机器学习（进阶）纳米学位

文档归类

吕奇峰

项目报告

项目背景

自然语言处理（NLP）[1]是人工智能极为重要的一部分。

是计算机科学，人工智能，语言学关注计算机和人类（自然）语言之间的相互作用的领域。因此，自然语言处理是与人机交互的领域有关的。在自然语言处理面临很多挑战，包括自然语言理解，因此，自然语言处理涉及人机交互的智能。在 NLP 诸多挑战涉及自然语言理解，即计算机源于人为或自然语言输入的意思，和其他涉及到自然语言生成。

机器学习下的自然语言处理不同于一般的语言处理算法。在数据量不足，计算机算力也不大的时候，人们普遍大规模的使用规则模型去理解语意。将注意力主要集中在规则上，不同的规则将会输出不同的结果。而机器学习则改变着这种形态。机器学习通过海量的数据，淡化人为的规则，将更多的注意力放在了数据本身。通过巨大的算力，进行巨量的运算而自然生成自然语言的模型。这种做法为自然语言处理带来了跨越性的进展，突破了传统的瓶颈。

一些最早使用的算法，如决策树，产生硬的 if-then 规则类似于手写的规则，是再普通的系统体系。然而，越来越多的研究集中于统计模型，这使得基于附加实数值的权重，每个输入要素柔软，概率的决策。此类模型具有能够表达许多不同的可能的答案，而不是只有一个相对的确定性，产生更可靠的结果时，这种模型被包括作为较大系统的一个组成部分的优点。

自然语言处理研究逐渐从词汇语义进一步的到叙事的理解。然而人类水平的自然语言处理，是一个人工智能的极限问题。它是相当让人工智能和人一样聪明。这是自然语言处理的未来，因此密切结合人工智能发展。

问题描述

自然语言处理有个非常复杂的问题解决过程。首先是一个难点是分词，只是分词现在已经有非常多的库可以使用，也已经有相对比较成熟的算法，得到比较好的成功率。分词对于不同语言的含

义是不一样的，比如中文，正向思维是把一个句子分成短语，短语分成词。而逆向思维则是字组成词，词组成短语，短语组成句子。而英文则不太一样，因为的最小单位是字母，字母组成词，词组成习惯短语，然后再组成句子。

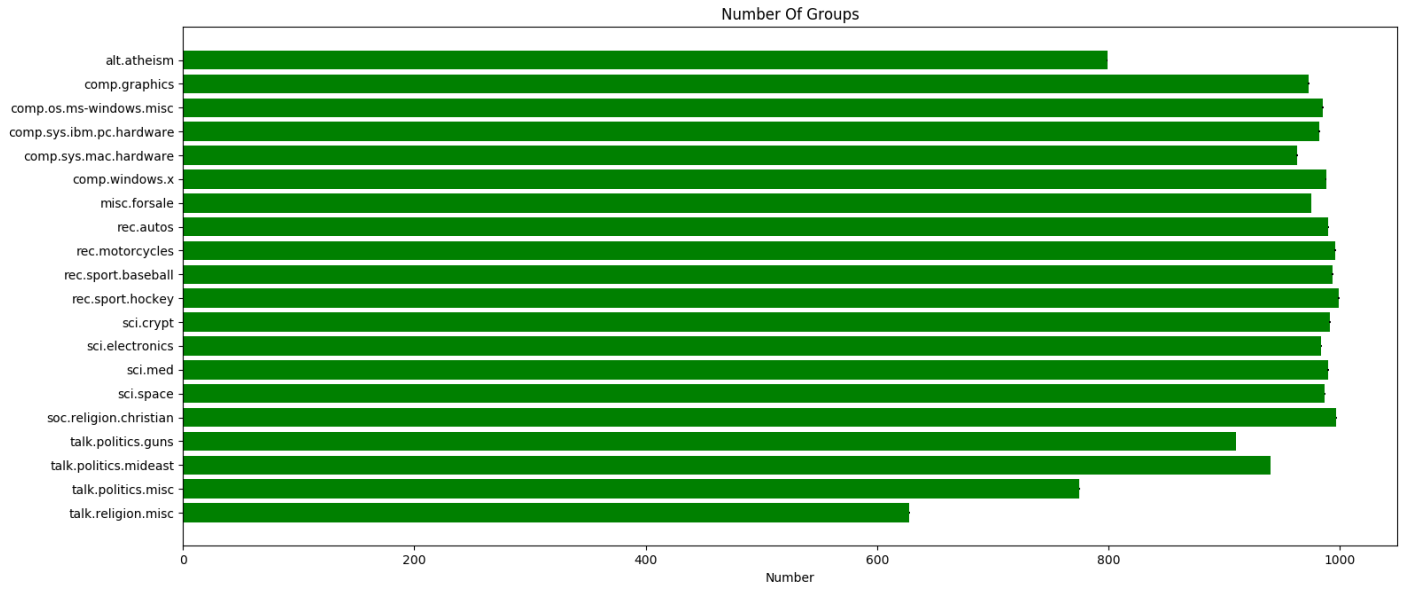
分词之后就是词、语句以及文章的表达。以英文为例，最常见的词语表述方式比如”cat“、”dog“，这些都是利用字母表示意思。统计语言处理里面，比较容易利用字母来描述概率模型，比如 ngram 模型 ，计算两个单词或者多个单词同时出现的概率，但是这些符号难以直接表示词与词之间的关联，也难以直接作为机器学习模型输入向量。对句子或者文章的表示，可以采用词袋子模型，即将段落或文章表示成一组单词，例如两个句子：”She loves cats.“、”He loves cats too.“ 我们可以构建一个词频字典：{"She": 1, "He": 1, "loves": 2 "cats": 2, "too": 1}。根据这个字典，我们能将上述两句话重新表达为下述两个向量: [1, 0, 1, 1, 0]和[0, 1, 1, 1, 1]，每 1 维代表对应单词的频率。

近年来，借助深度学习概念和性能强劲的硬件平台，Geofrey Hinton, Tomas Mikolov, Richard Socher 等学者深入开展了针对词向量的研究，将自然语言处理推向了新的高度。以词向量为基础，可以方便引入机器学习模型对文本进行分类、情感分析、预测、自动翻译等。最简单的词向量就是独热编码(one-hot encoder)，比如有三个单词”man"、”husband“、”dog“，将之分别表示为[0,0,1]，[0,1,0]，[1,0,0]，这些词向量可以作为机器学习模型的输入数值向量，但是它们依然难以表达关联性，而且当词库单词量庞大时，独热编码的维度也会十分巨大，给计算和存储带来不少问题。Mikolov、Socher 等人提出了 Word2Vec、GloVec 等词向量模型，能够比较好的解决这个问题，即用维数较少的向量表达词以及词之间的关联性。关于这些词向量模型的具体原理，可以阅读他们所发表的论文，主要是英文，中文网站上也出现了不少精彩的翻译和解读，可以参考某些关于自然语言处理的中文博客。

本项目目的就是利用上述自然语言处理技术结合所学机器学习知识对文档进行准确分类。在明确有已归类的正确信息下进行学习，所以属于监督学习，而整个数据包含 20 个分类，所以总体来说，该问题是一个多分类的监督学习任务。

分析数据

分类文本数据使用经典的 20 类新闻包，里面大约有 20000 条新闻，比较均衡地分成了 20 类，是比较常用的文本数据之一。每个类的数量如下图所示，该图代码在 Number_Plot.py：



数据以 txt 文件保存，除了正文，还有 Header，Footer 等各种附加信息。正文的风格，字数都有很大差异。随机选取一节加以分析：

Path: cantaloupe.srv.cs.cmu.edu!magnesium.club.cc.cmu.edu!news.sei.cmu.edu!cis.ohio-state.edu!zaphod.mps.ohio-state.edu!cs.utexas.edu!uunet!news.tek.com!vice!bobbe
From: bobbe@vice.ICO.TEK.COM (Robert Beauchaine)
Newsgroups: alt.atheism
Subject: Re: Amusing atheists and agnostics
Message-ID: <11860@vice.ICO.TEK.COM>
Date: 20 Apr 93 15:37:10 GMT
References: <timmbake.735204406@mcl> <madhausC5rFqo.9qL@netcom.com>
Organization: Tektronix Inc., Beaverton, Or.
Lines: 18

In article <madhausC5rFqo.9qL@netcom.com> madhaus@netcom.com (Maddi Hausmann) writes:
>
>"Clam" Bake Timmons = Bill "Shit Stirrer Connor"
>

Sorry, gotta disagree with you on this one Maddi (not the
resemblence to Bill. The nickname).

I prefer "Half" Bake'd Timmons

/\

Bob Beauchaine bobbe@vice.ICO.TEK.COM

They said that Queens could stay, they blew the Bronx away,
and sank Manhattan out at sea.

^^

其文本内容与 E-mail 类似。有一系列头标签，比如 Path,From,Newsgroups,Subject,Message-ID,Date,References,Organization,还有行数 Lines. 文本内容短的只有 10 行左右，而长的则有上千行。

解决方法

解决问题首先需要了解可以使用的工具。首先来看 CountVectorizer。这是一个将句子分成单词，然后变成矢量。在 CountVectorizer.py 中，我对其进行了简单的参数测试。

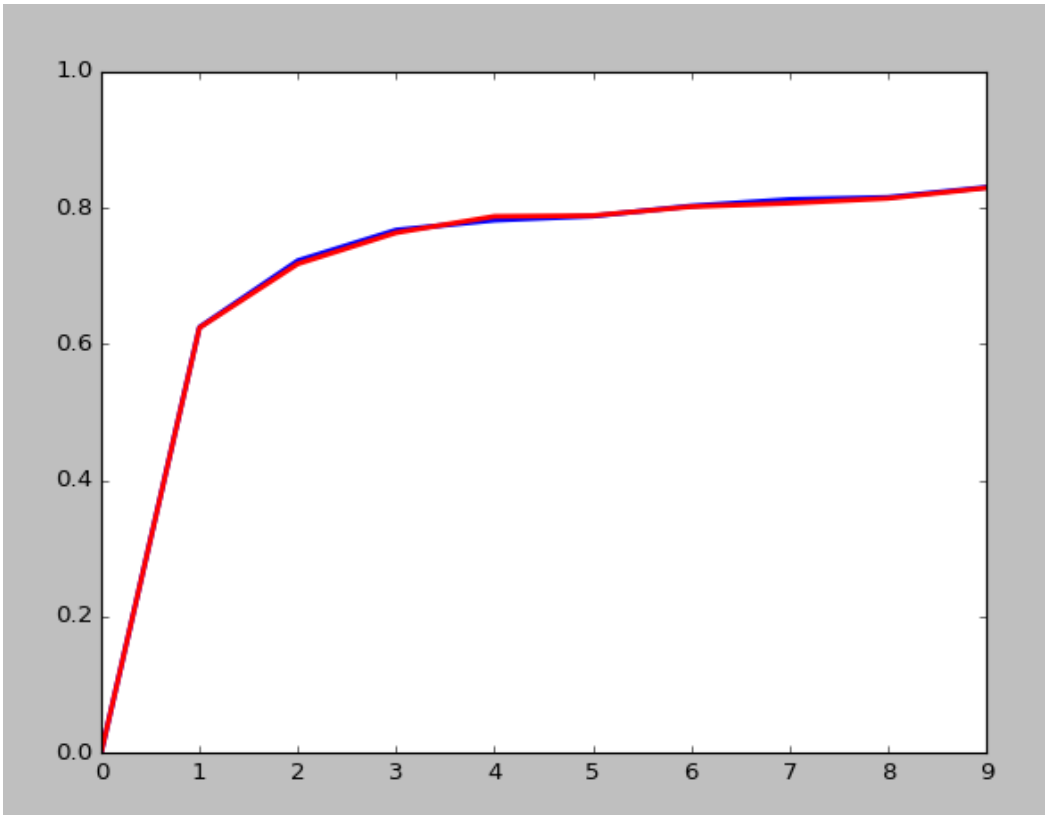
ngram_range，这个参数非常有意思，它可以让 CountVectorizer 不仅仅整合单个单词做特征，同时可以让两个以上的单词连起来做特征，比如 ngram_range=（1，2）就会让特征包含两个连续的词。但这个参数虽然增加了特征，但会让计算量增多，并且过多的连续单词，也并不会增加整个文档的识别率。

stop_words，有很多词汇其实并没有领域内的意思，比如 the，a，of 等，这种词汇加入到词向量中，只会增加计算量以及误判的可能，所以将一些常见词抛弃是很有必要的，而 SKlearn 已经在库中集成了这部分，只需要将 stop_words='english' 就可以屏蔽一些停止词。

另外还有 token_pattern, 在运行各种测试的时候，我发现 CountVectorizer 经常会把数字也作为特征加进词向量中。但我并不认为数字作为分类的标准是一件很好的事情。我希望可以把数字部分的词向量全部删除。而 CountVectorizer 并没有特别原生的方法支持我这个思路。但我发现了 token_pattern，这个选取词的过程，参数是正则表达式，例如使用'(?u)\\b[a-zA-Z]{2,10}\\b' 作为 pattern 时，就可以将全部的数字，或者含数字单词都删掉。同时还可以规定单词的字母数，比如我觉得当字母超过 10 个，就可能是乱打的词汇，进行屏蔽。下图为不同参数后的词向量结果：

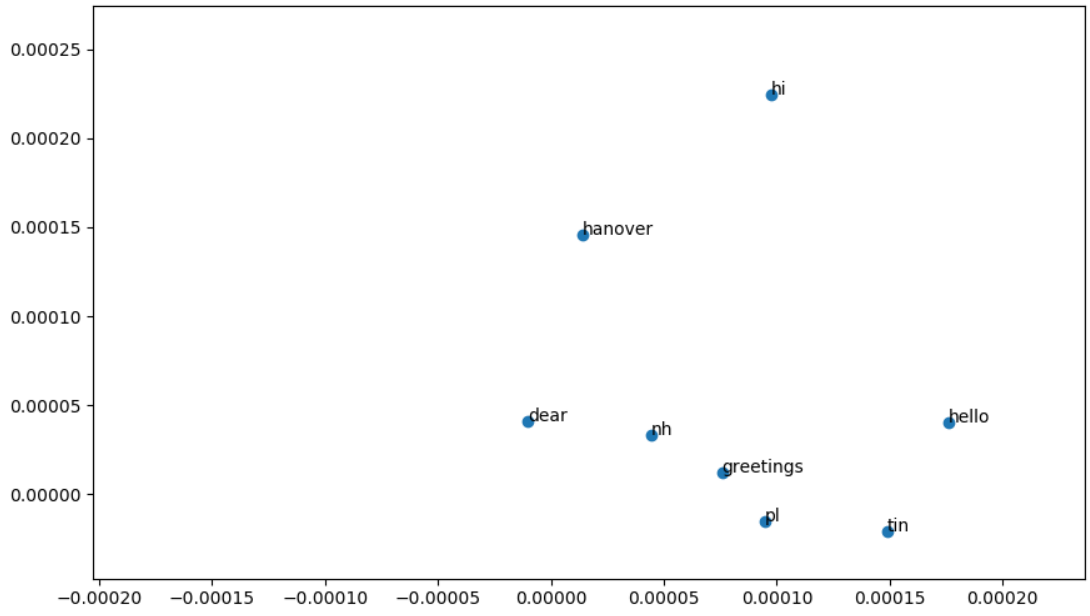
```
('he is a boy. 123 22', 'She is a girl. 123 23')
[u'123', u'22', u'23', u'boy', u'girl', u'he', u'is', u'she']
  (0, 1)        1
  (0, 0)        1
  (0, 3)        1
  (0, 6)        1
  (0, 5)        1
  (1, 2)        1
  (1, 4)        1
  (1, 7)        1
  (1, 0)        1
  (1, 6)        1
[u'123', u'123 22', u'123 23', u'22', u'23', u'boy', u'boy 123', u'girl', u'girl
123', u'he', u'he is', u'is', u'is boy', u'is girl', u'she', u'she is']
  (0, 1)        1
  (0, 6)        1
  (0, 12)       1
  (0, 10)       1
  (0, 3)        1
  (0, 0)        1
  (0, 5)        1
  (0, 11)       1
  (0, 9)        1
  (1, 2)        1
  (1, 8)        1
  (1, 13)       1
  (1, 15)       1
  (1, 4)        1
  (1, 7)        1
  (1, 14)       1
  (1, 0)        1
  (1, 11)       1
[u'123', u'123 22', u'123 23', u'22', u'23', u'boy', u'boy 123', u'girl', u'girl
123']
  (0, 1)        1
  (0, 6)        1
  (0, 3)        1
  (0, 0)        1
  (0, 5)        1
  (1, 2)        1
  (1, 8)        1
  (1, 4)        1
  (1, 7)        1
  (1, 0)        1
[u'boy', u'girl']
  (0, 0)        1
  (1, 1)        1
```

对于删除数字后的结果，我做了一个学习曲线的对比，Without_Number_Plot.py，如图所示，去除数字后并没有显著影响学习曲线，所以表示数字是可以被忽略的。

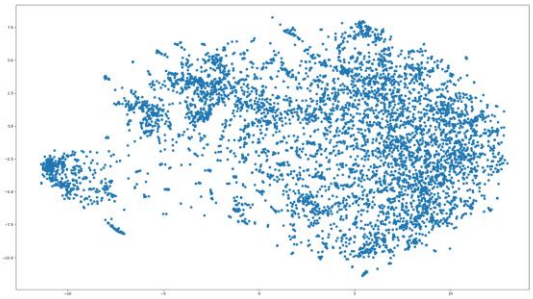


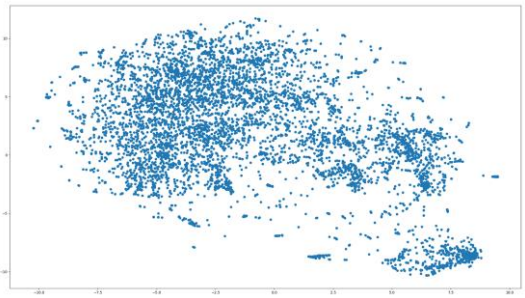
另一个工具是 TfidfTransformer，它可以将上面的词向量进行 TF-IDF 运算，得出权值。

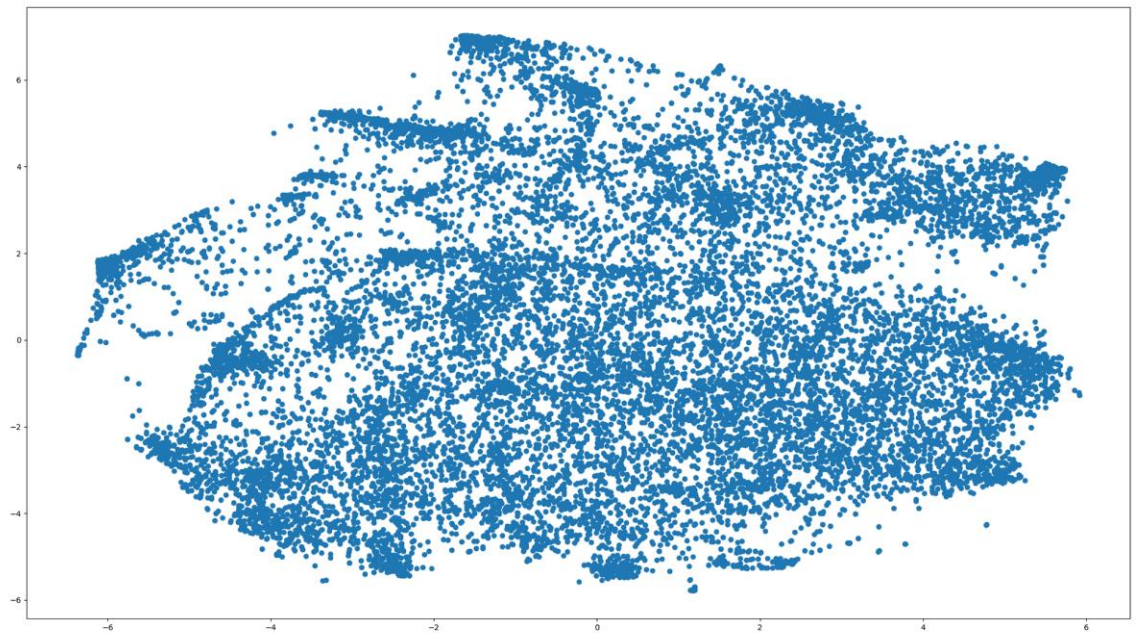
利用 Word2Vec[2] 方式即词向量模型表示每篇文档，利用文本数据对词向量进行训练，将每个单词表示成向量形式。 首先用 Word2Vec_CreateModel.py 建立 Word2Vec 的模型，然后在 Word2Vec_OneWordPic.py 汇出某一个单词的相关词汇。如下图：



然后我尝试将整个 Word2Vec 的全部点都展示在一张图上，Word2Vec_BigPicture.py，如下：





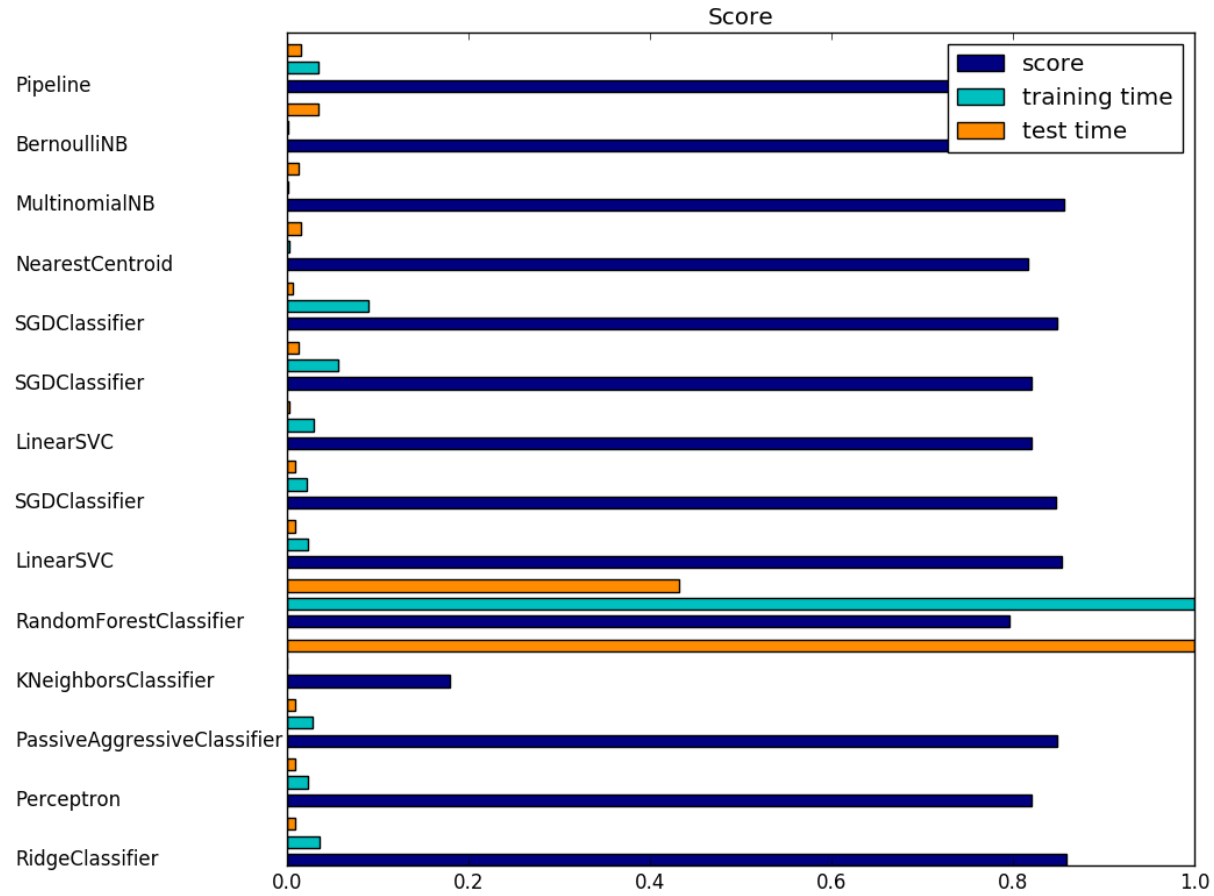


由不同的参数，Word2Vec 的整体图像差异度挺大，相对而言，我觉得词向量以及 TF-IDF 是一种比较稳定的算法。而 Word2Vec，则变动幅度较大，有一定的随机性。针对文档分类而言，Word2Vec，并不是一种非常有效的方法。所以，最终还是以词向量和 TF-IDF 的结合，作为特征的基础。

基准模型

首先对比一下各种模型大致的区别，ALL_MODEL.py，此代码摘自互联网，仅做一些修改，为了更了解各种模型，因为本人计算机算力有限，并且时间紧迫，所以只节选了 7 类子类的数据进行计算。在此文件中，普遍准确率低于 90%。所以设定准确率目标为 90%。

这 7 类分别是：['alt.atheism','comp.graphics','misc.forsale','rec.autos','sci.crypt', 'soc.religion.christian','talk.politics.guns',]



由图中可知，大部分的算法的分数基本一致，但训练的时间有较明显的差异，而测试时间则对于 K 邻近算法而言最为严重。选择 MultinomialNB、LinearSVC、以及 SGDClassifier，作为进一步了解使用的模型。

然后对涉及到的模型进行一些描述：

1. Tf-idf，其 TF 表示词频，idf 表示逆向文件词频。实际上是 tf*idf。Idf 需要稍作解释：如果某词条 A 的总词频 TF 少，但是在某一类文档的词频多，则词条 A 更能区分文档的类别。
2. Word2Vec，这个模型是将全部词汇变成向量的形式，其中词与词之间的距离，就是词向量的关键。Word2Vec 有很多实现方法，比如 Gensim 的包 gensim.models.word2vec ，也可以使用 Tensorflow，或者自己写模型代码。Tensorflow 可以更好的调用 GPU 资源让性能达到最优。但因为本人没有很好的 GPU，所以并没有使用 Tensorflow，直接调用 gensim.models.word2vec 的包来实现。而和性能有关的参数比如 size 词向量的维度，min_count 词向量建立的最小频度，sample 采样率。通过调节这些参数，可以优化模型。
3. MultinomialNB，是多项式的朴素贝叶斯分类器。区别于普通的朴素贝叶斯分类器，它会做一些平滑处理。朴素贝叶斯分类器之所以被称为朴素，是因为它有一个基本的假设：给定目标值时属性之间相互条件独立。虽然这在现实世界几乎不可能，但是基于这个假设的模型所得出的结论，还是值得参考的。其中的 Beta 指的是贝塔分布，区别于正态分布的左右对称，贝塔分布是一种非对称的分布方式。

4. LinearSVC 回归，线性支持向量机，支持向量机的模型是构造一个超平面，进而对数据进行区分归类，可以用于分类，回归。SVM 的核函数有不同种类，决定了超平面的类型，比如以下：

   线性：$K(v1,v2)=<v1,v2>$

   多项式：$K(v_1,v_2)=(\gamma<v_1,v_2>+c)^n$

   RBF：$K(v_1,v_2)=\exp(-\gamma||v_1-v_2||^2)$

   Sigmoid：$K(v_1,v_2)=\tanh(\gamma<v_1,v_2>+c)$

   SVM 的目的其实就是找一个区分数据最合适的超平面。而寻找最优的超平面，可以被认为是等价于找到最大的间距。

5. SGDClassifier，这个模型也是一个线性函数的模型，不同的是，它使用了随机梯度下降的方式进行训练，所以每一次训练没有使用全部的样本，收敛速度会加快。

评估指标

本项目使用准确率作为评估指标。虽然二十分类这种多分类的问题可以用混淆矩阵来做细化的评估指标，但是我个人还是喜欢以一个分数来评价整个模型。所以可以将多分类，通过是非判断，简化为二分类的问题。从而建立二分类的评估标准和指标。公式如下：

准确率$=\frac{\sum_{i=1}^{n}A(y_i=Y_i)}{n}$

其中 y_i 表示预测分类，Y_i 表示实际分类。而 A 函数表示如果两者相等，A=1，否则 A=0 。

.

结论

首先，通过 MultinomialNB_Search.py 找一下较优的参数，如下：

```
Performing grid search...
pipeline: ['vect', 'tfidf', 'clf']
parameters:
{'clf__alpha': (0.01, 0.0001),
 'tfidf__use_idf': (True, False),
 'vect__max_df': (0.5, 0.75),
 'vect__min_df': (0.01,),
 'vect__ngram_range': ((1, 2),)}
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed:   17.9s finished
done in 21.753s

Best score: 0.774
Best parameters set:
        clf__alpha: 0.01
        tfidf__use_idf: False
        vect__max_df: 0.5
        vect__min_df: 0.01
        vect__ngram_range: (1, 2)
----------------------TEST------------------------
             precision    recall  f1-score   support

          0       0.65      0.44      0.52       319
          1       0.79      0.82      0.81       389
          2       0.84      0.85      0.85       390
          3       0.72      0.73      0.73       396
          4       0.85      0.69      0.76       396
          5       0.58      0.85      0.69       398
          6       0.76      0.71      0.73       364

avg / total       0.74      0.73      0.73      2652
```

主要的参数是 alpha，以及是否使用 idf，还有 df 的上限。但准确率一般，只是在 74 左右。

然后通过 SGDClassifier_Search.py 找一下较优的参数，结果如下：

```
Performing grid search...
pipeline: ['vect', 'tfidf', 'clf']
parameters:
{'clf__alpha': (0.001, 0.0001),
 'clf__penalty': ('l2', 'l1'),
 'tfidf__use_idf': (True,),
 'vect__max_df': (0.5,),
 'vect__min_df': (0.01,),
 'vect__ngram_range': ((1, 2),)}
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[Parallel(n_jobs=-1)]: Done  10 out of  12 | elapsed:   11.4s remaining:    2.2s
[Parallel(n_jobs=-1)]: Done  12 out of  12 | elapsed:   11.7s finished
done in 15.330s

Best score: 0.779
Best parameters set:
        clf__alpha: 0.001
        clf__penalty: 'l2'
        tfidf__use_idf: True
        vect__max_df: 0.5
        vect__min_df: 0.01
        vect__ngram_range: (1, 2)
----------------------TEST------------------------
             precision    recall  f1-score   support

          0       0.62      0.43      0.51       319
          1       0.75      0.82      0.78       389
          2       0.66      0.88      0.76       390
          3       0.81      0.68      0.74       396
          4       0.80      0.68      0.74       396
          5       0.69      0.81      0.75       398
          6       0.73      0.68      0.70       364

avg / total       0.72      0.72      0.72      2652
```

参数与最开始算法类似，得出的结果仍然不高，只是在 72%左右。

最后是 LinearSVC_Search.py 结果如下：

```
Performing grid search...
pipeline: ['vect', 'tfidf', 'svm']
parameters:
{'tfidf__use_idf': (True,),
 'vect__max_df': (0.5,),
 'vect__min_df': (0.01,),
 'vect__ngram_range': ((1, 2),)}
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    6.2s finished
done in 9.735s

Best score: 0.760
Best parameters set:
        tfidf__use_idf: True
        vect__max_df: 0.5
        vect__min_df: 0.01
        vect__ngram_range: (1, 2)
----------------------TEST------------------------
             precision    recall  f1-score   support

          0       0.55      0.48      0.51       319
          1       0.77      0.80      0.78       389
          2       0.81      0.84      0.82       390
          3       0.66      0.77      0.71       396
          4       0.75      0.69      0.72       396
          5       0.68      0.75      0.71       398
          6       0.75      0.63      0.68       364

avg / total       0.71      0.71      0.71      2652
```

经过逐个测试，最后使用 Final_Run.py 得出较高的准确率，91%。

对于文本的预处理，包含去除 english 的停止词，并且将 n-gram 设置为（1,2），设置最小 df 为 0.005，最大的 df 为 0.5。并且去除数字和超过 20 个字母长度的异常词。

在这些预处理之前，词袋子的前一百个为：

[u'00', u'000', u'00000000', u'00000000b', u'00000001', u'00000001b', u'00000010', u'00000010b', u'00000011', u'00000011b', u'00000100', u'00000100b', u'00000101', u'00000101b', u'00000110', u'00000110b', u'00000111', u'00000111b', u'000005102000', u'00001000', u'00001000b', u'00001001', u'00001001b', u'00001010', u'00001010b', u'00001011', u'00001011b', u'00001100', u'00001100b', u'00001101', u'00001101b', u'00001110', u'00001110b', u'00001111', u'00001111b', u'0001', u'00010000', u'00010000b', u'00010001', u'00010001b', u'00010010', u'00010010b', u'00010011', u'00010011b', u'000100255pixel', u'00010100', u'00010100b', u'00010101', u'00010101b', u'00010110', u'00010110b', u'00010111', u'00010111b', u'00011000', u'00011000b', u'00011001', u'00011001b', u'00011010', u'00011010b', u'00011011', u'00011011b', u'00011100', u'00011100b', u'00011101', u'00011101b', u'00011110', u'00011110b', u'00011111', u'00011111b', u'000152', u'000359', u'0004', u'000406', u'0005111312', u'0005111312na3em', u'0005895485', u'000601', u'0007', u'000710', u'000k', u'000mi', u'000miles', u'000s', u'000usd', u'001', u'0010', u'00100000', u'00100000b', u'00100001', u'00100001b', u'00100010', u'00100010b', u'00100011', u'00100011b', u'00100100', u'00100100b', u'00100101', u'00100101b', u'00100110', u'00100110b']

做过预处理之后，词袋子的前一百个为：

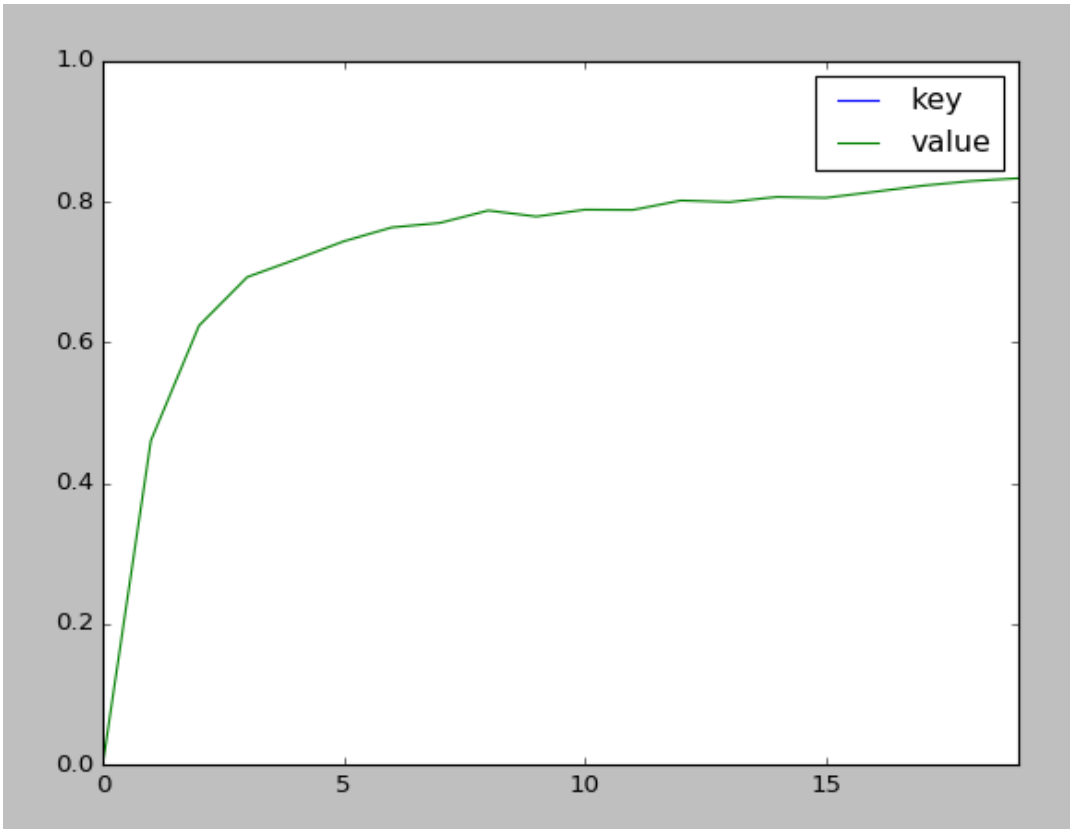
[u'aaron', u'aaron ray', u'ab', u'abiding', u'abiding citizens', u'ability', u'able', u'absence', u'absolute', u'absolutely', u'abstract', u'absurd', u'abuse', u'ac', u'ac uk', u'academic', u'academic computer', u'accept', u'acceptable', u'acceptance', u'acceptance wiretap', u'accepted', u'accepting', u'accepting jeesus', u'access', u'access digex', u'access online', u'access unix', u'accident', u'accidental', u'accidents', u'according', u'account', u'accounts', u'accuracy', u'accurate', u'achieve', u'acknowledge', u'aclu', u'acm', u'acns', u'acquire', u'acs', u'acs oakland', u'acs ohio', u'act', u'action', u'actions', u'active', u'activities', u'activity', u'acts', u'actual', u'actually', u'ad', u'adam', u'add', u'added', u'adding', u'addition', u'additional', u'address', u'addressed', u'addresses', u'adequate', u'administration', u'admit', u'adobe', u'adopt', u'ads', u'advance', u'advanced', u'advantage', u'advice', u'advocate', u'affair', u'affect', u'affected', u'afraid', u'age', u'agencies', u'agency', u'agent', u'agents', u'ages', u'ago', u'agree', u'agreed', u'agreement', u'ah', u'ahead', u'ai', u'ai uga', u'aid', u'ain', u'air', u'aka', u'al', u'alan', u'albert']

虽然准确率相差无几，但是对于人的观察来说，明显可以感觉到词袋子中的词汇更有意义。

可视化图表

首先做一个测试样本和准确率的关系图。以 20 为样本倍数，初始值设定样本数以及准确率皆为 0，随着样本数的增多，准确率也稳步上升，到达一定程度后上升变得缓慢，逐渐稳定。

代码为：Sample_Number_Plot.py



思考与改进

该项目问题的定义非常清晰，就是对文本数据的分类。

而分析的过程，就是对数据本身，包括格式，内容，特征等各种因素进行研究。同时也需要分析算法模型如何与实际数据结合，进而得出更好的结果。

文本处理的预处理同样包含很多内容。对于文本的标签行我觉得可以全部忽略。并且对全部的标点符号，特殊符号都需要去除。停止词同样全部筛选掉。min_df 是个很有作用的参数，在没有设置这个参数时，可以看到关键词很多都是一些看似乱码的字符串，但是当 min_df 定义为足够的数值后，可以明显看出关键字的定义大部分可以理解，虽然正确率甚至有所降低，但对于语言的理解是更进一步的。

进而做最后选择的方法的描述。对于 TF-IDF 实际是以某词在某文出现次数占全文字数的百分比，以及含某词的文数占总文数的百分比的乘积来表示。而 Word2Vec 实际是对于相似词的聚合。我计划通过这两种模型的结合，得出某相似意义的词在某文全文字数百分比，和含某相似意义的词占总文书百分比的乘积来进行优化算法。但实际上，本人对于 Word2Vec 的理解，未能用于文本分类，所以最终还是以 Tf-idf 为主要的特征值。

而具体的分类算法我觉得 SVM 会有比较高的分数，但最终是 MultinomialNB 有更好的准确率，可能与模型本身的参数设置有关。

因为本人计算机算力有限，并且时间紧迫，所以只节选了 7 类子类的数据进行计算。在此文件中，普遍准确率低于 90%。所以设定准确率目标为 90%。

我还尝试了使用 SelectKBest，对特征的维度进行降维，指定特征的个数。又尝试了 GradientBoostingClassifier 进行分类。最终仍是 SGDClassifier 有最高的精确度。

最终准确率达到 91.3%。

最后仍进行尝试性改进，对模型进行集成融合，尝试使用两种模型：朴素贝叶斯加上 SVC 同时进行分析，如 EnsembleVoteClassifier_test.py 所示，准确率提高 0.1%达到 91.4%。

同时想对 Textcnn 模型进行尝试，Textcnn 想对 Word2Vec 模型而言，更贴近本项目的目标，如果使用 Textcnn，从模型上而言可以直接根据 Textcnn 的结果来对文本进行分类。但是因为我使用的是 windows 系统，并且 python 版本为 2.7，无法使用 tensorflow，所以并没有对其进行实践性尝试。留待以后电脑升级后进行尝试。

以上是我对这个项目的报告。

谢谢。

参考

【1】 NLP：https://baike.baidu.com/item/nlp/25220

【2】 word2vec：https://baike.baidu.com/item/Word2vec/22660840?fr=aladdin

【3】 SVM： https://baike.baidu.com/item/svm