

✓ 0: Loading the MNIST fashion data set

It is already split out into a train and test set, but the X and y values (label or target) are in single file

I'm going to get you started here a bit, but pay attention to how I load the data here and the data formats used.

This data came as two csv files, with the filenames as shown

I got the data files from [kaggle.com](https://www.kaggle.com/zalando-research/fashionmnist), this data set is widely distributed

I loaded this as a pandas data frame, this is a relatively reliable, easy data frame to use

I think this file has a header

See

<https://www.kaggle.com/zalando-research/fashionmnist>

✓ load the pandas and numpy libraries

used for the data frame tools (Pandas) and to define matrices and do linear algebra (Numpy)

```
import pandas as pd
import numpy as np
```

The next steps load the test and training data into pandas data frames

Pandas has a dataframe structure much like the R dataframe, or an SQL table

There are many pandas member functions that do useful operations on the data frame, here the `read_csv()` member function is used to load csv files into data frames.

The infile style variables need to have the full path name to the location of the data files in us

```
train_infile="fashion-mnist_train.csv"

test_infile="fashion-mnist_test.csv"

train_df=pd.read_csv(train_infile)

test_df=pd.read_csv(test_infile)
```

Let's look at the available member function for a pandas data frame

```
dir(test_df)
```

```
tz_localize ,  
'unstack',  
'update',  
'value_counts',  
'values',  
'var',  
'where',  
'xs']
```

```
test_df.columns[0:5]
```

```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4'], dtype='object')
```

```
test_df.shape
```

```
(10000, 785)
```

```
train_df.shape
```

```
(60000, 785)
```

Okay, I'm expecting 28 x 28 greyscale images again, we have the first column as the label, the rest of this is the pixels

Most sklearn models will accept pandas dataframes as input data, so I don't think we need to do much here except split out the first column as y and the rest of the df as X

pandas has a member function called pop that removes a row from the dataframe. We'll use that to both set y_train equal to the labels, and X_train to the remaining df

```
y_train=train_df.pop('label')  
X_train=train_df
```

```
print(y_train.shape)  
print(X_train.shape)
```

```
(60000,)  
(60000, 784)
```

```
y_test=test_df.pop('label')  
X_test=test_df
```

Labels

Each training and test example is assigned to one of the following labels:

0 T-shirt/top 1 Trouser 2 Pullover 3 Dress 4 Coat 5 Sandal 6 Shirt 7 Sneaker 8 Bag 9 Ankle boot

The % symbol indicates that this is a magic function, that is to say a function command for the jupyter notebook server, not to the python kernel

This particular command causes plots created using the matplotlib library to print in the notebook not in a new window

```
%matplotlib inline
```

✓ 1: Data plots

Okay here is the visualization of one image, a shirt

Note: I use a location slice of the X_train dataframe, X_train.loc[0,:] to get row zero, all entries, or the first image in the array. I then force that into the np.array form so I can use the reshape() member function to reshape the row of data into a 28 x 28 image.

I don't think that pandas easily allows the reshape maneuver, so that's why I converted to an np.array. The reshape operation produces an np matrix that can be plotted with imshow. There may be a better way to



Also I checked, and we can feed X_train into the training input of the classifier as a pd.dataframe, there is no need to change the format

Most sklearn models will accept either pandas dataframes or np matrices as inputs, which is a help

```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[0,:])
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



```
y_train[0]
```

```
2
```

✓ Question/Action

What type of clothing is this image supposed to be? Insert a cell with your answer below

This is the 2nd kind in the target so Pullover.

✓ Question/Action

Show images of a sandal and a sneaker from this data set, show them in cells below

Show all your steps

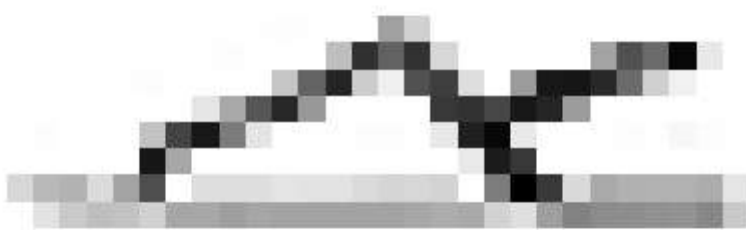
```
#Going to show sandal(5) and a sneaker(7)
#print(y_train[7])
#print(y_train[21])

mask = y_train.isin([5])
#mask[:21]
SandleIndex = np.nonzero(np.array(mask))
SandleIndex=SandleIndex[0][0]
mask = y_train.isin([7])
#mask[:21]
SneakerIndex = np.nonzero(np.array(mask))[0][0]
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[SandleIndex,:])
some_digit_image = some_digit.reshape(28, 28)

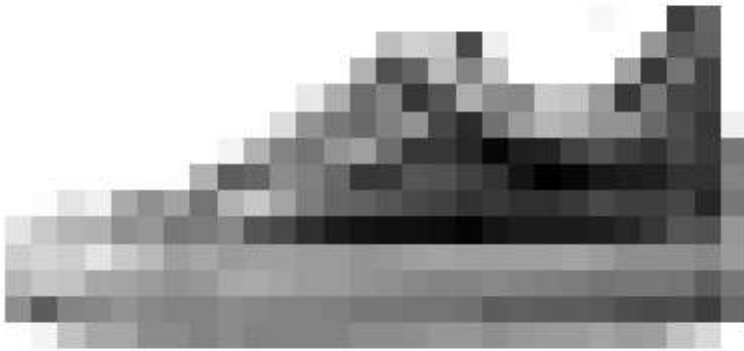
plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[SneakerIndex,:])
some_digit_image = some_digit.reshape(28, 28)

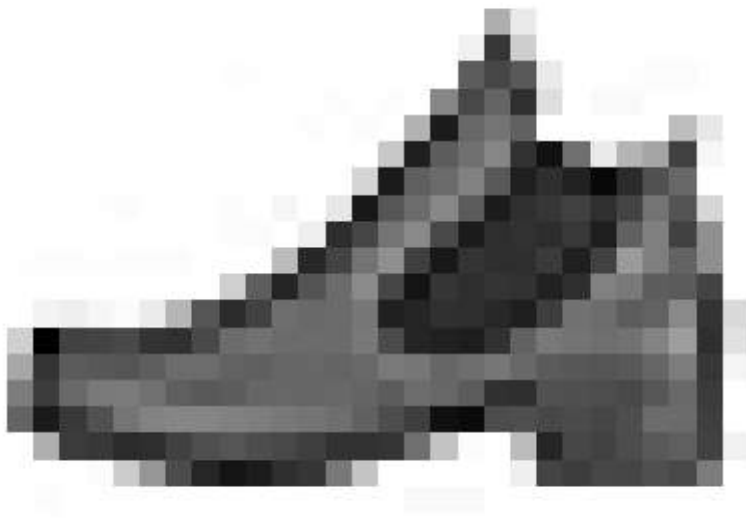
plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = np.array(X_train.loc[1,:])
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



```
y_train[1]
```

9

✓ 2: Time to Build some models

Lets build two different classifying neural net models

fclf - this should classify each image to one of the ten label classes (0-9)

fclf2- classify everything as either a pullover (2) or not a pullover, this is a binary categorization

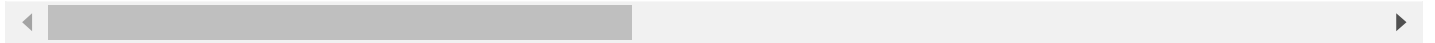
```
y_train_2=(y_train==2)
```

```
y_test_2=(y_test==2)
```

Okay, go build some models-Assignment

1.) For each model find the accuracy, the confusion matrix, the precision and the recall, label these all/ Look at the confusion matrix, explain which classes of objects were most likely to be confused with each other and which were most distinct. Explain why you think this happens, does it make sense?

I ran these quickly (so I know this works) and got 87.7 % accuracy for the X-train data set using all 97.7 % accuracy for the binary classification (ie the pullover detector). See if you can beat the q

- 
- 2.) Also, create the ROC curve for the binary classifier and compute the AUC for the ROC, for the binary classifier, but not for the 10 element classifier
 - 3.) When you are done with steps 1 and 2, use your two classifier models to classify the test data. Is there evidence of overfitting? What tells you this?

Print your completed jupyter notebook to a pdf file, you can use the browser to print to pdf. Upload this to dropbox in D2L to submit the homework.

Model fclf, classify fashion image to 10 categories

- ✓ First lets build fclf - this should classify each image to one of the ten label classes (0-9)

To improve performance I am going to increase the hidden layer sizes.

```
# multi catagory classification
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, max_iter=500,hidden_layer_siz
clf.fit(X_train, y_train)
```

Iteration 1, loss = 3.01173995
Iteration 2, loss = 1.49290983
Iteration 3, loss = 1.26351452
Iteration 4, loss = 1.08092941
Iteration 5, loss = 0.95217684
Iteration 6, loss = 0.85769123
Iteration 7, loss = 0.80354189
Iteration 8, loss = 0.76014074
Iteration 9, loss = 0.74044255
Iteration 10, loss = 0.71849393
Iteration 11, loss = 0.69391751
Iteration 12, loss = 0.67467273
Iteration 13, loss = 0.64428157
Iteration 14, loss = 0.60659959
Iteration 15, loss = 0.58760437
Iteration 16, loss = 0.56473928
Iteration 17, loss = 0.54435815
Iteration 18, loss = 0.52756067
Iteration 19, loss = 0.51277337
Iteration 20, loss = 0.50020870
Iteration 21, loss = 0.48862021
Iteration 22, loss = 0.47115685
Iteration 23, loss = 0.45939473
Iteration 24, loss = 0.44485045
Iteration 25, loss = 0.44223176
Iteration 26, loss = 0.43590556
Iteration 27, loss = 0.43400206
Iteration 28, loss = 0.42161737
Iteration 29, loss = 0.41504866
Iteration 30, loss = 0.40827832
Iteration 31, loss = 0.40675772
Iteration 32, loss = 0.41039817
Iteration 33, loss = 0.40400556
Iteration 34, loss = 0.39645916
Iteration 35, loss = 0.39929223
Iteration 36, loss = 0.38581689
Iteration 37, loss = 0.38296057
Iteration 38, loss = 0.38041096
Iteration 39, loss = 0.38127683
Iteration 40, loss = 0.37627755
Iteration 41, loss = 0.37115507
Iteration 42, loss = 0.36196779
Iteration 43, loss = 0.36314739
Iteration 44, loss = 0.36479858
Iteration 45, loss = 0.36045970
Iteration 46, loss = 0.36232922
Iteration 47, loss = 0.36014691
Iteration 48, loss = 0.35215833
Iteration 49, loss = 0.34433603
Iteration 50, loss = 0.34761822
Iteration 51, loss = 0.34487601
Iteration 52, loss = 0.34295914
Iteration 53, loss = 0.34052425
Iteration 54, loss = 0.34973542
Iteration 55, loss = 0.33952980
Iteration 56, loss = 0.34037304



✓ prediction outputs

```
clf.predict_proba(X_test[:10])
```

```
array([[7.88660135e-001, 4.08424110e-012, 2.12187289e-003,  
        5.64464633e-005, 1.83001612e-004, 2.21532380e-014,  
        2.08674937e-001, 2.32385998e-041, 3.03606953e-004,  
        3.11961391e-019],  
       [1.43123110e-018, 9.99999971e-001, 6.14255229e-011,  
        2.95366603e-009, 2.53569794e-008, 2.19339125e-016,  
        4.72335248e-015, 4.94034477e-018, 1.38457733e-010,  
        6.47158134e-013],  
       [2.75030243e-005, 3.88606506e-010, 9.93694451e-001,  
        8.64433828e-008, 1.93639261e-003, 3.08176696e-028,  
        4.26083844e-003, 1.07118339e-077, 8.07279506e-005,  
        1.55091177e-034],  
       [2.66618857e-003, 2.00013317e-012, 8.48774935e-001,  
        7.01247696e-007, 2.06802377e-003, 4.61260823e-027,  
        1.46317861e-001, 6.79144286e-078, 1.72290738e-004,  
        4.85043336e-034],  
       [4.04627450e-006, 6.88796427e-006, 3.11677318e-004,  
        7.42240698e-001, 2.44069999e-001, 1.16913128e-014,  
        1.12369561e-002, 8.24790934e-041, 2.12973505e-003,  
        1.15941764e-015],  
       [1.40882125e-001, 6.15313390e-003, 2.18419246e-001,  
        2.83419456e-001, 8.75277917e-002, 1.25063212e-007,
```

```

2.08576861e-001, 2.10148960e-019, 5.50212572e-002,
3.87704774e-009],
[6.01579857e-003, 9.04466829e-005, 2.26402723e-004,
4.12583657e-004, 3.29345412e-004, 8.38378799e-004,
3.59958170e-003, 4.03441856e-004, 9.87882592e-001,
2.01428896e-004],
[7.66940166e-003, 1.95697079e-007, 3.27664370e-001,
7.41143571e-003, 1.29062052e-001, 1.23177653e-017,
5.18746072e-001, 2.51462314e-051, 9.44647241e-003,
1.73883344e-021],
[9.09433406e-135, 0.00000000e+000, 1.74961470e-264,
1.53418953e-251, 3.93143123e-096, 1.00000000e+000,
6.72208982e-100, 4.40900342e-069, 2.23453206e-059,
7.80183512e-053],
[9.49801650e-001, 1.13665360e-008, 1.90179245e-003,
2.10664005e-004, 3.65418674e-004, 4.64221909e-009,
4.63334352e-002, 8.81407674e-024, 1.38702378e-003,
2.23591384e-012]])

```

```
clf.predict(X_test[:20])
```

```
# this is looking vary clean based on these first few outputs
```

```
array([0, 1, 2, 2, 3, 3, 8, 6, 5, 0, 3, 2, 4, 6, 8, 5, 6, 3, 6, 4])
```

```
y_pred=clf.predict(X_train)
```

✓ Confusion Matrix

```
from sklearn.metrics import confusion_matrix
```

```
my_cm=confusion_matrix(y_train,y_pred,labels=[0,1,2,3,4,5,6,7,8,9])
```

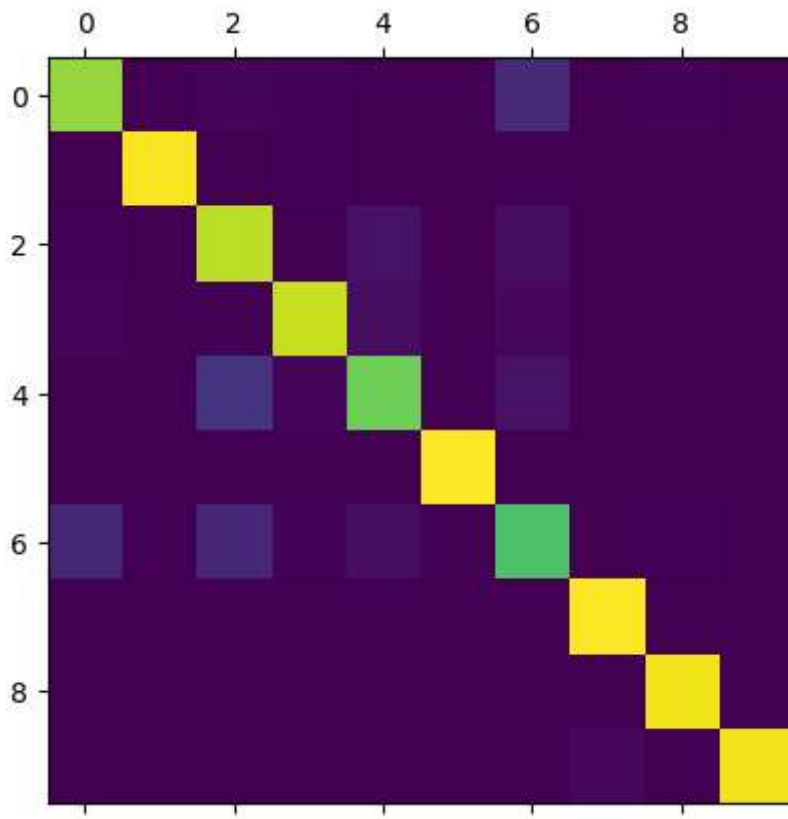
```
my_cm
```

```

array([[5047,    6,   92,   64,    6,    0,  735,    0,   50,    0],
       [  4, 5921,    6,   55,    8,    0,   3,    0,    3,    0],
       [ 60,   4, 5380,   19,  300,    0,  232,    0,    5,    0],
       [116,  28,   35, 5472,  212,    0,  134,    0,    3,    0],
       [ 10,   7,  917,  116, 4663,    0,  284,    0,    3,    0],
       [  1,   0,   0,   0,   0, 5972,    0,   20,    6,    1],
       [660,   3,  668,   67,  223,    0, 4324,    0,   55,    0],
       [  0,   0,   0,   0,   0,    0,   0, 5986,    3,   11],
       [ 18,   3,   43,   23,   26,    0,   30,    0, 5857,    0],
       [  0,   1,   0,   0,   0,   4,    0,  129,    2, 5864]])

```

```
#Heatmap
import matplotlib.pyplot as plt
plt.matshow(my_cm)
plt.show()
```



add up all the correct predictions

```
my_cm.trace()
```

54486

Sum everything

```
my_cm.sum()
```

60000

Now we compute the accuracy $\text{Accuracy} = (\text{sum along the diagonals}) / (\text{sum of all cells})$

```
my_cm.trace()/my_cm.sum()
```

0.9081

Accuracy of 90% as I increased the size of the hidden layers.

- ✓ fclf2- classify everything as either a pullover (2) or not a pullover, this is a binary categorization

```
from sklearn.neural_network import MLPClassifier
clf2 = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, max_iter=500,hidden_layer_si
clf2.fit(X_train, y_train_2)
```



```
Iteration 93, loss = 0.08173352
Iteration 94, loss = 0.08087841
Iteration 95, loss = 0.08108362
Iteration 96, loss = 0.07963252
Iteration 97, loss = 0.08174124
Iteration 98, loss = 0.08185383
Iteration 99, loss = 0.07878595
Iteration 100, loss = 0.07952181
Iteration 101, loss = 0.08022948
Iteration 102, loss = 0.07916108
Iteration 103, loss = 0.07921105
Iteration 104, loss = 0.07717043
Iteration 105, loss = 0.07959943
Iteration 106, loss = 0.07716020
Iteration 107, loss = 0.07549642
Iteration 108, loss = 0.07568700
Iteration 109, loss = 0.07439545
Iteration 110, loss = 0.07623094
Iteration 111, loss = 0.07597674
Iteration 112, loss = 0.07625656
Iteration 113, loss = 0.07371278
Iteration 114, loss = 0.07628979
Iteration 115, loss = 0.07459080
Iteration 116, loss = 0.07122457
Iteration 117, loss = 0.07346776
Iteration 118, loss = 0.07485099
Iteration 119, loss = 0.07224224
Iteration 120, loss = 0.07289476
Iteration 121, loss = 0.07119516
Iteration 122, loss = 0.07142392
Iteration 123, loss = 0.06942276
Iteration 124, loss = 0.07120889
Iteration 125, loss = 0.07055523
Iteration 126, loss = 0.07293400
Iteration 127, loss = 0.07198840
Iteration 128, loss = 0.07099159
Iteration 129, loss = 0.07032805
Iteration 130, loss = 0.07031218
Iteration 131, loss = 0.07255654
Iteration 132, loss = 0.06942990
Iteration 133, loss = 0.06749503
Iteration 134, loss = 0.07042220
Iteration 135, loss = 0.07233699
Iteration 136, loss = 0.07066676
Iteration 137, loss = 0.06905253
Iteration 138, loss = 0.06911568
Iteration 139, loss = 0.06954439
Iteration 140, loss = 0.06955509
Iteration 141, loss = 0.06938016
Iteration 142, loss = 0.06793332
Iteration 143, loss = 0.06806531
Iteration 144, loss = 0.06839155
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
```

```
▼ MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(28, 14, 7), max_iter=500,
random_state=1, verbose=True)
```

