I added some new features: Median Frequency, Mean Frequency, Marginal Discrete Wavelet Transform and four features from continuous wavelet transform (Mean coefficient, Minimum coefficient, Mean scale, Median scale).

With Marginal Discrete Wavelet Transform is the performance worse. So only other new features are used.

In old features there are several features duplicated (IEMG - MAV, SSI-VAR-RMS), so MAV, VAR, RMS are discarded.

I also tried to directly use Discrete Wavelet Transform and flatten the output to an array, but the performance is not good, using PCA didn’t improve either.

Before, I just replace the invalid value of emg data with mean value, but I didn’t notice that in some files invalid value are too much, so I directly drop them out this time. The result gets a little better.

I used lowpass filter on emg data, but the generalization gets worse, I think noise is good for generalization.

# Test

Use data include all patients but not the whole dataset. Not shuffling and split the data to 20% for test and 80% for next split. Then shuffle the rest and split it in 20% for validation and 80% for training. Features extracted from 8 channels. During training is early stopping used. If validation set is not improved in 30 rounds, training will be stopped.

## compare features from left and right limb

xgboost:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | train | valid | test | rest data |
| baseline | 1 | 0.970425 | 0.955262 | 0.95969 |
| LEFT | 1 | 0.957699 | 0.923573 | 0.934614 |
| RIGHT | 1 | 0.956623 | 0.928735 | 0.939252 |

DNN:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | train | valid | test | rest data |
| baseline | 0.999104 | 0.983689 | 0.94924 | 0.958637 |
| LEFT | 0.990051 | 0.963255 | 0.905936 | 0.922966 |
| RIGHT | 0.988706 | 0.959491 | 0.940493 | 0.931798 |

## Compare Feature from each 2 channels

Xgboost:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | train | valid | test | rest data |
| baseline | 1 | 0.970425 | 0.955262 | 0.95969 |
| LEFT\_TA\_LEFT\_TS | 1 | 0.902133 | 0.841268 | 0.818 |
| LEFT\_TA\_LEFT\_BF | 1 | 0.920954 | 0.8731 | 0.866087 |
| LEFT\_TA\_LEFT\_RF | 1 | 0.911095 | 0.858474 | 0.901252 |
| LEFT\_TS\_LEFT\_BF | 0.999597 | 0.908227 | 0.872957 | 0.835015 |
| LEFT\_TS\_LEFT\_RF | 1 | 0.898369 | 0.845856 | 0.863981 |
| LEFT\_BF\_LEFT\_RF | 1 | 0.91468 | 0.898623 | 0.888875 |
| RIGHT\_TA\_RIGHT\_TS | 1 | 0.923284 | 0.87353 | 0.881644 |
| RIGHT\_TA\_RIGHT\_BF | 0.998252 | 0.912888 | 0.89002 | 0.897363 |
| RIGHT\_TA\_RIGHT\_RF | 1 | 0.923821 | 0.866934 | 0.875 |
| RIGHT\_TS\_RIGHT\_BF | 1 | 0.921312 | 0.906223 | 0.865601 |
| RIGHT\_TS\_RIGHT\_RF | 1 | 0.922746 | 0.828219 | 0.882495 |
| RIGHT\_BF\_RIGHT\_RF | 1 | 0.928123 | 0.85489 | 0.861287 |

DNN:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | train | valid | test | rest data |
| baseline | 0.999104 | 0.983689 | 0.94924 | 0.958637 |
| LEFT\_TA\_LEFT\_TS | 0.918075 | 0.89335 | 0.79782 | 0.810728 |
| LEFT\_TA\_LEFT\_BF | 0.934119 | 0.917369 | 0.911959 | 0.891731 |
| LEFT\_TA\_LEFT\_RF | 0.923139 | 0.898727 | 0.859335 | 0.898254 |
| LEFT\_TS\_LEFT\_BF | 0.940528 | 0.909661 | 0.861629 | 0.83149 |
| LEFT\_TS\_LEFT\_RF | 0.922825 | 0.89335 | 0.84686 | 0.863596 |
| LEFT\_BF\_LEFT\_RF | 0.924573 | 0.90984 | 0.876255 | 0.873298 |
| RIGHT\_TA\_RIGHT\_TS | 0.947923 | 0.921312 | 0.843418 | 0.859585 |
| RIGHT\_TA\_RIGHT\_BF | 0.936808 | 0.918265 | 0.892601 | 0.896512 |
| RIGHT\_TA\_RIGHT\_RF | 0.946847 | 0.922029 | 0.864783 | 0.864345 |
| RIGHT\_TS\_RIGHT\_BF | 0.934522 | 0.911095 | 0.883568 | 0.847897 |
| RIGHT\_TS\_RIGHT\_RF | 0.936046 | 0.917189 | 0.82951 | 0.872144 |
| RIGHT\_BF\_RIGHT\_RF | 0.954645 | 0.926869 | 0.883711 | 0.872286 |

## Parameter

model = xgb.XGBClassifier(max\_depth=8,

learning\_rate=0.3,

n\_estimators=1000,

silent=True,

eval\_metrics='error',

objective='binary:logistic',

seed=100,

reg\_lambda = 1,

)

DNN:

input\_ = layers.Input(shape=feature.shape[1:])

l1 = layers.Dense(128,activation='elu')(input\_)

drop1 = layers.Dropout(0.2)(l1)

l2 = layers.Dense(64,activation='elu')(drop1)

drop2 = layers.Dropout(0.2)(l2)

output = layers.Dense(1,activation='sigmoid')(drop2)

model = Model(inputs=[input\_],outputs=[output])