Using Machine Translation for Converting *Python 2* to *Python 3* Code

Karan Aggarwal, Mohammad Salameh, and Abram Hindle
Department of Computing Science
Univeristy of Alberta
Edmonton, Canada
{kaggarwa,msalameh,abram.hindle}@ualberta.ca

Abstract

In this paper, we have tried to use Statistical machine translation in order to convert Python 2 code to Python 3 code. We use data from two projects and achieve a high BLEU score. We also investigate the cross-project training and testing to analyze the errors so as to ascertain differences with previous case. We have described a pilot study on modeling programming languages as natural language to build translation models on the lines of natural languages. This can be further worked on to translate between versions of a programming language or cross-programming-languages code translation.

1 Introduction

Statistical machine translation(SMT) techniques have been successfully used for the translation task between natural languages. In this paper, we construct the translation model of translating programming languages - from python 2 to python 3. We are trying to investigate the application of SMT techniques, as done traditionally in natural languages to the programming languages.

Python 2 is quite similar to the Python 3 except for a few syntactical differences. We use code data from two projects - `Django` and `NLTK` written in python 2. From this data, we constructed a corpus of python 2 methods and their corresponding python 3 methods.

We also address one more question - does using these techniques produces different results for models trained on one project and tested for translation on other project. In order to do that, we train the translation model on the `Django` and test it on the `NLTK`. Thirdly, we also test this model on the test data in the first case - consisting of `Django` and `NLTK` code to see comparative results.

We achieved an improvement of 1.29 over the baseline of 98.07 BLEU score obtained with simple rule based baseline system. Moreover, in the second case of training on `Django` and testing on `NLTK`, we get an improvement of 1.1 over the baseline score of 97.36 BLUE score. In the third case, we get an improvement of 1.23 over the baseline score of 98.07.

2 Related Works

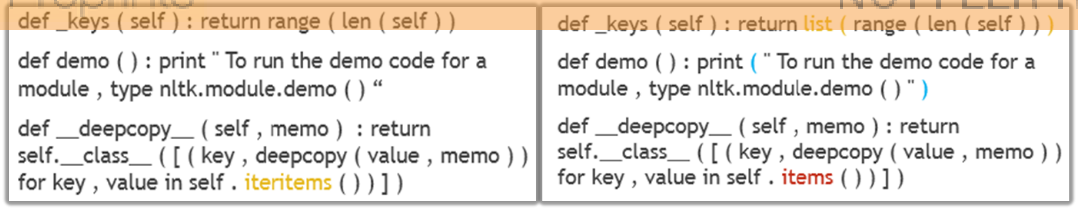
Not much work has been done in the area to best of our knowledge. Hindle et al. (2012) have constructed a simple *n*-grams model to model the programming languages as natural language. They have proposed to use natural processing techniques, and statistical machine translation approach for tasks like code summarization.
Haiduc et al. (2010) have worked on summarizing the source code by using Latent Semantic Indexing and Vector Space Model to generate summaries for a java code. Recent works as Nguyen et al. (2013), and Movshovitz-Attias and Cohen (2013) have used *n-grams* language models to show the predictability of the software, and using that to generate the project documentation for the source code. However, we found no works which use statistical machine translation techniques for the related tasks.

3 Background

In this section we describe briefly the differences between python 2 and python 3, and about the machine translation system.

3.1 Differences between Python 2 and Python 3

There are some differences in Python 2 and Python 3, though not major. Python 3 was introduced in 2009 while Python 2 had been ongoing since early 2000s. Some major changes are replacing print statement in python 2 with print function in python



```
def _keys ( self ) : return range ( len ( self ) )

def demo ( ) : print " To run the demo code for a
module , type nltk.module.demo ( ) “

def __deepcopy__ ( self , memo )  : return
self.__class__ ( [ ( key , deepcopy ( value , memo ) )
for key , value in self . iteritems ( ) ) ] )
```

```
def _keys ( self ) : return list ( range ( len ( self ) ) )

def demo ( ) : print ( " To run the demo code for a
module , type nltk.module.demo ( ) " )

def __deepcopy__ ( self , memo ) : return
self.__class__ ( [ ( key , deepcopy ( value , memo ) )
for key , value in self . items ( ) ) ] )
```

Figure 1: Three instances from the bilingual dataset. Image on the Left shows code in python 2 whereas image on right shows the code in python 3. Differences in both the codes is highlighted with colors.

3, treating all the strings as unicode in Python 3, removal of `long` datatype in python 3 and using list() with the range function.

3.2 Machine Translation System

The basic task of the machine translation systems is to maximize the probability of a translated sentence in target language given the sentence in the original language. For our problem, we can write this as :

$$P(py3|py2) = P(py2|py3)P(py3) \tag{1}$$

The probablility $P(py2|py3)$, is constructed by using sentence alignments in the translation model, where as $P(py3)$ is calculated using the language model constructed from python 3 code base. Figure 2 shows the workflow for the construction and evaluation on this translation model.

4 Methodology

In this section we describe our methodology - construction of the dataset, preprocessing steps, and baseline system.

4.1 Dataset Construction

As the translation system requires aligned sentences as input, we used the methods in the python code to align with the corresponding method in the python 3 code. This is so because, one line of code in python 2 can be aligned to two lines of code in python 3 code. Hence, it becomes difficult to align them programatically. Methods represent a semantic unit which can treated independently just like the individual sentences in the natural languages. We firstly collected two python 2 projects - `Django` and `NLTK`. Using an automated tool, 2to3 by Peterson (2014), we get the corresponding code base in the python 3. Then, we parsed the code to extract the methods from the original python 2 code base and the corresponding python 3 code base. So that, we get a bilingual corpus of the methods in the python 2 and their corresponding methods in python 3. We extracted 8114 and 4820 methods from the code base of `Django` and `NLTK` respectively. Figure 1 shows the three instances of this bilingual dataset.

Training Set	Translation	Baseline
`Django+NLTK`	99.36%	98.07%
`Django` tested on `NLTK`	98.46%	97.36%
`Django` tested on `Django+NLTK`	99.30%	98.07%

Table 1: BLEU score results on the different training and testing data.

4.2 Preprocessing

Preprocessing involves removing all the comments in the code, replacing the special characters like ’ |’ with OR , which is used as a separator character by the translation tool. We need to add spaces before and after some characters like - ’(’, ’)’ , ’”’ etc. as they need to treated as different tokens as opposed to say tokens like ’print(’.

4.3 Baseline System

We use rule based implementation for devising a baseline system. We used the following rules to construct the baseline system - replacing `print` with `print()`, replacing raw_input with input, range(x) with list(range(x)) etc. in the python 2 dataset to get our approximate translation.

5 Results and analyses

We use Giza++ and Moses tools for the sentence alignment and building the translation model respectively. This gives us the python 3 language model as well as the translation model. The python 3 language model is a simple n-grams model with n = 1,2, 3 , 4 and 5.

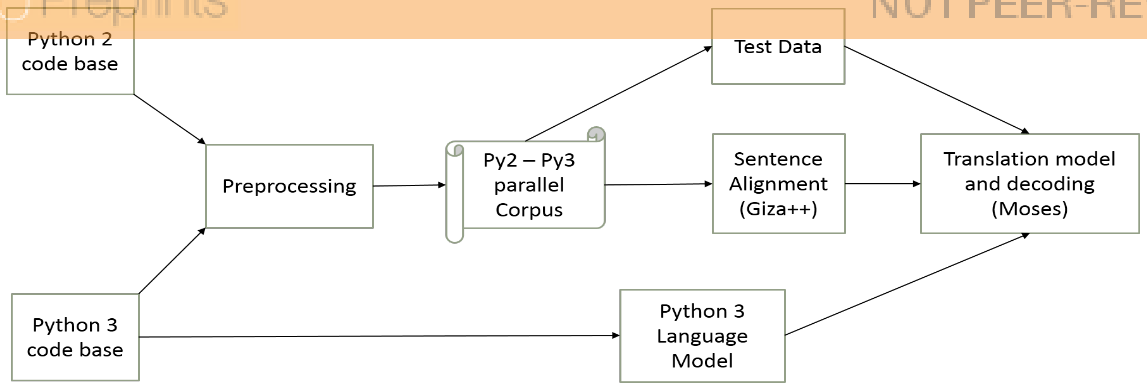

Python 2 code base
Preprocessing
Py2 – Py3 parallel Corpus
Test Data
Sentence Alignment (Giza++)
Translation model and decoding (Moses)
Python 3 code base
Python 3 Language Model

Figure 2: Work flow of the translation system

5.1 Experiment

For the first experiment, we set aside 1000 methods each for testing and development data. For the second experiment also we set aside 1000 methods from `Django` for development data, and 1000 methods from `NLTK` the testing data.
Similarly, for the third experiment we set 1000 methods from `Django` for development data and take the same test data as the first experiment (without overlap with development or training data). All these datasets were constructed randomly.

5.2 Results

The results are shown in Table 1. We can see that, with the machine translation approach, we achieve a score of 99.37, an improvement of 1.29 or 1.31%over the baseline BLEU score of 98.07 when training , development as well as test data is from both the projects.
When instead we use training data from `Django` and test it on the data from the `NLTK` we see an improvement of 1.1 or 1.12% points with BLEU score of the 98.46 over the baseline system's score of 97.36. In the third experiment, with the training data from `Django` and testing data same as the first experiment, we get a BLEU score of 99.30.

5.3 Analysis

Given these high BLEU scores, we can say that python 2 and python 3 code are quite similar except few major differences. The improvement shown by the translation system is significant, though not huge, given the fact that languages are so similar.
A relatively lower performance in the second case and third case can be explained by the difference in the training data and the testing data, as two projects use different packages. For example, the `NLTK` project uses packages *tkinder* and *urllib2* in python 2 code which have been revised to *Tkinter* and *urllib* in python 3. They have not been used in `Django`, so that the model trained on `Djnago` is not able to translate their occurrences in the test data containing `NLTK` methods referencing them.

The common errors in the translation were for example , translated output was missing with missing right bracket in print (x) or list(range(x)) frequently, and list (range(x)) was not being added for the range(x) in translation sometimes. Others included with the ot translating iteritem() to item(), and adding list() at undesired places. This can be primarily explained by the construction of translation model using the *n*-grams model. It is limited by 5- grams we have used while training the model, whereas generally the print statements are too long to take the whole of the print statement into context to add the right bracket,and so is for the list() function being omitted.

We can improve the performance by adding a post processing step, combining with the rule based systems.

6 Conclusion

In this paper, we have described a pilot study on modeling programming languages as natural language and using that for building translation models like the natural language. We successfully demonstrate the use of conventional statistical machine translation models to translate code from python 2 to python 3, which requires some post processing.
We also demonstrated the differences in the interproject training and testing performance, conclud-

ing that project style differences do affect the performance, albeit marginally. Hence, through this study we demonstrate that programming languages can be treated as natural languages and SMT models can be applied on them.

7 Future Work

Some future directions can be building bilingual language corpuses like say between a project written in Java as well as C++, which could then be used to translate the code written in say Java to C++. However, this would essentially require a lost of overhead in terms of preprocessing as well as post processing.
One important avenue to explore could be code translation to its English text description or vice versa, which would firstly involve creating reliable bilingual corpus. This could help in creating code summarization systems, which could help developers understand the code written by someone else.

8 Acknowledgements

We would like to thank Dr. Greg Kondrak, for his help with experimental designs.

References

Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the use of automated text summarization techniques for summarizing source code. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 35–44. IEEE.

Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 837–847. IEEE.

Dana Movshovitz-Attias and William W Cohen. 2013. Natural language models for predicting programming comments.

Tung Thanh Nguyen, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2013. A statistical semantic language model for source code. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 532–542. ACM.

Benjamin Peterson. 2014. 2to3 tool for converting python 2 code to python 3 code. `https://docs.python.org/2/library/2to3.html`.