qda-lda

November 14, 2017

1 1. Data Preparation

```
In [234]: import numpy as np
          import sklearn as sk
          import matplotlib.pyplot as plt
          from sklearn.datasets import load_digits
          %matplotlib inline
          plt.rcParams['figure.figsize'] = (10, 6)
```

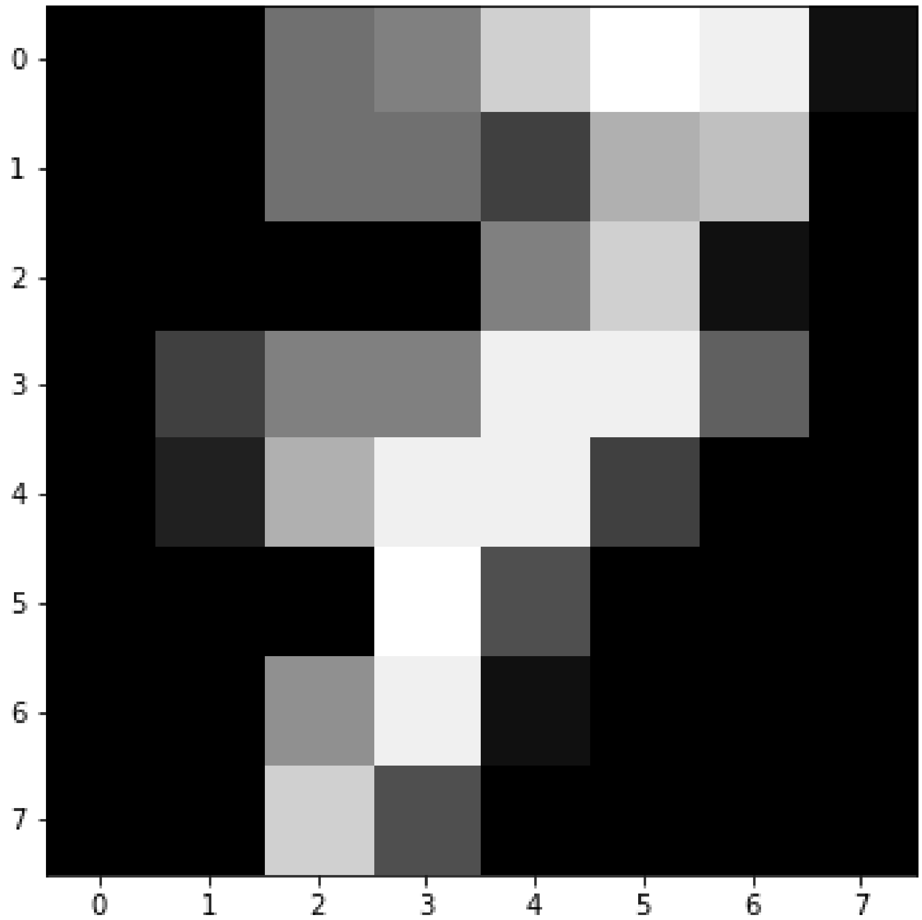
```
In [235]: digits=load_digits()
          print(digits.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

```
In [236]: data=digits['data']
          images=digits['images']
          target=digits['target']
          target_names=digits['target_names']
```

```
In [237]: img = images[(np.where(target == 7)[0][0])]
          print(img.shape)
          plt.figure()
          plt.gray()
          plt.imshow(img,interpolation="nearest");
```

```
(8, 8)
```



In [238]:
```
def filter_numbers(number1,number2):
    """Returns subset of digits sample data containing only the numbers passed to the

       Takes number1 and number2 of the target_names as its arguments
       Returns two arrays: data, target
    """
    #if number1 and number2:
    if True:
        indices = np.where((digits['target'] == number1) | (digits['target'] == number
        #print(indices)
        data = (digits['data'])[indices]
        target = (digits['target'])[indices]
        #print(indices)
    return data, target
```

In [239]:
```
data17,target17=filter_numbers(1,7)
np.shape(data17)
```

Out[239]: (361, 64)

Split data in train and test set

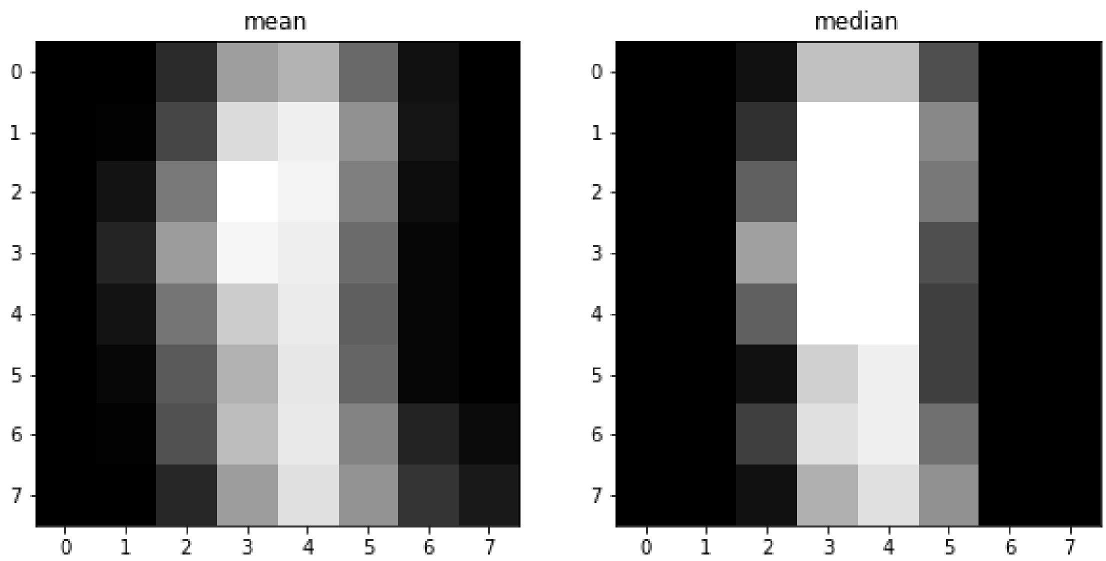
In [240]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = \
train_test_split(data17,target17, test_size=0.2)
```

Plot the "mean" picture of both numbers. Pixel [19] is nearly black for number 7 and white for number 1. Also pixel [61] is black for number 7 and grey for number 1.

In [241]:
```
def mean_plot(number):
    img = np.mean([images[j] for j in range(len(images)) if target[j]==number], axis =
    return img

def median_plot(number):
    img = np.median([images[j] for j in range(len(images)) if target[j]==number], axis
    return img
```

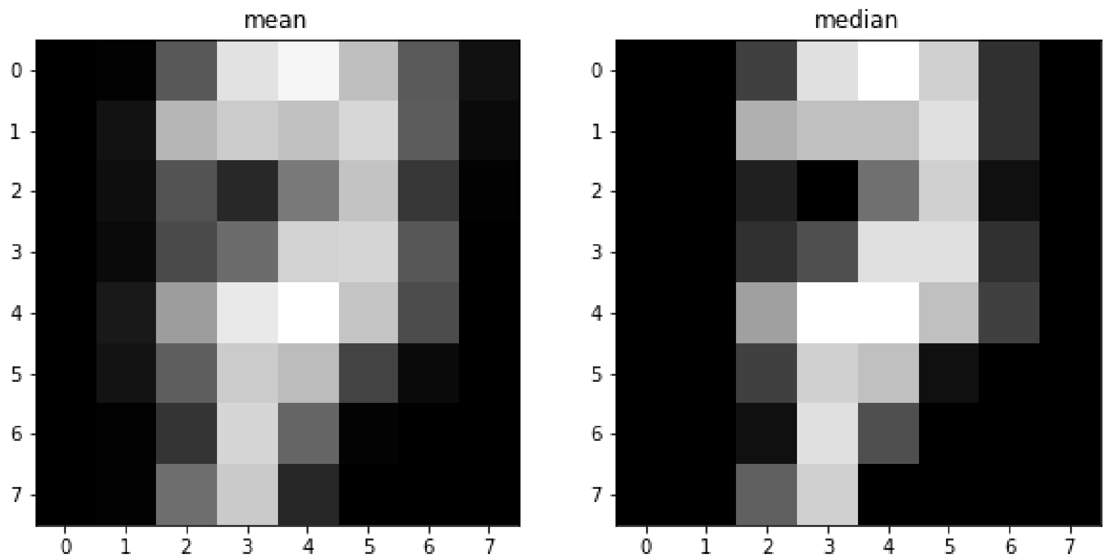
In [242]:
```
fig,ax=plt.subplots(1,2)
plt.gray()
ax[0].imshow(mean_plot(1))
ax[0].set_title("mean")
ax[1].imshow(median_plot(1))
ax[1].set_title("median")
```

Out[242]: Text(0.5,1,'median')



In [243]:
```
fig,ax=plt.subplots(1,2)
plt.gray()
ax[0].imshow(mean_plot(7))
ax[0].set_title("mean")
ax[1].imshow(median_plot(7))
ax[1].set_title("median")
```

Out[243]:
```
Text(0.5,1,'median')
```



In [244]:
```
print("Max difference mean: ",np.argmax(np.abs(mean_plot(7)-mean_plot(1))))
print("Min difference mean: ",np.argmin(np.abs(mean_plot(7)-mean_plot(1))))
print("Max difference median: ",np.argmax(np.abs(median_plot(7)-median_plot(1))))
print("Min difference median: ",np.argmin(np.abs(median_plot(7)-median_plot(1))))
print(np.abs(median_plot(7)-median_plot(1))) ## find some nice pixels
```

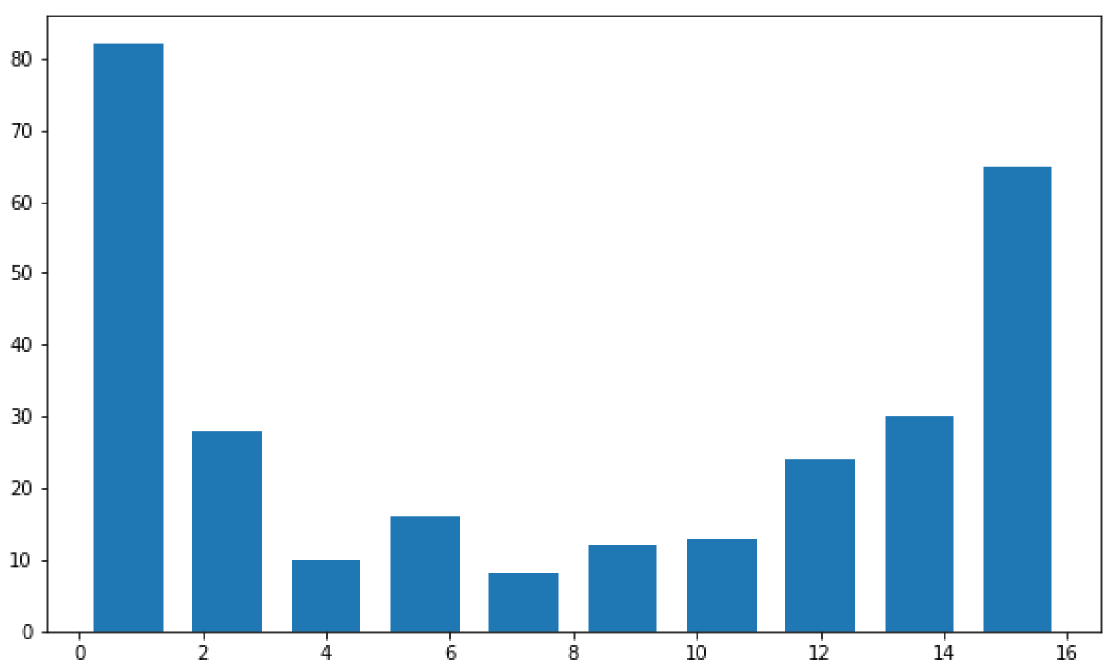
```
Max difference mean:  19
Min difference mean:  0
Max difference median:  19
Min difference median:  0
[[ 0.    0.    3.    2.    4.    8.    3.    0. ]
 [ 0.    0.    8.    4.    4.    5.5   3.    0. ]
 [ 0.    0.    4.   16.    9.    5.5   1.    0. ]
 [ 0.    0.    7.   11.    2.    9.    3.    0. ]
 [ 0.    0.    4.    0.    0.    8.    4.    0. ]
 [ 0.    0.    3.    0.    3.    3.    0.    0. ]
 [ 0.    0.    3.    0.   10.    7.    0.    0. ]
 [ 0.    0.    5.    2.   14.    9.    0.    0. ]]
```

1.0.1 Dimension Reduction

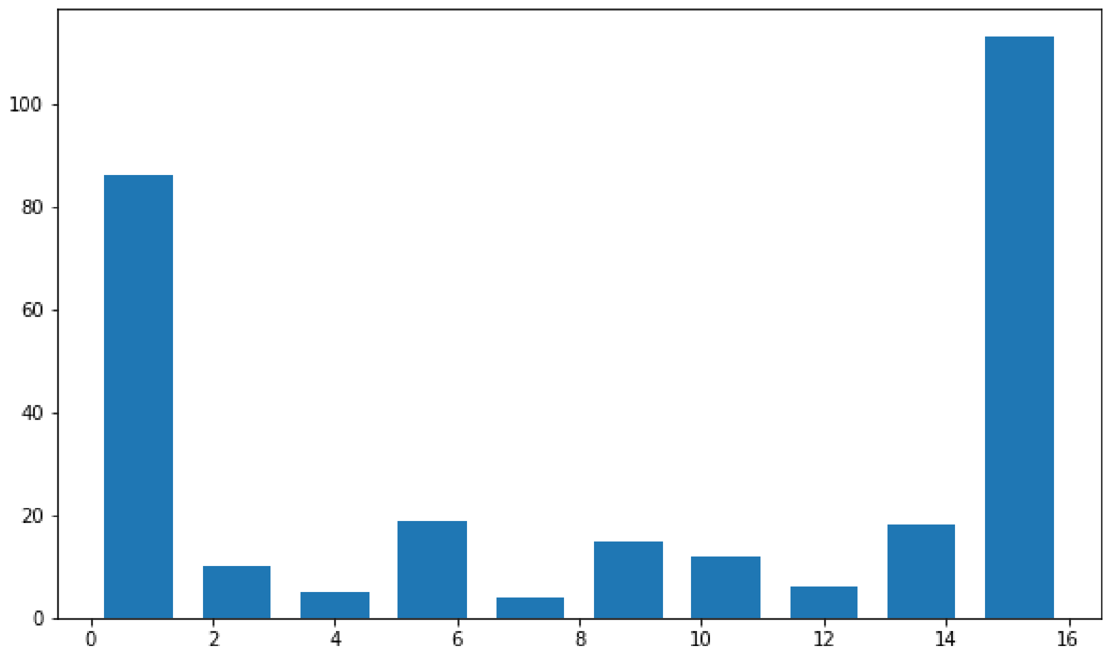
```
In [245]: def reduced_dim(x):
              res=np.empty([np.shape(x)[0],2])
              for i in range(len(x)):
                  # Define the features here:

                  res[i][0]=x[i][60]# Feature 1
                  res[i][1]=x[i][19]          # Feature 2

              return res
```

```
In [246]: reduced_x=reduced_dim(X_train)
          print(np.histogram(reduced_dim(X_train)[:,0]))
          #plt.plot(np.histogram(reduced_dim(X_train)[:,0]))
          hist, bins = np.histogram(reduced_dim(X_train)[:,0])
          width = 0.7 * (bins[1] - bins[0])
          center = (bins[:-1] + bins[1:]) / 2
          plt.bar(center, hist, align='center', width=width)
          plt.show()
          hist, bins = np.histogram(reduced_dim(X_train)[:,1])
          width = 0.7 * (bins[1] - bins[0])
          center = (bins[:-1] + bins[1:]) / 2
          plt.bar(center, hist, align='center', width=width)
```

```
(array([82, 28, 10, 16,  8, 12, 13, 24, 30, 65]), array([ 0. ,   1.6,   3.2,   4.8,   6.4,   8.
        14.4,  16. ]))
```



Out[246]: <Container object of 10 artists>



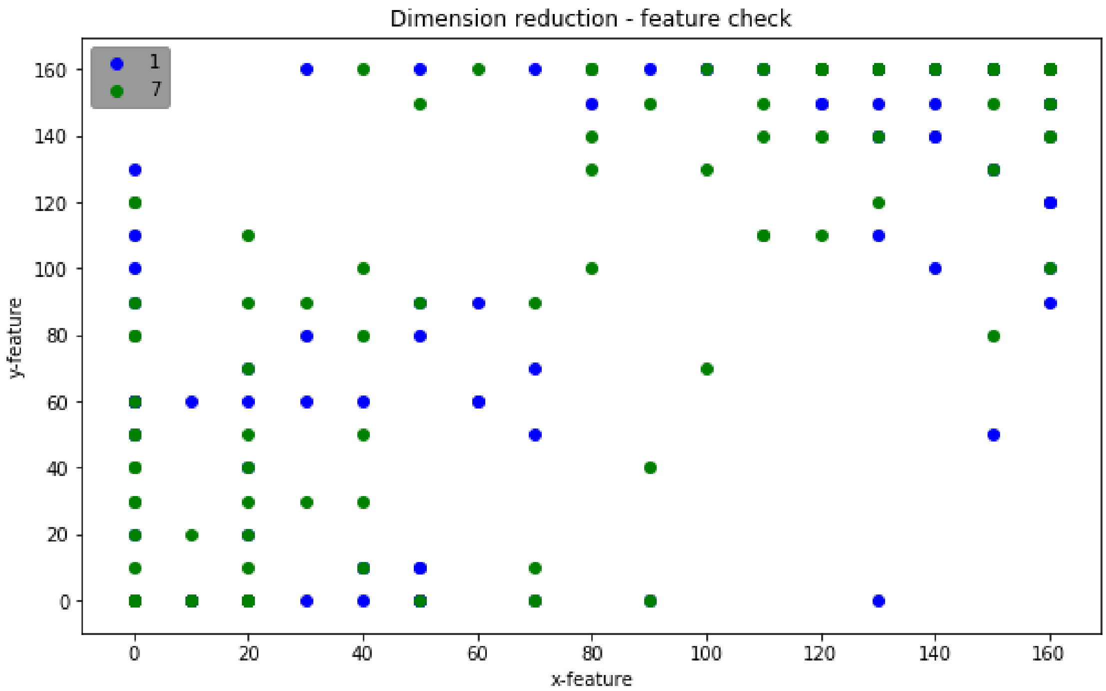
The histograms show that the selected features divide the features very nicely into two distinct classes

In [247]:
```
# To differentiate in the scatter plot between the two classes

reduced1=[reduced_x[i] for i in range(len(reduced_x)) if target17[i]==1]
reduced7=[reduced_x[i] for i in range(len(reduced_x)) if target17[i]==7]
```

In [248]:
```
x1=[reduced1[i][0]*10 for i in range(len(reduced1))]
#x1=x1/np.max(x1)
y1=[reduced1[i][1]*10 for i in range(len(reduced1))]
#y1=y1/np.max(y1)
x7=[reduced7[i][0]*10 for i in range(len(reduced7))]
#x7=x7/np.max(x7)
y7=[reduced7[i][1]*10 for i in range(len(reduced7))]
#y7=y7/np.max(y7)
plt.scatter(x1,y1,color='b',label='1')
plt.scatter(x7,y7,color='g',label='7')

legend = plt.legend(frameon = 1)
frame = legend.get_frame()
frame.set_color('grey')
```

```
        plt.xlabel("x-feature")
        plt.ylabel('y-feature')
        plt.title('Dimension reduction - feature check')
```

Out[248]: Text(0.5,1,'Dimension reduction - feature check')



2 2. Nearest Mean

In [249]:
```
def nearest_mean(training_features, training_labels, test_features):
    mean1=np.empty([2,1])
    mean7=np.empty([2,1])
    mean1=sum([training_features[i] for i in range(\
        len(training_features)) if training_labels[i]==1])
    mean1=mean1/np.count_nonzero(training_labels==1)
    mean7=sum([training_features[i] for i in range(\
        len(training_features)) if training_labels[i]==7])
    mean7=mean7/np.count_nonzero(training_labels==7)
    predicted_labels=np.empty([len(test_features),1])
    for i in range(len(predicted_labels)):
        dis1=np.linalg.norm(mean1-test_features[i])
        dis2=np.linalg.norm(mean7-test_features[i])
        if dis1 <= dis2:
            predicted_labels[i]=1
```

```
            else:
                predicted_labels[i]=7
        return predicted_labels
```

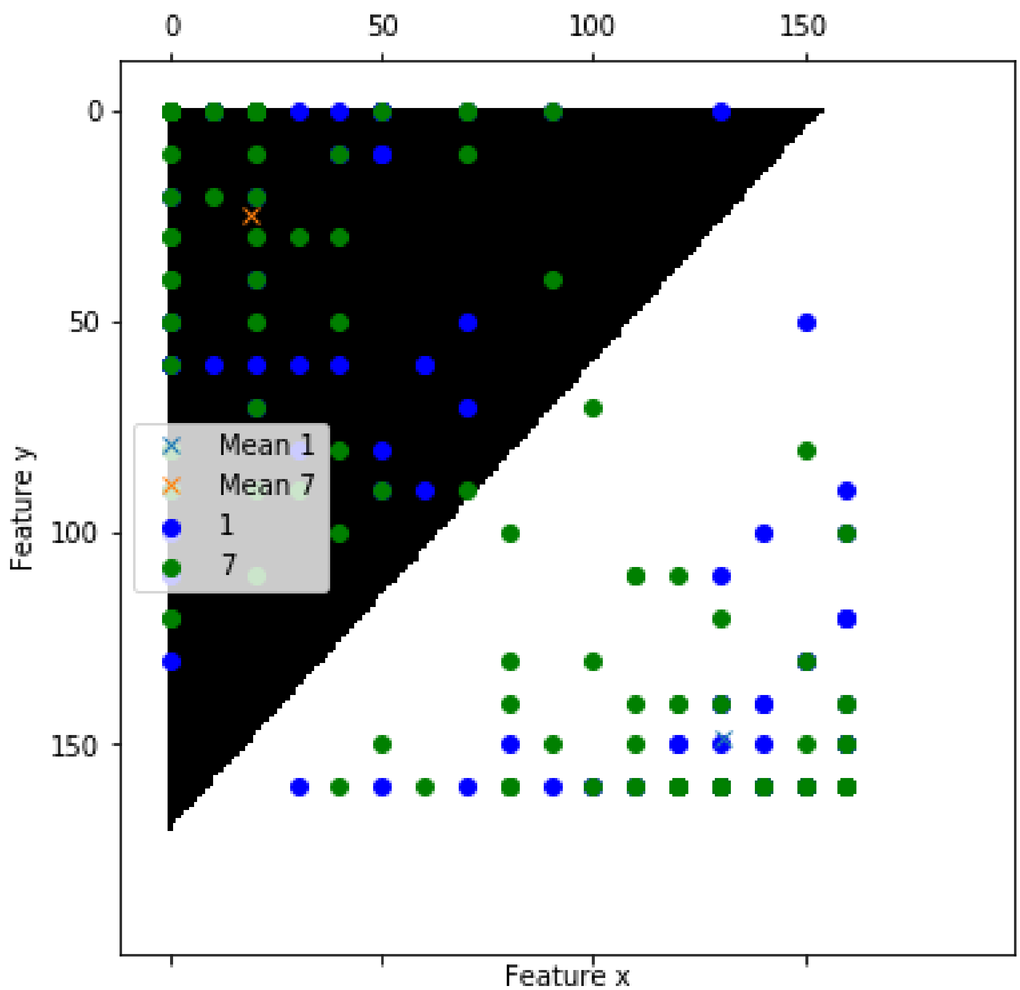
In [250]:
```
reduced_test=reduced_dim(X_test)
predicted_labels=nearest_mean(reduced_x,y_train,reduced_test)
counter=0
for x,y in zip(predicted_labels,y_test):
    if x==y:
        counter+=1
print("Elements classified correctly: ",counter," (in Total: ",len(y_test)," elements.
```

```
Elements classified correctly:  72  (in Total:  73  elements.)
```

In [251]:
```
training_features=reduced_x
training_labels=y_train

mean1=np.empty([2,1])
mean7=np.empty([2,1])
mean1 = np.mean([training_features[j] for j in range(len\
        (training_features)) if training_labels[j]==1], axis = 0)

mean7 = np.mean([training_features[j] for j in range(len\
        (training_features)) if training_labels[j]==7], axis = 0)


#mean1=sum([training_features[i] for i in range(\
#         len(training_features)) if training_labels[i]==1])
#mean1=mean1/np.count_nonzero(training_labels==1)
#mean7=sum([training_features[i] for i in range(\
#         len(training_features)) if training_labels[i]==7])
#mean7=mean7/np.count_nonzero(training_labels==7)


a=0
matrix=np.empty([200,200])
for i in range(200):
    for j in range(200):
        i_=i/10
        j_=j/10
        dis1=np.linalg.norm(mean1-np.array([i_,j_]))
        dis2=np.linalg.norm(mean7-np.array([i_,j_]))
        if dis1>dis2:
            a=0
        else:
            a=1
        matrix[i,j]=a
```

```
In [252]: plt.matshow(matrix);
          plt.plot(mean1[0]*10,mean1[1]*10,'x',label="Mean 1")
          plt.plot(mean7[0]*10,mean7[1]*10,'x',label="Mean 7")

          plt.xlabel("Feature x")
          plt.ylabel("Feature y")
          plt.scatter(x1,y1,color='b',label='1')
          plt.scatter(x7,y7,color='g',label='7')
          plt.legend(loc=6)
```

```
Out[252]: <matplotlib.legend.Legend at 0x7fad169390b8>
```



2.1 QDA

```
In [253]: def fit_qda(training_features, training_labels):
              """
```

```
    Fits D dimensional Gaussian to the given training data.

    TRAINING LABELS: 0, 1

    Input N x D vector of training features, N x 1 vector of corresp. instances
    Returns 2 x D matrix of means of the 2 classes, 2 x D x D array of the covariances

    def vec_generator(matrix):
        """ Returns one vector at a time"""
        for i in range(len(matrix[:,0])):
            yield matrix[i,:]
    features_class_0 = training_features[np.where(training_labels == 0)]
    features_class_1 = training_features[np.where(training_labels == 1)]
    mu = np.array([np.mean(features_class_0,axis=0),np.mean(features_class_1,axis=0)])

    covmat = np.array([np.mean([np.outer(i,j) for i,j in zip(vec_generator(features_cl
                              ,np.mean([np.outer(i,j) for i,j in zip(vec_generator(fe

    p = np.array([len(features_class_0)/len(training_features), 1 - len(features_class
    return mu, covmat, p


def predict_qda(mu, covmat, p, test_features):
    """
    Predicts test instances of the given test_features given the parameter of the mult
    the feature distribution

    Input 2 x D matrix of means of the 2 classes, 2 x D x D array of the covariances,

    Returns M x 1 vector of class labels (0 and 1) """

    b = np.array(np.log(np.linalg.det(covmat)/p**2)) # precompute constant factor of l


    inv_covmat = np.linalg.inv(covmat) # inverse of covmat; for both classes simultane

    likelihood_class_0 = [np.inner(np.array(i-mu[0,:]),inv_covmat[0,:,:].dot(np.array(
    likelihood_class_1 = [np.inner(np.array(i-mu[1,:]),inv_covmat[1,:,:].dot(np.array(

    test_instances = np.argmin(np.array([likelihood_class_0,likelihood_class_1]),axis=

    return test_instances
```

In [254]:
```
### recycle filtered numbers; map labels 1 -> 0 , 7 -> 1
X_train, X_test, y_train, y_test = \
train_test_split(data17,target17, test_size=0.2)

y_train[np.where(y_train==1)],y_train[np.where(y_train==7)] = 0, 1
```

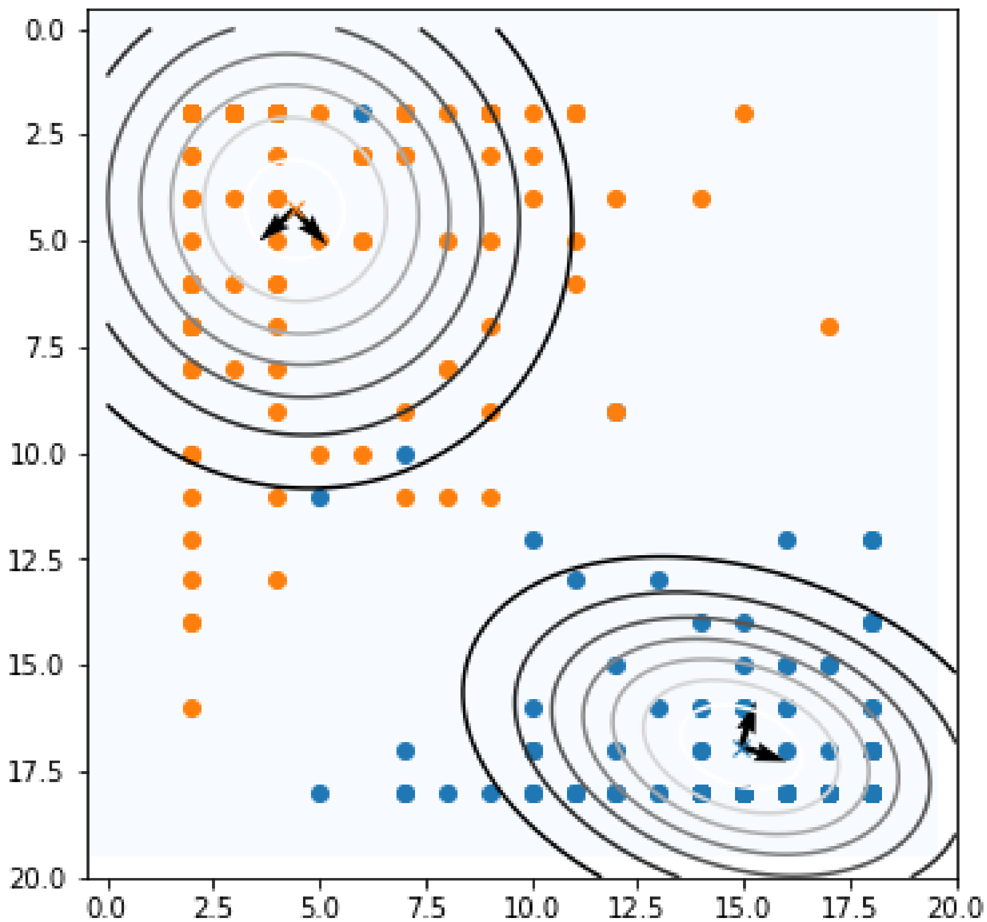
```
y_test[np.where(y_test==1)],y_test[np.where(y_test==7)] = 0, 1
reduced_x_train =reduced_dim(X_train)
reduced_x_test =reduced_dim(X_test)
[mu, covmat, p] = fit_qda(reduced_x_train,y_train)

print("Training data: %d out of %d elements correct" % (np.sum(predict_qda(mu,covmat,p
print("Test data: %d out of %d elements correct" % (np.sum(predict_qda(mu,covmat,p,red
```

```
Training data: 284 out of 288 elements correct
Test data: 72 out of 73 elements correct
```

2.2 Visualization

In [255]:
```
### adjust plots

transx = 2
transy = 2



x_train_class0 = reduced_x_train[np.where(y_train==0)] #### seperate classes
x_train_class1 = reduced_x_train[np.where(y_train==1)]


grid = np.zeros((20,20)) ## background
plt.imshow(grid,cmap="Blues",aspect=1)


plt.scatter(x_train_class0[:,0]+transx,x_train_class0[:,1]+transy, ) ## training data
plt.scatter(x_train_class1[:,0]+transx,x_train_class1[:,1]+transy)

plt.plot(mu[0,0]+transx,mu[0,1]+transy,'x') ## means
plt.plot(mu[1,0]+transx,mu[1,1]+transy,'x')



#### bivariat gaussians
x = np.linspace(0,20,100)
y = np.linspace(0,20,100)
x,y = np.meshgrid(x,y)
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x
pos[:, :, 1] = y
from scipy.stats import multivariate_normal
var = multivariate_normal(mean=mu[0]+[transx,transy], cov=covmat[0])
plt.contour(x,y,var.pdf(pos))
var = multivariate_normal(mean=mu[1]+[transx,transy], cov=covmat[1])
```

```
plt.contour(x,y,var.pdf(pos))


######## eigenvalue /vector decomposition

val, vec = np.linalg.eig(covmat[0])
for i,j in zip(val,vec):
    plt.quiver(mu[0,0]+transx,mu[0,1]+transy,np.sqrt(i)*j[0],np.sqrt(i)*j[1])


val, vec = np.linalg.eig(covmat[1])
for i,j in zip(val,vec):
    plt.quiver(mu[1,0]+transx,mu[1,1]+transy,np.sqrt(i)*j[0],np.sqrt(i)*j[1])
```



2.3 Performance evaluation

In [256]:
```
from sklearn.model_selection import KFold
kf=KFold(n_splits=10)
score=[]
for train_index, test_index in kf.split(data_red):
    #print("TRAIN:", train_index, "TEST:", test_index)

    X_train, X_test = data_red[train_index], data_red[test_index]
    y_train, y_test = target17[train_index], target17[test_index]

    y_train[np.where(y_train==1)],y_train[np.where(y_train==7)] = 0, 1
    y_test[np.where(y_test==1)],y_test[np.where(y_test==7)] = 0, 1

    mu, covmat, p = fit_qda(X_train,y_train)

    #print("Training data: %d out of %d elements correct" % (np.sum(predict_qda(mu,cov
    correct=np.sum(predict_qda(mu,covmat,p,X_test)==y_test)
    print("Test data: %d out of %d elements correct" % (correct,len(y_test)))
    score.append(correct/len(y_test))
print('Average rate of prediction: ',np.mean(score),'+-',np.std(score))
```

```
Test data: 36 out of 37 elements correct
Test data: 32 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 35 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 35 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 36 out of 36 elements correct
Average rate of prediction:  0.980630630631 +- 0.0329660801145
```

3 LDA

In [257]:
```
def fit_lda(training_features, training_labels):
    """
    Fits D dimensional Gaussian to the given training data.

    TRAINING LABELS: 0, 1

    Input N x D vector of training features, N x 1 vector of corresp. instances
    Returns 1 x D matrix of means of the 2 classes, 1 x D x D array of the covariances

    def vec_generator(matrix):
        """ Returns one vector at a time"""
```

```
        for i in range(len(matrix[:,0])):
            yield matrix[i,:]
    features_class_0 = training_features[np.where(training_labels == 0)]
    features_class_1 = training_features[np.where(training_labels == 1)]
    mu = np.array([np.mean(features_class_0,axis=0),np.mean(features_class_1,axis=0)])


    ### assume same covariance matrix for bot classes. Hence calculate them and take t
    covmat_feature1 = np.mean([np.outer(i,j) for i,j in zip(vec_generator(features_cla
    covmat_feature0 = np.mean([np.outer(i,j) for i,j in zip(vec_generator(features_cla

    covmat = np.array((covmat_feature0+covmat_feature1)/2) ## mean covariance matrix

    return mu, covmat




def predict_lda(mu, covmat, test_features):
    """
    Predicts test instances of the given test_features given the parameter of the mult
    the feature distribution

    Input 2 x D matrix of means of the 2 classes, 2 x D x D array of the covariances,

    Returns M x 1 vector of class labels (0 and 1) """

    b = np.array(np.log(np.linalg.det(covmat)/p**2)) # precompute constant factor of l


    inv_covmat = np.linalg.inv(covmat) # inverse of covmat; for both classes simultane

    w = 2*np.inner((mu[1,:]-mu[0,:]),inv_covmat) # w_1 - w_0
    b_ = -np.inner(mu[1,:],inv_covmat.dot(mu[1,:]))-b[1]+np.inner(mu[0,:],inv_covmat.d

    test_instances = ([0 if np.inner(w,test_features[i,:])+b_ < 0 else 1 for i in rang
    return test_instances
```

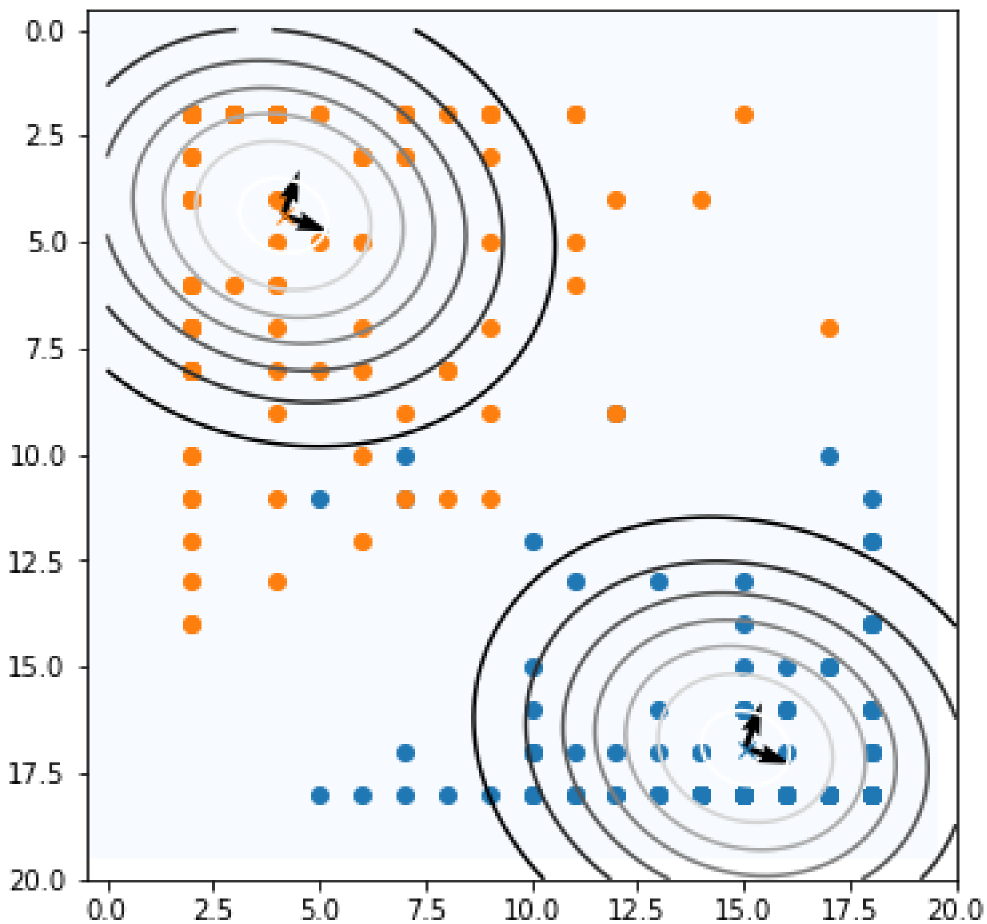
In [258]:
```
### recycle filtered numbers; map labels 1 -> 0 , 7 -> 1
X_train, X_test, y_train, y_test = \
train_test_split(data17,target17, test_size=0.2)

y_train[np.where(y_train==1)],y_train[np.where(y_train==7)] = 0, 1
y_test[np.where(y_test==1)],y_test[np.where(y_test==7)] = 0, 1
reduced_x_train =reduced_dim(X_train)
reduced_x_test =reduced_dim(X_test)
[mu, covmat] = fit_lda(reduced_x_train,y_train)

print("Training data: %d out of %d elements correct" % (np.sum(predict_lda(mu,covmat,r
```

```
print("Test data: %d out of %d elements correct" % (np.sum(predict_lda(mu,covmat,reduc
```

```
Training data: 282 out of 288 elements correct
Test data: 70 out of 73 elements correct
```

```
In [259]: ### adjust plots

          transx = 2
          transy = 2



          x_train_class0 = reduced_x_train[np.where(y_train==0)] #### seperate classes
          x_train_class1 = reduced_x_train[np.where(y_train==1)]


          grid = np.zeros((20,20)) ## background
          plt.imshow(grid,cmap="Blues",aspect=1)


          plt.scatter(x_train_class0[:,0]+transx,x_train_class0[:,1]+transy, ) ## training data
          plt.scatter(x_train_class1[:,0]+transx,x_train_class1[:,1]+transy)

          plt.plot(mu[0,0]+transx,mu[0,1]+transy,'x') ## means
          plt.plot(mu[1,0]+transx,mu[1,1]+transy,'x')



          #### bivariat gaussians
          x = np.linspace(0,20,100)
          y = np.linspace(0,20,100)
          x,y = np.meshgrid(x,y)
          pos = np.empty(x.shape + (2,))
          pos[:, :, 0] = x
          pos[:, :, 1] = y
          from scipy.stats import multivariate_normal
          var = multivariate_normal(mean=mu[0]+[transx,transy], cov=covmat)
          plt.contour(x,y,var.pdf(pos))
          var = multivariate_normal(mean=mu[1]+[transx,transy], cov=covmat)
          plt.contour(x,y,var.pdf(pos))



          ######## eigenvalue /vector decomposition

          val, vec = np.linalg.eig(covmat)
          for i,j in zip(val,vec):
```

```
    plt.quiver(mu[0,0]+transx,mu[0,1]+transy,np.sqrt(i)*j[0],np.sqrt(i)*j[1])
    plt.quiver(mu[1,0]+transx,mu[1,1]+transy,np.sqrt(i)*j[0],np.sqrt(i)*j[1])
```




```
In [260]: from sklearn.model_selection import KFold
          kf=KFold(n_splits=10)
          score=[]
          for train_index, test_index in kf.split(data_red):
              #print("TRAIN:", train_index, "TEST:", test_index)

              X_train, X_test = data_red[train_index], data_red[test_index]
              y_train, y_test = target17[train_index], target17[test_index]

              y_train[np.where(y_train==1)],y_train[np.where(y_train==7)] = 0, 1
              y_test[np.where(y_test==1)],y_test[np.where(y_test==7)] = 0, 1

              mu, covmat = fit_lda(X_train,y_train)

              #print("Training data: %d out of %d elements correct" % (np.sum(predict_qda(mu,co
```

```
            correct=np.sum(predict_lda(mu,covmat,X_test)==y_test)
            print("Test data: %d out of %d elements correct" % (correct,len(y_test)))
            score.append(correct/len(y_test))
        print('Average rate of prediction: ',np.mean(score),'+-',np.std(score))
```

```
Test data: 36 out of 37 elements correct
Test data: 33 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 34 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 36 out of 36 elements correct
Test data: 35 out of 36 elements correct
Test data: 34 out of 36 elements correct
Test data: 36 out of 36 elements correct
Average rate of prediction:  0.975075075075 +- 0.0289945344051
```

We see that the prediction rates of QDA and LDA are within their error of equal quality. Looking at the scatter plot of the QDA we do not see striking differences to the LDA scatter plot. The nearest mean method gives predictions with similar success rate. This can be assigned to the fact that the initially chosen pixels seperated the classes very well.