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, 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)

In [1]:
```
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

In [2]:
```
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

In [3]:
```
dir(test_df)
```

Out[3]:
```
['T',
 '_AXIS_LEN',
 '_AXIS_ORDERS',
 '_AXIS_TO_AXIS_NUMBER',
 '_HANDLED_TYPES',
 '__abs__',
 '__add__',
 '__and__',
 '__annotations__',
 '__array__',
 '__array_priority__',
 '__array_ufunc__',
 '__bool__',
 '__class__',
 '__contains__',
 '__copy__',
 '__dataframe__',
 '__deepcopy__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__finalize__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__iand__',
 '__ifloordiv__',
 '__imod__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__invert__',
 '__ior__',
 '__ipow__',
 '__isub__',
 '__iter__',
 '__itruediv__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__matmul__',
 '__mod__',
 '__module__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__nonzero__',
```

```
'__or__',
'__pos__',
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rmatmul__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__setitem__',
'__setstate__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__weakref__',
'__xor__',
'_accessors',
'_accum_func',
'_add_numeric_operations',
'_agg_examples_doc',
'_agg_summary_and_see_also_doc',
'_align_frame',
'_align_series',
'_append',
'_arith_method',
'_as_manager',
'_attrs',
'_box_col_values',
'_can_fast_transpose',
'_check_inplace_and_allows_duplicate_labels',
'_check_inplace_setting',
'_check_is_chained_assignment_possible',
'_check_label_or_level_ambiguity',
'_check_setitem_copy',
'_clear_item_cache',
'_clip_with_one_bound',
'_clip_with_scalar',
'_cmp_method',
'_combine_frame',
'_consolidate',
'_consolidate_inplace',
'_construct_axes_dict',
'_construct_result',
'_constructor',
'_constructor_sliced',
'_create_data_for_split_and_tight_to_dict',
'_data',
```

```
 '_dir_additions',
 '_dir_deletions',
 '_dispatch_frame_op',
 '_drop_axis',
 '_drop_labels_or_levels',
 '_ensure_valid_index',
 '_find_valid_index',
 '_flags',
 '_from_arrays',
 '_get_agg_axis',
 '_get_axis',
 '_get_axis_name',
 '_get_axis_number',
 '_get_axis_resolvers',
 '_get_block_manager_axis',
 '_get_bool_data',
 '_get_cleaned_column_resolvers',
 '_get_column_array',
 '_get_index_resolvers',
 '_get_item_cache',
 '_get_label_or_level_values',
 '_get_numeric_data',
 '_get_value',
 '_getitem_bool_array',
 '_getitem_multilevel',
 '_getitem_nocopy',
 '_gotitem',
 '_hidden_attrs',
 '_indexed_same',
 '_info_axis',
 '_info_axis_name',
 '_info_axis_number',
 '_info_repr',
 '_init_mgr',
 '_inplace_method',
 '_internal_names',
 '_internal_names_set',
 '_is_copy',
 '_is_homogeneous_type',
 '_is_label_or_level_reference',
 '_is_label_reference',
 '_is_level_reference',
 '_is_mixed_type',
 '_is_view',
 '_iset_item',
 '_iset_item_mgr',
 '_iset_not_inplace',
 '_item_cache',
 '_iter_column_arrays',
 '_ixs',
 '_join_compat',
 '_logical_func',
 '_logical_method',
 '_maybe_cache_changed',
 '_maybe_update_cacher',
 '_metadata',
 '_mgr',
 '_min_count_stat_function',
 '_needs_reindex_multi',
 '_protect_consolidate',
```

```
 '_reduce',
 '_reduce_axis1',
 '_reindex_axes',
 '_reindex_columns',
 '_reindex_index',
 '_reindex_multi',
 '_reindex_with_indexers',
 '_rename',
 '_replace_columnwise',
 '_repr_data_resource_',
 '_repr_fits_horizontal_',
 '_repr_fits_vertical_',
 '_repr_html_',
 '_repr_latex_',
 '_reset_cache',
 '_reset_cacher',
 '_sanitize_column',
 '_series',
 '_set_axis',
 '_set_axis_name',
 '_set_axis_nocheck',
 '_set_is_copy',
 '_set_item',
 '_set_item_frame_value',
 '_set_item_mgr',
 '_set_value',
 '_setitem_array',
 '_setitem_frame',
 '_setitem_slice',
 '_slice',
 '_stat_axis',
 '_stat_axis_name',
 '_stat_axis_number',
 '_stat_function',
 '_stat_function_ddof',
 '_take',
 '_take_with_is_copy',
 '_to_dict_of_blocks',
 '_to_latex_via_styler',
 '_typ',
 '_update_inplace',
 '_validate_dtype',
 '_values',
 '_where',
 'abs',
 'add',
 'add_prefix',
 'add_suffix',
 'agg',
 'aggregate',
 'align',
 'all',
 'any',
 'apply',
 'applymap',
 'asfreq',
 'asof',
 'assign',
 'astype',
 'at',
```

```
 'at_time',
 'attrs',
 'axes',
 'backfill',
 'between_time',
 'bfill',
 'bool',
 'boxplot',
 'clip',
 'columns',
 'combine',
 'combine_first',
 'compare',
 'convert_dtypes',
 'copy',
 'corr',
 'corrwith',
 'count',
 'cov',
 'cummax',
 'cummin',
 'cumprod',
 'cumsum',
 'describe',
 'diff',
 'div',
 'divide',
 'dot',
 'drop',
 'drop_duplicates',
 'droplevel',
 'dropna',
 'dtypes',
 'duplicated',
 'empty',
 'eq',
 'equals',
 'eval',
 'ewm',
 'expanding',
 'explode',
 'ffill',
 'fillna',
 'filter',
 'first',
 'first_valid_index',
 'flags',
 'floordiv',
 'from_dict',
 'from_records',
 'ge',
 'get',
 'groupby',
 'gt',
 'head',
 'hist',
 'iat',
 'idxmax',
 'idxmin',
 'iloc',
```

```
 'index',
 'infer_objects',
 'info',
 'insert',
 'interpolate',
 'isetitem',
 'isin',
 'isna',
 'isnull',
 'items',
 'iterrows',
 'itertuples',
 'join',
 'keys',
 'kurt',
 'kurtosis',
 'label',
 'last',
 'last_valid_index',
 'le',
 'loc',
 'lt',
 'mask',
 'max',
 'mean',
 'median',
 'melt',
 'memory_usage',
 'merge',
 'min',
 'mod',
 'mode',
 'mul',
 'multiply',
 'ndim',
 'ne',
 'nlargest',
 'notna',
 'notnull',
 'nsmallest',
 'nunique',
 'pad',
 'pct_change',
 'pipe',
 'pivot',
 'pivot_table',
 'pixel1',
 'pixel10',
 'pixel11',
 'pixel12',
 'pixel13',
 'pixel14',
 'pixel15',
 'pixel16',
 'pixel17',
 'pixel18',
 'pixel19',
 'pixel2',
 'pixel20',
 'pixel21',
```

```
                'pixel22',
                'pixel23',
                'pixel24',
                'pixel25',
                'pixel26',
                'pixel27',
                'pixel28',
                'pixel29',
                'pixel3',
                'pixel30',
                'pixel31',
                'pixel32',
                'pixel33',
                'pixel34',
                'pixel35',
                'pixel36',
                'pixel37',
                'pixel38',
                'pixel39',
                'pixel4',
                'pixel40',
                'pixel41',
                'pixel42',
                'pixel43',
                'pixel44',
                'pixel45',
                'pixel46',
                'pixel47',
                'pixel48',
                'pixel49',
                'pixel5',
                'pixel50',
                'pixel51',
                'pixel52',
                'pixel53',
                'pixel54',
                'pixel55',
                'pixel56',
                'pixel57',
                'pixel58',
                'pixel59',
                'pixel6',
                'pixel60',
                'pixel61',
                'pixel62',
                'pixel63',
                'pixel64',
                'pixel65',
                'pixel66',
                'pixel67',
                'pixel68',
                'pixel69',
                'pixel7',
                'pixel70',
                'pixel71',
                'pixel72',
                'pixel73',
                'pixel74',
                'pixel75',
                'pixel76',
```

```
 'pixel77',
 'pixel78',
 'pixel79',
 'pixel8',
 'pixel80',
 'pixel81',
 'pixel82',
 'pixel83',
 'pixel84',
 'pixel85',
 'pixel86',
 'pixel87',
 'pixel88',
 'pixel89',
 'pixel9',
 'pixel90',
 'pixel91',
 'pixel92',
 'pixel93',
 'pixel94',
 'pixel95',
 'pixel96',
 'pixel97',
 'pixel98',
 'pixel99',
 'plot',
 'pop',
 'pow',
 'prod',
 'product',
 'quantile',
 'query',
 'radd',
 'rank',
 'rdiv',
 'reindex',
 'reindex_like',
 'rename',
 'rename_axis',
 'reorder_levels',
 'replace',
 'resample',
 'reset_index',
 'rfloordiv',
 'rmod',
 'rmul',
 'rolling',
 'round',
 'rpow',
 'rsub',
 'rtruediv',
 'sample',
 'select_dtypes',
 'sem',
 'set_axis',
 'set_flags',
 'set_index',
 'shape',
 'shift',
 'size',
```

```
 'skew',
 'sort_index',
 'sort_values',
 'squeeze',
 'stack',
 'std',
 'style',
 'sub',
 'subtract',
 'sum',
 'swapaxes',
 'swaplevel',
 'tail',
 'take',
 'to_clipboard',
 'to_csv',
 'to_dict',
 'to_excel',
 'to_feather',
 'to_gbq',
 'to_hdf',
 'to_html',
 'to_json',
 'to_latex',
 'to_markdown',
 'to_numpy',
 'to_orc',
 'to_parquet',
 'to_period',
 'to_pickle',
 'to_records',
 'to_sql',
 'to_stata',
 'to_string',
 'to_timestamp',
 'to_xarray',
 'to_xml',
 'transform',
 'transpose',
 'truediv',
 'truncate',
 'tz_convert',
 'tz_localize',
 'unstack',
 'update',
 'value_counts',
 'values',
 'var',
 'where',
 'xs']
```

In [4]:
```
test_df.columns[0:5]
```

Out[4]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4'], dtype='object')
```

In [5]:
```
test_df.shape
```

Out[5]:
```
(10000, 785)
```

In [6]:
```
train_df.shape
```

Out[6]:
```
(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 skearn 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

In [7]:
```
y_train=train_df.pop('label')
X_train=train_df
```

In [8]:
```
print(y_train.shape)
print(X_train.shape)
```

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

In [9]:
```
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 libary to print in the notebook not in a new window

In [10]:
```
%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 do this.  Hmm.
```

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

In [11]:
```
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()
```

In [12]:
```
y_train[0]
```

Out[12]:
```
2
```

Question/Action

What type of cloting 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

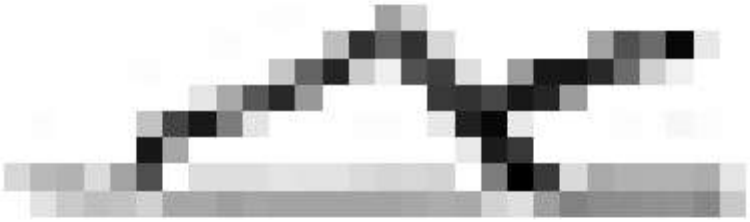
Show images of a sandal and a sneaker from this data set, show them in cells below

Show all your steps

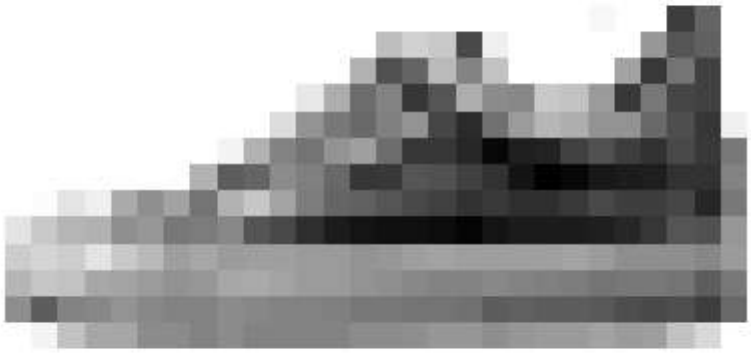
In [13]:
```
#Going to show sandal(5) and a sneaker(7)
#print(y_train[7])
#print(y_train[21])


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]
```

In [ ]:
```

```

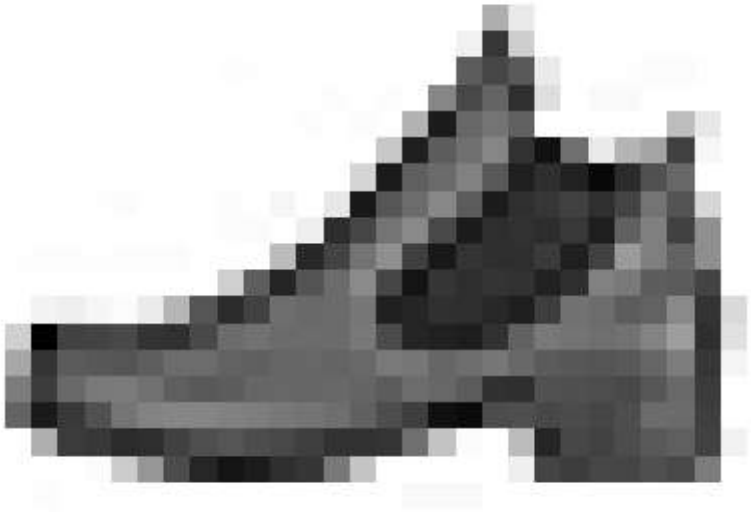
In [14]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

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()
```



In [15]:
```
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()
```



In [16]:
```
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()
```



In [17]:
```
y_train[1]
```

Out[17]:
```
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

In [18]:
```
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 10 classes and
97.7 % accuracy  for the binary classification (ie the pullover
detector).  See if you can beat the quick results I got.       Post
your results in the discussion section of D2L for this week.  Discuss
```

```
what you did to beat my score
```

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.

In [22]:
```
# multi catagory classification
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, max_iter=500,hidden_lay

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 = 0.74071699
Iteration 10, loss = 0.71992289
Iteration 11, loss = 0.69498030
Iteration 12, loss = 0.65479113
Iteration 13, loss = 0.62746111
Iteration 14, loss = 0.59328588
Iteration 15, loss = 0.58704561
Iteration 16, loss = 0.56617290
Iteration 17, loss = 0.55038780
Iteration 18, loss = 0.54214097
Iteration 19, loss = 0.53381012
Iteration 20, loss = 0.52889403
Iteration 21, loss = 0.50821149
Iteration 22, loss = 0.50475929
Iteration 23, loss = 0.50099703
Iteration 24, loss = 0.48823128
Iteration 25, loss = 0.48066324
Iteration 26, loss = 0.48427558
Iteration 27, loss = 0.47471865
Iteration 28, loss = 0.46706446
Iteration 29, loss = 0.46301338
Iteration 30, loss = 0.46183543
Iteration 31, loss = 0.45288493
Iteration 32, loss = 0.44283905
Iteration 33, loss = 0.44759004
Iteration 34, loss = 0.44385572
Iteration 35, loss = 0.43309015
Iteration 36, loss = 0.43914283
Iteration 37, loss = 0.43572342
Iteration 38, loss = 0.42392193
Iteration 39, loss = 0.42643397
Iteration 40, loss = 0.43354650
Iteration 41, loss = 0.41869926
Iteration 42, loss = 0.40742222
Iteration 43, loss = 0.41771007
Iteration 44, loss = 0.41401984
Iteration 45, loss = 0.41173289
Iteration 46, loss = 0.41391602
Iteration 47, loss = 0.43219970
Iteration 48, loss = 0.41754065
Iteration 49, loss = 0.40944117
Iteration 50, loss = 0.40672242
Iteration 51, loss = 0.41118798
Iteration 52, loss = 0.40192434
Iteration 53, loss = 0.39669850
Iteration 54, loss = 0.39084078
Iteration 55, loss = 0.39804849
Iteration 56, loss = 0.40436958
Iteration 57, loss = 0.39793278
Iteration 58, loss = 0.39621937
Iteration 59, loss = 0.38609852
Iteration 60, loss = 0.39851793
```

```
Iteration 61, loss = 0.38956521
Iteration 62, loss = 0.38762192
Iteration 63, loss = 0.39172386
Iteration 64, loss = 0.37681862
Iteration 65, loss = 0.38654563
Iteration 66, loss = 0.37867232
Iteration 67, loss = 0.39791409
Iteration 68, loss = 0.38217361
Iteration 69, loss = 0.39023654
Iteration 70, loss = 0.37500529
Iteration 71, loss = 0.36843976
Iteration 72, loss = 0.37181539
Iteration 73, loss = 0.37394381
Iteration 74, loss = 0.37184180
Iteration 75, loss = 0.36591970
Iteration 76, loss = 0.37254339
Iteration 77, loss = 0.36271590
Iteration 78, loss = 0.36953195
Iteration 79, loss = 0.36905773
Iteration 80, loss = 0.36072667
Iteration 81, loss = 0.36312409
Iteration 82, loss = 0.35446600
Iteration 83, loss = 0.35891275
Iteration 84, loss = 0.34908594
Iteration 85, loss = 0.36807002
Iteration 86, loss = 0.39533154
Iteration 87, loss = 0.35205459
Iteration 88, loss = 0.35211317
Iteration 89, loss = 0.34699683
Iteration 90, loss = 0.35156941
Iteration 91, loss = 0.34863219
Iteration 92, loss = 0.36369958
Iteration 93, loss = 0.34930115
Iteration 94, loss = 0.34839154
Iteration 95, loss = 0.33510515
Iteration 96, loss = 0.35424524
Iteration 97, loss = 0.33912510
Iteration 98, loss = 0.34122082
Iteration 99, loss = 0.33556389
Iteration 100, loss = 0.33697050
Iteration 101, loss = 0.36096525
Iteration 102, loss = 0.35152700
Iteration 103, loss = 0.33536601
Iteration 104, loss = 0.33325380
Iteration 105, loss = 0.33229259
Iteration 106, loss = 0.32235253
Iteration 107, loss = 0.33294542
Iteration 108, loss = 0.32488096
Iteration 109, loss = 0.32520606
Iteration 110, loss = 0.33507492
Iteration 111, loss = 0.33531570
Iteration 112, loss = 0.33842996
Iteration 113, loss = 0.32469261
Iteration 114, loss = 0.32421156
Iteration 115, loss = 0.32015625
Iteration 116, loss = 0.32738395
Iteration 117, loss = 0.32589614
Iteration 118, loss = 0.32298735
Iteration 119, loss = 0.31630763
Iteration 120, loss = 0.30907182
```

```
Iteration 121, loss = 0.31298654
Iteration 122, loss = 0.31264618
Iteration 123, loss = 0.31097222
Iteration 124, loss = 0.33040078
Iteration 125, loss = 0.32684590
Iteration 126, loss = 0.31039409
Iteration 127, loss = 0.33007555
Iteration 128, loss = 0.30772820
Iteration 129, loss = 0.31415958
Iteration 130, loss = 0.31295699
Iteration 131, loss = 0.30940005
Iteration 132, loss = 0.29922357
Iteration 133, loss = 0.32413835
Iteration 134, loss = 0.31213957
Iteration 135, loss = 0.29900907
Iteration 136, loss = 0.31372807
Iteration 137, loss = 0.30875980
Iteration 138, loss = 0.30057243
Iteration 139, loss = 0.31420285
Iteration 140, loss = 0.30136406
Iteration 141, loss = 0.30385283
Iteration 142, loss = 0.30164876
Iteration 143, loss = 0.31410024
Iteration 144, loss = 0.29944695
Iteration 145, loss = 0.30462179
Iteration 146, loss = 0.28714868
Iteration 147, loss = 0.29783341
Iteration 148, loss = 0.29602423
Iteration 149, loss = 0.30178636
Iteration 150, loss = 0.29980097
Iteration 151, loss = 0.29146905
Iteration 152, loss = 0.30756637
Iteration 153, loss = 0.29417022
Iteration 154, loss = 0.29321751
Iteration 155, loss = 0.28149216
Iteration 156, loss = 0.28330084
Iteration 157, loss = 0.28575766
Iteration 158, loss = 0.30897121
Iteration 159, loss = 0.29747836
Iteration 160, loss = 0.29859504
Iteration 161, loss = 0.29281105
Iteration 162, loss = 0.29393801
Iteration 163, loss = 0.29189864
Iteration 164, loss = 0.28807453
Iteration 165, loss = 0.29403225
Iteration 166, loss = 0.28118665
Iteration 167, loss = 0.28317904
Iteration 168, loss = 0.28989026
Iteration 169, loss = 0.28646838
Iteration 170, loss = 0.27796494
Iteration 171, loss = 0.28531746
Iteration 172, loss = 0.28207046
Iteration 173, loss = 0.28117502
Iteration 174, loss = 0.28206774
Iteration 175, loss = 0.29045917
Iteration 176, loss = 0.30543444
Iteration 177, loss = 0.28493231
Iteration 178, loss = 0.28428712
Iteration 179, loss = 0.27722737
Iteration 180, loss = 0.28217762
```

```
Iteration 181, loss = 0.28251284
Iteration 182, loss = 0.27618934
Iteration 183, loss = 0.27691964
Iteration 184, loss = 0.28390293
Iteration 185, loss = 0.27963575
Iteration 186, loss = 0.27063410
Iteration 187, loss = 0.27266894
Iteration 188, loss = 0.27162090
Iteration 189, loss = 0.26974672
Iteration 190, loss = 0.26727682
Iteration 191, loss = 0.27270629
Iteration 192, loss = 0.27171512
Iteration 193, loss = 0.27261739
Iteration 194, loss = 0.26650492
Iteration 195, loss = 0.28348722
Iteration 196, loss = 0.26883685
Iteration 197, loss = 0.28647118
Iteration 198, loss = 0.27550712
Iteration 199, loss = 0.26911626
Iteration 200, loss = 0.27067847
Iteration 201, loss = 0.27144764
Iteration 202, loss = 0.27043691
Iteration 203, loss = 0.27505217
Iteration 204, loss = 0.26813297
Iteration 205, loss = 0.27402962
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

Out[22]:

MLPClassifier

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

prediction outputs

In [24]:

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

Out[24]:
```
array([[9.31739650e-001, 6.16362198e-005, 1.09991332e-003,
        2.55270534e-003, 1.68522453e-004, 4.48340385e-005,
        5.84922959e-002, 3.06156353e-006, 5.83530156e-003,
        2.07973552e-006],
       [3.29474679e-017, 9.99999996e-001, 6.60667054e-016,
        2.21178613e-009, 7.37123342e-010, 1.09426641e-018,
        3.54877942e-019, 1.92911295e-010, 4.27977476e-014,
        8.79375645e-010],
       [9.70858118e-003, 7.20311121e-009, 8.43359470e-001,
        4.72993536e-005, 2.03016564e-002, 3.94472669e-048,
        1.22118943e-001, 8.83120683e-054, 4.46404368e-003,
        4.14802867e-046],
       [2.53959745e-001, 1.23272834e-004, 5.74443878e-001,
        3.52174531e-003, 1.54316419e-002, 1.69421160e-022,
        1.36364564e-001, 2.63767586e-023, 1.61551523e-002,
        2.77514620e-021],
       [4.67791878e-007, 4.83021231e-005, 8.23149217e-008,
        9.49788391e-001, 5.00498110e-002, 4.50276023e-021,
        8.76634266e-005, 8.34198438e-032, 2.52823280e-005,
        3.36248482e-022],
       [1.94838052e-001, 6.50671509e-006, 2.15224073e-001,
        9.69319233e-003, 2.44531959e-002, 1.28443419e-023,
        5.39572232e-001, 1.80137231e-027, 1.62127483e-002,
        1.01656748e-023],
       [1.16884545e-002, 2.70195358e-004, 1.96626017e-003,
        7.50145781e-004, 8.78857032e-004, 1.62182051e-003,
        1.26033404e-002, 3.13815864e-004, 9.69829090e-001,
        7.80206680e-005],
       [5.41123163e-002, 5.60647843e-008, 9.60722911e-002,
        3.38233768e-003, 1.97157173e-002, 8.53918910e-031,
        8.19054529e-001, 1.42550510e-037, 7.66275260e-003,
        1.71587813e-031],
       [3.64204720e-045, 0.00000000e+000, 0.00000000e+000,
        0.00000000e+000, 0.00000000e+000, 1.00000000e+000,
        1.25285410e-234, 1.81734777e-024, 5.89834242e-148,
        2.96042830e-046],
       [9.66524548e-001, 3.86900633e-005, 2.58496316e-004,
        2.70698228e-004, 2.80260039e-005, 3.87230539e-006,
        2.93874650e-002, 5.06533899e-007, 3.48747014e-003,
        2.27774937e-007]])
```

In [25]:
```
clf.predict(X_test[:20])

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

Out[25]:
```
array([0, 1, 2, 2, 3, 6, 8, 6, 5, 0, 3, 4, 4, 6, 8, 5, 6, 3, 6, 4],
      dtype=int64)
```

In [26]:
```
y_pred=clf.predict(X_train)
```

Confusion Matrix

In [27]:
```
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
```

Out[27]:
```
array([[5287,    3,   43,  105,    9,    1,  484,    1,   67,    0],
       [   6, 5917,    1,   68,    4,    0,    1,    0,    3,    0],
       [ 147,    1, 4443,   27,  685,    0,  675,    0,   22,    0],
       [ 157,   12,    3, 5638,   99,    1,   81,    1,    7,    1],
       [  28,    3,  215,  343, 4996,    0,  406,    0,    9,    0],
       [   0,    0,    0,    4,    0, 5979,    0,    1,   12,    4],
       [ 899,    1,  216,  114,  207,    0, 4484,    0,   79,    0],
       [   1,    0,    0,    0,    0,   24,    0, 5882,    2,   91],
       [  30,    1,   16,   21,   21,    3,   53,    5, 5850,    0],
       [   0,    0,    0,    0,    0,   13,    1,   72,    1, 5913]],
      dtype=int64)
```

add up all the correct predictions

In [28]:
```
my_cm.trace()
```

Out[28]:
```
54389
```

Sum everything

In [29]:
```
my_cm.sum()
```

Out[29]:
```
60000
```

Now we computer the accuracy Accuracy= (sum along the diagonals)/(sum of all cells)

In [30]:
```
my_cm.trace()/my_cm.sum()
```

Out[30]:
```
0.9064833333333333
```

In [ ]:
```

```

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

In [31]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, max_iter=500,hidden_lay

clf.fit(X_train, y_train_2)
```

```
Iteration 1, loss = 0.33152647
Iteration 2, loss = 0.17161349
Iteration 3, loss = 0.15113460
Iteration 4, loss = 0.14102883
Iteration 5, loss = 0.13372817
Iteration 6, loss = 0.13099708
Iteration 7, loss = 0.12559481
Iteration 8, loss = 0.12497487
Iteration 9, loss = 0.12217216
Iteration 10, loss = 0.12064623
Iteration 11, loss = 0.12009686
Iteration 12, loss = 0.11887566
Iteration 13, loss = 0.11518302
Iteration 14, loss = 0.11347581
Iteration 15, loss = 0.11126001
Iteration 16, loss = 0.11680107
Iteration 17, loss = 0.11417252
Iteration 18, loss = 0.11171394
Iteration 19, loss = 0.11088090
Iteration 20, loss = 0.10844318
Iteration 21, loss = 0.11303384
Iteration 22, loss = 0.10799046
Iteration 23, loss = 0.11087234
Iteration 24, loss = 0.10860177
Iteration 25, loss = 0.10617928
Iteration 26, loss = 0.10862779
Iteration 27, loss = 0.10807904
Iteration 28, loss = 0.10481950
Iteration 29, loss = 0.10633957
Iteration 30, loss = 0.10689126
Iteration 31, loss = 0.10330957
Iteration 32, loss = 0.10418850
Iteration 33, loss = 0.10245793
Iteration 34, loss = 0.11108966
Iteration 35, loss = 0.10064503
Iteration 36, loss = 0.10262237
Iteration 37, loss = 0.09817878
Iteration 38, loss = 0.10227063
Iteration 39, loss = 0.10367949
Iteration 40, loss = 0.10037866
Iteration 41, loss = 0.09818956
Iteration 42, loss = 0.10152246
Iteration 43, loss = 0.09925959
Iteration 44, loss = 0.09844151
Iteration 45, loss = 0.09773979
Iteration 46, loss = 0.09713425
Iteration 47, loss = 0.09773515
Iteration 48, loss = 0.09696000
Iteration 49, loss = 0.09985230
Iteration 50, loss = 0.09642001
Iteration 51, loss = 0.09515176
Iteration 52, loss = 0.09415637
Iteration 53, loss = 0.09353004
Iteration 54, loss = 0.09269436
Iteration 55, loss = 0.09359979
Iteration 56, loss = 0.09487613
Iteration 57, loss = 0.09657351
Iteration 58, loss = 0.09314639
Iteration 59, loss = 0.09137600
Iteration 60, loss = 0.09312757
```

```
Iteration 61, loss = 0.09206497
Iteration 62, loss = 0.09454637
Iteration 63, loss = 0.09237567
Iteration 64, loss = 0.09056524
Iteration 65, loss = 0.09160856
Iteration 66, loss = 0.08852130
Iteration 67, loss = 0.08808569
Iteration 68, loss = 0.08925780
Iteration 69, loss = 0.08733154
Iteration 70, loss = 0.08927723
Iteration 71, loss = 0.08960775
Iteration 72, loss = 0.08852421
Iteration 73, loss = 0.08672168
Iteration 74, loss = 0.08848134
Iteration 75, loss = 0.08735876
Iteration 76, loss = 0.09132535
Iteration 77, loss = 0.08772213
Iteration 78, loss = 0.08615016
Iteration 79, loss = 0.08529329
Iteration 80, loss = 0.08637692
Iteration 81, loss = 0.08673274
Iteration 82, loss = 0.08384119
Iteration 83, loss = 0.08527760
Iteration 84, loss = 0.08630236
Iteration 85, loss = 0.08385789
Iteration 86, loss = 0.08120860
Iteration 87, loss = 0.08479953
Iteration 88, loss = 0.08333714
Iteration 89, loss = 0.08195083
Iteration 90, loss = 0.08154542
Iteration 91, loss = 0.08285526
Iteration 92, loss = 0.08544039
Iteration 93, loss = 0.08114302
Iteration 94, loss = 0.08327020
Iteration 95, loss = 0.08177937
Iteration 96, loss = 0.08322019
Iteration 97, loss = 0.08173104
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

Out[31]:

MLPClassifier

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

In [32]:
```
#Testing how well the model works
```

In [33]:
```
silent_clf = MLPClassifier(solver='adam',activation='relu', alpha=1e-5, random_state=1
```

In [34]:
```
#Import cross validation to do the same thing as the multi_group classifier
```

In [35]:
```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(silent_clf, X_train, y_train_2, cv=3)
```

Get the confusion mattrix

In [36]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_2,y_train_pred)
```

Out[36]:
```
array([[52791,  1209],
       [ 1352,  4648]], dtype=int64)
```

Fraction of correct classification

In [37]:
```
n_correct = sum(y_train_pred == y_train_2)
print(n_correct / len(y_train_pred))
```

```
0.9573166666666667
```

Action

show all the performance measures used in the MNIST digit classifier answer seen in class

In [ ]:

In [ ]:

In [ ]: