In [4]:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB,GaussianNB
from sklearn.metrics import f1_score
from sklearn.pipeline import Pipeline
from scipy.sparse import issparse
from sklearn.model_selection import GridSearchCV
```

只修改了train_val()函数于Pipline

版本1，to_array无嵌入，GaussianNB单独

In [2]:

```
# def train_val(D,vect,nb,to_array=False):
#     X_train,X_test, y_train, y_test=D
#     pp=Pipeline([('vect',vect),('nb',nb)])

#     if to_array:#高斯的这个差一个to_array
#         X_train_vect=vect.fit_transform(X_train)
#         X_test_vect=vect.transform(X_test)
#         X_train_vect,X_test_vect=X_train_vect.toarray(),X_test_vect.toarray()
#         nb.fit(X_train_vect,y_train)
#         pred=nb.predict(X_test_vect)
#     else:
#         pp.fit(X_train,y_train)
#         pred=pp.predict(X_test)
#     f1_weighted=f1_score(y_test,pred,average='macro')
#     # df_X_train=pd.DataFrame(X_train_ved.toarray(),columns=vectorizer.get_feature_names_o
#     return f1_weighted
```

版本2，to_array嵌入，自定义转化 器

In [3]:

```
# 创建一个自定义转换器，用于将稀疏矩阵转换为密集矩阵
class SparseToDenseTransformer:
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        if issparse(X):#判断是否为稀疏矩阵
            return X.toarray()
        return X
def train_val(D,vect,nb,to_array=False):
    X_train,X_test, y_train, y_test=D
    if to_array:
        pp=Pipeline([(’vect’,vect),(’hiden’,SparseToDenseTransformer()),(’nb’,nb)])
    else:
        pp=Pipeline([(’vect’,vect),(’nb’,nb)])
    pp.fit(X_train,y_train)
    pred=pp.predict(X_test)
    f1_weighted=f1_score(y_test,pred,average=’macro’)
    # df_X_train=pd.DataFrame(X_train_ved.toarray(),columns=vectorizer.get_feature_names_out
    return f1_weighted
```

实验1：应该使用GaussianNB还是MultinomialNB?

In [4]:

```
#由于内存问题，对比时选用四类数据
news = fetch_20newsgroups(subset=’all’,categories=[’alt.atheism’, ’talk.religion.misc’,’comp
D =train_test_split(news.data,news.target,test_size=0.2, random_state=2023)
#控制变量，使用相同的特征提取器
vect=TfidfVectorizer(stop_words=’english’)
#使用两种分类器
mnb=MultinomialNB(alpha=0.1)
gnb=GaussianNB()
#预测且评价
print(’MultinomialNB的加权F1’,train_val(D,vect,mnb))
print(’GaussianNB的加权F1’,train_val(D,vect,gnb,True))
```

```
MultinomialNB的加权F1 0.9440695360360671
GaussianNB的加权F1 0.9276497366606062
```

实验1结论：性能上，MultinomialNB表现更优，且接收稀疏矩阵传入，大大减少了时空复杂度，因此该选用MultinomialNB。对于后续实验，选定分类器为MultinomialNB，此时由于其接收稀疏矩阵的特性，可以选取所有的数据集，而不用固定4类

实验2：使用相同的训练集和测试集，比较CountVectorizer和TfidfVectorizer的效果

实验 3：考察停用词的作用

In [5]:

```python
news = fetch_20newsgroups(subset='all')
D =train_test_split(news.data,news.target,test_size=0.2, random_state=2023)
#控制变量，使用相同的分类器
mnb=MultinomialNB()
#使用两种特征提取器
cv_sw=CountVectorizer(stop_words='english')
tv_sw=TfidfVectorizer(stop_words='english')
cv=CountVectorizer()
tv=TfidfVectorizer()
#预测且评价
print('CountVectorizer的加权F1(无停用词)',train_val(D,cv,mnb))
print('TfidfVectorizer的加权F1（无停用词）',train_val(D,tv,mnb))
print('CountVectorizer的加权F1(有停用词)',train_val(D,cv_sw,mnb))
print('TfidfVectorizer的加权F1（有停用词）',train_val(D,tv_sw,mnb))
```

```
CountVectorizer的加权F1(无停用词) 0.8354009079661232
TfidfVectorizer的加权F1（无停用词） 0.8341839021793735
CountVectorizer的加权F1(有停用词) 0.8661911054593976
TfidfVectorizer的加权F1（有停用词） 0.8689426147222381
```

结论：【回答问题二】一方面，在同有停用词或同无停用词的条件下，CountVectorizer与TfidfVectorizer的性能差异不大。【回答问题 三】另一方面，无论CountVectorizer或TfidfVectorizer都被停用词的引入显著提高了预测效果

实验 4：考察Tf–idf 平滑的作用

In [6]:

```python
tv=TfidfVectorizer()
tv_no_smooth=TfidfVectorizer(smooth_idf=False)
print('TfidfVectorizer的加权F1（有平滑）',train_val(D,tv,mnb))
print('TfidfVectorizer的加权F1（无平滑）',train_val(D,tv_no_smooth,mnb))
```

```
TfidfVectorizer的加权F1（有平滑） 0.8341839021793735
TfidfVectorizer的加权F1（无平滑） 0.8339279271757325
```

结论：有平滑小小的提升

实验五 交叉验证

In [5]:

```python
news = fetch_20newsgroups(subset='all')
X_train,X_test, y_train, y_test =train_test_split(news.data,news.target,test_size=0.2, rando
vect=TfidfVectorizer(stop_words='english')
nb=MultinomialNB()
parameters = {'nb__alpha': [1,2,3]}
pp=Pipeline([('vect',vect),('nb',nb)])
gs = GridSearchCV(pp,
                  parameters,
                  scoring = ['accuracy','f1_macro'],
                  verbose=2,
                  refit='accuracy',
                  cv=5,
                  n_jobs=-1)
# 执行多线程并行网格搜索。
time_= gs.fit(X_train, y_train)
gs.best_params_, gs.best_score_

# 输出最佳模型在测试集上的准确性。
print(gs.score(X_test, y_test))
print(gs.best_params_)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
0.8843501326259947
{'nb__alpha': 1}
```

结论:最优参数为1