Documentation of the Skills and Wages project

Pooja Bansal

06 March 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

Documentation on the rationale, Waf, and more background is at http://hmgaudecker.github.io/econ-project-templates/

The Python version of the template uses a modified version of Stachurski's and Sargent's code accompanying their Online Course [StachurskiSargent13] for Schelling's (1969, [Schelling69]) segregation model as the running exmaple.

1.1 Getting started

This assumes you have completed the steps in the README.md file **and everything worked.**

The logic of the project template works by step of the analysis:

1. Data management
2. The actual estimations / simulations / ?
3. Visualisation and results formatting (e.g. exporting of LaTeX tables)
4. Research paper and presentations.

It can be useful to have code and model parameters available to more than one of these steps, in that case see sections model_specifications, model_code, and *Code library*.

First of all, think about whether this structure fits your needs – if it does not, you need to adjust (delete/add/rename) directories and files in the following locations:

- Directories in **src/**;
- The list of included wscript files in **src/wscript**;
- The documentation source files in **src/documentation/** (Note: These should follow the directories in **src** exactly);
- The list of included documentation source files in **src/documentation/index.rst**

Later adjustments should be painlessly possible, so things won't be set in stone.

Once you have done that, move your source data to **src/original_data/** and start filling up the actual steps of the project workflow (data management, analysis, final steps, paper). All you should need to worry about is to call the correct task generators in the wscript files. Always specify the actions in the wscript that lives in the same directory as your main source file. Make sure you understand how the paths work in Waf and how to use the auto-generated files in the language you are using particular language (see the section *Project paths* below).

1.2 Project paths

A variety of project paths are defined in the top-level wscript file. These are exported to header files in other languages. So in case you require different paths (e.g. if you have many different datasets, you may want to have one path to each of them), adjust them in the top-level wscript file.

The following is taken from the top-level wscript file. Modify any project-wide path settings there.

```
def set_project_paths(ctx):
    """Return a dictionary with project paths represented by Waf nodes."""

    pp = OrderedDict()
    pp["PROJECT_ROOT"] = "."
    pp["IN_DATA"] = "src/original_data/"
    pp["LIBRARY"] = "src/library"
    pp["BLD"] = ""
    pp["OUT_DATA"] = f"{out}/out/data"
    pp["OUT_ANALYSIS"] = f"{out}/out/analysis"
    pp["OUT_FINAL"] = f"{out}/out/final"
    pp["OUT_FIGURES"] = f"{out}/out/figures"
```

As should be evident from the similarity of the names, the paths follow the steps of the analysis in the `src` directory:

1. **data_management** → **OUT_DATA**
2. **analysis** → **OUT_ANALYSIS**
3. **final** → **OUT_FINAL**, **OUT_FIGURES**, **OUT_TABLES**

These will re-appear in automatically generated header files by calling the `write_project_paths` task generator (just use an output file with the correct extension for the language you need – `.py`, `.r`, `.m`, `.do`)

By default, these header files are generated in the top-level build directory, i.e. `bld`. The Python version defines a dictionary `project_paths` and a couple of convencience functions documented below. You can access these by adding a line:

```
from bld.project_paths import XXX
```

at the top of you Python-scripts. Here is the documentation of the module:

bld.project_paths

Define a dictionary *project_paths* with path definitions for the entire project.

This module is automatically generated by Waf, never change it!

If paths need adjustment, change them in the root wscript file.

project_paths_join(*key*, **args*)

Given input of a *key* in the *project_paths* dictionary and a number of path arguments *args*, return the joined path constructed by:

```
os.path.join(project_paths[key], *args)
```

project_paths_join_latex(*key*, **args*)

Given input of a *key* in the *project_paths* dictionary and a number of path arguments *args*, return the joined path constructed by:

```
os.path.join(project_paths[key], *args)
```

and backslashes replaced by forward slashes.

CHAPTER

TWO

ORIGINAL DATA

Documentation of the different datasets in *original_data*.

In the original data section we stored the raw data files which will be manipulated in our python codes. These should not be manipulated and saved in the same folder as it would hamper project reproducibility. 4 file exist in our folder. All are state files which are readable in python

Along with this I have put an image taken which we generated later using our analysis. The original format was in graphviz dot file. In the figure code file I imported the image from this folder and exported it to the out_figures folder using the codes in final python file.

CHAPTER

THREE

DATA MANAGEMENT

Documentation of the code in *src.data_management*.

We start with merging all the relevant data files. SOEP has provided with the data for two different waves using different questionnaires. We first merge all of the datasets and then keep only the useful variables. Now, all our data for skill variables, be it cognitive or non-cognitive was categorical. So we first converted the categorical figures into numeric to perform mathematical operations on them. We then rename the important variables to be used later for our convenience. Then we restrict our data with individuals of age between 20 and 60 and drop individuals with occupations not useful in our analysis. After this, we dropped all the missing values by first replacing them nan. The variables for Cognitive abilities were standardised using the sklearn library. For personality we had 15 variables to be reduced to 5 be taking averages of 3 variables corresponding to a particular personality trait. Out of those 15, 4 had to be reversed since they represented the opposite quality corresponding the respective trait. We then created 5 variables by taking the average and standardising them. We then generated the experience and experience squared variables for the Mincer equation. We then took a log of wages to improve our regression model. For occupations, we combined similar categories and sorted them in order.

vgfghdjtrddyt

CHAPTER

FOUR

MAIN MODEL ESTIMATIONS / SIMULATIONS

Documentation of the code in *src.analysis*. This is the core of the project.

4.1 Regression

We're all familiar with the idea of linear regression as a way of making quantitative predictions. In simple linear regression, a real-valued dependent variable Y is modeled as a linear function of a real-valued independent variable X plus noise. In multiple regression, we let there be multiple independent variables. We perform OLS regression with the variables generated in the first step of Data Management.

We run the regression using statsmodel python package.

First we investigate the returns to earnings by using the basic Mincer Equation which used 3 variables [Years of education, Years of Experience and Square of years of experience.] We will then expand the basic specification by adding the cognitive skills : Fluency and Symbol test score which we standardised. We then introduce only non cognitive skills in the basic specification : Openness, Conscientiousness, Extraversion , Agreeableness and Neuroticism.

We then add both skills (Cognitive and Non-Cognitive)to the specification.

We define our independent variables as X and dependent variables as Y, For each specification, we generate a new matrix. We fit our linear model using the OLS module offered by the sm library and then print the summary which contains the regression output for all the defined model.

After the first 4 regression, we perform regression for different occupational groups by defining a loop for the values in the our column occupation which contains : 1,2,3,4 thus generates results for 4 different outputs for all categories.

4.2 Decision Tree

This is not a part of my seminar paper. It's an analysis to confirm the validity of my results.

Linear regression is a global model, where there is a single predictive formula holding over the entire data-space. An alternative approach is to sub-divide, or partition, the space into smaller regions, where the interactions are more manageable. We then partition the sub-divisions again this is called recursive partitioning until finally we get to chunks of the space which are so tame that we can fit simple models to them. The global model thus has two parts: one is just the recursive partition, the other is a simple model for each cell of the partition.

Prediction trees use the tree to represent the recursive partition. Each of the terminal nodes, or leaves, of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell

only. A point x belongs to a leaf if x falls in the corresponding cell of the partition. To figure out which cell we are in, we start at the root node of the tree, and ask a sequence of questions about the features. The interior nodes are labeled with questions, and the edges or branches between them labeled by the answers. For classic regression trees, the model in each cell is just a constant estimate of Y. Advantages of Decision Trees:

- Making predictions is fast (no complicated calculations, just looking up constants in the tree)
- It's easy to understand what variables are important in making the pre- diction (look at the tree)
- If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub-tree we do reach
- The model gives a jagged response, so it can work when the true regression surface is not smooth. If it is smooth, though, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves)
- There are fast, reliable algorithms to learn these trees

The basic regression-tree-growing algorithm then is as follows: #. Start with a single node containing all points.Calculate the prediction for leaf and Total Sum of Squares.

1. If all the points in the node have the same value for all the independent variables, stop. Otherwise, search over all binary splits of all variables for the one which will reduce S as much as possible. If the largest decrease in S would be less than some threshold δ, or one of the resulting nodes would contain less than q points, stop. Otherwise, take that split, creating two new nodes.
2. In each new node, go back to step 1.

We use the Top-Down Induction of Decision Trees. Main: #. **let** T := Node := a decision tree consisting of an empty root node #. **return** TDIT (E,Atts,T,Node) E: set of examples, Atts: set of attributes

Discussing functions:

TDIDT gain:

*. Entropy:

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). TDIDT algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy. WE choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

*. Get Information Gain: The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).We calculate the entropy using the defined entropy function. The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

Number of positives: Calculates the number of positives and negative for further splitting

Create tree TDIDT: TDIDT(E, Atts, T, Node): Now, with the information gain is a measure of how well a certain split is based on how well the classification accuracy is after the split. Here, a data set is split based on a condition of the parameter and the data associated with this is assigned to that particular

group. The best node, i.e the node with the highest information gain is set as the node condition and the same steps are repeated until further split doesn't give any improvement which is called a leaf node. For this node, the predicted output is set based on majority voting.

Classify : Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.A branch with entropy of 0 is a leaf node. A branch with entropy more than the threshold needs further splitting.

Test Data Output: Predicting the values using results generated from the decision tree and calculating the accuracy/percentage of match the validate our results.

Export tree node: Exports the generated nodes to the dot files if the index is none.

Update to Dot File: Update the file with the generated nodes until the splitting stops.

4.3 Decision Tree OUTPUT

The result is generate in a dit file which was later converted as an image on the graphviz website. However, the image was reloaded and saved using python but the image quality reduced by a great amount. Kindly refer to the image in original data folder. As we can interpret from the image, Extraversion is the most important predictor chosen by the Decision tree which is possible since our regression results prove that it is highly significant in all the cases. We then see that Agreeableness is not an important factor which is again validated by our regression results. Fluency test scores are relatively less important than the symbol test scores and so did our regression results conveyed. However, Openness is shown to be an important variables by the decision tree.

CHAPTER

FIVE

VISUALISATION AND RESULTS FORMATTING

Documentation of the code in *src.final*.

5.1 Descriptive Stats and Decision Tree

The code builds different plots for the data used and give the statistics. We have generated 3 plots using the seaborn and matplotlib python libraries. We imported our dataset and exported all our images in the out_figures folder. We generated one bar chart, one combined image of density plots. We also exported our decision tree image to this folder from the original data folder to keep all our generated images at one place.

CHAPTER

SIX

RESEARCH PAPER / PRESENTATIONS

Purpose of the different files (rename them to your liking):

- `research_paper.tex` contains the seminar paper "Labor Market Returns to Cognitive and NOn-Cognitive Skills across Different Occupations"
- `research_pres_30min.tex` contains conference presentation for the research paper
- `formulas` contains short files with the LaTeX formulas – put these into a library for re-use in paper and presentations.

CHAPTER

SEVEN

CODE LIBRARY

The directory *src.library* provides code that may be used by different steps of the analysis. Little code snippets for input / output or stuff that is not directly related to the model would go here.

No codes are stored in the library for the purpose of this project.

CHAPTER

EIGHT

REFERENCES

The references for the research paper has been cited using ref.bib in our research paper tex file. The following are the references for project understanding and coding which are listed in ref.bib :

1. A Primer on Scientific Programming with Python, Hans Petter Langtangen, Springer 2009
2. An Introduction to Computer Science Using {Python}, Jennifer Campbell and Paul Gries and Jason Montojo and Gregory V. Wilson, The Pragmatic Programmers 2009.
3. Best Practices for Scientific Computing, Gregory V. Wilson and D. A. Aruliah and C. Titus Brown and Neil P. Chue Hong and Matt Davis and Richard T. Guy and Steven H. D. Haddock and Katy Huff and Ian Mitchell and Mark Plumbley and Ben Waugh and Ethan P. White and Paul Wilson, 2012
4. Intelligent Learning and Analysis Systems, Lecture Papers, Prof. Dr. Stefan Wrobel, Dr. Tamas Horvath, Dr. Krisztian Buza, University of Bonn and Fraunhofer IAIS
5. Data Mining with Decision Trees: Theory and Applications, Oded Z. Maimon
6. Decision Tree Learning, Handbook, University of Princeton
7. Computational science: . . . Error, Zeeya Merali, Nature 2010
8. Data Crunching, Gregory V. Wilson, The Pragmatic Programmers 2005
9. Data Mining with Decision Trees, Lior Rokach, 2007
10. Find the Bug. A Book of Incorrect Programs, Adam Barr, Addison Wesley 2004
11. Decision Trees and Random Forests: A Visual Introduction For Beginners: A Simple Guide to Machine Learning with Decision Trees, Chris Smith

PYTHON MODULE INDEX

INDEX