Earth-Economy Modeling: Fine resolution ecosystems in general equilibrium

Justin Johnson

12/28/22

Table of contents

Introduction

This is a a work in progress that combines code from many different aspects of my reserach, including from land-use change modeling (SEALS), ecosystem services modeling (InVEST), general equilibrium modeling (GTAP), APEC 8222 (Big Data for Economists), APEC 8601 (Natural Resource Economics) and several sources.

It will eventually grow to form the core textbook content for new iterations of theses courses as well as the documentation for the respective models.

Part I

Python

1 Jupyter Intro

Jupyter notebooks are becoming dominant in replicable, open-soure science. A Jupyter notebook combines nicely-formatted conent with images, typesetting, equations, etc. with code blocks that can be directly run. Here, we're going to practice using Jupyter and use it to learn some basics of Python.

1.1 Jupyter has two kinds of cells

1. Markdown cells (like this one)
2. Code cells (like the next one)

Above the editor window here, you can click +Code or +Markdown to add new ones.

Alternatively, right-click on a cell for more options (like splitting a cell into two)

Markdown cells are meant for formatted content, pictures, lecture notes etc. and follows the same notation as R-markdown.

If you want to edit a markdown cell, double left-click it. To finalize the edits, click the checkmark above (or type ctrl-enter).

1.2 Code runs IN the notebook.

Select the python cell below. You can edit it freely. To run it, you can click the triangle button ("play button") to the upper right of that cell. Alternatively, you can ctrl-enter.

```
a = 5
b = 4
simple_summation = a + b
```

You know the cell has run successfully if it gets a green check at the bottom. Also notice that there is a number now in the [ ] box at the bottom left. This indicates which cell this was run in order of all the cells run.

Notice that it doesn't output anything, but note that the values are now stored in the Python Kernel (Jupyter Server) and are available to other parts of this notebook.

If you want to see a variable outputted, you can just type the variable name.

```
simple_summation
```

You can also use the print command, but this will supress non-printed variables.

```
print('Rounded: ', simple_summation)
```

1.3 Order matters

The second python cell would fail if the first one wasn't run.

You can run the whole program via the double-play Run icon above.

1.4 In-class exercise 1.1

Below, add two cells to this notebook.

First, create a markdown cell where you have a header and some paragraph text. To make something a header in Markdown language, just use a hashtag and a space before the title. To make it a paragraph, just separate it with a blank line in between. Finally, add a bulletted list with a few entries. To do this, just have a dash at the beginning of each new bullet.

Second, create a python cell. Save a variable that is the sum of all primes between 3 and 10. Print that sum.

2 Python Basics

- This notebook contains a very accelerated summary of the basics of python. If this is challenging, please refer to the textbook sections.
- Being able to successfully run this also means you have a working Python environment! You now have installed everything you need to run this course.

2.1 Python Basics

```
# Comments: The hashtag makes the rest of the line a comment. The more programming you do,
# Jupyter lets you write formatted text, but you'll still want to put comments in the raw

# Assign some text (a string) to a variable
some_text = 'This is the text.'

# Assign some numbers to variables
a = 5  # Here, we implicitly told python that a is an integer
b = 4.6  # Here, we told python that b is a floating point number (a decimal)
```

- Even though nothing is outputted above, our Python "Kernel" has the values to each variable stored for later use.

2.1.1 Important note: Python is not a "typed" language

- Notice that above, we added an integer and the float (a floating point number, i.e., one with a decimal point). Python "smartly" redefines variables so that they work together.
- This is different from other languages which require you to manually manage the "types" of your variables.

```
# Python as a calculator.
sum_of_two_numbers = a + b

# Printing output to the console
```

```
print('Our output was', sum_of_two_numbers)
```

- In the above, you'll notice the result was a float.
- If needed, you can demand that python specify something as a certain type, as below.

```
sum_as_int = int(sum_of_two_numbers)
sum_as_int_back_to_float = float(sum_as_int)

print('We lost some precision in this operation:', sum_as_int_back_to_float)
```

2.2 Other python types

```
# Reminder, this assumes you have setup an envioronment with conda using:
list_1 = [4, 5, 6]
print('list_1', list_1)
```

```
# You can embed lists in lists in lists, etc.
list_2 = [[5, 3, 5], [6, 6, 5]]
print(list_2)
```

```
# Dictionaries
dictionary_1 = {23: "Favorite number", 24: "Second favorite number"}
print('dictionary_1', dictionary_1)

# Here is a multi line string: (also discusses improved capabilities of an IDE editor)

things_you_can_do_in_vs_code_that_you_cant_do_without_an_ide = """
1.) Move back and forth in your history of cursor positions (using your mouse forward and
2.) Edit on multiple lines at the same time (hold alt and click new spots)
3.) Smartly paste DIFFERENT values
4.) Duplicate lines (ctrl-d)
5.) Introspection (e.g., jump between function definition and usages)
6.) Debugging (Interactively walk through your code one line at a time)
7.) Profiling your code (see which lines take the most time to compute.)
8.) Keep track of a history of copy-paste items and paste from past copies. (ctrl-shift-v)
"""
```

2.3 Looping

```
small_range = range(0, 10)
print('small_range:', small_range)

small_range_as_list = list(range(0, 10))
print('small_range_as_list:', small_range_as_list)

# Here is a for loop. Also note that python EXPLICITLY USES TAB-LEVEL to denote nested thi
# I.e., the inner part of the loop is tabbed 1 level up. Python does not use { like  R.
# I LOVE this notation and it's a big part of why python is so pretty and readable.
sum = 0 # Set the initial variable values
num = 0
sum_with_some = 0
for i in range(100, 136, 3):
    sum = sum + i
    num = num + 1

    # loop within a loop
    for j in range(200, 205):
        sum_with_some = sum + j

mean = sum / num
print('mean', mean)
```

2.4 Defining functions

```
# Functions
def my_function(input_parameter_1, input_parameter_2):
    product = input_parameter_1 * input_parameter_2
    return product

# Use the function
value_returned = my_function(2, 7)
print(value_returned)
```

```
# In-class exercise workspace
```

2.5 Importing packages

```
# Built-in packages via the Python Standard Library
import math
import os, sys, time, random

# Using imported modules
number_rounded_down = math.floor(sum_of_two_numbers)
print(number_rounded_down)
```

2.6 Using packages from elsewhere

When we used Mambaforge, we installed a ton of packages. These were not "built-in" to python like the ones above. Here we will import them into our notebook to use.

This will also illustrate the use of numpy. We'll use it so much we us the `as` code to name it something shorter.

```
import numpy as np # The as just defines a shorter name

# Create an 2 by 3 array of integers
small_array = np.array([[5, 3, 5], [6, 6, 5]])

print('Here\'s a small numpy array\n', small_array)

# Sidenote: from above backspace \ put in front of a character is the
# "escapce character," which makes python interpret the next thing as a string or special
```

2.7 Discussion point

The array above looks identical to the nested lists we made. It IS NOT! It is a numpy array that is ridiculously fast and can scale up to massive, massive data questions. The optional reading for today (Harris et al. 2020, Nature) discusses how these arrays have formed the backbone of modern scientific computing.

```
low = 3
high = 8
```

```
shape = (1000, 1000)

smallish_random_array = np.random.randint(low, high, shape)

print('Here\'s a slightly larger numpy array\n', smallish_random_array)
```

2.8 In-class exercise 2.1

Participation points note! I will call on a random table to show me their answer via their table's monitor.

Make a function that returns the square of a number. Combine the function with a loop to calculate the Sum of Squared Numbers from 1 to 100.

HINT, ** is the exponent operator in python.

BONUS: Make sure you're actually right by inserting a print statement in each step.

BONUS-bonus: Store each stage of the results in a list using `your_list = []` and `your_list.append(thing_to_add_to_your_list)`

Part II

Machine Learning

3 Linear Regression

This notebook will introduce you to one of the most-used machine learning packages: sklearn. We will start easy with an example very familiar to us all: OLS.

As always, start with our packages to import. The new one will be sklearn.

```python
# Import packages
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

In the imports above, we imported sub-packages of the sklearn package. Rather than loading the whole sklearn library (which is huge), line 4 just imports datasets and linear_model.

3.0.1 Load a specific dataset

Next, we will use the imported datasets object. It defines a useful function load_diabetes(). This will return a dataset object which contains the data and metadata.

```python
# Here we will load the diabetes dataset, which is a built-in dataset from sklearn.
full_dataset = datasets.load_diabetes()
print('full_dataset', full_dataset)
```

3.0.2 Inspect the dataset object to learn about it

It's overwhelming and hard to read what is printed out, but let's dig into this notation because it's frequently used and will help us understand different Python datatypes.

First, notice that the object starts with {, which means we can treat it like a python dicitonary. Dictionaries are standard ways of expressing key-value pairs. The standard notation for a dictionary is {key1: value1, key2: value2}

Below, we will use a method from the full_dataset object that returns just the keys. This should be easier to parse.

```
print('dictionary keys:', full_dataset.keys())
```

If we want, we can access just one entry in the dictionary using the key. A useful one is the key DESCR.

Print that out using the dictionary [] notation.

```
print(full_dataset['DESCR'])
```

3.0.3 Access the data array

Use the `'data'` key could also extract the data and assign it to a data_array variable for inspection. Let's also print out the `type` of the object to see what the data format is.

```
data_array = full_dataset['data']
print('data_array', data_array)
print(type(data_array))
```

It's a numpy array! That means we can use any of the numpy array functions on it.

3.1 In-Class Exercise:

Print out the mean BMI in the dataset, the sum BMI, and the sum of the squared BMI values. Explain why the sum of the squared BMI is what it is. To do this, you will need to access the right parts of the data array and slice out the right column.

HINT: You will need to read the DESCR to understand which column the BMI is stored in.

HINT2: To create a new variable with just the desired column of the array, you can use Array slicing notation like a = data_array[:, n] where the : means you want ALL rows, and the n means you want just column n.

HINT3: You may want to use np.sum(), np.mean(), and the ** exponent operator.

```
bmi = data_array[:, 2]
print(bmi)
```

```
# For conveinence, sklearn also just has an option to get the key parts for the regression
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

```
# Look at diabetes_X and notice there are lots of independent variables. Rather than print
# Array, which would be messy, just look at the .shape attribute.l
print('diabetes_X', diabetes_X)
print('diabetes_X', diabetes_X.shape)
```

For now, we're just going to use a single variable (a single column) for simplicity. The following line extracts just the second column from the array. The colon was necessary because we access arrays using the ROW, COLUMN notation, so we sliced out all ROWS (the colon indicates all) and the second COLUMN.

```
diabetes_X = diabetes_X[:, 2]
# diabetes_X = np.array([diabetes_X])
# diabetes_X = diabetes_X[:, np.newaxis, 2]
print('diabetes_X', diabetes_X.shape)
print('diabetes_X', diabetes_X)

# diabetes_X = diabetes_X.reshape((:, 1))
diabetes_X = diabetes_X.reshape((diabetes_X.shape[0], 1))

print('diabetes_X', diabetes_X.shape)
print('diabetes_X', diabetes_X)
```

3.1.1 Split into training and testing arrays (the manual way)

Next we are going to do a very rudimentary split of the data into training and testing sets using array slice notation. The following lines assigns the last all but the last 20 lines to the TRAIN set and the remaining 20 to the test set.

```
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

print(diabetes_X_test)
```

Create an empty LinearRegression object.

In the lines below, we will follow a relatively standardized process for running a model:

1. Create the model object.
2. Fit the model.

3. Predict with the model

The basic notation for sklearn below first creates a regression model object using the `linear_model` that we imported above. This model is "empty" in the sense that it has no coefficients identified. Just like othertimes we've encountered objects (like numpy array objects), this object has many functions (called methods) and attributes which can be accessed by the dot operator.

```
regression_object = linear_model.LinearRegression()
print('regression_object', regression_object)
```

3.1.1.1 Use the fit method

Use the fit method from our regression object. It takes two inputs, the independent variables (X) and dependent variables (y).

Below, we will ONLY use the training subset of the data we created above.

```
regression_object.fit(diabetes_X_train, diabetes_y_train)
print(regression_object)
```

3.1.1.2 Use the fitted model to predict values

Now the regression_object is "trained," which means we can also call it's predict() method which will take some other observations and (in the case of OLS), multiple the new observations against our trained coefficients to make a prediciton.

The predict method returned an array of numerical predictions, which we will look at.

```
diabetes_y_pred = regression_object.predict(diabetes_X_test)
print(diabetes_y_pred)
```

3.1.1.3 Look at the coefficients

More interesting might be to look at the coefficients. Once the model has been fit, it has a new attribute .coef_ which stores an array of coefficients. In this case it will only be an array of length 1 because we just have one input.

```
print('Coefficients: \n', regression_object.coef_)
```

You might be wondering why we are looking at the coefficients as a raw array rather than at a nicely formatted regression table. The reason is in cross-validation approaches, these coefficients might just be one step towards the final model performance check on unseen data.

3.1.1.4 Evaluating the fit

We can use sklearn's built in evaluation functions, such as for the mean squared error or other metrics.

```
mse = mean_squared_error(diabetes_y_test, diabetes_y_pred)
print('Mean squared error on the TEST data:',  mse)
```

```

# Or perhaps we want the r2 for the second independent variable (which is the only one we
r2_score_value = r2_score(diabetes_y_test, diabetes_y_pred)
print('r2 calculated on TEST data: ', r2_score_value)
```

```
# Finally, to prove to ourselves that we know what we are doing, let's plot this.
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.show()
```

3.2 Exercise 2.1: Machine Learning OLS Mashup.

Use loops to find which TWO variables best describe the data, as measured by R-squared. This is a hilariously brute-force approach to OLS model selection, but it is similar in some senses to Machine Learning and will be relevant to the cross-validation approaches we discuss next.

```
# Exercise 2.1 workspace and starter code



full_dataset = datasets.load_diabetes() # Load the full dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True) # Get just the data array

# Split into training and testing
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

```
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

highest_score = 0
for i in range(len(full_dataset['feature_names'])):
    for j in range(len(full_dataset['feature_names'])):

        diabetes_current_X_train = diabetes_X_train[:, [i, j]]
        diabetes_current_X_test = diabetes_X_test[:, [i, j]]

        # MISSING STUFF HERE.

        if r2_score_value > highest_score:
            highest_score = r2_score_value
            best_option = [i, j, r2_score_value]

print('best_option', best_option)
```

3.3 Just for completeness, let's look at this the way an econometritian would

Sklearn doesn't report summary statistics in the classic, econometric sense because it focuses on the train, test paradigm, which is not equivilent to a model performance report (which in the classic case is only reporting performance of the TRAINING data).

Nonetheless, Here's how I do it, using an alternative, more econometrics-focused package. You will need to conda install statsmodel if you want to uncomment this line and have it work. Note that because we're not splitting our data into training and testing, the r-squareds are not really comparable.

```
import statsmodels
from statsmodels.api import OLS

data_with_constant = statsmodels.api.add_constant(full_dataset.data)
result = OLS(full_dataset.target, data_with_constant).fit().summary()
print(result)
```

4 Support Vector Machines

In this section, we will use what we learned about fitting models and apply it to a very useful machine-learning algorithm.

First let's start with imports.

```python
import numpy as np
import scipy
import sklearn
from sklearn import datasets
import pandas as pd
import os
```

4.0.0.1 Load in some digit image data

One of the canonical datasets in sklearn is a series of images of handwritten digits. We've imported the datasets above, but now lets load it.

```python
digits = datasets.load_digits()

# First, take a look at the raw python object:
print('digits\n', digits)
```

```
digits
 {'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]]), 'target': array([0, 1, 2, ..., 8, 9, 8]), 'fran
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]), 'DESCR': ".. _digits_dataset:\n\nOptical rec
```

Not super helpful unless you're very good at reading python dictionary notation. Fortunately, one of the entries in this dataset is a description. Let's read that.

```
print('DESCR\n', digits['DESCR'])
```

```
DESCR
 .. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

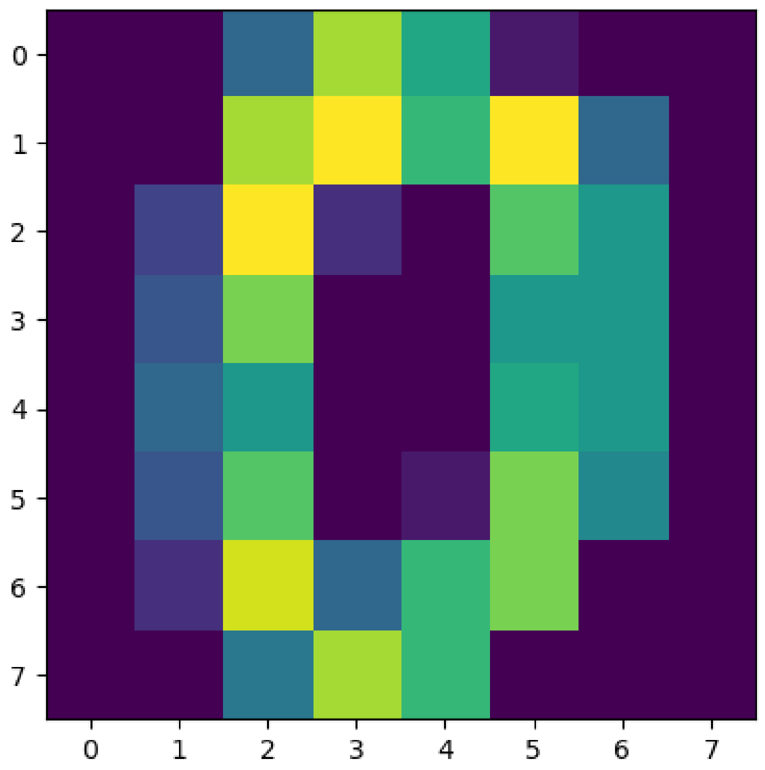
4.0.0.2 Extract one of the digits to inspect

Now that we're oriented, also look at one particular image of a digit, just so you know what it actually looks like. Below, we print just the first (index = 0) numeral of the 5620 they provide.

```
print('digits.images[0]\n', digits.images[0])
```

```
digits.images[0]
 [[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

```
# If you squint, maybe you can tel what image it is, but let's plot it to be sure.
import matplotlib
from matplotlib import pyplot as plt
plt.imshow(digits.images[0])
plt.show()
```



Notice also in the dataset that there is a 'targets' attribute in the dataset. This is the correct numeral that we are trying to make the model predict.

```
print('target', digits.target)
```

```
target [0 1 2 ... 8 9 8]
```

Our task now is to train a model that inputs the digit images and predicts the digit numeral. For this, we're going to use SVM, as discussed in lecture.

4.0.0.3 Import SVM and create a new (unfitted) model with it.

For now, the parameters are going to be manually set (gamme) but we'll address how to choose them later. Here, I want to illustrate the basic approach used in sklearn to Load, train, fit and predict the model

```python
from sklearn import svm

# Create the model object
classifier = svm.SVC(gamma=0.001)
```

At this point, classifier is not yet "trained", ie. not yet fit to the model. All ML algorithms in SKLEARN have a .fit() method, which we will use here, passing it the images and the targets.

Before we train it, we want to split the data into testing and training splits.

Class question: Remind me WHY are we splitting it here? What is the bad thing that would happen if we just trained it on all of them?

Before we can even split the data, however, we need to reshape it to be in the way the regression model expects.

In particular, the SVM model needs a 1-dimensional, 64 element array. BUT, the input digits we saw were 2-dimensional, 8 by 8 arrays.

This actually leads to a somewhat mind-blown example of how computers "think" differently than we do. We clearly think about a numeral in 2 dimensional space, but here we see that the computer doesn't are about the spatial relationship ship at all. It sees each individual pixel as it's own "Feature" to use the classification parlance. You could even reshuffle the order of those 64 digits and as long as you kept it consistent across the data, it would result in identical predictions.

Later on, we will talk about machine learning techniques that leverage rather than ignore this 2 dimensional, spatial nature of the data.

For now, let's just look at the data again. Rather than print it out, I really just want the shape so that i don't get inundated with text.

```python
print('digits.images shape', digits.images.shape)
```

```
digits.images shape (1797, 8, 8)
```

```python
n_samples = len(digits.images)
n_features = digits.images[0].size

print('n_samples', n_samples)
print('n_features', n_features)

data = digits.images.reshape((n_samples, n_features))
```

```
# Now check the shame again to see that it's right.
print('data shape', data.shape)
```

```
n_samples 1797
n_features 64
data shape (1797, 64)
```

```
# Now that we've arranged our data in this shape, we can split it into training and testin
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, digits.target, test_size=0.5, sh

print('X_train', X_train)
print('y_train', y_train)
```

```
X_train [[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  2. ... 14.  0.  0.]
 [ 0.  1. 12. ...  0.  0.  0.]
 [ 0.  0.  0. ...  3.  0.  0.]]
y_train [0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0
 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9
 5 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4
 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2
 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5 2 8 2 0 0 1 7 6 3 2 1 7 3 1 3 9 1
 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 5 4 8 8 4 9 0 8 9 8 0 1 2
 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9
 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5 2 8
 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2
 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 3 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2
 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7
 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 1
 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8
 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5 2
 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7
 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1
 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1
 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9
 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5
 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4
 7 2 8 2 2 5 7 9 5 4]
```

4.0.0.4 Fit the model

Finally, now that we've split it, we can call the classifier's fit method which takes the TRAINING data as input.

```python
classifier.fit(X_train, y_train)
```

```
SVC(gamma=0.001)
```

Now, our classifier object has it's internal parameters fit so that when we give it new input, it predicts what it thinks the correct classification is.

```python
predicted = classifier.predict(X_test)

# Looking at the predicted won't be very intuitive, but you could glance.
print('predicted', predicted)
```

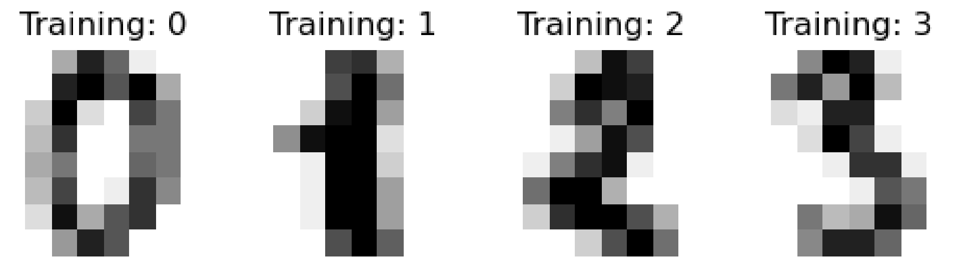
```
predicted [8 8 4 9 0 8 9 8 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 9 6 7 8 9
 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 9 1 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9
 1 5 0 9 5 2 8 2 0 0 1 7 6 3 2 1 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1
 7 5 4 4 7 2 8 2 2 5 7 9 5 4 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 7 8 2 0
 1 2 6 3 3 7 3 3 4 6 6 6 9 9 1 5 0 9 5 2 8 2 0 0 1 7 6 3 2 1 5 4 6 3 1 7 9
 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0
 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9
 5 2 8 2 0 0 1 7 6 3 2 2 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 8 7 5 4
 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 9 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2
 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 6 2 8 3 0 0 1 7 6 3 2 1 7 4 6 3 1 3
 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 0
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0 9
 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9 5
 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4
 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 1 2 3 4 5 1 7 8 9 0 1 2 3 4 5 6 9 0
 1 2 3 4 5 6 7 8 9 4 9 5 5 6 5 0 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2
 6 8 7 7 7 3 4 6 6 6 9 9 1 5 0 9 5 2 8 0 1 7 6 3 2 1 7 9 6 3 1 3 9 1 7 6 8
 4 3 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 2 5 7 3 5 9 4 5 0 8 9 8 0 1 2 3 4 5 6 7
 8 9 0 1 2 8 4 5 6 7 8 9 0 1 2 5 4 5 6 7 8 9 0 9 5 5 6 5 0 9 8 9 8 4 1 7 7
 7 5 1 0 0 2 2 7 8 2 0 1 2 6 8 8 7 5 8 4 6 6 6 4 9 1 5 0 9 5 2 8 2 0 0 1 7
 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 5 1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7
 9 5 4 8 8 4 9 0 8 9 8]
```

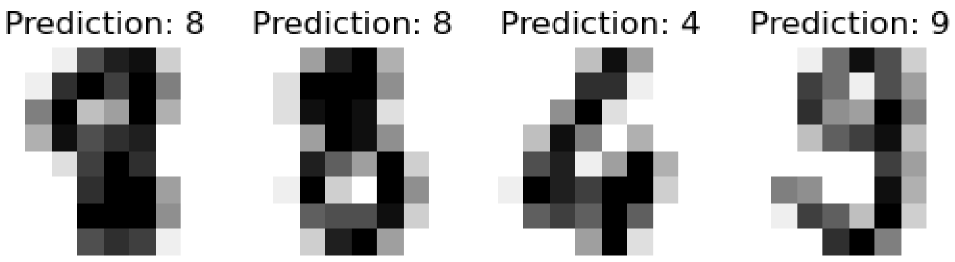
4.0.0.5 Plot some results

Let's plot a few of them in nicer format. Don't worry about learning the plotting code but it's a useful example to show the power.

```python
_, axes = plt.subplots(2, 4)
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(axes[0, :], images_and_labels[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for ax, (image, prediction) in zip(axes[1, :], images_and_predictions[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Prediction: %i' % prediction)
plt.show()
```





```python
from sklearn import metrics

print("Classification report:\n", metrics.classification_report(y_test, predicted))
```

```
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```

4.0.0.6 Confusion matrix

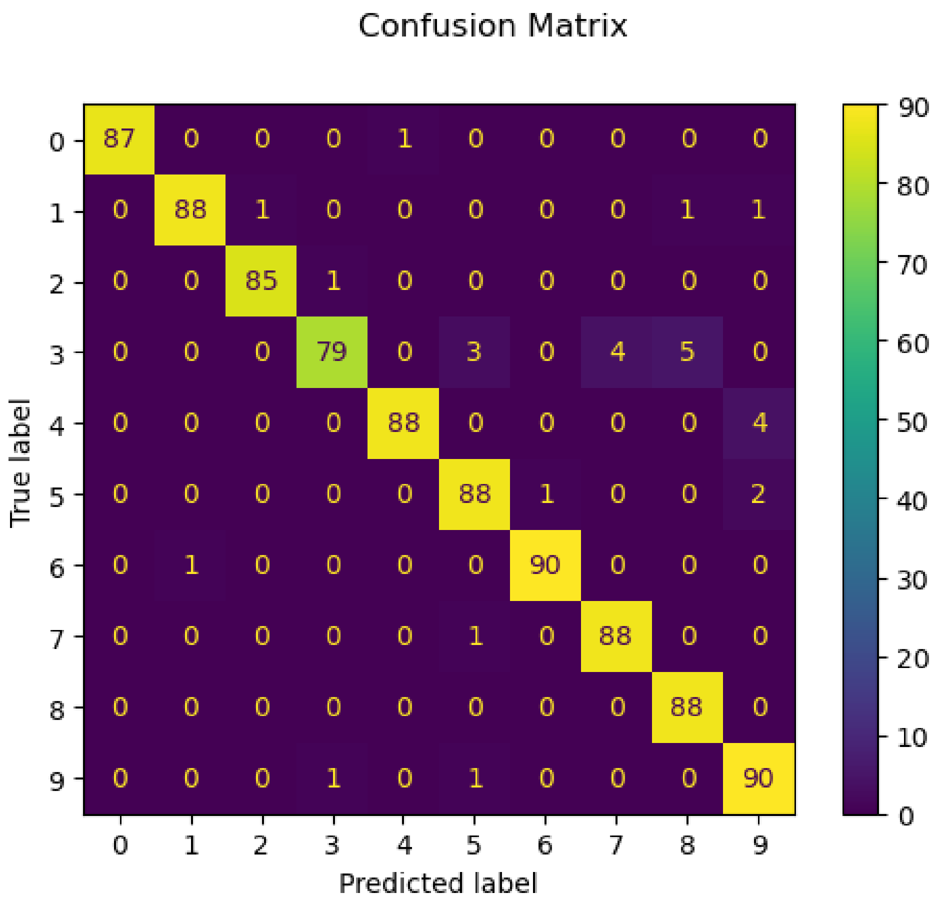
A more convenient way of looking at the results is t the confusion matrix. This is a built in metric for sklearn. It plots the predicted labels vs. the true labels.

```
disp = metrics.plot_confusion_matrix(classifier, X_test, y_test)
disp.figure_.suptitle("Confusion Matrix")

# print("Confusion matrix:\n", disp.confusion_matrix)

# Finally, show it so that you can look at it and see how good we did.
plt.show()
```

```
c:\Users\jajohns\AppData\Local\mambaforge\envs\8222env1\lib\site-packages\sklearn\utils\depre
  warnings.warn(msg, category=FutureWarning)
```



	0	1	2	3	4	5	6	7	8	9
0	87	0	0	0	1	0	0	0	0	0
1	0	88	1	0	0	0	0	0	1	1
2	0	0	85	1	0	0	0	0	0	0
3	0	0	0	79	0	3	0	4	5	0
4	0	0	0	0	88	0	0	0	0	4
5	0	0	0	0	0	88	1	0	0	2
6	0	1	0	0	0	0	90	0	0	0
7	0	0	0	0	0	1	0	88	0	0
8	0	0	0	0	0	0	0	0	88	0
9	0	0	0	1	0	1	0	0	0	90

QUESTION: Which digit was hardest to categorize and what was it most frequently confused as?

Part III

Appendices

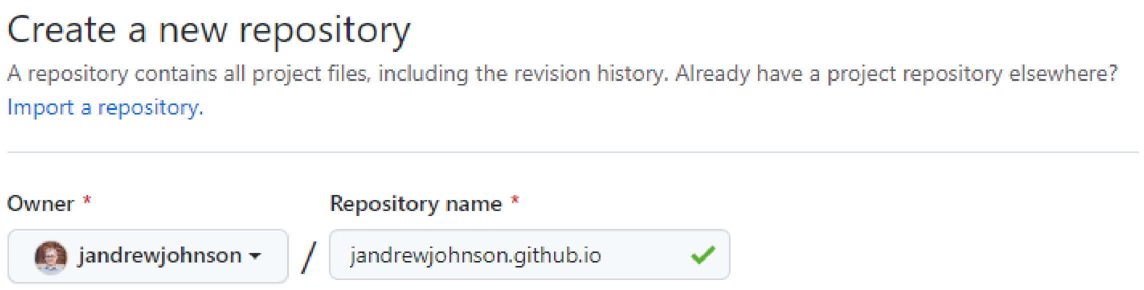
5 Hosting notes

5.1 Pasting images

NOTE: This is easiest to do in ipynb NOT in qmd because of the built-in options for pasting images into notebooks.

Note that there are two methods: paste (ctrl-v) directly into a markdown cell of a ipynb. This adds it as an "attachment" where the raw binary image code is written into text in the cell's attribute (and thus there is no external file saved). The other is the use the Paste Image VS Code plug in, which lets you actually write a file in a gnerated location for the PNG. This is called by ctrl-alt-v

Option 1



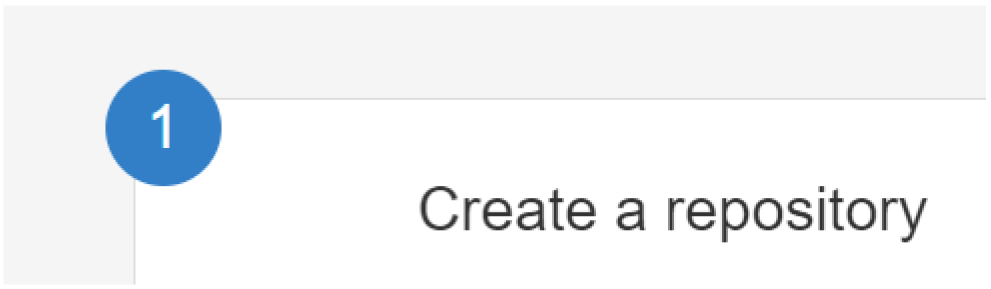
Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * jandrewjohnson / Repository name * jandrewjohnson.github.io

Figure 5.1: image.png

Option 2



6 Cross References

For tables produced by executable code cells, include a label with a tbl- prefix to make them cross-referenceable. For example:

```python
from IPython.display import Markdown
# from tabulate import tabulate
# table = [["Sun",696000,1989100000],
#          ["Earth",6371,5973.6],
#          ["Moon",1737,73.5],
#          ["Mars",3390,641.85]]
# Markdown(tabulate(
#   table,
#   headers=["Planet","R (km)", "mass (x 10^29 kg)"]
# ))
```

Table 6.1: **?(caption)**