L7-Topic Models

目标

用 Topic Models 做情感分类的研究，数据集：http://ai.stanford.edu/~amaas/data/sentiment/

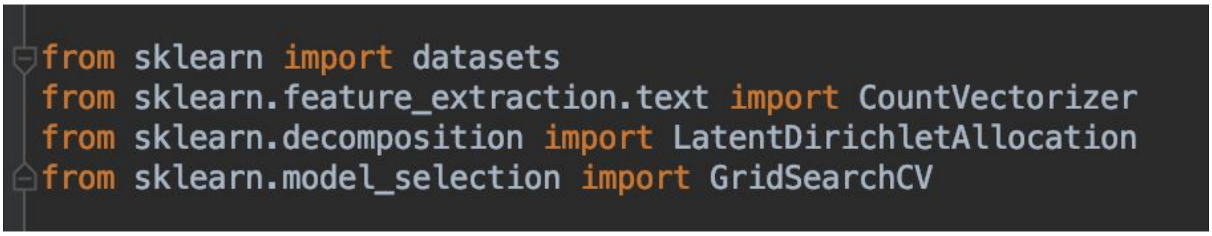
分析

Topic Models 是用来做无监督学习使用的。所谓无监督学习就是给定的数据没有被事先分好类的学习过程。这个实际应用比如因为有些数据比较复杂我们不了解(缺乏足够的先验知识)难以人工标注类别或进行人工类别标注的成本太高，这个时候无监督学习就能发挥很大的作用。

根据作业4 L4-LogisticRegression 的数据集，我们在 L4 使用了逻辑回顾进行了电影影评数据的分类。其中提到训练数据集中存在一个 unsup 的文件夹，这个文件夹里面有50000 条未分类的数据，那么我们通过使用 Topic model LDA 对这个数据集进行划分。

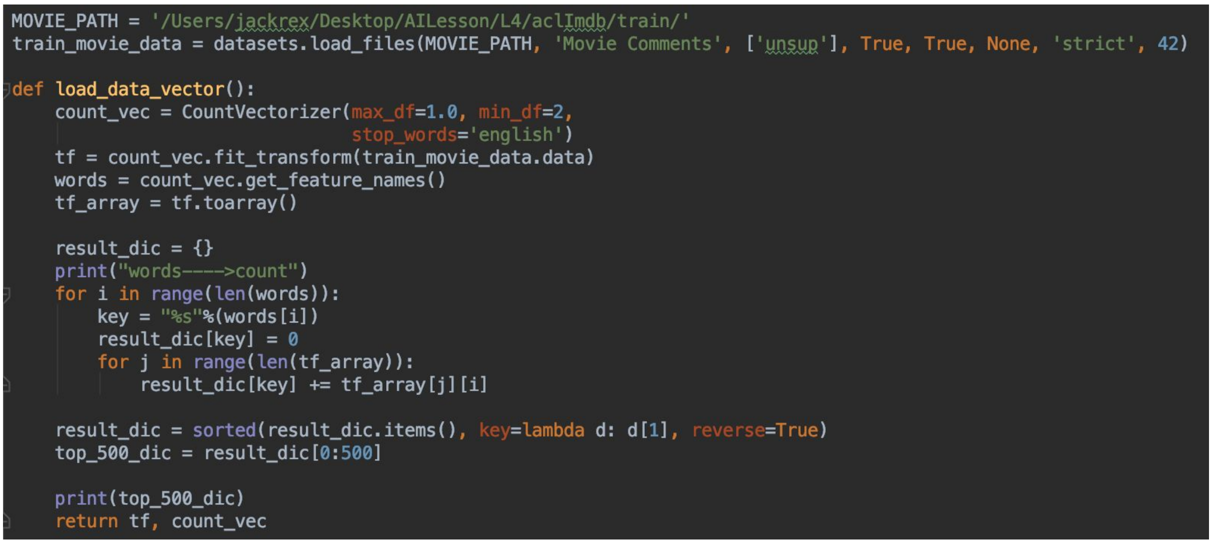
设计

根据需要导入 sklearn 所需要的类



```
from sklearn import datasets
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.model_selection import GridSearchCV
```

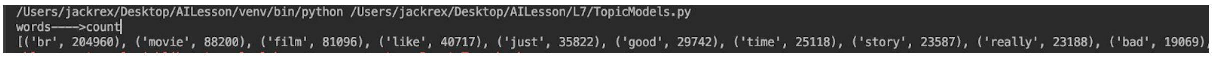
首先我们通过加载 unsup 数据，使用 CountVectorizer 进行分词和矩阵转化，由于LDA是基于词频统计的，因此一般不用TF-IDF来做文档特征。



```
MOVIE_PATH = '/Users/jackrex/Desktop/AILesson/L4/aclImdb/train/'
train_movie_data = datasets.load_files(MOVIE_PATH, 'Movie Comments', ['unsup'], True, True, None, 'strict', 42)

def load_data_vector():
    count_vec = CountVectorizer(max_df=1.0, min_df=2,
                                stop_words='english')
    tf = count_vec.fit_transform(train_movie_data.data)
    words = count_vec.get_feature_names()
    tf_array = tf.toarray()

    result_dic = {}
    print("words---->count")
    for i in range(len(words)):
        key = "%s"%(words[i])
        result_dic[key] = 0
        for j in range(len(tf_array)):
            result_dic[key] += tf_array[j][i]

    result_dic = sorted(result_dic.items(), key=lambda d: d[1], reverse=True)
    top_500_dic = result_dic[0:500]

    print(top_500_dic)
    return tf, count_vec
```

我们定义词的频率出现2个以上为有效词，stop words 使用英文过滤，出于好奇我们筛出了unsup 里面前500个高频词汇。

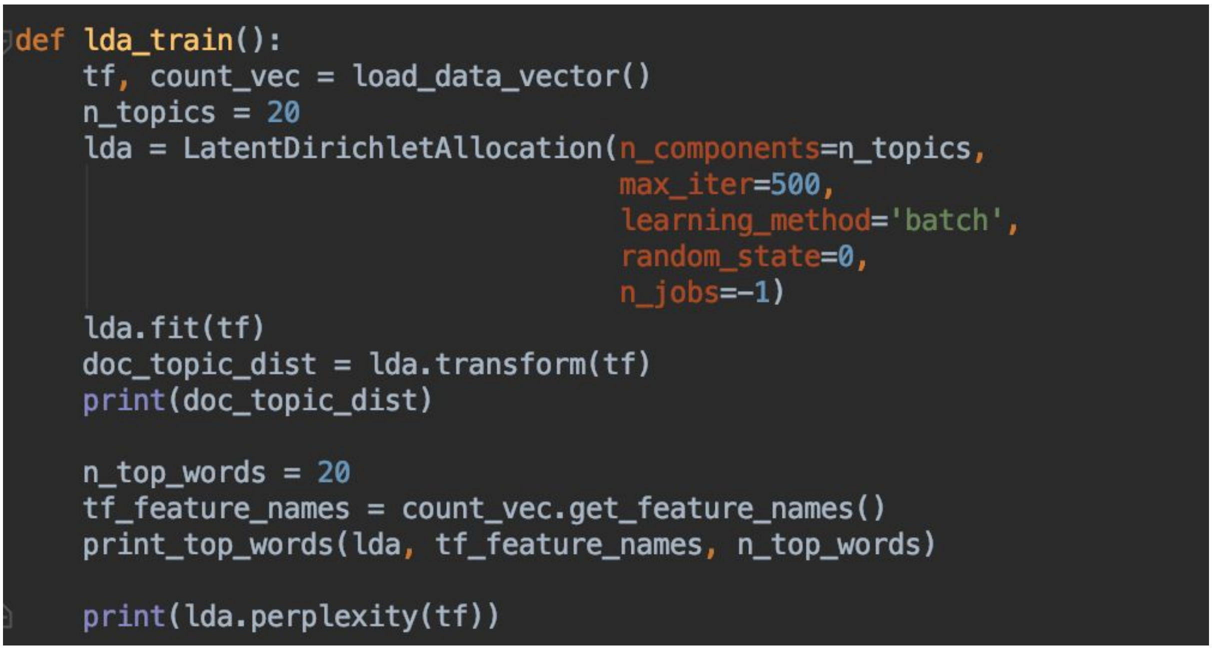
结果如下：



```
/Users/jackrex/Desktop/AILesson/venv/bin/python /Users/jackrex/Desktop/AILesson/L7/TopicModels.py
words---->count
[('br', 204960), ('movie', 88200), ('film', 81096), ('like', 40717), ('just', 35822), ('good', 29742), ('time', 25118), ('story', 23587), ('really', 23188), ('bad', 19069),
```

其中词频最高的是 br 这个应该是要去掉的这是一个 bad case，接下来排名最高的是 movie film 我们可以从这个分布看到这50000 条数据讨论的是和电影相关的话题。 接下来我们看到 like just good time story really bad 等等，从 good like bad 这种带有情感色彩的词可以看到描述的内容是评判电影好坏的描述，而且从目前 good like bad 的分类来看，里面对电影评价好描述是是多于坏的。

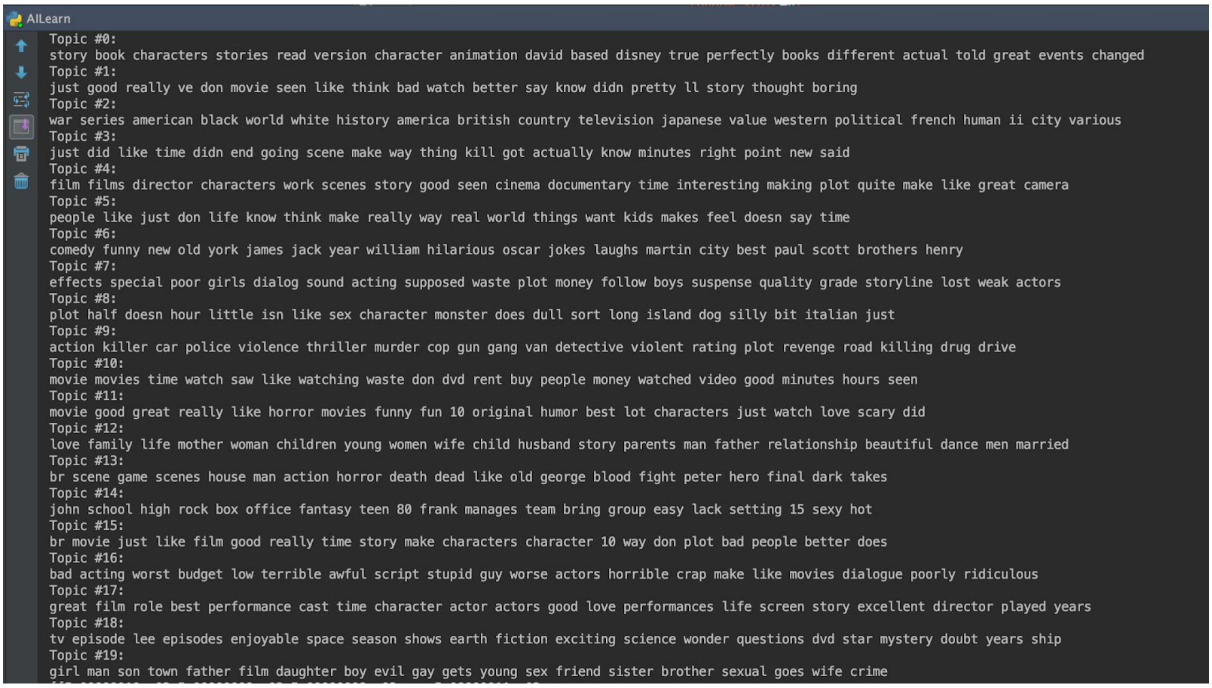
接下来我们使用LDA 进行数据拟合和分 Topic 我们定义分20组 topics， LDA由于是非监督的，没有特别好的评价标准，LDA的作者推荐使用困惑度(Perplexity)来衡量主题的好坏。



```
def lda_train():
    tf, count_vec = load_data_vector()
    n_topics = 20
    lda = LatentDirichletAllocation(n_components=n_topics,
                                    max_iter=500,
                                    learning_method='batch',
                                    random_state=0,
                                    n_jobs=-1)
    lda.fit(tf)
    doc_topic_dist = lda.transform(tf)
    print(doc_topic_dist)

    n_top_words = 20
    tf_feature_names = count_vec.get_feature_names()
    print_top_words(lda, tf_feature_names, n_top_words)

    print(lda.perplexity(tf))
```

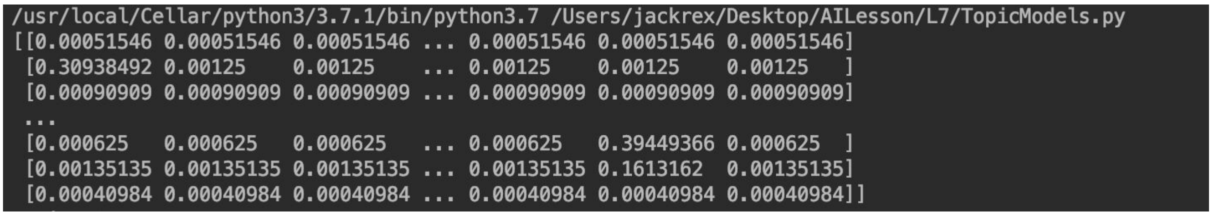
LDA 的参数选择，我们划分隐含主题数目，n_componets 为20个，这个可以不断的调整。max_iter 为EM 最大迭代次数，学习方法有两种，batch 和 online， online即在线变分推断EM算法，在batch的基础上引入了分步训练，将训练样本分批，逐步一批批的用样本更新主题词分布的算法。样本量不大只是用来学习的话用batch比较好，由于我们样本量比较大，所以选择online。最后我们开启所有 cpu 多核 n_jobs = -1 去跑这个计算，希望更快一点。

我们通过上述代码，打出当前主题的列表和困惑度，困惑度越大，代表分类越不清晰。其中20个Topic 如下图所示：



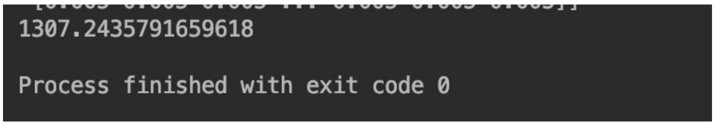
```
Topic #0:
story book characters stories read version character animation david based disney true perfectly books different actual told great events changed
Topic #1:
just good really ve don movie seen like think bad watch better say know didn pretty ll story thought boring
Topic #2:
war series american black world white history america british country television japanese value western political french human ii city various
Topic #3:
just did like time didn end going scene make way thing kill got actually know minutes right point new said
Topic #4:
film films director characters work scenes story good seen cinema documentary time interesting making plot quite make like great camera
Topic #5:
people like just don life know think make really way real world things want kids makes feel doesn say time
Topic #6:
comedy funny new old york james jack year william hilarious oscar jokes laughs martin city best paul scott brothers henry
Topic #7:
effects special poor girls dialog sound acting supposed waste plot money follow boys suspense quality grade storyline lost weak actors
Topic #8:
plot half doesn hour little isn like sex character monster does dull sort long island dog silly bit italian just
Topic #9:
action killer car police violence thriller murder cop gun gang van detective violent rating plot revenge road killing drug drive
Topic #10:
movie movies time watch saw like watching waste don dvd rent buy people money watched video good minutes hours seen
Topic #11:
movie good great really like horror movies funny fun 10 original humor best lot characters just watch love scary did
Topic #12:
love family life mother woman children young women wife child husband story parents man father relationship beautiful dance men married
Topic #13:
br scene game scenes house man action horror death dead like old george blood fight peter hero final dark takes
Topic #14:
john school high rock box office fantasy teen 80 frank manages team bring group easy lack setting 15 sexy hot
Topic #15:
br movie just like film good really time story make characters character 10 way don plot bad people better does
Topic #16:
bad acting worst budget low terrible awful script stupid guy worse actors horrible crap make like movies dialogue poorly ridiculous
Topic #17:
great film role best performance cast time character actor actors good love performances life screen story excellent director played years
Topic #18:
tv episode lee episodes enjoyable space season shows earth fiction exciting science wonder questions dvd star mystery doubt years ship
Topic #19:
girl man son town father film daughter boy evil gay gets young sex friend sister brother sexual goes wife crime
```

每个topic 由若干个主题词组成。同时也打出了使用 lda transform 后，每篇文本对于这个主题来说的概率如下



```
/usr/local/Cellar/python3/3.7.1/bin/python3.7 /Users/jackrex/Desktop/AILesson/L7/TopicModels.py
[[0.00051546 0.00051546 0.00051546 ... 0.00051546 0.00051546 0.00051546]
 [0.30938492 0.00125    0.00125    ... 0.00125    0.00125    0.00125   ]
 [0.00090909 0.00090909 0.00090909 ... 0.00090909 0.00090909 0.00090909]
 ...
 [0.000625   0.000625   0.000625   ... 0.000625   0.39449366 0.000625  ]
 [0.00135135 0.00135135 0.00135135 ... 0.00135135 0.1613162  0.00135135]
 [0.00040984 0.00040984 0.00040984 ... 0.00040984 0.00040984 0.00040984]]
```

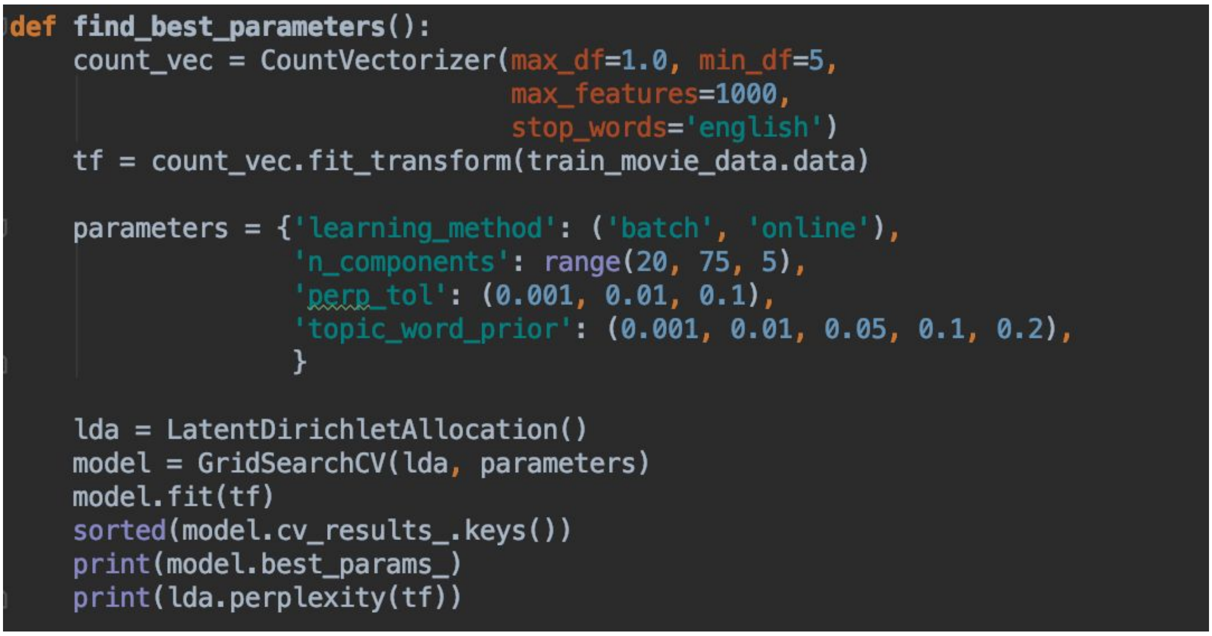
困惑度



```
1307.2435791659618

Process finished with exit code 0
```

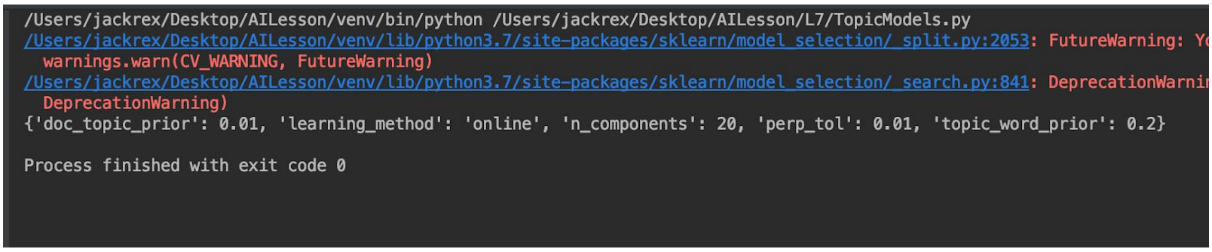
可以看到困惑度为，这个结果还是有点大的。

那么如何找到 LDA 的最优参数呢，如何减少困惑度？通过前两节说用的GridSearchCV 进行参数暴力搜索。



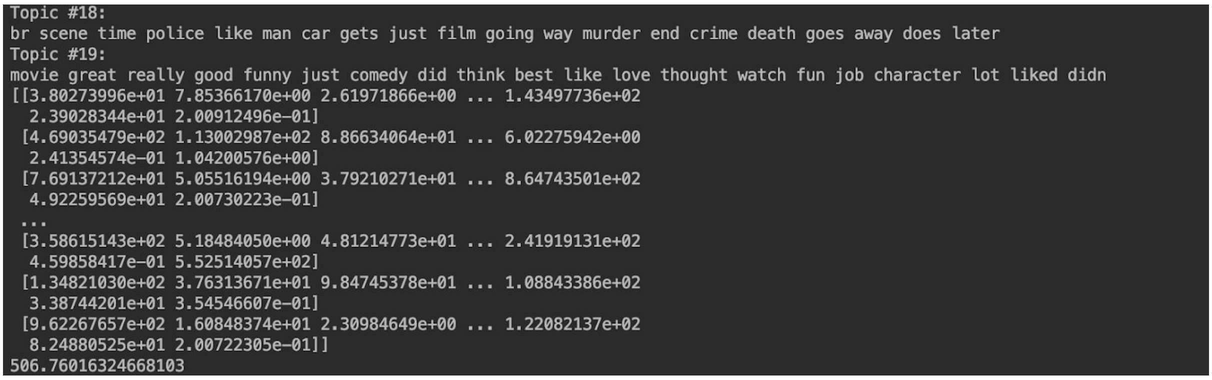
```python
def find_best_parameters():
    count_vec = CountVectorizer(max_df=1.0, min_df=5,
                                max_features=1000,
                                stop_words='english')
    tf = count_vec.fit_transform(train_movie_data.data)

    parameters = {'learning_method': ('batch', 'online'),
                  'n_components': range(20, 75, 5),
                  'perp_tol': (0.001, 0.01, 0.1),
                  'topic_word_prior': (0.001, 0.01, 0.05, 0.1, 0.2),
                  }

    lda = LatentDirichletAllocation()
    model = GridSearchCV(lda, parameters)
    model.fit(tf)
    sorted(model.cv_results_.keys())
    print(model.best_params_)
    print(lda.perplexity(tf))
```

最后通过 GridSearch 查找到最优的一组参数如下。



```
/Users/jackrex/Desktop/AILesson/venv/bin/python /Users/jackrex/Desktop/AILesson/L7/TopicModels.py
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWarning: Yo
  warnings.warn(CV_WARNING, FutureWarning)
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-packages/sklearn/model_selection/_search.py:841: DeprecationWarni
  DeprecationWarning)
{'doc_topic_prior': 0.01, 'learning_method': 'online', 'n_components': 20, 'perp_tol': 0.01, 'topic_word_prior': 0.2}

Process finished with exit code 0
```

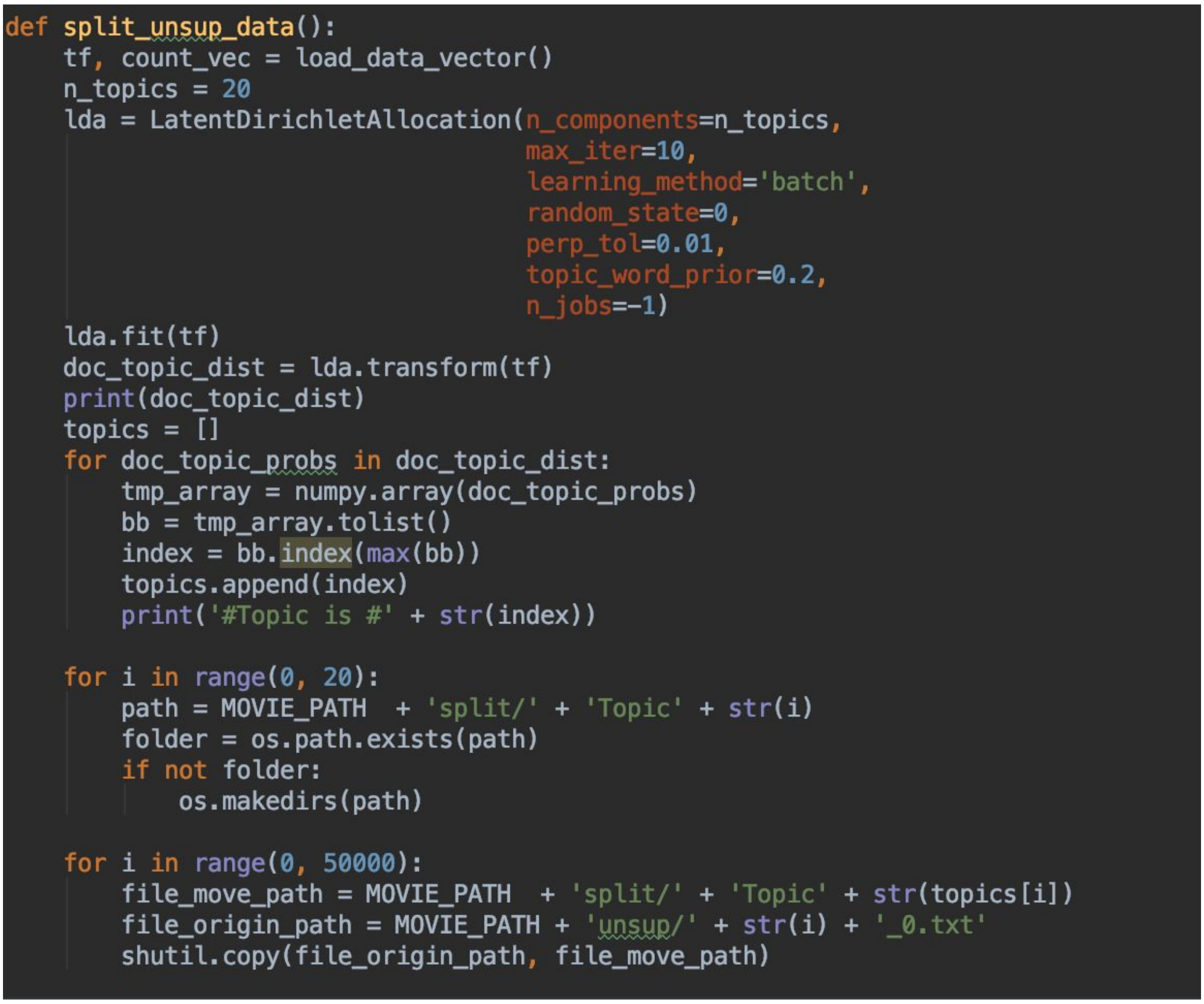
通过这组参数带入获得困惑度为 506。



```
Topic #18:
br scene time police like man car gets just film going way murder end crime death goes away does later
Topic #19:
movie great really good funny just comedy did think best like love thought watch fun job character lot liked didn
[[3.80273996e+01 7.85366170e+00 2.61971866e+00 ... 1.43497736e+02
  2.39028344e+01 2.00912496e-01]
 [4.69035479e+02 1.13002987e+02 8.86634064e+01 ... 6.02275942e+00
  2.41354574e-01 1.04200576e+00]
 [7.69137212e+01 5.05516194e+00 3.79210271e+01 ... 8.64743501e+02
  4.92259569e+01 2.00730223e-01]
 ...
 [3.58615143e+02 5.18484050e+00 4.81214773e+01 ... 2.41919131e+02
  4.59858417e-01 5.52514057e+02]
 [1.34821030e+02 3.76313671e+01 9.84745378e+01 ... 1.08843386e+02
  3.38744201e+01 3.54546607e-01]
 [9.62267657e+02 1.60848374e+01 2.30984649e+00 ... 1.22082137e+02
  8.24880525e+01 2.00722305e-01]]
506.76016324668103
```

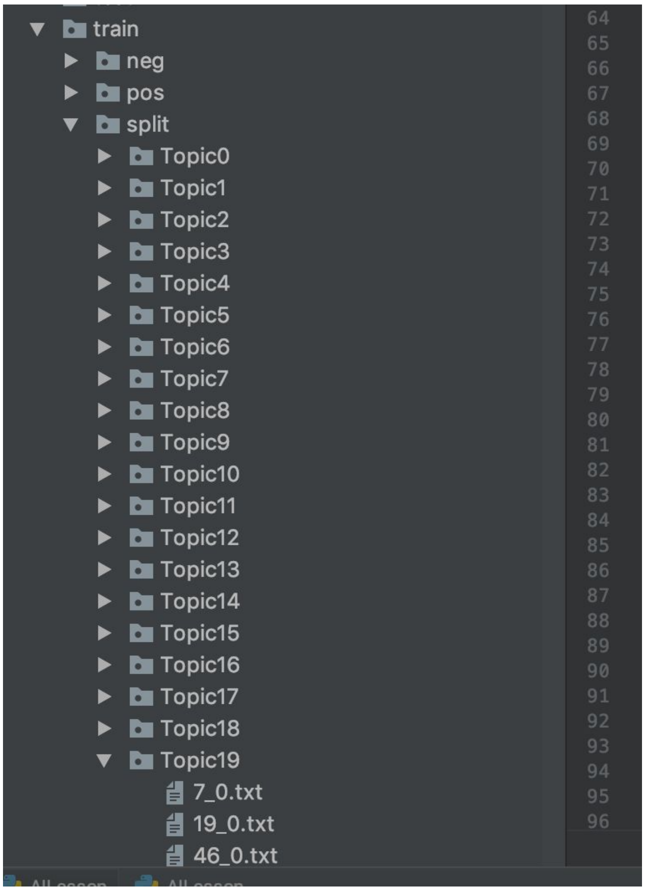
和之前比较，有一定减少。

OK，那么截至目前我们把没有分类的的50000条数据给他进行了分组，分为了 20个Topic，那么接下来，我们就可以用前几节课分类的方法对这些数据进行交叉验证。就类似最开始我们使用贝叶斯算法对20个邮件组进行分类一样。我们把目前已经分好的topic 分为2大组，train 和 test 组，开始我们的交叉验证。



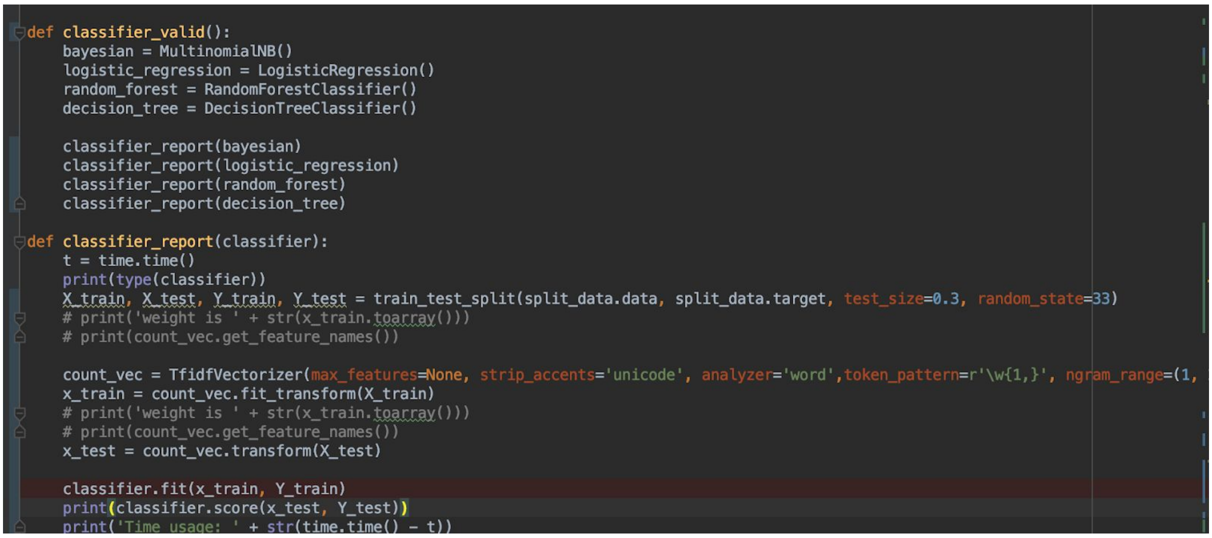
```
def split_unsup_data():
    tf, count_vec = load_data_vector()
    n_topics = 20
    lda = LatentDirichletAllocation(n_components=n_topics,
                                    max_iter=10,
                                    learning_method='batch',
                                    random_state=0,
                                    perp_tol=0.01,
                                    topic_word_prior=0.2,
                                    n_jobs=-1)
    lda.fit(tf)
    doc_topic_dist = lda.transform(tf)
    print(doc_topic_dist)
    topics = []
    for doc_topic_probs in doc_topic_dist:
        tmp_array = numpy.array(doc_topic_probs)
        bb = tmp_array.tolist()
        index = bb.index(max(bb))
        topics.append(index)
        print('#Topic is #' + str(index))

    for i in range(0, 20):
        path = MOVIE_PATH  + 'split/' + 'Topic' + str(i)
        folder = os.path.exists(path)
        if not folder:
            os.makedirs(path)

    for i in range(0, 50000):
        file_move_path = MOVIE_PATH  + 'split/' + 'Topic' + str(topics[i])
        file_origin_path = MOVIE_PATH + 'unsup/' + str(i) + '_0.txt'
        shutil.copy(file_origin_path, file_move_path)
```

我们在 train 目录下新建了一个文件夹叫 split，里面创建20个子文件夹 Topic0 - Topic19.
首先用 LDA 将所有的文本分为 20个 topic， transform 之后的矩阵 doc_topic_probs 包含了当前文本在该 topic 分类下的概率，我们选出概率最高的作为其文本的 topic。最终结果，我们在 train 文件夹下得到了20个 topic 的分组文件结构。



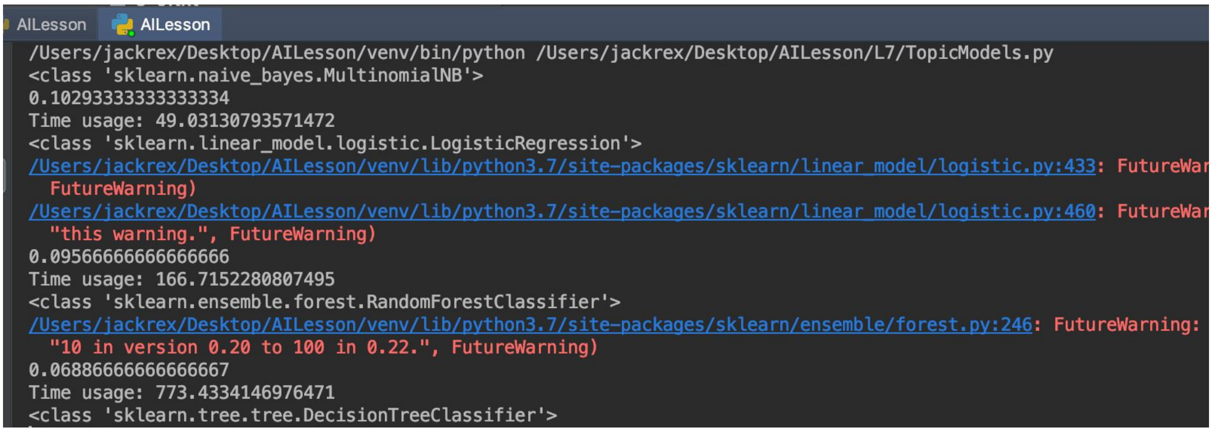
```
▼ train
    ▶ neg
    ▶ pos
    ▼ split
        ▶ Topic0
        ▶ Topic1
        ▶ Topic2
        ▶ Topic3
        ▶ Topic4
        ▶ Topic5
        ▶ Topic6
        ▶ Topic7
        ▶ Topic8
        ▶ Topic9
        ▶ Topic10
        ▶ Topic11
        ▶ Topic12
        ▶ Topic13
        ▶ Topic14
        ▶ Topic15
        ▶ Topic16
        ▶ Topic17
        ▶ Topic18
        ▼ Topic19
            7_0.txt
            19_0.txt
            46_0.txt
```

接下来我们使用第一次作业最简单的贝叶斯对20个 topic开始进行交叉验证。
编写代码如下



```python
def classifier_valid():
    bayesian = MultinomialNB()
    logistic_regression = LogisticRegression()
    random_forest = RandomForestClassifier()
    decision_tree = DecisionTreeClassifier()

    classifier_report(bayesian)
    classifier_report(logistic_regression)
    classifier_report(random_forest)
    classifier_report(decision_tree)

def classifier_report(classifier):
    t = time.time()
    print(type(classifier))
    X_train, X_test, Y_train, Y_test = train_test_split(split_data.data, split_data.target, test_size=0.3, random_state=33)
    # print('weight is ' + str(x_train.toarray()))
    # print(count_vec.get_feature_names())

    count_vec = TfidfVectorizer(max_features=None, strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}', ngram_range=(1,
    x_train = count_vec.fit_transform(X_train)
    # print('weight is ' + str(x_train.toarray()))
    # print(count_vec.get_feature_names())
    x_test = count_vec.transform(X_test)

    classifier.fit(x_train, Y_train)
    print(classifier.score(x_test, Y_test))
    print('Time usage: ' + str(time.time() - t))
```

我们定义 splite_data = datasets.load_files(MOVIE_PATH + 'split/', 'Topics', None, True, True, None, 'strict', 42), 使用前几节课学习的朴素贝叶斯 、逻辑回归, SVM, 决策树进行 7 : 3 的交叉验证。



```
/Users/jackrex/Desktop/AILesson/venv/bin/python /Users/jackrex/Desktop/AILesson/L7/TopicModels.py
<class 'sklearn.naive_bayes.MultinomialNB'>
0.10293333333333334
Time usage: 49.03130793571472
<class 'sklearn.linear_model.logistic.LogisticRegression'>
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWar
  FutureWarning)
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:460: FutureWar
  "this warning.", FutureWarning)
0.09566666666666666
Time usage: 166.7152280807495
<class 'sklearn.ensemble.forest.RandomForestClassifier'>
/Users/jackrex/Desktop/AILesson/venv/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning:
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
0.06886666666666667
Time usage: 773.4334146976471
<class 'sklearn.tree.tree.DecisionTreeClassifier'>
```

最后我们能看到结果其实并不是非常高。这里一部分原因是本身用分类器分类就有误差, 一部分原因是 topic models 的分组也存在分的不够纯的原因。而且我们定义使用概率最高作为该文本的 topic, 但现实生活中一件事情更大概率可能的是需要超过多个标签的来定义一个事务。所以后续的改进方法可以使用前3个或者5个标签 Topic 定义一篇文档会更加合适。

总结

通过 Topic model LDA 对无序数据的练习了解到了无监督学习的整个过程。更加深入的认识到了不同数据不同问题在最初解决方案上的最优选择是无比的重要。

另外除了推断出这些主题, LDA还可以推断每篇文章在主题上的分布。知道这些分布有什么用呢？

聚类, 将文章和多个类簇（主题）关联。聚类对整理和总结文章集合很有帮助。在主题上的分布提供了一个文章的简洁总结。
特征生成, LDA可以生成特征供其他机器学习算法使用。也就是先用Topic model 对无从下手的数据进行分类和特征提取, 再用这些特征用在像逻辑回归或者决策树这样的算法中用于预测任务。

实际用途, 比如推荐系统, 通过主题的近似度来进行相关推荐, 这个应用非常广泛。对海量数据继续挖掘等等, 快速了解数据有效信息。