Snore Detection: Analysis and Implementation

Wiwitthawin Chareonngam
6310501933

Content

- Objective and Hypothesis
- Literature Review
- Characteristic of Input Signal
- Analyze Signal
- Algorithm Testing
- Implementation

Objective and Hypothesis

Objective: analyze snore sound and classify by various machine learning model with implementation on board

Hypothesis: snore sound can be extracted by MFCC and classified by model

Literature Review

- Automatic and Unsupervised Snore Sound Extraction From Respiratory Sound Signals

Recorded by 2 microphones: tracheal microphone and ambient microphone. Signal analysis is performed in 3 steps: segmentation using V-BOX control chart; moving vertically trimmed box along time axis data, feature extraction using PCA to reduce dimension of feature space which founded 2 largest eigenvalues and results in 2D feature vector and unsupervised classification using FCM, unsupervised fuzzy C-means clustering algorithm, to label as snore/breath or noise. Overall accuracy is 98.6% for tracheal microphone and 93.1% on ambient microphone. Ambient microphone has larger variation in accuracy and PPV such that accuracy is 3.3% higher when applied with simple snorers and dropped by 1.5% when applied with OSA patients.

- Automatic Detection of Snoring Signals: Validation with Simple Snorers and OSAS Patients

System to detect composed of system detected changes of variance in signal and 2-layer feedforward multilayer neural network with black propagation algorithm trained with 625 selected events from normal snores and OSAS patients. There are 500 snores from database of 30 snorer with different AHI. Sound has been acquired at PPG sensor and digitized with 12-bit with 5 kHz sampling rate. The detector can achieve 82% sensitivity and 90% PPV

- An SVM-based Classification of Oral and Nasal Snoring Sounds with Kullback-Leibler Kernel

Compare Kullback-Leibler kernel (KL) with others in Support Vector Machine (SVM) Classifier. KL divergences is dissimilarity measure between 2 probabilistic density function and spectral density of signal has similar concept. It is necessary to adjust kernel-specific parameter and cost function parameter with trail and error under 10-fold CV test. KL kernel provide 99.7% classification rate for 10-fold CV test. KL and Laplace kernel is more effective for SVM-based classification of oral and nasal snoring sounds than Euclid distance-based kernels which has 98.8% classification rate.

Characteristic of Input Signal

- Signal in this project can categorized into 2 signals

1. Signal from Multiple Database (Kaggle) with 1 second's duration

500 snoring sound: 363 samples of snore without background sound, 137 samples consists of snoring sound with background sound

-> 450 sounds of 44.1kHz sampling rate and 50 sounds of 48 kHz

500 non-snoring sound: 10 categories of sound (baby crying, clock ticking, door opened, silence and vibration, toilet flashing, siren, rain and thunderstorm, streetcar sounds, talking, and television news.)

-> 58 sounds of 44.1kHz sampling rate and 442 sounds of 48 kHz

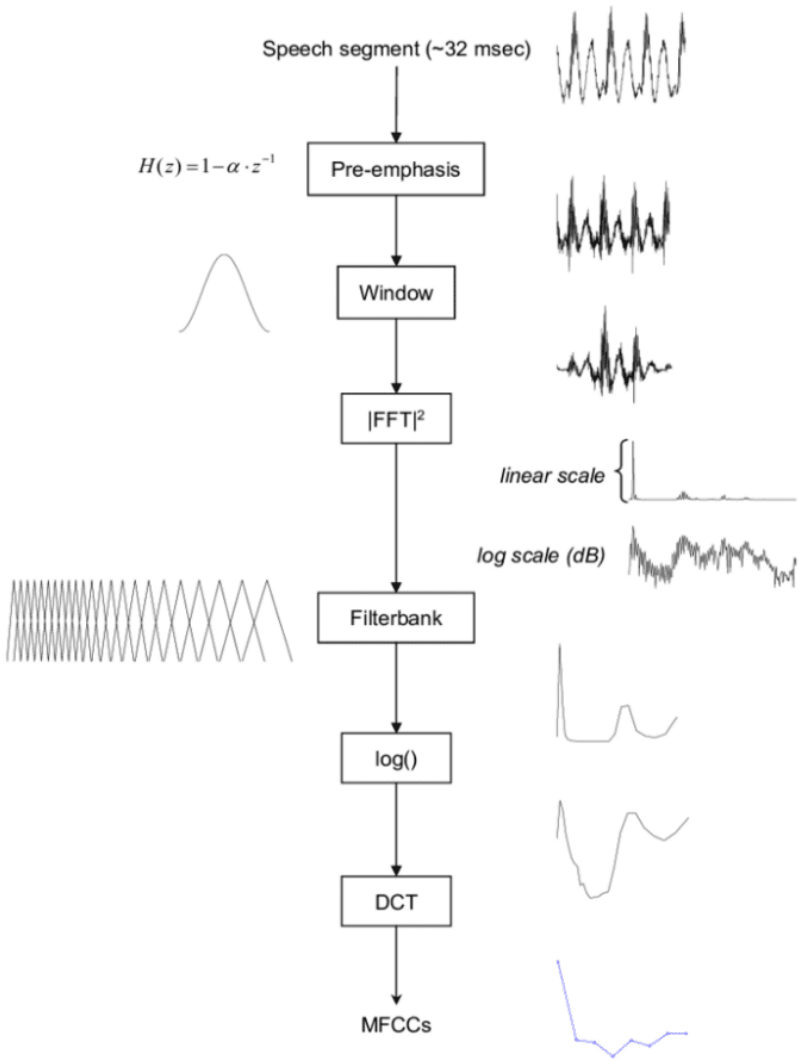
Characteristic of Input Signal

- Signal in this project can categorized into 2 signals

2. Signal from DSK6713 board with 8kHz sampling rate, 32-bits

Frame blocking will block into 256 samples with adjacent frame separated by 100 samples to be stationary. 256/8 kHz = 32ms

Analyze Signal

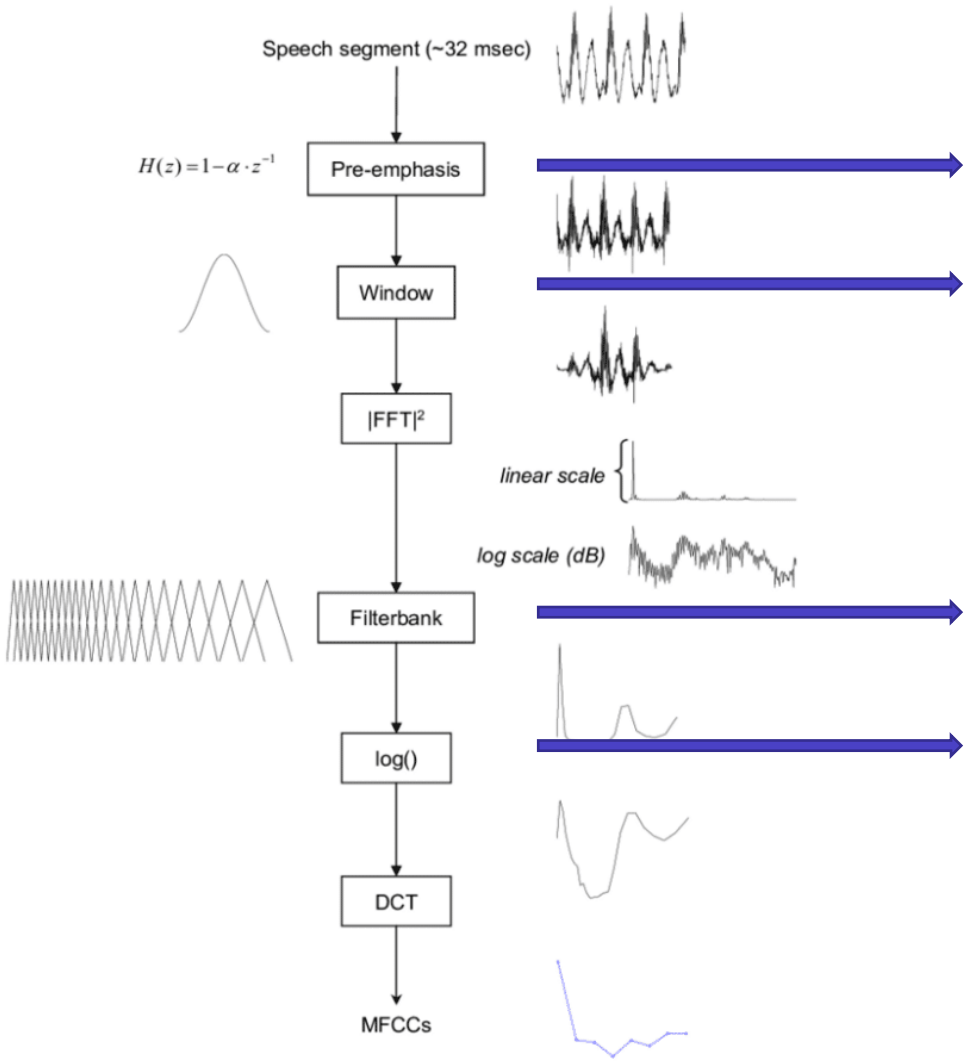


MFCC (Mel-frequency cepstral coefficient)

Feature extraction of signal due to human hearing system

*Cepstrum: inverse of the log of the magnitude of the signal

Analyze Signal



High-pass filter

Chop signal into 256 samples/frame

Map actual frequency into human hearing frequency

$$mel(f) = 1127 \ln(1 + \frac{f}{700})$$

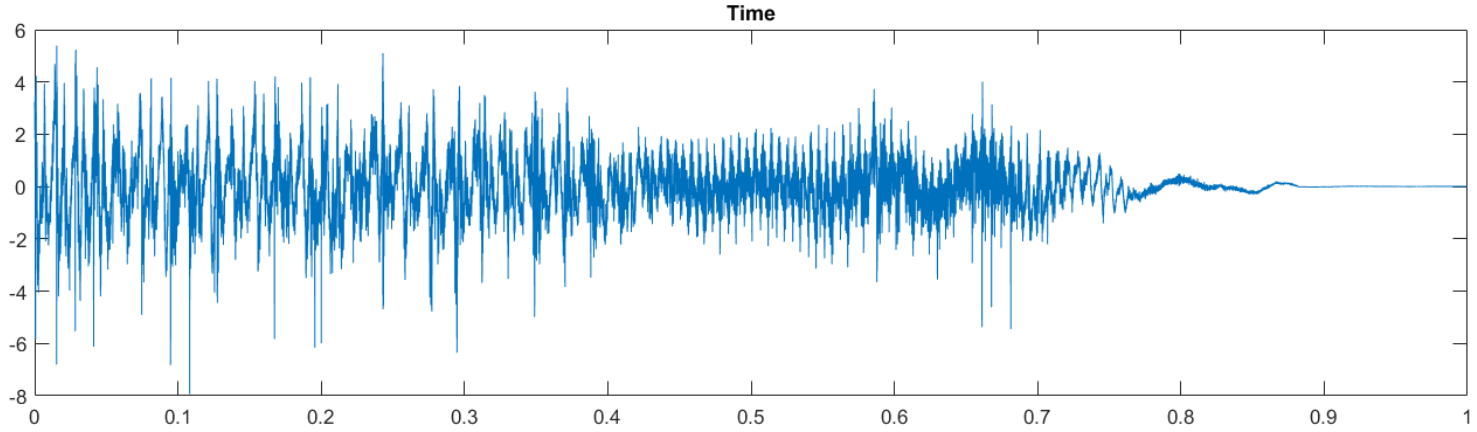
Human are less sensitive to lower frequency

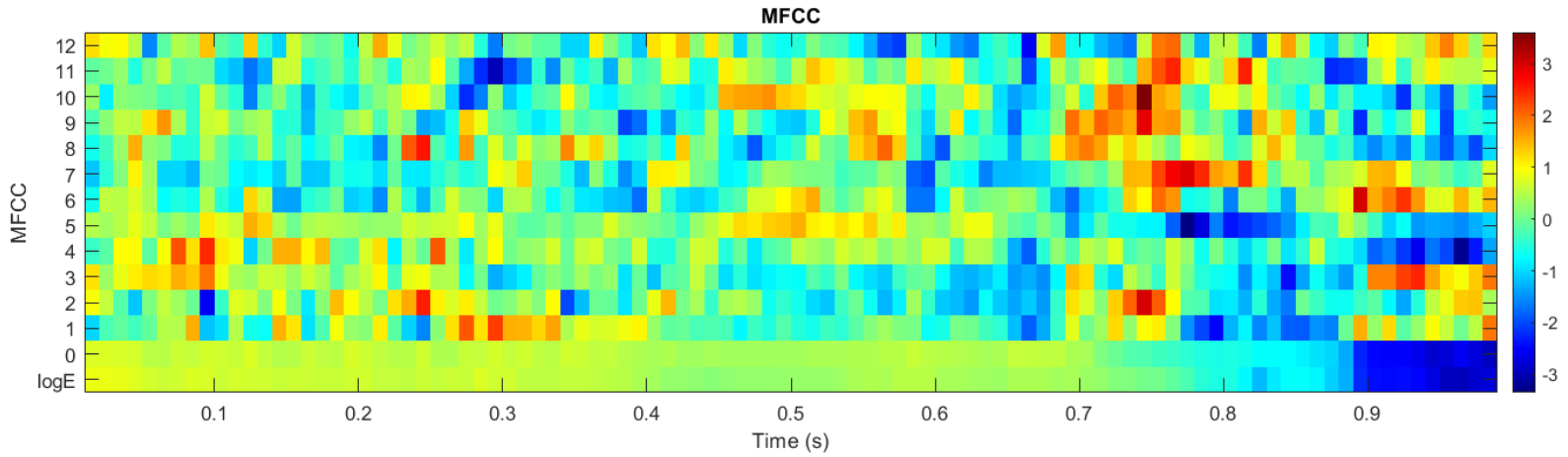
Analyze Signal

First analyze data in MATLAB

13 coefficients MFCC

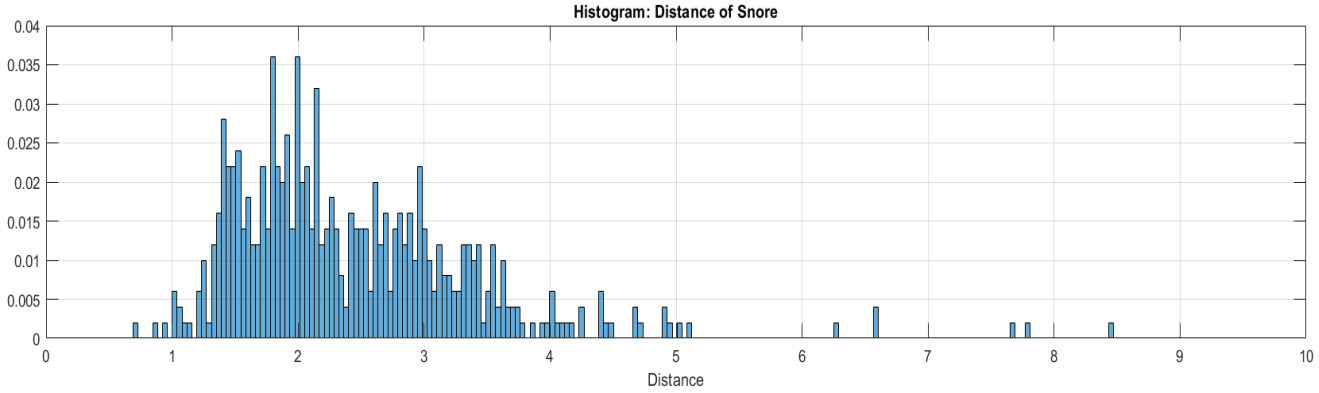
Too many Features to observe with eyes !!

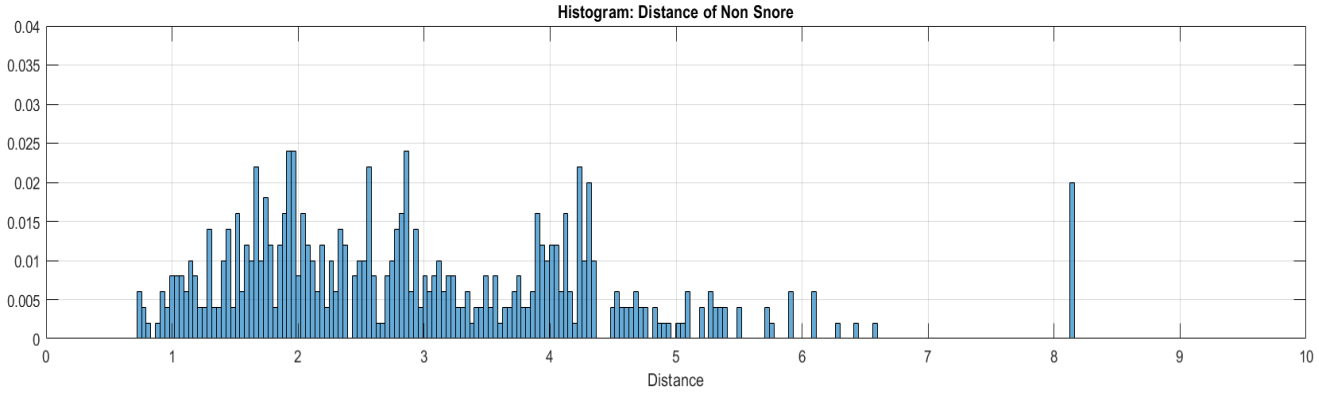




Analyze Signal

Histogram plot of Snore and Non-snore



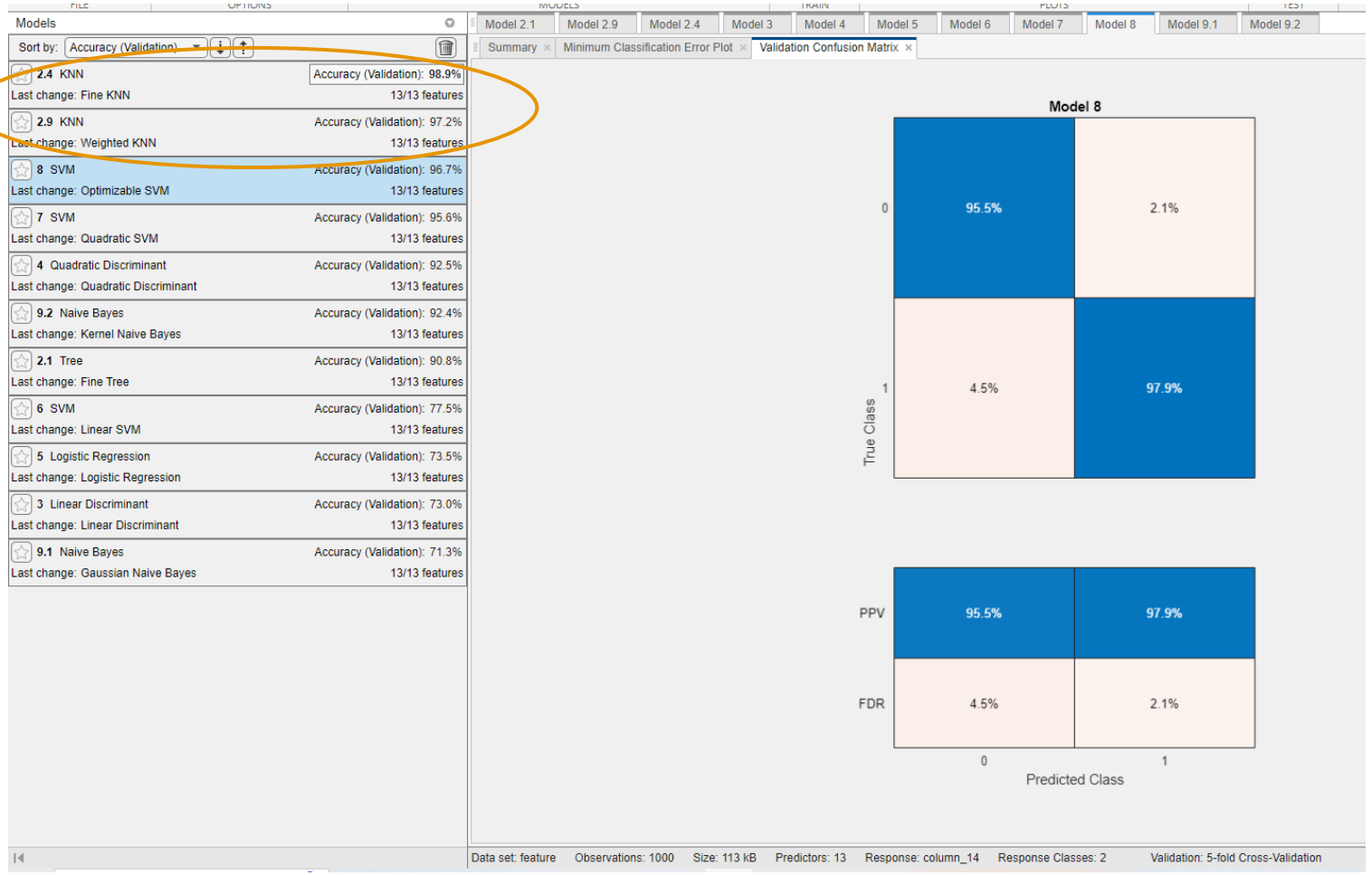


Histogram plot of Distance of Non-Snore to mean of Snore has less probability than Distance of snore to its mean and Non-Snore has more variance than Snore

With the result of the distance of Snore to its mean and Non-Snore to mean of snore, classifier learner in MATLAB can use to create model that best fitted for test data.

Analyze Signal

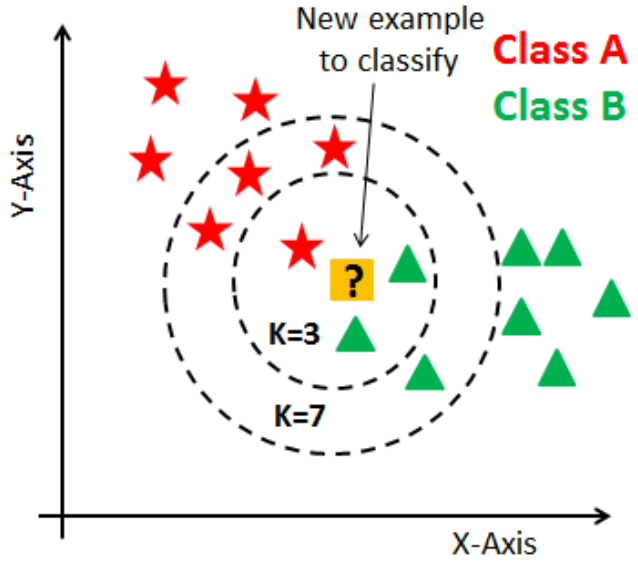
Train Model

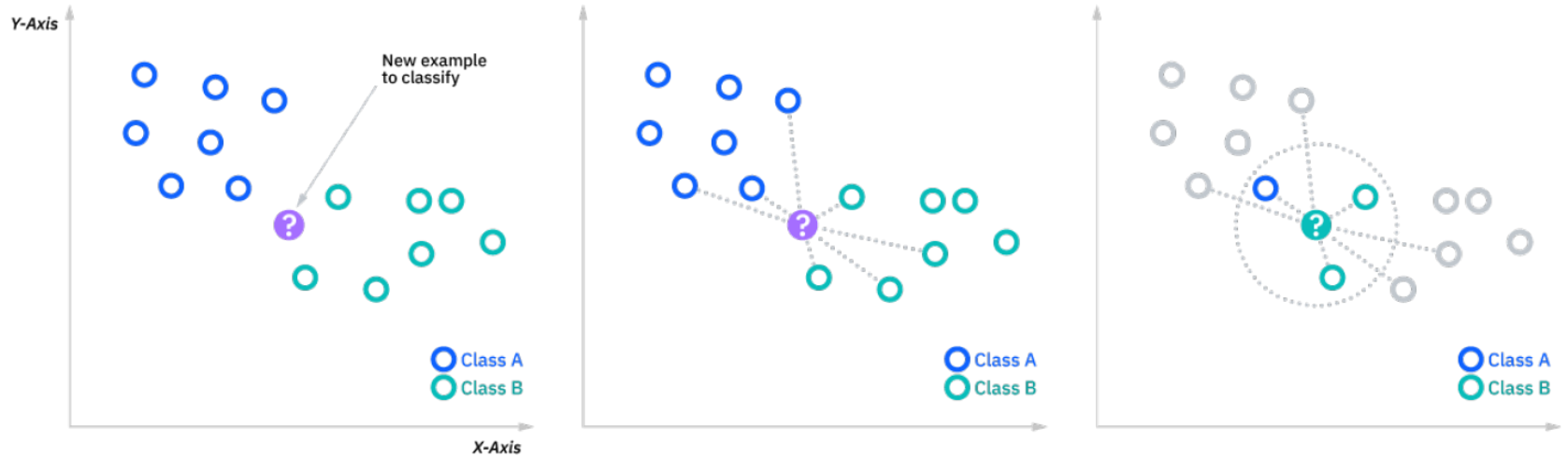


Analyze Signal

KNN Classification – K Nearest Neighbours

KNN is algorithm used for regression and classification with distance of training set and testing set. KNN algorithm is simple and easy to implement but has need high computational due to size of training and testing set. As KNN observe similarity of dataset with distance, it is memory-based model.





Analyze Signal

KNN on Python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

classifier = KNeighborsClassifier(n_neighbors = 3, metric = 'minkowski', p = 2)
ypred = kNN(X_train,y_train, X_test, k=1)
print(classification_report(y_test, ypred))
```

```
def kNN(Xtrain,Ytrain, X_test, k=1):
    Ytest = []
    for x in X_test:
        d = np.sqrt(np.sum((Xtrain - x)**2, axis=1))
        idx = np.argsort(d)
        (values, counts) = np.unique(Ytrain[idx[:k]], return_counts = True)
        ind = np.argmax(counts)
        Ytest.append(values[ind])
        print(values)
    return Ytest
```

Analyze Signal

Implement KNN in C for DSK6713

```
/* Identifying the Speaker */

    for (i = 0; i < N_TRAIN; i++)
    {
        d[i] = 0;
        for (j = 0; j < N_DIM; j++)
        {
            d[i] += pow(training_vector[i][j] - mfcc_vector[j], 2);
        }
        d[i] = sqrt(d[i]);
        idx[i] = i;
    }

    // sort the distances in ascending order
    for (i = 0; i < N_TRAIN-1; i++)
    {
        for (j = i+1; j < N_TRAIN; j++)
        {
            if (d[j] < d[i])
            {
                tmp = d[i];
                d[i] = d[j];
                d[j] = tmp;
                tmp_idx = idx[i];
                idx[i] = idx[j];
                idx[j] = tmp_idx;
            }
        }
    }
```

```
    // find the most common class among the k nearest neighbors
    for (i = 0; i < K; i++)
    {
        counts[i] = 0;
    }

    for (i = 0; i < K; i++)
    {
        values[i] = Ytrain[idx[i]];
        counts[i] = 1;
        for (j = i+1; j < N_TRAIN && d[j] == d[i]; j++)
        {
            counts[i]++;
            values[i] = Ytrain[idx[i]];
        }
    }

    ind = 0;
    for (i = 1; i < K; i++) {
        if (counts[i] > counts[ind]) {
            ind = i;
        }
    }

    Ytest = values[ind];
    printf("Predicted class: %d\n", Ytest);
}
}
```

Algorithm Testing

- KNN in MATLAB

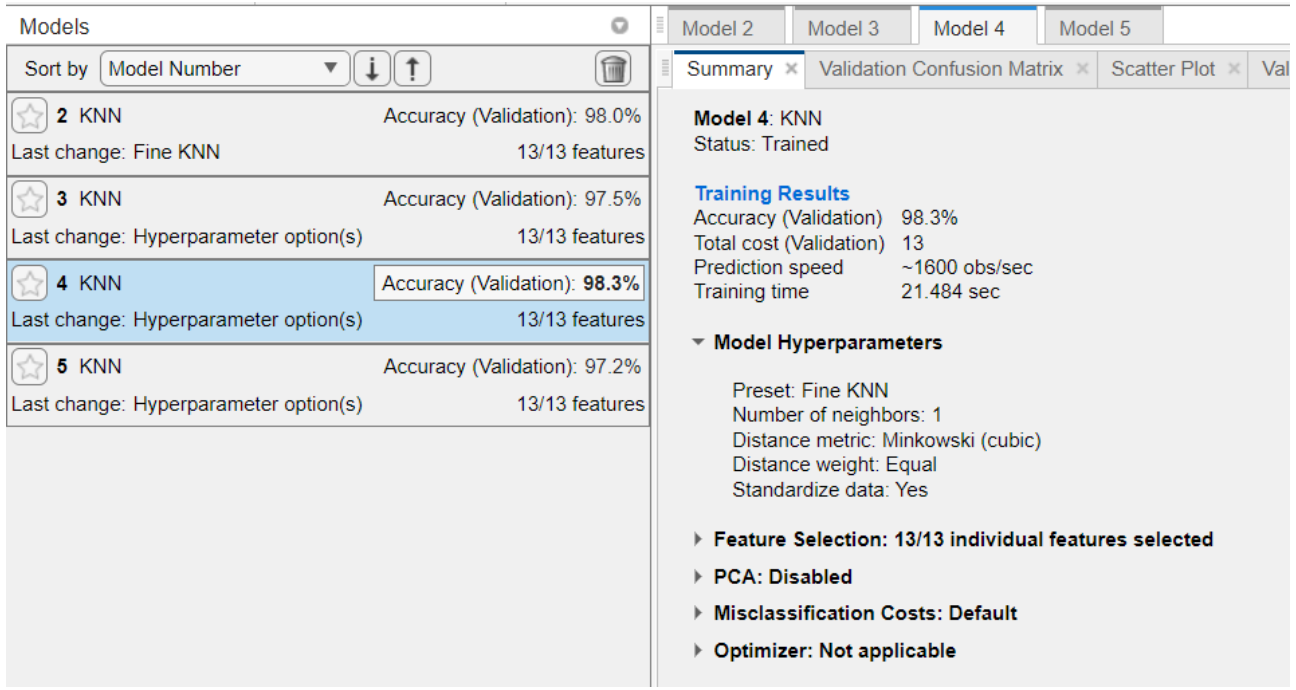
*5-fold cross validation, 25% testing set, fine KNN

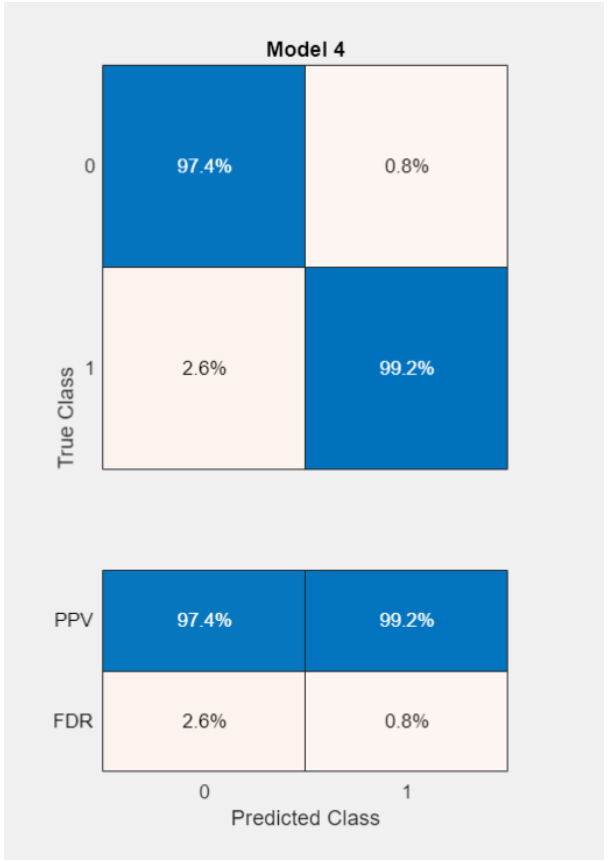
Euclidean 1

Euclidean 3

Minkowski 1

Minkowski 3





Algorithm Testing

- KNN in Python

Euclidean Distance with 35% testing set

K = 1

```
print(classification_report(y_test, ypred))
```

	precision	recall	f1-score	support
0	1.00	0.99	1.00	186
1	0.99	1.00	1.00	164
accuracy			1.00	350
macro avg	1.00	1.00	1.00	350
weighted avg	1.00	1.00	1.00	350

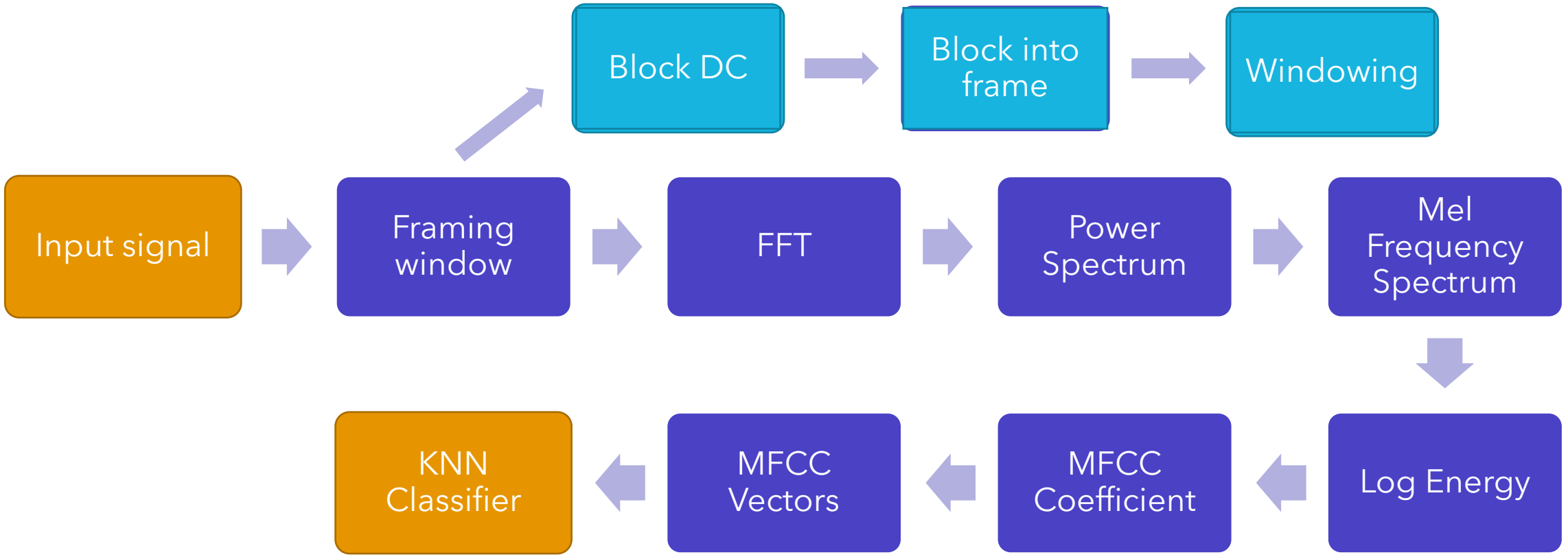
K = 3

```
print(classification_report(y_test, ypred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	176
1	1.00	1.00	1.00	174
accuracy			1.00	350
macro avg	1.00	1.00	1.00	350
weighted avg	1.00	1.00	1.00	350

Implementation

Implementation on board



Implementation

Implementation on board

Implementation is based on “speaker_recognition” project which contain two main modules: feature extraction and classification. This project uses vector quantization(VQ) to classify with distance compared to train dataset.
Instead of using old VQ, KNN need to adjust code due to its algorithm which require array to store k nearest neighbor and count the most likely class.

Implementation

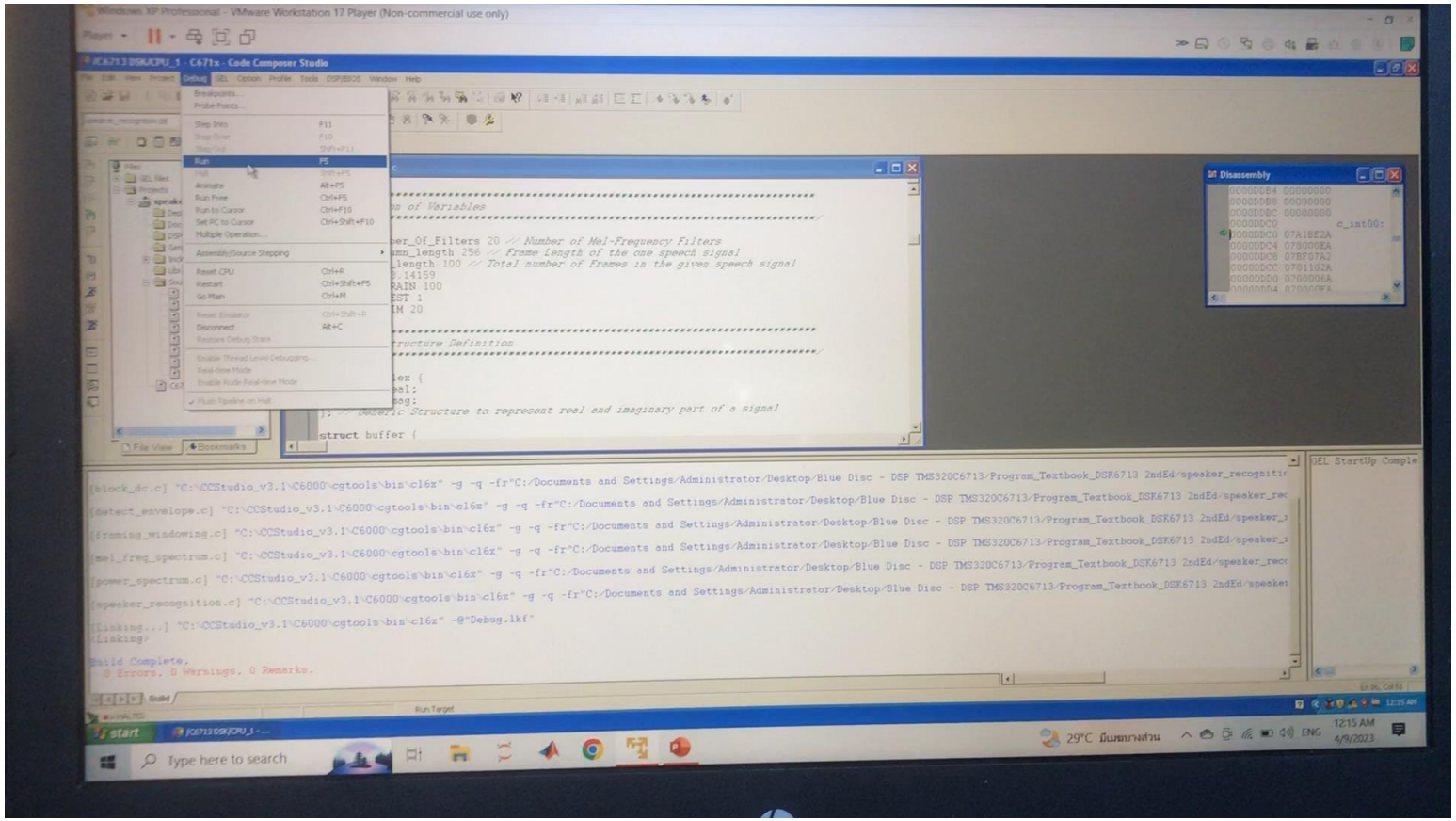
Train Dataset

```
#define Number_Of_Speakers 100 /* Snoring Sound = first 40 sounds, Non-snoring sound = last 60 sounds*/

#define Number_Of_Coefficients 20 /* Total Number of Co-efficients in the system */


float training_vector[Number_Of_Speakers][Number_Of_Coefficients] = {{10010410.000000, 13516.052734, 25562.792969, 7324.037598, 15248.061523, 13641.505859, 3374.666504, 44092.671875,
{9746594.000000, 13821.575195, 27676.712891, 5115.155762, 13002.924805, 13888.841797, 1521.278687, 42652.023438, 6633.749512, 33529.175781, 1619.663452, 20402.269531, 4253.306641, 19
{9688055.000000, 13296.713867, 27151.796875, 5496.197266, 12929.128906, 14364.613281, 1310.868164, 43022.750000, 6519.281738, 33943.062500, 1671.952515, 20018.363281, 4129.836426, 19
{9649633.000000, 15090.191406, 29689.462891, 7475.878906, 17026.605469, 14276.485352, 2721.877930, 45163.496094, 5056.300293, 32870.152344, 1487.993652, 21524.630859, 3803.941406, 17
{9708068.000000, 15085.206055, 29430.871094, 5827.071289, 15538.399414, 12502.755859, 1151.848755, 41237.476563, 4909.703613, 32788.945313, 1412.896851, 20906.628906, 4494.527832, 16
{9585839.000000, 16184.819336, 27541.263672, 6711.733887, 17358.640625, 11892.981445, 4398.141113, 45331.113281, 5944.652832, 31734.998047, 1054.533936, 22652.697266, 3796.283447, 16
{9714228.000000, 13612.994141, 27490.283203, 6850.639648, 15028.463867, 13787.876953, 1424.885010, 42926.968750, 4514.147461, 33467.859375, 1453.504028, 21511.970703, 4018.552734, 17
{9846439.000000, 10325.457031, 25232.210938, 6464.994629, 13991.591797, 13351.518555, 1283.795532, 39355.664063, 5196.960449, 30936.947266, 833.835938, 18823.962891, 4503.754395, 173
{9822189.000000, 8709.792969, 23314.003906, 6586.874023, 10332.830078, 15723.273438, 1050.216919, 42118.898438, 6016.399902, 33892.039063, 1566.700317, 19250.685547, 3810.612061, 201
{9781333.000000, 10098.750000, 23228.646484, 7330.597168, 12234.371094, 15587.029297, 2273.625732, 43572.652344, 5795.210938, 32723.041016, 1326.556641, 20465.798828, 4121.851563, 18
{9733538.000000, 11218.251953, 23863.572266, 7320.406738, 13308.748047, 14763.917969, 3333.849365, 44193.214844, 5557.871582, 31238.468750, 1449.881958, 19473.039063, 4366.697754, 18
{9807517.000000, 9328.405273, 23321.253906, 8975.378906, 13426.474609, 16018.988281, 1984.662476, 43596.535156, 4774.403320, 31623.769531, 1202.753784, 20326.552734, 3931.073486, 177
{9805672.000000, 10736.572266, 24784.976563, 7174.810547, 12769.881836, 14302.486328, 944.798889, 41919.269531, 4534.443359, 33230.281250, 1195.286621, 20668.080078, 3944.303223, 180
{9803152.000000, 7873.374023, 21411.001953, 7694.567871, 8942.846680, 17136.171875, 1059.695923, 43621.589844, 5791.123535, 33659.898438, 1745.652710, 18817.894531, 4204.068848, 1918
{9644285.000000, 14366.101563, 27393.867188, 6333.368164, 15102.688477, 12381.398438, 736.431580, 40360.417969, 4443.115723, 30072.957031, 2156.619629, 19431.410156, 5114.890137, 156
{9625144.000000, 12881.200195, 25694.041016, 7019.330566, 12901.965820, 12871.866211, 1136.377686, 40180.847656, 5047.071777, 33187.152344, 1223.639282, 20585.083984, 4572.855469, 16
{9621717.000000, 13668.648438, 26965.324219, 6814.522949, 14497.124023, 13214.705078, 889.963196, 41181.484375, 4177.370605, 32519.820313, 1828.132324, 20411.787109, 4620.062012, 158
{9592408.000000, 11136.582031, 24875.082031, 7533.574219, 13812.411133, 14486.256836, 2960.760010, 43345.574219, 5885.473145, 32438.914063, 1928.907227, 19838.923828, 4485.132813, 17
{9578022.000000, 12526.858398, 25560.652344, 8500.707031, 15234.305664, 13754.685547, 2591.277344, 42497.605469, 5170.057617, 32611.466797, 1401.212036, 20500.666016, 4392.949219, 16
{9704059.000000, 10205.462891, 23441.738281, 7195.319824, 12253.118164, 12981.365234, 1633.787354, 40199.457031, 5136.775879, 34567.390625, 685.406860, 20143.771484, 5047.600586, 151
{9847230.000000, 8640.238281, 22645.546875, 8968.007813, 11046.190430, 15405.331055, 1132.539185, 45238.535156, 3234.111328, 35726.894531, 1630.702515, 22175.783203, 4219.991211, 173
{9789733.000000, 9364.583008, 23557.816406, 7545.420898, 11973.372070, 13627.870117, 1583.860352, 40466.492188, 5045.461426, 35595.503906, 810.369873, 21080.449219, 4176.241699, 1700
{9869709.000000, 7082.454590, 20107.443359, 7982.224609, 8593.063477, 16673.933594, 1007.904602, 42856.285156, 5641.477539, 35740.992188, 1849.455322, 19046.017578, 4339.912598, 1865
{9758500.000000, 9772.577148, 21575.011719, 8907.457031, 12663.534180, 16440.417969, 3143.497559, 45583.328125, 5097.041016, 33082.105469, 1238.620972, 22328.435547, 3382.792236, 186
{9685987.000000, 11216.186523, 23941.445313, 6772.419922, 10953.171875, 13979.640625, 1035.935303, 42289.378906, 6002.603027, 36001.867188, 710.486023, 20460.980469, 3907.449219, 187
{9835748.000000, 10656.576172, 26860.099609, 6968.914551, 13544.100586, 12237.044922, 550.125305, 40939.187500, 3472.200439, 35443.531250, 1660.831177, 23304.554688, 3617.899902, 171
{9856695.000000, 10031.466797, 24875.107422, 6726.092773, 12824.695313, 13117.506836, 823.215759, 41886.453125, 3761.529785, 34457.695313, 1309.115601, 21236.281250, 3972.085938, 178
{9822827.000000, 10347.766602, 26471.773438, 6986.468262, 12359.633789, 14080.394531, 785.052612, 42830.488281, 3683.251465, 34560.679688, 1504.969849, 21353.751953, 3977.207520, 175
{9771828.000000, 11765.902344, 26141.955078, 6235.866699, 13145.622070, 14507.966797, 403.839447, 44630.328125, 2957.220459, 35332.007813, 1808.036621, 21887.597656, 3596.652100, 185
```

Real-Time Testing

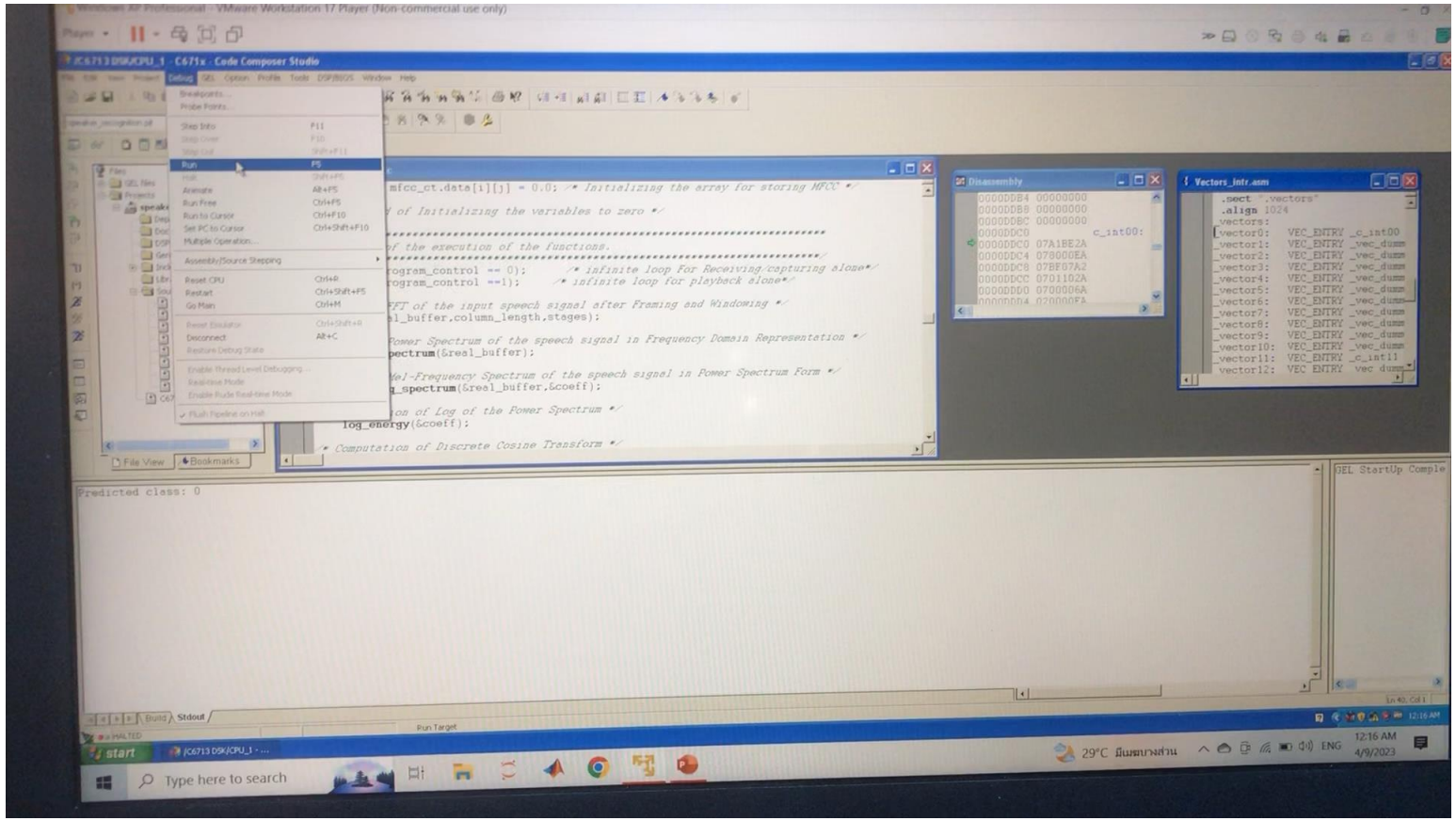
Snore Test Video

KNN, K=1, Silent Sound



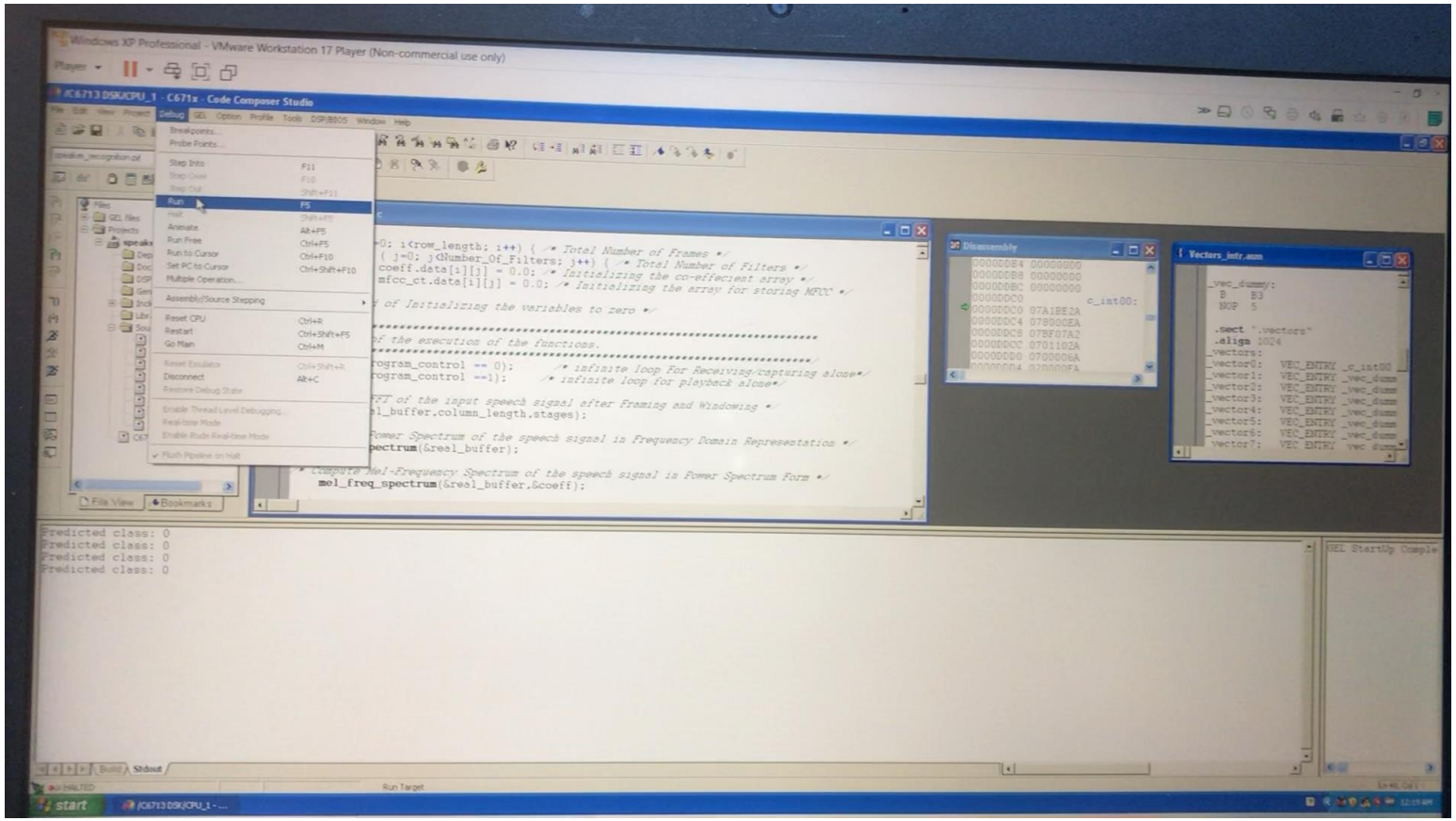
Snore Test Video

KNN, K=1, Noise Sound



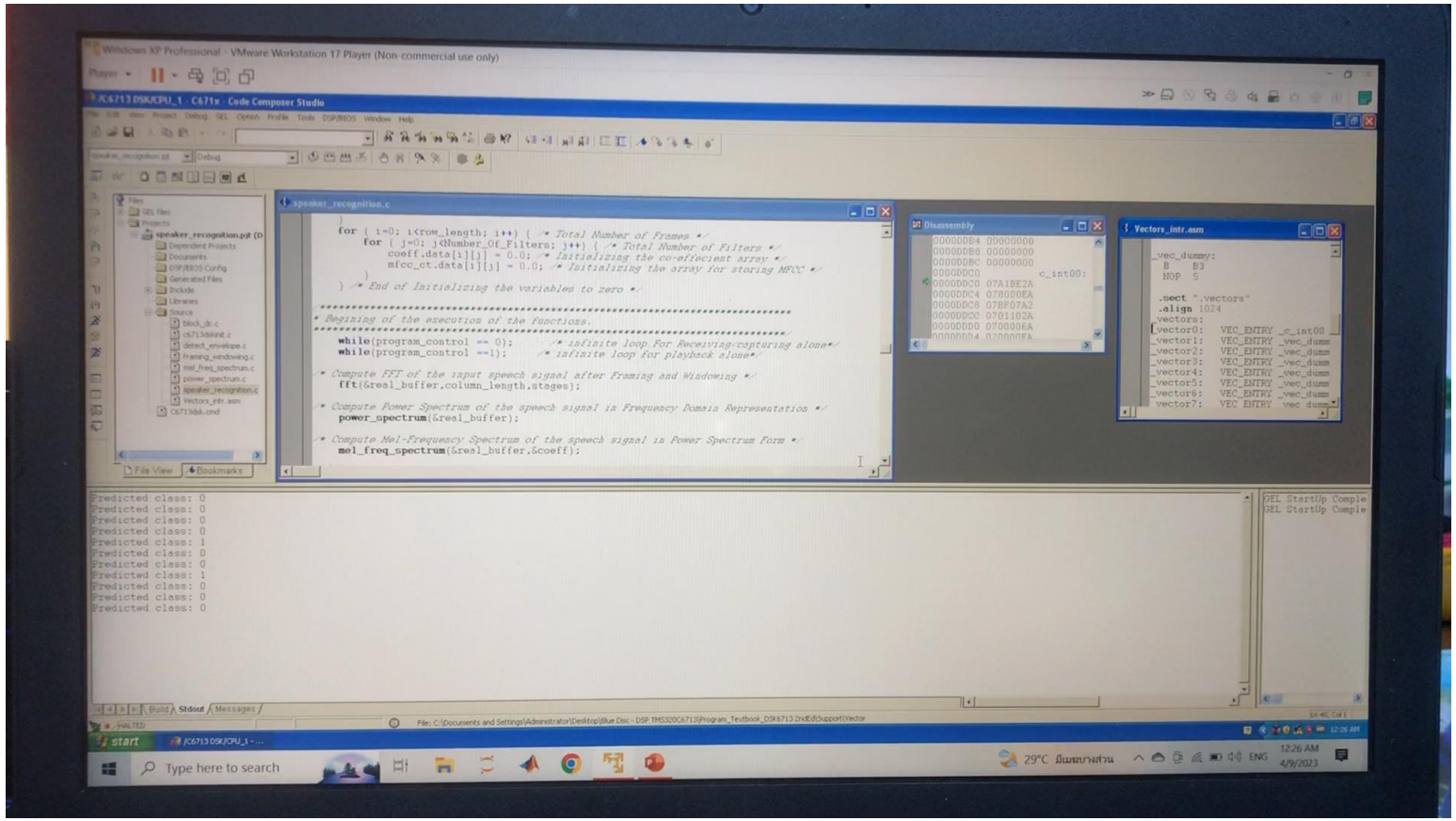
Snore Test Video

KNN, K=1, Snore sound (Real)



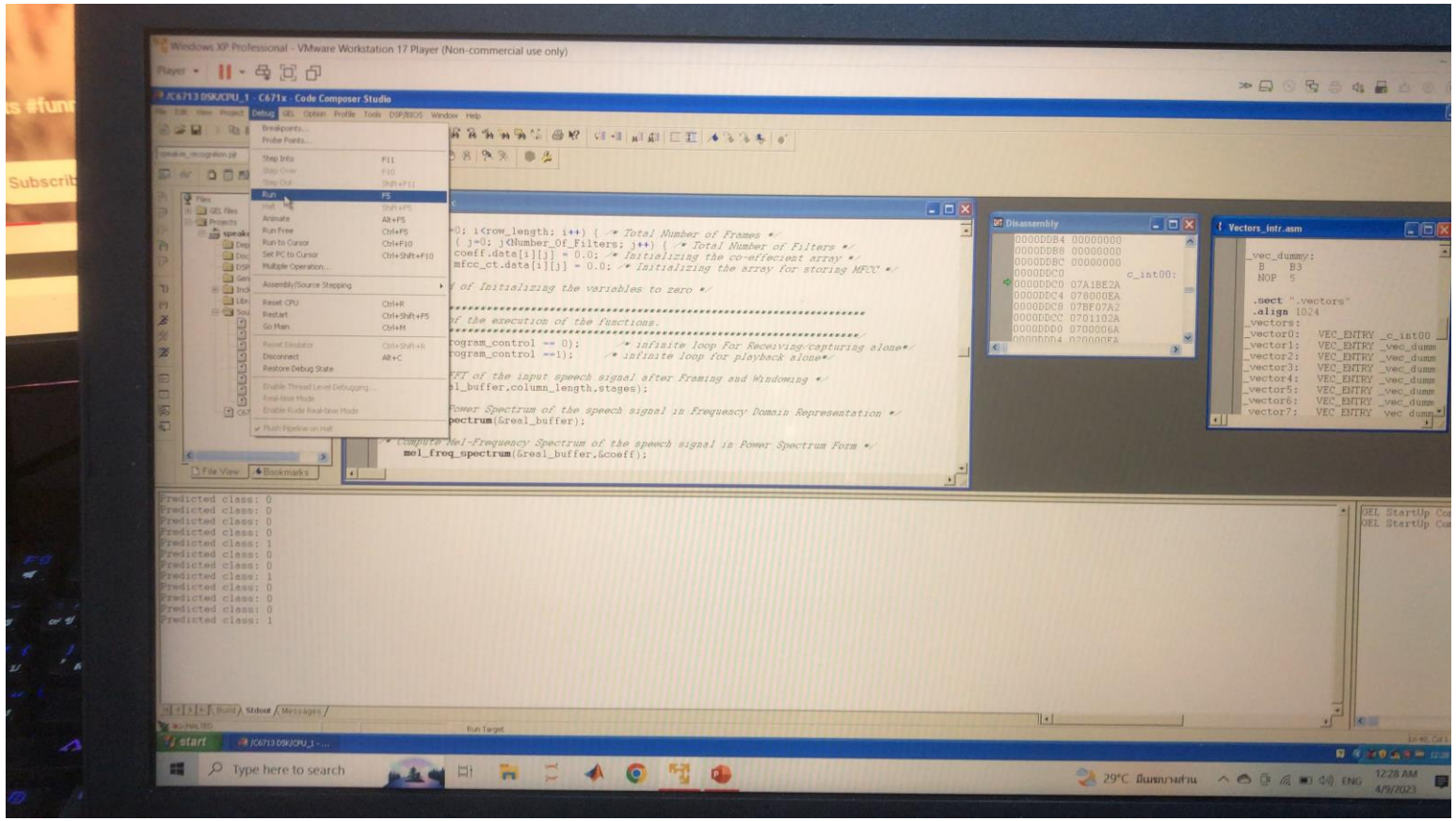
Snore Test Video

KNN, K=1, Snore sound YouTube



Snore Test Video

KNN, K=1, Snore sound YouTube



Wiwitthawin chareonngam

6310501933

Thank you