使用 FastText 进行文本分类 (linux 环境下)

第一步：下载 fasttext-0.9.1

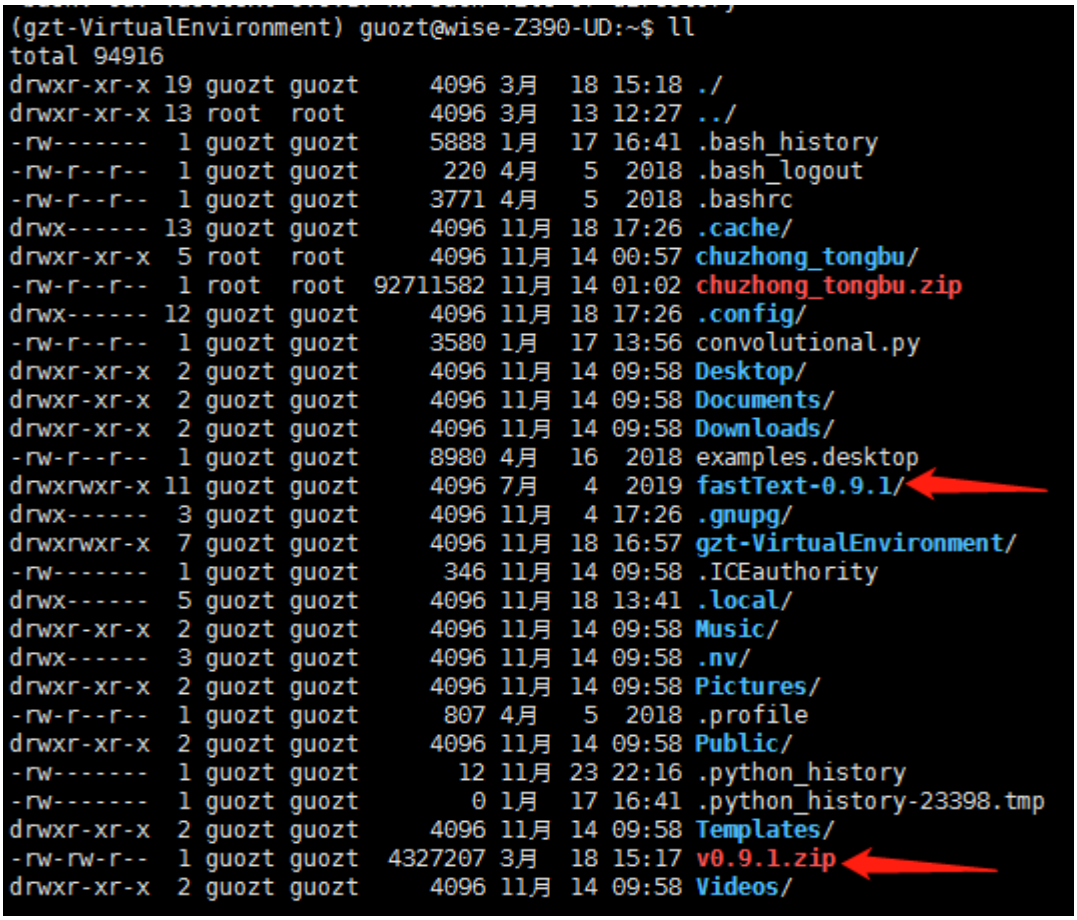
进入服务器后进入自己的虚拟环境，命令是 source gzt-VirtualEnvironment/bin/activate

首先我们需要做的便是安装搭建 fastText，需要系统支持 c++ 11 的 c++编译器，先从 GitHub 上下载 fastText 到本地(版本在更新，可以到 GitHub 上查看最近版本进行下载)：

wget https://github.com/facebookresearch/fastText/archive/v0.9.1.zip

这是解压 unzip v0.9.1.zip

unzip 后使用命令 ll 看一下文件夹，会看到 fastText-0.9.1 文件夹



之后进入这个文件夹 cd fastText-0.9.1，然后 make（执行 make 命令进行编译）。
执行 ./fasttext，便会打印出 fastText 支持的各种不同的命令，如：supervised 进行模型训练，quantize 量化模型以减少内存使用，test 进行模型测试，predict 预测最可能的标签等

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ make
make: Nothing to be done for 'opt'.
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext
usage: fasttext <command> <args>

The commands supported by fasttext are:

  supervised              train a supervised classifier
  quantize                quantize a model to reduce the memory usage
  test                    evaluate a supervised classifier
  test-label              print labels with precision and recall scores
  predict                 predict most likely labels
  predict-prob            predict most likely labels with probabilities
  skipgram                train a skipgram model
  cbow                    train a cbow model
  print-word-vectors      print word vectors given a trained model
  print-sentence-vectors  print sentence vectors given a trained model
  print-ngrams            print ngrams given a trained model and word
  nn                      query for nearest neighbors
  analogies               query for analogies
  dump                    dump arguments,dictionary,input/output vectors

(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

上述的命令包括：
supervised： 训练一个监督分类器
quantize：量化模型以减少内存使用量
test：评估一个监督分类器
predict：预测最有可能的标签
predict-prob：用概率预测最可能的标签
skipgram：训练一个 skipgram 模型
cbow：训练一个 cbow 模型
print-word-vectors：给定一个训练好的模型，打印出所有的单词向量
print-sentence-vectors：给定一个训练好的模型，打印出所有的句子向量
nn：查询最近邻居
analogies：查找所有同类词

第二步：获取数据及数据预处理

正如上面所说，我们需要带有标签的数据去训练我们的监督学习的分类器，本教程中，我们使用 cooking 相关数据构建我们的分类器，因此首先我们下载数据，进行如下命令操作：
>> wget https://dl.fbaipublicfiles.com/fasttext/data/cooking.stackexchange.tar.gz

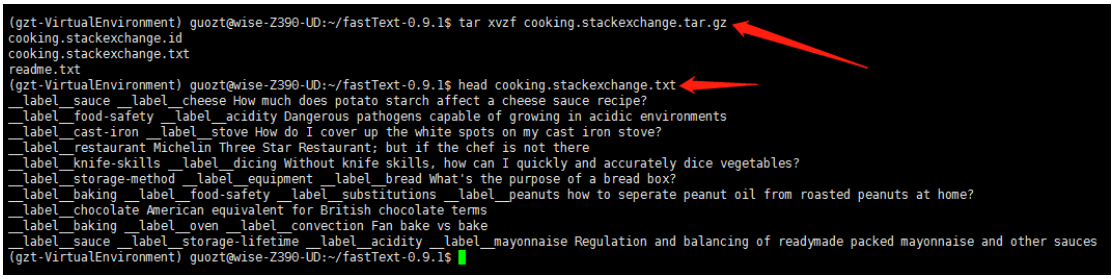
```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ wget https://dl.fbaipublicfiles.com/fasttext/data/cooking.stackexchange.tar.gz
--2020-03-18 16:49:46--  https://dl.fbaipublicfiles.com/fasttext/data/cooking.stackexchange.tar.gz
Resolving dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)... 104.20.22.166, 104.20.6.166, 2606:4700:10::6814:6a6, ...
Connecting to dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)|104.20.22.166|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 457609 (447K) [application/x-tar]
Saving to: 'cooking.stackexchange.tar.gz'

cooking.stackexchange.tar.gz           100%[=========================================================================>] 446.88K   217KB/s    in 2.1s

2020-03-18 16:49:49 (217 KB/s) - 'cooking.stackexchange.tar.gz' saved [457609/457609]

(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

>> tar xvzf cooking.stackexchange.tar.gz
（x 是解压的意思，v 是让你看过程，f 是指定文件，不带 z 或者 j 就是用 tar 格式压缩，z

是 gzip 格式，j 是 bzip 格式）

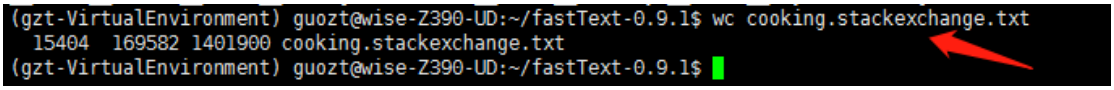
>> head cooking.stackexchange.txt

通过 head 命令便可看到文档形式，文档的每一行都包含一个标签，标签后面跟着相应的单词短语，所有的标签都以__label__前缀开始，这是 fastText 便是标签和单词短语的方式，训练的模型便是预测文档中给定单词短语预测其对应的标签



在训练分类器之前，我们需要将数据分割成训练集和验证集，我们将使用验证集来评估学习到的分类器对新数据的性能好坏，先通过下面命令来查看文档中总共含有多少数据：

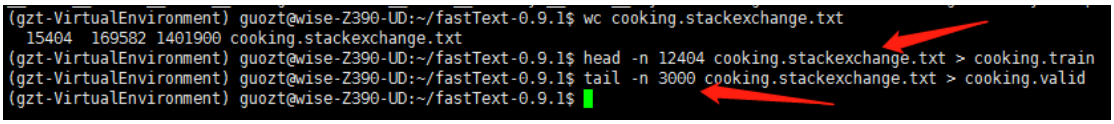
>> wc cooking.stackexchange.txt



可以看到我们数据中总共包含了 15404 个示例，我们把文档分成一个包含 12404 个示例的训练集和一个包含 3000 个示例的验证集，执行如下命令：

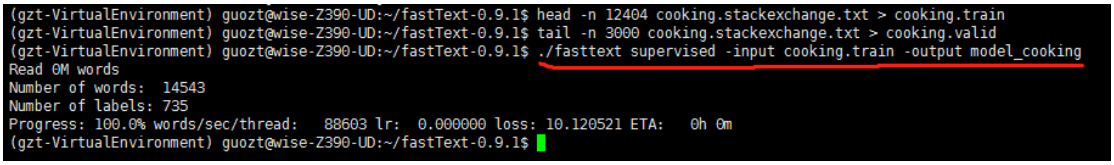
>> head -n 12404 cooking.stackexchange.txt > cooking.train

>> tail -n 3000 cooking.stackexchange.txt > cooking.valid



第三步：使用 fastText 快速搭建分类器

上面数据已经准备好了，接下来我们便开始训练我们的模型，首先执行如下命令进行模型的训练



-input 命令选项指示训练数据，-output 选项指示的是保存的模型的位置，在训练结束后，文件 model_cooking.bin 是在当前目录中创建的，model_cooking.bin 便是我们保存训练模型的文件。

模型训练好之后，我们可以交互式测试我们的分类器，即单独测试某一个句子所属的类别，可以通过以下命令进行交互式测试：

>> ./fasttext predict model_cooking.bin -

输入以上命令后，命令行会提示你输入句子，然后我们可以进行如下句子测试：

Which baking dish is best to bake a banana bread ?

上面句子可以得到预测的标签是 baking，显然这个预测结果是正确的，我们再进行尝试

Why not put knives in the dishwasher?

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext predict model_cooking.bin -
Which baking dish is best to bake a banana bread ?
__label__baking
Why not put knives in the dishwasher?
__label__baking
```

上面句子预测有的正确有的错误（可以多试试几个句子）。

在完全退出这个程序，甚至是退出虚拟环境和 xshell 后，打开，重新进入虚拟环境后进入 fastText-0.9.1 后可以看到之前下载的压缩文件，解压后的文件，train 文件和验证集文件，也有训练完成后的模型文件 model_cooking.bin 文件。

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~$ cd fastText-0.9.1
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ll
total 25360
drwxrwxr-x 11 guozt guozt     4096 3月  18 17:16 ./
drwxr-xr-x 19 guozt guozt     4096 3月  18 17:32 ../
drwxrwxr-x  2 guozt guozt     4096 7月   4  2019 alignment/
-rw-rw-r--  1 guozt guozt    44360 3月  18 15:21 args.o
drwxrwxr-x  2 guozt guozt     4096 7月   4  2019 .circleci/
-rwxr-xr-x  1 guozt guozt     1428 7月   4  2019 classification-example.sh*
-rwxr-xr-x  1 guozt guozt     3154 7月   4  2019 classification-results.sh*
-rw-rw-r--  1 guozt guozt     1933 7月   4  2019 CMakeLists.txt
-rw-rw-r--  1 guozt guozt      241 7月   4  2019 CODE_OF_CONDUCT.md
-rw-rw-r--  1 guozt guozt     2061 7月   4  2019 CONTRIBUTING.md
-rw-r--r--  1 guozt guozt    90095 4月  29  2017 cooking.stackexchange.id
-rw-rw-r--  1 guozt guozt   457609 1月  18  2019 cooking.stackexchange.tar.gz
-rw-r--r--  1 guozt guozt  1401900 4月  29  2017 cooking.stackexchange.txt
-rw-rw-r--  1 guozt guozt  1129498 3月  18 17:11 cooking.train
-rw-rw-r--  1 guozt guozt   272402 3月  18 17:12 cooking.valid
drwxrwxr-x  2 guozt guozt     4096 7月   4  2019 crawl/
-rw-rw-r--  1 guozt guozt    24240 3月  18 15:21 densematrix.o
-rw-rw-r--  1 guozt guozt    96216 3月  18 15:21 dictionary.o
drwxrwxr-x  2 guozt guozt     4096 7月   4  2019 docs/
-rw-rw-r--  1 guozt guozt     2102 7月   4  2019 eval.py
-rwxrwxr-x  1 guozt guozt   353048 3月  18 15:21 fasttext*
-rw-rw-r--  1 guozt guozt   202448 3月  18 15:21 fasttext.o
-rwxr-xr-x  1 guozt guozt     3310 7月   4  2019 get-wikimedia.sh*
-rw-rw-r--  1 guozt guozt       44 7月   4  2019 .gitignore
-rw-rw-r--  1 guozt guozt     1080 7月   4  2019 LICENSE
-rw-rw-r--  1 guozt guozt    62360 3月  18 15:21 loss.o
-rw-rw-r--  1 guozt guozt     1821 7月   4  2019 Makefile
-rw-rw-r--  1 guozt guozt       95 7月   4  2019 MANIFEST.in
-rw-rw-r--  1 guozt guozt     3136 3月  18 15:21 matrix.o
-rw-rw-r--  1 guozt guozt    12408 3月  18 15:21 meter.o
-rw-rw-r--  1 guozt guozt  6382062 3月  18 17:16 model_cooking.bin
```

为了验证学习到的分类模型的好坏，我们在验证集上对它进行测试，观察模型的精准率 precision 和召回率 recall：

命令是 ./fasttext test model_cooking.bin cooking.valid

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.153
R@1     0.0662
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

精准率 Precision 指的是预测为正样本中有多少是真正的正样本，

召回率 Recall 指的是样本中的正样本有多少被预测正确了，因此精准率看的是预测为某一类的样本中有多少是真正的属于这一类的，而召回率看的是在分类任务中某一类样本是否完全被预测正确 **至此，使用 fasttext 进行文本分类已经完成了，但是效果很差！**

模型优化：

上面通过使用默认参数运行 fastText 训练得到的模型在分类新问题上效果很差，接下来我们通过更改默认参数来提高性能。

（1）对数据进行预处理：

使用 cat cooking.stackexchange.txt 可以查看数据，我们发现有些单词包含大写字母和标点符号，因此改善模型性能的第一步就是应用一些简单的预处理，预处理可以使用命令行工具例如 sed、tr 来对文本进行简单的标准化操作，执行命令如下：

>> cat cooking.stackexchange.txt | sed -e "s/\([.\!?,'/()]\)/ \1 /g" | tr "[:upper:]" "[:lower:]" > cooking.preprocessed.txt

>> head -n 12404 cooking.preprocessed.txt > cooking.train

>> tail -n 3000 cooking.preprocessed.txt > cooking.valid

经过上述的处理后，再次分成训练集和验证集。

接下来我们在预处理的数据集上进行模型训练并进行测试，命令如下：

./fasttext supervised -input cooking.train -output model_cooking（这是模型训练命令）

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ cat cooking.stackexchange.txt | sed -e "s/\([.\!?,'/()]\)/ \1 /g" | tr "[:upper:]" "[:lower:]" > coo
king.preprocessed.txt
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ head -n 12404 cooking.preprocessed.txt > cooking.train
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ tail -n 3000 cooking.preprocessed.txt > cooking.valid
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext supervised -input cooking.train -output model_cooking
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   96494 lr:  0.000000 loss: 10.147063 ETA:   0h 0m
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

进行模型训练

由于对数据预处理，词典变小了，由原来的约 14K 个单词变成了约 9K

下面用验证集进行测试

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.169
R@1     0.0731
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

由此可见，进过对数据预处理后，精确率和召回率都有所提高。

（2）增加每个样例的使用次数和增大学习率

增加每个样例的使用次数

在默认情况下，fastText 在训练期间对每个训练用例仅重复使用五次，这太小，因为我们的训练集只有 12k 训练样例，因此我们可以通过-epoch 选项增加每个样例的使用次数，命令如下：./fasttext supervised -input cooking.train -output model_cooking -epoch 25（这里使每个样例的使用次数由默认的 5 次增加为 25 次）

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext supervised -input cooking.train -output model_cooking -epoch 25
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   99412 lr:  0.000000 loss:  7.191073 ETA:   0h 0m
```

进行测试：

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.519
R@1     0.224
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

从上面测试效果可以看出，精准率和召回率都有了大幅度提升，可见增加每个样例的使用次数对于数据集少的情况下效果提升明显。

增大学习率：

另一个增强算法能力的方法是改变模型的学习速度即学习率，这对应于处理每个示例后模型的更改程度，当学习率为 0 时意味着模型根本不会发生改变，因此不会学到任何东西，良好的学习率值在 0.1~1.0 的范围内，下面我们通过设置算法学习率为 learning rate = 1.0 进行模

型训练：./fasttext supervised -input cooking.train -output model_cooking -lr 1.0
之后使用./fasttext test model_cooking.bin cooking.valid 进行验证：

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext supervised -input cooking.train -output model_cooking -lr 1.0
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   96458 lr:  0.000000 loss:  6.736038 ETA:   0h 0m
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.573
R@1     0.248
```

由此可见，精确率和召回率都所有提高。
把两者结合起来：
./fasttext supervised -input cooking.train -output model_cooking -lr 1.0 -epoch 25

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext supervised -input cooking.train -output model_cooking -lr 1.0 -epoch 25
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   84499 lr:  0.000000 loss:  4.643222 ETA:   0h 0m
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.587
R@1     0.254
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

两者结合后，精确率和召回率又有所提高。哈哈哈哈~~~
(3) 使用 word n-grams
此方案中，我们使用单词 bigrams 而不是仅仅是 unigrams 来提高模型的性能，这对于词序很重要的分类问题尤其重要，例如情感分析。n-gram 是基于语言模型的算法，基本思想是将文本内容按照字节顺序进行大小为 N 的窗口滑动操作，最终形成窗口为 N 的字节片段序列。
训练模型命令如下：参数是 -wordNgrams 具体数字

./fasttext supervised -input cooking.train -output model_cooking -lr 1.0

-epoch 25 -wordNgrams 2

```
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext supervised -input cooking.train -output model_cooking -lr 1.0 -epoch 25 -wordNgrams 2
Read 0M words
Number of words:  8952
Number of labels: 735
Progress: 100.0% words/sec/thread:   99402 lr:  0.000000 loss:  3.225697 ETA:   0h 0m
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$ ./fasttext test model_cooking.bin cooking.valid
N       3000
P@1     0.609
R@1     0.263
(gzt-VirtualEnvironment) guozt@wise-Z390-UD:~/fastText-0.9.1$
```

通过几个步骤，可以看出我们将模型精准率从 12.4%提升到了 60.9%，总结一下主要包含以下步骤：
(1) 数据预处理
(2) 增加样本训练次数 epochs(使用参数 –epoch，标准范围[5, 50])
(3) 更改学习率 learning rate(使用参数 –lr，标准范围[0.1-1.0])
(4) 使用 word Ngrams(使用参数 –wordNgrams，标准范围[1-5])

什么是 Bigrams？
unigram 指的是单个不可分割的单元和标记，通常用做模型的输入，并且在不同的模型中 unigram 可以是单词或是字母，在 fastText 中，我们是在单词级别上进行训练模型，因此 unigram 是单词。类似的，bigram 值的是两个连续的单词的串联，n-grams 指的便是 n 个单词的串联。举个例子，现在有这样一句话：Last donut of the night，如果是 unigrams 则是 last,donut,of,the,night，而对于 bigrams 指的是 last donut,donut of,of the,the night。
提升训练速度
目前我们在几千个示例中训练我们的模型，训练只需要几秒钟，但如果数据集增大，标签增多，这时模型训练便会变慢，一个让训练变快的方案便是使用分层 softmax，而不是使用常

规 softmax，使用分层 softmax 是使用参数 －loss hs 实现，命令如下：

>> ./fasttext supervised -input cooking.train -output model_cooking -lr 1.0 -epoch 25 -wordNgrams 2 -bucket 200000 -dim 50 -loss hs

此时对于我们当前的数据集，训练速度应该不超过 1 秒

总结

本教程中我们简单介绍了如何使用 fastText 来训练强大的分类器，同时介绍了一些重要的参数选项，通过调参来进行模型优化。

三、使用 fastText 工具来构建词向量

现在机器学习中一个十分流行的做法便是用向量表示单词，即词向量化 wordEmbedding，这些向量可以捕捉到有关语言的一些隐藏信息，例如语法信息，语义信息等，好的词向量表示可以提升分类器的性能，在本教程中，我们展示如何使用 fastText 工具来构建词向量，安装 fastText 过程请参考上一讲

1、获取数据

为了计算词向量，我们需要一个大的文本语料库，根据语料库的不同，单词向量也将捕捉到不同的信息，在本教程中，我们关注 Wikipedia 的文章，当然也可以考虑其他语料库来源，例如新闻活着 Webcrawl，下载 Wikipedia 语料库执行如下命令：

wget https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

下载 Wikipedia 语料库需要挺长时间，如果不使用 Wikipedia 全部语料库信息，我们可以在 Wikipedia 前 10 亿字节信息进行词向量学习，此数据可以在 Matt Mahoney 网站上下载。

$ mkdir data

$ wget -c http://mattmahoney.net/dc/enwik9.zip -P data

$ unzip data/enwik9.zip -d data

这样我们便获得了 Wikipedia 的部分数据，因为 Wikipedia 语料库中包含大量的 HTML/XML 数据，因此需要对数据进行预处理，我们可以使用与 fastText 自带的 wikifil.pl 脚本对其进行预处理，这个脚本最初由 Matt Mahoney 创建，因此可以在下面网址上找到：http://mattmahoney.net/，执行如下命令对数据进行预处理：

$ perl wikifil.pl data/enwik9 > data/fil9

我们可以执行如下命令检查我们的文件数据：

$ head -c 80 data/fil9

anarchism originated as a term of abuse first used against early working class

可以观察到我们的文本经过了很好的处理，接下来可以用文本来学习词向量

2、训练词向量

数据集已经取到了，现在我们可以使用如下的简单命令在上述数据集上训练我们的词向量

$ mkdir result

$ ./fasttext skipgram -input data/fil9 -output result/fil9

分解上述命令：./fasttext 使用 skipgram 模型调用二进制 fastText 可执行文件，当然也可以使

用 cbow 模型，-input 表示输入数据路径，-output 表示训练的词向量模型所在路径，当 fastText 运行时，屏幕会显示进度和估计的完成时间，程序完成后，结果目录应该出现如下两个文件，可通过下面命令查看：

```
$ ls -l result
-rw-r-r-- 1 bojanowski 1876110778 978480850 Dec 20 11:01 fil9.bin
-rw-r-r-- 1 bojanowski 1876110778 190004182 Dec 20 11:01 fil9.vec
```

fil9.bin 文件是一个二进制文件，它存储了整个 fastText 模型，随后可以进行加载，fil9.vec 文件是一个包含单词向量的文本文件，每一行对应词汇表中的每个单词，可通过如下命令查看 fil9.vec 中的信息

```
$ head -n 4 result/fil9.vec
218316 100
the -0.10363 -0.063669 0.032436 -0.040798 0.53749 0.00097867 0.10083 0.24829 ...
of -0.0083724 0.0059414 -0.046618 -0.072735 0.83007 0.038895 -0.13634 0.60063 ...
one 0.32731 0.044409 -0.46484 0.14716 0.7431 0.24684 -0.11301 0.51721 0.73262 ...
```

从上面结果可见，第一行显示的是单词向量和向量维度，接下来几行是词汇表中所有单词的单词向量，顺序是按照频率降低的顺序进行排序

3、skipgram VS cbow

fastText 为计算单词表示提供了两种模型：skipgram 和 cbow，这和 word2vec 一样，cbow 全称：Continuous-bag-of-words，skipgram 模型运行机理是通过附近的词来预测目标单词，而 cbow 模型则是根据目标词的上下文来预测目标词，这里的上下文指的便是目标词周围的固定大小窗口中包含的单词包，下面通过例子便能够体会到上下文的含义。例如：给出这样一个句子：

Poets have been mysteriously silient on the subject of cheese

其目标词是 slient，skipgram 模型是通过目标词附近的词去预测 slient，例如 subjector, mysteriously, 而 cbow 模型则是通过目标词的上下文词来预测 slient，如：{been, mysteriously, on, the},并且使用单词的向量预测目标，下面一个示例图展示了二者的差异，使用的句子是

I am selling these fine leather jackets

上面已经使用 skipgram 模型对数据集进行了训练，如果想用 cbow 模型训练之行如下命令

./fasttext cbow -input data/fil9 -output result/fil9

从实际效果中看，我们会发现 skipgram 模型对于单词信息的处理效果要优于 cbow 模型

4、模型参数调优

上面的训练都是使用的默认的参数运行 fastText，但根据数据的不同，这些参数可能不是最优的，让我们介绍一下子向量的一些关键参数。

模型中最重要的两个参数是：词向量大小维度、subwords 范围的大小，

词向量维度越大，便能获得更多的信息但同时也需要更多的训练数据，同时如果它们过大，模型也就更难训练速度更慢，默认情况下使用的是 100 维的向量，但在 100-300 维都是常用到的调参范围。

subwords 是一个单词序列中包含最小(minn)到最大(maxn)之间的所有字符串(也即是

n-grams)，默认情况下我们接受 3-6 个字符串中间的所有子单词，但不同的语言可能有不同的合适范围

$ ./fasttext skipgram -input data/fil9 -output result/fil9 -minn 2 -maxn 5 -dim 300

下面介绍另外两个参数：epoch、learning rate、epoch 根据训练数据量的不同，可以进行更改，epoch 参数即是控制训练时在数据集上循环的次数，默认情况下在数据集上循环 5 次，但当数据集非常大时，我们也可以适当减少训练的次数，另一个参数学习率，学习率越高模型收敛的速度就越快，但存在对数据集过度拟合的风险，默认值时 0.05，这是一个很好的折中，当然在训练过程中，也可以对其进行调参，可调范围是[0.01, 1]，下面命令便尝试对这两个参数进行调整：

$ ./fasttext skipgram -input data/fil9 -output result/fil9 -epoch 1 -lr 0.5

最后 fastText 是多线程的，默认情况下使用 12 个线程，如果你的机器只有更少的 CPU 核数，也可以通过如下参数对使用的 CPU 核数进行调整

$ ./fasttext skipgram -input data/fil9 -output result/fil9 -thread 4

5、打印词向量

直接从 fil9.vec 文件中搜索和打印词向量是十分麻烦的，但幸运的是 fastText 提供了打印词向量的功能，我们可以通过 fastText 中 print-word-vectors 功能打印词向量，例如，我们可以使用以下命令打印单词 asparagus、pidgey 和 yellow 单词的词向量：

$ echo "asparagus pidgey yellow" | ./fasttext print-word-vectors result/fil9.bin

asparagus 0.46826 -0.20187 -0.29122 -0.17918 0.31289 -0.31679 0.17828 -0.04418 ...

pidgey -0.16065 -0.45867 0.10565 0.036952 -0.11482 0.030053 0.12115 0.39725 ...

• 0.040719 -0.30155 ...

一个很好的功能是我们可以查询到未出现在数据中的单词，实际上，单词是由字符串的总和组成，只要未知的单词是由已知的字串构成，就可以得到单词的词向量，举个例子下面尝试一下查询拼写出错的单词：

$ echo "enviroment" | ./fasttext print-word-vectors result/fil9.bin

结果仍然可以查询到词向量，但是至于效果怎么样，我们可以在下一节找到答案

6、临近词向量查询

检查单词向量质量的一种简单的方法是查看此单词的临近词，可以通过临近词比较来查看词向量对于语义的表达。最临近词向量查询可以通过 fastText 提供的 nn 功能来实现，例如我们可以通过运行一下命令来查询单词 10 个最近邻居：

$ ./fasttext **nn** result/fil9.bin

Pre-computing word vectors... done.

然后命令行便会提示我们输入需要查询的词，我们尝试一下 asparagus

Query word? asparagus
beetroot 0.812384
tomato 0.806688
horseradish 0.805928
spinach 0.801483
licorice 0.791697
lingonberries 0.781507
asparagales 0.780756
lingonberry 0.778534
celery 0.774529
beets 0.773984
从上面结果可以看出效果不错，查询词之间有很大的共性，再尝试查询 pidgey,结果如下

Query word? pidgey
pidgeot 0.891801
pidgeotto 0.885109
pidge 0.884739
pidgeon 0.787351
pok 0.781068
pikachu 0.758688
charizard 0.749403
squirtle 0.742582
beedrill 0.741579
charmeleon 0.733625
上面提到了如果单词拼写出错可能影响词向量的查询，那如果单词拼写错误，如果查询其临近词结果如何，下面展示一下效果：

Query word? enviroment
enviromental 0.907951
environ 0.87146
enviro 0.855381
environs 0.803349
environnement 0.772682
enviromission 0.761168
realclimate 0.716746
environment 0.702706
acclimatation 0.697196
ecotourism 0.697081
可以看出虽然单词拼写出错，但是查询结果还是捕获到了单词的主要信息，拼写出错的单词也与合理的单词匹配，虽然还是有一些影响，但整体方向是正确的。

为了找到词向量临近的单词，我们需要计算的单词之间的相似度得分。模型训练的单词是由连续的单词向量表示，因此我们可以对其进行相似度的比较，一般情况下，我们使用余弦相

似度去衡量两个单词之间的相似度，我们可以计算词汇表中任意单词和所有其他单词之间的相似度，并显示 10 个最相似单词，当然被查询单词本身肯定排在顶部，相似度为 1

7、单词类比

在相似度问题中，有时会进行单词类比，例如我们训练的模型能够知道法国是什么，并且知道柏林对于德国来说意味着什么。这个在 fastText 中是可以做到的，利用单词类比这个功能即可实现，例如下面我们输入三个单词，然后输出单词的类比单词：

```
$ ./fasttext analogies result/fil9.bin
Pre-computing word vectors... done.
Query triplet (A - B + C)? berlin germany france
paris 0.896462
bourges 0.768954
louveciennes 0.765569
toulouse 0.761916
valenciennes 0.760251
montpellier 0.752747
strasbourg 0.744487
meudon 0.74143
bordeaux 0.740635
pigneaux 0.736122
```

上面模型类比功能提供的最可能结果是巴黎，显然是十分准确，下面我们再来看一个不太明显的例子：

```
Query triplet (A - B + C)? psx sony nintendo
gamecube 0.803352
nintendogs 0.792646
playstation 0.77344
sega 0.772165
gameboy 0.767959
arcade 0.754774
playstationjapan 0.753473
gba 0.752909
dreamcast 0.74907
famicom 0.745298
```

从上面结果可以看出模型认为 psx 是索尼的游戏手柄，因此 nintendo 任天堂类比的是 gamecube，这个类比也比较合理。当然类比的质量也取决于训练模型的数据集，类比的结果也仅仅在数据集的范围内

8、 字符 n-grams 重要性

利用 subword-level 信息也即是 n-grams 对于构建未知单词词向量很有趣，例如 Wikipedia 中不存在 gearshift 这个单词，但是我们仍然能够查询到它的临近单词：

```
Query word? gearshift
```

gearing 0.790762
flywheels 0.779804
flywheel 0.777859
gears 0.776133
driveshafts 0.756345
driveshaft 0.755679
daisywheel 0.749998
wheelsets 0.748578
epicycles 0.744268
gearboxes 0.73986
效果还可以，因为大多数被检索到的单词共享大量的子串，当然也有些特殊的单词比较特殊，例如 cogwheel，我们可以看到 subword-level 对于未知单词查询所起到的效果，但是如果我们在训练模型的时候没有使用 subwords 这个参数，结果会如何，下面我们便进行尝试，运行以下命令训练没有 subwords 的模型：

$ ./fasttext skipgram -input data/fil9 -output result/fil9-none -maxn 0

此时训练的模型保存在 result/fil9-non.vec 和 result/fil9-non.bin，为了表明不加 subwords 模型的不同，我们再举一个 wikipedia 中不常见的单词如：accomodation，就类似于 accommodation 住宿这个单词，下面给出其相似词的查询结果：

$ ./fasttext nn result/fil9-none.bin
Query word? accomodation
sunnhordland 0.775057
accomodations 0.769206
administrational 0.753011
laponian 0.752274
ammenities 0.750805
dachas 0.75026
vuosaari 0.74172
hostelling 0.739995
greenbelts 0.733975
asserbo 0.732465
可以看出结果中的词没有任何意义，大多数词都是不想关的，我们再用使用了 subwords 的模型测试 accomodation 的相似词，结果便有明显的差别：

Query word? accomodation
accomodations 0.96342
accommodation 0.942124
accommodations 0.915427
accommodative 0.847751
accommodating 0.794353
accomodated 0.740381
amenities 0.729746

catering 0.725975
accomodate 0.703177
hospitality 0.701426
上面结果准确捕捉到相似度很高的 accommodation 这个单词，同时我们还捕获到语义相关的词如：便利设施 amenities 和寄宿 lodging，因此训练模型加上 subwords 参数对模型效果有很大的提升

9、 结论
在小节中，详细展示了如果在 wikipedia 上获得词向量，对于其他语言也都可以同样运行，下面网址提供了 fastText 在词向量上的多个预训练模型，可以参考使用预训练模型网址

四、常用命令备忘录

词向量的学习-使用 fastText 学习词向量执行以下命令：

$ ./fasttext skipgram -input data.txt -output model

取得词向量-将模型学习的词向量打印到文件中执行如下命令：

$ ./fasttext print-word-vectors model.bin < queries.txt

文本分类-训练一个文本分类模型执行如下命令：

$ ./fasttext supervised -input train.txt -output model

当一个模型训练结束后，我们可以通过在测试集上计算精准率 Precision 和召回率 Recall 进行模型评估，执行如下命令：

$ ./fasttext test model.bin test.txt 1

为了直接预测一段文本最可能的 k 个标签，执行如下命令：

$ ./fasttext predict model.bin test.txt k

为了直接预测一段文本的 k 个最可能的标签及其相关概率大小，可以执行如下命令：

$ ./fasttext predict-prob model.bin test.txt k

如果想要计算句子或段落的向量表示，执行如下命令：

$ ./fasttext print-sentence-vectors model.bin < text.txt

为了创建一个内存更小的模型可以执行如下命令

$ ./fasttext quantize -output model

所有其他的命令都类似下面 test 命令

$ ./fasttext test model.ftz test.txt

五、模型可选参数列表及默认值

$ ./fasttext supervised
Empty input or output path.

The following arguments are mandatory:
  -input   training file path

-output output file path

The following arguments are optional:
-verbose verbosity level [2]

The following arguments for the dictionary are optional:
-minCount minimal number of word occurrences [5]
-minCountLabel minimal number of label occurrences [0]
-wordNgrams max length of word ngram [1]
-bucket number of buckets [2000000]
-minn min length of char ngram [3]
-maxn max length of char ngram [6]
-t sampling threshold [0.0001]
-label labels prefix [__label__]

The following arguments for training are optional:
-lr learning rate [0.05]
-lrUpdateRate change the rate of updates for the learning rate [100]
-dim size of word vectors [100]
-ws size of the context window [5]
-epoch number of epochs [5]
-neg number of negatives sampled [5]
-loss loss function {ns, hs, softmax} [ns]
-thread number of threads [12]
-pretrainedVectors pretrained word vectors for supervised learning []
-saveOutput whether output params should be saved [0]

The following arguments for quantization are optional:
-cutoff number of words and ngrams to retain [0]
-retrain finetune embeddings if a cutoff is applied [0]
-qnorm quantizing the norm separately [0]
-qout quantizing the classifier [0]
-dsub size of each sub-vector [2]