DSP final project report

B08901090 王懷志

前言

我想要做的題目是，如何幫音檔分類，因為之前在用youtuber music的時候，就覺得明明這首歌就很抒情，怎麼被分到搖滾，很好奇背後分類的機制，因此上網查了一些資料，查到了如何用python做音檔分類。

今天就是要來介紹，這篇文章如何實現 Audio Genre Classification，同時講講我利用這些程式做了哪些事。

構想

這篇文章其實在很多方面都跟我們課堂中提到過的內容有關，例如其中在探討哪些feature能夠作為分類的依據（似第九章(speech recognition)中提過的prosodic features），而這篇文章先根據欲分類出的主題(類型)決定出適合判斷的features。

欲分類的類型有五個，環境音樂(Ambient)、D&B(Drum & Bass)、爵士(Jazz)、饒舌(Rap)、科技舞曲(techno)，之後我會試試加入其他類型，而他們選擇下面幾個特徵作為區分的features

- 節奏(Tempo)：rap的節奏很明顯會較其他類型來得快，而Jazz應該會較慢，但問題是，像環境音樂沒有節奏，而D&B有快有慢，所以這只能作為其中一個次要的feature。
- 頻段(Frequency band)：分成7個頻段(次低頻、低頻、中低頻、中頻、中高頻、presence、brilliance)，可以透過librosa.feature.spectral_contrast() 來計算每個頻段的 spectral contrast。
- MFCC： 課堂中學過的，取13個係數作為features，可以透過librosa.feature.mfcc()計算

實際的操作在Audio.py中，以下為部分的code。

.extract_features( )

```
def extract_features(self, *feature_list, save_local=False):
    """
    Specify a list of features to extract, and a feature vector will be
    built for you for a given Audio sample.

    Option to save vector output locally as a .pkl file in data/ directory

    Currently supported features: 'mfcc', 'spectral_contrast', 'tempo'
    """
    for feature in feature_list:
        if feature == 'mfcc':
            self._extract_mfcc()
        elif feature == 'spectral_contrast':
            self._extract_spectral_contrast()
        elif feature == 'tempo':
            self._extract_tempo()
        else:
            raise KeyError('Feature type not understood; see docstring.')

    if save_local:
        self._save_local(mem_clean=True)
```

.extract_mfcc( )

```python
def _extract_mfcc(self, n_mfcc=12):
    """
    Extract MFCC mean and std_dev vectors for a clip.
    Appends (2*n_mfcc,) shaped vector to
    instance feature vector
    """
    mfcc = librosa.feature.mfcc(self.y,
                                sr=self.sr,
                                n_mfcc=n_mfcc)

    mfcc_mean = mfcc.mean(axis=1).T
    mfcc_std = mfcc.std(axis=1).T
    mfcc_feature = np.hstack([mfcc_mean, mfcc_std])
    self._concat_features(mfcc_feature)
```

.extract_spectral_contrast( )

```python
def _extract_spectral_contrast(self, n_bands=3):
    """
    Extract Spectral Contrast mean and std_dev vectors for a clip.
    Appends (2*(n_bands+1),) shaped vector to
    instance feature vector
    """
    spec_con = librosa.feature.spectral_contrast(y=self.y,
                                                 sr=self.sr,
                                                 n_bands=n_bands)

    spec_con_mean = spec_con.mean(axis=1).T
    spec_con_std = spec_con.std(axis=1).T
    spec_con_feature = np.hstack([spec_con_mean, spec_con_std])
    self._concat_features(spec_con_feature)
```

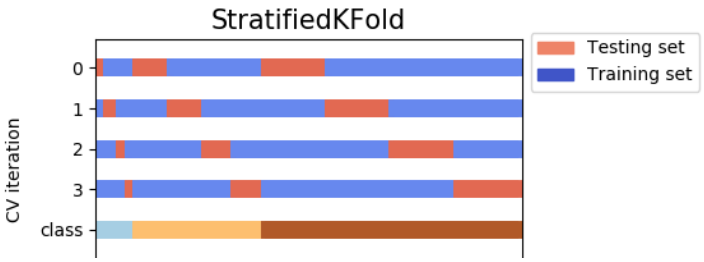
.extract_tempo( )

```python
def _extract_tempo(self):
    """
    Extract the BPM.
    Appends (1,) shaped vector to instance feature vector
    """
    tempo = librosa.beat.tempo(y=self.y, sr=self.sr)
    self._concat_features(tempo)
```

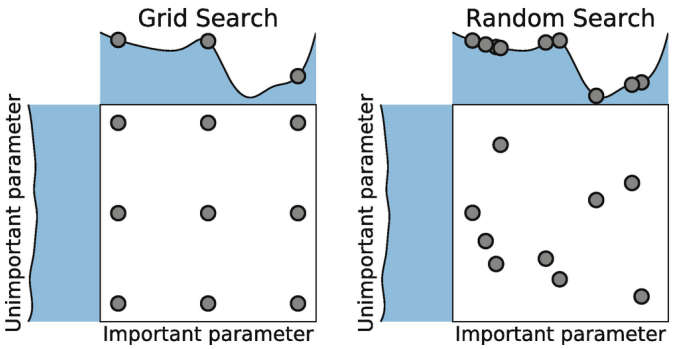
Training

Training 的部分是透過Grid Search尋找參數，Grid Search中會利用k-fold cross-validation(k-fold 交叉驗證)提取和驗證training data，原因是避免 training data 太少而產生的overfitting（k-fold cv: 將training set 拆分成k個，其中一個作為holdup set，另外k-1個拿來train，交叉驗證k次，最後在平均k次的結果)，過程中會以由Standard Scalar和Random Forest Classifier打包成的pipeline 作為Grid Search的model去train我的資料，最後找出最佳模型(Best Estimator)

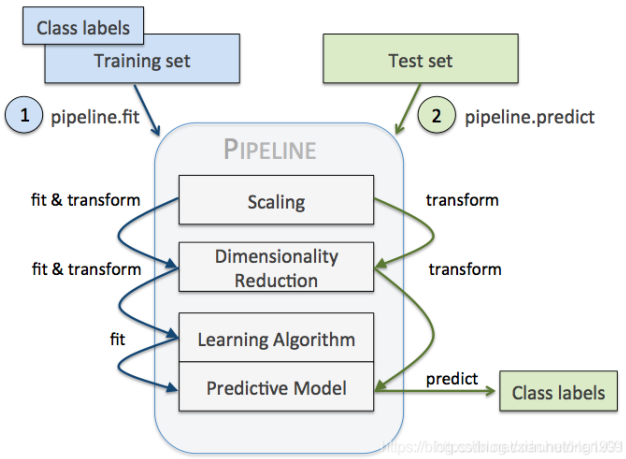
Stratified k-fold(分層 k-fold):



Grid search(網格搜尋):



Pipeline(流水線？):



寫成程式後可分成幾個步驟：

1. 利用sklearn的train_test_split( )將model拆成 train set 和 hold up set
2. 由Standard Scalar和Random Forest Classifier打包成的pipeline 作為Grid Search的model
3. 利用Grid Search 和 Stratified k-fold去train data
4. 找出最佳模型(Best Estimator)

.train_kfold( )

```
def train_kfold(self):
    """
    Using Pipeline objects as they don't leak transformations
    into the validation folds as shown here: https://bit.ly/2N7rdQ
    and here: https://bit.ly/346THQL

    Note that return_train_score=True and verbose=3 in GridSearchC
    is useful for debugging.
    """

    # Save a holdout test set that WON'T go through RepeatedKFold
    # We will not fit any paramter choices to the holdout test set
    X_cv, X_test, y_cv, y_test = train_test_split(
            self.X,
            self.y,
            random_state=42,
            stratify=self.y,
            **self.cfg['tt_test_dict'])

    self.holdout_test_set = (X_test, y_test)

    # From the non-holdout-test data, split off a validation piece
    X_train, X_val, y_train, y_val = train_test_split(
        X_cv,
        y_cv,
        random_state=42,
        stratify=y_cv,
        **self.cfg['tt_val_dict'])
```

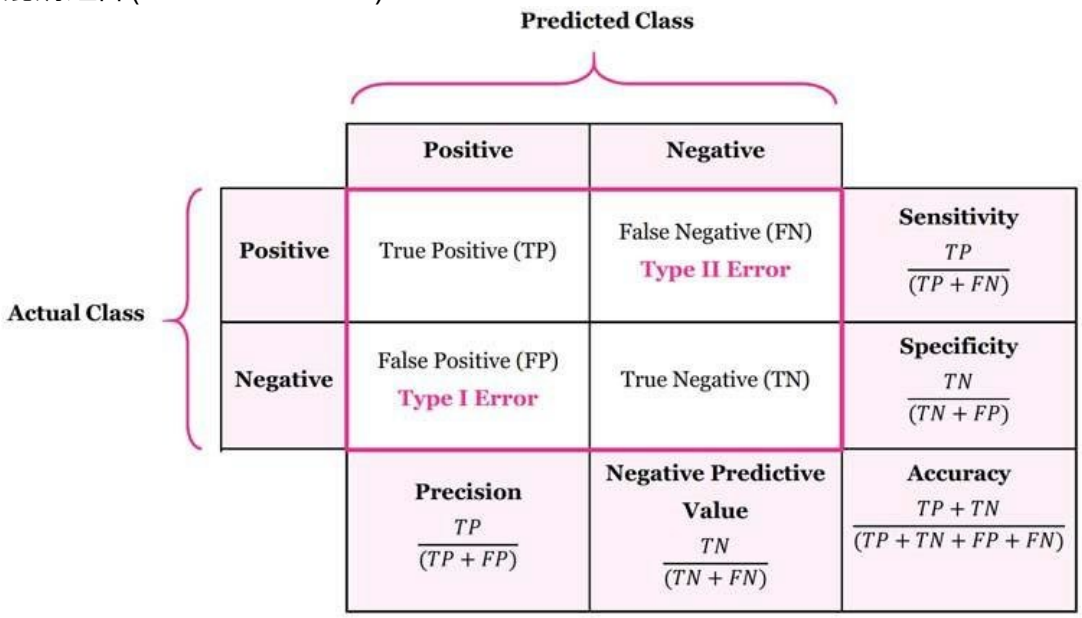
```
# Note these val sets won't go into GridSearchCV
# We'll predict on these in the ._predict() method
self.holdout_val_set = (X_val, y_val)

pipe = Pipeline([
    ('scaler', self.cfg['scaler']),
    ('model', self.cfg['base_model'])
])

# Use stratification within KFold Split inside GridSearchCV
# I believe sklearn now defaults to StratifiedKFold if you pass
# an integer to the cv argument in GridSearchCV, but it did not
kf = StratifiedKFold(**self.cfg['kf_dict'])

# Perform KFold many times according to our Param Grid Search
grid_search = GridSearchCV(estimator=pipe,
                           param_grid=self.cfg['param_grid'],
                           cv=kf,
                           return_train_score=True,
                           verbose=3,
                           **self.cfg['grid_dict'])

# refit the best estimator on the FULL train set
grid_search.fit(X_train, y_train)
self.best_estimator = grid_search.best_estimator_
```

Predict

Train出一個最佳模型之後，下一步就是將holdup set 拿來測試這個模型，並且評估他的表現了，那要如何評估表現呢？需要利用機器學習中的幾個指標：正確率(Accurancy)、準確率(Precision)和敏感度(sensitivity)，。

上述指標可透過混淆矩陣中的四個值來做計算，分別是：真陰性(TN)、真陽性(TP)、偽陰性(FN)、偽陽性(FP)。

混淆矩陣(confusion matrix)



程式碼如下：

```
def _parse_conf_matrix(self, cnf_matrix):
    TP = np.diag(cnf_matrix)
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)

    TP = TP.astype(float)
    FP = FP.astype(float)
    TN = TN.astype(float)
    FN = FN.astype(float)

    return TP, FP, TN, FN
```

而如何用混淆矩陣和holdup set去預測，首先需要將平均和變異數標準化，這邊利用 sklearn.preprocessing 中的 Standard Scalar，然後再透過 train 好的 model 預測的結果。預測結果跟實際結果的落差就透過混淆矩陣來表示。

程式碼如下：

```
def _predict(self, holdout_type):
    if holdout_type == "val":
        X_holdout, y_holdout = self.holdout_val_set

    elif holdout_type == "test":
        X_holdout, y_holdout = self.holdout_test_set

    scaler = self.best_estimator['scaler']
    model = self.best_estimator['model']

    X_holdout_scaled = scaler.transform(X_holdout)
    y_pred = model.predict(X_holdout_scaled)
    cnf_matrix = confusion_matrix(y_holdout, y_pred)

    TP, FP, TN, FN = self._parse_conf_matrix(cnf_matrix)

    return TP, FP, TN, FN
```

為什麼要分 “val”(holdup validation)跟 “test”(holdup test)，因為最後要比較有參與驗證跟完全沒有參與的set最後的prediction會不會差很多，差很多代表可能有overfitting的問題。

遇到的問題(重要)

我原本以為載入音檔然後再跑一次程式就可以了，但是因為這個程式要用到很多package，而有些package跟我電腦中已經載好的package是不相容的，因此以下是我的解決辦法。建造虛擬環境是看這篇

1. 首先要先載好anaconda，因為要利用anaconda建造虛擬環境
2. 載好anaconda之後，在terminal輸入 conda create --name demo python=3.7
3. 建好環境後，輸入 conda activate demo
4. 會看到base 改成demo，接著開始安裝套件，要安裝的套件有 pandas, numpy, scikit-learn=0.23.2 , librosa=0.8.1
5. 如何指定版本呢？輸入 conda install PACAKGE_NAME=VERSION
   ex. conda install librosa=0.8.1
6. 載好之後就可以跑python3 main.py了
7. 跑完後應該會出現如下圖

```
val Set, per class:
TP:[1. 3. 2. 3. 1.], FP:[1. 2. 1. 2. 1.], TN:[13. 11. 13. 12. 12.], FN:[2. 1. 1. 0. 3.]
val False Positive Rate per Class: [0.07142857 0.15384615 0.07142857 0.14285714 0.07692308]
val False Negative Rate per Class: [0.66666667 0.25       0.33333333 0.         0.75      ]
val Accuracy per Class: [0.82352941 0.82352941 0.88235294 0.88235294 0.76470588]
test Set, per class:
TP:[3. 2. 3. 3. 3.], FP:[4. 3. 4. 2. 1.], TN:[19. 19. 19. 20. 21.], FN:[2. 4. 2. 3. 3.]
test False Positive Rate per Class: [0.17391304 0.13636364 0.17391304 0.09090909 0.04545455]
test False Negative Rate per Class: [0.4        0.66666667 0.4        0.5        0.5       ]
test Accuracy per Class: [0.78571429 0.75       0.78571429 0.82142857 0.85714286]
```

預測結果

文章中總共用了704筆資料(.mp3)作為 training data 跟 holdup test data，做出來的結果非常不錯，有參與驗證的validation set的正確率大約落在0.88~0.98之間，而沒有參與驗證的holdup test set的正確率則落在0.9~0.97之間，可以看出並沒有overfitting的問題，他有提到他的D&B(Drum & Bass)涵蓋的範圍較廣，音樂性質跟Techno或jazz都有些相似，所以容易有FN跟FP，但文章中並沒有提到rap的準確率也偏低的原因，我推測應該跟rap本身的data較多有關，因為全部704首歌中有320首左右是rap，因此它的資料分布範圍可能比其他類型來的廣。當然上述都是我的推測。

```
TP:[ 8.  9. 11. 52. 25.], FP:[1. 7. 1. 5. 4.], TN:[113. 100. 110.  61.  90.], FN:[1. 7. 1. 5. 4.]
val False Positive Rate per Class: [0.00877193 0.06542056 0.00900901 0.07575758 0.04255319]
val False Negative Rate per Class: [0.11111111 0.4375     0.08333333 0.0877193  0.13793103]
val Accuracy per Class: [0.98373984 0.88617886 0.98373984 0.91869919 0.93495935]

test Set, per class:
TP:[10. 16. 18. 91. 46.], FP:[ 2.  2.  4. 14.  9.], TN:[194. 182. 188. 100. 153.], FN:[ 6. 12.  2.  7.  4.]
test False Positive Rate per Class: [0.01020408 0.01086957 0.02083333 0.12280702 0.05555556]
test False Negative Rate per Class: [0.375      0.42857143 0.1        0.07142857 0.08       ]
test Accuracy per Class: [0.96226415 0.93396226 0.97169811 0.9009434  0.93867925]
```

```
print(model.encoder.classes_)
# array(['Ambient', 'Drum & Bass', 'Jazz', 'Rap', 'Techno'],
```

我做了什麼

首先，因為他沒有提供他的 database 因此為了測試，我上網抓了100首歌作為我的database去測試（因為電腦空間不足，無法載太多首），為了避免我上面提過資料不均的的情況，每個類型的歌都抓20首下來，以下是我的結果。

```
val Set, per class:
TP:[3. 1. 2. 3. 1.], FP:[3. 2. 0. 1. 2.], TN:[11. 13. 14. 13. 13.], FN:[1. 2. 2. 1. 2.]
val False Positive Rate per Class: [0.21428571 0.13333333 0.         0.07142857 0.13333333]
val False Negative Rate per Class: [0.25       0.66666667 0.5        0.25       0.66666667]
val Accuracy per Class: [0.77777778 0.77777778 0.88888889 0.88888889 0.77777778]
test Set, per class:
TP:[4. 2. 4. 4. 4.], FP:[3. 0. 3. 3. 3.], TN:[21. 24. 21. 21. 21.], FN:[2. 4. 2. 2. 2.]
test False Positive Rate per Class: [0.125 0.    0.125 0.125 0.125]
test False Negative Rate per Class: [0.33333333 0.66666667 0.33333333 0.33333333 0.33333333]
test Accuracy per Class: [0.83333333 0.86666667 0.83333333 0.83333333 0.83333333]
```

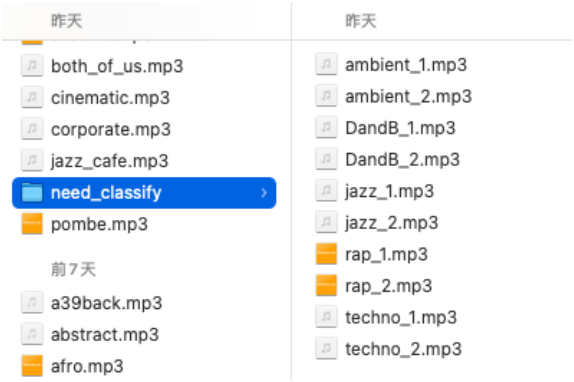
上圖的上半是 validation set 的數據，下半是 holdup test set 的數據，可以看到一樣沒有 over fitting的問題，不過也有可能是資料量太少導致的，可以看到在validation set 的 D&B 和Techno，可能也遇到原本的問題，所以False negative 都很高，我稍微調整一下程式碼後得到下圖

```
val Set, per class:
TP:[3. 1. 2. 3. 1.], FP:[3. 2. 0. 1. 2.], TN:[11. 13. 14. 13. 13.], FN:[1. 2. 2. 1. 2.]
val Precision per Class: [0.5        0.33333333 1.         0.75       0.33333333]
val Sensitivity per Class: [0.75       0.33333333 0.5        0.75       0.33333333]
val Accuracy per Class: [0.77777778 0.77777778 0.88888889 0.88888889 0.77777778]
test Set, per class:
TP:[4. 2. 4. 4. 4.], FP:[3. 0. 3. 3. 3.], TN:[21. 24. 21. 21. 21.], FN:[2. 4. 2. 2. 2.]
test Precision per Class: [0.57142857 1.         0.57142857 0.57142857 0.57142857]
test Sensitivity per Class: [0.66666667 0.33333333 0.66666667 0.66666667 0.66666667]
test Accuracy per Class: [0.83333333 0.86666667 0.83333333 0.83333333 0.83333333]
```

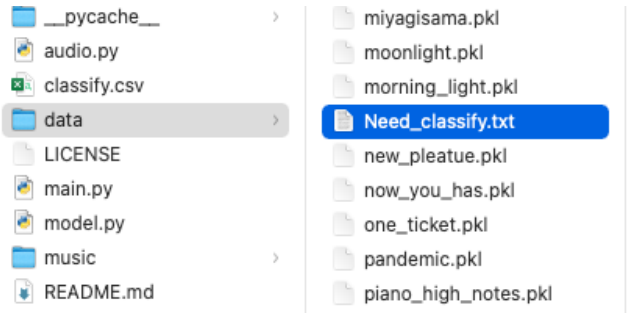
可以看到 D&B跟Techno 的 precision 跟 sensitivity 都很低，應該就是兩者的資料易混淆，解決方法可以增加資料庫或是再將類別細分成更多Genre，或是調整擷取的參數。

跑完資料後，我想要實際去將音樂做分類，因此我下載了10首音樂，但不提供這10首音樂的類型，希望能將這10首音樂的類型都分出來，並且輸出成一個新的檔案。
以下是我測試的流程。

1. 將10首歌載入 “need_classify” 資料夾中，用名字表示他們的類型



2. 在 “data” 的資料夾中新增一個文件，儲存這10首歌的位置跟名稱，但不提供類型




```
1   Name    Location    Genre
2   DandB_1     ./music/need_classify/DandB_1.mp3
3   DandB_2     ./music/need_classify/DandB_2.mp3
4   ambient_1   ./music/need_classify/ambient_1.mp3
5   ambient_2   ./music/need_classify/ambient_2.mp3
6   jazz_1  ./music/need_classify/jazz_1.mp3
7   jazz_2  ./music/need_classify/jazz_2.mp3
8   rap_1   ./music/need_classify/rap_1.mp3
9   rap_2   ./music/need_classify/rap_2.mp3
10  techno_1    ./music/need_classify/techno_1.mp3
11  techno_2    ./music/need_classify/techno_2.mp3
```

3. 如果讀取的資料的 “類型欄”是空的，就只讀取位置

```
def parse_audio_playlist(playlist):
    """
    Assumes an Apple Music playlist saved as plain text as parse input.
    Returns: zip object with (paths, genres)
    """

    df = pd.read_csv(playlist, sep=" \t")
    #print(df[["Genre"]])
    if df[["Genre"]].isnull().values.any():
        print('\nlist has empty genre!!!\n')
        df = df[["Location"]]
        paths = df["Location"].values.astype(str)
        return paths
    #print(type(df))
    df = df[["Location", "Genre"]]
```

4. 擷取10首歌的features

```
classify_data = parse_audio_playlist(playlist="data/Need_classify.txt")
audio_cd_features = []
for md in classify_data:
    path= md
    audio_cd = AudioFeature(path)
    audio_cd.extract_features("mfcc", "spectral_contrast", "tempo", save_local=True)
    audio_cd_features.append(audio_cd)
```

5. 利用建好的模型預測這10首歌的類型

```
model_cd = Model(cd_feature_matrix,genre_labels,model_cfg)
model = Model(feature_matrix, genre_labels, model_cfg)
model.train_kfold()
model.predict(holdout_type="val")
model.predict(holdout_type="test")
model_cd.best_estimator = model.best_estimator
prediction = model_cd._predict(holdout_type="classify")
print(prediction)
cout_genre_data(prediction,"data/Need_classify.txt","classify.csv")
```

這是結果：

```
1   Name    Location    Genre
2   DandB_1 ./music/need_classify/DandB_1.mp3   Drum & Bass
3   DandB_2 ./music/need_classify/DandB_2.mp3   Drum & Bass
4   ambient_1   ./music/need_classify/ambient_1.mp3 Ambient
5   ambient_2   ./music/need_classify/ambient_2.mp3 Ambient
6   jazz_1  ./music/need_classify/jazz_1.mp3    Jazz
7   jazz_2  ./music/need_classify/jazz_2.mp3    Drum & Bass
8   rap_1   ./music/need_classify/rap_1.mp3 Rap
9   rap_2   ./music/need_classify/rap_2.mp3 Rap
10  techno_1    ./music/need_classify/techno_1.mp3  Techno
11  techno_2    ./music/need_classify/techno_2.mp3  Ambient
```

雖然正確率還是沒有很高（只有八成），但有達到我要的效果，如果資料庫能夠有1000筆以上的話，應該就能有九成五左右的正確率了。

調整參數

剛剛有提到過，提高正確率的方式除了增加資料庫或是細分類型外，其實是也可以調整audio讀取的duration和offset，或是留多一點mfcc，所以這邊就是我測試後的結果。

Duration=10s, offset=15s, n_mfcc=12

```
val Set, per class:
TP:[3. 1. 2. 3. 1.], FP:[3. 2. 0. 1. 2.], TN:[11. 13. 14. 13. 13.], FN:[1. 2. 2. 1. 2.]
val Precision per Class: [0.5        0.33333333 1.         0.75       0.33333333]
val Sensitivity per Class: [0.75       0.33333333 0.5        0.75       0.33333333]
val Accuracy per Class: [0.77777778 0.77777778 0.88888889 0.88888889 0.77777778]
test Set, per class:
TP:[4. 2. 4. 4. 4.], FP:[3. 0. 3. 3. 3.], TN:[21. 24. 21. 21. 21.], FN:[2. 4. 2. 2. 2.]
test Precision per Class: [0.57142857 1.         0.57142857 0.57142857 0.57142857]
test Sensitivity per Class: [0.66666667 0.33333333 0.66666667 0.66666667 0.66666667]
test Accuracy per Class: [0.83333333 0.86666667 0.83333333 0.83333333 0.83333333]
```

Duration=20s, offset=15s, n_mfcc=12

```
val Set, per class:
TP:[4. 0. 2. 3. 0.], FP:[2. 3. 0. 2. 2.], TN:[12. 12. 14. 12. 13.], FN:[0. 3. 2. 1. 3.]
val Precision per Class: [0.66666667 0.         1.         0.6        0.        ]
val Sensitivity per Class: [1.   0.   0.5  0.75 0.  ]
val Accuracy per Class: [0.88888889 0.66666667 0.88888889 0.83333333 0.72222222]
test Set, per class:
TP:[4. 1. 3. 3. 4.], FP:[3. 1. 3. 4. 4.], TN:[21. 23. 21. 20. 20.], FN:[2. 5. 3. 3. 2.]
test Precision per Class: [0.57142857 0.5        0.5        0.42857143 0.5       ]
test Sensitivity per Class: [0.66666667 0.16666667 0.5        0.5        0.66666667]
test Accuracy per Class: [0.83333333 0.8        0.8        0.76666667 0.8       ]
```

```
 Name    Location    Genre
 DandB_1 ./music/need_classify/DandB_1.mp3   Techno
 DandB_2 ./music/need_classify/DandB_2.mp3   Drum & Bass
 ambient_1   ./music/need_classify/ambient_1.mp3 Ambient
 ambient_2   ./music/need_classify/ambient_2.mp3 Ambient
 jazz_1  ./music/need_classify/jazz_1.mp3    Ambient
 jazz_2  ./music/need_classify/jazz_2.mp3    Techno
 rap_1   ./music/need_classify/rap_1.mp3 Rap
 rap_2   ./music/need_classify/rap_2.mp3 Drum & Bass
 techno_1    ./music/need_classify/techno_1.mp3  Techno
 techno_2    ./music/need_classify/techno_2.mp3  Drum & Bass
```

Duration=10s, offset=30s, n_mfcc=12

```
val Set, per class:
TP:[2. 1. 2. 3. 1.], FP:[2. 3. 0. 0. 4.], TN:[12. 12. 14. 14. 11.], FN:[2. 2. 2. 1. 2.]
val Precision per Class: [0.5  0.25 1.   1.   0.2 ]
val Sensitivity per Class: [0.5        0.33333333 0.5        0.75       0.33333333]
val Accuracy per Class: [0.77777778 0.72222222 0.88888889 0.94444444 0.66666667]
test Set, per class:
TP:[4. 1. 3. 3. 3.], FP:[2. 6. 1. 4. 3.], TN:[22. 18. 23. 20. 21.], FN:[2. 5. 3. 3. 3.]
test Precision per Class: [0.66666667 0.14285714 0.75       0.42857143 0.5       ]
test Sensitivity per Class: [0.66666667 0.16666667 0.5        0.5        0.5       ]
test Accuracy per Class: [0.86666667 0.63333333 0.86666667 0.76666667 0.8       ]
```

```
 Name    Location    Genre
 DandB_1 ./music/need_classify/DandB_1.mp3   Techno
 DandB_2 ./music/need_classify/DandB_2.mp3   Drum & Bass
 ambient_1   ./music/need_classify/ambient_1.mp3 Ambient
 ambient_2   ./music/need_classify/ambient_2.mp3 Ambient
 jazz_1  ./music/need_classify/jazz_1.mp3    Rap
 jazz_2  ./music/need_classify/jazz_2.mp3    Drum & Bass
 rap_1   ./music/need_classify/rap_1.mp3 Rap
 rap_2   ./music/need_classify/rap_2.mp3 Rap
 techno_1    ./music/need_classify/techno_1.mp3  Drum & Bass
 techno_2    ./music/need_classify/techno_2.mp3  Techno
```

Duration=20s, offset=30s, n_mfcc=12

```
val Set, per class:
TP:[2. 1. 2. 3. 1.], FP:[3. 0. 0. 2. 4.], TN:[11. 15. 14. 12. 11.], FN:[2. 2. 2. 1. 2.]
val Precision per Class: [0.4 1.  1.  0.6 0.2]
val Sensitivity per Class: [0.5        0.33333333 0.5        0.75       0.33333333]
val Accuracy per Class: [0.72222222 0.88888889 0.88888889 0.83333333 0.66666667]
test Set, per class:
TP:[4. 1. 3. 4. 3.], FP:[2. 5. 1. 2. 5.], TN:[22. 19. 23. 22. 19.], FN:[2. 5. 3. 2. 3.]
test Precision per Class: [0.66666667 0.16666667 0.75       0.66666667 0.375     ]
test Sensitivity per Class: [0.66666667 0.16666667 0.5        0.66666667 0.5       ]
test Accuracy per Class: [0.86666667 0.66666667 0.86666667 0.86666667 0.73333333]
```

```
  Name    Location    Genre
 DandB_1 ./music/need_classify/DandB_1.mp3   Techno
 DandB_2 ./music/need_classify/DandB_2.mp3   Techno
 ambient_1   ./music/need_classify/ambient_1.mp3 Ambient
 ambient_2   ./music/need_classify/ambient_2.mp3 Ambient
 jazz_1  ./music/need_classify/jazz_1.mp3    Ambient
 jazz_2  ./music/need_classify/jazz_2.mp3    Drum & Bass
 rap_1   ./music/need_classify/rap_1.mp3 Rap
 rap_2   ./music/need_classify/rap_2.mp3 Rap
 techno_1    ./music/need_classify/techno_1.mp3  Drum & Bass
 techno_2    ./music/need_classify/techno_2.mp3  Drum & Bass
```

Duration=10s, offset=15s, n_mfcc=14

```
val Set, per class:
TP:[2. 1. 2. 3. 1.], FP:[2. 2. 1. 2. 2.], TN:[12. 13. 13. 12. 13.], FN:[2. 2. 2. 1. 2.]
val Precision per Class: [0.5        0.33333333 0.66666667 0.6        0.33333333]
val Sensitivity per Class: [0.5        0.33333333 0.5        0.75       0.33333333]
val Accuracy per Class: [0.77777778 0.77777778 0.83333333 0.83333333 0.77777778]
test Set, per class:
TP:[3. 2. 3. 4. 2.], FP:[4. 0. 7. 3. 2.], TN:[20. 24. 17. 21. 22.], FN:[3. 4. 3. 2. 4.]
test Precision per Class: [0.42857143 1.         0.3        0.57142857 0.5       ]
test Sensitivity per Class: [0.5        0.33333333 0.5        0.66666667 0.33333333]
test Accuracy per Class: [0.76666667 0.86666667 0.66666667 0.83333333 0.8       ]
```

```
  Name    Location    Genre
 DandB_1 ./music/need_classify/DandB_1.mp3   Drum & Bass
 DandB_2 ./music/need_classify/DandB_2.mp3   Drum & Bass
 ambient_1   ./music/need_classify/ambient_1.mp3 Ambient
 ambient_2   ./music/need_classify/ambient_2.mp3 Ambient
 jazz_1  ./music/need_classify/jazz_1.mp3    Jazz
 jazz_2  ./music/need_classify/jazz_2.mp3    Drum & Bass
 rap_1   ./music/need_classify/rap_1.mp3 Rap
 rap_2   ./music/need_classify/rap_2.mp3 Rap
 techno_1    ./music/need_classify/techno_1.mp3  Techno
 techno_2    ./music/need_classify/techno_2.mp3  Ambient
```

結論：

基本上，原本的設定(Duration=10s, offset=15s, n_mfcc=12) 應該是不錯。

Duration調至5s竟然噴錯，調至20s D&B錯誤率就提高了，Techno跟rap都被誤認成D&B。

Offset調高之後，rap的正確率有稍微提升(可能是因為多數rap會有前奏，刪掉後辨認度就會增加)，但其他項幾乎不變，Techno稍微降低。

Mfcc 調高後，反而使Ambient的正確率降低了一些。

因此我最後決定維持稍微提高Offset到20s，其餘維持。