课程名称：当代人工智能	年级：2020	上机实践成绩：
指导教师：李翔	姓名：林以任	学号：10205501411
上机实践名称：实验一 文本分类		上机实践日期：3.17

实验流程

- 该任务是一个 10 分类任务
- 自行划分训练集和验证集（使用 sklearn.model_selection 里面的 train_test_split 对训练集分割成训练集和验证集）
- 将文本映射成向量（使用 sklearn.feature_extraction.text 里面的 TfidfVectorizer 方法）
- 训练分类器（使用 svm 或者 Logistic 回归，进行最优超参数搜索）

数据预处理

由于 train_data.txt 是一个字典类型的文件，而 test.txt 是以每一行的第一个逗号和换行符进行分隔的文件。在这里我不知道如何调包将其转化为可以分析的数据，然后进一步向量化。所以在这里我先使用两个函数将其转化为 csv 文件。可以直接查看代码文件里面的 `txt_to_csv_train` 和 `txt_to_csv_test` 函数，在这里就不过多介绍。

之后我们让 X 存有文本值，而 y 存有标签值。我们先通过 `X = X.values.tolist()` 让 pd DataFrame 对象转化为一个列表，后面发现他是一个二维列表，就使用循环迭代的方式将其转化为一维列表。

之后我们创建一个 TfidfVectorizer 对象，设置停用词为英文，并且将其值赋给 tv：`tv = TfidfVectorizer(stop_words='english')` 然后就可以用我们刚刚得到的一维列表喂给这个对象，将其转化为向量。`X_fit = tv.fit_transform(list1).toarray()` 让我们得到了向量化后的矩阵 X_fit.

模型选择与机器学习

模型选择：支持向量机

支持向量机是一个分类模型，学习算法是求解凸二次规划的最优化算法。线性支持向量机的学习问题是如下的凸二次规划问题（原始问题）：

$$
\begin{aligned}
&\underset{w,b,\xi}{min}\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\
&s.t.\quad y_i(w\cdot x_i + b) \geq 1 - \xi_i \\
&\xi_i \geq 0
\end{aligned}
$$

其中分离超平面是

$$
w^* \cdot x + b^* = 0
$$

分类决策函数为

$$
f(x) = sign(w^* \cdot x + b^*)
$$

其中的 C 称为惩罚参数，一般由应用问题决定，C 值大的时候对误分类的惩罚增大，小的时候对误分类的惩罚减小。我们需要使 $\frac{1}{2}\|w\|^2$ 尽量小即间隔尽量大，同时让误分类点的个数尽量小，C 是调和两者的系数。我们应该让 C 值尽量小一些，允许容错，让模型的泛化能力更强一些。

我们对于 SVC，设置 kernel 值为线性核，然后参数搜索网格设置惩罚参数五个值，分别为 [0.5, 1, 2, 3, 4]. 并且设置四折交叉验证。设置线性核的原因是因为线性核的运行速度相对较快，而本身这个数据集的特征量庞大，所以选择运行效率相对较高的线性核。参数搜索完成之后，我们发现 C=2 是最优的值，代回到模型中进行训练。

```
grid = GridSearchCV(SVC(kernel='linear'),
 param_grid={"C":[0.5, 1, 2, 3, 4]}, cv=4)
print('start selecting...')
grid.fit(X_fit, y)
print("The best parameters are %s with a score of %0.2f"
 %(grid.best_params_, grid.best_score_))
```

```
4  grid = GridSearchCV(SVC(kernel='linear'), param_grid={"C":[0.5
   print('start selecting...')
   grid.fit(X_fit, y)
   print("The best parameters are %s with a score of %0.2f" %(gri

   start selecting...
   The best parameters are {'C': 2} with a score of 0.89
```

我们首先将训练集根据 train_test_split 方法分割成训练集和验证集，比例为 80:20. 采用随机分割的方式，重复多次，每次都随机选取不同的训练集和测试集，最后将多次的结果平均。它相比于 KFold，计算量较小，比较适用于这种样本量较大，数据特征较大的情况。

在这里，我们记录模型训练的时间，以及通过调用 model.score() 方法输出模型的分数。

```
# 数据集划分，进行模型的训练以及预测
X_train, X_val, y_train, y_val = train_test_split(X_fit, y, test_size=0.2,
random_state=42)
model = SVC(kernel='linear', C=2.0, random_state=42)
print('this model is running')
starttime = datetime.datetime.now()
model.fit(X_train, y_train)
endtime = datetime.datetime.now()
print('this model finishes running, running time:', (endtime -
starttime).seconds, 'seconds.')
score = model.score(X_val, y_val)
print("Model score:", score)
```

我们可以得出，如果给 svc 如上的参数，跑出来的模型分数为 0.958125.

```
(8000, 29697)
this model is running
this model finishes running, running time: 541 seconds.
Model score: 0.958125
```

模型选择：逻辑斯谛回归

逻辑回归是一种广义的线性回归分析模型，是一种预测分析。虽然它名字里带回归，但实际上逻辑回归是一种分类学习方法。它不是仅预测出 “类别”， 而是可以得到近似概率预测，这对于许多需要利用概率辅助决策的任务很有用。普遍应用于预测一个实例是否属于一个特定类别的概率，比如一封 email 是垃圾邮件的概率是多少。 因变量可以是二分类的，也可以是多分类的。因为结果是概率的，除了分类外还可以做 ranking model。LR 的应用场景很多，如点击率预测（CTR）、天气预测、一些电商的购物搭配推荐、一些电商的搜索排序基线等。

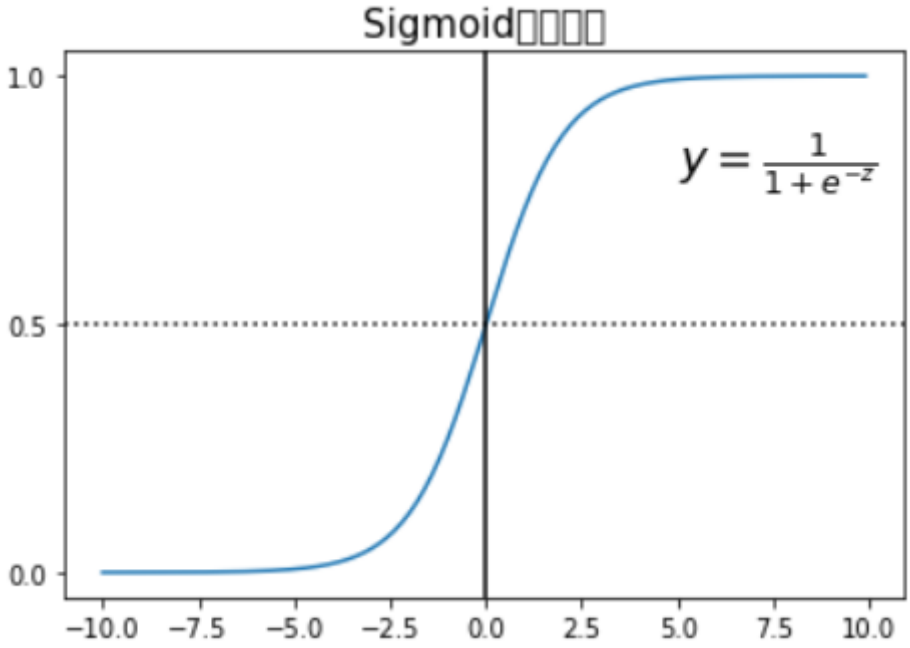
逻辑斯谛分布的分布函数是一种 “Sigmoid” 函数，呈现 *S* 型曲线，它将 ~z 值转化为一个接近 0 或 1 的 ~y 值。 对数几率回归公式如下：

$$\sim y = g(z) = \frac{1}{1+e^{-z}}, \sim z = w^T x + b$$

其中，$\sim y = \frac{1}{1+e^{-z}}$，被称做 **Sigmoid** 函数。

Logistic Regression 算法是将线性函数的结果映射到了 Sigmoid 函数中，即 $\sim y = \frac{1}{1+e^{(-w^T x+b)}}$

下图绘制了 Sigmoid 函数形状，如图所示，sigmoid 函数输出值范围在（0，1）之间，即代表了数据属于某一类别的概率，0.5 是作为判别的临界值。



使用 Sigmoid 函数的好处在于它能够将线性回归的结果转换为概率值，更直观地表示样本属于某一类的概率。这样，逻辑斯谛回归就可以解决分类问题，而不仅仅是回归问题。

与支持向量机一样，我们使用 GridSearchCV 进行参数选择。我们发现最优的超参数 C=10。

```
#%%
grid = GridSearchCV(LogisticRegression(max_iter=300),
 param_grid={"C":[1, 2, 5, 10], "penalty":['l2']}, cv=4)
print('start selecting...')
grid.fit(X_fit, y)
print("The best parameters are %s with a score of %0.2f"
 %(grid.best_params_, grid.best_score_))
```

```
lecting...
: parameters are {'C': 10, 'penalty': 'l2'} with a score of 0.90
```

进行模型的训练，发现运行时间远小于支持向量机，并且模型分数与支持向量机相差较小。

```
# 数据集划分，进行模型的训练以及预测
X_train, X_val, y_train, y_val = train_test_split(X_fit, y, te
model = LogisticRegression(penalty='l2', C=10, random_state=42,
print('this model is running')
starttime = datetime.datetime.now()
model.fit(X_train, y_train)
endtime = datetime.datetime.now()
print('this model finishes running, running time:', (endtime -
score = model.score(X_val, y_val)
print("Model score:", score)
```

```
this model is running
this model finishes running, running time: 41 seconds.
Model score: 0.95125
```

比较两种方法的输出

```
使用 SVM 和逻辑斯谛回归训练出来的数据有： 104 行的不同。
459, 6

1267, 5
```

在我训练的模型中，两个模型输出的差异有 104 行，占整个测试集的 5%。

如果比较一下两个机器学习方法的差异的话，从原理角度来讲，逻辑斯谛回归是把线性回归所得到的值通过 Sigmoid 函数映射到 [0, 1] 这个区间中，然后使用概率来进行判别。但是由于我们的数据在通过 TfidfVectorizer 向量化之后，有 29697 个特征，接近三万个。而这三万个特征如果全部选取，容易出现过拟合的问题，因为其中有严重的多重共线性。所以这也是逻辑斯谛回归在模型所得的分数上面较低的一个原因。而且在 Sigmoid 函数映射之后，值无限接近 0 或者 1 的时候，参数更新的梯度会明显下降，导致学习的效率变得很缓慢。不过 L2 正则化让我们降低了模型的特征数，减少了过拟合的程度。

而支持向量机在高维空间的表现相对较好，不仅是因为感知器权重向量的更新会用到所有的数据，它的分类决策函数也没有逻辑斯谛回归那样梯度下降得那么快但是后期乏力的问题，核函数也可以将高维的数据映射到低维空间中。所以支持向量机的模型分数较高。

回顾与总结

在做这个实验的时候，我踩了不少坑，总结如下。

首先，参考了网上的博客，使用 sklearn.datasets 里面的 load_files，希望把数据集文件转化为可以直接进行机器学习的数据，但是发现不行。于是只能先暴力采用转化为 csv 的方法然后分离特征值与训练数据，再对训练数据进行向量化。

在向量化的时候，使用了 nltk 包，不过总是报告如下的错误，就算使用 nltk.download() 也没用。于是转而使用 TfidfVectorizer，至少它能用啊，然后用了一下发现还挺好用的，直接把列表里面的句子给我分好了。

```
LookupError:
**********************************************************************
  Resource punkt not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt')

  For more information see: https://www.nltk.org/data.html

  Attempted to load tokenizers/punkt/english.pickle

  Searched in:
```

使用 TfidfVectorizer 的时候，感谢这篇文章给出的指南与例子分析：[朴素贝叶斯分类-实战篇-如何进行文本分类 - 简书 (jianshu.com)]

在参数搜索方面，GridSearchCV 在搜索 SVM 的超参数的时候，搜索了一个晚上。对于 8000*30000 这样较大数据的模型来说，进行超参数搜索是非常浪费时间的。这让我感到深有体会。

此外，为了让花了大力气训练好的模型可以直接用于预测数据，我使用了如下代码来把模型保存到本地，以便直接进行预测数据时候的使用。

```
# 保存该 model, save the model as a pickle object in Python

with open('text_classifier', 'wb') as picklefile:
    pickle.dump(model, picklefile)

# 从本地加载训练好的模型
with open('text_classifier', 'rb') as training_model:
    model = pickle.load(training_model)
```

回头在文件资源管理器里面看了一下，大小有 900 多MB，有些吓人。因为逻辑斯谛回归的模型较为简单，所以就没有在本地保存，反正训练一个模型只要几十秒。

这是我所做的第一个文本分类的任务，使用了两种机器学习的方法并且建模与进行了对比。在这中间我踩了不少坑，也积累了不少的经验，掌握了不少的新技能，也从底层深入地更加了解和重新学习了之前统计与机器学习的内容。这次实验让我收获颇丰。