CMLS HW1

April 2021

1 Introduction

Our group worked on **Assignment 3**: implement a classifier system able to predict the audio effect used in recordings of electric guitar and bass. We divided our work in steps as presented in the sections of this document, from the choice of the feature through the selection of a classification method, for each phase we present the decisions we made. In general, we used as a reference the second laboratory on Classification, mainly because we implemented the Support Vector Machine with majority voting for multiclass classification.

2 Feature choice

Due to the fact that both these effects imply a significant change in the power spectrum, It is efficient to use as decision feature the Root Mean Square, that leads to good predictions.

3 Dataset selection

4 Multi-class Weighted Support Vector Machine

SVMs are generally binary classifier, so, in order to manage 3 classes, we used one SVC for each couple of class: Distortion/Tremolo, Distortion/NoFX and Tremolo/NoFX. Each one misclassify the missing class, hence, applying majority voting, we are able to train and test the SVM with all the instances of the samples.

In our training set we face the problem of having an unbalance amount of tracks for every class type. It contains half of no-effect tracks compared to the effected ones, hence we have found a way to solve this problem. The SVM model works well also with unbalanced training-set taking care of setting efficiently the right value of an internal parameter called C.
C determines the number of tolerated severity apply to the margin violation (and to the hyperplane), it controls the trade-off between maximizing the separation margin between classes minimizing the number of misclassified instances.

Specifically, each example in the training dataset has its own penalty term C used in the calculation of the margin when fitting the SVM model. The value of an example's C-value can be calculated as a weighting of the global C-value, where the weight is defined proportional to the class distribution.

$$C_i = weight_i * C$$

By default it is suppose a balanced training-set, each class has the same weighting, which means that the softness of the margin is symmetrical.
A larger weighting factor C can be used for the minority class, allowing the margin to be softer (smaller penalty for misclassified examples), whereas a smaller C can be used for the majority class, forcing the margin to be harder and preventing misclassified examples. This has the effect of encouraging the margin to contain the majority class with less flexibility, but allow the minority class to be flexible with misclassification of majority class examples onto the minority class side if needed.

5 Cross-validation for SVM parameters

In order to select appropriate parameters for the SVM, we tested our classifier through Cross-validation using the function GridSearchCV from scikit-learns, in doing so we were able to fine-tune the machine learning algorithm and to avoid overfitting the model. We decided also to run the grid-search only on the classifier between Tremolo and NoFX, because those were the classes that would presents more problems during the classification. The code snippet and the result is shown below, the score measures are precision and recall, while the C parameters were chosen arbitrarily within an order of magnitude:

```
tuned_parameters = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'kernel': ['rbf']},
 ]

scores = ['precision', 'recall']

for score in tqdm(scores):
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(
        SVC(), tuned_parameters, scoring='%s_micro' % score
    )
    clf.fit(X_train12, y_train_12)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    [...]
```

Output:

```
# Tuning hyper-parameters for precision

Best parameters set found on development set:

{'C': 1000, 'kernel': 'rbf'}

Grid scores on development set:

0.759 (+/-0.121) for {'C': 1, 'kernel': 'linear'}
0.735 (+/-0.073) for {'C': 10, 'kernel': 'linear'}
0.710 (+/-0.103) for {'C': 100, 'kernel': 'linear'}
0.689 (+/-0.085) for {'C': 1000, 'kernel': 'linear'}
0.916 (+/-0.044) for {'C': 1, 'kernel': 'rbf'}
0.984 (+/-0.035) for {'C': 10, 'kernel': 'rbf'}
0.993 (+/-0.019) for {'C': 100, 'kernel': 'rbf'}
0.993 (+/-0.016) for {'C': 1000, 'kernel': 'rbf'}
[...]

# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'C': 1000, 'kernel': 'rbf'}

Grid scores on development set:

0.759 (+/-0.121) for {'C': 1, 'kernel': 'linear'}
0.735 (+/-0.073) for {'C': 10, 'kernel': 'linear'}
0.710 (+/-0.103) for {'C': 100, 'kernel': 'linear'}
0.689 (+/-0.085) for {'C': 1000, 'kernel': 'linear'}
0.916 (+/-0.044) for {'C': 1, 'kernel': 'rbf'}
0.984 (+/-0.035) for {'C': 10, 'kernel': 'rbf'}
0.993 (+/-0.019) for {'C': 100, 'kernel': 'rbf'}
0.993 (+/-0.016) for {'C': 1000, 'kernel': 'rbf'}
[...]
```

6 Results