Homework 3: 词性标注与文本分类

学号：16307130194 姓名：陈中钰

In [1]:
```
%matplotlib inline
```

导入nltk库和nltk.corpus中的brown语料：

In [2]:
```
import nltk
# nltk.download()
# from nltk.book import *
from nltk.corpus import brown
print(brown.categories())
```

```
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned',
'lore', 'mystery', 'news', 'religion', 'reviews', 'romance', 'science_fiction']
```

1. brown语料库处理

从brown语料库中导入使用universal tagset标注的词语数据：

In [3]:
```
tagged_words = brown.tagged_words(tagset='universal')
tagged_words[0]
```

Out[3]:
```
('The', 'DET')
```

数据的数量：

In [4]:
```
len(tagged_words)
```

Out[4]:
```
1161192
```

数据中使用的全部标注tag：

In [5]:
```
set([tag for (word, tag) in tagged_words])
```

Out[5]:
```
{'.',
 'ADJ',
 'ADP',
 'ADV',
 'CONJ',
 'DET',
 'NOUN',
 'NUM',
 'PRON',
 'PRT',
 'VERB',
 'X'}
```

写程序处理布朗语料库，找到以下问题的答案：

1.1 哪些名词常以它们复数形式而不是它们的单数形式出现？（只考虑常规的复数形式，-s后缀形式的）。

首先统计所有NOUN的词频：

In [6]:
```
fdist = nltk.FreqDist([word for (word, tag) in tagged_words if tag == 'NOUN'])
```

然后要判断一个NOUN是否符合常规的复数形式：词本身不以's'结尾，是单数形式，而且加上's'后的复数形式的词也存在。对这些符合要求的词，计算它的复数形式的出现次数占两种形式的出现次数的比例，按照比例从大到小排序，并显示前100个词。

In [7]:

```
words = []
for word in fdist:
    if not word.endswith('s') and word + 's' in fdist:
        words.append((word, fdist[word + 's'] / (fdist[word] + fdist[word + 's'])))
words.sort(key=lambda w:w[1], reverse=True)

words[0:100]
```

```
Out[7]: [('Corp', 0.9882352941176471),
 ('headquarter', 0.9824561403508771),
 ('Catholic', 0.9705882352941176),
 ('stair', 0.9583333333333334),
 ('relative', 0.9545454545454546),
 ('tear', 0.9411764705882353),
 ('mean', 0.94),
 ('employe', 0.9375),
 ('Motor', 0.9347826086956522),
 ('stockholder', 0.9285714285714286),
 ('Persian', 0.9230769230769231),
 ('investor', 0.9166666666666666),
 ('rib', 0.9166666666666666),
 ('microorganism', 0.9166666666666666),
 ('Height', 0.9166666666666666),
 ('Mill', 0.9130434782608695),
 ('kenning', 0.9090909090909091),
 ('Pop', 0.9090909090909091),
 ('Mar', 0.9047619047619048),
 ('assessor', 0.9047619047619048),
 ('ill', 0.9),
 ('subordinate', 0.9),
 ('Relation', 0.9),
 ('gut', 0.9),
 ('elder', 0.9),
 ('Brook', 0.8888888888888888),
 ('Result', 0.8888888888888888),
 ('resource', 0.8888888888888888),
 ('recruit', 0.8888888888888888),
 ('accommodation', 0.8888888888888888),
 ('Scot', 0.8888888888888888),
 ('survivor', 0.8888888888888888),
 ('bun', 0.8888888888888888),
 ('shipment', 0.8823529411764706),
 ('narcotic', 0.875),
 ('teamster', 0.875),
 ('Problem', 0.875),
 ('calorie', 0.875),
 ('cosmetic', 0.875),
 ('libertie', 0.875),
 ('syllable', 0.875),
 ('eyelid', 0.875),
 ('dune', 0.875),
 ('circumstance', 0.8723404255319149),
 ('planner', 0.8666666666666667),
 ('saving', 0.8636363636363636),
 ('volunteer', 0.8620689655172413),
 ('voter', 0.8571428571428571),
 ('Artist', 0.8571428571428571),
 ('teen-ager', 0.8571428571428571),
 ('savage', 0.8571428571428571),
 ('Seed', 0.8571428571428571),
 ('epicycle', 0.8571428571428571),
 ('duct', 0.8571428571428571),
 ('stray', 0.8571428571428571),
 ('Picture', 0.8571428571428571),
 ('troop', 0.8524590163934426),
 ('follower', 0.85),
 ('Trustee', 0.8461538461538461),
 ('Phillip', 0.8461538461538461),
 ('parent', 0.84375),
 ('commercial', 0.8333333333333334),
 ('Dodger', 0.8333333333333334),
 ('megaton', 0.8333333333333334),
 ('Idea', 0.8333333333333334),
 ('Eagle', 0.8333333333333334),
 ('Pilgrim', 0.8333333333333334),
 ('Realtor', 0.8333333333333334),
 ('batten', 0.8333333333333334),
 ('resultant', 0.8333333333333334),
 ('Parson', 0.8333333333333334),
 ('adherent', 0.8333333333333334),
 ('runner', 0.8333333333333334),
 ('concentrate', 0.8333333333333334),
 ('Investor', 0.8333333333333334),
 ('Fellow', 0.8333333333333334),
 ('shunt', 0.8333333333333334),
 ('Direction', 0.8333333333333334),
 ('retailer', 0.8333333333333334),
 ('congratulation', 0.8333333333333334),
 ('tektite', 0.8333333333333334),
 ('sausage', 0.8333333333333334),
 ('romantic', 0.8333333333333334),
 ('shred', 0.8333333333333334),
 ("Lewis'", 0.8333333333333334),
 ('Blue', 0.8333333333333334),
 ('cheekbone', 0.8333333333333334),
 ('invader', 0.8333333333333334),
 ('Han', 0.8301886792452831),
 ('tactic', 0.8260869565217391),
 ('Protestant', 0.8235294117647058),
 ('facet', 0.8181818181818182),
 ('pamphlet', 0.8181818181818182),
```

```
 ('subsystem', 0.8181818181818182),
 ('minute', 0.8177966101694916),
 ('defect', 0.8125),
 ('acre', 0.8076923076923077),
 ('expenditure', 0.8076923076923077),
 ('milligram', 0.8076923076923077),
 ('trader', 0.8064516129032258)]
```

1.2 哪个词的不同词性标记数目最多?

使用set()统计每个词的词性标记，并输出词性标记数目最多的词语：

In [8]:
```python
pos = {}
for (word, tag) in tagged_words:
    if word not in pos:
        pos[word] = set(tag)
    else:
        pos[word].add(tag)
max_count = 0
max_count_word = ''
for word in pos:
    if len(pos[word]) > max_count:
        max_count, max_count_word = len(pos[word]), word
max_count, max_count_word
```

Out[8]:
```
(9, 'damn')
```

词性标记数目最多的词语是'damn'，拥有9种词性标记。

1.3 按频率递减的顺序列出标记。前20个最频繁的词性标记代表什么?

使用FreqDist()统计词性标记的数量，并用most_common()按照词性标记的数量从大到小的顺序输出词性标记：

In [9]:
```python
fdist = nltk.FreqDist([tag for (word, tag) in tagged_words])
fdist.most_common(20)
```

Out[9]:
```
[('NOUN', 275558),
 ('VERB', 182750),
 ('.', 147565),
 ('ADP', 144766),
 ('DET', 137019),
 ('ADJ', 83721),
 ('ADV', 56239),
 ('PRON', 49334),
 ('CONJ', 38151),
 ('PRT', 29829),
 ('NUM', 14874),
 ('X', 1386)]
```

其中，'NOUN'表示名词，'VERB'表示动词，'.'表示标点符号，'ADP'表示adpositions(prepositions and postpositions)，'DET'表示限定词，'ADJ'表示形容词，'ADV'表示副词，'PRON'表示代词，'CONJ'表示连词，'PRT'表示particles or other function words，'NUM'表示数词，'X'表示其他词性。

1.4 名词后面最常见的是哪些词性标记? 这些标记代表什么?

使用bigrams()生成二元词组，过滤出前一个词为名词的二元词组，并用FreqDist()统计这些二元词组的词性标记数量，用most_common()按数量从多到少的顺序输出词性标记：

In [10]:
```python
word_tag_pairs = nltk.bigrams(tagged_words)
noun_post = [post[1] for (pre, post) in word_tag_pairs if pre[1] == 'NOUN']
fdist = nltk.FreqDist(noun_post)
fdist.most_common()
```

Out[10]:
```
[('.', 78326),
 ('ADP', 67460),
 ('VERB', 43819),
 ('NOUN', 41309),
 ('CONJ', 16451),
 ('ADV', 7307),
 ('PRON', 5492),
 ('PRT', 4940),
 ('DET', 4504),
 ('ADJ', 3603),
 ('NUM', 2255),
 ('X', 92)]
```

其中，'.'表示标点符号，'ADP'表示adpositions(prepositions and postpositions)，'VERB'表示动词，'NOUN'表示名词，'CONJ'表示连词，'ADV'表示副词，'PRON'表示代词，'PRT'表示particles or other function words，'DET'表示限定词，'ADJ'表示形容词，'NUM'表示数词，'X'表示其他词性。

2. nltk分类器

使用nltk提供的任意分类器，以及你能想到的特征，建立一个名字性别分类器。从将名字语料库分成3个子集开始：400个词为测试集，400个词为开发集，剩余的个词为训练集。然后从示例的名字性别分类器开始，逐步改善。使用开发集检查你的进展。一旦你对你的分类器感到满意，在测试集上检查它的最终性能。相比在开发测试集上的性能，它在测试集上的性能如何?

从nltk的corpus中导入names数据，给名字标注上性别的标签，获得总体数据，并shuffle数据：

```
In [11]: from nltk.corpus import names
         import random

         labeled_names = ([(name, 'male') for name in names.words('male.txt')] +
                          [(name, 'female') for name in names.words('female.txt')])
         random.seed(1)
         random.shuffle(labeled_names)
         len(labeled_names)
```

```
Out[11]: 7944
```

从总体数据中划分出400个数据作为开发集、400个数据作为测试集，剩下的数据作为训练集：

```
In [12]: dev_amount, test_amount = 400, 400

         dev_names = labeled_names[:dev_amount]
         test_names = labeled_names[dev_amount:dev_amount + test_amount]
         train_names = labeled_names[dev_amount + test_amount:]
         len(train_names), len(dev_names), len(test_names)
```

```
Out[12]: (7144, 400, 400)
```

2.1 以名字最后一个字母为特征

提取最后一个字母为特征：

```
In [13]: def get_features(word):
             return {'last1letter': word[-1]}
         get_features('Alice')
```

```
Out[13]: {'last1letter': 'e'}
```

使用训练集训练一个简单贝叶斯分类器，并用开发集测试精度：

```
In [14]: train_set = [(get_features(name), label) for name, label in train_names]
         dev_set = [(get_features(name), label) for name, label in dev_names]
         classifier = nltk.NaiveBayesClassifier.train(train_set)
         nltk.classify.accuracy(classifier, dev_set)
```

```
Out[14]: 0.775
```

从开发集中筛选出预测错误的数据，并进行分析：

In [15]:
```
errors = []
for name, label in dev_names:
    pred = classifier.classify(get_features(name))
    if pred != label:
        errors.append((name, label, pred))
sorted(errors)
```

```
Out[15]: [('Alexis', 'female', 'male'),
 ('Ambrosi', 'male', 'female'),
 ('Anthony', 'male', 'female'),
 ('Archie', 'male', 'female'),
 ('Ardeen', 'female', 'male'),
 ('Babs', 'female', 'male'),
 ('Barney', 'male', 'female'),
 ('Broddy', 'male', 'female'),
 ('Candis', 'female', 'male'),
 ('Casey', 'male', 'female'),
 ('Cherin', 'female', 'male'),
 ('Chevy', 'male', 'female'),
 ('Conway', 'male', 'female'),
 ('Danny', 'male', 'female'),
 ('Dell', 'male', 'female'),
 ('Dennie', 'male', 'female'),
 ('Devan', 'female', 'male'),
 ('Doloritas', 'female', 'male'),
 ('Doralynn', 'female', 'male'),
 ('Dot', 'female', 'male'),
 ('Drake', 'male', 'female'),
 ('Eddy', 'male', 'female'),
 ('Eleanor', 'female', 'male'),
 ('Fan', 'female', 'male'),
 ('Felicdad', 'female', 'male'),
 ('Gene', 'male', 'female'),
 ('Germain', 'female', 'male'),
 ('Giovanni', 'male', 'female'),
 ('Glynis', 'female', 'male'),
 ('Griffith', 'male', 'female'),
 ('Gwenn', 'female', 'male'),
 ('Harvey', 'male', 'female'),
 ('Helmuth', 'male', 'female'),
 ('Hervey', 'male', 'female'),
 ('Hewie', 'male', 'female'),
 ('Ingeborg', 'female', 'male'),
 ('Jean', 'female', 'male'),
 ('Jerry', 'male', 'female'),
 ('Julie', 'male', 'female'),
 ('Keil', 'male', 'female'),
 ('Keith', 'male', 'female'),
 ('Krishna', 'male', 'female'),
 ('Lay', 'male', 'female'),
 ('Lindy', 'male', 'female'),
 ('Locke', 'male', 'female'),
 ('Lonny', 'male', 'female'),
 ('Margaux', 'female', 'male'),
 ('Marilyn', 'female', 'male'),
 ('Megan', 'female', 'male'),
 ('Mendel', 'male', 'female'),
 ('Michael', 'male', 'female'),
 ('Micky', 'male', 'female'),
 ('Mikael', 'male', 'female'),
 ('Mischa', 'male', 'female'),
 ('Monty', 'male', 'female'),
 ('Morgen', 'female', 'male'),
 ('Mose', 'male', 'female'),
 ('Niall', 'male', 'female'),
 ('Parnell', 'male', 'female'),
 ('Parrnell', 'male', 'female'),
 ('Patrice', 'male', 'female'),
 ('Peg', 'female', 'male'),
 ('Phylys', 'female', 'male'),
 ('Prince', 'male', 'female'),
 ('Raphael', 'male', 'female'),
 ('Robbyn', 'female', 'male'),
 ('Roddie', 'male', 'female'),
 ('Rolfe', 'male', 'female'),
 ('Sandy', 'male', 'female'),
 ('Say', 'male', 'female'),
 ('Scarlett', 'female', 'male'),
 ('Shelby', 'male', 'female'),
 ('Shurlocke', 'male', 'female'),
 ('Sly', 'male', 'female'),
 ('Stacy', 'male', 'female'),
 ('Stanley', 'male', 'female'),
 ('Steve', 'male', 'female'),
 ('Sunny', 'male', 'female'),
 ('Terrill', 'male', 'female'),
 ('Theo', 'female', 'male'),
 ('Thomasin', 'female', 'male'),
 ('Tobie', 'male', 'female'),
 ('Uli', 'male', 'female'),
 ('Veradis', 'female', 'male'),
 ('Vinnie', 'male', 'female'),
 ('Waine', 'male', 'female'),
 ('Wayne', 'male', 'female'),
 ('Westbrooke', 'male', 'female'),
 ('Wylie', 'male', 'female'),
 ('Zolly', 'male', 'female')]
```

仅以最后一个字母来判断性别，精度并不是很高。通过分析错误预测的数据可以发现，以'ly'、'yn'结尾的一般是女性，而以'ie'结尾的一般是男性，因此可以考虑以最后两个字母作为特征。

2.2 以名字最后一个、最后两个字母为特征

提取最后一个、最后两个字母为特征：

In [16]:
```
def get_features(word):
    return {'last1letter': word[-1], 'last2letter': word[-2:]}
get_features('Alice')
```

Out[16]:
```
{'last1letter': 'e', 'last2letter': 'ce'}
```

使用训练集训练一个简单贝叶斯分类器，并用开发集测试精度：

In [17]:
```
train_set = [(get_features(name), label) for name, label in train_names]
dev_set = [(get_features(name), label) for name, label in dev_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, dev_set)
```

Out[17]:
```
0.7925
```

精度提高了将近2%。接着从开发集中筛选出预测错误的数据，并进行分析：

In [18]:
```python
errors = []
for name, label in dev_names:
    pred = classifier.classify(get_features(name))
    if pred != label:
        errors.append((name, label, pred))
sorted(errors)
```

Out[18]:
```
[('Aeriel', 'female', 'male'),
 ('Aggy', 'female', 'male'),
 ('Alexis', 'female', 'male'),
 ('Ambrosi', 'male', 'female'),
 ('Angy', 'female', 'male'),
 ('Annabal', 'female', 'male'),
 ('Annabel', 'female', 'male'),
 ('Anthony', 'male', 'female'),
 ('Archie', 'male', 'female'),
 ('Ardeen', 'female', 'male'),
 ('Babs', 'female', 'male'),
 ('Bamby', 'female', 'male'),
 ('Barbey', 'female', 'male'),
 ('Broddy', 'male', 'female'),
 ('Candis', 'female', 'male'),
 ('Cecil', 'female', 'male'),
 ('Ceciley', 'female', 'male'),
 ('Cherin', 'female', 'male'),
 ('Christel', 'female', 'male'),
 ('Clary', 'female', 'male'),
 ('Courtney', 'female', 'male'),
 ('Danny', 'male', 'female'),
 ('Dennie', 'male', 'female'),
 ('Devan', 'female', 'male'),
 ('Doloritas', 'female', 'male'),
 ('Dorothy', 'female', 'male'),
 ('Dot', 'female', 'male'),
 ('Eddy', 'male', 'female'),
 ('Eleanor', 'female', 'male'),
 ('Fan', 'female', 'male'),
 ('Felicdad', 'female', 'male'),
 ('Gene', 'male', 'female'),
 ('Germain', 'female', 'male'),
 ('Giovanni', 'male', 'female'),
 ('Glynis', 'female', 'male'),
 ('Griffith', 'male', 'female'),
 ('Haleigh', 'female', 'male'),
 ('Helmuth', 'male', 'female'),
 ('Hewie', 'male', 'female'),
 ('Ingeborg', 'female', 'male'),
 ('Jean', 'female', 'male'),
 ('Julie', 'male', 'female'),
 ('Keeley', 'female', 'male'),
 ('Keith', 'male', 'female'),
 ('Krishna', 'male', 'female'),
 ('Lindy', 'male', 'female'),
 ('Lonny', 'male', 'female'),
 ('Lorry', 'female', 'male'),
 ('Madel', 'female', 'male'),
 ('Maisey', 'female', 'male'),
 ('Marabel', 'female', 'male'),
 ('Megan', 'female', 'male'),
 ('Mischa', 'male', 'female'),
 ('Monty', 'male', 'female'),
 ('Morgen', 'female', 'male'),
 ('Mose', 'male', 'female'),
 ('Patrice', 'male', 'female'),
 ('Peg', 'female', 'male'),
 ('Perl', 'female', 'male'),
 ('Prince', 'male', 'female'),
 ('Roddie', 'male', 'female'),
 ('Sandy', 'male', 'female'),
 ('Scarlett', 'female', 'male'),
 ('Shirley', 'female', 'male'),
 ('Sly', 'male', 'female'),
 ('Stacy', 'male', 'female'),
 ('Steve', 'male', 'female'),
 ('Sunny', 'male', 'female'),
 ('Sybil', 'female', 'male'),
 ('Tabby', 'female', 'male'),
 ('Taffy', 'female', 'male'),
 ('Terry', 'female', 'male'),
 ('Theo', 'female', 'male'),
 ('Thomasin', 'female', 'male'),
 ('Tobie', 'male', 'female'),
 ('Uli', 'male', 'female'),
 ('Veradis', 'female', 'male'),
 ('Vicky', 'female', 'male'),
 ('Vinnie', 'male', 'female'),
 ('Waine', 'male', 'female'),
 ('Wayne', 'male', 'female'),
 ('Wylie', 'male', 'female'),
 ('Zolly', 'male', 'female')]
```

通过分析错误预测的数据可以发现，以'Ma'、'Do'、'Ba'开头的都是女性，因此可以考虑以开头两个字母作为特征。

2.3 以名字最后一个、最后两个、前两个字母为特征

提取最后一个、最后两个、前两个字母为特征：

In [19]:
```
def get_features(word):
    return {'last1letter': word[-1], 'last3letter': word[-2:], 'first2letter': word[:2]}
get_features('Alice')
```

Out[19]:
```
{'last1letter': 'e', 'last3letter': 'ce', 'first2letter': 'Al'}
```

使用训练集训练一个简单贝叶斯分类器，并用开发集测试精度：

In [20]:
```
train_set = [(get_features(name), label) for name, label in train_names]
dev_set = [(get_features(name), label) for name, label in dev_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, dev_set)
```

Out[20]:
```
0.825
```

从开发集中筛选出预测错误的数据：

In [21]:
```python
errors = []
for name, label in dev_names:
    pred = classifier.classify(get_features(name))
    if pred != label:
        errors.append((name, label, pred))
sorted(errors)
```

Out[21]:
```
[('Alexis', 'female', 'male'),
 ('Ambrosi', 'male', 'female'),
 ('Anthony', 'male', 'female'),
 ('Archie', 'male', 'female'),
 ('Ardeen', 'female', 'male'),
 ('Babs', 'female', 'male'),
 ('Bamby', 'female', 'male'),
 ('Barbey', 'female', 'male'),
 ('Broddy', 'male', 'female'),
 ('Candis', 'female', 'male'),
 ('Casey', 'male', 'female'),
 ('Cherin', 'female', 'male'),
 ('Christel', 'female', 'male'),
 ('Clary', 'female', 'male'),
 ('Conway', 'male', 'female'),
 ('Danny', 'male', 'female'),
 ('Dennie', 'male', 'female'),
 ('Devan', 'female', 'male'),
 ('Doloritas', 'female', 'male'),
 ('Dot', 'female', 'male'),
 ('Eddy', 'male', 'female'),
 ('Eleanor', 'female', 'male'),
 ('Felicdad', 'female', 'male'),
 ('Gene', 'male', 'female'),
 ('Germain', 'female', 'male'),
 ('Giovanni', 'male', 'female'),
 ('Haleigh', 'female', 'male'),
 ('Havivah', 'female', 'male'),
 ('Hewie', 'male', 'female'),
 ('Ingeborg', 'female', 'male'),
 ('Jean', 'female', 'male'),
 ('Julie', 'male', 'female'),
 ('Keith', 'male', 'female'),
 ('Krishna', 'male', 'female'),
 ('Lay', 'male', 'female'),
 ('Lindy', 'male', 'female'),
 ('Locke', 'male', 'female'),
 ('Lonny', 'male', 'female'),
 ('Loren', 'male', 'female'),
 ('Megan', 'female', 'male'),
 ('Mendel', 'male', 'female'),
 ('Micky', 'male', 'female'),
 ('Mischa', 'male', 'female'),
 ('Monty', 'male', 'female'),
 ('Morgen', 'female', 'male'),
 ('Mose', 'male', 'female'),
 ('Patrice', 'male', 'female'),
 ('Peg', 'female', 'male'),
 ('Perl', 'female', 'male'),
 ('Prince', 'male', 'female'),
 ('Roddie', 'male', 'female'),
 ('Scarlett', 'female', 'male'),
 ('Stacy', 'male', 'female'),
 ('Steve', 'male', 'female'),
 ('Sunny', 'male', 'female'),
 ('Sybil', 'female', 'male'),
 ('Tabby', 'female', 'male'),
 ('Taffy', 'female', 'male'),
 ('Terry', 'female', 'male'),
 ('Theo', 'female', 'male'),
 ('Thomasin', 'female', 'male'),
 ('Tobie', 'male', 'female'),
 ('Trudy', 'female', 'male'),
 ('Uli', 'male', 'female'),
 ('Veradis', 'female', 'male'),
 ('Vinnie', 'male', 'female'),
 ('Waine', 'male', 'female'),
 ('Wayne', 'male', 'female'),
 ('Wylie', 'male', 'female'),
 ('Zolly', 'male', 'female')]
```

精度提高了超过3%，达到了82.5%，效果还是很不错的。在测试集上检查模型的最终性能：

In [22]:
```python
test_set = [(get_features(name), label) for name, label in test_names]
nltk.classify.accuracy(classifier, test_set)
```

Out[22]:
```
0.8175
```

模型在开发测试集上的性能为82.5%，而它在测试集上的性能为81.75%，要稍微低一点。因为使用了训练集来训练模型，并通过在开发集上测试得到的精度来选择一个最好的模型，以防止模型过拟合。而测试集代表的是模型从来没有见过的、更接近现实的数据，所以精度会稍微低一点。

In [ ]: