# HW1 - Basics of ML

Include your code in the relevant cells below. Subparts labeled as questions (Q1.1, Q1.2, etc.) should have their answers filled in or plots placed prominently, as appropriate.

## Important notes:

1. On this and future homeworks, depending on the data size and your hardware configuration, experiments may take too long if you use the complete dataset. This may be challenging, as you may need to run multiple experiments. So, if an experiment takes too much time, start first with a smaller sample that will allow you to run your code within a reasonable time. Once you complete all tasks, before the final submission, you can allow longer run times and run your code with the complete set. However, if this is still taking too much time or causing your computer to freeze, it will be OK to submit experiments using a sample size that is feasible for your setting (indicate it clearly in your submission). Grading of the homework will not be affected from this type of variations in the design of your experiments.

1. You can switch between 2D image data and 1D vector data using the numpy functions flatten() and resize()

# S1: Understanding the data

- Load MNIST FASHION dataset (hint: consider asking Bard)

Q1.1: What is the number of features in the training dataset: ___

Q1.2: What is the number of samples in the training dataset: ___

Q1.1: What is the number of features in the testing dataset: ___

Q1.4: What is the number of samples in the testing dataset: ___

Q1.3: What is the dimensionality of each data sample: ___

In [15]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:

```
## Using tensorflow data
##!pip install tensorflow
import tensorflow as tf
mnist = tf.keras.datasets.fashion_mnist
(X_tr, Y_tr), (X_te, Y_te) = mnist.load_data()
```

In [ ]:

```
## Or using sklearn data
from sklearn.datasets import fetch_openml
```

```
mnist = fetch_openml('Fashion-MNIST', version=1)
mnist.target = mnist.target.astype(np.int32)
X_tr, Y_tr, X_te, Y_te  = train_test_split(mnist.data, mnist.target, random_st
```

In [8]:

```
print('X_tr: ' + str(X_tr.shape))
print('Y_tr: ' + str(Y_tr.shape))
print('X_te:  '  + str(X_te.shape))
print('Y_te:  '  + str(Y_te.shape))
```

```
X_tr: (60000, 28, 28)
Y_tr: (60000,)
X_te:  (10000, 28, 28)
Y_te:  (10000,)
```

In [9]:

```
print('Unique labels: ' + str(np.unique(Y_tr, return_counts=True)))
```

```
Unique labels: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([600
0, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000]))
```

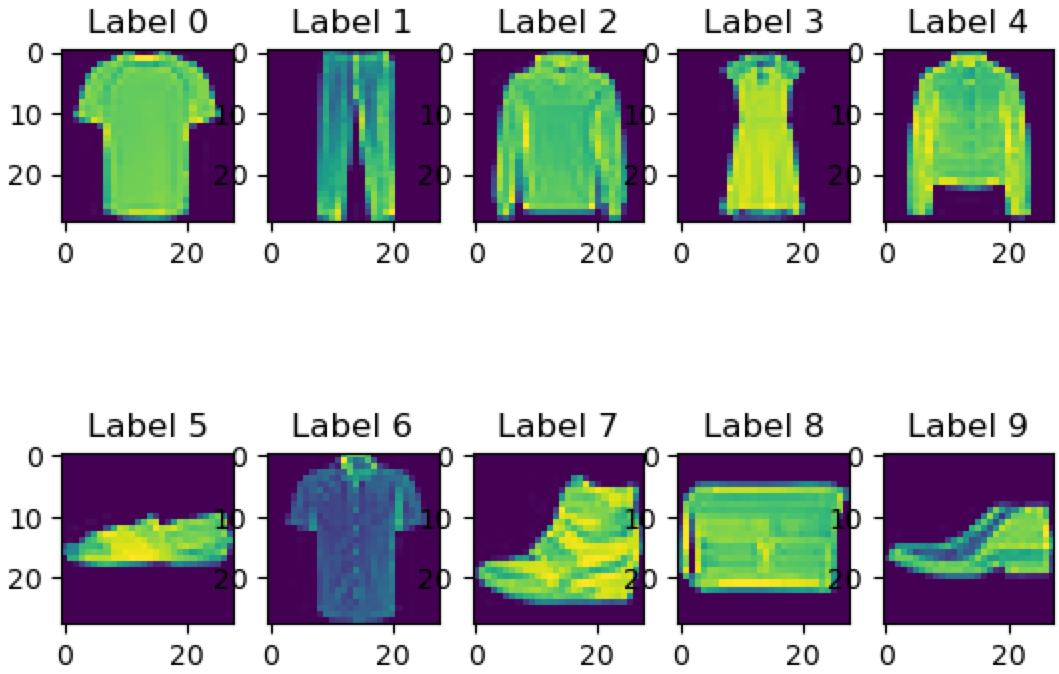
## S2: Viewing the data

- Select one random example from each category from the training set. Display the 2D image with the name of the category

Q2.1: Visualize the example image: ___

In [12]:

```
## Read categories
labels = np.unique(Y_tr)

## Create 2x5 subplot
fig, ax = plt.subplots(nrows = 2, ncols = 5)

## Loop to plot each category
for i, tmpl in enumerate(labels):

  ## Select a random image with the current label
  indAll = np.where(Y_tr == tmpl)[0]
  indSel = indAll[np.random.randint(indAll.shape)]
  selImg = X_tr[indSel,:,:].squeeze()

  ## Convert loop index to 2D index for the 2x5 plot
  a1, a2 = np.unravel_index(i, [2,5])

  ## Plot image
  ax[a1, a2].imshow(selImg)
  ax[a1, a2].set_title('Label ' + str(tmpl))

## Show image
plt.show()
```



## S3: Sub-sampling the data

- Reduce training and testing sample sizes by **randomly selecting** %10 of the initial samples

Q3.1: What is the distribution of each label in the initial train data (i.e. percentage of each label): ___

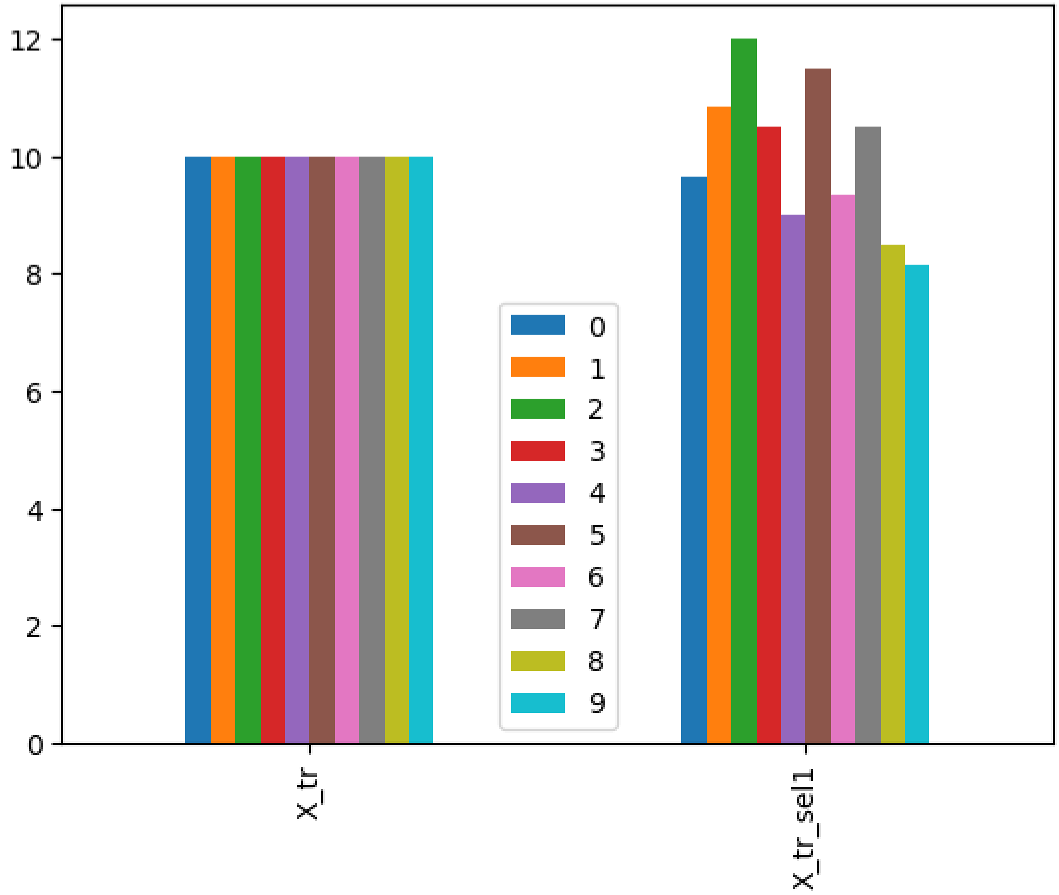
Q3.2: What is the distribution of each label in the reduced train data: ___

In [13]:
```
## Randomly sample data
def sample_data(X, Y, p):
  ## Shuffle array indices
  num_sample = X.shape[0]
  ind_shuf = np.random.permutation(num_sample)

  ## Select p percent of the shuffled indices
  num_sel = int(num_sample / 100 * p )
  ind_sel = ind_shuf[0:num_sel]

  ## Select data
  X_out = X[ind_sel, :]
  Y_out = Y[ind_sel]

  return X_out, Y_out

X_tr_sel1, Y_tr_sel1 = sample_data(X_tr, Y_tr, 1)
X_te_sel1, Y_te_sel1 = sample_data(X_te, Y_te, 1)
```

In [16]:
```
## Find number of samples for each label
_, c1 = np.unique(Y_tr, return_counts=True)
_, c2 = np.unique(Y_tr_sel1, return_counts=True)

## Find percentage of samples for each label
```

```
p1 = (100*c1 / Y_tr.shape[0]).reshape(1,-1)
p2 = (100*c2 / Y_tr_sel1.shape[0]).reshape(1,-1)

print(p1)
print(p2)

## or
p12 = np.concatenate((p1, p2), axis=0)
df = pd.DataFrame(data = p12, columns = labels, index = ['X_tr','X_tr_sel1'])

df.plot.bar()
plt.show()
```

```
[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]
[[ 9.66666667 10.83333333 12.         10.5         9.         11.5
   9.33333333 10.5         8.5         8.16666667]]
```



## S4: Sub-sampling the data (again)

- Reduce training and testing sample sizes by selecting **the first** %10 of the initial samples

Q4.1: What is the distribution of each label in the initial train data (i.e. percentage of each label):

___

Q4.2: What is the distribution of each label in the reduced train data: ___

Q4.3: What are your comments/interpretation on comparison of the results for S3 and S4

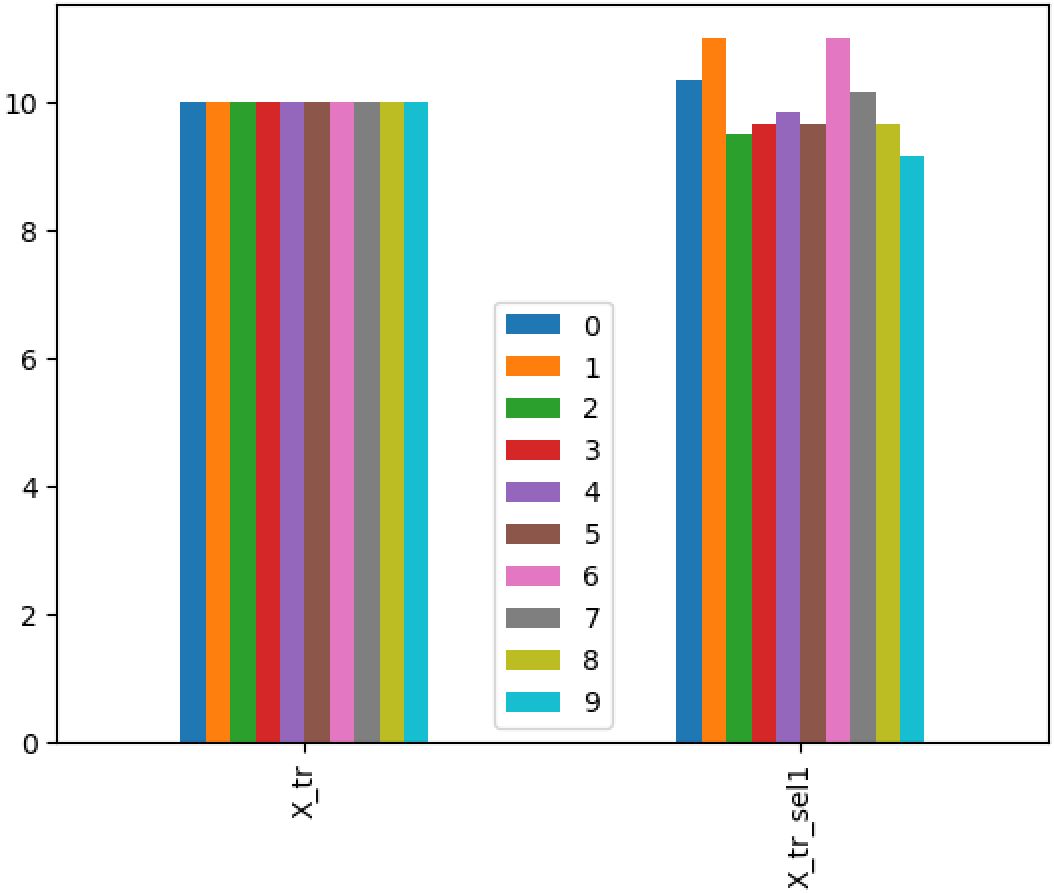
In [17]:

```
## Randomly sample data
def sample_data(X, Y, p, is_shuffle = False):
  ## Shuffle array indices
  num_sample = X.shape[0]

  if is_shuffle == True:
    ind_shuf = np.random.permutation(num_sample)
  else:
    ind_shuf = np.arange(0,num_sample)

  ## Select p percent of the shuffled indices
  num_sel = int(num_sample / 100 * p )
  ind_sel = ind_shuf[0:num_sel]

  ## Select data
  X_out = X[ind_sel, :]
  Y_out = Y[ind_sel]

  return X_out, Y_out

X_tr_sel2, Y_tr_sel2 = sample_data(X_tr, Y_tr, 1)
X_te_sel2, Y_te_sel2 = sample_data(X_te, Y_te, 1)
```

In [18]:

```
## Find number of samples for each label
_, c1 = np.unique(Y_tr, return_counts=True)
_, c2 = np.unique(Y_tr_sel2, return_counts=True)

## Find percentage of samples for each label
p1 = (100*c1 / Y_tr.shape[0]).reshape(1,-1)
p2 = (100*c2 / Y_tr_sel2.shape[0]).reshape(1,-1)

print(p1)
print(p2)

## or
p12 = np.concatenate((p1, p2), axis=0)
df = pd.DataFrame(data = p12, columns = labels, index = ['X_tr','X_tr_sel1'])

df.plot.bar()
plt.show()
```

```
[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]
[[10.33333333 11.          9.5         9.66666667  9.83333333  9.66666667
  11.         10.16666667  9.66666667  9.16666667]]
```



! For the rest of the HW, please discard sub-sampled data from S3 and use subsampled data from S4

## S5: Exploring the dataset

- Select all **train** images in category "coat". Create and display a single pixel-wise "average image" for this category.
- Create and display a single pixel-wise "standard deviation image" for this category?
- Repeat the items above for "coat" images in the **test** set. Compare the average and standard deviation images.
- Repeat the items above for a different category you select.

Q5.1: Plot the 2D mean and std images for coat in training and testing sets: ___

Q5.2: Plot the 2D mean and std images for the category you selected in training and testing sets: ___

Q5.3: Comment on differences between the mean and std images from training and testing datasets? ___

In [25]:
```
## Select images
X = X_tr_sel2
Y = Y_tr_sel2
indAll = np.where(Y == 4)[0]
imgAll = X[indAll, :, :]
```

```
print('Num sel: ' + str(indAll.shape))
print('Num img mat: ' + str(imgAll.shape))
```

```
Num sel: (59,)
Num img mat: (59, 28, 28)
```

In [26]:

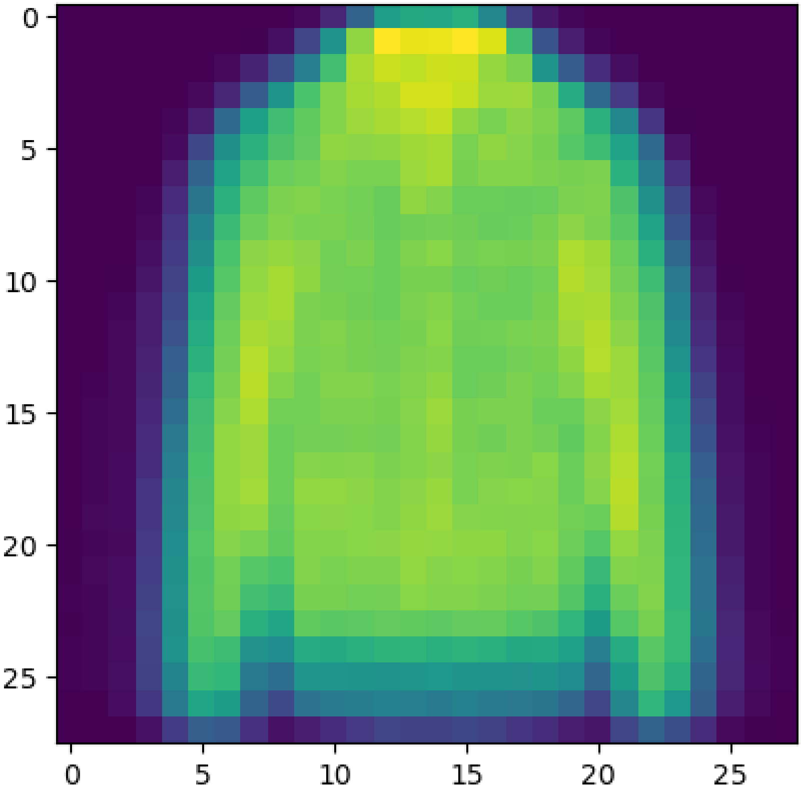
```
## Calculate avg img
img_mean = np.mean(imgAll, axis = 0)
img_mean.shape
```

Out[26]:

```
(28, 28)
```

In [27]:

```
## Show mean img
plt.imshow(img_mean)
plt.show()
```



## S6: Image distances

- In the training set, find the image in category coat that is most dissimilar to the mean image of category coat. Show it as a 2D image
- In the training set, find the image in category coat that is most similar to mean image of category coat. Show it as a 2D image
- In the training set, find the image in category sandal that is most similar to mean image of category coat. Show it as a 2D image

**Hint:** You can use the "euclidean distance" as your similarity metric. Given that an image i is represented with a flattened feature vector v_i , and the second image j with v_m, the distance between these two images can be calculated using the vector norm of their differences ( | v_i - v_j | )

Q6.1: What is the index of most dissimilar image in category coat: ___

Q6.2: Plot the most dissimilar category coat image in 2D: ___

Q6.3: Plot the most similar category coat image in 2D: ___

In [28]:

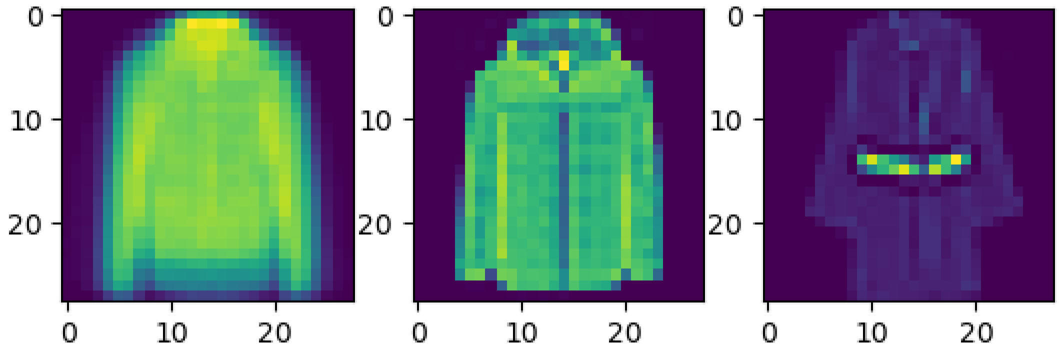
```
## Find pixelwise "distance" of each image to the mean image
vec_mean = img_mean.flatten()
arr_d = np.zeros(indAll.shape[0])
for i, ind_sel in enumerate(indAll):
  img_sel = X[ind_sel, :, :].flatten()

  d_sel = np.sqrt(np.dot(vec_mean - img_sel, vec_mean - img_sel))
  #d_sel = np.linalg.norm(img_sel - vec_mean)
  #d_sel = np.sqrt(np.square(img_sel - vec_mean).sum())

  arr_d[i] = d_sel
```

In [29]:

```
ind_similar = indAll[arr_d.argmin()]
ind_dissimilar = indAll[arr_d.argmax()]
```

In [30]:

```
## Show similar / dissimilar images
fig, ax = plt.subplots(nrows = 1, ncols = 3)
ax[0].imshow(img_mean)
ax[1].imshow(X[ind_similar,:,:])
ax[2].imshow(X[ind_dissimilar,:,:])
plt.show()
```



## S7: Image distances, part 2

- Repeat questions S3 and S4 after binarizing the images first

Q7.1: What is the index of most dis-similar category 3 image: ___

Q7.2: What is the index of most similar category 3 image: ___

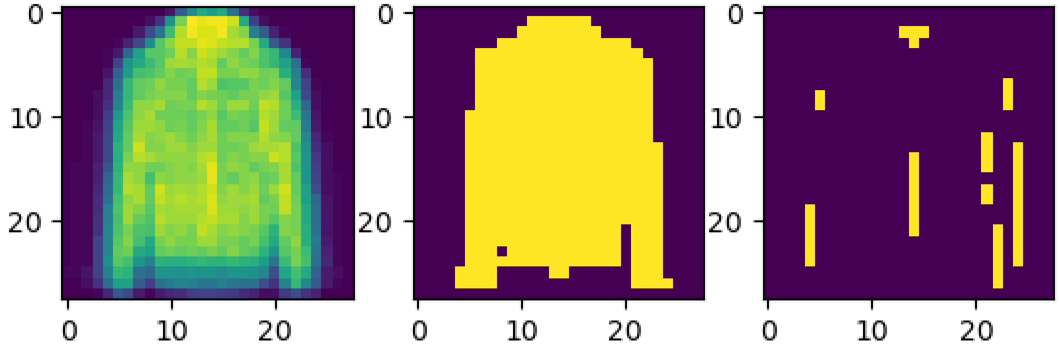
Q7.3: Did the answer change after binarization? How do you interprete this finding?: ___

In [32]:

```
## Select images
X = X_tr_sel2
Y = Y_tr_sel2

## Binarize images
```

```
X = (X>128).astype(int)

indAll = np.where(Y == 4)[0]
imgAll = X[indAll, :, :]
print('Num sel: ' + str(indAll.shape))
print('Num img mat: ' + str(imgAll.shape))

## Calculate avg img
img_mean = np.mean(imgAll, axis = 0)
img_mean.shape

## Find pixelwise "distance" of each image to the mean image
vec_mean = img_mean.flatten()
arr_d = np.zeros(indAll.shape[0])
for i, ind_sel in enumerate(indAll):
  img_sel = X[ind_sel, :, :].flatten()

  d_sel = np.sqrt(np.dot(vec_mean - img_sel, vec_mean - img_sel))
  #d_sel = np.linalg.norm(img_sel - vec_mean)
  #d_sel = np.sqrt(np.square(img_sel - vec_mean).sum())

  arr_d[i] = d_sel

ind_similar = indAll[arr_d.argmin()]
ind_dissimilar = indAll[arr_d.argmax()]

## Show similar / dissimilar images
fig, ax = plt.subplots(nrows = 1, ncols = 3)
ax[0].imshow(img_mean)
ax[1].imshow(X[ind_similar,:,:])
ax[2].imshow(X[ind_dissimilar,:,:])
plt.show()
```

```
Num sel: (59,)
Num img mat: (59, 28, 28)
```



## S8: Binary classification between category coat and sandal (split train data)

- Select images from these two categories in the training dataset
- Split them into two sets (Set1, Set2) with a %60 and %40 random split
- Replace category labels as 0 (for coat) and 1 (for sandal)
- Use Set1 to train a linear SVM classifier with default parameters and predict the class labels for Set2

- Use Set2 to train a linear SVM classifier with default parameters and predict the class labels for Set1

Q8.1: What is the prediction accuracy using the model trained on Set1: ___

Q8.2: What is the prediction accuracy using the model trained on Set2: ___

In [33]:
```
from sklearn.model_selection import train_test_split

## Select data
X = X_tr_sel2
Y = Y_tr_sel2

indSel = np.where((Y == 4) | (Y == 5))[0]
X = X[indSel, :, :]
Y = Y[indSel]

## Replace labels
Y[Y == 4] = 0
Y[Y == 5] = 1

## Flatten images
X = X.reshape(X.shape[0], -1)

## Create train test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, rando
```

In [34]:
```
from sklearn import svm

# Create the svm classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

# Train the model
clf.fit(X_train, y_train)

# Predict the label
y_pred = clf.predict(X_test)
```

In [35]:
```
from sklearn import metrics
print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy:  0.9574468085106383
```

In [36]:
```
# Train the model
clf.fit(X_test, y_test)

# Predict the label
y_pred = clf.predict(X_train)

print('Accuracy: ', metrics.accuracy_score(y_train, y_pred))
```

```
Accuracy:  0.9857142857142858
```

## S9: k-NN Error Analysis

- In training and testing datasets select the images in categories: 'Trouser', 'Pullover', 'Dress', 'Coat', `'Sandal', 'Shirt'`

- Train k-NN classifiers using 4 to 40 nearest neighbors with a step size of 4
- Calculate and plot overall testing accuracy for each experiment

Q9.1: For k=4 what is the label that was predicted with lowest accuracy: ___

Q9.2: For k=20 what is the label that was predicted with lowest accuracy: ___

Q9.3: What is the label pair that was confused most often (i.e. class A is labeled as B, and vice versa): ___

Q9.4: Visualize 5 mislabeled samples with their actual and predicted labels

## Bonus:

- We describe each image by using a reduced set of features (compared to n=784 initial features for each pixel value) as follows:

    1. Binarize the image (background=0, foreground=1)

    2. For each image row i, find n_i, the sum of 1's in the row (28 features)

    3. For each image column j, find n_j, the sum of 1's in the column (28 features)

    4. Concatenate these features into a feature vector of 56 features

Repeat classification experiments in S9 using this reduced feature set.

Q10.1: What is the prediction accuracy using the model trained using the train data: ___

Q10.2: What is the prediction accuracy using the model trained using the test data: ___