0.前期准备

1700011044 张灏宇

```
## import packages which we need
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
%matplotlib inline
```

```
## python env
import sys
print(sys.version)
```

```
3.8.5 (default, Sep  4 2020, 07:30:14)
[GCC 7.3.0]
```

0.1 任务说明

本任务的目的是基于一些病人的基因表达的数据集，对患**acute myeloid leukemia (AML)**和**acute lymphoblastic leukemia (ALL)**的病人进行区分。因为病人数仅仅几十个，而基因表达的数据纬度达到几千，样本数和特征纬度数量级上的差异会导致直接利用分类方法在原始数据纬度下进行训练效果很差，因此需要利用数据降维方法提取重要的特征纬度，将原始数据降维再训练。本项目将比较利用**NMF**和**SVD**分解方法分别进行降维处理后进行分类，比较两降维方法的效果和特征提取的能力。

0.2 数据集导入

因为数据集太大大，因此不不随着作业上传到教学平台，仅仅上传代码和报告部分。另开设了了一一个github仓 库上传了了整个项目目。

github网网址：https://github.com/zhanghaoyu9931/mathematics_model_porj2

```
## read dataset
sys.path.append('../data')
train_data = pd.read_csv('../data/data_set_ALL_AML_train.csv')
test_data = pd.read_csv('../data/data_set_ALL_AML_independent.csv')
labels = pd.read_csv('../data/actual.csv')

## take a look at the data
print('There are gene expression data of {} patients.\
'.format(labels.shape[0]))
train_data.head()
```

```
There are gene expression data of 72 patients.
```

数据集表格的一部分预览：

	Gene Description	Gene Accession Number	1	call	2	call.1	3	call.2	4	call.3	...	29	call.33	30	call.34
0	AFFX-BioB-5_at (endogenous control)	AFFX-BioB-5_at	-214	A	-139	A	-76	A	-135	A	...	15	A	-318	A
1	AFFX-BioB-M_at (endogenous control)	AFFX-BioB-M_at	-153	A	-73	A	-49	A	-114	A	...	-114	A	-192	A
2	AFFX-BioB-3_at (endogenous control)	AFFX-BioB-3_at	-58	A	-1	A	-307	A	265	A	...	2	A	-95	A
3	AFFX-BioC-5_at (endogenous control)	AFFX-BioC-5_at	88	A	283	A	309	A	12	A	...	193	A	312	A
4	AFFX-BioC-3_at (endogenous control)	AFFX-BioC-3_at	-295	A	-264	A	-376	A	-419	A	...	-51	A	-139	A

5 rows × 78 columns

对于训练和测试数据，每列代表一个病人的基因表达情况，第一行为基因的描述信息，第二行为基因编号。每一个病人的每个基因有一个表达数值，为**int**，以及基因的表达水平，为一个字符，分别为{A:absent, P:present, M:marginal}。

```
## explain the call

call_values = train_data['call'].unique()
print('Values for call columns contain: {}'.format(call_values))
```

```
Values for call columns contain: ['A' 'P' 'M']
```

1.数预处理及降维

该部分包含原始数据的基因表达处理为熟悉的格式（去掉call列等）以及利用NMF、SVD两种不同方法对数据进行特征纬度降低。

```
# remove columns that contain Call data
from preprocess import *

train_X = data_process_X(train_data)
test_X = data_process_X(test_data)

## to accomplish that every elements of array is nonnegative
train_X -= train_X.min()
test_X -= test_X.min()

train_X.head()
```

Gene Accession Number	AFFX-BioB-5_at	AFFX-BioB-M_at	AFFX-BioB-3_at	AFFX-BioC-5_at	AFFX-BioC-3_at	AFFX-BioDn-5_at	AFFX-BioDn-3_at	AFFX-CreX-5_at	AFFX-CreX-3_at	AFFX-BioB-5_st	...	U48730_at	U58516_at
1	262	174	249	124	246	232	678	287	334	421	...	155	287
2	337	254	306	319	277	390	149	295	183	289	...	139	613
3	400	278	0	345	165	140	512	96	288	0	...	285	975
4	341	213	572	48	122	205	637	210	131	246	...	210	611
5	370	202	231	204	311	506	483	341	152	467	...	126	425

5 rows × 7129 columns

1.1 SVD分解

SVD分解进行数据降维的基本原理是对于原始矩阵A_{m*n}，要对矩阵进行分解得到$A_{m*n} = U_{m*k}\Sigma_{k*k}V_{n*k}^T$，其中k为希望降低到的**特征纬度**，满足$k < rank(A)$。其中U为左奇异矩阵，由$A * A^T$的k个**正交单位**特征向量组成，V为右奇异矩阵，由$A^T * A$的k个**正交单位**特征向量组成，满足$U * U^T = I_{m*m}$，$V * V^T = I_{n*n}$。Σ_{k*k}为对角矩阵，对角线元素$A * A^T$的前k个特征值开根号。
$X_{m*k}^{reduction} = A_{m*n} * V_{n*k}$为变换到降维空间后的数据集。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, mean_absolute_error


def svd_decomp(A, k):
    # obtain svd
    A = np.array(A)
    U, S, V = np.linalg.svd(A)

    # inspect shapes of the matrices
    print(U.shape, S.shape, V.shape)

    A_recover = U[:, :k] @ np.diag(S[:k]) @ (V[:k, :])
    transfer = V[:k, :].T
#     print(np.abs(A - A_decomp).sum()/(A.shape[0]*A.shape[1]))
    return A_recover, transfer

def svd_trans(data, trans):
    ## translate to reduction space from original data
    return data@trans

SVD_train_list = []
SVD_test_list = []
## default range from 10-20
for i in range(5, 25):
    train_A = train_X.copy()
    train_A = np.array(train_A)
    test_A = test_X.copy()
    test_A = np.array(test_A)

    A_recover, trans = svd_decomp(train_A, i)

    ## translate
    train_decom = svd_trans(train_A, trans)
    test_decom = svd_trans(test_A, trans)

    scaler = StandardScaler()
    train_decom = scaler.fit_transform(train_decom)
    test_decom = scaler.fit_transform(test_decom)

    SVD_train_list.append(train_decom)
    SVD_test_list.append(test_decom)
#     print(train_decom.shape, test_decom.shape)
```

```
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
(38, 38) (38,) (7129, 7129)
```

1.2 NMF降维

非负矩阵分解NMF也可用于矩阵降维，表示为$A_{m*n} = H_{m*k} * W_{k*n}$其中H为权重矩阵，每行元素为降维后的特征向量，而W的每一行表示降维后的空间每个单位向量是由原来n维空间的单位向量的线性组合而来。因此降维后的每一个单位向量都是原来空间单位向量的非负线性组合。

```
from sklearn import decomposition

nmf_decom = decomposition.NMF(n_components=6, init='nndsvda', tol=5e-3)

NMF_train_list = []
NMF_test_list = []

for i in range(5, 25):
    nmf_decom = decomposition.NMF(n_components=i, init='nndsvda', tol=5e-3)

    train_A = train_X.copy()
    train_A = np.array(train_A)
    test_A = test_X.copy()
    test_A = np.array(test_A)

    nmf_decom.fit(train_A)
    train_decom = nmf_decom.transform(train_A)
    test_decom = nmf_decom.transform(test_A)

    scaler = StandardScaler()
    train_decom = scaler.fit_transform(train_decom)
    test_decom = scaler.fit_transform(test_decom)

    NMF_test_list.append(test_decom)
    NMF_train_list.append(train_decom)
```

1.3 Label数据处理和训练测试集构建

将原本是str类型的label数据变为可以训练的数字数据。

```
## translate label to number from str
l_encoder = LabelEncoder()
labels['CancerColumn'] = l_encoder.fit_transform(labels['cancer'])

print('\n--- Cancer types before label encoding: \n', labels['cancer'].unique())

print('\n--- Cancer types after label encoding: \n', labels['CancerColumn'].unique())
## drop the 'cancer' column
labels = labels.drop('cancer', axis = 1)
```

```
--- Cancer types before label encoding:
 ['ALL' 'AML']

--- Cancer types after label encoding:
 [0 1]
```

label数据处理：

```
## construct labels for train and test

train_y = labels[labels.patient <= 38].reset_index(drop=True)
test_y = labels[labels.patient > 38].reset_index(drop=True)

train_y = train_y.iloc[:,1]
train_y = np.array(train_y)
test_y = test_y.iloc[:,1]
test_y = np.array(test_y)
```

2.SVM算法进行二分类

2.1 对于SVD分解降维的数据进行分类。

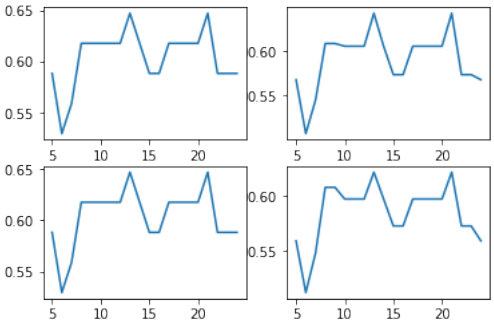
```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from Metrics import *

accuracy = []
precise = []
recall = []
```

```
f1 = []

## find best parameter

for (X_train, X_test) in zip(SVD_train_list, SVD_test_list):
    print('**Now data of {} features are processing.\n'.format(X_train.shape[1]))
    ## Y label
    Y_train = train_y
    Y_test = test_y

    SVM_clf = svm.SVC(class_weight='balanced', probability=True)
    search_space = {'C': np.logspace(-3, 3, 5)}
    gridsearch = GridSearchCV(SVM_clf, param_grid=search_space, scoring='f1_macro', refit=True, cv=10,
n_jobs=-1)
    gridsearch.fit(X_train,Y_train)

    SVM_clf = gridsearch.best_estimator_
    SVM_clf.fit(X_train, Y_train)

    h_a, h_p, h_r, h_f = print_metrics(SVM_clf, X_test, Y_test)

    accuracy.append(h_a)
    precise.append(h_p)
    recall.append(h_r)
    f1.append(h_f)
```

```
**Now data of 5 features are processing.

Confusion matrix

Predicted   0  1
Actual
0          16  4
1          10  4
accuracy = 0.588
precision = 0.568
recall = 0.588
f1 = 0.559
**Now data of 6 features are processing.

Confusion matrix

Predicted   0  1
Actual
0          14  6
1          10  4
accuracy = 0.529
precision = 0.508
recall = 0.529
f1 = 0.512
```

结果可视化如下：

```
## visulization of the result

feature_num = range(5, 25)
plt.figure()
plt.subplot(221)
plt.plot(feature_num, recall)
plt.subplot(222)
plt.plot(feature_num, precise)
plt.subplot(223)
plt.plot(feature_num, accuracy)
plt.subplot(224)
plt.plot(feature_num, f1)
```



从结果上看，该分类方法的最终效果在0.6左右波动，优于Zone model，但模型预测效果不特别理想，很大程度上是因为训练集数据太少，训练集和测试集数据量相差不大，导致模型泛化性不佳。

2.2 对NMF降维的数据进行分类。

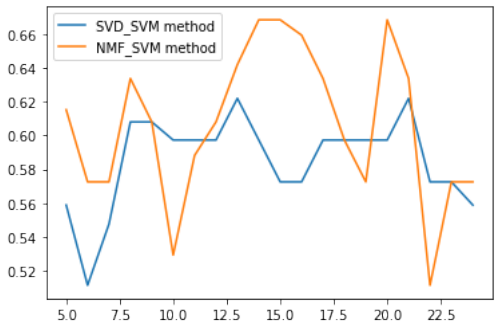
```
accuracy_1 = []
precise_1 = []
recall_1 = []
f1_1 = []

## find best parameter

for (X_train, X_test) in zip(NMF_train_list, NMF_test_list):
    print('**Now data of {} features are processing.\n'.format(X_train.shape[1]))
    ## Y label
    Y_train = train_y
    Y_test = test_y

    SVM_clf = svm.SVC(class_weight='balanced', probability=True)
    search_space = {'C': np.logspace(-3, 3, 5)}
    gridsearch = GridSearchCV(SVM_clf, param_grid=search_space, scoring='f1_macro', refit=True, cv=10,
n_jobs=-1)
    gridsearch.fit(X_train,Y_train)

    SVM_clf = gridsearch.best_estimator_
    SVM_clf.fit(X_train, Y_train)

    h_a, h_p, h_r, h_f = print_metrics(SVM_clf, X_test, Y_test)

    accuracy_1.append(h_a)
    precise_1.append(h_p)
    recall_1.append(h_r)
    f1_1.append(h_f)
```

```
**Now data of 5 features are processing.

Confusion matrix

Predicted   0  1
Actual
0          14  6
1           7  7
accuracy = 0.618
precision = 0.614
recall = 0.618
f1 = 0.615
**Now data of 6 features are processing.

Confusion matrix

Predicted   0  1
Actual
0          15  5
1           9  5
accuracy = 0.588
precision = 0.574
recall = 0.588
f1 = 0.573
```

3.结果分析

对于该二分类问题，F1分数可以很好的衡量模型的最终效果，因此画出F1曲线的结果对比图来**看NMF方法和SVD方法降维效果**的对比情况。

```
plt.figure()
plt.plot(feature_num, f1)
plt.plot(feature_num, f1_1)
plt.legend(['SVD_SVM method', 'NMF_SVM method'])
```



从结果上可以看出，SVD和NMF方法比较而言：

1.总体效果NMF方法更好，黄色曲线大部分的得分都在蓝色曲线上方。

2.在降的维数比较低时，SVD方法效果很差，几乎只有0.5的得分，而NMF有较为不错的效果，推测应该因为SVD分解降到低纬度时所包含的主成分较少，不能完全描述原始data的特征信息；而对于NMF方法，低纬度空间的特征向量也会包含原始特征空间的大部分信息，因此效果较好。

3.SVD分解在特征纬度上升时效果先稳步上升再基本稳定，这体现了SVD的原理，特征纬度越高包含的数据信息量越大。而NMF则波动较大，不同的特征数可能对应的模型效果差别很大。