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 [47]:
```
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 [48]:
```
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 [49]:
```
dir(test_df)
```

Out[49]:
```
['T',
 '_AXIS_LEN',
 '_AXIS_ORDERS',
 '_AXIS_TO_AXIS_NUMBER',
 '_HANDLED_TYPES',
 '__abs__',
 '__add__',
 '__and__',
 '__annotations__',
 '__array__',
 '__array_priority__',
 '__array_ufunc__',
 '__array_wrap__',
 '__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_by_level',
'_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_axes_from_arguments',
'_construct_result',
'_constructor',
```

```
 '_constructor_sliced',
 '_convert',
 '_count_level',
 '_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',
 '_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',
'_typ',
'_update_inplace',
'_validate_dtype',
'_values',
'_where',
'abs',
'add',
'add_prefix',
'add_suffix',
'agg',
'aggregate',
'align',
'all',
'any',
'append',
'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',
 'iteritems',
 'iterrows',
 'itertuples',
 'join',
 'keys',
 'kurt',
 'kurtosis',
 'label',
 'last',
 'last_valid_index',
 'le',
 'loc',
 'lookup',
 'lt',
 'mad',
 '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',
 'slice_shift',
 '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 [50]:
```
test_df.columns[0:5]
```

Out[50]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4'], dtype='object')
```

In [51]:
```
test_df.shape
```

Out[51]:
```
(10000, 785)
```

In [52]:
```
train_df.shape
```

Out[52]:
```
(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 [53]:
```
y_train=train_df.pop('label')
X_train=train_df
```

In [54]:
```
print(y_train.shape)
print(X_train.shape)
```

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

In [55]:
```
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 [56]:
```
%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 [57]:
```
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 [58]:
```
y_train[0]
```

Out[58]: 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

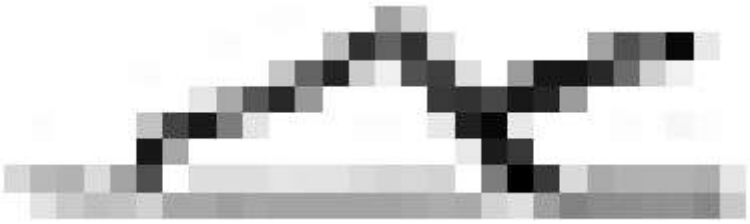
Show images of a sandal and a sneaker from this data set, show them in cells below

Show all your steps

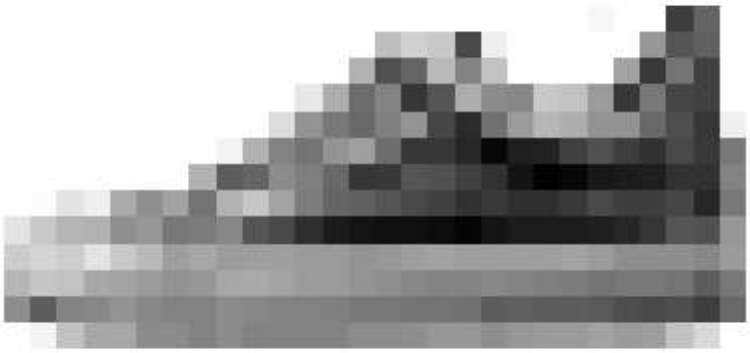
In [59]:
```
#Going to show sandal(5) and a sneaker(7)
#print(y_train[7])
#print(y_train[21])


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 [59]:
```

```

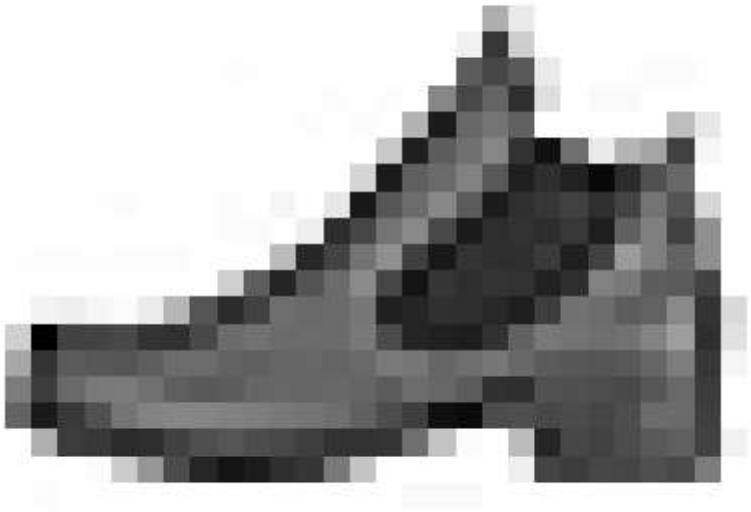
In [60]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

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 [61]:
```
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 [62]:
```
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 [63]:
```
y_train[1]
```

Out[63]:
```
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 [64]:
```
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 [65]:
```
# 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 = 0.74044255
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
Iteration 57, loss = 0.33639833
Iteration 58, loss = 0.33212819
Iteration 59, loss = 0.33651220
Iteration 60, loss = 0.33060810
```

```
Iteration 61, loss = 0.33235731
Iteration 62, loss = 0.32922606
Iteration 63, loss = 0.31956990
Iteration 64, loss = 0.32594625
Iteration 65, loss = 0.33260728
Iteration 66, loss = 0.32091184
Iteration 67, loss = 0.32543906
Iteration 68, loss = 0.31792830
Iteration 69, loss = 0.32275393
Iteration 70, loss = 0.31653049
Iteration 71, loss = 0.31504114
Iteration 72, loss = 0.31700043
Iteration 73, loss = 0.31595449
Iteration 74, loss = 0.31769520
Iteration 75, loss = 0.31051678
Iteration 76, loss = 0.31057300
Iteration 77, loss = 0.31022485
Iteration 78, loss = 0.30475509
Iteration 79, loss = 0.31187765
Iteration 80, loss = 0.31272900
Iteration 81, loss = 0.30669858
Iteration 82, loss = 0.31131854
Iteration 83, loss = 0.30513291
Iteration 84, loss = 0.30145611
Iteration 85, loss = 0.30773963
Iteration 86, loss = 0.30266236
Iteration 87, loss = 0.30250924
Iteration 88, loss = 0.30960456
Iteration 89, loss = 0.29663081
Iteration 90, loss = 0.29794881
Iteration 91, loss = 0.30077294
Iteration 92, loss = 0.29513005
Iteration 93, loss = 0.30074123
Iteration 94, loss = 0.30553552
Iteration 95, loss = 0.29619158
Iteration 96, loss = 0.29730174
Iteration 97, loss = 0.30301800
Iteration 98, loss = 0.29702207
Iteration 99, loss = 0.29071679
Iteration 100, loss = 0.28875091
Iteration 101, loss = 0.28815826
Iteration 102, loss = 0.29667725
Iteration 103, loss = 0.28858403
Iteration 104, loss = 0.29281273
Iteration 105, loss = 0.29162119
Iteration 106, loss = 0.28779132
Iteration 107, loss = 0.29437542
Iteration 108, loss = 0.28801573
Iteration 109, loss = 0.28517978
Iteration 110, loss = 0.28475474
Iteration 111, loss = 0.29587571
Iteration 112, loss = 0.28198834
Iteration 113, loss = 0.28378152
Iteration 114, loss = 0.28040167
Iteration 115, loss = 0.28351153
Iteration 116, loss = 0.28447500
Iteration 117, loss = 0.27898370
Iteration 118, loss = 0.28392275
Iteration 119, loss = 0.28186668
Iteration 120, loss = 0.29216912
```

```
Iteration 121, loss = 0.27642755
Iteration 122, loss = 0.27815964
Iteration 123, loss = 0.28640951
Iteration 124, loss = 0.29923831
Iteration 125, loss = 0.27826349
Iteration 126, loss = 0.27660392
Iteration 127, loss = 0.28540170
Iteration 128, loss = 0.28012941
Iteration 129, loss = 0.27301732
Iteration 130, loss = 0.27765056
Iteration 131, loss = 0.28488949
Iteration 132, loss = 0.28708146
Iteration 133, loss = 0.27596037
Iteration 134, loss = 0.27893109
Iteration 135, loss = 0.27358410
Iteration 136, loss = 0.28228717
Iteration 137, loss = 0.26948778
Iteration 138, loss = 0.26931804
Iteration 139, loss = 0.27816528
Iteration 140, loss = 0.27199058
Iteration 141, loss = 0.26699335
Iteration 142, loss = 0.27148594
Iteration 143, loss = 0.27420168
Iteration 144, loss = 0.27806917
Iteration 145, loss = 0.26651546
Iteration 146, loss = 0.26903612
Iteration 147, loss = 0.26698617
Iteration 148, loss = 0.26907560
Iteration 149, loss = 0.27137525
Iteration 150, loss = 0.28236167
Iteration 151, loss = 0.27661457
Iteration 152, loss = 0.27862800
Iteration 153, loss = 0.26775649
Iteration 154, loss = 0.26651023
Iteration 155, loss = 0.26292505
Iteration 156, loss = 0.25957818
Iteration 157, loss = 0.26638366
Iteration 158, loss = 0.26587732
Iteration 159, loss = 0.27489827
Iteration 160, loss = 0.26387739
Iteration 161, loss = 0.26180335
Iteration 162, loss = 0.27275766
Iteration 163, loss = 0.27047074
Iteration 164, loss = 0.25964435
Iteration 165, loss = 0.25797193
Iteration 166, loss = 0.26011627
Iteration 167, loss = 0.27587736
Iteration 168, loss = 0.27362187
Iteration 169, loss = 0.26210193
Iteration 170, loss = 0.25869749
Iteration 171, loss = 0.27169172
Iteration 172, loss = 0.26053182
Iteration 173, loss = 0.25598939
Iteration 174, loss = 0.26153590
Iteration 175, loss = 0.28045363
Iteration 176, loss = 0.26151651
Iteration 177, loss = 0.27018539
Iteration 178, loss = 0.26291939
Iteration 179, loss = 0.25183456
Iteration 180, loss = 0.26304893
```

```
Iteration 181, loss = 0.26071354
Iteration 182, loss = 0.25314055
Iteration 183, loss = 0.25252226
Iteration 184, loss = 0.26122677
Iteration 185, loss = 0.27055056
Iteration 186, loss = 0.26204419
Iteration 187, loss = 0.26019841
Iteration 188, loss = 0.25223640
Iteration 189, loss = 0.27166005
Iteration 190, loss = 0.25356201
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

Out[65]:

MLPClassifier

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

prediction outputs

In [66]:

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

Out[66]:
```
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]])
```

In [69]:
```
clf.predict(X_test[:20])

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

Out[69]:
```
array([0, 1, 2, 2, 3, 3, 8, 6, 5, 0, 3, 2, 4, 6, 8, 5, 6, 3, 6, 4])
```

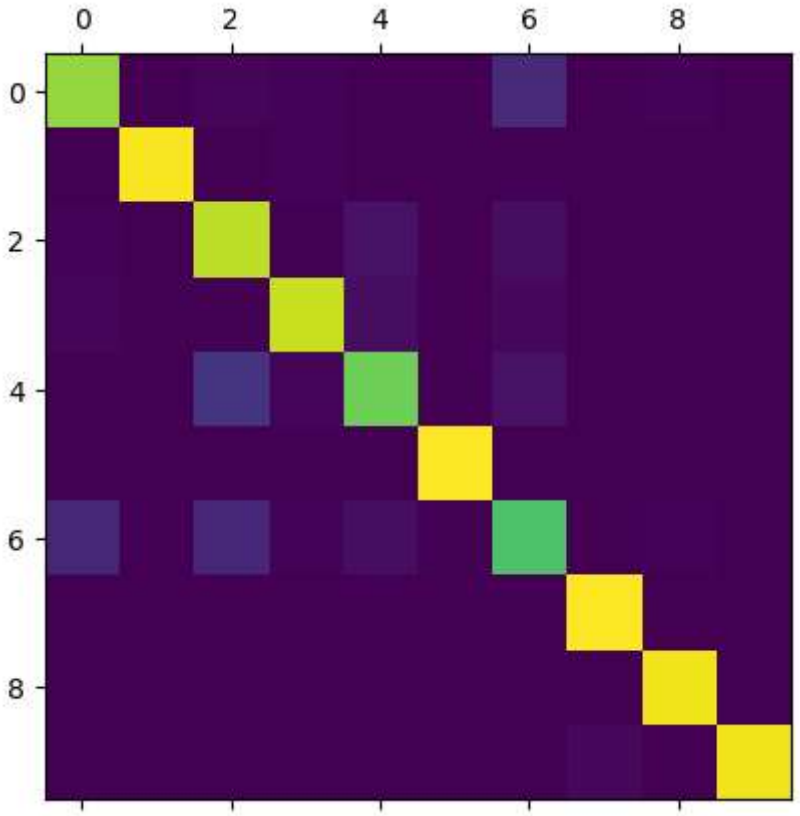
In [71]:
```
y_pred=clf.predict(X_train)
```

Confusion Matrix

In [72]:
```
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[72]:
```
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]])
```

In [73]:
```
#Heatmap
import matplotlib.pyplot as plt
plt.matshow(my_cm)
plt.show()
```



add up all the correct predictions

In [74]:
```
my_cm.trace()
```

Out[74]:
```
54486
```

Sum everything

In [75]:
```
my_cm.sum()
```

Out[75]:
```
60000
```

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

In [76]:
```
my_cm.trace()/my_cm.sum()
```

Out[76]:
```
0.9081
```

In [76]:
```

```

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 [77]:
```
from sklearn.neural_network import MLPClassifier
clf2 = MLPClassifier(solver='adam', alpha=1e-5, random_state=1, max_iter=500,hidden_la

clf2.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 = 0.12399669
Iteration 11, loss = 0.12400881
Iteration 12, loss = 0.11694571
Iteration 13, loss = 0.12049170
Iteration 14, loss = 0.11554896
Iteration 15, loss = 0.11504019
Iteration 16, loss = 0.11585217
Iteration 17, loss = 0.11067332
Iteration 18, loss = 0.11238504
Iteration 19, loss = 0.11160633
Iteration 20, loss = 0.11122248
Iteration 21, loss = 0.10918764
Iteration 22, loss = 0.11131201
Iteration 23, loss = 0.10956015
Iteration 24, loss = 0.10840050
Iteration 25, loss = 0.10640325
Iteration 26, loss = 0.10930399
Iteration 27, loss = 0.10820456
Iteration 28, loss = 0.10477764
Iteration 29, loss = 0.10608269
Iteration 30, loss = 0.10374945
Iteration 31, loss = 0.10535643
Iteration 32, loss = 0.10310087
Iteration 33, loss = 0.10449555
Iteration 34, loss = 0.10225443
Iteration 35, loss = 0.10161436
Iteration 36, loss = 0.10253789
Iteration 37, loss = 0.10156350
Iteration 38, loss = 0.09902311
Iteration 39, loss = 0.09920910
Iteration 40, loss = 0.09874424
Iteration 41, loss = 0.09776275
Iteration 42, loss = 0.10074844
Iteration 43, loss = 0.10310273
Iteration 44, loss = 0.09731159
Iteration 45, loss = 0.09910075
Iteration 46, loss = 0.09966355
Iteration 47, loss = 0.09682798
Iteration 48, loss = 0.09591319
Iteration 49, loss = 0.09747499
Iteration 50, loss = 0.09908632
Iteration 51, loss = 0.09515244
Iteration 52, loss = 0.09369867
Iteration 53, loss = 0.09261250
Iteration 54, loss = 0.09401035
Iteration 55, loss = 0.09223416
Iteration 56, loss = 0.09518279
Iteration 57, loss = 0.09443433
Iteration 58, loss = 0.09355429
Iteration 59, loss = 0.09107517
Iteration 60, loss = 0.09264840
```

```
Iteration 61, loss = 0.09262439
Iteration 62, loss = 0.09214509
Iteration 63, loss = 0.08994946
Iteration 64, loss = 0.09192354
Iteration 65, loss = 0.08918986
Iteration 66, loss = 0.08833911
Iteration 67, loss = 0.08743912
Iteration 68, loss = 0.08985447
Iteration 69, loss = 0.08842975
Iteration 70, loss = 0.08655475
Iteration 71, loss = 0.08901259
Iteration 72, loss = 0.08699165
Iteration 73, loss = 0.08867522
Iteration 74, loss = 0.08572628
Iteration 75, loss = 0.08572075
Iteration 76, loss = 0.08527992
Iteration 77, loss = 0.08786922
Iteration 78, loss = 0.08590756
Iteration 79, loss = 0.08516359
Iteration 80, loss = 0.08470184
Iteration 81, loss = 0.08444050
Iteration 82, loss = 0.08454018
Iteration 83, loss = 0.08285734
Iteration 84, loss = 0.08476819
Iteration 85, loss = 0.08281064
Iteration 86, loss = 0.08484499
Iteration 87, loss = 0.08334495
Iteration 88, loss = 0.08446098
Iteration 89, loss = 0.08453064
Iteration 90, loss = 0.08536418
Iteration 91, loss = 0.08264390
Iteration 92, loss = 0.07989107
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
ing.
```

Out[77]:

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 [78]:
```
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 [79]:
```
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 [80]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_2,y_train_pred)
```

Out[80]:
```
array([[52484,  1516],
       [ 1145,  4855]])
```

Fraction of correct classification

In [81]:
```
n_correct = sum(y_train_pred == y_train_2)
print(n_correct / len(y_train_pred))
```

```
0.95565
```

from sklearn.metrics import precision_score, recall_score# Action show all the performance measures used in the MNIST digit classifier answer seen in class

In [82]:
```
from sklearn.metrics import precision_score, recall_score

#Precision score binary
print(precision_score(y_train_2,y_train_pred))

#Recall Score binary

print(recall_score(y_train_2, y_train_pred))

# F1
from sklearn.metrics import f1_score
f1_score(y_train_2, y_train_pred)



#Multi Group
#Precision score



#print(precision_score(y_train_2,y_train_pred))

#Recall Score

#print(recall_score(y_train_2, y_train_pred))
```

Out[82]:
```
0.7620467744467116
0.8091666666666667
0.784900169751839
```

In [84]:
```
y_scores = cross_val_predict(clf2, X_train, y_train_2, cv=3,
                             method="predict_proba")
```

```
Iteration 1, loss = 0.43249631
Iteration 2, loss = 0.16164563
Iteration 3, loss = 0.14417393
Iteration 4, loss = 0.13376104
Iteration 5, loss = 0.12633499
Iteration 6, loss = 0.12994702
Iteration 7, loss = 0.12125034
Iteration 8, loss = 0.12179675
Iteration 9, loss = 0.12060490
Iteration 10, loss = 0.11688528
Iteration 11, loss = 0.11531744
Iteration 12, loss = 0.11522110
Iteration 13, loss = 0.11412945
Iteration 14, loss = 0.11074945
Iteration 15, loss = 0.10985893
Iteration 16, loss = 0.10734753
Iteration 17, loss = 0.10695205
Iteration 18, loss = 0.10624128
Iteration 19, loss = 0.10467604
Iteration 20, loss = 0.10807539
Iteration 21, loss = 0.10831494
Iteration 22, loss = 0.10465721
Iteration 23, loss = 0.10473355
Iteration 24, loss = 0.10300108
Iteration 25, loss = 0.10179097
Iteration 26, loss = 0.09833465
Iteration 27, loss = 0.09875941
Iteration 28, loss = 0.09842269
Iteration 29, loss = 0.09901449
Iteration 30, loss = 0.09716319
Iteration 31, loss = 0.09527364
Iteration 32, loss = 0.09566552
Iteration 33, loss = 0.09521972
Iteration 34, loss = 0.09381298
Iteration 35, loss = 0.09269918
Iteration 36, loss = 0.09261716
Iteration 37, loss = 0.09001201
Iteration 38, loss = 0.09004859
Iteration 39, loss = 0.08977282
Iteration 40, loss = 0.08833992
Iteration 41, loss = 0.08826957
Iteration 42, loss = 0.08859251
Iteration 43, loss = 0.08969456
Iteration 44, loss = 0.08765409
Iteration 45, loss = 0.08808721
Iteration 46, loss = 0.08564227
Iteration 47, loss = 0.08574329
Iteration 48, loss = 0.08284066
Iteration 49, loss = 0.08701282
Iteration 50, loss = 0.08426783
Iteration 51, loss = 0.08712070
Iteration 52, loss = 0.08444838
Iteration 53, loss = 0.08412404
Iteration 54, loss = 0.08205782
Iteration 55, loss = 0.08236361
Iteration 56, loss = 0.08206422
Iteration 57, loss = 0.08084754
Iteration 58, loss = 0.07975280
Iteration 59, loss = 0.07783345
Iteration 60, loss = 0.08112747
```

```
Iteration 61, loss = 0.08102499
Iteration 62, loss = 0.08074008
Iteration 63, loss = 0.07783111
Iteration 64, loss = 0.07811814
Iteration 65, loss = 0.07544484
Iteration 66, loss = 0.07492987
Iteration 67, loss = 0.07585344
Iteration 68, loss = 0.07534180
Iteration 69, loss = 0.07541964
Iteration 70, loss = 0.07444031
Iteration 71, loss = 0.07259460
Iteration 72, loss = 0.07830913
Iteration 73, loss = 0.07577749
Iteration 74, loss = 0.07475818
Iteration 75, loss = 0.07157549
Iteration 76, loss = 0.07041779
Iteration 77, loss = 0.07378269
Iteration 78, loss = 0.07086114
Iteration 79, loss = 0.07266929
Iteration 80, loss = 0.07212583
Iteration 81, loss = 0.07324141
Iteration 82, loss = 0.06829483
Iteration 83, loss = 0.07177421
Iteration 84, loss = 0.07256212
Iteration 85, loss = 0.07428985
Iteration 86, loss = 0.06924861
Iteration 87, loss = 0.06907457
Iteration 88, loss = 0.06896510
Iteration 89, loss = 0.06957997
Iteration 90, loss = 0.06894689
Iteration 91, loss = 0.07018594
Iteration 92, loss = 0.06856400
Iteration 93, loss = 0.06679575
Iteration 94, loss = 0.06772668
Iteration 95, loss = 0.06742302
Iteration 96, loss = 0.06756913
Iteration 97, loss = 0.06638866
Iteration 98, loss = 0.06582958
Iteration 99, loss = 0.06532889
Iteration 100, loss = 0.06562890
Iteration 101, loss = 0.06959429
Iteration 102, loss = 0.06556843
Iteration 103, loss = 0.06443928
Iteration 104, loss = 0.06338390
Iteration 105, loss = 0.06520040
Iteration 106, loss = 0.06481440
Iteration 107, loss = 0.06151865
Iteration 108, loss = 0.06244771
Iteration 109, loss = 0.06432479
Iteration 110, loss = 0.06437346
Iteration 111, loss = 0.06273867
Iteration 112, loss = 0.06372632
Iteration 113, loss = 0.06280710
Iteration 114, loss = 0.06180369
Iteration 115, loss = 0.06074704
Iteration 116, loss = 0.06163459
Iteration 117, loss = 0.06041897
Iteration 118, loss = 0.05899469
Iteration 119, loss = 0.06226125
Iteration 120, loss = 0.06077810
```

```
Iteration 121, loss = 0.06197306
Iteration 122, loss = 0.06122918
Iteration 123, loss = 0.06288483
Iteration 124, loss = 0.06068103
Iteration 125, loss = 0.05996551
Iteration 126, loss = 0.06006684
Iteration 127, loss = 0.05730984
Iteration 128, loss = 0.05630896
Iteration 129, loss = 0.06532132
Iteration 130, loss = 0.06179318
Iteration 131, loss = 0.06119414
Iteration 132, loss = 0.05796066
Iteration 133, loss = 0.06322680
Iteration 134, loss = 0.05687995
Iteration 135, loss = 0.05710969
Iteration 136, loss = 0.05724452
Iteration 137, loss = 0.05893549
Iteration 138, loss = 0.05780603
Iteration 139, loss = 0.05833082
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
Iteration 1, loss = 0.44268963
Iteration 2, loss = 0.15767688
Iteration 3, loss = 0.14224262
Iteration 4, loss = 0.13483636
Iteration 5, loss = 0.13060669
Iteration 6, loss = 0.12730211
Iteration 7, loss = 0.12490134
Iteration 8, loss = 0.12401171
Iteration 9, loss = 0.12061005
Iteration 10, loss = 0.11788101
Iteration 11, loss = 0.11623798
Iteration 12, loss = 0.11830344
Iteration 13, loss = 0.11563432
Iteration 14, loss = 0.11536668
Iteration 15, loss = 0.11230227
Iteration 16, loss = 0.11819689
Iteration 17, loss = 0.10881809
Iteration 18, loss = 0.11078301
Iteration 19, loss = 0.10439628
Iteration 20, loss = 0.10751629
Iteration 21, loss = 0.10681401
Iteration 22, loss = 0.10325003
Iteration 23, loss = 0.10293796
Iteration 24, loss = 0.10801625
Iteration 25, loss = 0.10576169
Iteration 26, loss = 0.10094043
Iteration 27, loss = 0.09821945
Iteration 28, loss = 0.10265177
Iteration 29, loss = 0.09936623
Iteration 30, loss = 0.09702778
Iteration 31, loss = 0.09612883
Iteration 32, loss = 0.09975923
Iteration 33, loss = 0.10159656
Iteration 34, loss = 0.09862279
Iteration 35, loss = 0.09718282
Iteration 36, loss = 0.09707457
Iteration 37, loss = 0.09533079
Iteration 38, loss = 0.09490079
Iteration 39, loss = 0.09074703
```

```
Iteration 40, loss = 0.09724358
Iteration 41, loss = 0.09195239
Iteration 42, loss = 0.09339005
Iteration 43, loss = 0.09188291
Iteration 44, loss = 0.08991269
Iteration 45, loss = 0.09232884
Iteration 46, loss = 0.08936649
Iteration 47, loss = 0.08914854
Iteration 48, loss = 0.08923117
Iteration 49, loss = 0.08775657
Iteration 50, loss = 0.08988806
Iteration 51, loss = 0.08795033
Iteration 52, loss = 0.08937827
Iteration 53, loss = 0.08475502
Iteration 54, loss = 0.08719188
Iteration 55, loss = 0.09245032
Iteration 56, loss = 0.08641192
Iteration 57, loss = 0.08530056
Iteration 58, loss = 0.09126174
Iteration 59, loss = 0.08617502
Iteration 60, loss = 0.08321009
Iteration 61, loss = 0.08303255
Iteration 62, loss = 0.08225643
Iteration 63, loss = 0.08226426
Iteration 64, loss = 0.08928909
Iteration 65, loss = 0.08101936
Iteration 66, loss = 0.08180634
Iteration 67, loss = 0.08354990
Iteration 68, loss = 0.08101710
Iteration 69, loss = 0.07966241
Iteration 70, loss = 0.08129852
Iteration 71, loss = 0.08012093
Iteration 72, loss = 0.08119902
Iteration 73, loss = 0.07696930
Iteration 74, loss = 0.08058176
Iteration 75, loss = 0.08108856
Iteration 76, loss = 0.07765037
Iteration 77, loss = 0.08893606
Iteration 78, loss = 0.07958387
Iteration 79, loss = 0.08351974
Iteration 80, loss = 0.08722313
Iteration 81, loss = 0.08063036
Iteration 82, loss = 0.07704773
Iteration 83, loss = 0.07603920
Iteration 84, loss = 0.07881772
Iteration 85, loss = 0.07520008
Iteration 86, loss = 0.07596455
Iteration 87, loss = 0.07758488
Iteration 88, loss = 0.07390182
Iteration 89, loss = 0.07509806
Iteration 90, loss = 0.07317563
Iteration 91, loss = 0.07675128
Iteration 92, loss = 0.07466015
Iteration 93, loss = 0.07502234
Iteration 94, loss = 0.07539360
Iteration 95, loss = 0.07261516
Iteration 96, loss = 0.07549967
Iteration 97, loss = 0.08499622
Iteration 98, loss = 0.07734451
Iteration 99, loss = 0.07819483
```

```
Iteration 100, loss = 0.07271044
Iteration 101, loss = 0.07332773
Iteration 102, loss = 0.07191630
Iteration 103, loss = 0.07024985
Iteration 104, loss = 0.07452294
Iteration 105, loss = 0.07388366
Iteration 106, loss = 0.07651458
Iteration 107, loss = 0.06986687
Iteration 108, loss = 0.07012914
Iteration 109, loss = 0.06810230
Iteration 110, loss = 0.07224002
Iteration 111, loss = 0.07167435
Iteration 112, loss = 0.07234839
Iteration 113, loss = 0.06849735
Iteration 114, loss = 0.06992190
Iteration 115, loss = 0.06788028
Iteration 116, loss = 0.07250719
Iteration 117, loss = 0.06889873
Iteration 118, loss = 0.06723496
Iteration 119, loss = 0.06894178
Iteration 120, loss = 0.07301330
Iteration 121, loss = 0.07051698
Iteration 122, loss = 0.06803291
Iteration 123, loss = 0.07124892
Iteration 124, loss = 0.07520417
Iteration 125, loss = 0.07006056
Iteration 126, loss = 0.06915624
Iteration 127, loss = 0.06884401
Iteration 128, loss = 0.06785771
Iteration 129, loss = 0.06713582
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
Iteration 1, loss = 0.43917443
Iteration 2, loss = 0.17459085
Iteration 3, loss = 0.15283754
Iteration 4, loss = 0.14030890
Iteration 5, loss = 0.13235937
Iteration 6, loss = 0.12723095
Iteration 7, loss = 0.12130584
Iteration 8, loss = 0.11828551
Iteration 9, loss = 0.11642485
Iteration 10, loss = 0.11422811
Iteration 11, loss = 0.11324301
Iteration 12, loss = 0.11231087
Iteration 13, loss = 0.10965236
Iteration 14, loss = 0.10548198
Iteration 15, loss = 0.10496361
Iteration 16, loss = 0.10159056
Iteration 17, loss = 0.09998656
Iteration 18, loss = 0.10380645
Iteration 19, loss = 0.10013142
Iteration 20, loss = 0.09729550
Iteration 21, loss = 0.09842358
Iteration 22, loss = 0.09660353
Iteration 23, loss = 0.09319911
Iteration 24, loss = 0.09368904
Iteration 25, loss = 0.09384041
Iteration 26, loss = 0.09110548
Iteration 27, loss = 0.09199763
Iteration 28, loss = 0.09007049
```

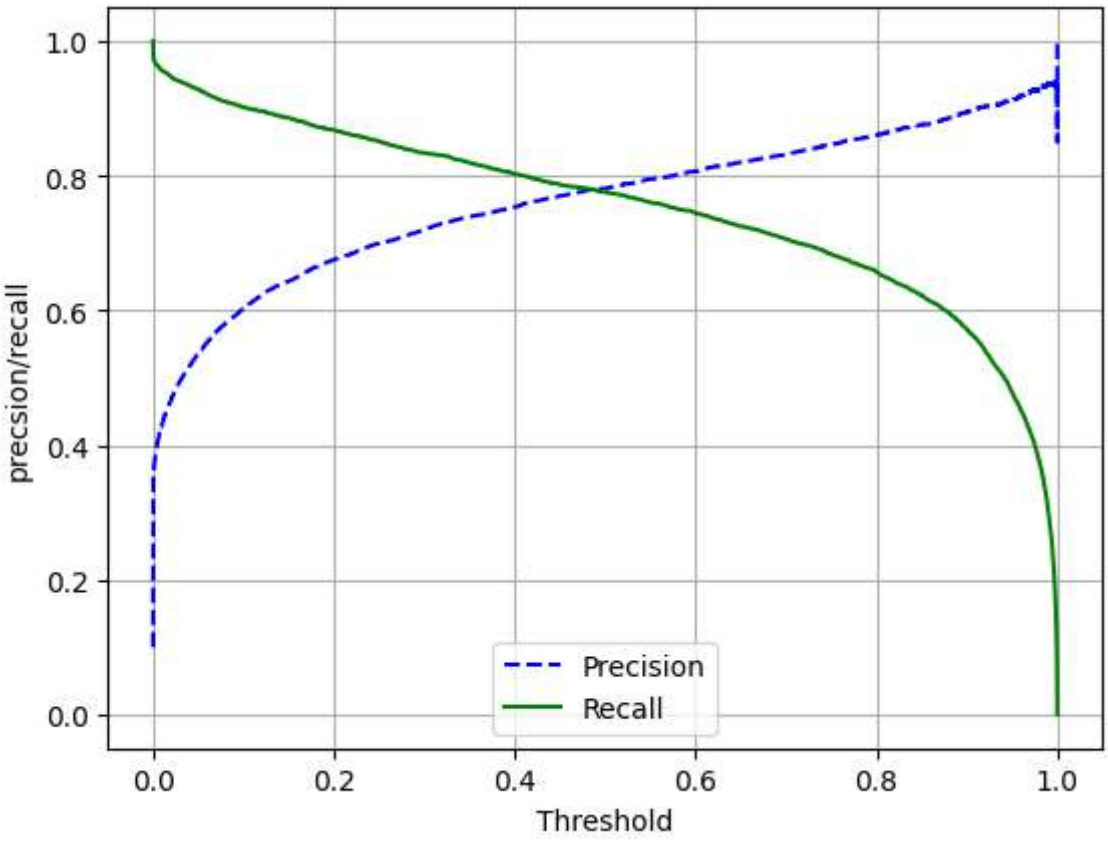
```
Iteration 29, loss = 0.09085651
Iteration 30, loss = 0.08998927
Iteration 31, loss = 0.09208888
Iteration 32, loss = 0.09035250
Iteration 33, loss = 0.09289944
Iteration 34, loss = 0.08626202
Iteration 35, loss = 0.08421344
Iteration 36, loss = 0.08992927
Iteration 37, loss = 0.08356255
Iteration 38, loss = 0.08739251
Iteration 39, loss = 0.09003292
Iteration 40, loss = 0.08815914
Iteration 41, loss = 0.08522018
Iteration 42, loss = 0.09059338
Iteration 43, loss = 0.08420079
Iteration 44, loss = 0.08130804
Iteration 45, loss = 0.08061869
Iteration 46, loss = 0.08267426
Iteration 47, loss = 0.08331593
Iteration 48, loss = 0.07955330
Iteration 49, loss = 0.07964667
Iteration 50, loss = 0.08043650
Iteration 51, loss = 0.07956757
Iteration 52, loss = 0.08261783
Iteration 53, loss = 0.07883278
Iteration 54, loss = 0.08155018
Iteration 55, loss = 0.08017742
Iteration 56, loss = 0.07865698
Iteration 57, loss = 0.07739598
Iteration 58, loss = 0.07814972
Iteration 59, loss = 0.07773701
Iteration 60, loss = 0.07656106
Iteration 61, loss = 0.07412289
Iteration 62, loss = 0.07384815
Iteration 63, loss = 0.07498528
Iteration 64, loss = 0.07766341
Iteration 65, loss = 0.07437205
Iteration 66, loss = 0.07247486
Iteration 67, loss = 0.07193145
Iteration 68, loss = 0.07909193
Iteration 69, loss = 0.07368185
Iteration 70, loss = 0.07041849
Iteration 71, loss = 0.07592337
Iteration 72, loss = 0.07321187
Iteration 73, loss = 0.07474143
Iteration 74, loss = 0.07204428
Iteration 75, loss = 0.07374104
Iteration 76, loss = 0.06881383
Iteration 77, loss = 0.07087028
Iteration 78, loss = 0.07043172
Iteration 79, loss = 0.06893292
Iteration 80, loss = 0.06744466
Iteration 81, loss = 0.07033688
Iteration 82, loss = 0.06796050
Iteration 83, loss = 0.06861827
Iteration 84, loss = 0.06810827
Iteration 85, loss = 0.06816717
Iteration 86, loss = 0.06722704
Iteration 87, loss = 0.06345571
Iteration 88, loss = 0.06724535
```

```
Iteration 89, loss = 0.06674669
Iteration 90, loss = 0.06663698
Iteration 91, loss = 0.06851118
Iteration 92, loss = 0.06484172
Iteration 93, loss = 0.06634592
Iteration 94, loss = 0.06539022
Iteration 95, loss = 0.06867486
Iteration 96, loss = 0.06150902
Iteration 97, loss = 0.06091265
Iteration 98, loss = 0.06403464
Iteration 99, loss = 0.06734213
Iteration 100, loss = 0.06054349
Iteration 101, loss = 0.06214333
Iteration 102, loss = 0.05982194
Iteration 103, loss = 0.06019965
Iteration 104, loss = 0.06078993
Iteration 105, loss = 0.05947618
Iteration 106, loss = 0.06161618
Iteration 107, loss = 0.05992462
Iteration 108, loss = 0.05956563
Iteration 109, loss = 0.06108409
Iteration 110, loss = 0.05951509
Iteration 111, loss = 0.05945800
Iteration 112, loss = 0.05655967
Iteration 113, loss = 0.05784795
Iteration 114, loss = 0.06189005
Iteration 115, loss = 0.05671205
Iteration 116, loss = 0.05518294
Iteration 117, loss = 0.05638035
Iteration 118, loss = 0.05877867
Iteration 119, loss = 0.05900164
Iteration 120, loss = 0.05871186
Iteration 121, loss = 0.05691848
Iteration 122, loss = 0.05694358
Iteration 123, loss = 0.05464538
Iteration 124, loss = 0.05540512
Iteration 125, loss = 0.05648246
Iteration 126, loss = 0.05748399
Iteration 127, loss = 0.05625556
Iteration 128, loss = 0.05742653
Iteration 129, loss = 0.05375491
Iteration 130, loss = 0.05690617
Iteration 131, loss = 0.05421513
Iteration 132, loss = 0.05758323
Iteration 133, loss = 0.05279493
Iteration 134, loss = 0.05180295
Iteration 135, loss = 0.05386430
Iteration 136, loss = 0.05484135
Iteration 137, loss = 0.06635445
Iteration 138, loss = 0.05679234
Iteration 139, loss = 0.05922675
Iteration 140, loss = 0.05999864
Iteration 141, loss = 0.05192673
Iteration 142, loss = 0.05112176
Iteration 143, loss = 0.05148466
Iteration 144, loss = 0.05510470
Iteration 145, loss = 0.04880725
Iteration 146, loss = 0.05108049
Iteration 147, loss = 0.05003214
Iteration 148, loss = 0.05185616
```

```
Iteration 149, loss = 0.05170106
Iteration 150, loss = 0.04961976
Iteration 151, loss = 0.05169553
Iteration 152, loss = 0.05100017
Iteration 153, loss = 0.05179447
Iteration 154, loss = 0.05042182
Iteration 155, loss = 0.04942131
Iteration 156, loss = 0.05294189
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
ing.
```

In [91]:
```
#Precision-recall curve

from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_2, y_scores[:,1])
```

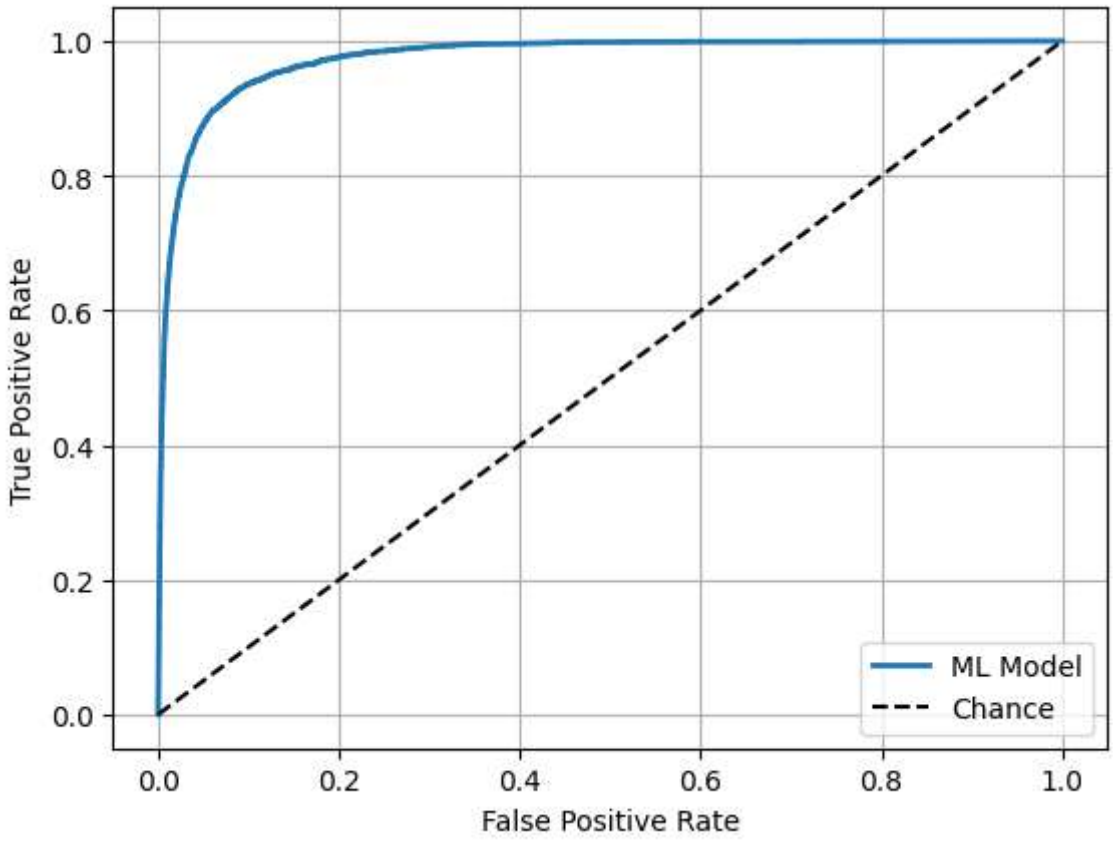
In [86]:
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.legend()
    plt.grid()
    plt.ylabel("precsion/recall")
    plt.xlabel("Threshold")
    [...] # highlight the threshold and add the legend, axis label, and grid

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



The receiver-operator curve

In [88]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_2, y_scores[:,1])
```

In [89]:
```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label="ML Model")
    plt.plot([0, 1], [0, 1], 'k--',label="Chance") # Dashed diagonal
    plt.legend() # Add axis labels and grid
    plt.grid()
    plt.ylabel("True Positive Rate")
    plt.xlabel("False Positive Rate")

plot_roc_curve(fpr, tpr)
plt.show()
```



In [90]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_2, y_scores[:,1])
```

Out[90]:
```
0.9746333580246912
```