ADS Project 5: Urban Sound Classification

Group 4: Tiantian Chen, Yijia Li, Han Lin, Qian Shi, Xiaoyu Zhou

Step 0: Introduction

This is a project to classify unban sounds automatically with machine learning models. Our data comes from Urban Sound Dataset (https://serv.cusp.nyu.edu/projects/urbansounddataset/), which contains 1302 labeled sound recordings from 10 different classes.

We extracted 193 features from these recordings, covering the features of melspectrogram, mfcc, chorma-stft, spectral_contrast and tonnetz.

We splitted the original data and used 70% of them to train the models of GBM, Linear Regression, Random Forest, XGBoost and SVM (with 3 different kernels: linear, rbf, poly), which were applied to classify the left 30% data. Best result was produced by XGBoost, with a accuracy rate of around 75%. Apart from total accuracy rate, we also compared the training time consumed by different models and accuracy rate for each class.

Step 1: Import needed packages

Packages for R

```r
list.of.packages <- c("gbm","randomForest","ggplot2","gridExtra","png","grid")


packages.needed=setdiff(list.of.packages,
                        intersect(installed.packages()[,1],
                                  list.of.packages))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}


library(gbm)
library(randomForest)
library(ggplot2)
library(gridExtra)
library(png)
library(grid)
```

Packages for Python

```
import glob
import os
import librosa
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score
import time
import xgboost as xgb
```

Step 2: Extract Features from audio files

We use Librosa library in python to extract features.

We extracted 5 kinds of features: mfccs,chroma,mel,contrast,tonnetz.

- melspectro gram (Compute a Mel-scaled power)
- mfcc: Mel-frequency cepstral coefficients
- chorma-stft: chromagram from a waveform/power spectrogram
- spectral_contrast: Compute spectral contrast, using method
- tonnetz: Computes the tonal centroid features (tonnetz)

And the output of feature extraction is a feature matrix with 1102 rows, 193 columns.

2.1 Functions to extract features

```
## function to load audio files
## input: file_paths
## output: parsed audio files
def load_sound_files(file_paths):
    raw_sounds = []
    for fp in file_paths:
        X,sr = librosa.load(fp)
        raw_sounds.append(X)
    return raw_sounds

## function to extract features from an audio file
## input: file name
## output: 5 types of features -- mfccs,chroma,mel,contrast,tonnetz
def extract_feature(file_name):
    X, sample_rate = librosa.load(file_name)
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
    tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),
    sr=sample_rate).T,axis=0)
    return mfccs,chroma,mel,contrast,tonnetz


## function to extract features of files in a dictionary
## input: parent dictionary, sub dictionary and default file_extention
## output: features and labels extracted
def parse_audio_files(parent_dir,sub_dirs,file_ext="*.wav"):
    features, labels = np.empty((0,193)), np.empty(0) #193 #63
    for label, sub_dir in enumerate(sub_dirs):
        for fn in glob.glob(os.path.join(parent_dir, sub_dir, file_ext)):
            try:
              mfccs, chroma, mel,contrast,tonnetz = extract_feature(fn)
            except Exception as e:
              print("Error encountered while parsing file: ", fn)
              continue
            ext_features = np.hstack([mfccs,chroma,mel,contrast,tonnetz])
            features = np.vstack([features,ext_features])
            labels = np.append(labels, label+1) #fn.split('/')[2].split('-')[1]
        for fn in glob.glob(os.path.join(parent_dir, sub_dir, "*.mp3")):
            try:
              mfccs, chroma, mel, contrast,tonnetz = extract_feature(fn)
            except Exception as e:
              print("Error encountered while parsing file: ", fn)
              continue
            ext_features = np.hstack([mfccs,chroma,mel,contrast,tonnetz])
            features = np.vstack([features,ext_features])
            labels = np.append(labels, label+1) #fn.split('/')[2].split('-')[1]
    return np.array(features), np.array(labels, dtype = np.int)
```

2.2 Extract features of 10 types of sounds

```
features,labels = parse_audio_files('../data',['1','2','3','4','5','6','7','8','9','10'])
```

Step 3: Apply models

3.1 Logistic Regression

```
##### Read train and test data #####
X_train = pd.read_csv('../data/x_train.csv',header = 0)
y_train = pd.read_csv('../data/y_train.csv',header = 0)['0']
X_test = pd.read_csv('../data/x_test.csv',header = 0)
y_test = pd.read_csv('../data/y_test.csv',header = 0)['0']


##### Calculate class weight #####
train_weight = []
for i in range(10):
    train_weight.append(sum(y_train==i)/len(y_train))
```

```
class_weight = dict(zip(list(range(10)), train_weight))
C_range = [0.01,0.1,0.5,1,5]
solver_range=['newton-cg', 'sag', 'lbfgs']

##### Cross validation #####
accuracy_mat = np.empty(shape = (len(C_param_range),len(solver)))
for i in range(len(C_range)):
    for j in range(len(solver_range)):
        logreg = LogisticRegression(C=C_range[i],multi_class='multinomial',
                                    solver=solver_range[j],max_iter=4000,
                                    class_weight=class_weight)
        #logreg.fit(X_train, y_train)
        accuracy_mat[i,j] = np.mean(cross_val_score(logreg, X_train, y_train,
                                    cv=8,scoring = 'accuracy'))

accuracy_mat = pd.DataFrame(accuracy_mat)

##### Train data using optimal parameters #####
logreg = LogisticRegression(C=0.1,multi_class='multinomial',solver='lbfgs',
                            max_iter=4000,class_weight=class_weight)
start = time.time()
logreg.fit(X_train, y_train)
end = time.time()
end - start

##### Predict on test data #####

y_pred = logreg.predict(X_test)

##### Calculate accuracy #####
accuracy = np.mean(y_pred==y_test)
accuracy

##### Calculate accuracy for each label
each_accuracy = []
for i in range(10):
    acc = sum(y_pred[y_test == i] == y_test[y_test==i])/sum(y_test==i)
    acc = float(str(round(acc,2)))
    each_accuracy.append(acc)

each_accuracy
```

- The classification accuracy for logistic regression is 60.42%.
- The classification accuracy for each label is [0.18, 0.51, 0.56, 0.82, 0.5, 0.48, 0.66, 0.17, 0.53, 0.64].
- The traning time for logistic regression is 3.96 seconds.

3.2 GBM

```
##### Load data #####
feature_train=read.csv("../data/x_train.csv",header = T)
lable_train=read.csv("../data/y_train.csv",header = T)
```

```r
train_data=data.frame(cbind(lable_train,feature_train[,-1]))
colnames(train_data)[1]="y"
features =train_data
label_train=train_data
y=label_train[,1]
X=features[,-1]
#source("/Users/Xiaoyu/Desktop/ads-proj5/tune gbm.r")
err_cv
colnames(err_cv) = c("mean of cv.error","sd of cv.error")
rownames(err_cv) = c("depth = 3", "depth = 5", "depth = 7", "depth = 9","depth = 11")
#write.csv(err_cv,file = "./Users/Xiaoyu/Desktop/ads-proj5/err_cv_for_baseline.csv")


##### Train model #####

#We chose depth=7, n.trees = 500,shrinkages = 0.01,

depth=7
shrinkage=0.01
train<-function(X, y, depth, shrinkage){
  fit_gbm = gbm.fit(X, y,
                    distribution = "multinomial",
                    n.trees = 500,
                    interaction.depth = depth,
                    shrinkage = shrinkage,
                    bag.fraction = 0.5,
                    verbose=FALSE)
  best_iter <- gbm.perf(fit_gbm, method="OOB", plot.it = FALSE)
  return(model=list(fit=fit_gbm, iter=best_iter))
}
mm=train(X,y,7,0.01)

##### Make prediction #####
feature_test=read.csv("../data/x_test.csv",header = T)
lable_test=read.csv("../data/y_test.csv",header = T)
test_data=data.frame(cbind(lable_test,feature_test[,-1]))
test_data2= test_data[,-1]

test.gbm <- function(model, test.data)
{
  pred<- predict(model$fit, newdata = test.data, n.trees = model$iter, type="response")
  pred<-data.frame(pred)
  return(apply(pred,1,which.max)-1)
}
nn=test.gbm(mm,test_data2)
lable_test$pd=nn
sum(lable_test$X0==lable_test$pd)/nrow(lable_test)
```

- The classification accuracy for GBM is 68.27795%.
- The classification accuracy for each label is [0.36, 0.57, 0.73, 0.88, 0.6, 0.52, 0.74, 0.17, 0.47, 0.73].
- The traning time for GBM is 20mins.

3.3 Random Forest

```r
##### Load and process data #####
training_y<- read.csv("../data/y_train.csv")
training_x<- read.csv("../data/x_train.csv")
testing_y<- read.csv("../data/y_test.csv")
testing_x<- read.csv("../data/x_test.csv")
colnames(training_y)<- "y"
colnames(testing_y)<- "y"

##### Train and Test Function #####
rf_train<- function(dat, label, N){
  library(randomForest)
  df <- as.data.frame(cbind(dat, label))
  set.seed(11)
  fit <- randomForest(as.factor(y)~.,data = df, importance = TRUE, ntree = N)
  return(fit)
}

rf_test<- function(model, dat){
  pred <-predict(model,newdata=dat)
  return (pred)
}

##### Chooce the best parameter use cross validation #####
N<-c(300,400,500,600,700,800,900)
for(i in 1:7){
  tm_train=NA
  tm_train <- system.time(result <- rfcv(training_x,
                         as.factor(unlist(training_y)), cv.fold=3, N=N[i]))
  cat("Accuracy rate for " ,N[i]," trees is ", 1-result$error.cv[1], "\n" )
  cat("Elapsed time for " ,N[i]," trees is ", tm_train[3], "seconds \n")
}

#Best N
#Accuracy rate for  900  trees is  0.6108949
#Elapsed time for  900  trees is  46.791 seconds

##### Train the model with the best N and test the model using test dataset #####
tm_train=NA
tm_train <- system.time(fit <- rf_train(training_x,training_y,N=900))
cat("Elapsed time for training is ", tm_train[3], "seconds \n")

tm_test <- system.time(pred_test <- rf_test(fit, testing_x))
error <- mean(pred_test != as.factor(unlist(testing_y)))
cat("Accuracy rate = ", 1-error, "\n")
cat("Elapsed time for testing is ", tm_test[3], "seconds \n")

##### Accuracy rate for each label #####
correct_list=c(0,0,0,0,0,0,0,0,0,0)
label_list=c(0,0,0,0,0,0,0,0,0,0)
for (i in 1:length(pred_test)){
  for (j in 0:9){
    if (as.factor(unlist(testing_y))[i] == j){
      label_list[j+1] = label_list[j+1]+1
      if (as.factor(unlist(testing_y))[i] == pred_test[i]){
      correct_list[j+1] = correct_list[j+1]+1
      }
    }
  }
}
accuracy_list <- correct_list/label_list
round(accuracy_list, 2)
```

- The classification accuracy for random forest is 66.46%.
- The classification accuracy for each label is [0.18, 0.57, 0.73, 0.90, 0.50, 0.35, 0.83, 0.33, 0.33, 0.69].
- The traning time for random forest is 37.09 seconds.

3.4 Support Vector Machine (SVM)

```
##### Read train and test data #####
train_featu = pd.read_csv("x_train.csv")
train_label = pd.read_csv("y_train.csv")
train_label = np.array(train_label).reshape(len(train_label), )

test_featu = pd.read_csv("x_test.csv")
test_label = pd.read_csv("y_test.csv")
test_label = np.array(test_label).reshape(len(test_label), )


##### Choose tuning parameters #####
tuning_paras = {'kernel': ('rbf', 'poly'),
                'C': (1, 3, 5, 7, 15, 30, 50, 100, 300, 500, 1000),
                'C_poly': (1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 30, 50, 70, 100),
                'gamma_rbf': (0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 1, 3, 5, 10, 30, 50, 100),
                'gamma_poly': (0.01, 0.03, 0.05, 0.07, 0.09, 0.1, 0.2)}
```

3.4.1 Linear SVM:

```
##### Train parameters with CV #####
acu_linear = np.empty(shape=(len(tuning_paras['C']), 5))

for c in range(len(tuning_paras['C'])):
    svm_linear = svm.LinearSVC(C=tuning_paras['C'][c])
    # Cross-validation with K = 5
    acu_linear[c] = cross_val_score(svm_linear, train_featu, train_label, cv=5)
    print("C =", tuning_paras['C'][c], "finished !")

print("Accuray:", acu_linear)


##### Calculate CV accuracy #####
cv_acu = np.mean(acu_linear, axis=1)
```

```python
##### Choose the best parameter #####
max_ind = np.argmax(cv_acu)
best_c = tuning_paras['C'][max_ind]


##### Predict on test data #####
start=datetime.now()
svm_linear = svm.LinearSVC(C = best_c).fit(train_featu, train_label)
pred_label = svm_linear.predict(test_featu)
print ("Time:", datetime.now() - start)


##### Calculate total accuracy #####
total_acu = np.sum(pred_label == test_label) / len(test_label)
print("Total accuracy:", total_acu)


##### Calculate the accuracy for each class #####
categories = ["air_conditioner", "car_horn", "children_playing",
              "dog_bark", "drilling", "enginge_idling",
              "gun_shot", "jackhammer", "siren", "street_music"]
for i in range(10):
    acu = np.sum(pred_label[test_label == i] == i) / np.sum(test_label == i)
    print("The accuracy for {0} is {1}".format(categories[i], acu))
```

- The classification accuracy for linear SVM is 49.55%.
- The classification accuracy for each label is [0.18, 0.46, 0.80, 0.36, 0.47, 0.48, 0.49, 0.33, 0.33, 0.71].
- The traning time for linear SVM is 1.63 seconds.

3.4.2 SVM with RBF kernel:

```python
##### Train parameters with CV and Calculate the CV accuracy #####
acu_rbf = np.empty(shape=(len(tuning_paras['C']), len(tuning_paras['gamma_rbf'])))

for c in range(len(tuning_paras['C'])):
    for g in range(len(tuning_paras['gamma_rbf'])):
        svm_rbf = svm.SVC(kernel='rbf', C=tuning_paras['C'][c],
                          gamma=tuning_paras['gamma_rbf'][g])
        # Cross-validation with K = 5
        acu_rbf[c][g] = np.mean(cross_val_score(svm_rbf, train_featu, train_label, cv=5))
    print("C =", tuning_paras['C'][c], "finished !")

print("Accuray:", acu_rbf)


##### Choose the best parameter #####
best_c = tuning_paras['C'][np.unravel_index(acu_rbf.argmax(), acu_rbf.shape)[0]]
best_gamma = tuning_paras['gamma_rbf'][np.unravel_index(acu_rbf.argmax(),
                                                        acu_rbf.shape)[1]]
```

```
##### Predict on test data #####
start=datetime.now()
svm_rbf = svm.SVC(kernel='rbf', C = best_c, gamma = best_gamma).fit(
                                                    train_featu, train_label)
pred_label = svm_rbf.predict(test_featu)
print ("Time:", datetime.now() - start)


##### Calculate total accuracy #####
total_acu = np.sum(pred_label == test_label) / len(test_label)
print("Total accuracy:", total_acu)


##### Calculate the accuracy for each class #####
categories = ["air_conditioner", "car_horn", "children_playing",
              "dog_bark", "drilling", "enginge_idling",
              "gun_shot", "jackhammer", "siren", "street_music"]
for i in range(10):
    acu = np.sum(pred_label[test_label == i] == i) / np.sum(test_label == i)
    print("The accuracy for {0} is {1}".format(categories[i], acu))
```

- The classification accuracy for SVM with RBF kernel is 25.68%.
- The classification accuracy for each label is [0, 0, 0, 1, 0, 0, 0.03, 0, 0, 0].
- The traning time for SVM with RBF kernel is 0.43 seconds.

3.4.3 SVM with polynomial kernel (degree=3):

```
##### Train parameters with CV and Calculate the CV accuracy #####
acu_poly = np.empty(shape=(len(tuning_paras['C_poly']), len(tuning_paras['gamma_poly'])))

for c in range(len(tuning_paras['C_poly'])):
    for g in range(len(tuning_paras['gamma_poly'])):
        svm_poly = svm.SVC(kernel='poly',degree=3,C=tuning_paras['C_poly'][c],
                                           gamma=tuning_paras['gamma_poly'][g])
        # Cross-validation with K = 5
        acu_poly[c][g] = np.mean(cross_val_score(svm_poly, train_featu, train_label,
                                                                         cv=5))
        #print("gamma =", tuning_paras['gamma_poly'][g], "finished !")
    print("C =", tuning_paras['C_poly'][c], "finished !")

print("Accuray:", acu_poly)


##### Choose the best parameter #####
best_c = tuning_paras['C_poly'][np.unravel_index(acu_poly.argmax(), acu_poly.shape)[0]]
best_gamma = tuning_paras['gamma_poly'][np.unravel_index(acu_poly.argmax(),
                                                        acu_poly.shape)[1]]


##### Predict on test data #####
start=datetime.now()
```

```
svm_poly = svm.SVC(kernel='poly', C = best_c, gamma = best_gamma).fit(
                                                       train_featu, train_label)
pred_label = svm_poly.predict(test_featu)
print ("Time:", datetime.now() - start)


##### Calculate total accuracy #####
total_acu = np.sum(pred_label == test_label) / len(test_label)
print("Total accuracy:", total_acu)


##### Calculate the accuracy for each class #####
categories = ["air_conditioner", "car_horn", "children_playing",
              "dog_bark", "drilling", "enginge_idling",
              "gun_shot", "jackhammer", "siren", "street_music"]
for i in range(10):
    acu = np.sum(pred_label[test_label == i] == i) / np.sum(test_label == i)
    print("The accuracy for {0} is {1}".format(categories[i], acu))
```

- The classification accuracy for SVM with polynomial kernel (degree=3) is 51.96%.
- The classification accuracy for each label is [0.27, 0.57, 0.61, 0.58, 0.37, 0.17, 0.74, 0.42, 0.4, 0.51].
- The traning time for SVM with polynomial kernel (degree=3) is 0.98 seconds.

3.5 XGBoost

```
# -----------------------------------
# --- Train Model: Baseline Model ---
# -----------------------------------
print("Baseline:")
start = time.time()
model_baseline = xgb.XGBClassifier(learning_rate = 0.1
                        , objective = 'multi:softmax'
            , n_estimators = 140
                        , max_depth = 1
                        , min_child_weight = 0
                        , subsample = 1
                        , colsample_bytree = 1
                        , gamma = 0
                        , seed = 1234
                        , nthread = -1)
model_baseline.fit(X_train, y_train)
print ("Training finished in %d seconds." % (time.time()-start))

pred = model_baseline.predict(X_test)
y_label = y_test.values
print ('classification error before tuning=%f' %
(sum([pred[i] != y_label[i] for i in range(len(y_label))]) / float(len(y_label)) ))

# Training finished in 7 seconds.
# classification error=0.368580
# ----------------------
# --- Model Tuning ---
# ----------------------

xg_train = xgb.DMatrix(X_train, label=y_train)
xg_test = xgb.DMatrix(X_test, label=y_test)

params = dict()
params['objective'] = 'multi:softmax'
params['num_class'] = 10
params['eta'] = 0.1
params['max_depth'] = 3
params['min_child_weight'] = 1
params['colsample_bytree'] = 1
params['subsample'] = 1
params['gamma'] = 0
params['seed']=1234

#### to find best max_depth
%%time
scores = []
for max_depth in [1,3,5,7,9,11]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = max_depth
    params['min_child_weight'] = 1
    params['colsample_bytree'] = 1
    params['subsample'] = 1
    params['gamma'] = 0
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(max_depth,best_score,best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                   params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth','min_child_weight',
                                      'colsample_bytree','subsample','gamma',
                                      'best_iteration'])
best_max_depth = scores.sort_values(by='score',ascending=True)['max_depth'].values[0]
print('best max_depth is', best_max_depth)

#### to find best min_child_weight
```

```
%%time
scores = []
for min_child_weight in [0.01, 0.1, 1, 10, 100]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = best_max_depth
    params['min_child_weight'] = min_child_weight
    params['colsample_bytree'] = 1
    params['subsample'] = 1
    params['gamma'] = 0
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(min_child_weight, best_score, best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],
                  params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                   params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth',
                                      'min_child_weight',
                                      'colsample_bytree','subsample',
                                      'gamma','best_iteration'])
best_min_child_weight = scores.sort_values(by='score',ascending=True)
                                           ['min_child_weight'].values[0]
print ('best min_child_weight is', best_min_child_weight)

#### to find best min_child_weight
%%time
scores = []
for min_child_weight in [0.01, 0.1, 1, 10, 100]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = best_max_depth
    params['min_child_weight'] = min_child_weight
    params['colsample_bytree'] = 1
    params['subsample'] = 1
    params['gamma'] = 0
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(min_child_weight, best_score, best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],
                                     params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                          params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth',
                                      'min_child_weight',
                                      'colsample_bytree','subsample',
                                      'gamma','best_iteration'])
best_min_child_weight = scores.sort_values(by='score',ascending=True)
                                        ['min_child_weight'].values[0]
print ('best min_child_weight is', best_min_child_weight)

#### to find best colsample_bytree
%%time
scores = []
for colsample_bytree in [0.1, 0.5, 1]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = best_max_depth
    params['min_child_weight'] = best_min_child_weight
    params['colsample_bytree'] = colsample_bytree
    params['subsample'] = 1
    params['gamma'] = 0
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(colsample_bytree, best_score, best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],
                   params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                  params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth',
                                      'min_child_weight',
                                      'colsample_bytree','subsample',
                                      'gamma','best_iteration'])
```

```
best_colsample_bytree = scores.sort_values(by='score',ascending=True)
                                       ['colsample_bytree'].values[0]
print ('best colsample_bytree is', best_colsample_bytree)

#### to find best subsample
%%time
scores = []
for subsample in [0.1, 0.25, 0.5, 0.75, 1]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = best_max_depth
    params['min_child_weight'] = best_min_child_weight
    params['colsample_bytree'] = best_colsample_bytree
    params['subsample'] = subsample
    params['gamma'] = 0
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(subsample, best_score, best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],
                   params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                   params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth',
                                      'min_child_weight',
                                      'colsample_bytree','subsample',
                                      'gamma','best_iteration'])
best_subsample = scores.sort_values(by='score',ascending=True)['subsample'].values[0]
print ('best subsample is', best_subsample)

#### to find best gamma
%%time
scores = []
for gamma in [0.005,0.01, 0.05,0.1]:
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = best_max_depth
    params['min_child_weight'] = best_min_child_weight
    params['colsample_bytree'] = best_colsample_bytree
    params['subsample'] = best_subsample
    params['gamma'] = gamma
    params['seed']=1234

    cv_results = xgb.cv(params, xg_train,
                        num_boost_round=1000000,
                        nfold=3,
                        metrics={'merror'},
                        seed=1234,
                        callbacks=[xgb.callback.early_stop(50)])
    best_iteration = len(cv_results)
    best_score = cv_results['test-merror-mean'].min()
    print(gamma, best_score, best_iteration)
    scores.append([best_score,params['eta'],params['max_depth'],
                   params['min_child_weight'],
                   params['colsample_bytree'],params['subsample'],
                   params['gamma'],best_iteration])

scores = pd.DataFrame(scores,columns=['score','eta','max_depth',
                                      'min_child_weight',
                                      'colsample_bytree','subsample',
                                      'gamma','best_iteration'])
best_gamma = scores.sort_values(by='score',ascending=True)['gamma'].values[0]
print ('best gamma is', best_gamma)

####
from bayes_opt import BayesianOptimization
%%time
def xgb_evaluate(min_child_weight,
                 colsample_bytree,
                 max_depth,
                 subsample,
                 gamma):
    params = dict()
    params['objective'] = 'multi:softmax'
    params['num_class'] = 10
    params['eta'] = 0.1
    params['max_depth'] = int(max_depth)
    params['min_child_weight'] = min_child_weight
    params['colsample_bytree'] = colsample_bytree
    params['subsample'] = subsample
    params['gamma'] = gamma
    params['verbose_eval'] = True

    cv_result = xgb.cv(params, xg_train,
                       num_boost_round=100000,
                       nfold=5,
                       metrics={'merror'},
                       seed=1234,
                       callbacks=[xgb.callback.early_stop(50)])

    return -cv_result['test-merror-mean'].min()


xgb_BO = BayesianOptimization(xgb_evaluate,
                              {'max_depth': (2, 4),
                               'min_child_weight': (0.05, 0.6),
                               'colsample_bytree': (0.9, 1),
                               'subsample': (0.9, 1),
                               'gamma': (0.001, 0.1)
                              }
                             )

xgb_BO.maximize(init_points=5, n_iter=20)

xgb_BO_scores = pd.DataFrame(xgb_BO.res['all']['params'])
xgb_BO_scores['score'] = pd.DataFrame(xgb_BO.res['all']['values'])
xgb_BO_scores = xgb_BO_scores.sort_values(by='score',ascending=False)
xgb_BO_scores.head()

#### to find best_iteration
%%time
params = dict()
params['objective'] = 'multi:softmax'
params['num_class'] = 10
params['eta'] = 0.1
params['max_depth'] = int(xgb_BO_scores['max_depth'][0])
params['min_child_weight'] = int(xgb_BO_scores['min_child_weight'][0])
params['colsample_bytree'] = xgb_BO_scores['colsample_bytree'][0]
params['subsample'] = xgb_BO_scores['subsample'][0]
params['gamma'] = xgb_BO_scores['gamma'][0]
params['seed']=1234

cv_results = xgb.cv(params,
                    xg_train,
                    num_boost_round=1000000,
                    nfold=5,
                    metrics={'merror'},
                    seed=1234,
                    callbacks=[xgb.callback.early_stop(50)],
                    verbose_eval=50
                   )

best_score = cv_results['test-merror-mean'].min()
best_iteration = len(cv_results)


#### result display
print("Tuned Model:")
xgb_BO_scores = pd.read_csv("xgb_BO_scores.csv")
# xgb_BO_scores = pd.read_csv("xgb_BO_scores_new.csv")
params = dict()
params['max_depth'] = int(xgb_BO_scores['max_depth'][0])
params['min_child_weight'] = int(xgb_BO_scores['min_child_weight'][0])
params['colsample_bytree'] = xgb_BO_scores['colsample_bytree'][0]
params['subsample'] = xgb_BO_scores['subsample'][0]
params['gamma'] = xgb_BO_scores['gamma'][0]
```

```
start = time.time()
model_tuned = xgb.XGBClassifier(learning_rate = 0.2
                        , objective = 'multi:softmax'
                        , n_estimators = 200
                        , max_depth = params['max_depth']
                        , min_child_weight = params['min_child_weight']
                        , subsample = params['subsample']
                        , colsample_bytree = params['colsample_bytree']
                        , gamma = params['gamma']
                        , seed = 1234
                        , nthread = -1
                       )

model_tuned.fit(X_train, y_train)

print ("Training finished in %d seconds." % (time.time()-start))

pred = model_tuned.predict(X_test)
y_label = y_test.values
print ('classification error=%f' % (sum([pred[i] != y_label[i]
               for i in range(len(y_label))]) / float(len(y_label)) ))

# Training finished in 11 seconds.
# classification error=0.253988
```

3.6 2-layer neural network

```
##### Change labels using one-hot coding #####
def one_hot_encode(labels):
    n_labels = len(labels)
    n_unique_labels = len(np.unique(labels))
    one_hot_encode = np.zeros((n_labels,n_unique_labels))
    one_hot_encode[np.arange(n_labels), labels] = 1
    return one_hot_encode

tr_features = X_train
ts_features = X_test
tr_labels = one_hot_encode(y_train)
ts_labels = one_hot_encode(y_test)


##### Define parameters #####
training_epochs = 1000
n_dim = tr_features.shape[1]
n_classes = 10
n_hidden_units_one = 5
n_hidden_units_two = 5
sd = 1 / np.sqrt(n_dim)
learning_rate = 0.00001


##### Build Neural network #####
```

```python
X = tf.placeholder(tf.float32,[None,n_dim])
Y = tf.placeholder(tf.float32,[None,n_classes])

W_1 = tf.Variable(tf.random_normal([n_dim,n_hidden_units_one], mean = 0, stddev=sd))
b_1 = tf.Variable(tf.random_normal([n_hidden_units_one], mean = 0, stddev=sd))
h_1 = tf.nn.tanh(tf.matmul(X,W_1) + b_1)

W_2 = tf.Variable(tf.random_normal([n_hidden_units_one,n_hidden_units_two],
mean = 0, stddev=sd))
b_2 = tf.Variable(tf.random_normal([n_hidden_units_two], mean = 0, stddev=sd))
h_2 = tf.nn.sigmoid(tf.matmul(h_1,W_2) + b_2)

W = tf.Variable(tf.random_normal([n_hidden_units_two,n_classes], mean = 0, stddev=sd))
b = tf.Variable(tf.random_normal([n_classes], mean = 0, stddev=sd))
y_ = tf.nn.softmax(tf.matmul(h_2,W) + b)

init = tf.global_variables_initializer()
#init = tf.initialize_all_variables()


##### Define cost function #####
cost_function = -tf.reduce_sum(Y * tf.log(y_))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)

correct_prediction = tf.equal(tf.argmax(y_,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))


##### Calculate accuracy #####
cost_history = np.empty(shape=[1],dtype=float)
y_true, y_pred = None, None
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(training_epochs):
        _,cost = sess.run([optimizer,cost_function],feed_dict={X:tr_features,Y:tr_labels})
        cost_history = np.append(cost_history,cost)

    y_pred = sess.run(tf.argmax(y_,1),feed_dict={X: ts_features})
    y_true = sess.run(tf.argmax(ts_labels,1))
    print("Test accuracy: ",round(sess.run(accuracy,
        feed_dict={X: ts_features,Y: ts_labels}),3))

fig = plt.figure(figsize=(10,8))
plt.plot(cost_history)
plt.axis([0,training_epochs,0,np.max(cost_history)])
plt.show()

#p,r,f,s = precision_recall_fscore_support(y_true, y_pred, average="micro")
#print("F-Score:", round(f,3))
```

- The accuracy for 2-layer neural network is 26%.

Step 4: Result comparison plot

In this step, we drew different plots to compare the result.

The final result could be found in Step 5: Summary.

```
df <- data.frame(model = rep(c('xgb', 'lr', 'svm (linear)',
                               'gbm', 'rf', 'svm (rbf)', 'svm (poly)'), each=10),
                 class = c("air_conditioner", "car_horn", "children_playing",
                           "dog_bark", "drilling", "enginge_idling",
                           "gun_shot", "jackhammer", "siren", "street_music"),
                 #class = as.character(rep(c(0:9),5)),
                 accuracy=c(0.36,0.69,0.76,0.86,0.67,0.43,0.89,0.33,0.47,0.76,
                            0.18, 0.51, 0.56, 0.82, 0.5, 0.48, 0.66, 0.17, 0.53, 0.64,
                            0.18, 0.46, 0.80, 0.36, 0.47, 0.48, 0.49, 0.33, 0.33, 0.71,
                            0.36, 0.57, 0.73, 0.88, 0.6, 0.52, 0.74, 0.17, 0.47, 0.73,
                            0.18, 0.57, 0.73, 0.90, 0.50, 0.35, 0.83, 0.33, 0.33, 0.69,
                            0, 0, 0, 1, 0, 0, 0.03, 0, 0, 0,
                            0.27, 0.57, 0.61, 0.58, 0.37, 0.17, 0.74, 0.42, 0.4, 0.51))
df$model<- factor(df$model, levels = c('xgb', 'lr', 'svm (linear)',
                                       'gbm', 'rf', 'svm (rbf)', 'svm (poly)'))
df$class<- factor(df$class, levels = c("air_conditioner", "car_horn", "children_playing",
                                       "dog_bark", "drilling", "enginge_idling",
                                       "gun_shot", "jackhammer", "siren", "street_music"))


##### barplot #####
ggplot(data=df, aes(x=class, y=accuracy, fill=model)) +
  geom_col(position='dodge', width=0.7, alpha=0.8)+
  ggtitle("Accuracy for 10 different classes audios",
          subtitle="(with 5 different models)")+
  ylim(0, 1)+
  labs(x = "", y = "")+
  coord_flip()+
  theme_grey(11)+
  theme(legend.title = element_blank())+
  scale_fill_manual(values = c("#ee4035", "#7bc043", "#f37736",
                               "#fcec4d", "#0392cf", "#c79ddc", "#9a8262"))+
  theme(axis.text.y = element_text(face="bold", size=10),
        plot.title = element_text(face="bold", size=16),
        plot.subtitle = element_text(face="italic", size=10))

##### bubble plot #####
ggplot(df, aes(x=class, y=accuracy, color=model))+
  geom_point(size=5, alpha=0.7)+
  ggtitle("Accuracy for 10 different classes audios",
          subtitle="(with 5 different models)")+
  ylim(0, 1)+
  labs(x = "", y = "")+
  coord_flip()+
  scale_color_manual(values = c("#ee4035", "#7bc043", "#f37736",
                                "#fcec4d", "#0392cf", "#c79ddc", "#9a8262"))+
  theme_grey(11)+
  theme(legend.title=element_blank())+
  theme(axis.text.y = element_text(face="bold", size=10),
        plot.title = element_text(face="bold", size=16),
        plot.subtitle = element_text(face="italic", size=10))
```

Step 5: Summary

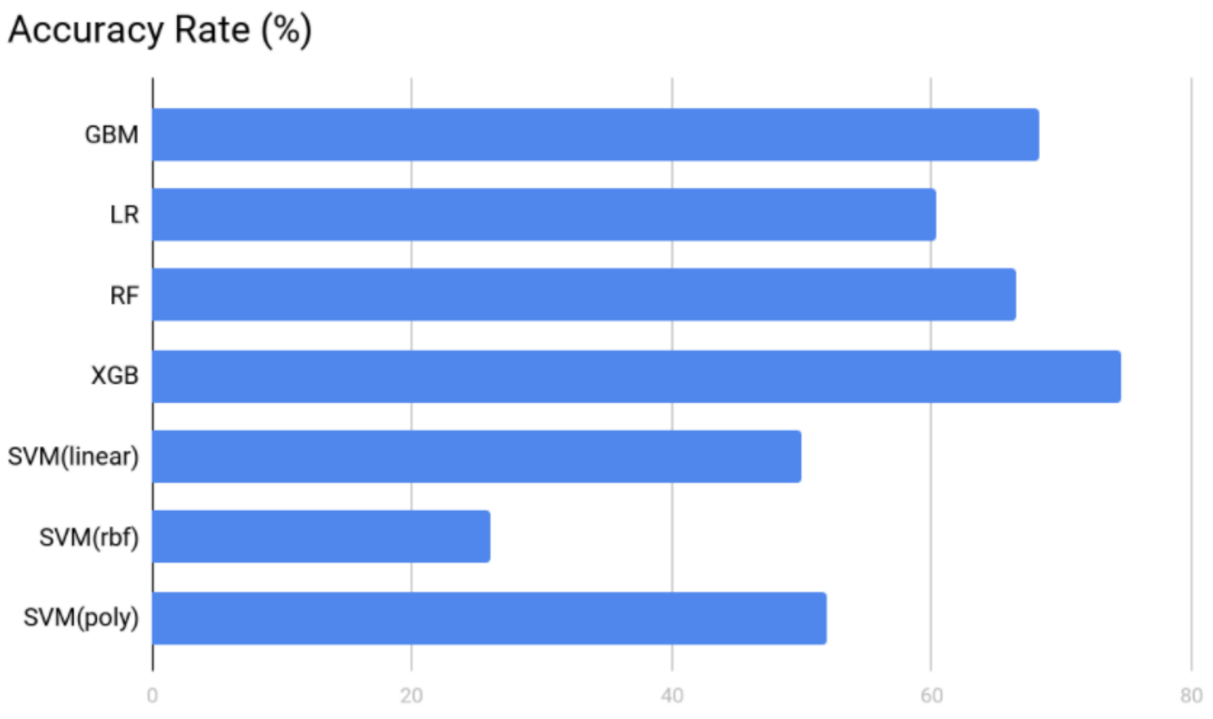
5.1 Result analysis

As we can see, the best model is XGB, with classification accuracy 74.61%, and training time 13 seconds.

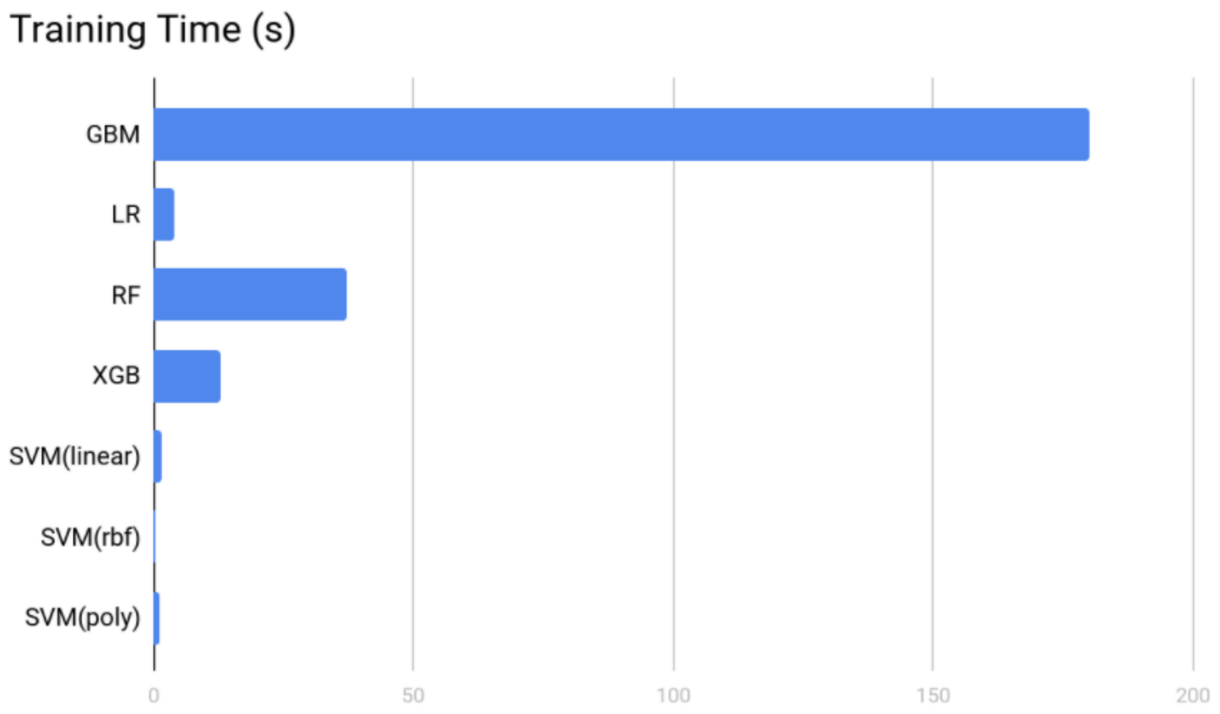
And some other intersting findings:

- 1.Except SVM with RBF kernel, dog barks, gun shots and street musics can be well recognized with most of the models.
- 2.For SVM with RBF kernel, it perfectly recognized all dog barks, but did most badly for other kinds of sound.
- 3.For jackhammer and air condition, almost all the models performed terribly, but interestingly had roughly the same low-accuracy for them.
- 4.While the accuracy rate differs among different models, the relevant performances on different classes of each model are quite similar.
- 5.Traning time of most models is short, except GBM.

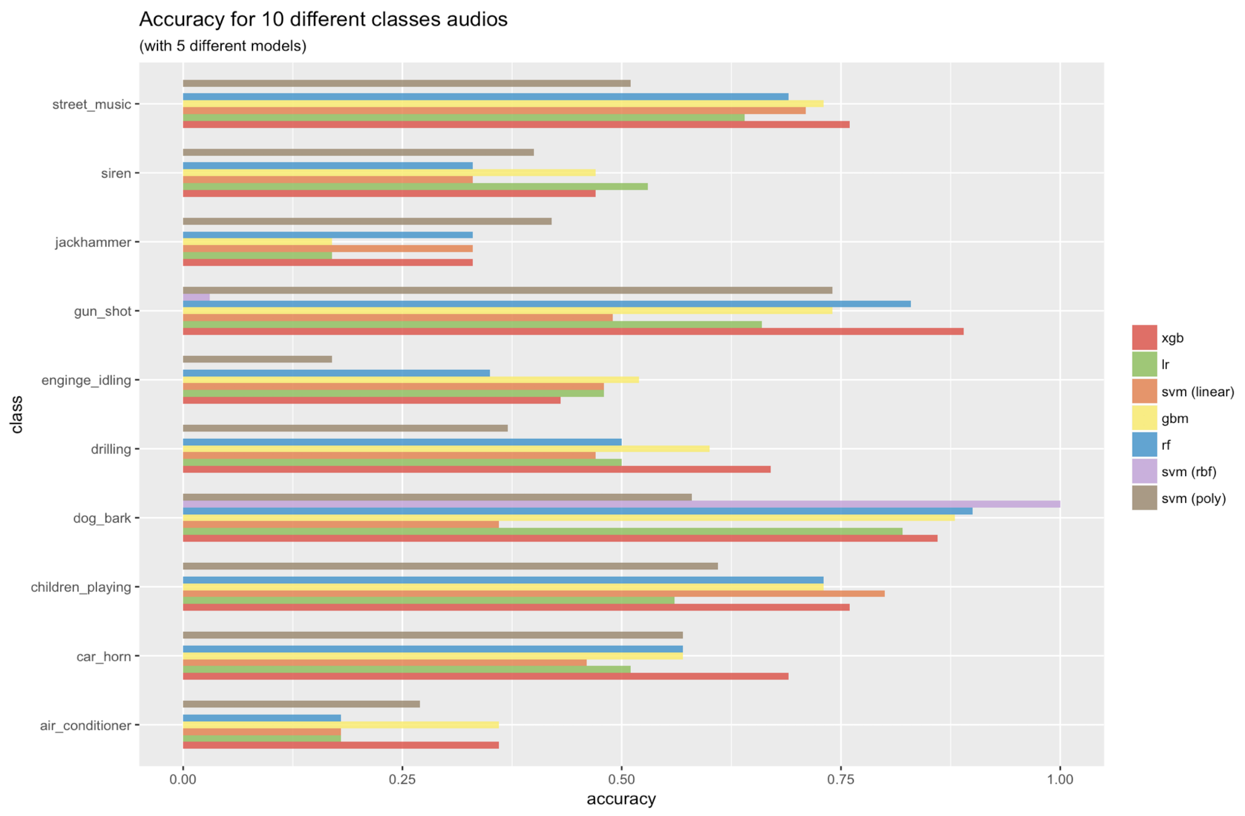
Total accuracy for each model



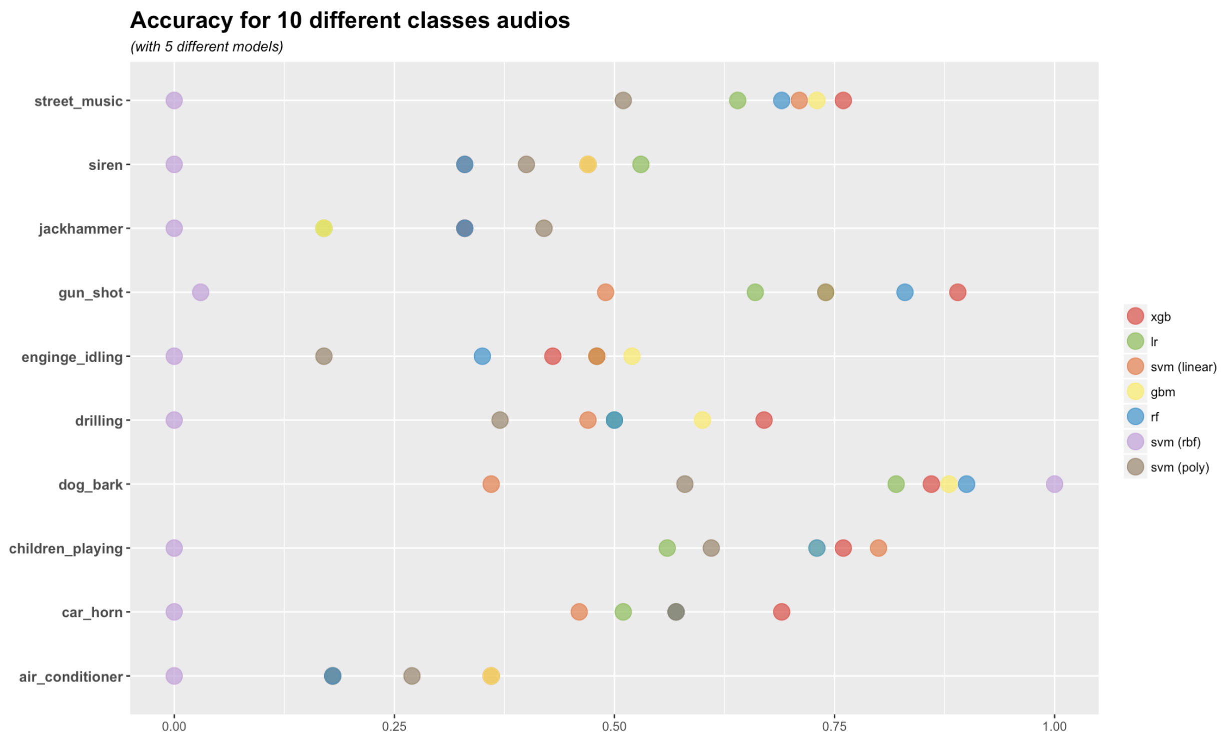
Training time comparison for each model



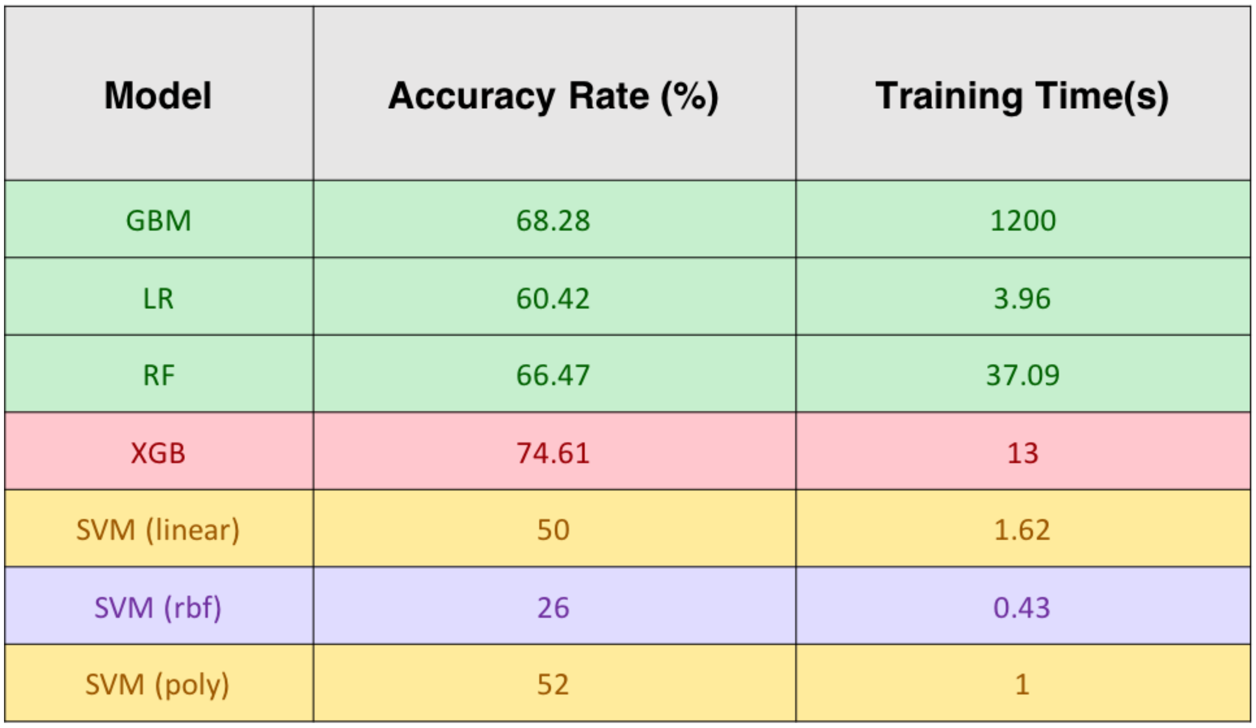
Accuracy for each class for each model (barplot)



Accuracy for each class for each model (bubbleplot)



Accuracy and training time table



Model	Accuracy Rate (%)	Training Time(s)
GBM	68.28	1200
LR	60.42	3.96
RF	66.47	37.09
XGB	74.61	13
SVM (linear)	50	1.62
SVM (rbf)	26	0.43
SVM (poly)	52	1

5.2 Improvement

Our accuracy rate is not that satisfying, the reasons could be:

- The size training data is too small, we only have 1102 training data for 10 class classification problem.
- The number of class is too much.

So there are still many improvements we can make on this project:

- Use a larger dataset on the official website which contains more than 8000 audio records with these 10 different classes. Intuitively, more data available, more accurate for model performs.
- Set up a more complex CNN with at least 10 layers, and with some other techniques, such as adding zero-padding or dropout layer to classify those audios.
- Try to use different combinations within those five kinds of features we extracted. We have assumed that perhaps some kinds of the features are much more outstanding than others. So why not try to ignore those “unuseful” features, at some points, reduce dimensions, and train the model on the subset of the features. The result could be exciting, or not.
- Try to extract other kinds of features.