第十次作业讲评

杨礼铭

一、数据预处理

- 数据标准化:
  - 当特征的量纲不一致时，需要消除量纲影响。
  - 当算法对数据的分布和尺度敏感时，如支持向量机（SVM）、逻辑回归、神经网络、线性回归等。
  - 当数据特征分布接近正态分布时，标准化能更好地保持数据的分布特性。
- 数据归一化
  - 当数据的原始范围过大或各属性间的范围差异较大时，需要将数据转换到一个较小的范围内。
  - 当算法对数据的尺度敏感时，如K-近邻（KNN）、K-均值（K-means）、神经网络等。
  - 当不需要关注数据的分布形状时，归一化可以简化计算过程。
- 数据正交化适用于消除特征间的相关性，降低模型的过拟合风险和降低特征维度。

一、数据预处理

- 1.1 注意axis参数

```python
def normalize(data, axis=None):
    # answer start
    min_vals = data.min(axis=axis)
    max_vals = data.max(axis=axis)
    return (data - min_vals) / (max_vals - min_vals)
    # answer end

def standardize(data, axis=None):
    # answer start
    mean_vals = data.mean(axis=axis)
    std_vals = data.std(axis=axis)
    return (data - mean_vals) / std_vals
    # answer end
```

```python
def normalize(data, axis=None):
    # 获取数据的最小值和最大值
    data_min = np.min(data, axis=axis, keepdims=True)
    data_max = np.max(data, axis=axis, keepdims=True)
    # 数据归一化处理
    return (data - data_min) / (data_max - data_min)


def standardize(data, axis=None):
    # 获取数据的均值和标准差
    data_mean = np.mean(data, axis=axis, keepdims=True)
    data_std = np.std(data, axis=axis, keepdims=True)
    # 数据标准化处理
    return (data - data_mean) / data_std
```

一、数据预处理

- 1.2

PCA：将n维特征根据方差映射到新的k维正交特征（主成分）。所以对这些数据做正交化可以认为就是对这些数据进行等维度的PCA

```python
from sklearn.decomposition import PCA

# answer start
pca = PCA(n_components=2)
orthogonalized_data = pca.fit_transform(data)
# answer end
```

```python
from sklearn.decomposition import PCA

# TODO:
pca = PCA(n_components=data.shape[-1])
orthogonalized_data = pca.fit_transform(data)
```

一、数据预处理

- 1.3


```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 3))
plt.rc('axes', unicode_minus=False)
# Plot original data
plt.subplot(1, 4, 1)

# answer start
plt.scatter(data[:, 0], data[:, 1])
plt.title("Original Data")
# answer end

# Plot normalized data
plt.subplot(1, 4, 2)
# answer start
plt.scatter(normalized_data[:, 0], normalized_data[:, 1])
plt.title("Normalized Data")
# answer end

# Plot standardized data
plt.subplot(1, 4, 3)
# answer start
plt.scatter(standardized_data[:, 0], standardized_data[:, 1])
plt.title("Standardized Data")
# answer end

# Plot orthogonalized data
plt.subplot(1, 4, 4)
# answer start
plt.scatter(orthogonalized_data[:, 0], orthogonalized_data[:, 1])
plt.title("Orthogonalized Data")
# answer end

plt.show()
```

一、数据预处理

- 1.4

```python
# Original data
kmeans_orig = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred_orig = kmeans_orig.fit_predict(data)
silhouette_orig = silhouette_score(data, y_pred_orig)

# Normalized data
kmeans_norm = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred_norm = kmeans_norm.fit_predict(normalized_data)
silhouette_norm = silhouette_score(normalized_data, y_pred_norm)

# Standardized data
kmeans_std = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred_std = kmeans_std.fit_predict(standardized_data)
silhouette_std = silhouette_score(standardized_data, y_pred_std)

# Orthogonalized data
kmeans_orth = KMeans(n_clusters=n_clusters, random_state=random_state)
y_pred_orth = kmeans_orth.fit_predict(orthogonalized_data)
silhouette_orth = silhouette_score(orthogonalized_data, y_pred_orth)

# Output silhouette scores
print("Silhouette Score (Original Data):", silhouette_orig)
print("Silhouette Score (Normalized Data):", silhouette_norm)
print("Silhouette Score (Standardized Data):", silhouette_std)
print("Silhouette Score (Orthogonalized Data):", silhouette_orth)
```

三、特征工程与特征筛选

- 3.2

```python
#TODO: 实现支持向量机分类器的训练和预测
clf = SVC(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

三、特征工程与特征筛选

- 3.3（选做）SelectKBest默认方差分析打分（F检验）

```
"""
自己选择特征为['key','danceability','instrumentalness','speechiness']。观察特征的热力图，分数较高(abs>0.2)的
特征对之间相关性较大(至少线性相关性较大)，可以只保留一个，再观察特征分布图，选出分布差异较大的特征(对不同种类歌曲
分别度较高)，选出了这组特征
然后将特征的值标准化，可以让各指标值处于同一数量级别，避免数值太大的问题，分类时更关注特征之间的差异，标准化处理可以
防止数值较高的指标在分析中过于突出，数值水平较低指标作用减弱的问题，测试结果也表明标准化能提高分类准确率

使用selectKbest对所有特征评分，选取分数最高的4个特征['danceability','speechiness','duration_ms','loudness']
比较数据标准化前后的测试结果，发现标准化能显著提高分类准确率

"""
```

```
Accuracy (multiple features selected by me, original data): 0.6055555555555555
Accuracy (multiple features selected by selectKbest, original data): 0.4488888888888889
Accuracy (multiple features selected by me, standardized data): 0.6288888888888889
Accuracy (multiple features selected by SelectKBest, standardized data): 0.6755555555555556
```

```python
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import StandardScaler
#display(data.head())
# TODO:
std_scaler = StandardScaler()
clf=SVC(random_state=42)
select_features=['key','danceability','instrumentalness','speechiness']

all_features=['mode','speechiness','acousticness','instrumentalness',
    'liveness','valence','tempo','duration_ms','energy','key','loudness','danceability']
selector=SelectKBest(k=12)
selector.fit(data[all_features],y.values)
scores=-np.log10(selector.pvalues_)
pdfeature=pd.DataFrame(index=range(12),columns=['feature'])
pdfeature.loc[:,'feature']=all_features
pdscore=pd.DataFrame(index=range(12),columns=['score'])
pdscore.loc[:,'score']=scores
print('SelectKBest:\n', pd.concat([pdfeature,pdscore],axis=1).sort_values(by='score'),'\n')

selectKbest_feature=['danceability','speechiness','duration_ms','loudness']

X_orig=pd.DataFrame()
for feature in select_features:
    X_orig=pd.concat([X_orig,data[[feature]]],axis=1)
X_train, X_test, y_train, y_test = train_test_split(X_orig, y, test_size=0.2, random_state=42)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (multiple features selected by me, original data):", accuracy)

sX_orig=pd.DataFrame()
for feature in selectKbest_feature:
    sX_orig=pd.concat([sX_orig,data[[feature]]],axis=1)
sX_train, sX_test, sy_train, sy_test = train_test_split(sX_orig, y, test_size=0.2, random_state=42)
clf.fit(sX_train,sy_train)
sy_pred = clf.predict(sX_test)
accuracy = accuracy_score(sy_test, sy_pred)
print("Accuracy (multiple features selected by selectKbest, original data):", accuracy)

X_std = pd.DataFrame()
for feature in select_features:
    data.loc[:,feature]=std_scaler.fit_transform(data[feature].to_numpy().reshape(-1,1))
    X_std=pd.concat([X_std,data[[feature]]],axis=1)

X_std_train, X_std_test, y_std_train, y_std_test = train_test_split(X_std, y, test_size=0.2, random_state=42
clf.fit(X_std_train,y_std_train)
y_std_pred = clf.predict(X_std_test)
accuracy = accuracy_score(y_std_test, y_std_pred)
print("Accuracy (multiple features selected by me, standardized data):", accuracy)


sX_std=pd.DataFrame()

for feature in selectKbest_feature:
    data.loc[:,feature]=std_scaler.fit_transform(data[feature].to_numpy().reshape(-1,1))
    sX_std=pd.concat([sX_std,data[[feature]]],axis=1)
sX_std_train, sX_std_test, sy_std_train, sy_std_test = train_test_split(sX_std, y, test_size=0.2, random_sta
clf.fit(sX_std_train,sy_std_train)
sy_std_pred = clf.predict(sX_std_test)
accuracy = accuracy_score(sy_std_test, sy_std_pred)
print("Accuracy (multiple features selected by SelectKBest, standardized data):", accuracy)
```

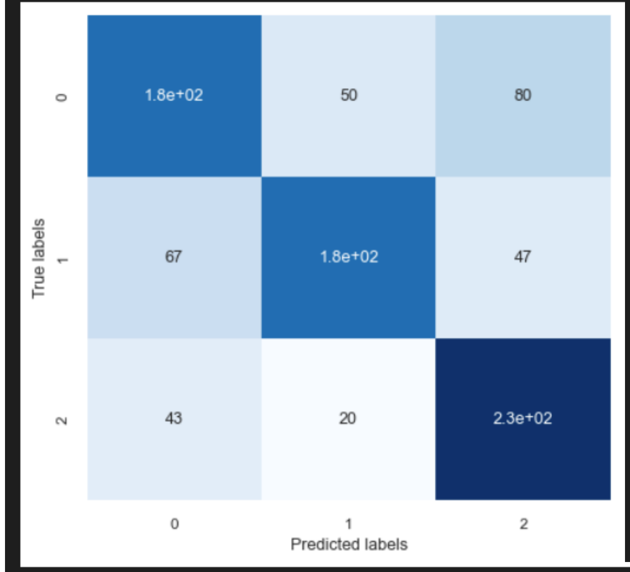
2000012733 缪陆奕

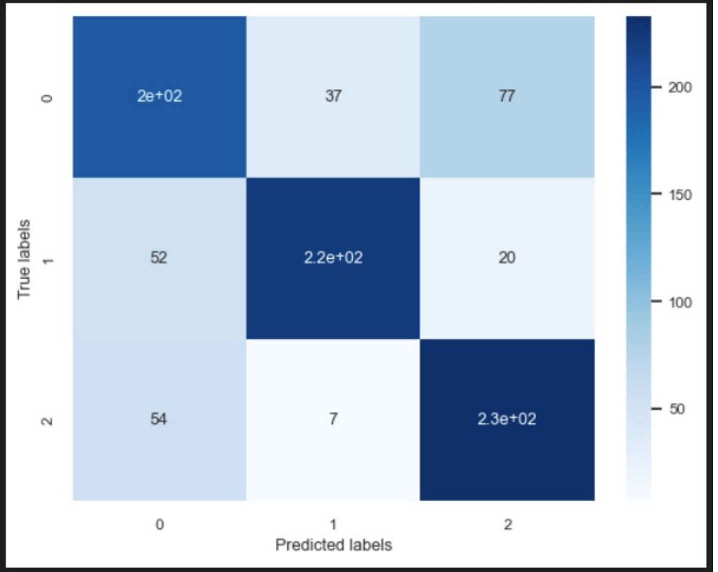
三、特征工程与特征筛选

- 3.3（选做）
- 找了最不相关的5个特征+数据标准化

```
Accuracy (manually selected features): 0.6588888888888889
              precision    recall  f1-score   support

         pop       0.62      0.58      0.60       311
         rap       0.72      0.61      0.66       295
        rock       0.65      0.79      0.71       294

    accuracy                           0.66       900
   macro avg       0.66      0.66      0.66       900
weighted avg       0.66      0.66      0.66       900
```

```
Accuracy (SelectKBest features): 0.7255555555555555
              precision    recall  f1-score   support

         pop       0.65      0.63      0.64       311
         rap       0.84      0.76      0.79       295
        rock       0.71      0.79      0.75       294

    accuracy                           0.73       900
   macro avg       0.73      0.73      0.73       900
weighted avg       0.73      0.73      0.73       900
```






```python
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest, chi2, f_classif

# 根据第零题中绘制的热力图，找到最不相关的五个特征
X_manuallyselect = data[['energy', 'loudness', 'acousticness','danceability','duration_ms']]
y=data['playlist_genre']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_manuallyselect, y, test_size=0.2, random_state=42)

# 对数据进行标准化处理
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#实现支持向量机分类器的训练和预测
clf=SVC(random_state=42)
clf.fit(X_train,y_train)

# 在测试集上进行预测并计算准确率
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (manually selected features):", accuracy)

# 输出classification_report和confusion matrix
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

#使用SelectKBest选择特征
features=['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness','acousticness', 'instrumentalness', 'liveness', 'valence',
X=data[features]
y=data['playlist_genre']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#选取特征
selector=SelectKBest(f_classif,k=5)
X_train_selected=selector.fit_transform(X_train,y_train)
X_test_selected=selector.transform(X_test)
selected_features=selector.get_support(indices=True)
X_kbest=data[[features[i] for i in selected_features]]

# 对数据进行标准化处理
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#实现支持向量机分类器的训练和预测
clf=SVC(random_state=42)
clf.fit(X_train,y_train)

# 在测试集上进行预测并计算准确率
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (SelectKBest features):", accuracy)

# 输出classification_report和confusion matrix
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

2000013158 陈紫怡

三、特征工程与特征筛选

- 3.3（选做）
- 找了在不同分类分布有显著差异的几个特征

```
              precision    recall  f1-score   support

         pop       0.45      0.83      0.58       311
         rap       0.51      0.19      0.27       295
        rock       0.53      0.40      0.46       294

    accuracy                           0.48       900
   macro avg       0.50      0.47      0.44       900
weighted avg       0.50      0.48      0.44       900

[[258  19  34]
 [171  55  69]
 [143  33 118]]
              precision    recall  f1-score   support

         pop       0.56      0.64      0.60       311
         rap       0.84      0.65      0.74       295
        rock       0.66      0.71      0.69       294

    accuracy                           0.67       900
   macro avg       0.69      0.67      0.67       900
weighted avg       0.69      0.67      0.67       900

[[199  25  87]
 [ 81 193  21]
 [ 73  11 210]]
```

```python
from sklearn.feature_selection import SelectKBest
# TODO:

# baseline
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest, f_classif

X_audio = data[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
        'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
        'duration_ms']]
y = data['playlist_genre']

selector = SelectKBest(score_func=f_classif, k=5)
selector.fit(X_audio, y)
X_audio_selected = selector.transform(X_audio)

X_train, X_test, y_train, y_test = train_test_split(X_audio_selected, y, test_size=0.2, random_state=42)
clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# my trial
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix

X = data[['tempo', 'duration_ms', 'speechiness', 'loudness', 'energy', 'danceability']]
y = data['playlist_genre']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))


# Notes
"""
在这一问中，我观察到在下面这几个特征上不同分类的分布有显著差异，因此选择这几个分布
['tempo', 'duration_ms', 'speechiness', 'loudness', 'energy', 'danceability']
同时对它们进行标准化
在准确率上得到了显著提升
"""
```

2000013064 张泽楷

三、特征工程与特征筛选

- 3.3（选做）加入文本特征（TFIDF）

```
Choice of SelectKBest:['danceability' 'speechiness' 'duration_ms']
Accuracy (SelectKBest): 0.44888888888888889
Accuracy (self): 0.68777777777777778
              precision    recall  f1-score   support

         pop       0.62      0.50      0.55       311
         rap       0.84      0.76      0.79       295
        rock       0.63      0.82      0.71       294

    accuracy                           0.69       900
   macro avg       0.70      0.69      0.69       900
weighted avg       0.69      0.69      0.68       900

[[154  37 120]
 [ 48 223  24]
 [ 45   7 242]]
```

```python
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
# TODO:

char = data[['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
       'duration_ms']]

#SelectKBest

#choose
skb = SelectKBest(k=3)
skb.fit(char,data['playlist_genre'])
a = np.where(skb.get_support(),['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
       'duration_ms'],None)
char_skb = a[a!=None]
print('Choice of SelectKBest:{}'.format(char_skb))

#predict
X_skb = data[char_skb]
y = data['playlist_genre']
X_train_skb, X_test_skb, y_train_skb, y_test_skb = train_test_split(X_skb, y, test_size=0.2, random_state=42)
clf = SVC(random_state = 42)
clf.fit(X_train_skb, y_train)
y_pred_skb = clf.predict(X_test_skb)

accuracy_skb = accuracy_score(y_test_skb, y_pred_skb)
print("Accuracy (SelectKBest):", accuracy_skb)


#self
#选择方式：danceability在可视化结果中分类效果明显较好，再在热力图中选择与danceability相关性较高的两个：speechiness和valence;
#除此之外还选择了文本特征
#音频和文本分别测试的结果为：单独使用音频特征的accuracy为0.67，单独使用文本特征的accuracy为0.5455555555555556
X = data[['danceability','speechiness','valence','Lyrics_Processed']]
y = data['playlist_genre']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

count_vect = CountVectorizer(max_features = 9)#调参的结果QAQ
X_train_counts = count_vect.fit_transform(X_train['Lyrics_Processed'])
X_test_counts = count_vect.fit_transform(X_test['Lyrics_Processed'])

tf_transformer_train = TfidfTransformer().fit(X_train_counts)
X_train_tf = tf_transformer_train.transform(X_train_counts).toarray()#转换scipy的稀疏矩阵为ndarray
tf_transformer_test = TfidfTransformer().fit(X_test_counts)
X_test_tf = tf_transformer_test.transform(X_test_counts).toarray()

X_train['Lyrics_Processed'] = X_train_tf#把文本换成tf
X_test['Lyrics_Processed'] = X_test_tf

clf = SVC(random_state = 42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy (self):", accuracy)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

2000011793 王文俊