Hands on Machine Learning, Building a Neural Net Model

Chapter 3 from "Hands on Maching Learning with Scikit-learn, Keras and Tensorflow" by Geron

Using the MNIST numbers set

Using a simple perceptron neural net system

Updated January 2023

Yes, you are being thrown into the deep end of the pool

The lecture material for the coming week will start to explain this in more detail, if you feel lost during tonight's exercise, it's not your fault, but mine.

Work through this exercise with your pair programming partner, ask questions, talk to me, do some reading.

You will be able to then relate what happened tonight to the lectures and to the next homework.

Step 0: load a classic example data set and understand how it is formated or structured

This is a data prep phase

Note- Python starts array indexes with zero every time!

Second Note- I use the terms scikit-learn and the short version of the name sklearn as well. This is the same package, we'll see a lot of it.

Load the data set, same tactic as in the text

This is using one of the standard datasets in ML, the MNist numbers set. We are using an scikit-learn (sklearn) utility to load it.

Note the import statement here to load the package- this will run slowly the first on a given computer, it download the data from a repository

In [3]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
mnist.keys()

#'DESCR' is a description of the dataset
#'data' is the input data (usually a 2D NumPy array
#'target' is the labels were trying to classify to
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

Out[3]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

Many software tools come with these built in datasets that are really helpful in learning how the tools work

It is always important to understand how these example data sets are structured, because it will be necessary to format new data into the form need for the software tool. This is typically the most tedious, time consuming and annoying part of the process.

When looking at examples, we really need to understand the data formats.

In many packages or libraries used in R and Python, much of the package actually consists of definitions of storage classes or objects to make data handling easier for the calculations done in the package- so you will see new storage classes constantly, be aware of them

In [4]:
```
# ?mnist
```

Question/Action

Okay, a bunch is kind of an odd object

How does it appear to relate to a database like MongoDB? or other systems that use JSON like data storage

Try to connect novel data formats to formats you are more familiar with. Despite all the different names and formats, there are only a few underlying conceptual ideas.

In [5]:
```
#Bunch objects are dictionaries, whose entries can
```

What does the data look like? Here we assign the data to X and the target (aka label) to y

In [6]:
```
X,y=mnist["data"],mnist["target"]
print(X.shape)
print(y.shape)
```

```
(70000, 784)
(70000,)
```

In [7]:
```
for items in mnist:
print(items)
```

```
data
target
frame
categories
feature_names
target_names
DESCR
details
url
```

In [8]:
```
?X
```

It is in linear arrays of 784 values, these are 28 by 28 images, stored as linear vectors. Each line is a 28x28 image black and white image.

We have 70,000 data points (images) one per line in the variable X, X are the predictor variables for this system, which are actually images.

Y are the targets or labels

Use the magic command (which is Jupyter notebook command, rather than a Python command) %matplotlib inline, which causes the Python matplotlib package functions to plot to the Jupyter Notebook - you will want to use this command a lot.

In [9]:
```
%matplotlib inline
```

We'll plot a couple of the images, by reshaping them to 28 x 28 matrices and then using imshow to plot them

There is a convention for storing images in rows like this.

The image we have here are black and white and have only one color channel (black to white). Color images have three channels (aka layers), R, G, B.

Images are stored in 3 D matrices, usually as indexed as [channel, row, column]

For the MNIST digit images we have images that are 1 x 28 x 28

Below we convert on image on a row of the data matrix X into a 28 by 28 matrix we can plot

See:

https://machinelearningmastery.com/a-gentle-introduction-to-channels-first-and-channels-last-image-formats-for-deep-learning/

Note the use of the reshape function, were reshaped a 1 by 784 linear structure into a 28 by 28 matrix. There are a number of ways of doing this

In [10]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

# X is a pandas data frame,   X.iloc[0] is a pandas
#  row zero of the dataframe, we then force it to b
# into a matrix
```

```
some_digit = X.iloc[0].to_numpy()
some_digit_image = some_digit.reshape(28, 28)

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

What is the target for this image?

In [11]:
```
y[0] #the targeted image for the first item in the
```

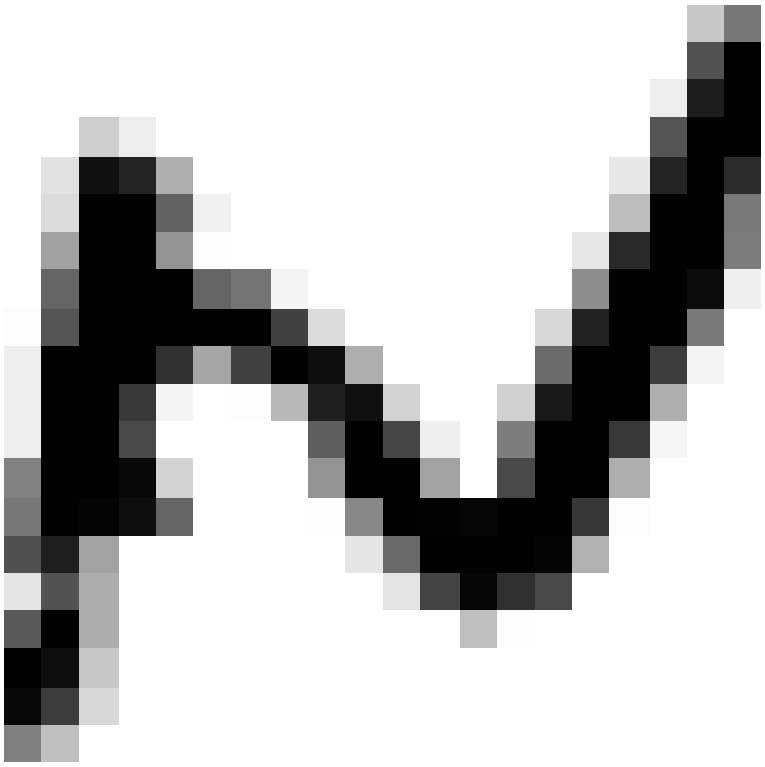
Out[11]:
```
'5'
```

Question/Action

Use google to look up the reshape function

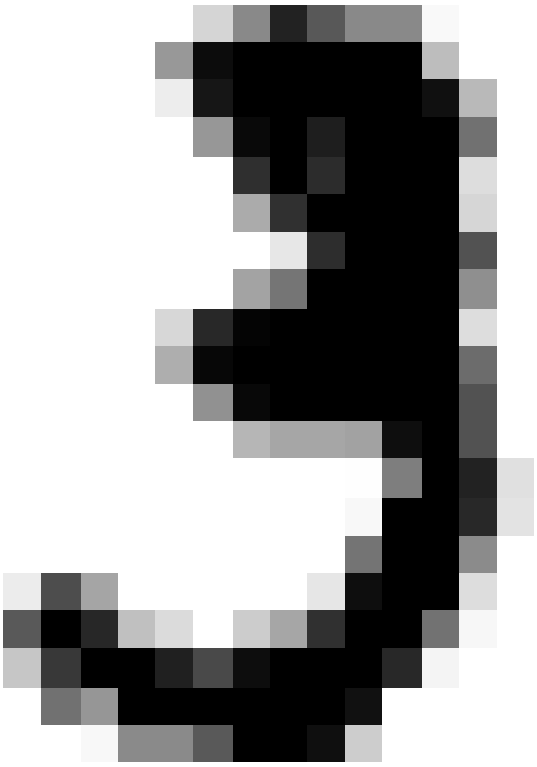
What options are there on how to carry out the reshaping? copy the cell above that does the reshaping and plotting and use a different option for reshaping. What happens?

In [12]:
```
# show variation in reshaping here
some_digit = X.iloc[0].to_numpy()
some_digit_image = some_digit.reshape(28, 28, order
plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



At this point, make some changes to look at some other images, pick and image number at random and plot it.

In [13]:
```
some_digit = X.iloc[10].to_numpy()
some_digit_image = some_digit.reshape(28, 28, order
plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```



Notice here that the target or label, y, is an integer value.

mnist["feature_names"] should show us the names associated with the integer coded target numbers, print out the feature names

In [14]:
```
mnist["feature_names"][0:10]
```

Out[14]:
```
['pixel1',
 'pixel2',
 'pixel3',
 'pixel4',
 'pixel5',
 'pixel6',
 'pixel7',
 'pixel8',
 'pixel9',
 'pixel10']
```

This is a manual split of the data into a training set and a test set. We will only use the test set at the very end to look the performance of the system. This is common in machine learning, using a test set to characterize how well the system works.

Here the first 60,000 points are used as a training set

Note: Using the first 60,000 as training and the rest as a test is not a good way to do this. The data should be randomly split, not sequentially- think about why--

Later, we will see some better tools for creating test and train sets

In [15]:
```
#Note that the data had already been randomly order
    # (we want each fold to have a equal representa

X_train, X_test, y_train, y_test = X[:60000], X[600
```

We are doing a binary classification, so the outcome is true or false

We could do a more complex classification, into all 10 categories, but that's a more complex task

We will create a neural net that can recognize the image of a number "5"

the label or target y is currently a set of numbers 0 to 9, we want a set of true and false labels, indicating wheter the object is a '5' or not

In [16]:
```
y_train_5 = (y_train == '5')  # True for all 5s, Fa
y_test_5 = (y_test == '5')
```

Part 1: Construction of a neural net classification Model

Followed by model training and then a bit of a look at the object that holds the classifier model

We are going to use a neural net, a perceptron classifier from sklearn

A number of issues to discuss here-

a.) This is a simple Machine Learning Classifier, using 3 hidden layers, each with 20, 10 and 5 neurons respectively. Note that input layer must have 784 neurons to handle the input data, while the output layer will have 2, the probabilities of each output category, true or false

b.) MLPClassifier is a more limited approach than using Tensorflow, but it does work reasonable

Some details:

The MLPClassifier sets the number of input neurons to be equal to the number of variables in the input data, in this case we will have 784 input neurons

The list of hidden layer sizes input to the function indicates we want 3 hidden layers of sizes 20,10,5 neurons respectively. I got better performance using 100,50, 20 or so, you may want to try altering or tinkering with the number of neurons per layer or the number of layers or both. The number of neurons and layers are called "hyper-parameters" of the model. Training will take much longer with a bigger network.

The output layer of the net work will have a number of outputs equal to the number of classes used.

The verbose=True entry indicates that the training procedure will give us an a constant updating on progress, indicating the "loss" which is the value of the objective or error function being used. Most classifiers use the cross-entropy objective (loss) function, which is what is being used here

The solver is "adam" which is one implementation of a gradient descent,

alpha is an L2 regularization parameter- more on regularization later- this is meant to reduce overfitting problems

warm start=False indicates the training starts from a new random set of parameters (weights) instead of starting from the results of the last training operation by the member function fit.

The learning rate, tolerance and other parameters are all set to the default values, see the sklearn manual below for more information

There is a nice discussion of the model at

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Look at the website above before you run the model

Here we create an instance of a classifier with the desired structure and operating parameters

We then fit the model to the training predictors X_train and the labels (or targets) y_train_5

Sklearn models all work this way

The fitting process will run rather slowly, it will typically end at roughly 100 iterations, maybe 2 to 5 minutes run time at maximum

In [17]:
```
#Note: There are 3 hidden layers of 20, 10, and 5 n
#the input layer must have 784 neurons to handle al
#the output only has 2 true or false

from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, rand

clf.fit(X_train, y_train_5)
```

```
Iteration 1, loss = 0.27028725
Iteration 2, loss = 0.08038020
Iteration 3, loss = 0.05730453
Iteration 4, loss = 0.04481164
Iteration 5, loss = 0.03709973
Iteration 6, loss = 0.03257554
Iteration 7, loss = 0.02849133
Iteration 8, loss = 0.02612611
Iteration 9, loss = 0.02331228
Iteration 10, loss = 0.02080288
Iteration 11, loss = 0.02111216
Iteration 12, loss = 0.01944051
Iteration 13, loss = 0.01709165
Iteration 14, loss = 0.01835985
Iteration 15, loss = 0.01533054
Iteration 16, loss = 0.01499016
Iteration 17, loss = 0.01485069
Iteration 18, loss = 0.01697578
Iteration 19, loss = 0.01363250
Iteration 20, loss = 0.01286795
Iteration 21, loss = 0.01432031
Iteration 22, loss = 0.01220789
Iteration 23, loss = 0.01094152
Iteration 24, loss = 0.01048439
Iteration 25, loss = 0.01001958
Iteration 26, loss = 0.00931842
Iteration 27, loss = 0.00888720
Iteration 28, loss = 0.00712077
Iteration 29, loss = 0.00775784
Iteration 30, loss = 0.00693138
Iteration 31, loss = 0.01122541
Iteration 32, loss = 0.00837785
Iteration 33, loss = 0.00878483
Iteration 34, loss = 0.00721910
Iteration 35, loss = 0.00620920
Iteration 36, loss = 0.00835780
Iteration 37, loss = 0.00663585
Iteration 38, loss = 0.00509979
Iteration 39, loss = 0.00793709
Iteration 40, loss = 0.00806922
```

```
Iteration 41, loss = 0.00566000
Iteration 42, loss = 0.00561217
Iteration 43, loss = 0.00445315
Iteration 44, loss = 0.00644081
Iteration 45, loss = 0.00445338
Iteration 46, loss = 0.00477962
Iteration 47, loss = 0.00533793
Iteration 48, loss = 0.00461763
Iteration 49, loss = 0.00614947
Iteration 50, loss = 0.00336004
Iteration 51, loss = 0.00462674
Iteration 52, loss = 0.00398499
Iteration 53, loss = 0.00417663
Iteration 54, loss = 0.00506586
Iteration 55, loss = 0.00395338
Iteration 56, loss = 0.00315240
Iteration 57, loss = 0.00383450
Iteration 58, loss = 0.00464641
Iteration 59, loss = 0.00210746
Iteration 60, loss = 0.00181633
Iteration 61, loss = 0.00464446
Iteration 62, loss = 0.00357308
Iteration 63, loss = 0.00286724
Iteration 64, loss = 0.00264295
Iteration 65, loss = 0.00261309
Iteration 66, loss = 0.00293135
Iteration 67, loss = 0.00229525
Iteration 68, loss = 0.00312292
Iteration 69, loss = 0.00319813
Iteration 70, loss = 0.00313642
Iteration 71, loss = 0.00179899
Training loss did not improve more than tol=0.00010
0 for 10 consecutive epochs. Stopping.
```

Out[17]:

▾ MLPClassifier

```
MLPClassifier(alpha=1e-05, hidden_layer_size
s=(20, 10, 5), max_iter=500,
              random_state=1, verbose=True)
```

In [18]:
```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, rand
clf.fit(X_train, y_train_5)
```

Out[18]:
```
MLPClassifier(alpha=1e-05, max_iter=500, ran
dom_state=1)
```

clf is an object, an instance of the class MLPClassifier, which has a lot of stored data, like the model structure, all the fitted parameter values

We can use the dir() member function of clf to see what member functions and variables are within the clf object

In [19]:
```
dir(clf)
```

Out[19]:
```
['__abstractmethods__',
 '__annotations__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_impl',
 '_backprop',
 '_check_feature_names',
 '_check_n_features',
 '_check_solver',
 '_compute_loss_grad',
 '_estimator_type',
 '_fit',
 '_fit_lbfgs',
 '_fit_stochastic',
```

```
'_forward_pass',
'_forward_pass_fast',
'_get_param_names',
'_get_tags',
'_init_coef',
'_initialize',
'_label_binarizer',
'_loss_grad_lbfgs',
'_more_tags',
'_no_improvement_count',
'_optimizer',
'_parameter_constraints',
'_predict',
'_random_state',
'_repr_html_',
'_repr_html_inner',
'_repr_mimebundle_',
'_score',
'_unpack',
'_update_no_improvement_count',
'_validate_data',
'_validate_input',
'_validate_params',
'activation',
'alpha',
'batch_size',
'best_loss_',
'best_validation_score_',
'beta_1',
'beta_2',
'classes_',
'coefs_',
'early_stopping',
'epsilon',
'feature_names_in_',
'fit',
'get_params',
'hidden_layer_sizes',
'intercepts_',
'learning_rate',
```

```
 'learning_rate_init',
 'loss',
 'loss_',
 'loss_curve_',
 'max_fun',
 'max_iter',
 'momentum',
 'n_features_in_',
 'n_iter_',
 'n_iter_no_change',
 'n_layers_',
 'n_outputs_',
 'nesterovs_momentum',
 'out_activation_',
 'partial_fit',
 'power_t',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'random_state',
 'score',
 'set_params',
 'shuffle',
 'solver',
 't_',
 'tol',
 'validation_fraction',
 'validation_scores_',
 'verbose',
 'warm_start']
```

In [20]:
```
clf.hidden_layer_sizes
```

Out[20]:
```
(100,)
```

All sklearn model classes have a fit() and a predict() member function, we can predict values using a trained model using the predict member function

Question/Action

What type of object is X_test?

I had to change the code in the line below

```
clf.predict(X_test.iloc[:10,:])
```

When I ran it in January 2022, it was

```
clf.predict(X_test[:10,:])
```

Why would code that ran fine in 2022 no longer work in 2023

In [21]:
```
# clf.predict(X_test[:10,:])
# NeuralNetwork model from skilearns was updated  w
```

In [22]:
```
clf.predict(X_test.iloc[:10,:])
```

Out[22]:
```
array([False, False, False, False, False, False, Fa
lse, False, False,
       False])
```

In [23]:
```
# run code here - what type of data is X_test
print('X_test is this data type', type(X_test))

# why doesn't X_test[:10,:] work, but X_test.iloc[:

# Answer:
# ``.iloc[]`` is primarily integer position based (
```

```
# ``length-1`` of the axis), but may also be used w
# array.
```

```
X_test is this data type <class 'pandas.core.frame.
DataFrame'>
```

Predictions

We have trained the model in clf, we will now use it to predict the class of our first 10 test images

In [24]:
```
clf.predict(X_test.iloc[:10,:])
```

Out[24]:
```
array([False, False, False, False, False, False, Fa
lse, False, False,
       False])
```

Notice that clf.predict gave us true/false values, assignments to specific categories of the number equaling 5 or not

These are some options, we can get a probability estimate

In [25]:
```
clf.predict_proba(X_test.iloc[:10,:])
```

Out[25]:
```
array([[1.00000000e+000, 1.97390010e-094],
       [1.00000000e+000, 8.62378066e-068],
       [1.00000000e+000, 1.51868716e-063],
       [1.00000000e+000, 2.68887644e-039],
       [1.00000000e+000, 2.87660591e-147],
       [1.00000000e+000, 9.56756319e-110],
       [1.00000000e+000, 5.21066851e-075],
       [1.00000000e+000, 7.25272747e-025],
       [1.00000000e+000, 6.01487755e-016],
       [1.00000000e+000, 8.91301907e-140]])
```

we can now look at the number of iterations used in fitting the model

In [26]:
```
clf.n_iter_
```

Out[26]:
```
85
```

Question

Find out what the activation function was, what the tolerance value was, and the learning rate was

In [27]:
```
dir(clf)
```

Out[27]:
```
['__abstractmethods__',
 '__annotations__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_abc_impl',
 '_backprop',
 '_check_feature_names',
 '_check_n_features',
 '_check_solver',
 '_compute_loss_grad',
 '_estimator_type',
 '_fit',
 '_fit_lbfgs',
 '_fit_stochastic',
```

```
'_forward_pass',
'_forward_pass_fast',
'_get_param_names',
'_get_tags',
'_init_coef',
'_initialize',
'_label_binarizer',
'_loss_grad_lbfgs',
'_more_tags',
'_no_improvement_count',
'_optimizer',
'_parameter_constraints',
'_predict',
'_random_state',
'_repr_html_',
'_repr_html_inner',
'_repr_mimebundle_',
'_score',
'_unpack',
'_update_no_improvement_count',
'_validate_data',
'_validate_input',
'_validate_params',
'activation',
'alpha',
'batch_size',
'best_loss_',
'best_validation_score_',
'beta_1',
'beta_2',
'classes_',
'coefs_',
'early_stopping',
'epsilon',
'feature_names_in_',
'fit',
'get_params',
'hidden_layer_sizes',
'intercepts_',
'learning_rate',
```

```
 'learning_rate_init',
 'loss',
 'loss_',
 'loss_curve_',
 'max_fun',
 'max_iter',
 'momentum',
 'n_features_in_',
 'n_iter_',
 'n_iter_no_change',
 'n_layers_',
 'n_outputs_',
 'nesterovs_momentum',
 'out_activation_',
 'partial_fit',
 'power_t',
 'predict',
 'predict_log_proba',
 'predict_proba',
 'random_state',
 'score',
 'set_params',
 'shuffle',
 'solver',
 't_',
 'tol',
 'validation_fraction',
 'validation_scores_',
 'verbose',
 'warm_start']
```

In [28]:
```
clf.hidden_layer_sizes
```

Out[28]:
```
(100,)
```

In [29]:
```
clf.n_features_in_
```

Out[29]:
```
784
```

In [30]:
```
# Question
# How many features where there in the input layer?

# Answer:
# input layer has 784
# the hidden layer size was  (20, 10, 5) 3 hidden l
```

In [30]:
```

```

Part 2: How well does this classifier actually work?

We will use a number of different types of measures and metrics of performance from sklearn

Create a second version of the classifier, but with the verbose option set to false, so it will not show the change in the cross entropy as the network is trained.

In [31]:
```
silent_clf = MLPClassifier(solver='adam',activation
```

Below is a manual cross validation to look at the correct classification rate

We want to estimate how likely the model (predictor) is going to be if we use it on new images

What we are doing is splitting the data into 3 sections (my_split=3) and training silent_clf on 2/3 of the data and testing how well it does on the remaining sections

of the data. The remaining 1/3 the model was not trained on is called the validation data

The performance of the method on the validation data is called a crossed validation estimate. If we estimated the performance of the model on the training data, that would be called a resubstitution estimate of the performance.

If we had totally new, unused data that we use to estimate performance, that is called a test set estimate.

Note- this requires 3 separate fittings of the model, it will be slow

In [32]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

my_splits=3

# this splits the data into the training "fold" and

skfolds = StratifiedKFold(n_splits=my_splits,shuffl


for train_index, test_index in skfolds.split(X_trai
    # make a clone of the model
    clone_clf = clone(silent_clf)
    #set up the test and train sets from the fold o
    X_train_folds = X_train.iloc[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_test_fold = y_train_5[test_index]
    # fit the cloned model to the data and predict
    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
```

```
    # compute the number of times the the predicted
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))
```

```
0.9916
0.99295
0.992
```

In [33]:
```
#Another way to preform the task above
# from sklearn.model_selection import cross_val_sco
# cross_val_score(silent_clf, X_train, y_train_5, c
```

We are up around 99% or higher rate of correct assignments. Pretty cool.

But this is an unbalanced set, 10% of the data are "5"s, 90% are not

If we simply made a predictor that always returned "False" it would be 90% correct

Here is a better way to do this, using a built-in function to do the cross validation, using a built-in sklearn function

Notice here a whole set of predictions created by the model, all in one function call

In [34]:
```
#This is why accuracy is not the best measure espec
# a better approach to this is to use a confusion m
```

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

y_train_pred = cross_val_predict(silent_clf, X_trai
```

```
# the cross_val_predict() performs k-folds cross-va
# these are clean predictions meaning the model is
```

In [36]:
```
y_train_pred
```

Out[36]:
```
array([ True, False, False, ...,  True, False, Fals
e])
```

Personal Question:

Does this mean that y_train_pred will have 3 different fit models?

Bc how can it find the best over all model between the 3 sets without overfitting problems

Question

In general, resubsitution rates are untrustworthy. Why?

We prefer independent test results over cross validation estimates, why is this?

Why are cross validation estimates easier to obtain and work with?

Answer

- resubstitution rate are untrustworthly bc it is training and test the model on the same data that can lead to bias in the accuracy
- we prefer cross validation because this gives folds the data into different training and testing data and will give a better measure of calculating accuracy in the model
- cross validation estimated are easier to obtain and work with because of the nature of folding the data into training and testing based on the desired amount of fold.

Next we will create a confusion matrix to look at the performance of the model

The ideas related to classification metrics, confusion matrices and ROC curves are all discussed in Geron, Chapter 3

https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch03.html#idm45022187098104

There is another discussion in Provost and Fawcett that is very business oriented

https://learning.oreilly.com/library/view/data-science-for/9781449374273/ch08.html#eightdot4_the_area_under_t

A confusion matrix shows the correct label (y) and the predicted category as labels on a matrix, so we can see what types of mistakes the classifier is making and how many

The columns are the predictions, rows are the actual correct classes

Prediction

actual {5} {not 5} {5} | True Positives False Negative | {not 5} | False Positive True Negative |

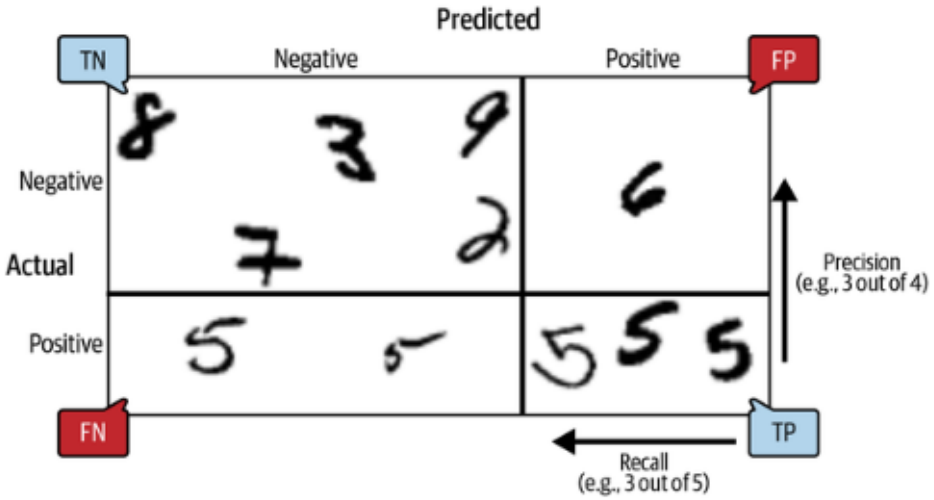
In [37]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_5,y_train_pred)
# Each row explains an actual class (First row Nega
# Each columns explains a predicted class

# True Negatives, False Positives (Type I error)
# False Negatives (Type II error), True Positives
```

Out[37]:
```
array([[54372,   207],
       [  267,  5154]])
```

What is the fraction of correct classifications?

In [38]:
```
n_correct = sum(y_train_pred == y_train_5)
print(n_correct / len(y_train_pred))
```

```
0.9921
```



In [39]:
```
from sklearn.metrics import precision_score, recall

#TP- True positive,   a positive y classed as posit
#FP- False positive,  a negative y classed as a pos

#FN  -False negative,   a positive y classed as neg
#T   -True negative,    a negative classed as a neg

# Precision = TP/(FP+TP)
print(precision_score(y_train_5, y_train_pred))
# When predicting postive item it can correctly ass

#recall,   TP/(TP+FN)
print(recall_score(y_train_5, y_train_pred))
# When looking at the actual postive value it can d
```

```
0.9613878007834359
0.9507470946319867
```

In your own words,

what is Precision?

What is recall?

There are time talking about precision and recall, where you will care more one over the other

For example:

```
# - A trained classifier to YouTube
detecting sensitive information
harmful to kids
# - We would prefer a classifier that
rejects many good videos (low
recall), but only keeps safe ones
(high precision)
```

Alternative example:

```
# - a security alarm that detects
shoplifters
# - We would prefer a lot more false
alarms (low precision), in the
exchange of catching all the
shoplifters (high recall)
```

Remember we can't have it both ways increasing the precision, decreases the reduces the recall, and vice versa

```
# This is called the precision/recall
tradeoff
```

If you can't have a lot of incorrect approvals you need high precision

If you can't let the targeted group go unchecked you need high recall

Part 3: More advanced ideas on describing classifier behavior

We can plot precision and recall as a function of the Threshold

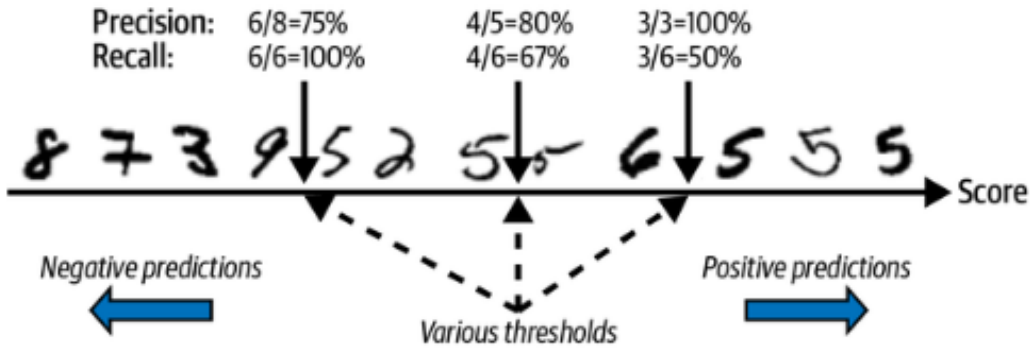
The output of the classifier as a probability value allows us to decide which category to assign each specimen.

We would state a classification rule such as:

If probability of class 1 for a specimen is greater than the threshold level, assign the specimen to class 1, if not assign it to class 2.

Changing the threshold value alters the number of TP, FP, TN an FN results we will see and thus the precision and recall.

Plotting precision and recall is helpful in understanding the model behavior



Note:

- Increasing the threshold increases the precision and decreases the recall

In [42]:
```
y_scores = cross_val_predict(silent_clf, X_train, y
                             method="predict_proba"
```

In [43]:
```
# ?cross_val_predict
```

In [44]:
```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_
```

In [45]:
```
print(precisions)
```

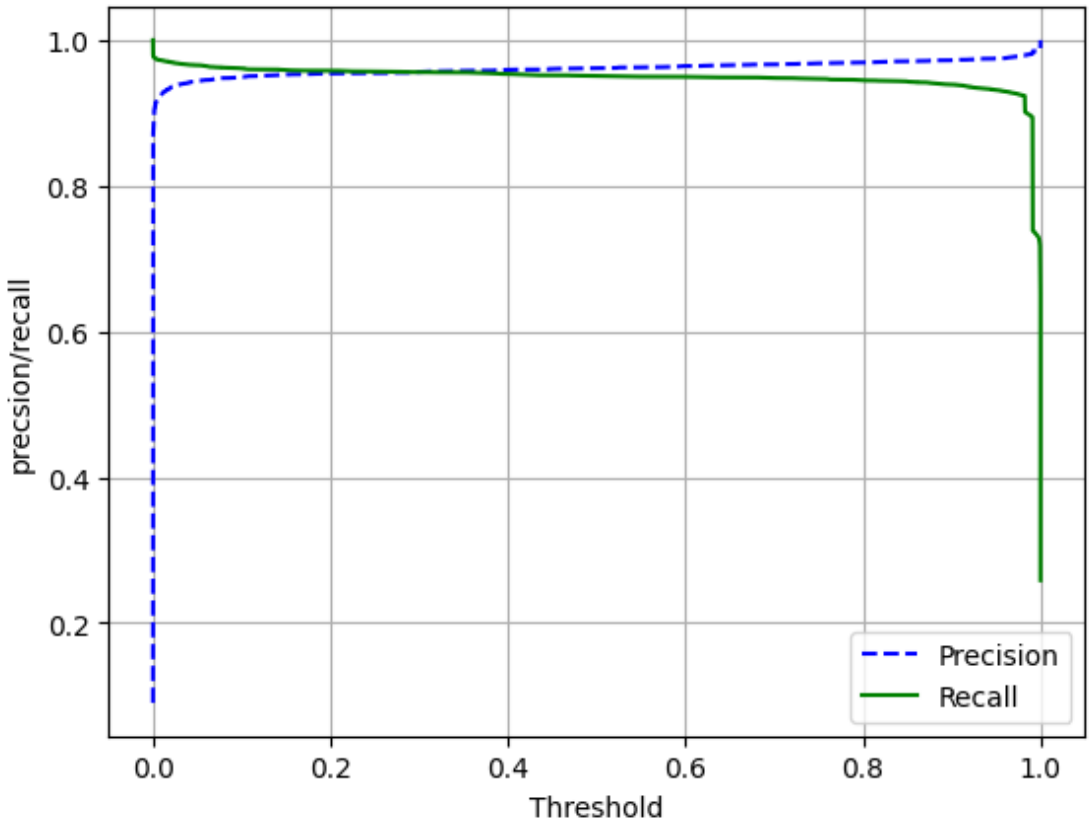
```
[0.09035    0.09035151 0.09035301 ... 1.         1.
1.        ]
```

In [46]:
```
print(recalls)
```

```
[1.         1.         1.         ... 0.27633278 0.
25862387 0.        ]
```

This is a precision-recall curve

In [47]:
```
def plot_precision_recall_vs_threshold(precisions,
    plt.plot(thresholds, precisions[:-1], "b--", la
    plt.plot(thresholds, recalls[:-1], "g-", label=
    plt.legend()
    plt.grid()
    plt.ylabel("precsion/recall")
    plt.xlabel("Threshold")
    [...] # highlight the threshold and add the leg

plot_precision_recall_vs_threshold(precisions, reca
plt.show()
```



The receiver-operator curve

This works great for binary classifiers such as this one

It plots the TP rate verses the FP rate, both are altered by changing the threshold level.

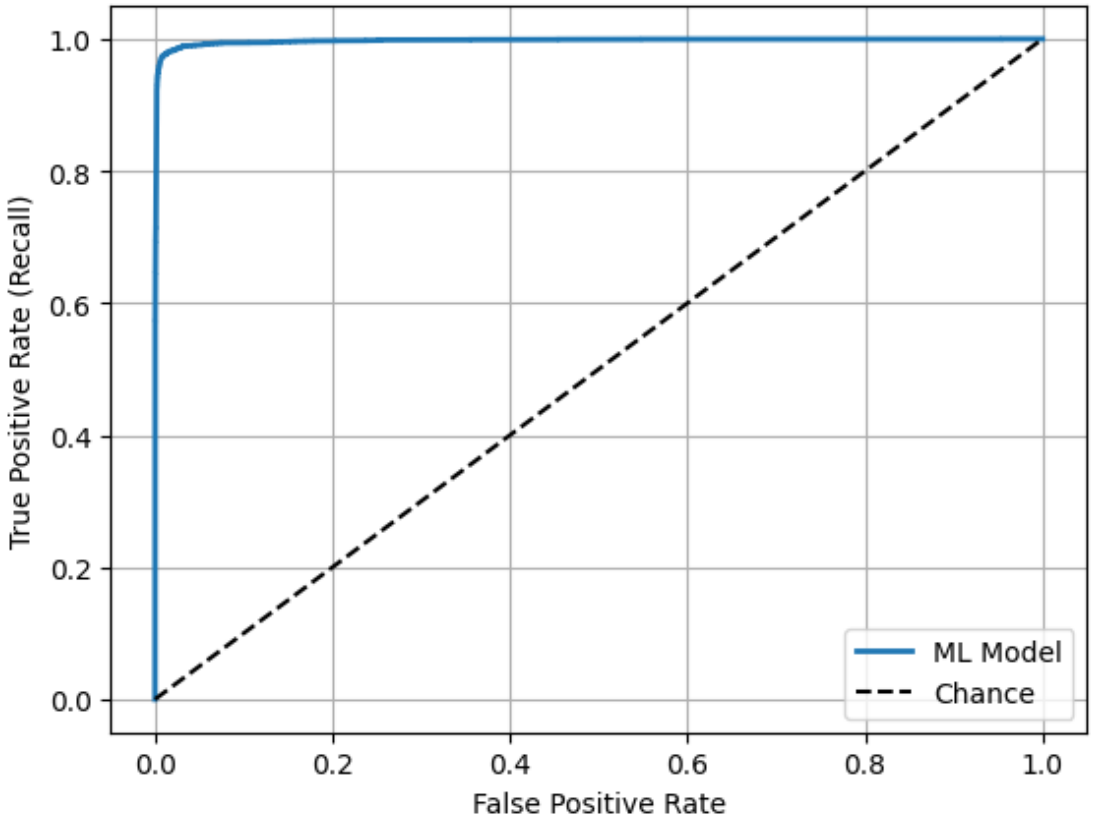
In a perfect model, changing the thresshold will have no effect on the outputs, and the area under the RC curve will be 1.

For a random classification, p=0.5 for all models, and we get the dashed line in the plot, and an area under the curve of 0.5

Here's what this looks like

In [48]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_score
```

In [49]:
```
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")
    plt.legend() # Add axis labels and grid
    plt.grid()
    plt.ylabel("True Positive Rate (Recall)") # Als
    plt.xlabel("False Positive Rate") # Also called

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



This is a really impressive ROC curve, there is a tiny rounding area in the upper right corner, that indicates some reduction in the True Positive Rate when the False Positive rate is zero.

In [50]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores[:,1])
```

Out[50]:
```
0.9972886266964508
```

4 Final Step: evaluate the performance of your best model using the Test Data

Now let's look at the actual test set- this should be the last step in an analysis!

When you have the final model chosen, and you have optimized it and studied it and thrashed the living tar out of the input step, you want to test that final model against the test data, which you have never really looked at.

You only meet a test set for the first time once. It is only "new" data once. Remember that after you look at data, you are in an a-posteriori world, not the a-priori world of new data. (Remember the Monte Hall problem?).

So, a test set is only useable once. After that, it could be training data, but never truly test data again.

Overfitting and Underfitting

If your model is too simple, it doesn't make full use of the information in the predictors X and has less than ideal peformance. Your model has prediction bias, since it misses important features of the data

If your model is too complex, it tries to be two precise or detailed for what you know about the data (ie you drew a regression line through all the points on a graph). THis is called a prediction variance.

You want a model in the sweet spot between prediction and variance.

Predict on the X-test set (only once!) and then characterize performance

In [51]:
```
pred_test=clf.predict(X_test)
```

In [52]:
```
pred_test_prob=clf.predict_proba(X_test)
```

In [53]:
```
confusion_matrix(y_test_5, pred_test)
```

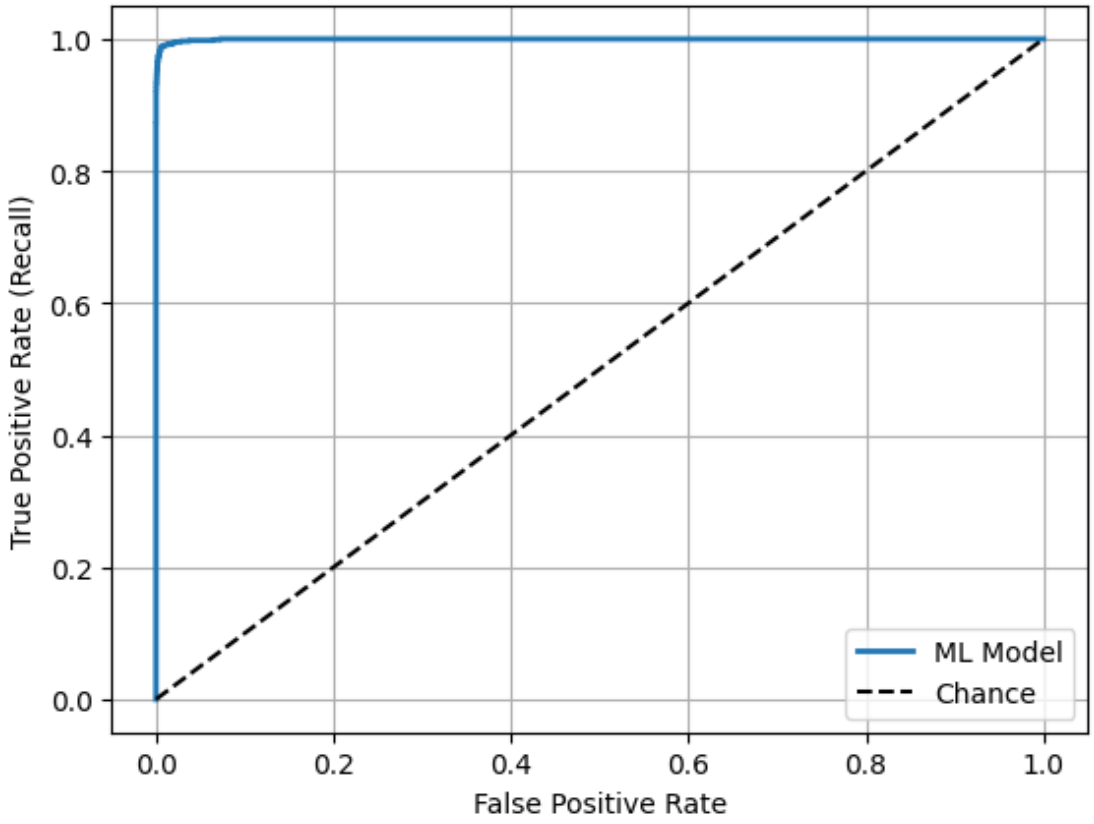
Out[53]:
```
array([[9093,   15],
       [  35,  857]])
```

In [54]:
```
n_correct = sum(y_test_5 == pred_test)
print(n_correct / len(y_test_5))
```

```
0.995
```

In [55]:
```
fpr, tpr, thresholds = roc_curve(y_test_5, pred_tes
```

In [56]:
```
plot_roc_curve(fpr, tpr)
plt.show()
```



In [57]:
```
roc_auc_score(y_train_5, y_scores[:,1])
```

Out[57]:
```
0.9972886266964508
```

Questions

Is there any evidence of overfitting here?

How could we look for signs of bias?

Answer:

- There does not look to be any signs of overfitting of the model. We can conclude this by observing no change in the roc_auc_score for the training data and the test data. If the model was overfit,

meaning there was bias to the training data, we would see a decrease in the roc_auc_score

Coming up on the homework

1.) Use the cells of this example notebook to create a new classifier that uses all ten image classes, meaning using y_train rather than y_train_5. This is a multiple category classifier rather than a binary classifier.

You may want to increase the number of neurons in the model.

Use the accuracy, precision and recall calculations, but not the ROC plot or the precision vs recall plots, those don't work for multiple category models. Do calculate and look at the confusion matrix

2.) Load the MNIST Fashion data set, which is a tougher problem. It is as set of black and white images of pieces of clothing

These are 28 x 28 greyscale images, just as we saw above

Build a neural net classifier for this data set,attempt to optimize it and describe the performance as above. Start with a binary classifier to try to detect one particular clothing item, then go on to try to classify using all ten categories

In [57]: