华北水利水电大学

自然语言处理

实验报告

20 22 ——20 23 学年

第 一 学期

实验报告序号：实验报告（三）

实验名称：文本分类与聚类的实现

学生专业班级：

学生姓名：

学　　号：

专　　业：人工智能

一、实验目的

1.掌握文本分类和聚类的基本原理。

2.能够熟练地运用分词、文本向量化、分类和聚类技术，对集合中的文本进行分类和聚类。

二、实验内容

1.基于给定“从政”、“国际”、“经济”、“体育”四个领域的 json 文件，划分训练集和数据集，基于训练集训练文本分类器，并在测试集上进行准确率的测试。

2.对以上数据进行文本聚类，并对聚类结果进行可视化。

三、实验要求

(1)使用两种不同的方法实现文本的向量化。

(2)文本分类中，训练集和测试集的比例设置为 8:2 或 9:1，采用 K 折-交叉验证方法划分数据集，并计算分类模型的平均准确率。

(3)文本分类中，采用朴素贝叶斯、KNN、SVM、决策树等分类方法中的任意两种，进行分类模型的训练和测试，并比较它们的优劣和特点。

(4)文本聚类中，采用 DBSCAN、层次聚类和 KMeans 等聚类方法中的任意两种，进行文本聚类，并比较它们的优劣和特点。

(5)对文本聚类的结果进行可视化。

(6)实验报告撰写：以分析为主，写问题分析、结果分类、所使用的算法的思想和效果比较等，画流程图，不要放太多代码，关键代码或流程应有文字说明或解释。

四、文本分类

4.1 采用文本分类的实现流程分析。

采用文本分类的实现流程主要包括文本读取、文本向量化、利用 K 折交叉验证划分数据集和测试集、训练模型和使用模型进行预测，最后做出预测过程的折线图，能够更好的反映预测过程的正确率变化。其流程图如下图 1 所示。

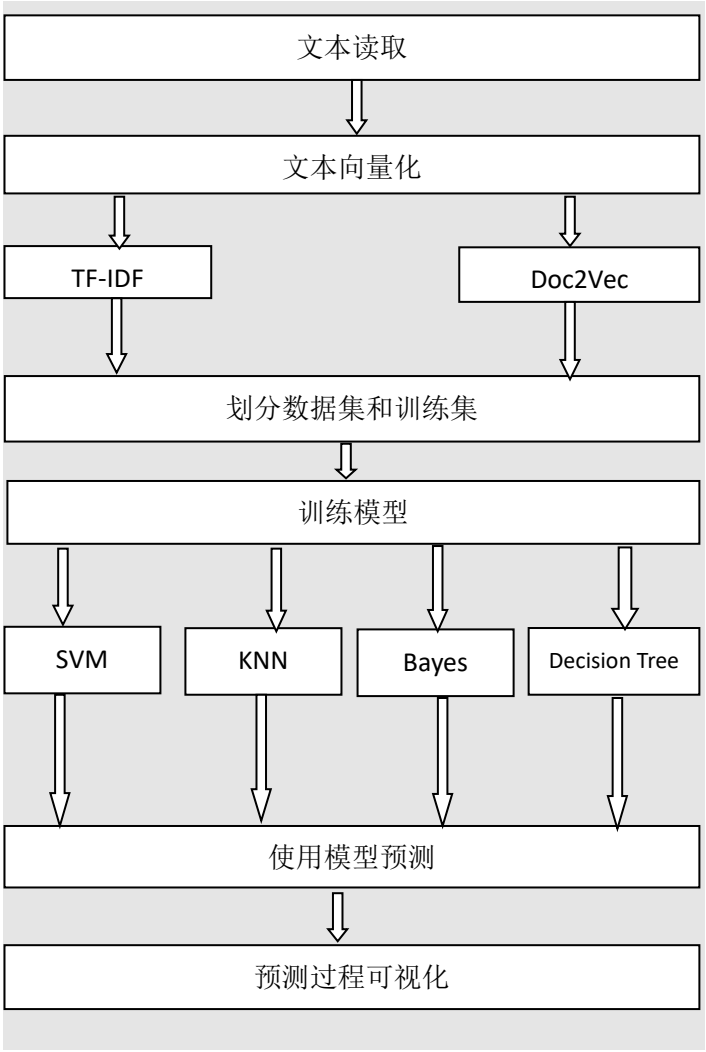

图 1 文本分类流程图

4.2 文本分类的实现步骤、关键代码及解释分析。

步骤 1：文本读取。对文本进行分类，最基本的一步就是将文本从文档中读取到编程语言的数据结构中，这样才能进行正常的处理。这里的数据集成在 json 文档中，因此需要 json 这个库来对其进行处理，然后将其中的文本信息存储在列表中。其关键核心代码和注释如下：

```
def read_json(path):
    text = []
    f_read = open(path, 'r', encoding='utf8', errors='ignore')
    for line in f_read:
        line = line.replace('\\u0009', '').replace('\\n', '')
        obj = json.loads(line)
        sent = obj['contentClean']
        text.append(sent)
    return text
```

该函数的功能是处理 json 文档，传的参数是 json 文档的路径（绝对路径和相对路径均可）。该函数首先定义一个空的文档列表，这个文档列表是用来存放每次读取的文本数据，之后读取函数的参数文件内容，读取（解码）的格式为 Utf-8 模式并忽略其中可能造成的错误。利用 for 循环对 json 表进行遍历，将文本内容赋值给 sent，将 sent 当作列表的一个元素尾插到 text 列表中。最后返回该列表就得到最后的文档，其中每一个列表元素对应一条文本段。

步骤 2：文本向量化。文本向量化有两种方法，分别是利用 TF-IDF 算法和 Doc2vec 算法。TF-IDF 算法在实验 1 中已经做了详细的解释和按照定义从 0 开始的实现。但是为了使用方便，调用 Sklearn 中的 TfidfVectorizer 函数更为方便和快捷。利用 TF-IDF 算法进行文本向量化的核心代码如下所示。

```
def __init__(self):
    self.corpus = read_json('从政.json') + read_json('体育.json') + read_json('国际.json') + read_json('经济.json')
    self.labels = [0] * 500 + [1] * 500 + [2] * 500 + [3] * 500
    self.corpus, self.labels = shuffle_data(self.corpus, self.labels)
    self.vectorizer = TfidfVectorizer(stop_words=stop_words(), max_features=1000)
    self.X = self.vectorizer.fit_transform(self.corpus).toarray()
```

在代码中，是通过类的定义来实现 TF-IDF 算法和 Doc2Vec 算法以及他们分类和聚类操作的实现。上述代码是在 TF-IDF 类中初始化定义的类属性。在该初始化中，首先读取了全部的 json 文件并将其返回的结果连接成为了一个长列表 self.corpus，该列表就是所有的数据集。再获取数据集之后要定义每一个类的表示，不妨设“从政”的标签为 0，“体育”的标签为 1，“国际”的标签为 2，“经济”的标签为 3。由于每一个类型的数据分别有 500 条，因此对应的标签乘 500 并连接就组成了标签数据。之后需要打乱才能做交叉验证，但是打乱的同时有需要保证标签和数据相互对应，因此定义了一个全局函数 shuffle_data，该函数的原理是打乱下标，然后按照打乱后的下标匹配对应的数据和标签数据。其代码如下：

```
def shuffle_data(x_train, y_train):
    c = list(zip(x_train, y_train))
    shuffle(c)
    x_train, y_train = zip(*c)  # 打乱顺序
    x_train = list(x_train)
    y_train = list(y_train)  # 打乱结果转换成列表
    return x_train, y_train
```

函数传入的参数是数据和标签，在函数执行的过程中，首先会两个数据打包，之

后打乱，之后将打包的列表的所有元素随机排序，然后解开这个“包”，其返回的数据类型是元组类型，因此还要将其转化为列表类型，之后返回就能得到乱序后的结果。将乱序后的数据（corpus）通过 TfidfVectorizer().fit_transform()方法进行向量化，向量化后的结果是一个列表，但是再后面训练的时候，只能读取 np.arry 类型的数据，因此还需要加上一个.toarry()方法使其转化为 np.arry 类型的数据。

步骤 3：划分训练集和测试集。使用 5 折交叉验证（训练集：测试集==4：1）的方法来划分数据集。其核心代码和注释如下：

```
self.X_train = self.X[:int(self.X.shape[0] * 0.8)]  # 训练集 5 折交叉（训练集：测试集 = 4：1）
        self.Y_train = self.Y[:int(self.Y.shape[0] * 0.8)]
        self.X_test = self.X[int(self.X.shape[0] * 0.8):]  # 测试集 5 折交叉（训练集：测试集 = 4：1）
        self.Y_test = self.Y[int(self.Y.shape[0] * 0.8):]
```

由上述代码可以看出，训练集选取的是前 80%的数据，也就是 1600 条数据，测试集选取的是后 20%的数据，也就是 400 条数据。这里没有用到 train_test_split 函数的原因是、是尽量少用库，保证代码创新性。

步骤 4：训练模型。训练模型是该文本分类的最重要的一个步骤，在训练模型的时候可以采取多种机器学习算法，一般选取 SVM、Bayes、Decision Tree、KNN。在本实验中，为了得到最好的效果，我使用了上述 4 个模型，各个算法对应的代码如下所示：

4.1 SVM 进行模型训练代码

```
def TF_IDF_SVM(self):
    clf = svm.SVC(kernel='linear', probability=True)
    try:
        clf = joblib.load("SVM.m")
    except:
        print('开始训练模型.....')
        clf.fit(self.X_train, self.Y_train)
        joblib.dump(clf, "SVM.m")
        print('模型训练结束。')
```

首先声明这里并不是只是为了训练，这里原本是为了预测，因此出现了先加载模型的代码结构。当加载失败（没有 SVM.m 的模型）时，需要训练模型。训练模型利用 fit 函数来开始，前面的 clf 是定义的 SVM 算法及其里面的超参数。当模型训练结束后，将其保存下来（这里用到了 dump 函数），以便于测试的时候使用而不用重新训练模型。

4.2 Bayes 进行模型训练代码

```
def TF_IDF_Bayes(self):
    ber_model = BernoulliNB(alpha=0.001)
    try:
        ber_model = joblib.load("Bayes.m")
    except:
        print('开始训练模型.....')
        ber_model.fit(self.X_train, self.Y_train)
        joblib.dump(ber_model, "Bayes.m")
        print('模型训练结束。')
```

和 4.1 一样，这里并不是只是为了训练，这里原本是为了预测，因此出现了先加载模型的代码结构。当加载失败（没有 Bayes.m 的模型）时，需要训练模型。训练模型利用 fit 函数来开始，前面的 clf 是定义的 Bayes 算法及其里面的超参数。当模型训练结束后，将其保存下来（这里用到了 dump 函数），以便于测试的时候使用而不用重新训练模型。

4.3 KNN 进行模型训练代码

```
def TF_IDF_KNN(self):
    KNN = KNeighborsClassifier(n_neighbors=3)
    try:
        KNN = joblib.load("KNN.m")
    except:
        print('开始训练模型.....')
        KNN.fit(self.X_train, self.Y_train)
        joblib.dump(KNN, "KNN.m")
        print('模型训练结束。')
```

和 4.1 一样，这里并不是只是为了训练，这里原本是为了预测，因此出现了先加载模型的代码结构。当加载失败（没有 KNN.m 的模型）时，需要训练模型。训练模型利用 fit 函数来开始，前面的 clf 是定义的 KNN 算法及其里面的超参数。当模型训练结束后，将其保存下来（这里用到了 dump 函数），以便于测试的时候使用而不用重新训练模型。

4.4 Decision Tree 进行模型训练代码

```
def TF_IDF_Tree(self):
    clf = tree.DecisionTreeClassifier(criterion='entropy')
    try:
        clf = joblib.load("Tree.m")
    except:
        print('开始训练模型.....')
        clf.fit(self.X_train, self.Y_train)
```

```
        joblib.dump(clf, "Tree.m")
        print('模型训练结束。')
```

和 4.1 一样，这里并不是只是为了训练，这里原本是为了预测，因此出现了先加载模型的代码结构。当加载失败（没有 Tree.m 的模型）时，需要训练模型。训练模型利用 fit 函数来开始，前面的 clf 是定义的决策树算法及其里面的超参数。当模型训练结束后，将其保存下来（这里用到了 dump 函数），以便于测试的时候使用而不用重新训练模型。

上述代码重复性太高（相信老师您读完一定有这个感觉），因此利用字典传递和抽象，我定义了如下函数用来将 4.1 到 4.4 的功能全部集成在一个函数中。（本来想用装饰器，结果没有调通）。代码如下所示：

4.5 整合 KNN、SVM、Bayes、Decision Tree 的代码

```
def classify(self, model):
    model_list = {
        "SVM": svm.SVC(kernel='linear', probability=True),
        "Bayes": BernoulliNB(alpha=0.001),
        "KNN": KNeighborsClassifier(n_neighbors=3),
        "Tree": tree.DecisionTreeClassifier(criterion='entropy')
    }
    try:
        clf = joblib.load('models/TF_IDF/'+model + '1.m')
     except:
         print('开始训练模型.....')
         clf = model_list[model]
         clf.fit(self.X_train, self.Y_train)
         joblib.dump(clf, 'models/TF_IDF/'+ model + '1.m')
         print('模型训练结束。')
```

传入的参数为模型的名称。当函数在接收到名称并执行时，先调用现有模型，当 models 文件夹没有模型时，会在模型字典里面查询对应的算法和其超参数，并将其赋值给 clf。之后还是按照正常的程序训练模型并且保存。

步骤 5：使用模型预测。使用模型预测分为加载模型、模型预测、返回结果三个步骤。模型加载在步骤 4 代码中已经说过，就不再赘述。下面给出使用模型预测的核心代码以及必要解释。

```
print('开始预测.....')
correct = 0
Precision = []
for i in range(int(self.X.shape[0] * 0.8), self.X.shape[0]):
    mark = clf.predict(self.X[i:i + 1])
```

```
6.      if self.Y[i] == mark[0]:
7.          correct += 1
8.      Precision.append(correct / (i - 1600 + 1))
9. print('预测结束。利用%s 算法在%d 个样本中，一共有%d 个样本被预测正确
    ' % (model, int(self.X.shape[0] * 0.2), correct), end='\t')
10. print('Accuracy:%.2f%%' % (correct / 400 * 100))
```

代码实际上是 4.5 代码的后半部分，当加载模型（clf）结束后，利用 for 循环对测试集中的每一个数据（已经完成向量化）进行预测分类，用到的函数是 predict()函数，返回的结果是一个一维向量。为了统计正确率，我在它预测的时候对她预测正确的阳历进行一个计数，所以在第 2 行定义了一个计数器 correct，为了能在最后做出正确率的折线图，需要统计每一次的正确率，因此要在第 3 行定义一个 Precision 列表用于装该次预测的正确率。

步骤 6：结果可视化。为了能更好的看出模型的优劣，挑选出最优的模型，一般需要绘制精度曲线。精度曲线能反映在训练整个过程中，正确率的变化值。其核心代码如下所示。

```
1. num = [i for i in range(1, 401)]
2. plt.plot(num, Precision,c='red')
3. #plt.scatter(num,Precision, c='red')
4. plt.grid(True, linestyle='--', alpha=0.5)
5. plt.xlabel("累计测试样本个数", fontdict={'size': 16})
6. plt.ylabel("实时准确率", fontdict={'size': 16})
7. plt.title(model + 'Precision', fontdict={'size': 20})
8. plt.show()
```

在步骤 5 中已经获取了每一次的正确率，因此只需要把每一次预测的序号求出来就可以了。利用列表生成式生成一个 1 到 400 的列表并赋值给 num。以 num 为横坐标，Precision 为纵坐标，通过 matplotlib 工具包中的 pyplot 函数的 plot 函数就能画出折线图。4、5、6、7 三行分别为设置布局、横坐标名称、纵坐标名称和标题。

步骤 7：利用 Doc2vec 进行文本向量化。为了方便起见，这里还是调用了 gensim 的工具包来对文本进行处理进而实现文本向量化。其核心代码和注释如下：

```
1. def convert_data(self, data):
2.     train_text = []
3.     for i, sent in enumerate(data):
4.         # 改变成 Doc2vec 所需要的输入样本格式，
5.         # 由于 gensim 里 Doc2vec 模型需要的输入为固定格式，输入样本为：[句
    子，句子序号],这里需要
6.         tagged_doc = gensim.models.doc2vec.TaggedDocument(sent, tags=[i]
    )
```

```
        train_text.append(tagged_doc)
    d_model = Doc2Vec(train_text, min_count=5, windows=3, vector_size=1000, sample=0.001, nagetive=5)
    try:
        d_model = gensim.models.doc2vec.Doc2Vec.load("doc2vec_model")
    except:
        d_model.train(train_text, total_examples=d_model.corpus_count, epochs=10)
        d_model.save("doc2vec_model")  # 保存模型
        d_model = gensim.models.doc2vec.Doc2Vec.load("doc2vec_model")
    return d_model.docvecs.vectors_docs,d_model
```

上述代码首先定义了一个空列表用于存放训练的数据。在 for 循环中 enumerate()函数用于将一个可遍历的数据对象组合为一个索引序列，同时列出数据和数据下标，比如[‘1’，‘2’]经过 enumerate()作用后得到[(0，‘1’),(1，‘2’)]利用 TaggedDocument()将 sent 向量化，并将向量化后的结果尾插到 train_text 列表里。之后利用 Doc2Vec 方法来训练模型，并将结果保存。最后返回向量化后的结果和模型。

步骤 8：划分训练集和测试集。使用 5 折交叉验证（训练集：测试集==4：1）的方法来划分数据集。其核心代码和注释如下：

```
self.X_train, self.d_model = self.convert_data(self.corpus)
self.X_train = self.X_train[:1600]
self.Y_train = self.labels[:1600]
self.X_test = self.corpus[1600:]
self.Y_test = self.labels[1600:]
```

由上述代码可以看出，训练集选取的是前 1600 条数据，因为总数是 2000 条。测试集选取后 400 条数据，也就是后 20%的数据。这里也没有用到 train_test_split 函数。原因同步骤 3。

步骤 9：训练模型。训练模型是文本分类的最重要的一个步骤，在训练模型的时候可以采取多种机器学习算法，一般选取 SVM、Bayes、Decision Tree、KNN。在本实验中，为了得到最好的效果，我使用了上述 4 个模型并向 4.5 那样封装后的代码如下所示：

```
def classify(self, model):
    model_list = {
        "SVM": svm.SVC(kernel='linear', probability=True),
        "Bayes": BernoulliNB(alpha=0.001),
```

```
        "KNN": KNeighborsClassifier(n_neighbors=3),
        "Tree": tree.DecisionTreeClassifier(criterion='entropy')
    }
    try:
        clf = joblib.load('models/Doc2vec/' + model + '2.m')
     except:
        print('开始训练模型.....')
        clf = model_list[model]
        clf.fit(self.X_train, self.Y_train)
        joblib.dump(clf, 'models/Doc2vec/'+model + '2.m')
        print('模型训练结束。')
```

传入的参数为模型的名称。当函数在接收到名称并执行时，先调用现有模型，当 models 文件夹没有模型时，会在模型字典里面查询对应的算法和其超参数，并将其赋值给 clf。之后还是按照正常的程序训练模型并且保存。

步骤 10：基于 Doc2vec 的预测和可视化的代码和过程同 TF-IDF,步骤同步骤 5 和步骤 6.

4.3 结果及分析

实验结果：

结果 1：利用 TF-IDF 结合 SVM 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 2 所示。

```
开始预测........
预测结果为:  [3, 1, 3, 0, 0, 3, 3, 0, 1, 0, 0, 3, 3, 3, 3, 3, 0, 1, 1, 1, 3, 2, 2, 1, 3,
预测结束。利用SVM算法在400个样本中，一共有314个样本被预测正确    Accuracy:78.50%

Process finished with exit code 0
```

图 2 TF-IDF 和 SVM 算法预测结果

结果 2：利用 TF-IDF 结合 SVM 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 3 所

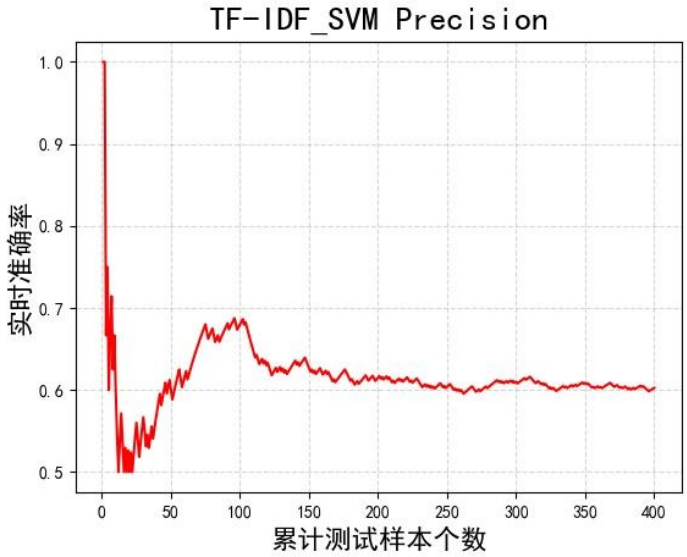

图 3 TF-IDF 和 SVM 算法预测精确度图

结果 3：利用 TF-IDF 结合 KNN 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 4 所示

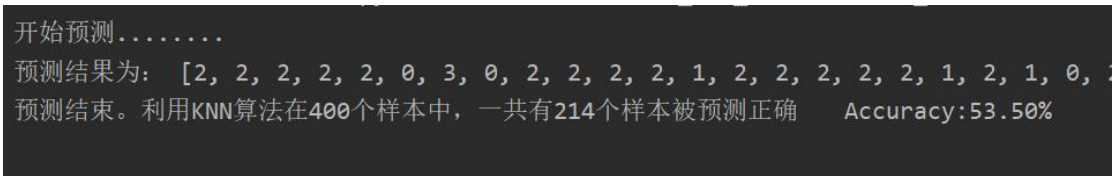

图 4 TF-IDF 和 KNN 算法预测结果

结果 4：利用 TF-IDF 结合 KNN 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 5 所示

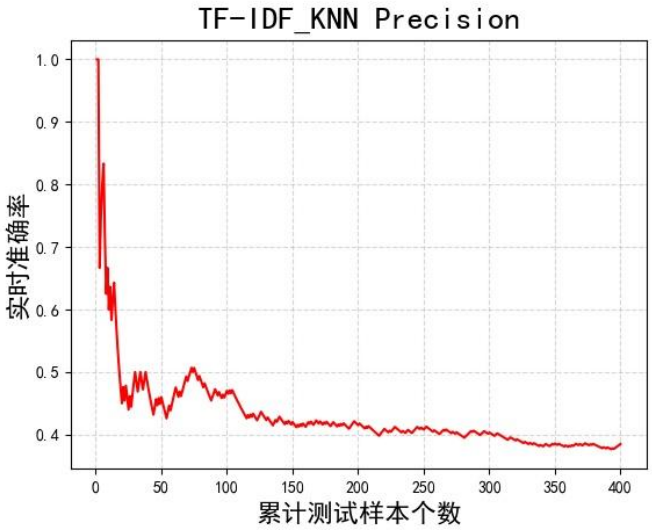

图 5 TF-IDF 和 KNN 算法预测精确度图

结果 5：利用 TF-IDF 结合 Bayes 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 6 所示

```
开始预测.....
预测结果为:  [3, 1, 0, 0, 2, 0, 1, 3, 1, 1, 1, 2, 1, 0, 3, 1, 1, 1, 1, 2, 1, 1, 3,
预测结束。利用Bayes算法在400个样本中，一共有304个样本被预测正确  Accuracy:76.00%
```

图 6 TF-IDF 和 Bayes 算法预测结果

结果 6：利用 TF-IDF 结合 Bayes 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 7 所示

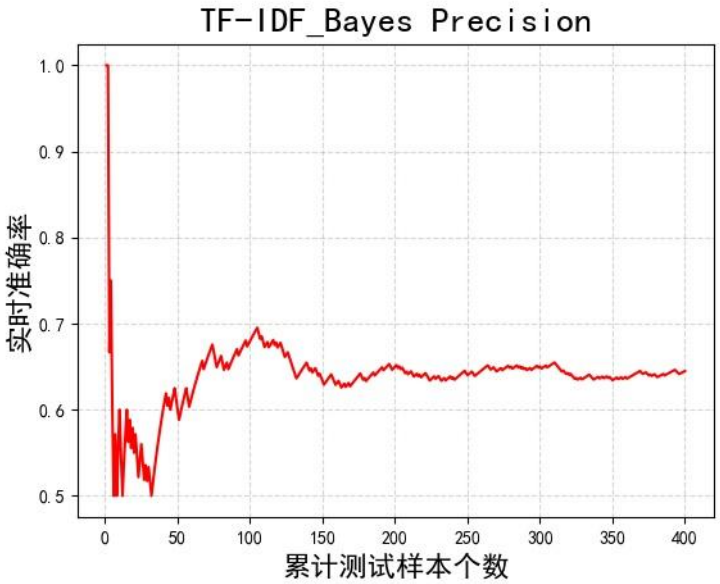

图 7 TF-IDF 和 Bayes 算法预测精确度图

结果 7：利用 TF-IDF 结合 Decision Tree 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 8 所示

```
开始预测.....
预测结果为:  [3, 1, 2, 0, 3, 1, 0, 0, 0, 1, 3, 3, 1, 1, 0, 1, 1, 0, 1, 2, 1, 1, 1,
预测结束。利用Tree算法在400个样本中，一共有324个样本被预测正确   Accuracy:81.00%

Process finished with exit code 0
```

图 8 TF-IDF 和 Decision Tree 算法预测结果

结果 8：利用 TF-IDF 结合 Decision Tree 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 9 所示

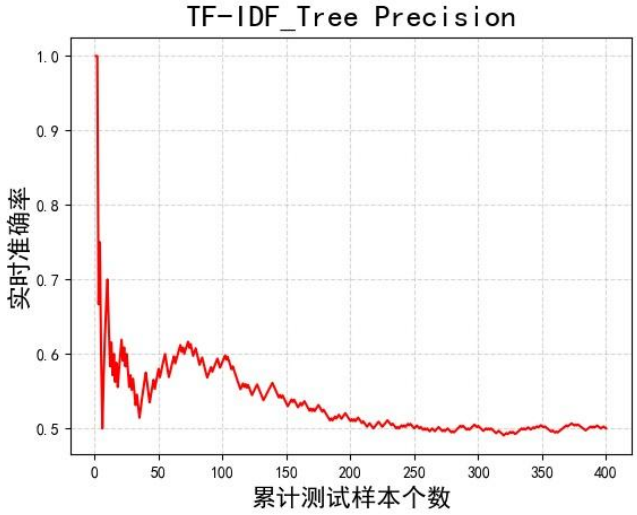

图 9 TF-IDF 和 Decision Tree 算法预测精确度图

结果 9：利用 Doc2vec 结合 SVM 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 10 所示

```
开始预测.........
预测结果为： [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
预测结束。利用SVM算法在400个样本中，一共有94个样本被预测正确      Accuracy:23.50%
```

图 10 Doc2vec 和 SVM 算法预测结果

结果 10：利用 Doc2vec 结合 SVM 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 11 所示

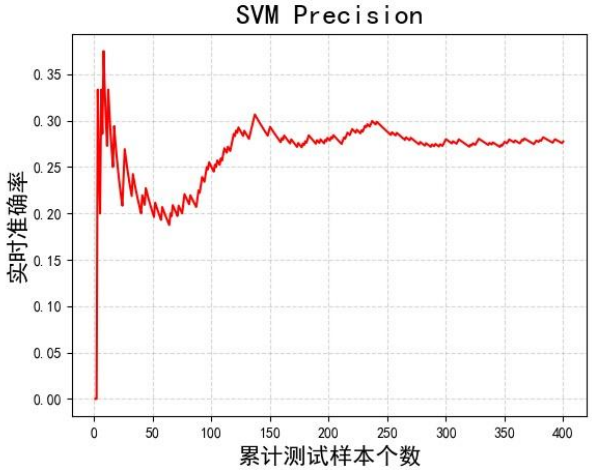

图 11 Doc2vec 和 SVM 算法预测精确度图

结果 11：利用 Doc2vec 结合 KNN 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 12 所示

```
开始预测........
预测结果为： [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
预测结束。利用KNN算法在400个样本中，一共有111个样本被预测正确    Accuracy:27.75%
开始预测........
```

图 12 Doc2vec 和 KNN 算法预测结果

结果 12：利用 Doc2vec 结合 KNN 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 13 所示

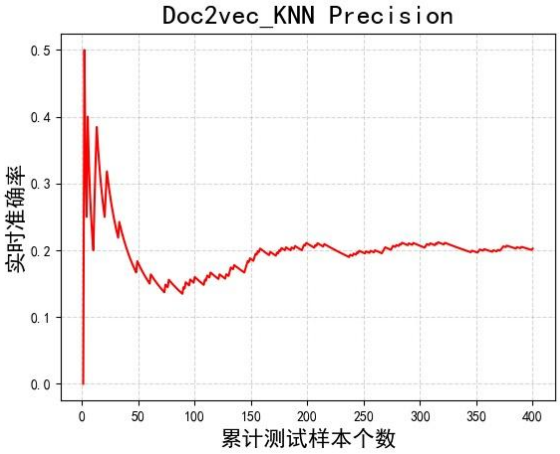

图 13 Doc2vec 和 KNN 算法预测精确度图

结果 13：利用 Doc2vec 结合 Bayes 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 14 所示

```
预测结果为： [0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
预测结束。利用Bayes算法在400个样本中，一共有83个样本被预测正确    Accuracy:20.75%
```

图 14 Doc2vec 和 Bayes 算法预测结果

结果 14：利用 Doc2vec 结合 Bayes 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 15 所示

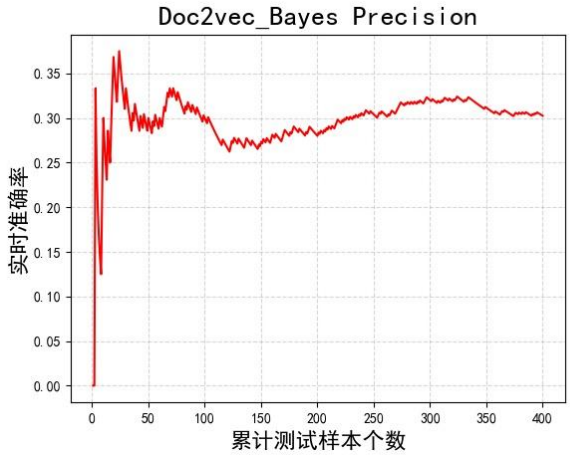

图 15 Doc2vec 和 Bayes 算法预测精确度图

结果 15：利用 Doc2vec 结合 Decision Tree 算法训练模型，并加载训练后的模型进行预测得到的结果如下图 16 所示

```
开始预测.........
预测结果为： [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
预测结束。利用Tree算法在400个样本中，一共有100个样本被预测正确   Accuracy:25.00%
```

图 16 Doc2vec 和 Decision Tree 算法预测结果

结果 16：利用 Doc2vec 结合 Decision Tree 算法训练模型，并加载训练后的模型进行预测得到的结果预测精确度图如下图 17 所示

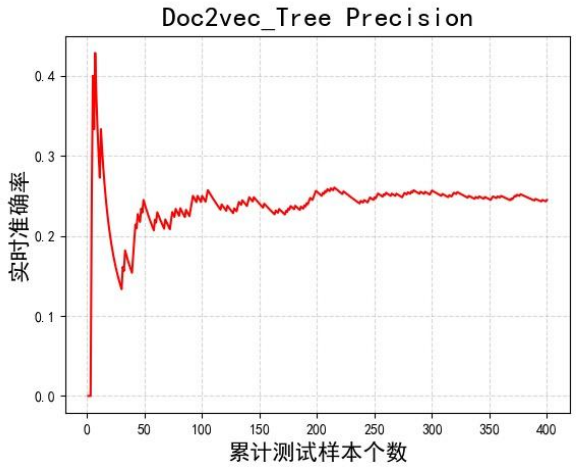

图 17 Doc2vec 和 Decision Tree 算法预测精确度图

结果 5：

结果表格样例：

方法	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	平均确率
Tfidf+SVM	78.00%	79.00%	78.50%	77.75%	77.75%	80.25%	81.00%	77.75%	77.00%	79.50%	78.65
Tfidf+KNN	69.50%	64.75%	63.25%	69.25%	65.25	65.50%	66.50%	67.00%	63.75%	68.50%	66.37
Tfidf+Bayes	75.75	77.25%	75.75%	75.50%	76.25	77.25%	73.50%	75.00%	74.50%	74.50%	75.57
Tfidf+Decision tree	81.00%	80.00%	82.5%	82.50%	77.00%	82.50%	81.25%	80.00%	78.00%	79.00%	80.40
Doc2vec+SVM	26.25%	27.25%	28.00%	25.25%	25.25%	22.00%	23.25%	24.75%	23.75%	26.00%	

Doc2vec+KNN	26.25%	27.25%	28.00%	25.25%	25.25%	22.00%	23.25%	24.75%	23.75%	26.00%	
Doc2vec+Bayes	27.25%	25.25%	28.75%	25.75%	26.25%	23.75%	21.25%	24.50%	25.75%	27,75%	
Doc2vec+Decision tree	26.25%	27.25%	28.00%	25.25%	25.25%	22.00%	23.25%	24.75%	23.75%	26.00%	

对各种实现方式的原理、效果及优劣进行对比分析。

所谓文本分类，就是将每个文本向量化，得到一定维度的向量，这个向量就相当于这篇文本通过训练模型，构建一个端到端的模型结构，即输入文本向量，就能得到该文本向量属于哪一个类别。在这中间，向量化和训练模型可以采取多种方法，在本实验报告中就采用了 TF-IDF 和 Doc2vec 两种方式进行文本向量化，采用 SVM、KNN、Bayes 和 Decision Tree 算法来训练模型。

TF-IDF 文本向量化原理：在实验 1 我们知道 TF-IDF 可以做词频统计，统计的是最重要的前 N 个词，事实上，他得到的数据是一个高维向量。该向量的列数是总文本集中出现的所有单词数，每列对应一个单词；行数是文章总数，每一行代表一篇文章。每一行的元素，就是这篇文章每个词语的 TF-IDF 值这样就得出了一个总文本的特征矩阵，每一行就代表这一篇文章的特征量。

Doc2vec 文本向量化原理：每一句话用唯一的向量来表示，用矩阵 D 的某一列来代表。每一个词也用唯一的向量来表示，用矩阵 W 的某一列来表示。每次从一句话中滑动采样固定长度的词，取其中一个词作预测词，其他的作为输入词。输入词对应的词向量 Word Vector 和本句话对应的句子向量 Paragraph vector 作为输入层的输入，将本句话的向量和本次采样的词向量相加求平均或者累加构成一个新的向量 X，进而使用这个向量 X 预测此次窗口内的预测词。

SVM 原理：SVM 的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。例如在本实验中，SVM 就是根据所有向量化好的文本数据以及他们的标签数据加以训练，最后得到若干个可以将其分开的超平面，再通过超参数的约束能够找到最满足要求的超平面，该平面能最大限度地将所有文本类别分隔开。

KNN 原理：当预测一个新的值 x 的时候，根据它距离最近的 K 个点是什么类别来判断 x 属于哪个类别。这个的原理其实和 Kmeans 算法很像，但是他们分属于分类算法和积累算法。

Bayes 原理：根据词向量计算该文本属于哪一个类别的概率来进行分类，可以用来处理多分类问题。在训练过程中，就是不断改变其权重的一个过程，找到最优的评价概率的模型。当该模型确定下来之后，所有进入该模型的文本向量就会按照这种概率估计的算法，得到分属于每一类别的概率，最后取最大概率。

Decision Tree 原理：用树形数据结构来展示决策规则和分类结果的模型，作为一种归纳学习算法，其重点是将看似无序、杂乱的已知数据，通过某种技术手段将它们转化成可以预测未知数据的树状模型，每一条从根结点（对最终分类结果贡献最大的属性）到叶子结点（最终分类结果）的路径都代表一条决策的规则。

对比分析：

由上述是研究过和图标中的数据可以看出，TF-IDF+Decision 无疑是其中最好的文本分类算法，在仅有 1600 条数据的训练集上，其预测的准确率还是达到了 80%以上。而且利用 TF-IDF 算法进行文本向量化最后通过若干分类模型进行文本分类得到的结果普遍都在 66%以上，但是反观 Doc2Vec 算法进行文本向量化，其不管用何种算法，得到的数据结果都接近 0.25，和盲猜的概率接近，造成这种结果的原因可能有两个，一个是样本数据过小，模型没有拟合；另外的原因是 Doc2Vec 算法进行文本向量化自身的原因。

五、文本分类

4.1 采用文本聚类的实现流程分析。

补充内容

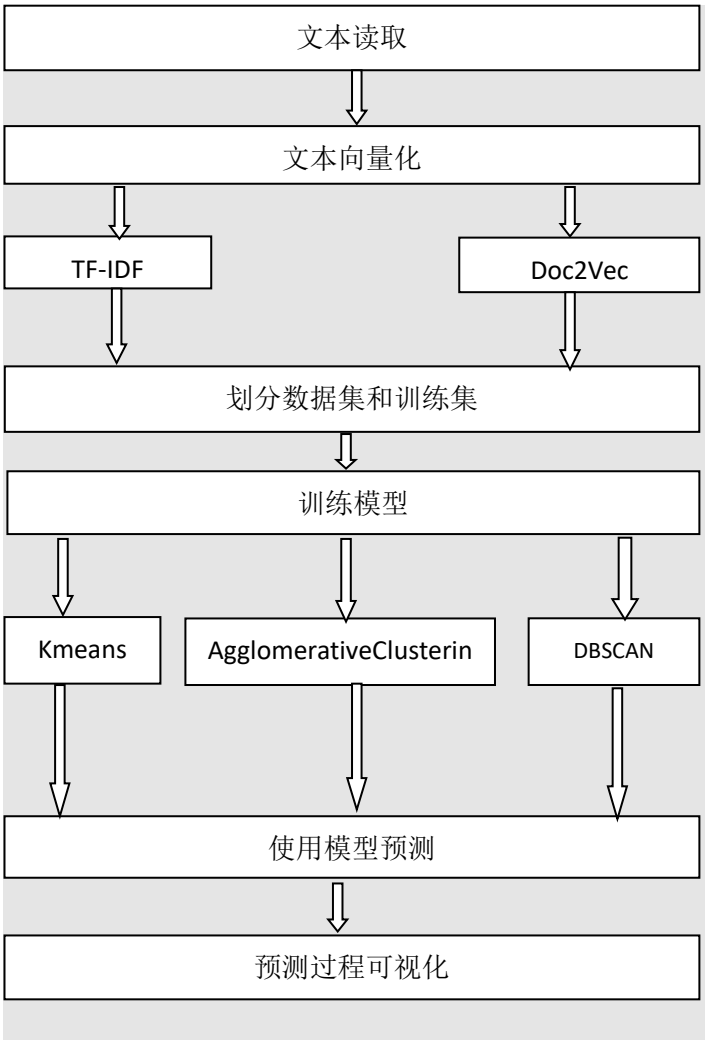

图 18 文本聚类流程图

4.2 文本聚类实现步骤、关键代码及解释分析。

补充内容

步骤 1：文本读取。其代码和代码关键描述见文本分类步骤 1。

步骤 2：文本向量化。其代码和代码关键描述见文本分类步骤 2。

步骤 3：划分数据集和训练集。其代码和代码关键描述见文本分类步骤 3。

步骤 4：训练模型。训练模型是该文本分类的最重要的一个步骤，在训练模型的时候可以采取多种机器学习算法，一般选取 AgglomerativeClustering、Kmeans、DBSCAN 算法。在本实验中，为了得到最好的效果，在 TF-IDF 文本向量化之后我使用了上述 3 个模型，并将它们整合到一个函数里面，其核心代码和解释如下所示：

```
model_result = {
    "AgglomerativeClustering": AgglomerativeClustering(n_clusters=4),
    "Kmeans": KMeans(n_clusters=4, random_state=10),
    "DBSCAN": DBSCAN(eps=0.1)
```

```
}
try:
    clf = joblib.load('models/TF_IDF/'+ model + '1.m')
except:
    clf = model_result[model].fit(self.X_train)
    joblib.dump(clf,'models/TF_IDF/'+ model + '1.m')
```

传入的参数为模型的名称。当函数在接收到名称并执行时，先调用现有模型，当 models 文件夹没有模型时，会在模型字典里面查询对应的算法和其超参数，并将其赋值给 clf。之后还是按照正常的程序训练模型并且保存。

步骤 5：使用模型预测与可视化。使用模型预测分为加载模型、模型预测、返回结果三个步骤。模型加载在步骤 4 代码中已经说过，就不再赘述。下面给出使用模型预测的核心代码以及必要解释。

```
result = clf.predict(self.X_test)
print('利用%s 做聚类的结果是：'%model,result)
colors = np.array(["red", "green", "black", "orange", "purple", "beige",
    "cyan", "magenta"])
i = 1
# print(tsne)  # 400*2 的向量
for i in range(tsne.shape[0]):
    plt.scatter(tsne[:, 0], tsne[:, 1], marker='.', c=colors[result[i]])

plt.show()
```

代码实际上是步骤 4 代码的后半部分，当加载模型（clf）结束后对 self.X_test (测试集)进行预测，由于测试集是一个列表，可以将整个列表传到模型中将其聚类，最开始向量化的结果降维成二维矩阵，之后可以分别以矩阵的第一列和第二列的值作为横纵坐标就能做出其对应的散点图，其中同一类的元素散点颜色保持一致。

步骤 6：选取 AgglomerativeClustering、Kmeans、DBSCAN 算法，在 TF-IDF 文本向量化之后我使用了上述 3 个模型，并将它们整合到一个函数里面，其核心代码和解释同步骤 4.

步骤 7：使用模型预测和可视化。使用模型预测分为加载模型、模型预测、返回结果三个步骤。模型加载在步骤 4 代码中已经说过，就不再赘述。其对应的代码和解释同步骤 5。

4.3 聚类结果及可视化

实验结果：

结果 1：利用 TF-IDF 结合 AgglomerativeClustering 算法训练模型，并加载训练

后的模型进行聚类得到的结果如下图 19 所示。

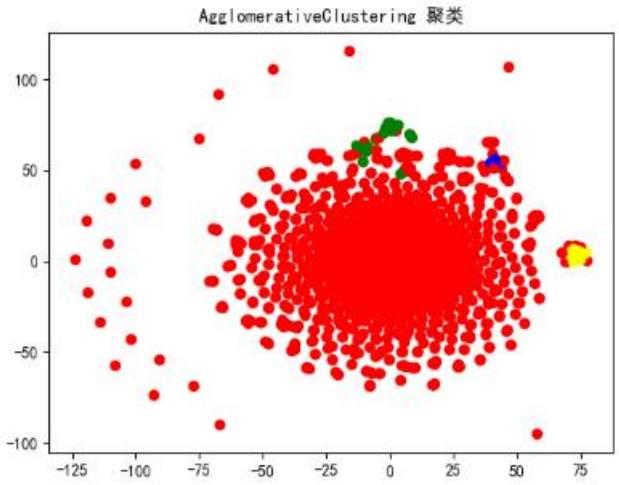

图 19　TF-IDF 结合 AgglomerativeClustering 算法聚类效果图

结果 2：利用 TF-IDF 结合 Kmeans 算法训练模型，并加载训练后的模型进行聚类得到的结果如下图 20 所示。

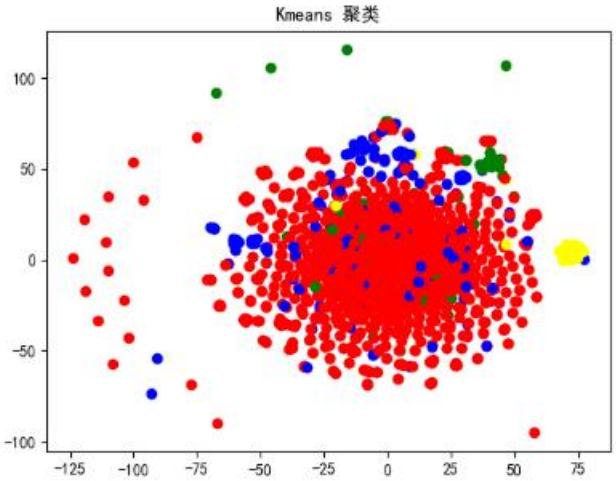

图 20　TF-IDF 结合 Kmeans 算法聚类效果图

结果 3：利用 TF-IDF 结合 DBSCAN 算法训练模型，并加载训练后的模型进行聚类得到的结果如下图 21 所示。

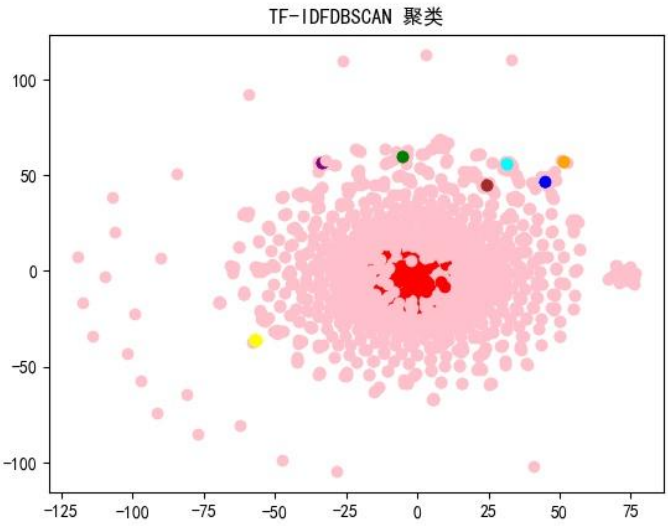

图 21 TF-IDF 结合 DBSCAN 算法聚类效果图

结果 4：利用 Doc2vec 结合 AgglomerativeClustering 算法训练模型，并加载训练后的模型进行聚类得到的结果如下图 22 所示。

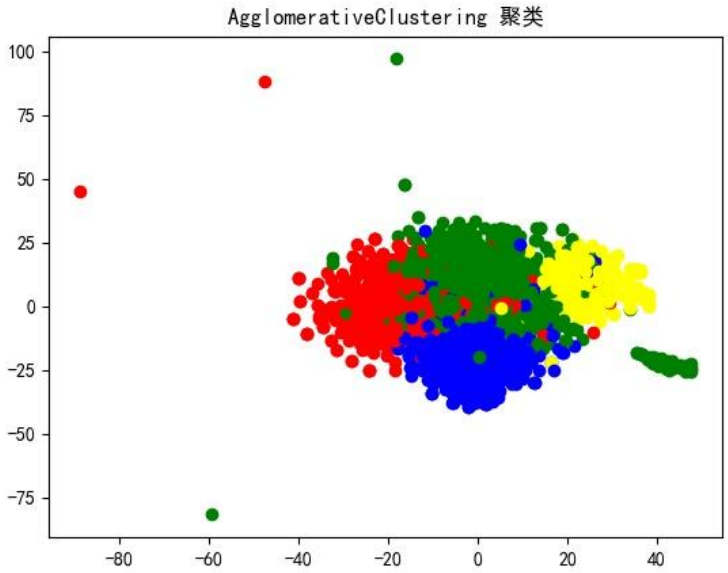

图 22 Doc2vec 结合 AgglomerativeClustering 聚类效果图

结果 5：利用 Doc2vec 结合 Kmeans 算法训练模型，并加载训练后的模型进行聚类得到的结果如下图 23 所示。

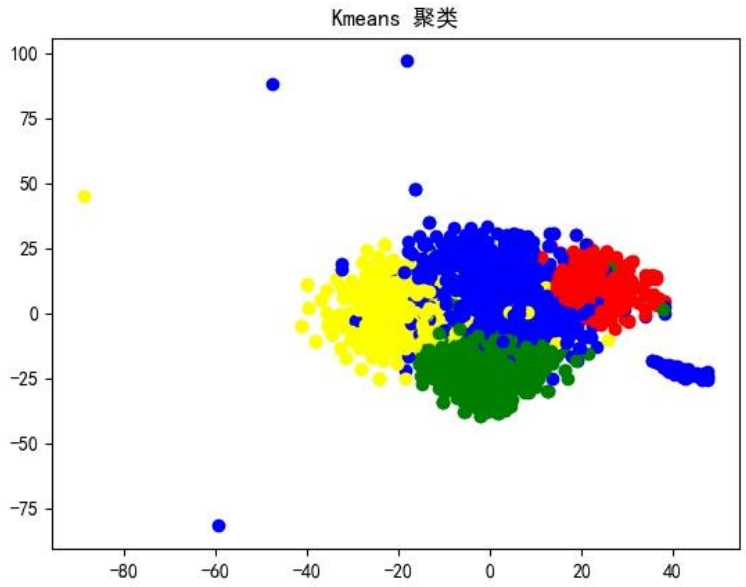

图 23 Doc2vec 结合 Kmeans 聚类效果图

结果 6：利用 Doc2vec 结合 DBSCAN 算法训练模型，并加载训练后的模型进行聚类得到的结果如下图 24 所示。

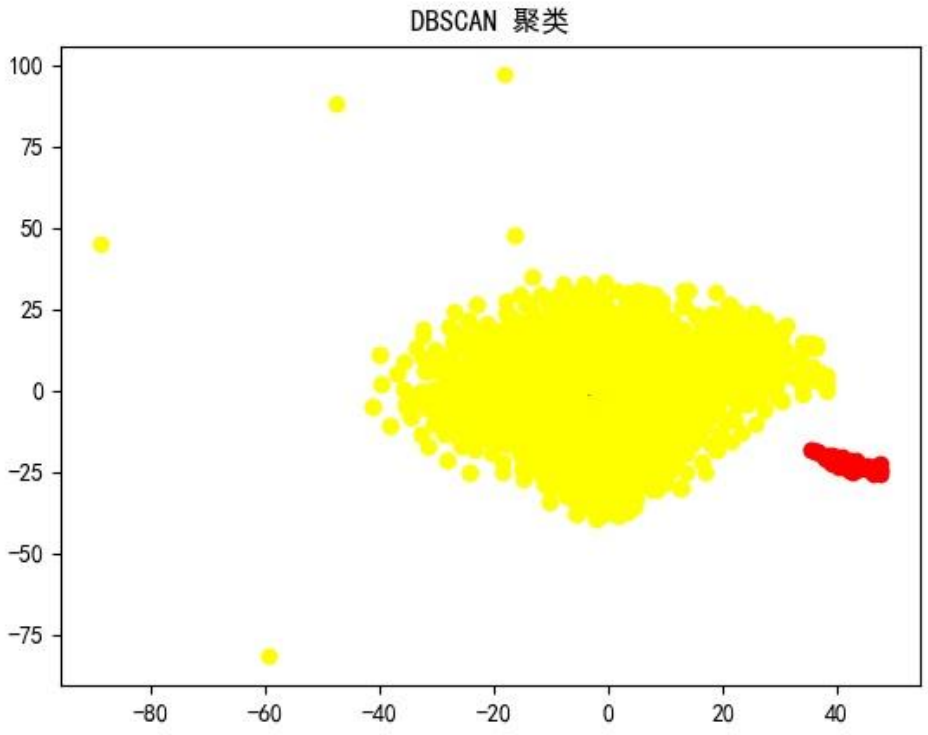

图 24 Doc2vec 结合 DBSCAN 聚类效果图

结果表格样例：

方法	聚类系数
Tfidf+AgglomerativeClustering	
Tfidf+KMeans	
Tfidf+DBSCAN	
Doc2vec+AgglomerativeClustering	
Doc2vec+KMeans	
Doc2vec+DBSCAN	

根据聚类系数，对其中最好的聚类结果进行可视化。

4.4 结果分析

对各种实现方式的原理、效果及优劣进行对比分析。