NET Institute*

www.NETinst.org

Working Paper #18-05

October 2018

Technology Adoption in a Hierarchical Network

Xintong Han
Concordia University and CIREQ

Lei Xu
Toulouse School of Economics

* The Networks, Electronic Commerce, and Telecommunications (“NET”) Institute, http://www.NETinst.org, is a non-profit institution devoted to research on network industries, electronic commerce, telecommunications, the Internet, “virtual networks” comprised of computers that share the same technical standard or operating system, and on network issues in general.
Technology Adoption in a Hierarchical Network

Xintong Han and Lei Xu*

Preliminary and Incomplete.

September 30, 2018

Abstract

This paper studies the effect of network structure on technology adoption, in the setting of the Python programming language. A major release of Python, Python 3, provides more advanced but backward-incompatible features to Python 2. We model the dynamics of Python 3 adoption made by package developers. Python packages form a hierarchical network through dependency requirements. The adoption decision involves not only updating one’s own source code, but also dealing with dependency packages lacking Python 3 support. We build a dynamic model of technology adoption where each package makes an irreversible decision to provide support for Python 3. The optimal timing of adoption requires a prediction of all future states, for the package itself as well as each of its dependencies. With a complete dataset of package characteristics for all historical releases, we are able to draw the complete hierarchical structure of the network, and simplify the estimation by grouping packages into different layers based on the dependency relationship. We study how individual adoption decisions can propagate along the links in such a hierarchical network. We also test the effectiveness of various counterfactual policies that can promote a faster adoption process.

*Han: Concordia University and CIREQ, Montreal, Canada. Email: xintong.han@concordia.ca. Xu: Toulouse School of Economics, Toulouse, France. Email: lei.xu@tse-fr.eu. We thank Victor Aguirregabiria, Luis Cabral, Allan Collard-Wexler, Jacques Cremer, Olivier De Groote, Pierre Dubois, Daniel Ershov, Ana Gazmuri, Matthew Gentry, Gautam Gowrisankaran, Philip Haile, Doh-Shin Jeon, Thierry Magnac, Ariel Pakes, John Rust, Mario Samano, Paul Seabright, Elie Tamer, Kevin Williams, as well as seminar and conference participants at TSE, Concordia, Edinburgh, and EARIE for helpful comments. Financial support from TSE Digital Center and the NET Institute is gratefully acknowledged. All errors are our own.

1Most functionalities on Python are achieved through third-party packages. Packages are also known as libraries, subroutines, or modules in other programming languages.
1 Introduction

As the modern economy gets more disaggregated, firms tend to specialize and form a hierarchical network. Firms are no longer independent: their decisions and idiosyncratic shocks can propagate through the network (Acemoglu et al. (2012)). This paper investigates how individual decisions to adopt a new technology can propagate through a hierarchical network, as well as what types of policies can be implemented to promote a faster adoption process.

One important feature in technology adoption is known as network effects or network externalities (Katz and Shapiro (1985)). Take the classic example of fax machine. A user benefits from a fax machine by sending or receiving faxes from others. Network effects have been typically modeled as a function of total number of users on the same network or using the same technology, which can be viewed as a “reduced-form” approach of the underlying network of decision makers. In reality, the detailed network structure matters for analyzing such network effects (Brancaccio, Kalouptsidi, and Papageorgiou (2017)). For example, a user’s decision to purchase a fax machine is more affected by the decisions of others in frequent communication with her, instead of the total number of users in the world. This paper integrates the detailed network structure with a dynamic model of technology adoption, and empirically shows that the decision of one agent can have significant impact on other connected agents.

We study the role of network structure on technology adoption in the setting of the Python programming language, in particular, an on-going transition process from Python 2 to Python 3. Python 3 is a major release in 2008 which has experienced slow adoption. It provides several fundamental improvements that are incompatible to Python 2.

Like other programming languages, most functionalities on Python are provided by third-party packages. These packages form a hierarchical network through dependency require-
ments, namely, each (downstream) package relies on functionalities provided by other (upstream) packages. In order to use a package, the end user has to install both the package itself as well as all its (upstream) dependency packages.

The incompatibility between Python 2 and Python 3 implies adoption costs for both the end users and third-party packages. In order to use the new features in Python 3, one has to update its own source code; if one or more of its dependencies lack Python 3 support, then additional adoption cost is incurred. Our research focuses mainly on the adoption decisions of the third-party packages.

Both the Python programming language and almost all the third-party packages are Open Source Software (OSS). The motivations of OSS contribution by software developers can be multifold. We assume that developers of a package make the adoption decision collectively and model utility as a function of user downloads. For any of the most commonly-known motivations (such as altruism, career incentives, ego gratification, etc.), more user downloads are always better for the package developers.

In order to understand how the dynamics of technology adoption are affected by network structure, we build a dynamic model of technology adoption in which each package makes an irreversible decision to adopt Python 3, i.e. making the package available for other Python 3 users or packages. The intertemporal tradeoff comes mainly from higher benefits from more user downloads versus lower adoption cost due to future adoption decisions of dependencies. The solution of a dynamic choice model requires the specification of transition probabilities for each of the state variables. The complexity of this dynamic model comes from the nested states variables. Each agent’s state variable includes not only one’s own characteristics and idiosyncratic shocks, but also those of its dependencies, as well as those of the dependencies of dependencies, and so on.

With a complete dataset of package characteristics, historical releases and user download statistics, we introduce a novel estimation approach based on a unique characteristic of this

---

6 Upstream packages are dependencies for downstream packages. Downstream packages are built using functionalities provided upstream packages.

7 The typical solutions (or costs) include looking for alternative dependency packages that provide Python 3 support, or update the necessary components of the dependency to support Python 3.

8 All the data come from the Python Package Index project, which is the largest repository for Python packages, and it records historical downloads information for more than 113,000 packages from 2005 until now.
dependency network, namely, unidirectionality.

The rich dataset allows us to draw a complete hierarchical structure of all packages. We group packages into various layers based on the dependency relationship: Layer 0 includes packages without any dependencies; layer 1 includes packages only using packages in layer 0 as dependencies; layer 2 includes packages only using packages in layer 0 & 1 as dependencies; and so on. Then we calculate the choice probabilities for all the packages by layer, starting with layer 0, then layer 1, layer 2, and so on. With the adoption probabilities of upstream packages, together with simplifying assumptions on the belief of future adoption probabilities for each of the dependencies, we are able to solve the dynamic choice model and calculate the adoption probability for each package at all time period.

The adoption cost of a new technology can be heterogeneous across agents (Gowrisankaran and Stavins (2004), Ryan and Tucker (2011)). Such heterogeneous effects are controlled in our model through the incorporation of the actual links between packages and different package characteristics.

We can also use the variations of dependency characteristics to identify the discount factor, which is typically not separately identified in many dynamic discrete choice settings.

Our results show that packages benefit from more user downloads, and dependencies that lack Python 3 support pose significant barriers in the adoption decisions of downstream packages. The cost of dealing with one Python 3-incompatible dependency is equivalent to, on average, one third of updating one’s own source code.

Using modularity optimization tools developed in the literature of social networks, we group packages into various “communities” based on the dependency network, such as communities of web development, data analysis, etc. We measure both local (within-community) and global (across-community) network effects. These communities differ significantly from each other in both size and sparseness, which provides nice variation for the identification of our model. With the estimated results, we can measure the heterogeneous effects of network structure on technology diffusion through each of the “communities”.

The Python Software Foundation, which is a nonprofit organization that fosters the develop-

---

9The idea of “global” network effects is equivalent to “long distance network externalities” in previous literature (see Economides and White (1994))
opment of Python communities and helps oversee various issues related to Python, including the transition from Python 2 to Python 3. Like many other new technology or standards with network effects, the adoption process is known to have “excess inertia”, even though the majority of users are better off with the new technology (Farrell and Saloner (1985)). We conduct various counterfactual policies of “sponsorship”. A sponsor is some entity who is willing to invest to promote a new technology, and it can have a huge impact on the success of a new technology or standard (Katz and Shapiro (1986)). The structural model allows us to conduct counterfactual exercises of how various policies can foster a faster technology diffusion process throughout the whole network.

Our paper contributes mainly to the literature of technology adoption. Previous papers have modeled network effects using a more “reduced-form” approach, which can be a poor approximation to the actual effects that agents have on each other through a network, especially in cases where agents have heterogeneous characteristics and links. Our paper is one of the first to link the literature of network effects and the literature of social networks, by building a dynamic model of technology adoption that incorporates the detailed network structure.

2 Background: the Python Programming Language

Python is a general-purpose programming language\(^\text{10}\) It has a syntax that allows users to express concepts in fewer lines of code compared to most other languages, and it has been widely used in “Intro to Computer Science” courses in universities. The first version was released in 1989, but it did not gain much popularity until late 2000s. During the past few years, Python has become the fastest-growing major programming language. Figure 1 plots the numbers of visits to questions related to a particular programming language on Stack Overflow, the largest Q&A website for programming-related matters. Python has grown to be the number one based on this measure.

\(^\text{10}\)A general-purpose programming language is a computer language that is broadly applicable across application domains. Examples of general-purpose programming languages include C, Java, and Python. It is in contrast to domain-specific language, such as MATLAB (numerical computing), Stata and R (statistical analysis), etc.
Backward compatibility has been a widely disputed topic in the software industry. The tradeoff is very clear: easier user transition to the new technology vs. higher cost of development and slowing innovation. In order to introduce several key features to Python, the core teams decided to break the backward compatibility in a major release in 2008, namely, Python 3. Users and package developers who would like to run their code on Python 3 would have to make sure that all the source code is compatible to Python 3.

The transition process has indeed been longer than many had expected. Due to a large existing Python 2 user base, many packages are reluctant to provide Python 3 support. For those who did, they usually provide two versions for each release: one for Python 2, and another for Python 3 users.

Figure 1 plots the proportion of packages that provide Python 3 support. It has experienced a steady growth, during which timothy total number of packages has also been

---

11 Some of the major new features include newer classes, Unicode encoding, and float division. Please refer to [http://python.org/](http://python.org/) for more detailed information.

12 Some packages are developed to help users to transit to Python 3 easier automatically. But in most cases, users and package developers would still have to test and manually modify much code.
2.1 Motivation of Package Developers

The motivations of developers’ contribution are multi-fold. There are a large literature on the motivations behind private contributions to public goods such as Open Source Software.\(^\text{13}\) Under any of the most common motivations (such as altruism, career, ego gratification, etc.), more downloads are always better for the developers.\(^\text{14}\) Therefore, we assume that the payoff of package developers as a function of user downloads.\(^\text{15}\)

3 Data

All of our data come from Python Package Index (PyPI). It is a repository of software for the Python programming language. In another word, it is a website where Python packages can be stored and downloaded by others. It provides some easy search functions for users to

\(^\text{13}\) Nearly all the Python packages in our study are open source, which is also free of charge for anyone to use. A limited number of packages offer free downloads but requires payment for usage.

\(^\text{14}\) von Krogh et al. (2012) and Xu, Nian, and Cabral (2016) provide overviews of the literature.

\(^\text{15}\) Similar assumption has also been used in other papers in the study of OSS (See Gandal and Fershtman 2011 and Gandal and Stettner 2016).
find and install packages they would like to use.

When uploading packages to PyPI, package developers usually provide various information related to each release, such as the description of the package, contact information of owner, whether they support for Python 2 or Python 3, what other packages are required as dependencies, etc. Table 1 lists some basic information related to a package. The dependency requirement comes from “requires_dist” section, and the Python 2/3 support status comes from “classifiers” section. Before 2015, PyPI also records cumulative downloads statistics for each file on it. After 2015, the downloads statistics system stopped working for a few months, and in late 2015, they provided a new system which provided more detailed information for each download. Table 2 and 3 show the sample data for the old and new
systems. The new system provides much more detailed data over the old system. However, so far it stores only about two years of data. In this draft, we use data from the old system, i.e. until 2015. We do plan to take advantage of the rich information available in the new system in future versions of our research.

The official statistics show that it currently holds more than 130k packages and 900k releases. Like any other such platform, there are a long tail effect where most downloads come from a handful of packages and most packages are not regularly maintained. Our analysis would require a further selection of more actively maintained packages.

Figure 3 plots the total downloads of top packages as percentage of all downloads. Top
100 packages account for about 50% of all downloads; top 500 packages account for 70% of all downloads; etc.

Most packages on PyPI are small packages and are not widely used by users. They are developed probably just as a hobby for personal use. For example, 40% of all the packages have only 1 or 2 releases. We select the best-maintained and most important packages on PyPI for our estimation, based on the following criteria:

- Time Duration (Last Release - First Release Date) ≥ 1 Year: 12.9%
- Downloads Per year ≥ 2000: 30.8%
- Total Number of Releases ≥ 5: 38.9%
- Total Releases / Time Duration ≥ 1: 92.4%
- Some Python 2/3 Support Info Available: 59.9%

This selection criteria give us 3357 packages and 13056 observations for our analysis. Other selection measures will also be used for robustness checks.

Table 4: Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of packages</td>
<td>143,584</td>
<td>4,005</td>
</tr>
<tr>
<td>Total downloads (in percentage)</td>
<td>100.00%</td>
<td>51.69%</td>
</tr>
<tr>
<td>Average logarithm of downloads per package</td>
<td>5.87</td>
<td>11.51</td>
</tr>
<tr>
<td>(min, max)</td>
<td>(0.0, 19.3)</td>
<td>(8.8, 19.3)</td>
</tr>
<tr>
<td>Average number of dependencies</td>
<td>0.72</td>
<td>3.12</td>
</tr>
<tr>
<td>(min, max)</td>
<td>(0, 174)</td>
<td>(0, 79)</td>
</tr>
<tr>
<td>Average logged package size (MB)</td>
<td>2.78</td>
<td>4.19</td>
</tr>
<tr>
<td>(min, max)</td>
<td>(0.0, 13.2)</td>
<td>(0.5, 10.6)</td>
</tr>
</tbody>
</table>

Table 4 shows the summary statistics of several key variables between the whole population and our selected sample. The selected sample includes most of the most popular packages on Python, consisting 51.69% of all the downloads on PyPI. The average number of downloads for each package in the selected sample is also much higher than the whole population. On average, the selected sample has 3.12 dependencies, vs. 0.72 in the whole
population. These packages are also larger in the file size. Consistent with other figures in this section, Table 4 implies that our analysis focuses on those well-maintained packages with regular releases that faced an adoption decision of Python 3. These packages are also the most popular packages in the Python community.

All the decision and state variables in our model are measured in 6-month periods. For example, 2013/01/01 to 2013/06/30 is one period, and 2013/07/01 to 2013/12/31 is another.

### 3.1 Communities in Python

Python is a general-purpose programming language, meaning that it is not designed for a specific community of users; rather, it is designed in a flexible way for the use by different communities (e.g. data analysis, web development, etc). The needs of communities are met by third party packages, most of which are designed to achieve certain tasks in a specific field. Those packages that serve users of a given community tend to be more closely linked through dependencies. Using the latest heuristic method of modularity optimization, we are able to extract the community structure of the dependency network.
Figure 4 shows the 10 major communities through plotting a sparse matrix which represents the links between any two packages. Both x and y axes represent the packages. If two packages are linked, then there is a black dot. Therefore, each black square represents a community; a larger square means more packages in that community; a darker (or dense) one means packages in that community are more closely linked to each other.

3.2 Vertical Network with a Hierarchical Representation

One particular characteristic of our model is the existence of the vertical network. Each package $i$ has some linked with some upstream packages and downstream packages. Let use $G \in \{0, 1\}^{n \times n}$ a $n \times n$ matrix to present the relationship between all packages. For each element $g_{i,j} \in G$, if package $j$ is a direct upstream package that has been cited in package $i$’s report when releasing and appears in package $i$’s source code, we would observe $g_{i,j} = 1$. We denote $U_i = \{j; g_{i,j} \in J and g_{i,j} = 1\}$ the set of upstream firms. We assume in our paper that $G$ is time-invariant and is commonly known by packages. Furthermore, elements in $G$ are assumed to be arranged in row by their hierarchy, and there are in total $L$ hierarchies. One can use block matrices to define $G$:

$$G = \begin{bmatrix}
0 & 0 & \cdots & 0 & 0 \\
H_{2,1} & 0 & \cdots & \cdots & 0 \\
H_{3,1} & H_{3,2} & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & 0 & 0 \\
H_{L,1} & H_{L,2} & \cdots & H_{L,L-1} & 0
\end{bmatrix},$$

where diagonal blocks and upper triangular part of the matrix $G$ are zeros, block matrices $H_{i,j}$ for $i > j$ and $j = 2, \ldots, L$ represents the dependencies between packages in hierarchy $i$ and its upper hierarchy $j$. We further denote $H = \{H_1, \ldots, H_L\}$ a set indicating the element’s hierarchy level where $H_\ell$ contains all packages belonging to hierarchy $\ell$. For instance, if the network structure is defined as in Figure 5, we have:
Figure 5: Example of Hierarchical Network

\[ G = \begin{bmatrix} 0 & 0 & 0 \\ H_{2,1} & 0 & 0 \\ H_{3,1} & H_{3,2} & 0 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \]

\[ H = \{H_1, H_2, H_3\}, \]

where \( H_{2,1} = (1, 1) \) and \( H_{3,1} = (0, 1) \), \( H_{3,2} = 1 \) and \( H_1 = \{A, B\} \), \( H_2 = \{C\} \), \( H_3 = \{D\} \). Furthermore, we have \( U_A = U_B = \emptyset \), \( U_C = \{A, B\} \) and \( U_D = \{B, C\} \).

4 Model Setting

Our model is based on the single agent dynamic choice framework developed by Rust (1987) to analyze technology adoption decisions by packages of Python programming language. Each package \( i \) at time \( t \) can be described by a state variable \( S_{i,t} \). Given the current state \( S_{i,t} \), each pkg \( i \) makes an irreversible decision to adopt Py3, namely, to make the pkg
Let $d_{i,t}$ be the binary irreversible decision that she made, we have

$$d_{i,t} = \begin{cases} 
1 & \text{if package } i \text{ adopts Python 3} \\
0 & \text{otherwise.} 
\end{cases}$$

(1)

Packages are linked through dependency requirements. In order to incorporate the dependency network of packages and how packages affect each other through their adoption decisions, the state variable includes both one's own characteristics as well as those of dependencies. Thus, the state variable can be specified as the following:

$$S_{i,t} = \left\{ x_{i,t-1}, z_{i,t-1}, \nu_{i,t}, d_{i,t-1}, \{S_{j,t}, d_{j,t}\}_{j \in U_{i,t}} \right\}. $$

The dependency network among packages is incorporated in the last component of the state variable. One implicit assumption is a sequential game play, i.e. package $i$ observes the adoption decision made by its dependencies. Section 5.2 provides more discussion and implications on this assumption.

All the adoptions are irreversible, meaning that in each time period $t$, only packages without Python 3 support make the adoption decisions. The adoption decision comes with a one-time adoption cost, and it affects the transition dynamics of the state variable $S_{i,t}$.

### 4.1 Value Function and Bellman Equation

Given the state $S_{i,t}$, a package’s flow utility can be written as:

$$u(S_{i,t}, d_{i,t}; \theta) - C(S_{i,t}, d_{i,t}; \theta) + \nu_{i,t}$$

(2)

$$= u(S_{i,t}, d_{i,t}; \theta) - 1(d_{i,t-1} = 0, d_{i,t} = 1) C(S_{i,t}; \theta) + \nu_{i,t},$$

(3)

$^{16}$More examples of irreversible decisions as well as discussion can be found in Rust and Phelan (1997) and Aguirregabiria and Mira (2010).
where \( u(S_{i,t}, d_{i,t}; \theta) \) is the reward function of the indirect utility, which is a function of user downloads as discussed in Section 5.1. \( C(S_{i,t}; \theta) \) is the one-time cost to adopt Python 3.

The value function for packages without Python 3 support can be written as the following dynamic problem:

\[
V_\theta(S_{i,t}, d_{i,t-1} = 0) = \max_{\{d_{i,t+\tau}\}_{\tau=0}^{\infty}} \mathbb{E}_t \left\{ \sum_{\tau=0}^{\infty} \beta^\tau \left( u(S_{i,t+\tau}, d_{i,t+\tau}; \theta) - d_{i,t+\tau} C(S_{i,t+\tau}; \theta) + \nu_{i,t+\tau}^{d_{i,t+\tau}} \right) | S_{i,t}; \theta \right\}. \tag{4}
\]

The Bellman equation implies at each period \( t \), the ex ante value function, conditional on \( d_{i,t-1} = 0 \), can be represented by:

\[
V_\theta(S_{i,t}, d_{i,t-1} = 0) = \max_{d_{i,t} \in \{0, 1\}} u(S_{i,t}, d_{i,t}; \theta) - d_{i,t} C(S_{i,t}; \theta) + v_{i,t}^{d_{i,t}} + \beta \mathbb{E}_t V_\theta(S_{i,t+1}|S_{i,t}, d_{i,t}) \tag{5}
\]

\[
= \max\{v_{\theta}(S_{i,t}, d_{i,t-1} = 0, d_{i,t} = 0), v_{\theta}(S_{i,t}, d_{i,t-1} = 0, d_{i,t} = 1)\} \tag{6}
\]

\[
= \max\{u(S_{i,t}, d_{i,t} = 0; \theta) + v_{0,t}^{d_{i,t}} + \beta \mathbb{E}_t V_\theta(S_{i,t+1}|S_{i,t}, d_{i,t} = 0), \tag{7}
\]

\[
u_{i,t}^{d_{i,t}} = u(S_{i,t}, d_{i,t} = 1; \theta) - C(S_{i,t}; \theta) + v_{1,t}^{d_{i,t}} + \beta \mathbb{E}_t V_\theta(S_{i,t+1}|S_{i,t}, d_{i,t} = 1)\}. \tag{8}
\]

The representation of value function for packages with Python 3 support is more straightforward, since the irreversibility condition implies that no further decisions are being made.

\[
V_\theta(S_{i,t}, d_{i,t-1} = 1) = v_{\theta}(S_{i,t}, d_{i,t-1} = 1, d_{i,t} = 1) \tag{9}
\]

\[
= \mathbb{E}_t \left\{ \sum_{\tau=0}^{\infty} \beta^\tau \left( u(S_{i,t+\tau}, d_{i,t+\tau} = 1; \theta) + \nu_{i,t+\tau}^{1} \right) | S_{i,t}; \theta \right\} \tag{10}
\]

\[
= \sum_{\tau=0}^{\infty} \beta^\tau \int \left( u(S_{i,t+\tau}, d_{i,t+\tau} = 1; \theta) + \nu_{i,t+\tau}^{1} \right) dF_\theta(S_{i,t+1}|S_{i,t}). \tag{11}
\]

We assume that \( \nu_{i,t}^{d_{i,t}} \) are iid errors that follows Type 1 Extreme Value Distribution. Then the expected value function \( \mathbb{E}_t V_\theta(S_{i,t+1}|S_{i,t}, d_{i,t}) \) can be calculated using the following equation:
\begin{align}
\mathbb{E}_t V_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 0) \\
&= \int V_{\theta}(S_{i,t+1}, d_{i,t} = 0) dF_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 0) \\
&= \int \log\left\{ \sum_{d_{i,t+1} \in \{0,1\}} \exp(v_{\theta}(S_{i,t+1}, d_{i,t} = 0, d_{i,t+1})) \right\} dF_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 0)
\end{align}

\begin{align}
\mathbb{E}_t V_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 1) \\
&= \int V_{\theta}(S_{i,t+1}, d_{i,t} = 1) dF_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 1) \\
&= \sum_{\tau=1}^{\infty} \beta^{\tau-1} \int \left( u(S_{i,t+\tau}, d_{i,t+\tau} = 1; \theta) + v_{i,t+\tau} \right) dF_{\theta}(S_{i,t+1}|S_{i,t}, d_{i,t} = 1).
\end{align}

Given the parameter \( \theta \) and the transition dynamics described by \( F_{\theta} \), \( \text{EV} \) is iterated until convergence, which is then used to calculate choice-specific value functions \( v_{\theta}(S_{i,t}, d_{i,t-1} = 0, d_{i,t}) \). The choice-specific value functions allow us to compute the predicted probability of adopting Python 3 using the standard logit formula:

\begin{align}
P(d_{i,t} = 1|S_{i,t}, d_{i,t-1} = 0; \theta) &= \frac{v_{\theta}(S_{i,t}, d_{i,t-1} = 0, d_{i,t} = 1)}{\sum_{d' \in \{0,1\}} v_{\theta}(S_{i,t}, d_{i,t-1} = 0, d')} \\
P(d_{i,t} = 0|S_{i,t}, d_{i,t-1} = 0; \theta) &= 1 - P(d_{i,t} = 0|S_{i,t}, d_{i,t-1} = 0; \theta) \\
P(d_{i,t} = 1|S_{i,t}, d_{i,t-1} = 1; \theta) &= 1 \\
P(d_{i,t} = 0|S_{i,t}, d_{i,t-1} = 1; \theta) &= 0.
\end{align}

5 \hspace{1em} \textbf{Parametrization & Transition Matrix}

5.1 \hspace{1em} \textbf{Payoff Function}

As discussed in Section 2.1, we model the utility of package developers as a function of user downloads. Let \( x_{i,t} = \log(\text{DOWNLOADS}_{i,t}) \), which is the logarithm of total number of times a package \( i \) is downloaded by users at time \( t \). We specify the payoff function as a
linear function of user downloads:

\[ u(S_{i,t}, d_{i,t}; \theta) = \alpha^x x_{i,t}(x_{i,t-1}, d_{i,t}; \theta). \] (25)

The evolution of \( x_{i,t} \) relies on the package’s adoption status/decision \( d_{i,t} \). We assume that the evolution of the demand following a first-order Markov process:

\[ x_{i,t} = \rho_0 + \rho_{ar} \cdot d_{i,t} \cdot AR_t + \rho_1 \cdot x_{i,t-1} + \nu_{i,t}^d \text{ with } d = \begin{cases} 1 & \sum_{\tau \leq t} d_{i,\tau} = 1 \\ 0 & \sum_{\tau \leq t} d_{i,\tau} = 0 \end{cases}. \] (26)

\[ d = \begin{cases} 1 & \sum_{\tau \leq t} d_{i,\tau} = 1 \\ 0 & \sum_{\tau \leq t} d_{i,\tau} = 0 \end{cases}. \] (27)

where \( d \) indicates if the package \( i \) has adopted before \( t + 1 \), \( z_i \) is a time-invariant vector capturing the package \( i \)'s specific characteristics, \( AR_t \) is the current Python 3 adoption rate among all packages, \( \nu_{i,t}^0 \) and \( \nu_{i,t}^1 \) are two white noises that are normally and interdependently distributed with mean 0 and variances \( \sigma_{d_0}^2 \) and \( \sigma_{d_1}^2 \).

The AR1 evolution of user demand is estimated outside the model of Python 3 adoption decisions by package developers. Then the estimates are used as input to the structural model.

The estimates of the AR1 demand equation essentially serves as the counterfactual demand for a package deciding whether to adopt Python 3. However, the estimates of the AR1 process might suffer from endogeneity problem. The AR1 process is estimated using the evolution of user downloads in a result of the actual adoption decision. Those packages who have adopted Python 3 might experience a positive persistent demand shock that is unobservable to the econometrician. In another word, due to the endogeneity issue, the benefit of Python 3 adoption inferred by the AR1 estimates can be over-exaggerated.

In order to control for the endogeneity, we introduce a two-step estimation approach using a synthetic instrumental variable. Given an initial AR1 estimates, first we estimate the model of technology adoption. Then in the second step, we use the predicted adoption probability as an instrumental for the endogenous variable \( d_{i,t} \) and re-estimate the AR1 process. Then
the new estimates are fed back to step 1 to re-estimate the model of technology adoption. Step 1 and 2 are conducted interactively until all the estimates converge to a fixed point. More discussions on identification can be found in Section 6.1.

5.2 Adoption Cost

We model the cost function as a function of agent’s decision \( d_{i,t} \), the number of coding lines (internal cost) \( code_i \) and the network constraint \( \mu_{i,t} \) that represents the number of dependencies without Python 3 support at time \( t \), adjusted by an “importance” measure of that dependency. The switching cost is thereby defined as below:

\[
C_{i,t} = AC_0 + \alpha \mu_{i,t},
\]

and we further assume that \( \mu_{i,t} \) admits the following representation:

\[
\mu_{i,t} = \sum_{j \in U_i} \mathbb{1}\{d_{j,t} = 0\}.
\]

\( \mu_{i,t} \) is built as a raw measure of the adoption cost due to incompatible dependencies. In reality, such barrier might differ across different dependencies. For example, it might be much easier to find a dependency alternative to another small dependency compared to a large one. In the estimation stage, we also include \( \mu_{i,t}^{adj} = \sum_{j \in U_i} \mathbb{1}\{d_{j,t} = 0\} z_{i,t} \), that is, \( \mu_{i,t} \) adjusted using the characteristics of the dependencies.

**Assumption 1.** At time \( t \), a package \( i \) observes Python 3 adoption decisions made by its dependencies, namely, \( d_{j,t} \) for all \( j \in U_{i,t} \).

This is an assumption on the information set available to a package when making decisions. In particular, the question is that if a dependency \( j \in U_{i,t} \) makes adoption decision at time \( t \), would package \( i \) know it or not? If not, then it’s more appropriate to model the game as a simultaneous-move game where the decisions are based on information from the previous period. If yes (as stated in Assumption 1), then it’s more appropriate to model it
as a sequential-move game, and the downstream packages have better ideas of their adoption costs.

Assumption 1 simplifies our model estimation only slightly. Since the adoption cost comes from the dependencies without Python 3 support, at time $t$, a package $i$ has to weigh the tradeoff between adopting today vs later, taking into consideration of the adoption probability of each of the Python 3-incompatible dependencies. With this assumption, the set of dependencies without Python 3 support at time $t$ is weakly smaller, which can alleviate some computational burden in certain cases.

We prefer the latter setup for the following reasons: First, at any moment in time, a upstream package tend to have more public information available regarding the plans for future releases. A upstream package is usually much more popular compared to its downstream counterparts (in terms of downloads). They tend to pre-announce their plans for future releases, including Python 3 adoption decisions. Second, on the other hand, for a downstream package who wants to adopt Python 3 but faces incompatible dependencies, it’s likely that it would pay more attention to such decision by its dependencies. In such a case, it is more likely that the Python 3 adoption decisions of upstream packages are available to downstream packages as soon as they are made.

Assumption 1 implies a sequential-move game where upstream packages make decisions first, followed by the downstream packages. The exact order of play in the model is defined based on the network structures, and it will be specified in detail in later sections.

We do not think that a simultaneous-move game (without Assumption 1) and a sequential one (with Assumption 1) would produce significantly different results. The only observations that can give rise to different estimates are cases when a package and its dependencies adopt Python 3 in the same time period, which are rare occasions in the data.

If package $i$ observes $d_{j,t} = 1$, then the perceived AC is much smaller than the case without observing $d_{j,t}$. In this case, the cost parameter with a simultaneous-move model is underestimated compared to a sequential one.
5.3 State Variables & Transition Probability

As mentioned in the previous sections, one important component of the adoption cost comes from the dependency network when the dependency packages lack Python 3 support, which is represented by the last component of the state variables $\{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}}$. We model a dynamic model of sequential decisions where upstream packages make adoption decisions before downstream packages, and downstream packages observe the decisions made by upstream packages at time $t$, namely $d_{j,t}$.

Package $i$ cares about $\{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}}$ in so far as it cares about its own adoption cost (a function of $\mu_{i,t}$), as well as its future evolution based given the current states. The law of motion of $\mu_{i,t}$ can be computed in the following way:

$$P(\mu'_{i,t+1} = \mu' | \mu_{i,t} = \mu, \{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}}; \theta) = P(\mu' - \mu = \Delta \mu | \{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}}; \theta),$$

where $\Delta \mu$ is the change of the number of packages lacking Python 3 support from time $t$ to $t+1$ and $\Delta \mu \in \{0, -1, -2, ..., -\mu\}$, i.e. adoption cost in future periods might be lower. Again, $\{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}}$ is important in predicting dependency $j$’s probability of Python 3 adoption at a future date. Denote package $i$’s belief that its dependency $j$ will adopt Python 3 at time $t+1$ as $\hat{p}_{j,t+1} = P(d_{j,t+1} = 1 | d_{j,t} = 0, S_{j,t}; \theta)$, and $\hat{p}_{j,t+1}^0 = P(d_{j,t+1} = 0 | d_{j,t} = 0, S_{j,t}; \theta)$.

Then we can write equation [30] as:

$$P(\mu' - \mu = \Delta \mu | \{\hat{p}_{j,t+1} \}_{j \in U_{i,t}}; \theta).$$

With $\mu_{i,t} \equiv \sum_{j \in U_{i,t}} 1(d_{j,t} = 0)$, we can further simplify [31] by denoting the set of dependencies without Python 3 support as $\Omega_{i,t} \equiv \{j \in U_{i,t} | d_{j,t} = 0\}$, thus $\mu_{i,t} = |\Omega_{i,t}|$.

Then equation [31] can be written in the following way:

$$P(\mu' - \mu = \Delta \mu | \{\hat{p}_{j,t+1} \}_{j \in \Omega_{i,t}}; \theta).$$

The adoption decisions by different packages in $\Omega_{i,t}$ can lead to the same value of $\Delta \mu$. 
Define $\mathcal{O}_{i,t}$ as the power set of $\Omega_{i,t}$ that contains all possible subsets of $\Omega_{i,t}$. Further we denote $\mathcal{O}_{i,t}^k = \{ o \in \mathcal{O}_{i,t} \mid |o| = k \}$, i.e. all the elements of set $\mathcal{O}_{i,t}$ with the same cardinality of $k$.

Then equation 32 can be written as:

$$
\sum_{\mathcal{O}' \in \mathcal{O}_{i,t}^\prime} \mathbf{P}(\mathcal{O}' \mid \Omega_{i,t}, \{ \hat{p}_{j,t+1} \}_{j \in \Omega_{i,t}}, \theta) = \sum_{\mathcal{O}' \in \mathcal{O}_{i,t}^\prime} \left( \prod_{j \in \mathcal{O}'} \hat{p}_{j,t+1}^0 \prod_{j \in \Omega_{i,t} \setminus \mathcal{O}'} \hat{p}_{j,t+1}^1 \right).
$$

The burdensome mathematical notation can be easily understood with a simple example. Suppose that at time $t$, package $i$ has 3 dependencies $U_{i,t} = \{ A, B, C \}$, and $d_{A,t} = 0$, $d_{B,t} = 0$, $d_{C,t} = 1$, leaving the set of dependency without Python 3 support being $\Omega_{i,t} = \{ A, B \}$. Suppose package $i$'s belief that each of the dependency will adopt Python 3 at time $t+1$ with the following probability: $\hat{p}_{A,t+1}^0 = a$, $\hat{p}_{A,t+1}^1 = 1 - a$, $\hat{p}_{B,t+1}^0 = b$, $\hat{p}_{B,t+1}^1 = 1 - b$.

The powerset of $\Omega_{i,t}$ is $\mathcal{O}_{i,t} = \{ \emptyset, \{ A \}, \{ B \}, \{ A, B \} \}$, $\mathcal{O}_{i,t}^0 = \{ \emptyset \}$, $\mathcal{O}_{i,t}^1 = \{ \{ A \}, \{ B \} \}$, $\mathcal{O}_{i,t}^2 = \{ \{ A, B \} \}$. Following equation 34 the transition from $\mu = 2$ to $\mu' = 1$ can be calculated as:

$$
\sum_{\mathcal{O}' \in \mathcal{O}_{i,t}^1, \mathcal{O}' = \{ \{ A \}, \{ B \} \}} \left( \prod_{j \in \mathcal{O}'} \hat{p}_{j,t+1}^0 \prod_{j \in \Omega_{i,t} \setminus \mathcal{O}'} \hat{p}_{j,t+1}^1 \right) = \prod_{j \in \{ A \}} \hat{p}_{j,t+1}^0 \prod_{j \in \{ A, B \} \setminus \{ A \}} \hat{p}_{j,t+1}^1 + \prod_{j \in \{ B \}} \hat{p}_{j,t+1}^0 \prod_{j \in \{ A, B \} \setminus \{ B \}} \hat{p}_{j,t+1}^1
$$

$$
= \hat{p}_{A,t+1}^0 \cdot \hat{p}_{B,t+1}^1 + \hat{p}_{B,t+1}^0 \cdot \hat{p}_{A,t+1}^1
$$

$$
= (1 - a) b + a (1 - b).
$$

### 5.4 Transition Matrix

The calculation of EV in equation 17 depends crucially on the specification of the transition matrix, or $P_{S'|S}$ in equation 17.

In our model, the utility function is mainly governed by two variables, namely, $x_{i,t}$,
the measure of downloads or popularity, and $\mu_{i,t}$, the measure of adoption cost due to dependencies. The construction of the transition matrix depends on the joint law of motion of $x_{i,t}$ and $\mu_{i,t}$.

The law of motion of $x_{i,t}$ is relatively easy. We assume that it follows an AR1 process as specified in equation \[26\] with the parameter values estimated outside the value function iteration.

On the other hand, the law of motion of $\mu_{i,t}$ is much more difficult. $\mu_{i,t} \equiv \Omega_{i,t} \equiv \sum_{j \in U_{i,t}} 1(d_{j,t} = 0)$, i.e. the number of dependencies of package $i$ without Python 3 support at time $t$.

One of the most important tradeoffs for package $i$ to adopt Python 3 today at $t$ vs. future periods $t + \tau$ is the decreasing adoption cost due to the decreasing number of dependencies without Python 3 support over time. Therefore, the solution to the dynamic adoption model depends on the calculation of future adoption probabilities for each of the dependencies.

The calculation is a formidable task due to the nature of the nested dependency network. The computational difficulty can be illustrated by forecasting the Python 3 adoption probability for each of package $i$’s dependency $j \in U_{i,t}$ at time $t + 1$:

$$\hat{p}_{j,t+1} = \int_{S_{j,t+1}} P(d_{j,t+1} = 1|S_{j,t+1}, d_{j,t} = 0, z_j; \theta) dP(S_{j,t+1}|S_{j,t}, d_{j,t} = 0, z_j; \theta).$$

The integral can then be computed through simulation of future states $S_{j,t+1}$. Let $S_{j,t+1}^m$ be the $m$th simulation, than the value of $\hat{p}_{j,t+1}$ can be obtained by:

$$\hat{p}_{j,t+1} = \frac{1}{M} \sum_{m=1}^{M} P(d_{j,t+1} = 1|S_{j,t+1}^m, d_{j,t} = 0, z_j; \theta).$$

This simulation method can be very computationally intensive. Note that the nested states are expressed as $(x_{i,t-1}, d_{i,t-1}, \nu_{i,t}, \{d_{j,t}, S_{j,t}\}_{j \in U_{i,t}})$. The simulation of the states of package $j$ requires the simulation of the states of $j$’s dependency $k \in U_{j,t+1}$, as well as the states of $k$’s dependency, and so on. Any slight changes in any of the linked dependencies, or the dependencies of each dependency, can affect $p_{j,t}$. In this way, the full solution approach by \cite{Rust1987} is no longer feasible due to the curse of dimensionality problem. In fact,
with the 3102 packages and 13056 observations, it is simpler to build the transition matrix dynamically for each package \( i \) at each time period \( t \). In later sections, we list the detailed steps regarding how to compute \( p_{j,t} \) for \( j \in \Omega_{i,t} \) given the hierarchical network.

A transition matrix used for the dynamic programming problem includes the transition probability not only from the current state to the next, but also from each of the possible states to all other states. Given the state \( S_{i,t} \), package \( i \) calculates \( \hat{p}_{j,t} \) for each of \( j \in \Omega_{i,t} \).

To construct the transition matrix, the belief of \( \hat{p}_{j,t+\tau} \) for \( \tau \in \mathbb{N} \) is also needed.

**Assumption 2.** Package \( i \) holds myopic expectations regarding the future Python 3 adoption probabilities by its dependencies, i.e. \( \hat{p}_{j,t+\tau} = \hat{p}_{j,t} \) for all \( \tau \in \mathbb{N} \).

This assumption implies that although package \( i \) can correctly calculate the adoption probability of its dependencies at time \( t \), the ability to forecast future probabilities is limited.

Assumption 2 greatly simplifies the model estimation. In a way, it bears certain similarity to the assumptions of inclusive value function, which is assumed to follow an AR1 process, in many previous papers (see Lee (2013) and Gowrisankaran and Rysman (2012)). In future versions of this paper, we also plan to explore the possibility to allow the forecast of the adoption probability to follow such an AR1 process.

Combining the transition probability specified in equation 34 and Assumption 2, the full transition matrix needed to calculate \( EV(S, d = 0; \theta) \) can be specified as the following:

\[
P(o' \in O_{i,t} | o \in O_{i,t}) = \begin{cases} 
0 & \text{if } o \notin o' \\
\prod_{j \in o'} \hat{p}_{j,t}^0 \prod_{j \in o \setminus o'} \hat{p}_{j,t}^1 & \text{if } o \subseteq o'.
\end{cases}
\]  

(41)

The calculation of the transition matrix, as specified in equation 41, can be illustrated using the same example in section 5.3. Recall that the set of dependencies without Python 3 support is \( \Omega_{i,t} = \{A, B\} \). The adoption probability of A and B at time \( t \) can be calculated by package \( i \). Assume that \( \hat{p}_{A,t}^1 = a \) and \( \hat{p}_{B,t}^1 = b \). Assumption 2 implies that \( \hat{p}_{A,t+\tau} = a \) and \( \hat{p}_{B,t+\tau} = b \) for all \( \tau \in \mathbb{N} \). The powerset of \( \Omega_{i,t} \) is \( O_{i,t} = \{\emptyset, \{A\}, \{B\}, \{A, B\}\} \), and
The transition matrix of $\mu$ can be calculated using equation 41:

$$TM(\mu_{i,t}) = \begin{bmatrix}
\{A, B\} & \{A\} & \{B\} & \emptyset \\
\{A, B\} & (1-a)(1-b) & a(1-b) & (1-a)b & ab \\
\{A\} & 0 & 1-a & 0 & a \\
\{B\} & 0 & 0 & 1-b & b \\
\emptyset & 0 & 0 & 0 & 1
\end{bmatrix}.$$

The corresponding value of $\mu$ for each row (or column) is 2, 1, 1, 0, respectively. By construction, the transition matrix considers both the number and the identities of dependencies that are without Python 3 support. For example, the situation with dependency A being the only one without Python 3 support is different from the situation when B is the only one. Both cases have the same value of $\mu$ which is 1, thus having the same adoption cost, but the value of waiting is different because A and B have different likelihood of adoption in future periods. The transition matrix calculated using equation 41 takes cares of all such cases.

6 Identification & Estimation

6.1 Identification

In this section, we provide a brief discussion about the identification of our structural parameters.

The law of motion of $x$ is modeled as a first-order Markov process, and it can be identified from the variation of $x_t$ over time. Following the existing literature of dynamic discrete choice models (Keane (1994), Keane and Wolpin (1997)), we estimate $AR(1)$ separately from the structural model of technology adoption for computational purposes.

Once the law of motion of $x$ is determined, one would be able to identify the fixed cost $AC_0$, $\alpha^\mu$ through the predicted variation of $\Delta\mu$ as a function of $(\mu, x, d, \theta)$.

In most setting of dynamic discrete choice models, the discount factor is typically assumed
and is not able to be separately identified from other parameters (Magnac and Thesmar (2002)). The identification of the discount factor requires variations that can shift expected discounted future utilities, but not current utilities (Abbring and Daljord (2018), De Groote and Verboven (2018)). We are able to identify the discount factor through the differences in dependency characteristics, which affect future values but not current utility. For example, two packages with the same characteristics and the same number but different dependencies have the same utility in the current period. However, the different dependencies have different future adoption probabilities, thus different transition matrix. If packages are myopic ($\beta = 0$), the two packages with different dependency characteristics should behave the same way; if packages are forward-looking ($\beta > 0$), larger differences in dependency characteristics should have a larger impact on the differences between the adoption probabilities of the two packages.

By assuming all packages share the same discount factor $\beta$, the variations of downloads allows us to identify $\alpha^x$.

6.2 Estimation

Our model of technology adoption is estimated using Maximum Likelihood Estimation (MLE). The likelihood function is defined as:

$$l(\theta) = \prod_{i=1}^{N} \prod_{t=1}^{T_i} \hat{p}_{i,t}^{0 \{d_{i,t}=0\}} \hat{p}_{i,t}^{1 \{d_{i,t}=1\}},$$

where $\hat{p}_{i,t}^1 \equiv \hat{p}(S_{i,t}; \theta)$, which is defined in equation [21].

6.3 Parameters

The parameters of interest in our model are the following:

$$\theta = \{\theta_D; \theta_P, \rho_{ar}; \alpha^x, AC_0, \alpha^\mu, \alpha^{size}, \beta\}.$$
It is convenient to group the parameters into $\theta_D$ and $\theta_S$. $\theta_D$ includes parameters for the demand function, which is modeled as an AR1 process of user downloads; $\theta_S$ includes parameters for the supply side, which is the model of technology adoption.

### 6.4 Hierarchical Network

The adoption decision of a package $i$ depends crucially on the adoption status of its dependencies, which is summarized as $\mu_{i,t}$ in the utility function. Further, Assumption X states that $\mu_{i,t}$ is known to package $i$ before making decisions at $t$. Assumption X allows us to model the adoption decisions sequentially. In every time period $t$, packages in layer 0 decide to adopt Python 3, followed by those in layer 1, then those in layer 2,..., etc. The probability of adopting Python 3 for each of the upstream packages is then used as given for the decision making process of the downstream package, in order to calculate the transition probability of $\mu_{i,t}$.

Through the dependency requirement, we group packages into different layers: $0, 1, 2, \cdots, L$. Figure 6 shows an example of the layered network. Each layer contains a set of packages with dependencies in the upstream layers. They can be grouped using the following iteration:

$$\mathcal{L}_0 = \{i \mid |U_i| = 0\}$$
\[ \mathcal{L}_1 = \{ i \mid U_i \subseteq \mathcal{L}_0 \} \]
\[ \mathcal{L}_2 = \{ i \mid U_i \subseteq \mathcal{L}_{0,1} \} \]
\[ \vdots \]
\[ \mathcal{L}_t = \{ i \mid U_i \subseteq \mathcal{L}_{0,1,\ldots,t-1} \} \]
\[ \vdots \]
\[ \mathcal{L}_L = \{ i \mid U_i \subseteq \mathcal{L}_{0,1,\ldots,L-1} \}. \]

where \( \mathcal{L}_{0,1,\ldots,t} = \mathcal{L}_0 + \mathcal{L}_1 + \cdots + \mathcal{L}_t \)

### 6.5 Estimation Procedure

Our algorithm begins by setting initial values for \( \theta_D \), which comes from OLS estimation of the AR1 process of user downloads, specified in equation \[26\]. Estimation then involves iterating on four steps, where the \( m \)th iteration follows:

**Step 1: Estimation of the Model of Technology Adoption**

- Given estimates of demand function \( \theta_D^m \) and initial guess of \( \theta_S = \{ \alpha_x, \alpha^u, AC_0 \} \):
  - for \( i \in \mathcal{L}_0 \), build transition matrix for each \( i, t \) and calculate \( \hat{p}^1(S_{i,t}; \theta) \)
  - for \( i \in \mathcal{L}_1 \), given \( \hat{p}^1(S_{j,t}; \theta) \ \forall j \in \mathcal{L}_0 \), build transition matrix for each \( i, t \) and calculate \( \hat{p}^1(S_{i,t}; \theta) \)
  - for \( i \in \mathcal{L}_2 \), given \( \hat{p}^1(S_{j,t}; \theta) \ \forall j \in \mathcal{L}_{0,1} \), build transition matrix for each \( i, t \) and calculate \( \hat{p}^1(S_{i,t}; \theta) \)
    \[ \vdots \]
  - for \( i \in \mathcal{L}_t \), given \( \hat{p}^1(S_{j,t}; \theta) \ \forall j \in \mathcal{L}_{0,1,\ldots,t-1} \), build transition matrix for each \( i, t \) and calculate \( \hat{p}^1(S_{i,t}; \theta) \)
    \[ \vdots \]
  - for \( i \in \mathcal{L}_L \), given \( \hat{p}^1(S_{j,t}; \theta) \ \forall j \in \mathcal{L}_{0,1,\ldots,L-1} \), build transition matrix for each \( i, t \) and calculate \( \hat{p}^1(S_{i,t}; \theta) \)
• Calculate likelihood function $l(\theta)$

• Update $\theta$ such that $\theta_{S}^{m+1} = \arg\max_{\theta_S} l(\theta_S; \theta_{D}^{m})$

**Step 2: Re-estimate Demand Using IV**

• Given $\theta_{D}^{m}$ and $\theta_{S}^{m+1}$, calculate the predicted adoption probability $\hat{p}_i^1(S_{i,t}; \theta_{D}^{m}, \theta_{S}^{m+1})$ for all $i, t$

• Re-estimate the AR1 process of demand function using $\hat{p}_i^1$ as IV for $D_{i,t}$, and get a new set of estimates $\theta_{D}^{m+1}$ using the standard IV estimation method.

**7 Preliminary Results**

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_{ar}$ $d_{i,t} \times AR_{t}$</td>
<td>0.165***</td>
<td>0.074***</td>
</tr>
<tr>
<td></td>
<td>(0.01)</td>
<td>(0.01)</td>
</tr>
<tr>
<td>$\rho_1$ $x_{i,t-1}$</td>
<td>0.898***</td>
<td>0.902***</td>
</tr>
<tr>
<td></td>
<td>(0.00)</td>
<td>(0.00)</td>
</tr>
<tr>
<td>$\rho_0$ Constant</td>
<td>1.069***</td>
<td>1.061***</td>
</tr>
<tr>
<td></td>
<td>(0.02)</td>
<td>(0.02)</td>
</tr>
<tr>
<td>$N$</td>
<td>54230</td>
<td>54230</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.804</td>
<td>0.803</td>
</tr>
</tbody>
</table>

Table 5 reports the parameter estimates of the AR1 process of user downloads, specified in equation 26. Both OLS and IV estimation show that once a package adopts Python 3, the demand function process changes significantly.

For the same level of user DL $x_{i,t-1}$, OLS estimates predict a higher $x_{i,t}$ from Python 3 adoption than IV estimation does. The differences between OLS and IV estimates provides more clues of endogeneity. In reality, many packages adopt Python 3 even though having a low level of downloads, probably due to an unobserved positive download shock associated to Python 3, namely, a large $\epsilon_{i,t}^1$. 
Table 6: Estimated Parameters of Adoption Model

<table>
<thead>
<tr>
<th>Nonlinear Parameters ($\theta_S$)</th>
<th>$\beta$</th>
<th>0.705*** (0.074)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha^x$</td>
<td>3.461*** (1.21)</td>
</tr>
<tr>
<td></td>
<td>$AC_0$</td>
<td>-2.484*** (0.099)</td>
</tr>
<tr>
<td></td>
<td>$\alpha^\mu$</td>
<td>-0.145*** (0.021)</td>
</tr>
<tr>
<td></td>
<td>$\alpha^{size}$</td>
<td>-0.045** (0.015)</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-8143</td>
<td></td>
</tr>
<tr>
<td>Number of Packages</td>
<td>4005</td>
<td></td>
</tr>
<tr>
<td>Number of Observations</td>
<td>23267</td>
<td></td>
</tr>
</tbody>
</table>

Table 6 summarizes the estimation results of our model of technology adoption. All the coefficients are scaled by the variance of the error terms. The estimated discount factor 0.705 is equivalent to a monthly discount factor of 0.943, which is comparable to the estimates in other papers, such as 0.943 (Lee (2013)) and 0.988 (De Groote and Verboven (2018)). In future versions, we will provide more discussion on the identifiability of the discount factor. The coefficient of $\alpha^x$, combined with the estimates in table 5, shows that developers benefit from more user downloads. The negative estimate of $AC_0$ shows a significant fixed adoption cost. Relative to the fixed cost, incompatible dependencies also serve as significant barriers to adoption. Note that the measure of $\mu_{i,t}$ used in the estimation is the number of incompatible dependencies weighted by the size of packages. The average size of a dependency is 4.89, thus, the average adoption cost due to one incompatible dependency is -0.71.

8 Model Prediction and Counterfactuals

This section is largely preliminary and incomplete. Results are likely to change in future versions.

Figure 7 compares the actual adoption rates against the model prediction. The adoption rates is calculated as the percentage of packages with Python 3 support among the sample of 4005 packages used in the model estimation.
Figure 7: Counterfactual Adoption Rates
Our preliminary model of technology adoption gives a poor fit of the actual adoption rates, especially in the later periods. One most likely reason of over-prediction in early periods but under-prediction in later periods is due to the myopic belief of upstream packages’ future adoption probability. If an incompatible dependency has a low adoption probability, the myopic assumption assumes low adoption probability for future periods, thus the downstream package is more likely to adopt Python 3 than in cases where the adoption probability of the dependency is rising over time. We are actively working on relaxing the myopic assumption to allow the future prediction to follow a first-order Markov process, following the assumption widely used in the literature of inclusive value in similar dynamic models \cite{Gowrisankaran2012, Lee2013}.

8.1 More Ideas on Counterfactual Policies

Katz and Shapiro\cite{Katz1985} predict that the success of a new technology and the speed of technology adoption depends highly on the “sponsorship”. A sponsor is an entity that is willing to make investment to promote it. Python is managed by the Python Foundation, and it is distributed free of charge as an OSS. The Python Foundation has been promoting for a faster transition to Python 3. However, given the limited, probably exogenous amount of effort, it is key to understand how the effect can be spent most effectively in order to help the whole industry to switch to Python 3 at a fast pace.

In this section, we strive to gain more insights on the question of how to target certain packages in order to facilitate a fast transition to Python 3. We assume that the Python Foundation has limited resources to spend on the promoting activity, which has heterogeneous effects on the adoption decisions of packages based on their characteristics. We assume that the objective is to maximize the overall adoption rate of Python 3 by a certain time, say 2015. The Python Foundation maximizes that objective given limited amount of resources.

Intuitively, a good target for promotion is one that satisfies the following criteria: (1) low promotion cost (e.g. fewer lines of code) (2) promotion is effective, i.e. the difference in adoption probability with and without promotion is large (3) key players in the network, namely, decisions of the targeted package can affect the decisions of many others.
9 Conclusion

New technologies are being developed faster than ever. But many fail to attract quick and widespread adoption. In many cases, a fast adoption of new technologies can be socially beneficial: slow adoption often leads to long time of incompatible products, and a more rapid adoption enables consumers to better enjoy the convenience brought by the latest technology.

Economic theory has long proposed the barrier to adopt new technology due to existing network effects. Our research explores the ways technology adoption can be affected due to the network structure in a disaggregated hierarchical network. We find strong evidence that the adoption decisions of downstream players are significantly affected by their upstream counterparts.

This paper also developed a framework to measure the impact of such hierarchical network on technology adoption. We extend existing dynamic choice models to incorporate the hierarchical network among Python packages. Taking advantage of the unidirectional property of this network, we detailed and estimated a computationally tractable model that allow each agent to anticipate future actions of others. We believe that the structural framework can be applied to analyze other industries with such a vertical network structure.
References


33


