MSc in AI
NCSR Demokritos - University of Piraeus

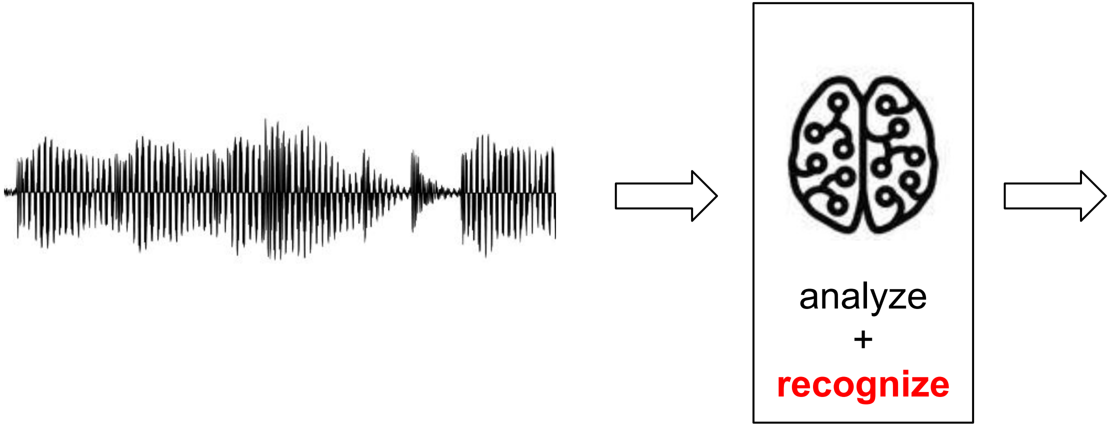
Course: **Machine Learning for Multimodal Data**

Lesson 3
Audio Classification / Regression

Theodoros Giannakopoulos

Audio Analysis Goal (again)

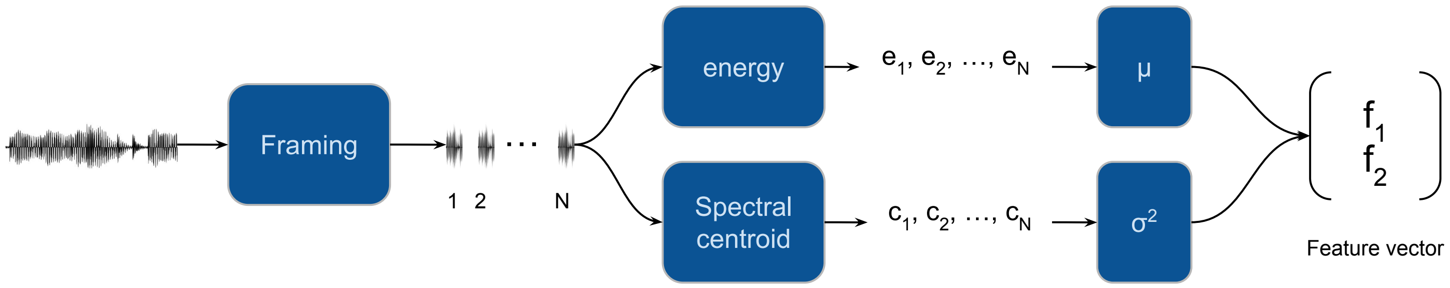
- Goal: extract high-level descriptions from raw audio signals (sounds)
- Using:
  - signal analysis: to extract features and representations
  - machine learning (supervised or unsupervised) to train models and to discover patterns
- Speech / Music / Audio
- Also referred to as “machine listening”



music
group:arctic_monkeys
genre:indie
genre:post_punk
singer:alex_turner
emotion_high_arousal
emotion_negative_valence
bpm:170
...

Audio representation

- Each audio segment represented by:
  - short-term feature sequences
  - segment-level audio feature statistics
  - statistics extracted on each short-term feature sequence
  - e.g: 13 MFCCs + (μ, σ^2, percentile_25, percentile_75) = 13 x 4 = 52 feature segment statistics
- One feature (statistic) vector for each segment
- Segments
  - Pre-segmented (using a criterion of content homogeneity)
  - result fix-sized signal segmentation



Long-term averaging

- For long recordings of homogeneous content (e.g. songs):
  - long-term average on the feature statistics
  - long recordings represented by a feature vector of the same dimension as their segments
- Makes sense in long-term stationary data
- Does not capture long-term temporal changes (long-term temporal information lost)

Classification / Regression - fundamentals

- Classification
  - Goal: given a set (feature vector, label) pairs learn a “mapping” from feature representations to label
  - Training data
  - Testing data
  - Multi-class:
    - not directly available in all classification methods
    - indirect methods: one vs all , one vs one
- Regression
  - Predict a continuous target value
  - E.g.:
    - Sound quality (1,..5)
    - Valence (-1, .., 1)
  - Ground truth can also be quantized in fewer values

Classifier Evaluation

- Split dataset to train and test subsets
  - Train the classifier using the train dataset
  - Predict labels using the test and compute performance measures
- Several types of splits:
  - Random shuffling
    - Keep a percentage of samples (e.g. 80%) for train
    - Rest (e.g. 20%) for testing
    - Compute performance measures
    - Repeat and aggregate (except if num of sample is very high)
  - K-fold cross-validation:
    - Split data into k folds (groups of samples)
    - Train using k-1 folds
    - Test and compute performance measures using 1 remaining
    - Repeat for all folds and aggregate
  - Leave-one-out ...

- Why:
  - Avoid overfitting
  - Use most of the data for training
- Parameter tuning:
  - Evaluate the classifier for different params
  - May also overfit on the test data (best param achieved only on the test data)
  - Solution: use a validation set to tune the param and then test the final selected model on the test set

Folds can be randomly selected but ***also predefined in the dataset***

In a speech emotion recognition problem folds can be either speakers or recording sessions → achieve speaker-independent (or session-independent) models

Classification Performance Measures

- Confusion matrix: CM(i,j) counts samples that belong to class i and classified to class j
- Recall: fraction of samples correctly classified to class i
- Precision: the fraction of samples correctly classified to class i if we take into account the total number of samples that were classified to that class i
- F1: harmonic mean of recall and precision
- Defined per class
- Also average measures
- Accuracy: proportion of data correctly classified
- *Depending on the application: some classes may require higher recall/precision rates*

$$CM(i,j), i = 1, .., N_c$$

$$Re(i) = \frac{CM(i,i)}{\sum_{m=1}^{N_c} CM(i,m)}$$

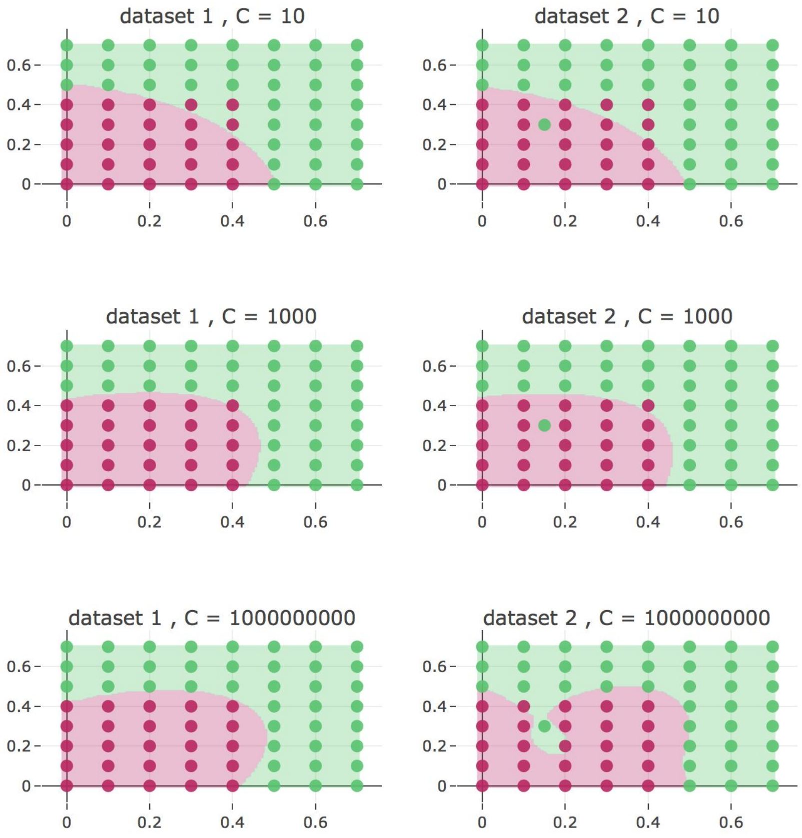
$$Pr(i) = \frac{CM(i,i)}{\sum_{m=1}^{N_c} CM(m,i)}$$

$$F_1(i) = \frac{2Re(i)Pr(i)}{Pr(i) + Re(i)}$$

Classification Overfitting

- The training method is overwhelmingly adjusted on the training data
- Very high performance on training data
- Significantly lower performance on testing data
- Method actually "learns noise"
- Usually happens in nonparametric and nonlinear models $\longrightarrow$ more flexibility in learning $\longrightarrow$ fit almost all training samples
- Underfitting poor performance also on the training data
- Goal:
    - Find a model that generalizes to testing data (not overfitted) but has also learnt the training data "good enough"
    - Always use cross validation!

Classification Overfitting - Example 21



- C parameter:
  - related to the cost function of the SVM
  - higher C ⟶ cost of misclassified samples is higher ⟶ fewer misclassified samples in the training data
  - **Overfitting**
- Example:
  - code in example21.py
  - binary classification task
  - 2nd dataset:
    - a new sample added from the **green** class
    - makes the discrimination task harder
  - low C: underfitting in both datasets
  - mid C: ok for both datasets
  - high C: overfitted especially for 2nd dataset

Classification task 1: speech vs music

- Goal (twofold):
  - break an audio stream to segments (**segmentation**)
  - label each segment as either speech or music (**classification**)
- Segmentation (covered next):
  - Joint vs Sequential
  - Supervised vs Unsupervised
- Classification task:
  - Input: pre-segmented segments
  - Output: audio class label (speech / music)
- Examples:
  - Review of typical approaches [1]
  - Deep learning approaches [2]
  - Dimensionality reduction [3]
  - Probabilistic models [4]

[1] Pikrakis, Aggelos, Theodoros Giannakopoulos, and Sergios Theodoridis. "An overview of speech/music discrimination techniques in the context of audio recordings." *Multimedia Services in Intelligent Environments*. Springer, Berlin, Heidelberg, 2008. 81-102.

[2] Papakostas, Michalis, and Theodoros Giannakopoulos. "Speech-Music Discrimination Using Deep Visual Feature Extractors." *Expert Systems with Applications* (2018).

[3] Alexandre-Cortizo, Enrique, Manuel Rosa-Zurera, and Francisco Lopez-Ferreras. "Application of fisher linear discriminant analysis to speech/music classification." *Computer as a Tool, 2005. EUROCON 2005. The International Conference on*. Vol. 2. IEEE, 2005.

[4] Pikrakis, Aggelos, Theodoros Giannakopoulos, and Sergios Theodoridis. "A speech/music discriminator of radio recordings based on dynamic programming and bayesian networks." *IEEE Transactions on Multimedia* 10.5 (2008): 846-857.

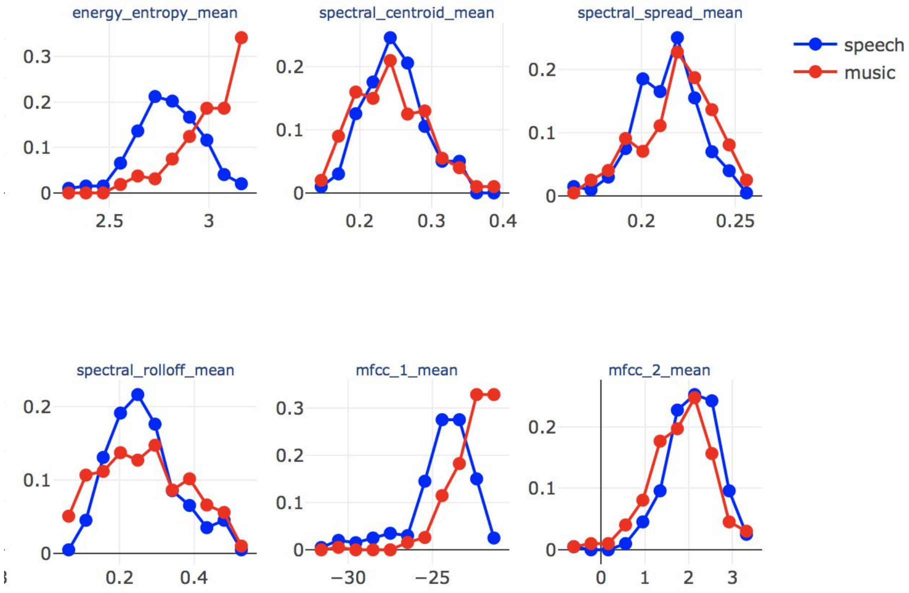
[5] Yang, Wanzhao, et al. "An RNN-Based Speech-Music Discrimination Used for Hybrid Audio Coder." *International Conference on Multimedia Modeling*. Springer, Cham, 2018.

Classification task 1: speech vs music (Example 18)

```python
"""!
@brief Example 18
@details speech music classification example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np, plotly, plotly.graph_objs as go
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
from sklearn.svm import SVC
import utilities as ut
name_1, name_2 = "mfcc_3_std", "energy_entropy_mean"
layout = go.Layout(title='Speech Music Classification Example',
                   xaxis=dict(title=name_1,), yaxis=dict(title=name_2,))

if __name__ == '__main__':
    # get features from folders (all classes):
    f1, _, fn1 = dW("../data/speech_music/speech", 1, 1, 0.1, 0.1)
    f2, _, fn2 = dW("../data/speech_music/music", 1, 1, 0.1, 0.1)
    # plot histograms for each feature and normalize
    ut.plot_feature_histograms([f1, f2], fn1, ["speech", "music"])
    # concatenate features to extract overall mean and std ...
    f1 = np.array([f1[:, fn1.index(name_1)], f1[:, fn1.index(name_2)]]).T
    f2 = np.array([f2[:, fn1.index(name_1)], f2[:, fn1.index(name_2)]]).T
    f = np.concatenate((f1, f2), axis = 0)
    mean, std = f.mean(axis=0), np.std(f, axis=0)
    f1 = (f1 - mean) / std; f2 = (f2 - mean) / std; f = (f - mean) / std
    # plot selected 2D features
    plt1 = go.Scatter(x=f1[:, 0], y=f1[:, 1], mode='markers', name="speech")
    plt2 = go.Scatter(x=f2[:, 0], y=f2[:, 1], mode='markers', name="music")
    # get classification decisions for grid
    y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0])))
    cl = SVC(kernel='rbf', C=0.1)
    cl.fit(f, y)
    x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.01)
    y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.01)
    xx, yy = np.meshgrid(x_, y_)
    Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False)
    plotly.offline.plot(go.Figure(data=[plt1, plt2, cs], layout=layout),
                        filename="temp2.html", auto_open=True)
```

Extract all segment feature statistics supported by pyAudioAnalysis (one feature matrix for each directory of wavs / class)

Plot one histogram for each feature to check respective discrimination ability



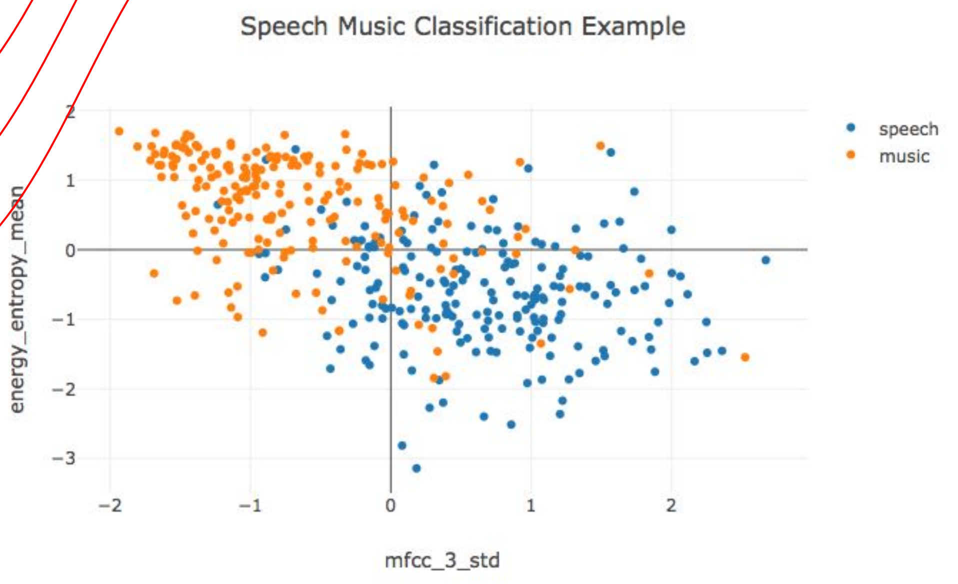
Classification task 1: speech vs music (Example 18)

```python
"""!
@brief Example 18
@details speech music classification example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np, plotly, plotly.graph_objs as go
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
from sklearn.svm import SVC
import utilities as ut
name_1, name_2 = "mfcc_3_std", "energy_entropy_mean"
layout = go.Layout(title='Speech Music Classification Example',
                   xaxis=dict(title=name_1,), yaxis=dict(title=name_2,))

if __name__ == '__main__':
   # get features from folders (all classes):
   f1, _, fn1 = dW("../data/speech_music/speech", 1, 1, 0.1, 0.1)
   f2, _, fn2 = dW("../data/speech_music/music", 1, 1, 0.1, 0.1)
   # plot histograms for each feature and normalize
   ut.plot_feature_histograms([f1, f2], fn1, ["speech", "music"])
   # concatenate features to extract overall mean and std ...
   f1 = np.array([f1[:, fn1.index(name_1)], f1[:, fn1.index(name_2)]]).T
   f2 = np.array([f2[:, fn1.index(name_1)], f2[:, fn1.index(name_2)]]).T
   f = np.concatenate((f1, f2), axis = 0)
   mean, std = f.mean(axis=0), np.std(f, axis=0)
   f1 = (f1 - mean) / std; f2 = (f2 - mean) / std; f = (f - mean) / std
   # plot selected 2D features
   plt1 = go.Scatter(x=f1[:, 0], y=f1[:, 1], mode='markers', name="speech")
   plt2 = go.Scatter(x=f2[:, 0], y=f2[:, 1], mode='markers', name="music")
   # get classification decisions for grid
   y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0])))
   cl = SVC(kernel='rbf', C=0.1)
   cl.fit(f, y)
   x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.01)
   y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.01)
   xx, yy = np.meshgrid(x_, y_)
   Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
   cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False)
   plotly.offline.plot(go.Figure(data=[plt1, plt2, cs], layout=layout),
                       filename="temp2.html", auto_open=True)
```

Select two particular features

Extract mean / std for all features (both classes) and **normalize**

Visualize 2D feature vectors(as points) for both classes



Classification task 1: speech vs music (Example 18)

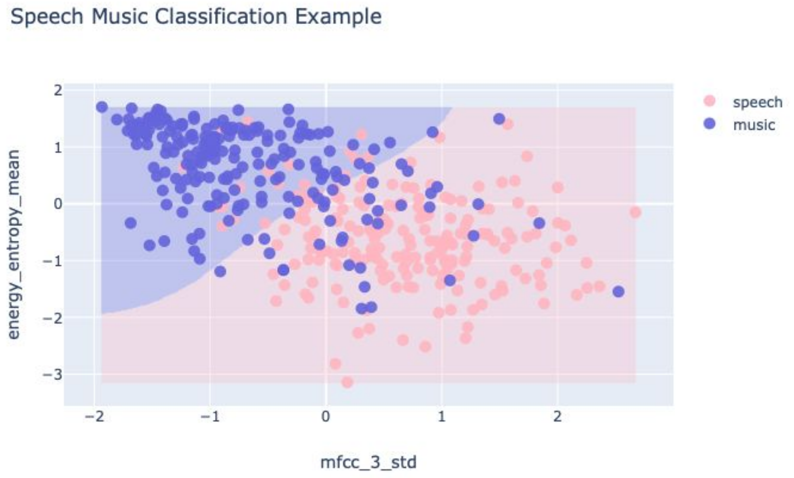
```python
"""!
@brief Example 18
@details speech music classification example
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np, plotly, plotly.graph_objs as go
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
from sklearn.svm import SVC
import utilities as ut
name_1, name_2 = "mfcc_3_std", "energy_entropy_mean"
layout = go.Layout(title='Speech Music Classification Example',
                   xaxis=dict(title=name_1,), yaxis=dict(title=name_2,))

if __name__ == '__main__':
   # get features from folders (all classes):
   f1, _, fn1 = dW("../data/speech_music/speech", 1, 1, 0.1, 0.1)
   f2, _, fn2 = dW("../data/speech_music/music", 1, 1, 0.1, 0.1)
   # plot histograms for each feature and normalize
   ut.plot_feature_histograms([f1, f2], fn1, ["speech", "music"])
   # concatenate features to extract overall mean and std ...
   f1 = np.array([f1[:, fn1.index(name_1)], f1[:, fn1.index(name_2)]]).T
   f2 = np.array([f2[:, fn1.index(name_1)], f2[:, fn1.index(name_2)]]).T
   f = np.concatenate((f1, f2), axis = 0)
   mean, std = f.mean(axis=0), np.std(f, axis=0)
   f1 = (f1 - mean) / std; f2 = (f2 - mean) / std; f = (f - mean) / std
   # plot selected 2D features
   plt1 = go.Scatter(x=f1[:, 0], y=f1[:, 1], mode='markers', name="speech",
                     marker=dict(size=10,color='rgba(255, 182, 193, .9)',))
   plt2 = go.Scatter(x=f2[:, 0], y=f2[:, 1], mode='markers', name="music",
                     marker=dict(size=10,color='rgba(100, 100, 220, .9)',))
   # get classification decisions for grid
   y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0])))
   cl = SVC(kernel='rbf', C=0.1)
   cl.fit(f, y)
   x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.01)
   y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.01)
   xx, yy = np.meshgrid(x_, y_)
   Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
   cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False,
                   colorscale= [[0, 'rgba(255, 182, 193, .3)'],
                                [1, 'rgba(100, 100, 220, .3)']])
   plotly.offline.plot(go.Figure(data=[plt1, plt2, cs], layout=layout),
```

Create class labels (y=0 for speech, y=1 for music)

Train SVM using feature matrix (f, [n_samples x 2]) and class labels (y, [n_samples x 1])

Generate mesh grid (2D space) and classify (using the trained SVM) each point in the grid

Create a heatmap based on the classification predictions



Classification task 2: musical genre classification

- Musical genre classification: applied topic in ***music information retrieval (MIR)*** for over 15 years [1]
- Important task in MIR for educational purposes and intro courses
- **However**
  - ill defined task
  - inherent difficulty in establishing definitions in musical genres
  - more profitable to pursue research in music similarity and generic tag classification

[1] Tzanetakis, George, and Perry Cook. "Musical genre classification of audio signals." IEEE Transactions on speech and audio processing 10.5 (2002): 293-302.

[2] Choi, Keunwoo, et al. "Convolutional recurrent neural networks for music classification." 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2017.

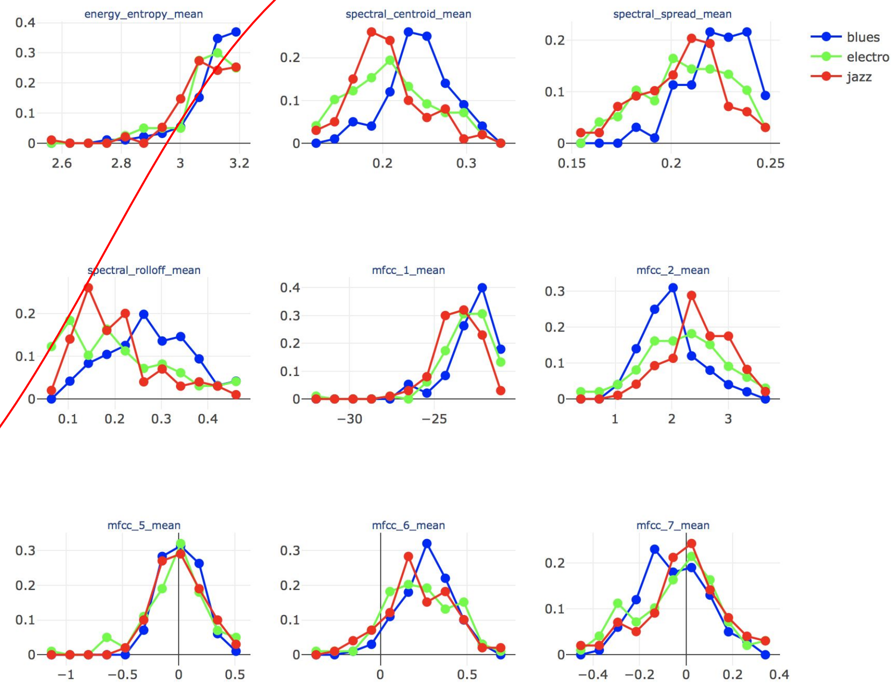
[3] Costa, Yandre MG, Luiz S. Oliveira, and Carlos N. Silla Jr. "An evaluation of convolutional neural networks for music classification using spectrograms." Applied soft computing 52 (2017): 28-38.

Classification task 2: musical genre classification - Example 19

```
...
name_1, name_2 = "spectral_entropy_std", "chroma_std_std"
layout = go.Layout(title='Musical Genre Classification Example',
                   xaxis=dict(title=name_1,), yaxis=dict(title=name_2,))

if __name__ == '__main__':
    # get features from folders (all classes):
    f1, _, fn1 = dW("../data/musical_genres_8k/blues", 2, 1, 0.1, 0.1)
    f2, _, fn2 = dW("../data/musical_genres_8k/electronic", 2, 1, 0.1, 0.1)
    f3, _, fn3 = dW("../data/musical_genres_8k/jazz", 2, 1, 0.1, 0.1)
    # plot histograms for each feature and normalize
    ut.plot_feature_histograms([f1, f2, f3], fn1, ["blues", "electro", "jazz"])
    # concatenate features to extract overall mean and std ...
    f1 = np.array([f1[:, fn1.index(name_1)], f1[:, fn1.index(name_2)]]).T
    f2 = np.array([f2[:, fn1.index(name_1)], f2[:, fn1.index(name_2)]]).T
    f3 = np.array([f3[:, fn1.index(name_1)], f3[:, fn1.index(name_2)]]).T
    f = np.concatenate((f1, f2, f3), axis = 0)
    mean, std = f.mean(axis=0), np.std(f, axis=0)
    f1 = (f1 - mean) / std; f2 = (f2 - mean) / std;  f3 = (f3 - mean) / std
    f = (f - mean) / std
    # plot selected 2D features
    plt1 = go.Scatter(x=f1[:, 0], y=f1[:, 1], mode='markers', name="blues",
                      marker=dict(size=10,color='rgba(255, 182, 193, .9)',))
    plt2 = go.Scatter(x=f2[:, 0], y=f2[:, 1], mode='markers', name="electronic",
                      marker=dict(size=10,color='rgba(100, 182, 150, .9)',))
    plt3 = go.Scatter(x=f3[:, 0], y=f3[:, 1], mode='markers', name="jazz",
                      marker=dict(size=10,color='rgba(100, 100, 220, .9)',))
    # get classification decisions for grid
    y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0]),
                       2 * np.ones(f3.shape[0])))
    cl = SVC(kernel='rbf', C=1)
    cl.fit(f, y)
    x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.01)
    y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.01)
    xx, yy = np.meshgrid(x_, y_)
    Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape) / 2
    cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False,
                   colorscale= [[0, 'rgba(255, 182, 193, .3)'],
                                [0.5, 'rgba(100, 182, 150, .3)'],
                                [1, 'rgba(100, 100, 220, .3)']])
    plotly.offline.plot(go.Figure(data=[plt1, plt2, plt3, cs], layout=layout),
                        filename="temp2.html", auto_open=True)
```

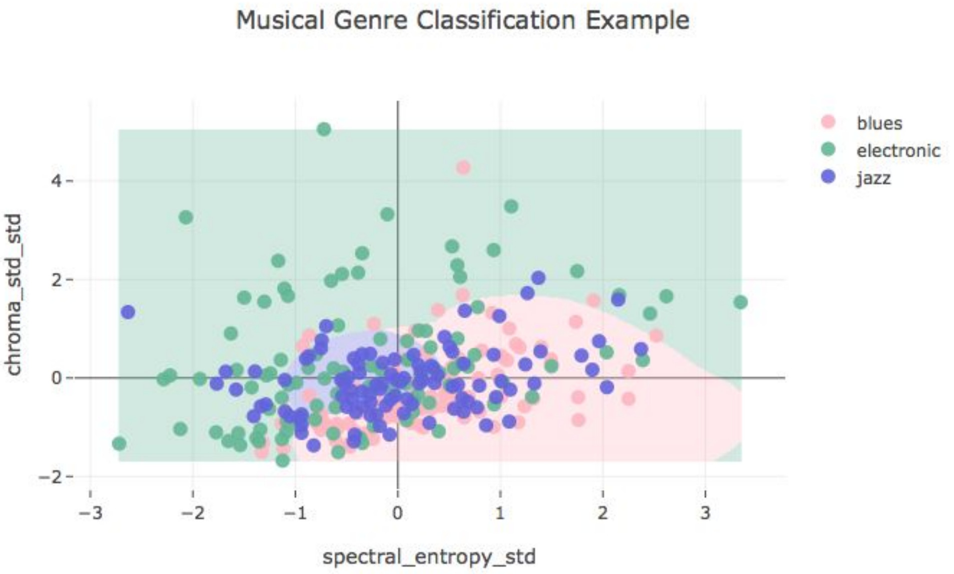
Same as Example18.py but three classes

Use colormaps and predefined colors



Classification task 2: musical genre classification - Example 19

```python
...
name_1, name_2 = "spectral_entropy_std", "chroma_std_std"
layout = go.Layout(title='Musical Genre Classification Example',
                   xaxis=dict(title=name_1,), yaxis=dict(title=name_2,))

if __name__ == '__main__':
   # get features from folders (all classes):
   f1, _, fn1 = dW("../data/musical_genres_8k/blues", 2, 1, 0.1, 0.1)
   f2, _, fn2 = dW("../data/musical_genres_8k/electronic", 2, 1, 0.1, 0.1)
   f3, _, fn3 = dW("../data/musical_genres_8k/jazz", 2, 1, 0.1, 0.1)
   # plot histograms for each feature and normalize
   ut.plot_feature_histograms([f1, f2, f3], fn1, ["blues", "electro", "jazz"])
   # concatenate features to extract overall mean and std ...
   f1 = np.array([f1[:, fn1.index(name_1)], f1[:, fn1.index(name_2)]]).T
   f2 = np.array([f2[:, fn1.index(name_1)], f2[:, fn1.index(name_2)]]).T
   f3 = np.array([f3[:, fn1.index(name_1)], f3[:, fn1.index(name_2)]]).T
   f = np.concatenate((f1, f2, f3), axis = 0)
   mean, std = f.mean(axis=0), np.std(f, axis=0)
   f1 = (f1 - mean) / std; f2 = (f2 - mean) / std;  f3 = (f3 - mean) / std
   f = (f - mean) / std
   # plot selected 2D features
   plt1 = go.Scatter(x=f1[:, 0], y=f1[:, 1], mode='markers', name="blues",
                     marker=dict(size=10,color='rgba(255, 182, 193, .9)',))
   plt2 = go.Scatter(x=f2[:, 0], y=f2[:, 1], mode='markers', name="electronic",
                     marker=dict(size=10,color='rgba(100, 182, 150, .9)',))
   plt3 = go.Scatter(x=f3[:, 0], y=f3[:, 1], mode='markers', name="jazz",
                     marker=dict(size=10,color='rgba(100, 100, 220, .9)',))
   # get classification decisions for grid
   y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0]),
                      2 * np.ones(f2.shape[0])))
   cl = SVC(kernel='rbf', C=1)
   cl.fit(f, y)
   x_ = np.arange(f[:, 0].min(), f[:, 0].max(), 0.01)
   y_ = np.arange(f[:, 1].min(), f[:, 1].max(), 0.01)
   xx, yy = np.meshgrid(x_, y_)
   Z = cl.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape) / 2
   cs = go.Heatmap(x=x_, y=y_, z=Z, showscale=False,
                   colorscale= [[0, 'rgba(255, 182, 193, .3)'],
                                [0.5, 'rgba(100, 182, 150, .3)'],
                                [1, 'rgba(100, 100, 220, .3)']])
   plotly.offline.plot(go.Figure(data=[plt1, plt2, plt3, cs], layout=layout),
                       filename="temp2.html", auto_open=True)
```



Classification Evaluation & Performance Measures

```
from sklearn.svm import SVC, SVR
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score

...
def svm_train_evaluate(X, y, k_folds, C=1, use_regressor=False):
    '''
    :param X: Feature matrix
    :param y: Labels matrix
    :param k_folds: Number of folds
    :param C: SVM C param
    :param use_regressor: use svm regression for training (not nominal classes)
    :return: confusion matrix, average f1 measure and overall accuracy
    '''
    # normalize
    mean, std = X.mean(axis=0), np.std(X, axis=0)
    X = (X - mean) / std
    # k-fold evaluation:
    kf = KFold(n_splits=k_folds, shuffle=True)
    f1s, accs, count_cm = [], [], 0
    for train, test in kf.split(X):
        x_train, x_test, y_train, y_test = X[train], X[test], y[train], y[test]
        if not use_regressor:
            cl = SVC(kernel='rbf', C=C)
        else:
            cl = SVR(kernel='rbf', C=C)
        cl.fit(x_train, y_train)
        y_pred = cl.predict(x_test)
        if use_regressor:
            y_pred = np.round(y_pred)
        # update aggregated confusion matrix:
        if count_cm == 0:
            cm = confusion_matrix(y_pred=y_pred, y_true=y_test)
        else:
            cm += (confusion_matrix(y_pred=y_pred, y_true=y_test))
        count_cm += 1
        f1s.append(f1_score(y_pred=y_pred, y_true=y_test, average='micro'))
        accs.append(accuracy_score(y_pred=y_pred, y_true=y_test))
    f1 = np.mean(f1s)
    acc = np.mean(accs)
    return cm, f1, acc
```

`utilities.py`

1. Normalize feature matrix
2. Split folds
3. For each fold
   a. Train an SVM
   b. Predict on test features
   c. Update overall conf matrix
   d. Update list of F1, Accs
4. Return overall confusion matrix, overall recall and overall F1

Note: can also be trained with regression mode (decision used as a classifier, i.e. rounded)

Classification Evaluation & Performance Measures

```python
def compute_class_rec_pre_f1(c_mat):
    '''
    :param c_mat: the [n_class x n_class] confusion matrix
    :return: rec, pre and f1 for each class
    '''
    n_class = c_mat.shape[0]
    rec, pre, f1 = [], [], []
    for i in range(n_class):
        rec.append(float(c_mat[i, i]) / np.sum(c_mat[i, :]))
        pre.append(float(c_mat[i, i]) / np.sum(c_mat[:, i]))
        f1.append(2 * rec[-1] * pre[-1] / (rec[-1] + pre[-1]))
    return rec,  pre, f1

def plotly_classification_results(cm, class_names):
    heatmap = go.Heatmap(z=np.flip(cm, axis=0), x=class_names,
                         y=list(reversed(class_names)),
                         colorscale=[[0, '#4422ff'], [1, '#ff4422']],
                         name="confusin matrix", showscale=False)
    rec, pre, f1 = compute_class_rec_pre_f1(cm)
    mark_prop1 = dict(color='rgba(150, 180, 80, 0.5)',
                      line=dict(color='rgba(150, 180, 80, 1)', width=2))
    mark_prop2 = dict(color='rgba(140, 200, 120, 0.5)',
                      line=dict(color='rgba(140, 200, 120, 1)', width=2))
    mark_prop3 = dict(color='rgba(50, 150, 220, 0.5)',
                      line=dict(color='rgba(50, 150, 220, 1)', width=3))
    b1 = go.Bar(x=class_names,  y=rec, name="rec", marker=mark_prop1)
    b2 = go.Bar(x=class_names,  y=pre, name="pre", marker=mark_prop2)
    b3 = go.Bar(x=class_names,  y=f1, name="f1", marker=mark_prop3)
    figs = plotly.tools.make_subplots(rows=1, cols=2,
                                      subplot_titles=["Confusion matrix",
                                                      "Performance measures"])
    figs.append_trace(heatmap, 1, 1); figs.append_trace(b1, 1, 2)
    figs.append_trace(b2, 1, 2); figs.append_trace(b3, 1, 2)
    plotly.offline.plot(figs, filename="temp.html", auto_open=True)
```

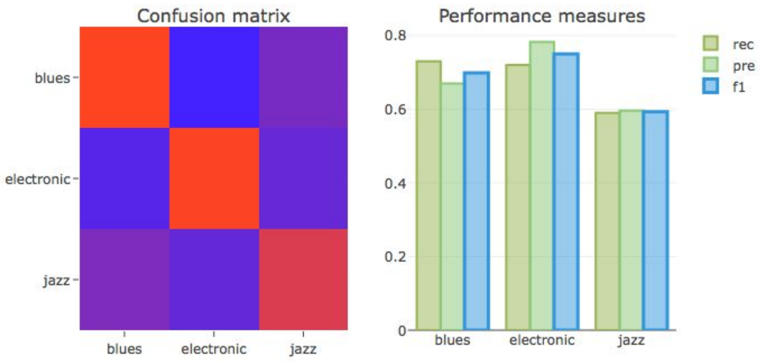
`utilities.py`

1. Get the (overall CM)
2. Compute Recall, Precision and F1 **per class** based on the CM*
3. Show CM and plot of class Recall/Precision/F1

* F1 based on the final CM may be slightly different from the average F1 (i.e. mean of F1 is not exactly the same as F1 based on overall CM) if classes are imbalanced

Classification task 2: musical genre classification - Example 20

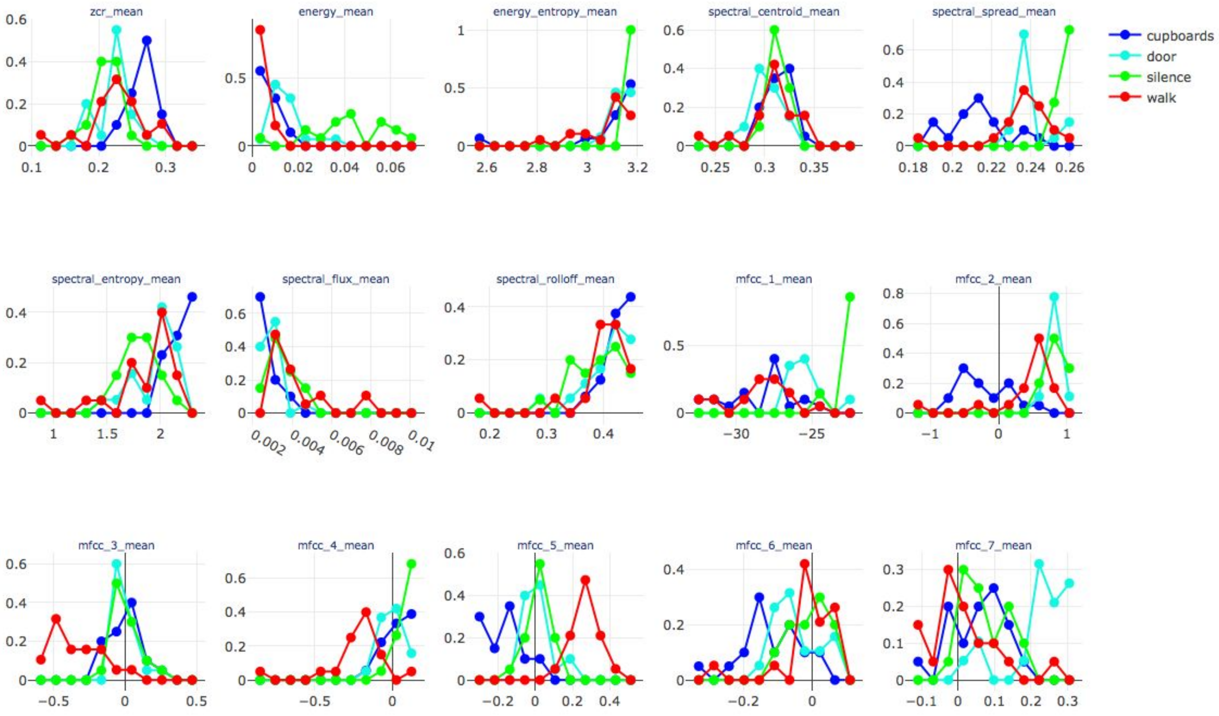
```python
"""!
@brief Example 20
@details Musical genre classification example. Classification performance
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import utilities as ut
from from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW


if __name__ == '__main__':
    # extract features, concatenate feature matrices and normalize:
    mw, stw = 2, .1
    f1, _, fn1 = dW("../data/musical_genres_8k/blues", mw, mw, stw, stw)
    f2, _, fn2 = dW("../data/musical_genres_8k/electronic", mw, mw, stw, stw)
    f3, _, fn3 = dW("../data/musical_genres_8k/jazz", mw, mw, stw, stw)
    x = np.concatenate((f1, f2, f3), axis=0)
    y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0]),
                        2 * np.ones(f2.shape[0])))
    # train svm and get aggregated (average) confusion matrix, accuracy and f1
    cm, acc, f1 = ut.svm_train_evaluate(x, y, 10, C=2)
    # visualize performance measures
    ut.plotly_classification_results(cm, ["blues", "electronic", "jazz"])
    print(acc, f1)
```



Check more on evaluation metrics:
http://scikit-learn.org/stable/modules/model_evaluation.html
And here for cross validation: http://scikit-learn.org/stable/modules/cross_validation.html

Classification task 3: audio event recognition

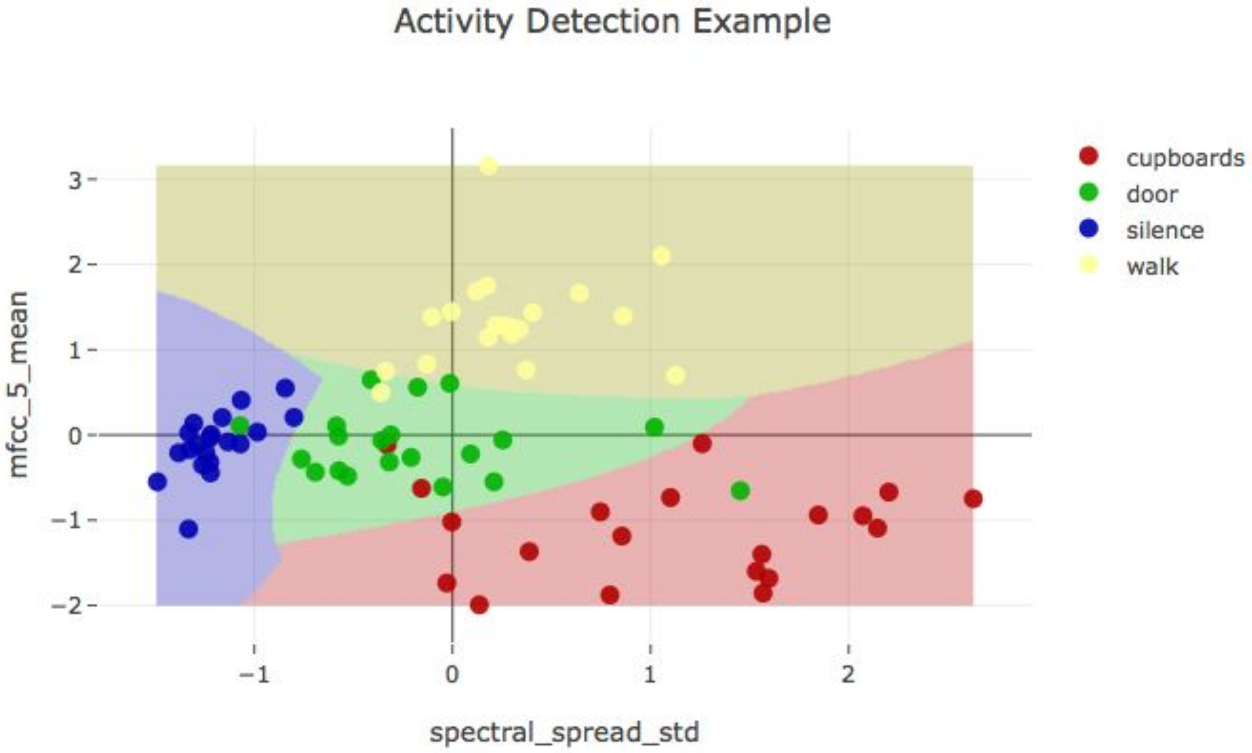
- example22.py
- 4 classes (activities):
    - cupboards
    - door
    - silence
    - Walk
- 20 samples / class



Feature-level discrimination

Classification task 3: audio event recognition

- example22.py
- 4 classes (activities):
  - cupboards
  - door
  - silence
  - Walk
- 20 samples / class

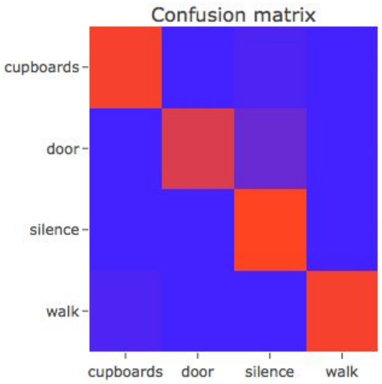


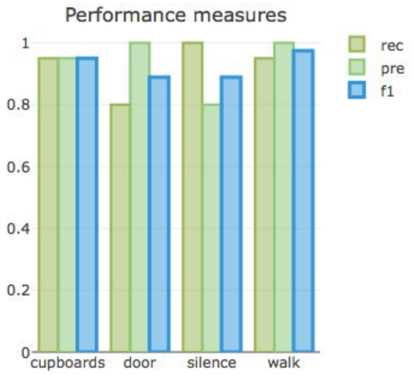
2-D feature discrimination example

Classification task 3: audio event recognition

- example23.py (classification performance)
- Overall Performance: ~90%

```python
"""!
@brief Example 23
@details Audio event detection. Classification performance
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
import utilities as ut
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW


if __name__ == '__main__':
    # extract features, concatenate feature matrices and normalize:
    f1, _, fn1 = dW("../data/activity_sounds/cupboards", 1, 1, 0.05, 0.05)
    f2, _, fn1 = dW("../data/activity_sounds/door", 1, 1, 0.05, 0.05)
    f3, _, fn1 = dW("../data/activity_sounds/silence", 1, 1, 0.05, 0.05)
    f4, _, fn1 = dW("../data/activity_sounds/walk", 1, 1, 0.05, 0.05)
    x = np.concatenate((f1, f2, f3, f4), axis=0)
    y = np.concatenate((np.zeros(f1.shape[0]), np.ones(f2.shape[0]),
                        2 * np.ones(f3.shape[0]), 3 * np.ones(f4.shape[0])))
    print(x.shape, y.shape)
    # train svm and get aggregated (average) confusion matrix, accuracy and f1
    cm, acc, f1 = ut.svm_train_evaluate(x, y, 2, C=2)
    # visualize performance measures
    ut.plotly_classification_results(cm, ["cupboards", "door", "silence",
                                          "walk"])
    print(acc, f1)
```

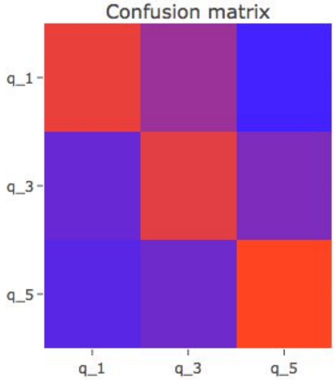


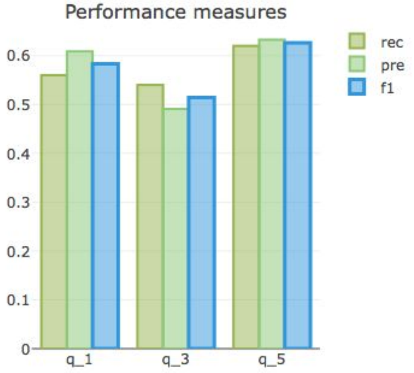


Soundscape quality recognition: classification

```python
"""!
@brief Example 24
@details Soundscape quality classification (through svm classifier)
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
import utilities as ut

if __name__ == '__main__':
    # get features from folders (all classes):
    f1, _, fn1 = dW("../data/soundScape_small/1/", 2, 1, 0.1, 0.1)
    f3, _, fn1 = dW("../data/soundScape_small/3/", 2, 1, 0.1, 0.1)
    f5, _, fn1 = dW("../data/soundScape_small/5/", 2, 1, 0.1, 0.1)

    x = np.concatenate((f1, f3, f5), axis=0)
    y = np.concatenate((np.zeros(f1.shape[0]), 1 * np.ones(f3.shape[0]),
                        2 * np.ones(f5.shape[0])))
    # train svm and get aggregated (average) confusion matrix, accuracy and f1
    cm, acc, f1 = ut.svm_train_evaluate(x, y, 10, C=10, use_regressor=False)
    # visualize performance measures
    ut.plotly_classification_results(cm, ["q_1",  "q_3", "q_5"])
    print(acc, f1)
```

- 3-class (bad, neutral, good quality)
- SVM classifier
- Accuracy 55%
- Extreme errors: 10%

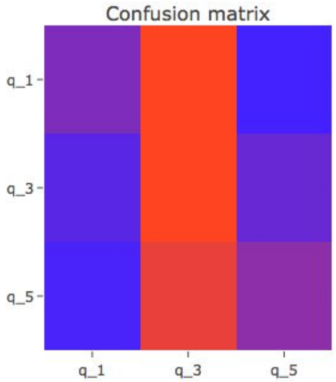


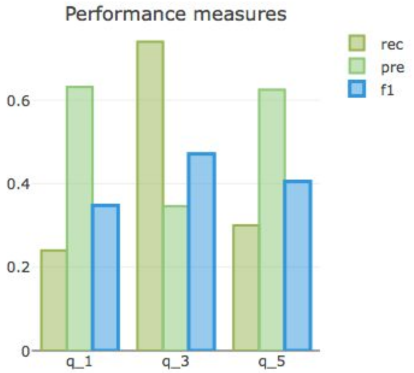


Soundscape quality recognition: regression (as classifier…)

```python
"""!
@brief Example 25
@details Soundscape quality classification (through regression and rounding)
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
import numpy as np
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
import utilities as ut

if __name__ == '__main__':
    # get features from folders (all classes):
    f1, _, fn1 = dW("../data/soundScape_small/1/", 2, 1, 0.1, 0.1)
    f3, _, fn1 = dW("../data/soundScape_small/3/", 2, 1, 0.1, 0.1)
    f5, _, fn1 = dW("../data/soundScape_small/5/", 2, 1, 0.1, 0.1)

    x = np.concatenate((f1, f3, f5), axis=0)
    y = np.concatenate((np.zeros(f1.shape[0]), 1 * np.ones(f3.shape[0]),
                        2 * np.ones(f5.shape[0])))
    # train svm and get aggregated (average) confusion matrix, accuracy and f1
    cm, acc, f1 = ut.svm_train_evaluate(x, y, 10, C=10, use_regressor=True)
    # visualize performance measures
    ut.plotly_classification_results(cm, ["q_1",  "q_3", "q_5"])
    print(acc, f1)
```

- 3-class (bad, neutral, good quality)
- SVM regression (results rounded ⟶ class ids)
- Accuracy 45%
- Extreme errors: 2%





Classification vs Regression? ⟶ Depending on the app requirements !

(also if linear was used instead of rbf in regression results would be more spread - she next slides)

Music attribute recognition through audio regression

- Data
  - 350 rock song parts (30 secs each)
  - spotify labels (taken from spotify API)
    - energy
    - Danceability
- Method
  - extract segment feature statistics using pyAudioAnalysis
  - long-term averaging
  - recognize through regression
  - linear svm kernel (rbf leads to predictions with very low spread around the average value)
- Regression measures:
  - MSE
  - **MAE**
  - Also classification measures applied if results are quantized (see previous example)

Music attribute recognition through audio regression

```python
def svm_train_evaluate_regression(X, y, k_folds, C=1):
    '''
    :param X: Feature matrix
    :param y: Continous Labels matrix
    :param k_folds: Number of folds
    :param C: SVM C param
    :return: MAE, MAE (baseline), all predictions and respective groundtruths
    '''
    # normalize
    mean, std = X.mean(axis=0), np.std(X, axis=0)
    X = (X - mean) / std
    # k-fold evaluation:
    ma, mi = y.max(), y.min()
    kf = KFold(n_splits=k_folds, shuffle=True)
    mae, r_mae, all_pred, all_gt = [], [], [], []
    for train, test in kf.split(X):
        x_train, x_test, y_train, y_test = X[train], X[test], y[train], y[test]
        cl = SVR(kernel='linear', C=C)
        cl.fit(x_train, y_train)
        y_pred = cl.predict(x_test)
        y_pred[y_pred < mi] = mi
        y_pred[y_pred > ma] = ma
        all_pred += y_pred.tolist()
        all_gt += y_test.tolist()
        y_pred_rand = np.ones(y_pred.shape) * y_train.mean()
        mae.append(mean_absolute_error(y_test, y_pred))
        r_mse.append(mean_absolute_error(y_test, y_pred_rand))
    return np.mean(mae), np.mean(r_mae), np.array(all_pred), np.array(all_gt)
```

utilities.py

Music attribute recognition through audio regression

```
import numpy as np, csv, os, plotly, plotly.graph_objs as go
from pyAudioAnalysis.MidTermFeatures import directory_feature_extraction as dW
import utilities as ut
from sklearn.externals import joblib
import argparse

def parseArguments():
   parser = argparse.ArgumentParser(prog='PROG')
   parser.add_argument('-t', '--target', nargs='+', required=True,
                       choices = ["energy", "dancability"])
   return parser.parse_args()

def get_hist_scatter(values, name):
   bins = np.arange(0, 1.1, 0.1)
   h_test = np.histogram(values, bins=bins)[0]
   h_test = h_test.astype(float) / h_test.sum()
   cbins = (bins[0:-1] + bins[1:]) / 2
   return go.Scatter(x=cbins, y=h_test, name=name)

if __name__ == '__main__':
   arg = parseArguments()
   target_type = arg.target[0]
   if os.path.isfile(target_type + ".bin"):
       x, y, filenames = joblib.load(target_type + ".bin")
   else:
       gt = {}
       with open('../data/music_data_small/{}.csv'.format(target_type)) \
               as csvfile:
           reader = csv.reader(csvfile, delimiter=',')
           for row in reader:
              gt[row[0]] = row[1]
       f, f_names, fn1 = dW("../data/music_data_small", 2, 2, 0.1, 0.1)
       x, y, filenames = [], [], []
       for i_f, f_name in enumerate(f_names):
           if os.path.basename(f_name) in gt:
               x.append(f[i_f]); filenames.append(f_names)
               y.append(float(gt[os.path.basename(f_name)]))
       x = np.array(x)
       y = np.array(y)
       joblib.dump((x, y, filenames), target_type + ".bin")
```

```
       figs = plotly.tools.make_subplots(rows=1, cols=2,
                                  subplot_titles=["Distribution of real
"
                                                  "y and predicted y",
                                                  "predicted (vert) "
                                                  "vs real (hor)"])
       mae, mae_r, all_pred, all_test = ut.svm_train_evaluate_regression(x, y,
                                                                 10, 1)
       sc1 = get_hist_scatter(all_pred, "pred")
       sc2 = get_hist_scatter(all_test, "real")
       figs.append_trace(sc1, 1, 1)
       figs.append_trace(sc2, 1, 1)
       plt2 = go.Scatter(x=all_test, y=all_pred, mode='markers',
showlegend=False)
       figs.append_trace(plt2, 1, 2)
       plotly.offline.plot(figs, filename="temp.html", auto_open=True)
       print("MAE={0:.3f}\nMAE Baseline = {1:.3f}".format(mae, mae_r))
       print("Dataset STD (gt): {0:.2f}".format(all_test.std()))
```

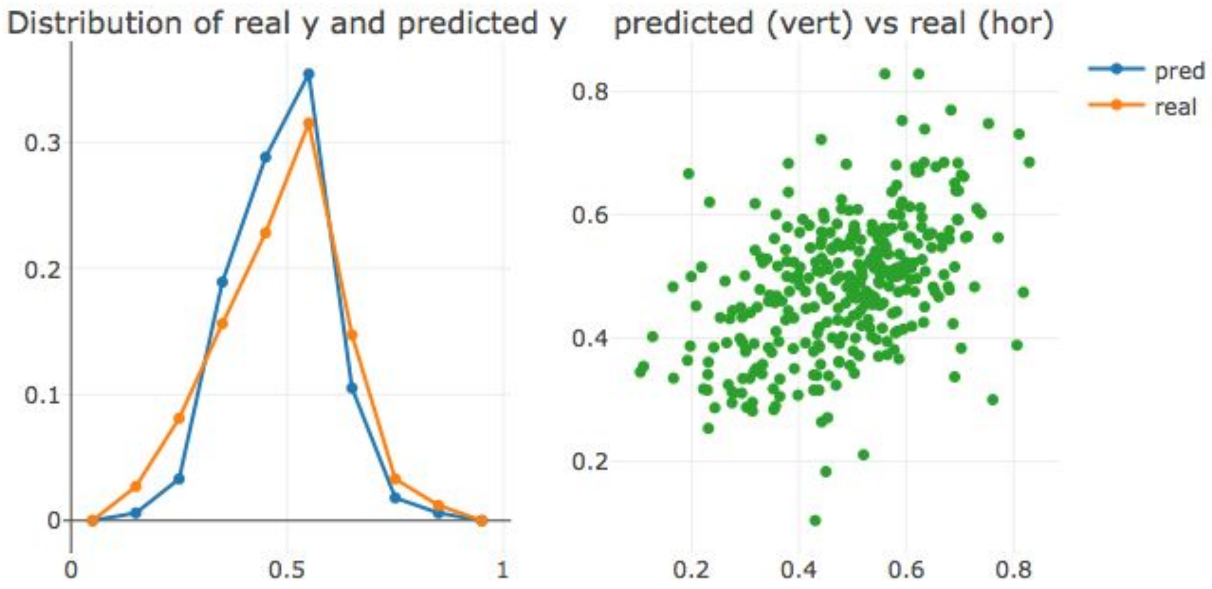
example26.py

Execution example

```
python3 example26.py -t dancability
python3 example26.py -t energy
```

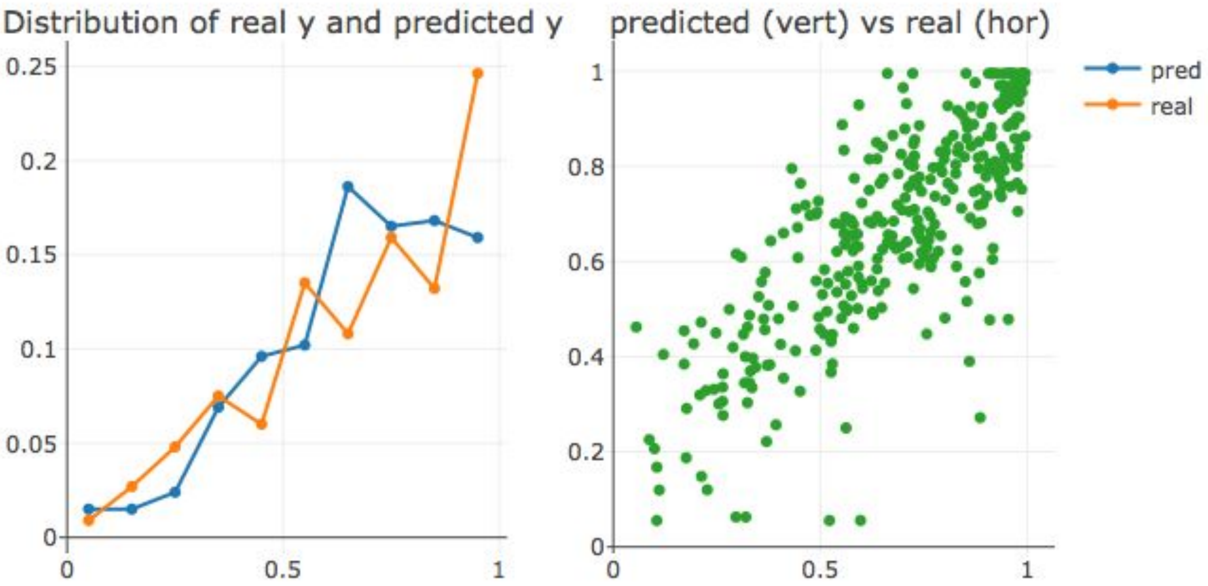
Music attribute recognition through audio regression

- Danceability
- MAE: 0.10
- MAE baseline: 0.115
- Prior target value std: 0.14



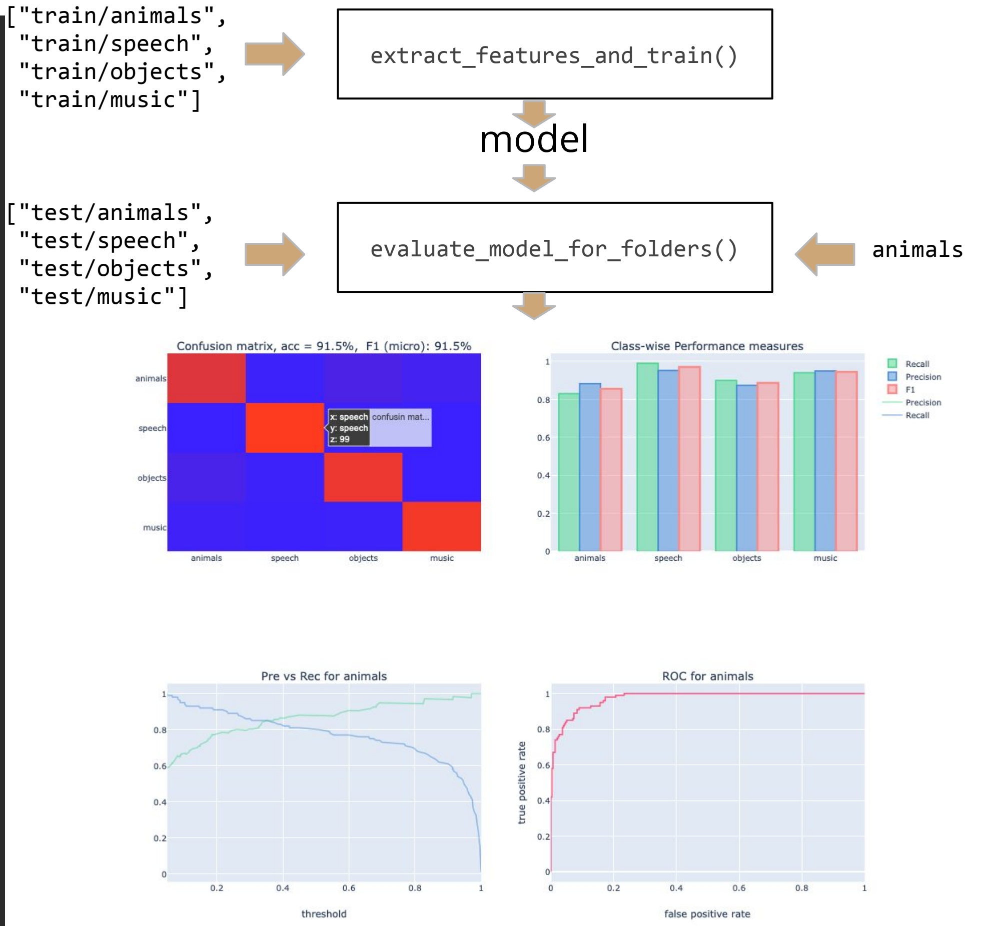
Music attribute recognition through audio regression

- Energy
- MAE: 0.11
- MAE baseline: 0.20
- Prior target value std: 0.24
- Only linear kernel in the SVM ca follow this non-Gaussian distribution
- Problem is harder in terms of MAE baseline
- (but std is higher)



Train classifier and evaluate on unknown data

```python
"""!
@brief Example 20_roc
@details General sound classification example, with focus on ROC
diagrams
@author Theodoros Giannakopoulos {tyiannak@gmail.com}
"""
from pyAudioAnalysis.audioTrainTest import
extract_features_and_train
from pyAudioAnalysis.audioTrainTest import
evaluate_model_for_folders
import os

if __name__ == '__main__':
   dirs = ["../data/general/train/animals",
           "../data/general/train/speech",
           "../data/general/train/objects",
           "../data/general/train/music"]
   class_names = [os.path.basename(d) for d in dirs]
   mw, stw = 2, .1
   extract_features_and_train(dirs, mw, mw, stw, stw, "svm_rbf",
                              "svm_general_4")

   dirs_test = ["../data/general/test/animals",
                "../data/general/test/speech",
                "../data/general/test/objects",
                "../data/general/test/music"]

   evaluate_model_for_folders(dirs_test, "svm_general_4",
                              "svm_rbf", "animals")
```



Train classifier / regressor using pyAudioAnalysis

- pyAudioAnalysis provides wrapper for a two-step audio classifier training:
  - Feature extraction:
    - Input: N classes of audio segments organized in N folders of segments (mp3 or wav files)
    - Output: a list of feature matrices (one feature matrix per class)
  - Classifier tuning and training:
    - Test different params (e.g. SVM C)
    - Random subsampling evaluation
    - Select param that maximizes F1 or Accuracy
  - Model saved in pickle (along with MEAN and STD used for normalization)
  - Then, one can use `pyAudioAnalysis.audioSegmentation.mtFileClassification()` to extract segment-level classifications for a long audio recording (supervised segmentation, see next course)
- Similar for regression

```
from pyAudioAnalysis.audioTrainTest import featureAndTrain
mt = 1.0
st = 0.05
dir_paths = ["../data/speech_music/speech", "../data/speech_music/music"]
featureAndTrain(dir_paths, mt, mt, st, st, "svm_rbf", "svm_speech_music")
```