Python Code Completion as “N”LP

Anthony Lu

6.864 Fall 2015

Task: code completion

```
import ast

def example_func(node):
  result = {}
  for field, value in ast.____(node):
    result[field] = value
  return result
```

```
Completions:
walk 0.0
iter_fields
-0.76751687
assess -5.2171769
format -5.2171769
get -5.2171769
load -5.2171769
log -5.2171769
setdefault -5.2171769
...
```

Data

Small dataset: source code of the ESP website <github.com/learning-unlimited/ESP-Website>

Large(r) dataset: top ~800 PyPI packages (>100000 downloads in the past month)

Test cases generated by withholding some training examples and replacing attribute/method names with blanks

AST (abstract syntax tree) representation

```
for thing in things:
    self.process(thing)
```

```
Module(body=[For(target=Name(id='thing', ctx=Store()),
iter=Name(id='things', ctx=Load()), body=[Expr(value=Call
(func=Attribute(value=Name(id='self', ctx=Load()),
attr='process', ctx=Load()), args=[Name(id='thing',
ctx=Load())], keywords=[], starargs=None, kwargs=None))],
orelse=[])])
```

Baseline: AST-based PCFG language model

P(node subtype | node type)

```
_ast.stmt: Counter({
       _ast.Assign: 87,
       _ast.Expr: 49,
       _ast.FunctionDef: 32,
       ...})
```

P(field values | subtype)

```
(_ast.Name, 'id'): Counter({
       'self': 65,
       'node': 55,
       't': 51,
       ...})
```

Each node assumed independent of surrounding and enclosing context.

In the code completion setup, effectively reduces to a naive strategy of always predicting the most common names overall in the training corpus.

Approach 1: better PCFG, ancestor annotation

P(node subtype | type, parents)

P(field values | subtype, parents)

Limited form of context dependence (types of all nodes/fields on the path to the completion site)

Does not take advantage of variable co-occurrence (all variables in surrounding context look the same)

```
for field, value in ast.____(node):
    result[field] = value
```

```
walk 0.0
iter_fields -0.76751687
assess -5.2171769
...
```

```
for field, value in ast.iter_fields(node):
    result[field] = ____(value)
```

```
getattr -2.22139125637
sum -2.22139125637
max -2.35280108894
...
```

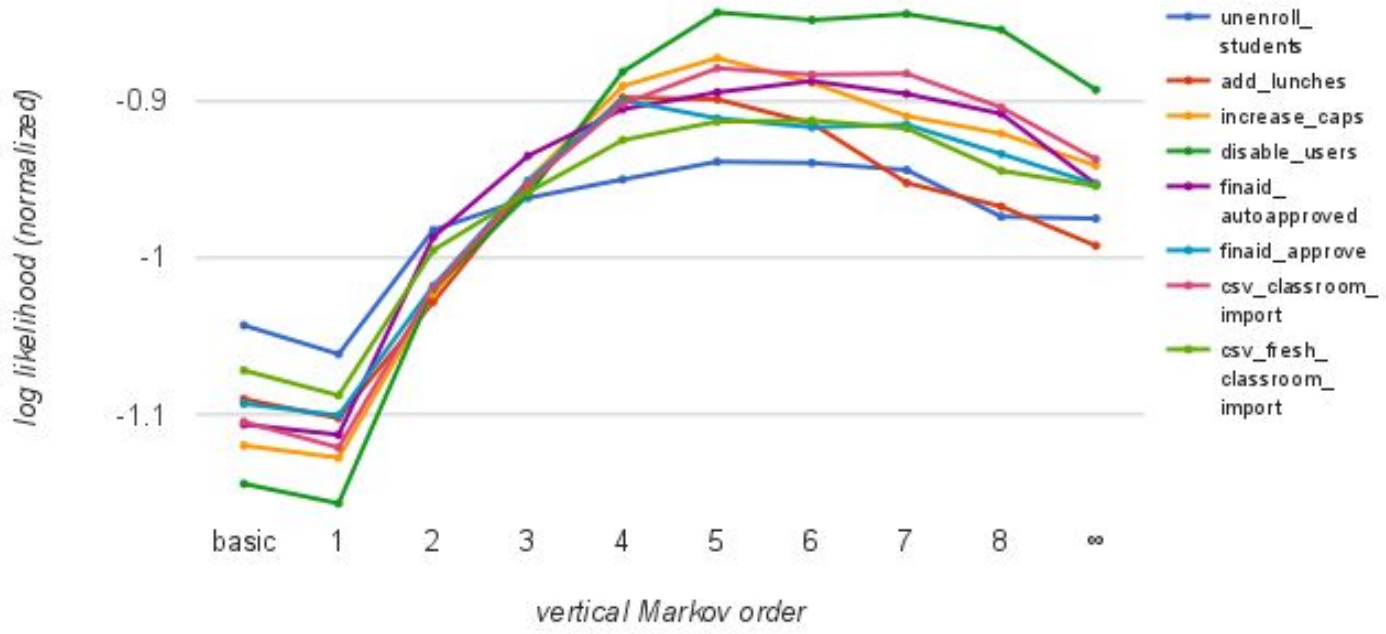
Approach 2: prediction using MaxEnt classifier

$P(\text{field value} \mid \text{contextual features } \Phi) \propto \exp(\Theta \cdot \Phi)$

Features:

- parent node types
- sibling node types
- parent/sibling field values and lengths
- variable co-occurrence: if siblings contain variable names, extract features from other places the variable name appears in the file

Evaluation: language model cross-entropy



Evaluation: predictive accuracy

Still running experiments. Preliminary results on a small test set:

	Recall of top result	top 5	top 10
Baseline	~8%	~15%	~23%
Approach 1 (order 5)	~15%	~38%	~53%
Approach 1 (order ∞)	~8%	~38%	~46%
Approach 2	~50%	~69%	~77%