In [176]:
```
import os
import torch
import librosa

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from datasetsmelmean import AudioDataset

import sklearn.metrics as met
# hyper parameters
MAX_EPOCH = 100
```

In [177]:
```
# path to urban sound 8k
data_root = "/home/wew016/UrbanSound8K/"
# path to label
label_path = "/home/wew016/UrbanSound8K/metadata/UrbanSound8K.csv"
```

In [178]:
```
# initialize dataset (feature can be "mfcc" or "spec")
audio_dataset = AudioDataset(3, DataRoot=data_root, LabelPath=label_path
, feature="mel_mean", mode="train")
```

```
verify mel_mean feature success
```

In [181]:
```
# initialize dataloader
data_loader = torch.utils.data.DataLoader(audio_dataset, batch_size=len(
audio_dataset), shuffle=True, num_workers=1)

for idx, data in enumerate(data_loader):
        x_train, y_train = data
        print(x_train.shape)
        print(y_train.shape)
```

```
torch.Size([7807, 128])
torch.Size([7807])
```

In [182]:
```
x_train=x_train.numpy()
x_train.shape
```

Out[182]:
```
(7807, 128)
```

In [183]:
```
y_train=y_train.numpy()
y_train.shape
```

Out[183]:
```
(7807,)
```

In [184]:
```
audio_dataset_test = AudioDataset(3, DataRoot=data_root, LabelPath=label
_path, feature="mel_mean", mode="test")
```

```
verify mel_mean feature success
```

In [186]:
```
# initialize dataloader
data_loader_test = torch.utils.data.DataLoader(audio_dataset_test, batch
_size=len(audio_dataset_test), shuffle=True, num_workers=1)
```

```
for idx, data in enumerate(data_loader_test):
        x_test, y_test = data
        print(x_test.shape)
        print(y_test.shape)
```

```
torch.Size([925, 128])
torch.Size([925])
```

In [187]:
```
x_test=x_test.numpy()
x_test.shape
```

Out[187]:
```
(925, 128)
```

In [189]:
```
y_test=y_test.numpy()
y_test.shape
```

Out[189]:
```
(925,)
```

XGBOOST Model

In [190]:
```
from xgboost import XGBClassifier
```

In [200]:
```
model_xgboost = XGBClassifier(learning_rate =0.2,
                      n_estimators=150,
                      max_depth=4,
                      num_class=10,
                      min_child_weight=5,
                      gamma=0.1,
                      subsample=0.5,
                      colsample_bytree=0.5,
                      objective='multi:softmax',
                      seed=50)
```

In [201]:
```
model_xgboost.fit(x_train,y_train)
```

Out[201]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.1, gpu_i
d=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.2, max_delta_step=0, max_depth=4,
              min_child_weight=5, missing=nan, monotone_constraints='()'
,
              n_estimators=150, n_jobs=0, num_class=10, num_parallel_tre
e=1,
              objective='multi:softprob', random_state=50, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, seed=50, subsample=0.
5,
              tree_method='exact', validate_parameters=1, verbosity=None
)
```

In [203]:
```
pred_train_xgboost = model_xgboost.predict(x_train)
met.accuracy_score(y_train,pred_train_xgboost)
```

Out[203]:
```
0.9953887536825925
```

In [204]:
```
pred_test_xgboost = model_xgboost.predict(x_test)
```

In [205]:
```
y_test
```

Out[205]:
```
array([7, 9, 5, 7, 3, 7, 4, 8, 8, 8, 5, 1, 7, 1, 9, 3, 7, 0, 3, 0, 8, 3,
       7, 2, 5, 1, 5, 7, 7, 3, 4, 3, 2, 6, 4, 5, 9, 9, 5, 8, 1, 8, 2, 6,
       4, 2, 7, 9, 3, 7, 8, 3, 5, 5, 5, 8, 4, 9, 4, 3, 5, 5, 5, 9, 6, 9,
       2, 0, 7, 7, 9, 8, 3, 1, 5, 3, 3, 8, 9, 2, 9, 7, 9, 4, 8, 4, 7, 8,
       9, 8, 8, 7, 2, 5, 5, 4, 9, 3, 7, 5, 0, 0, 7, 3, 9, 5, 0, 2, 7, 4,
       1, 4, 9, 3, 4, 9, 0, 4, 6, 4, 6, 7, 9, 8, 4, 5, 9, 7, 1, 9, 0, 0,
       8, 7, 3, 6, 7, 8, 4, 4, 9, 0, 7, 4, 2, 8, 9, 2, 2, 8, 3, 8, 5, 2,
       5, 3, 9, 2, 4, 4, 3, 2, 5, 7, 9, 3, 5, 2, 7, 6, 0, 0, 7, 9, 5, 4,
       3, 6, 4, 3, 8, 8, 4, 3, 7, 2, 2, 9, 2, 4, 8, 2, 4, 5, 1, 2, 0, 5,
       8, 5, 3, 9, 3, 4, 1, 5, 2, 8, 7, 0, 9, 5, 2, 3, 0, 2, 7, 2, 3, 2,
       4, 2, 4, 4, 9, 2, 9, 2, 5, 2, 4, 5, 2, 0, 9, 8, 5, 7, 0, 0, 2, 2,
       7, 8, 2, 9, 9, 5, 5, 7, 6, 7, 6, 8, 9, 0, 4, 9, 2, 3, 7, 9, 1, 3,
       6, 3, 7, 3, 2, 8, 3, 0, 7, 2, 5, 6, 7, 5, 5, 3, 5, 8, 9, 8, 9, 4,
       2, 3, 4, 4, 8, 9, 9, 2, 7, 5, 5, 5, 2, 8, 0, 2, 0, 7, 0, 8, 8, 7,
       5, 3, 1, 8, 0, 4, 5, 4, 4, 0, 2, 3, 7, 9, 5, 4, 8, 7, 0, 9, 8, 7,
       2, 7, 2, 7, 2, 0, 8, 4, 2, 9, 1, 9, 7, 7, 8, 2, 7, 0, 5, 4, 7, 3,
       8, 3, 5, 8, 0, 3, 9, 2, 5, 4, 7, 1, 3, 5, 9, 2, 6, 8, 2, 7, 5, 5,
       2, 8, 0, 8, 1, 1, 9, 1, 8, 8, 1, 4, 5, 0, 2, 7, 6, 8, 9, 4, 7, 0,
       8, 2, 0, 3, 1, 5, 5, 6, 9, 1, 4, 8, 0, 8, 4, 4, 9, 0, 5, 4, 8, 0,
       5, 9, 0, 1, 2, 7, 5, 0, 4, 5, 0, 3, 5, 2, 0, 0, 9, 2, 7, 7, 3, 3,
       7, 7, 9, 2, 4, 0, 5, 3, 3, 2, 7, 7, 9, 5, 5, 8, 6, 7, 7, 2, 0, 4,
       4, 2, 8, 0, 8, 5, 4, 0, 0, 4, 3, 7, 7, 7, 2, 0, 2, 4, 8, 7, 4, 4,
       5, 2, 2, 7, 4, 7, 5, 9, 5, 7, 3, 0, 8, 9, 5, 1, 4, 1, 7, 2, 9, 4,
       4, 2, 7, 7, 0, 8, 3, 9, 5, 5, 3, 0, 2, 7, 2, 3, 6, 4, 0, 9, 0, 7,
       9, 8, 7, 3, 3, 7, 0, 2, 1, 3, 5, 3, 2, 9, 7, 2, 8, 0, 2, 7, 9, 3,
       9, 3, 0, 5, 0, 4, 9, 8, 2, 9, 7, 9, 3, 6, 0, 5, 6, 3, 8, 1, 9, 2,
       3, 6, 8, 9, 4, 2, 4, 2, 6, 2, 0, 8, 8, 5, 3, 1, 3, 3, 7, 0, 3, 5,
       7, 5, 5, 8, 3, 7, 4, 5, 4, 9, 3, 2, 4, 4, 5, 5, 8, 7, 4, 8, 8, 7,
       4, 1, 0, 6, 0, 0, 8, 8, 1, 4, 7, 7, 7, 7, 8, 9, 8, 9, 3, 2, 5, 0,
       2, 7, 3, 0, 0, 7, 3, 4, 4, 3, 4, 3, 0, 2, 0, 9, 0, 4, 4, 5, 5, 1,
       5, 7, 0, 7, 0, 6, 4, 6, 5, 5, 6, 0, 8, 2, 7, 9, 8, 7, 9, 9, 5, 5,
       4, 0, 9, 9, 4, 9, 2, 8, 8, 2, 7, 1, 0, 8, 6, 3, 6, 2, 4, 2, 2, 5,
       1, 3, 0, 9, 8, 4, 9, 8, 8, 0, 2, 7, 3, 3, 7, 5, 2, 8, 5, 3, 2, 1,
       7, 7, 8, 9, 8, 3, 5, 2, 9, 3, 3, 6, 4, 1, 7, 1, 3, 6, 0, 0, 1, 3,
       4, 5, 8, 8, 5, 8, 5, 4, 8, 3, 5, 8, 9, 5, 0, 0, 5, 9, 8, 5, 4, 9,
       7, 7, 7, 3, 7, 8, 8, 9, 1, 5, 4, 8, 6, 5, 8, 6, 1, 8, 1, 8, 9, 9,
       5, 8, 8, 3, 2, 3, 9, 8, 0, 6, 8, 8, 7, 0, 2, 3, 3, 8, 0, 9, 7, 0,
       8, 2, 7, 7, 7, 0, 7, 8, 3, 4, 4, 8, 7, 0, 8, 9, 8, 3, 6, 2, 0, 8,
       0, 3, 8, 4, 7, 2, 3, 0, 4, 0, 1, 0, 7, 0, 9, 8, 8, 7, 4, 5, 0, 8,
       0, 0, 8, 8, 4, 8, 9, 9, 4, 5, 7, 4, 0, 5, 8, 1, 8, 0, 1, 1, 4, 8,
       4, 8, 1, 0, 5, 3, 2, 4, 3, 2, 0, 9, 9, 3, 9, 6, 0, 5, 4, 7, 7, 3,
       5, 9, 4, 2, 2, 3, 3, 6, 9, 7, 0, 1, 3, 5, 2, 3, 3, 5, 7, 9, 6, 4,
       7])
```

In [206]:
```
pred_test_xgboost
```

Out[206]:
```
array([9, 9, 4, 7, 2, 9, 8, 3, 5, 0, 4, 1, 9, 1, 9, 3, 7, 6, 3, 3, 8, 3,
       9, 3, 9, 6, 4, 7, 7, 2, 3, 8, 2, 6, 9, 5, 9, 2, 2, 8, 1, 8, 3, 6,
       4, 3, 7, 2, 3, 9, 8, 3, 6, 2, 7, 3, 4, 7, 3, 3, 5, 6, 4, 2, 6, 9,
       2, 5, 7, 9, 9, 5, 3, 1, 6, 8, 3, 8, 2, 2, 5, 0, 2, 4, 8, 4, 0, 8,
       2, 8, 8, 9, 2, 6, 9, 3, 2, 3, 7, 9, 5, 5, 4, 3, 2, 4, 7, 3, 9, 9,
       1, 4, 2, 8, 4, 0, 5, 3, 6, 9, 6, 4, 2, 7, 2, 4, 9, 4, 4, 2, 3, 5,
       8, 9, 3, 7, 9, 8, 3, 7, 9, 7, 9, 8, 9, 8, 7, 2, 2, 8, 3, 8, 2, 2,
       6, 3, 3, 6, 6, 4, 3, 2, 5, 9, 5, 3, 3, 2, 9, 6, 2, 3, 9, 2, 3, 3,
       3, 6, 2, 3, 8, 8, 7, 8, 7, 2, 2, 9, 3, 3, 8, 9, 8, 6, 1, 2, 5, 9,
       9, 6, 2, 2, 9, 4, 1, 5, 2, 8, 7, 2, 9, 3, 9, 3, 6, 9, 0, 9, 3, 3,
       3, 2, 4, 6, 9, 0, 8, 0, 4, 2, 3, 4, 0, 2, 9, 8, 2, 9, 5, 5, 2, 3,
       0, 9, 2, 9, 9, 6, 4, 4, 9, 9, 6, 2, 9, 5, 3, 2, 9, 3, 7, 9, 1, 3,
       6, 3, 4, 3, 2, 8, 3, 0, 9, 9, 6, 6, 9, 9, 5, 3, 2, 8, 3, 8, 3, 6,
       2, 3, 4, 8, 8, 0, 9, 3, 9, 2, 7, 3, 2, 8, 7, 2, 5, 4, 2, 8, 8, 9,
```

```
4, 9, 1, 8, 5, 4, 5, 4, 4, 5, 7, 3, 7, 0, 9, 3, 9, 9, 0, 2, 8, 7,
2, 9, 2, 9, 2, 5, 2, 8, 3, 2, 1, 4, 4, 7, 8, 2, 9, 2, 5, 3, 9, 2,
9, 3, 5, 8, 2, 3, 2, 2, 5, 4, 9, 1, 9, 2, 9, 3, 6, 9, 2, 4, 5, 5,
2, 9, 5, 9, 1, 1, 5, 1, 8, 8, 1, 4, 3, 5, 2, 9, 6, 8, 9, 3, 9, 3,
8, 2, 2, 3, 1, 2, 7, 6, 0, 1, 4, 8, 2, 7, 3, 3, 2, 5, 2, 2, 2, 5,
9, 2, 2, 1, 2, 9, 2, 2, 4, 9, 2, 2, 4, 2, 2, 7, 9, 2, 7, 9, 3, 2,
7, 7, 9, 2, 3, 2, 7, 3, 3, 4, 4, 7, 2, 2, 9, 7, 6, 9, 9, 6, 5, 4,
4, 9, 9, 5, 8, 6, 5, 2, 7, 4, 2, 9, 4, 9, 2, 2, 2, 6, 8, 9, 4, 2,
5, 3, 5, 9, 8, 7, 9, 9, 6, 9, 3, 2, 8, 9, 5, 1, 4, 1, 7, 2, 2, 4,
2, 2, 9, 9, 2, 9, 9, 2, 4, 4, 3, 0, 2, 7, 2, 8, 6, 2, 5, 0, 2, 5,
5, 3, 5, 3, 3, 9, 7, 2, 1, 3, 5, 3, 2, 9, 0, 2, 8, 2, 0, 9, 2, 4,
2, 8, 2, 9, 6, 8, 9, 8, 2, 3, 4, 7, 3, 6, 2, 4, 9, 2, 8, 1, 9, 3,
3, 6, 8, 9, 9, 3, 4, 2, 6, 2, 2, 9, 8, 7, 9, 1, 3, 3, 7, 0, 3, 4,
9, 6, 6, 8, 3, 9, 4, 0, 3, 3, 2, 2, 4, 4, 7, 7, 0, 9, 3, 3, 5, 7,
2, 1, 6, 6, 6, 2, 8, 8, 1, 8, 9, 4, 4, 2, 2, 9, 3, 9, 3, 2, 5, 5,
2, 5, 9, 5, 5, 9, 2, 4, 9, 9, 3, 3, 0, 3, 6, 3, 3, 4, 3, 4, 9, 3,
4, 9, 2, 9, 2, 4, 3, 7, 4, 6, 6, 6, 8, 2, 9, 2, 8, 9, 9, 9, 0, 0,
3, 6, 3, 0, 9, 3, 3, 8, 3, 2, 9, 1, 2, 9, 6, 8, 6, 0, 4, 2, 9, 2,
1, 3, 5, 2, 9, 8, 9, 8, 8, 2, 9, 4, 3, 3, 9, 6, 3, 8, 6, 8, 3, 1,
9, 9, 8, 9, 7, 3, 7, 3, 2, 3, 8, 6, 4, 1, 7, 1, 3, 2, 0, 2, 1, 3,
3, 4, 9, 9, 5, 7, 5, 3, 0, 8, 4, 8, 5, 9, 6, 2, 3, 7, 8, 7, 4, 2,
7, 7, 7, 9, 7, 8, 9, 2, 1, 5, 4, 8, 6, 2, 3, 6, 1, 8, 1, 8, 9, 2,
4, 8, 8, 3, 2, 3, 9, 3, 3, 7, 9, 8, 9, 6, 3, 3, 3, 3, 2, 2, 9, 7,
0, 2, 9, 9, 9, 7, 9, 8, 3, 4, 4, 8, 9, 2, 8, 2, 8, 3, 4, 9, 2, 7,
3, 8, 2, 4, 7, 2, 3, 7, 4, 6, 1, 5, 9, 5, 0, 8, 8, 9, 9, 3, 3, 2,
7, 5, 8, 8, 8, 8, 2, 9, 6, 2, 9, 3, 2, 6, 7, 1, 9, 5, 1, 1, 8, 8,
3, 3, 1, 5, 4, 2, 2, 7, 6, 2, 2, 2, 9, 2, 3, 6, 2, 9, 4, 7, 0, 3,
7, 7, 2, 9, 2, 3, 3, 9, 2, 9, 2, 1, 3, 3, 4, 9, 3, 6, 9, 2, 7, 3,
9])
```

In [207]:
```
met.accuracy_score(y_test,pred_test_xgboost)
```

Out[207]:
```
0.4205405405405405
```

In [208]:
```
met.confusion_matrix(y_test,pred_test_xgboost)
```

Out[208]:
```
array([[ 6,  0, 36,  8,  0, 29, 11, 10,  0,  0],
       [ 0, 40,  0,  1,  1,  0,  1,  0,  0,  0],
       [ 5,  0, 58, 19,  2,  1,  2,  1,  0, 12],
       [ 0,  0, 12, 66,  1,  0,  1,  0, 11,  9],
       [ 0,  0,  8, 27, 38,  1,  5,  3, 11,  7],
       [ 3,  0, 14,  8, 22, 18, 18, 10,  0, 14],
       [ 0,  0,  1,  0,  2,  0, 26,  4,  0,  3],
       [ 6,  0,  1,  0, 14,  3,  0, 29,  0, 67],
       [ 4,  0,  6, 10,  0,  3,  0,  7, 72, 17],
       [ 7,  0, 36,  9,  1,  5,  0,  5,  1, 36]])
```

SVM Model

In [213]:
```
import sklearn
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
svr = SVC()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 2, 4], 'gamma':[0.125,
```

```
 0.25, 0.5 ,1, 2, 4]}
clf = GridSearchCV(svr, parameters, scoring='f1_samples')
clf.fit(x_train, y_train)
print('The parameters of the best model are: ')
print(clf.best_params_)
```

In [275]:
```
model_svm = sklearn.svm.SVC(C=1000,
                            kernel='rbf',
                            degree=3,
                            gamma='auto',
                            coef0=0.0,
                            shrinking=True,
                            probability=False,
                            tol=0.001,
                            cache_size=200,
                            class_weight=None,
                            verbose=False,
                            max_iter=-1,
                            decision_function_shape=None,
                            random_state=None)
```

In [276]:
```
model_svm.fit(x_train,y_train)
```

Out[276]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [277]:
```
pred_train_svm = model_svm.predict(x_train)
met.accuracy_score(y_train,pred_train_svm)
```

Out[277]:
```
0.9354425515562956
```

In [279]:
```
pred_test_svm = model_svm.predict(x_test)
met.accuracy_score(y_test,pred_test_svm)
```

Out[279]:
```
0.21081081081081082
```

RandomForest Model

In [280]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [281]:
```
model_RF = RandomForestClassifier(n_estimators=100 ,criterion = "entropy
", bootstrap= False)
```

In [282]:
```
model_RF.fit(x_train,y_train)
```

Out[282]:
```
RandomForestClassifier(bootstrap=False, class_weight=None, criterion='en
tropy',
                       max_depth=None, max_features='auto', max_leaf_nod
es=None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [283]:
```
pred_train_RF = model_RF.predict(x_train)
met.accuracy_score(y_train,pred_train_RF)
```

Out[283]:
```
1.0
```

In [284]:
```
pred_test_RF = model_RF.predict(x_test)
met.accuracy_score(y_test,pred_test_RF)
```

Out[284]:
```
0.4043243243243243
```

KNN Model

In [287]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [290]:
```
model_KNN = KNeighborsClassifier(n_neighbors=10,
                                 weights='uniform',
                                 algorithm='auto',
                                 leaf_size=30,
                                 p=2,
                                 metric='minkowski',
                                 metric_params=None,
                                 n_jobs=None)
```

In [291]:
```
model_KNN.fit(x_train,y_train)
```

Out[291]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=
2,
                     weights='uniform')
```

In [293]:
```
pred_train_KNN = model_KNN.predict(x_train)
met.accuracy_score(y_train,pred_train_KNN)
```

Out[293]:
```
0.7289611886768285
```

In [294]:
```
pred_test_KNN = model_KNN.predict(x_test)
met.accuracy_score(y_test,pred_test_KNN)
```

Out[294]:
```
0.35135135135135137
```

KNN Model

In [ ]:
```

```