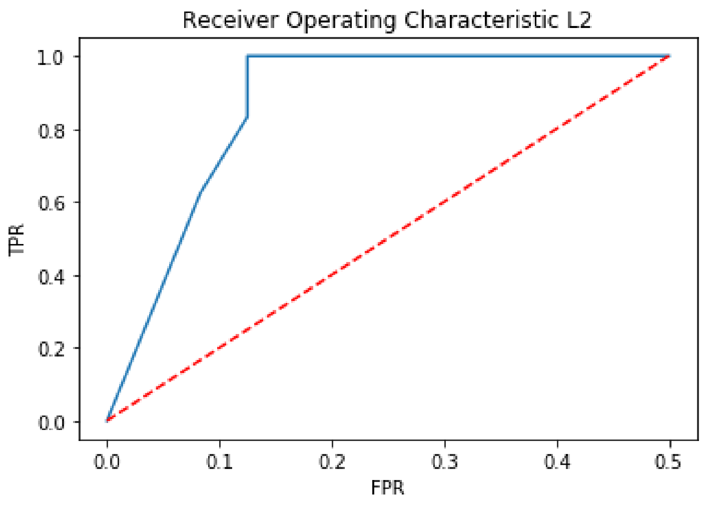


In [19]:

```
for i in range (0,110,10):
    threshold=i/100
    predicted=[]
#     print(threshold)
    for j in probab_l2:
        if(j[1]>threshold):
            predicted.append(1)
        else:
            predicted.append(0)
#     print(predicted)
    tpr=0
    fpr=0
    for j in range(len(predicted)):
        if(predicted[j]==y_numpy[j] and y_numpy[j]==1):
            tpr+=1
        if(predicted[j]!=y_numpy[j] and predicted[j]==1):
            fpr+=1
    tpr=tpr/sum(y_numpy)
    fpr=fpr/sum(y_numpy)
    curve_l2_x.append(fpr)
    curve_l2_y.append(tpr)

plt.plot(curve_l2_x,curve_l2_y)
plt.plot([0, max(curve_l2_x)], [0, max(curve_l2_y)],'r--')
plt.title('Receiver Operating Characteristic L2')
plt.ylabel('TPR')
plt.xlabel('FPR')
```

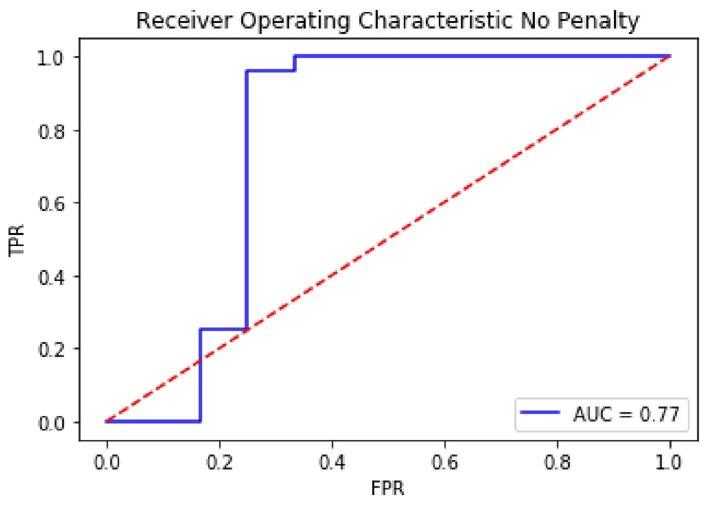
Out[19]:

```
Text(0.5, 0, 'FPR')
```



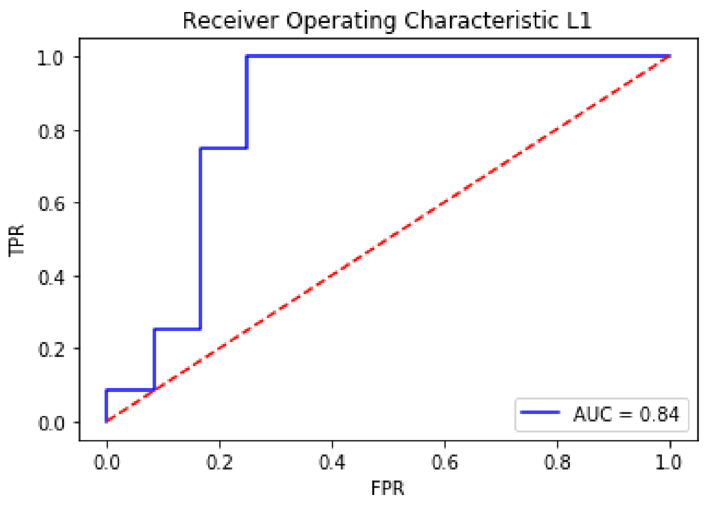
In [20]:

```
import sklearn.metrics as metrics
preds = probab_no[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_numpy, preds)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic No Penalty')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```



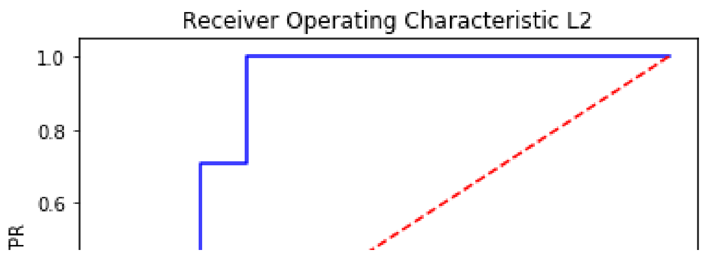
In [21]:

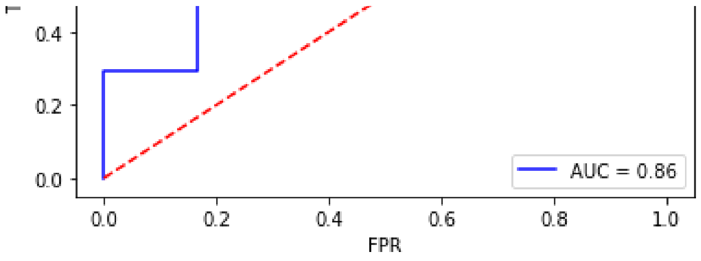
```
preds = probab_l1[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_numpy, preds)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic L1')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```



In [22]:

```
preds = probab_l2[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_numpy, preds)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic L2')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```





The difference between the inbuilt and implemented versions of the ROC-AUC curve is the resolution. I have implemented the curve for only 11 points of threshold, which is much lesser than the inbuilt version. Increasing the resolution results in the plots being similar.

In [1]:

```
import numpy as np
import idx2numpy
import random
from matplotlib import pyplot as plt
import cv2
import numpy as np
from sklearn.multiclass import OneVsOneClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
train_images = idx2numpy.convert_from_file('train-images.idx3-ubyte')
train_labels = idx2numpy.convert_from_file('train-labels.idx1-ubyte')
test_images = idx2numpy.convert_from_file('t10k-images.idx3-ubyte')
test_labels = idx2numpy.convert_from_file('t10k-labels.idx1-ubyte')
train_new=[]
test_new=[]
for i in train_images:
    i=np.array(i)
    train_new.append(i.flatten())
for i in test_images:
    i=np.array(i)
    test_new.append(i.flatten())
train_images=train_new
test_images=test_new
```

In [3]:

```
clf = OneVsOneClassifier(LogisticRegression(random_state=0,max_iter=1000)).fit(train_imag
es, train_labels)
# clf = OneVsRestClassifier(SVC()).fit(train_images, train_labels)
```

```
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [6]:

```
ac=accuracy_score(clf.predict(test_images),test_labels)
print("Accuracy=",ac)
fscore=precision_recall_fscore_support(clf.predict(test_images), test_labels, average='m
icro')
print("Precision=",fscore[0])
print("Recall=",fscore[1])
print("F1-Score=",fscore[2])
```

```
Accuracy= 0.9249
Precision= 0.9249
Recall= 0.9249
F1-Score= 0.9249
```

In [1]:

```
import numpy as np
import idx2numpy
import random
from matplotlib import pyplot as plt
import cv2
import numpy as np
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

In [2]:

```
train_images = idx2numpy.convert_from_file('train-images.idx3-ubyte')
train_labels = idx2numpy.convert_from_file('train-labels.idx1-ubyte')
test_images = idx2numpy.convert_from_file('t10k-images.idx3-ubyte')
test_labels = idx2numpy.convert_from_file('t10k-labels.idx1-ubyte')
train_new=[]
test_new=[]
for i in train_images:
    i=np.array(i)
    train_new.append(i.flatten())
for i in test_images:
    i=np.array(i)
    test_new.append(i.flatten())
train_images=train_new
test_images=test_new
```

In [3]:

```
clf = OneVsRestClassifier(LogisticRegression(random_state=0,max_iter=1000)).fit(train_ima
ges, train_labels)
# clf = OneVsRestClassifier(SVC()).fit(train_images, train_labels)
```

```
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
F:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [6]:

```
ac=accuracy_score(clf.predict(test_images),test_labels)
print("Accuracy=",ac)
fscore=precision_recall_fscore_support(clf.predict(test_images), test_labels, average='m
icro')
print("Precision=",fscore[0])
print("Recall=",fscore[1])
print("F1-Score=",fscore[2])
```

```
Accuracy= 0.9168
Precision= 0.9168
```

```
Recall= 0.9168
F1-Score= 0.9168
```

Ayush Madhan Sohini

2019156

ML - Assignment 1

Q1 $Y = X\theta + \epsilon \rightarrow$ Loss $= \epsilon = Y - X\theta$

$Y \rightarrow$ Vector of predicted values
$X \rightarrow$ Vector of i/p features
$\epsilon \rightarrow$ error value

The loss function needed to be minimized $\Rightarrow$

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(y(x^i) - x^i\theta \right)^2 \rightarrow \text{Mean Squared Loss}$$

$x_i \in R^d \rightarrow$ ith sample from data set of size N

In vector form $\rightarrow$

$$L(\theta) = \frac{1}{2N} (\underset{\substack{\downarrow \\ \text{predicted}}}{Y} - X\theta)^2 \quad \hookrightarrow \text{label vector}$$

$$= \frac{1}{2N} (Y - X\theta)^T (Y - X\theta)$$

$$= \frac{1}{2N} \left(Y^T - (X\theta)^T \right) (Y - X\theta)$$

$$= \frac{1}{2N}(Y^T - \theta^T X^T)(Y - X\theta)$$

$$= \frac{1}{2N}(Y^T Y - Y^T X\theta - \theta^T X^T Y + \theta^T X^T X\theta)$$

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{1}{2N}\left[-X^T Y - X^T Y + 2X^T X\theta\right]$$

$$= \frac{1}{2N}\left[2X^T X\theta - 2X^T Y\right]$$

To move in the direction of the optimal solution, we equate the derivative to zero

$$\frac{1}{2N}\left[2X^T X\theta - 2X^T Y\right] = 0$$

$$\therefore X^T X\theta = X^T Y$$

$$\theta = (X^T X)^{-1} X^T Y$$

Q2 Whenever the inverse of $X^T X$ exists, the closed form solution exists. If $X^T X$ is a singular matrix, the closed form solution won't exist.

Q3 The closed form solution is a better option ONLY when the size of the i/p matrix X is small or X is sparse. When X is a very large (suppose it has 10^5 entries), $X^T X$ would be a $10^5 \times 10^5$ matrix ie it has 10^{10} entries, which would be very difficult to store. Also performing $(X^T X)^{-1}$ is also computationally inefficient on such a large matrix. Also if $X^T X$ is singular, the inverse doesn't exist anyways. In such cases, the iterative methods like gradient descent are a better choice.

Q4

$$y(x^i) = \theta_0 + x_1^i \theta_1 + x_2^i \theta_2 \dots x_n^i \theta_n + \epsilon^i$$

$$y(x^1) = \theta_0 + x_1^1 \theta_1 + x_2^1 \theta_2 \dots x_n^1 \theta_n + \epsilon^1$$

$$\vdots$$

$$y(x^m) = \theta_0 + x_1^m \theta_1 \dots\dots x_n^m \theta_n + \epsilon^m$$

$$y(x^1) + \dots y(x^m) = m\theta_0 + \theta_1 \sum_{i=1} x_1^i$$

$$+ \theta_2 \sum_{i=1}^{m} x_2^i \dots,$$

$$+ \theta_n \sum_{i=1}^{m} x_n^i + \sum_{i=1}^{m} \epsilon^i$$

Dividing both sides by m

$$\frac{y(x^1) + \dots y(x^m)}{m} = \theta_0 + \frac{\theta_1 \sum_{i=1}^{m} x_1^i}{m}$$

$$\dots + \frac{\theta_n \sum_{i=1}^{m} x_n^i}{m} + \boxed{\frac{\sum_{i=1}^{m} \epsilon^i}{m}}$$

$= 0$ as ϵ follows a zero mean gaussian

$$\therefore \bar{Y} = \theta \bar{X} + \theta_0 \rightarrow \text{eqn of a line}$$

$$\text{where } \bar{X} = \begin{bmatrix} \sum_{i=1}^{m} x_1^i \\ \vdots \\ \sum_{i=1}^{m} x_n^i \end{bmatrix} \times \frac{1}{m}$$

$$\Theta = [\Theta_1 \ \Theta_2 \dots \Theta_n]$$

$$\bar{Y} = \begin{bmatrix} y(x^1) \\ \vdots \\ y(x^m) \end{bmatrix} \times \frac{1}{m}$$

Q5 Linear Regression model throws a continuous output. So if we can set a particular threshold ie if the o/p of the linear regressor is above a threshold then it belongs to class A else class B, it can work as a binary classifier. The threshold values can be determined using tools like the ROC-AUC curve. However the data is rarely distributed as a gaussian, so this is not a good classifier as in linear regression the error in the data is assumed to be a gaussian.