



Laboratorium Technologii Mowy

Projekt 1 - raport z trzech tygodni

Wydział: IMiR

Kierunek: Inżynieria Akustyczna

Grupa projektowa D: Agnieszka Fatla, Natalia Hebda, Mateusz Janik

tydzień #1

W pierwszym tygodniu realizacji projektu wykonano zadania:

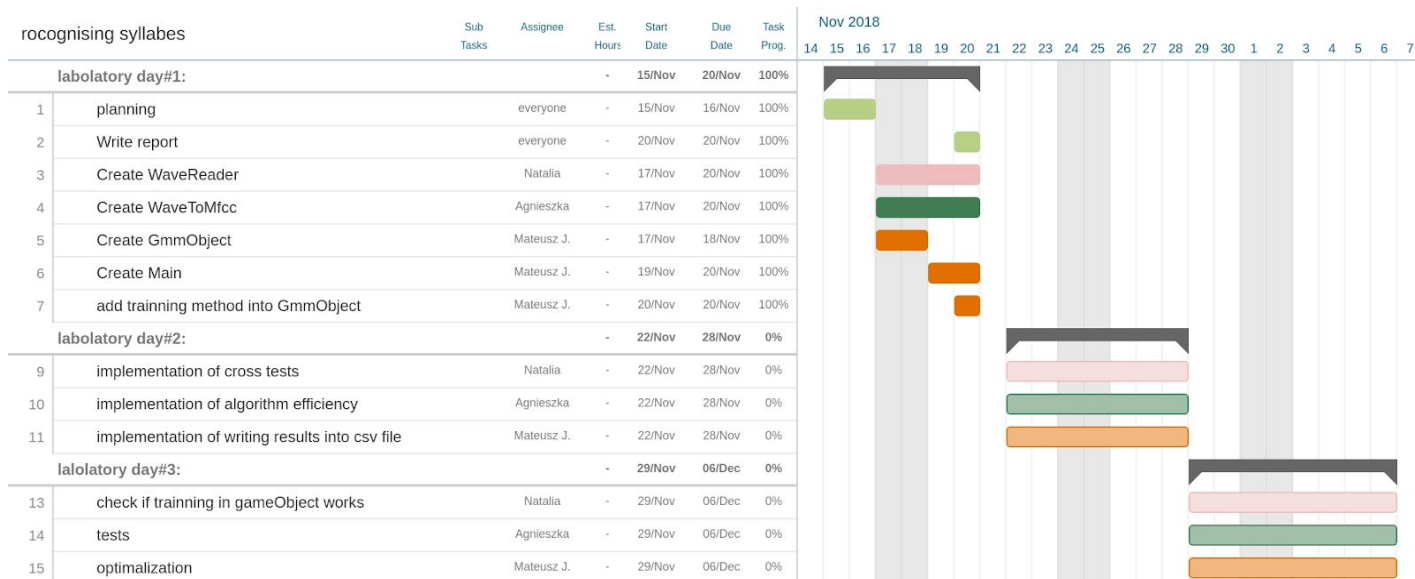
1. Wczytanie plików audio.
2. Odczytanie etykiet klas z nazw pliku.
3. Parametryzacja plików audio i zapis cech (wyznaczenie macierzy MFCC).
4. Trening modelu GMM.

Rozkład prac pomiędzy członków grupy przedstawiono na diagramie Gantta (rys. nr 1).

Kod programu zawiera 4 klasy. Klasa WaveReader wczytuje pliki audio, zapisuje sygnały w nich obecne do macierzy, w której w każdym wierszu są sygnały dla dziesięciu cyfr oraz imię wypowiadającej je osoby. Druga z klas WaveToMfcc z odczytanych plików tworzy macierze MFCC o tym samym układzie (każdy wiersz to inny mówca, każda kolumna to macierz MFCC dla innej cyfry; ostatnia kolumna zawiera imię mówcy), następnie utworzono listę, w której każdy kolejny element tworzy jedno wielkie MFCC dla danej liczby, sklejonego z MFCC każdego mówcy. Klasa GmmObject korzysta z biblioteki sklearn i wykorzystuje Gaussian mixture w celu parametryzacji plików audio.

W klasie Main następuje wywołanie potrzebnych funkcji z trzech klas, stworzenie tabeli z modelami GMM dla każdej z cyfr oraz trening tych modeli.

Kod źródłowy przedstawiono na rysunkach nr 2, 3, 4, 5.



Rys. 1. Diagram Gantta.

```

1  """this is wave file connector and reader"""
2
3  import ...
4
5
6
7  class WaveReader(object):
8
9      def __init__(self, path_folder):
10         self.path_folder_ = path_folder
11
12     def read_all(self):
13         matrix = np.zeros((22, 10), dtype=object)
14         names = np.chararray((22, 1), itemsize=5, unicode=True)
15         rate = 0
16         for i in range(0, 10):
17             i2 = 0
18             for file in os.listdir(self.path_folder_):
19                 if file.endswith("" + str(i) + '_.wav'):
20                     path = os.path.join(self.path_folder_, file)
21                     (rate, number_1) = wav.read(path)
22                     matrix[i2][i] = number_1
23                     i2 = i2 + 1
24                 if i == 9:
25                     filename, file_extension = os.path.splitext(os.path.basename(path))
26                     name = filename[0:-3]
27                     names[i2 - 1, 0] = name
28         matrix = np.concatenate((matrix, names), axis=1)
29         return (matrix, rate)
30
31

```

Rys. 2. Kod klasy WaveReader.

```

1  """ This program will convert wave files to MFCC """
2
3  import numpy as np
4  from python_speech_features import mfcc
5
6  class WaveToMfcc(object):
7
8      def __init__(self, array, rate):
9         self.wave_array_ = array
10         self.rate_ = rate
11
12     def create_mfcc(self):
13         mfcc_array = self.wave_array_
14         for row in range(0, len(self.wave_array_)):
15             for index in range(0, len(self.wave_array_[0]) - 1):
16                 mfcc_array[row][index] = mfcc(self.wave_array_[row][index], self.rate_, appendEnergy=False)
17         output = []
18         for i in range(0, 10):
19             mfcc_i = mfcc_array[0, i]
20             for row in range(1, len(mfcc_array)):
21                 mfcc_i = np.concatenate((mfcc_i, mfcc_array[row][i]), axis=0)
22             output.append(mfcc_i)
23         return output
24

```

Rys. 3. Kod klasy WaveToMfcc.

```

1  """this program is going to training data from MFCC input"""
2
3  import sklearn.mixture.gaussian_mixture
4
5  class GmmObject(object):
6
7      # constructor
8
9      def __init__(self, n_components, n_init_in, mfcc_input):
10         self.gmm_ = sklearn.mixture.gaussian_mixture.GaussianMixture(n_components=n_components,
11                                                                         max_iter=2000, init_params='random',
12                                                                         random_state=20, n_init=n_init_in)
13
14         self.untrained = True
15         self.mfcc_ = mfcc_input
16
17     # methods:
18
19     def train_data(self):
20         self.gmm_.fit(self.mfcc_)
21         self.untrained = False
22         print("Training successful")
23         print("number of iterations: " + str(self.gmm_.n_iter_))
24
25     def recognize(self, mfcc_input):
26         if self.untrained:
27             print("GMM object wasn't trained yet!")
28             return -99999
29         else:
30             outputlog = self.gmm_.score(mfcc_input)
31             if outputlog > -15:
32                 print("Object was recognised")
33                 return outputlog
34             elif outputlog > -30 and outputlog <= -15:
35                 print("Object was barely recognised")
36                 return outputlog
37             else:
38                 print("Object wasn't recognised")
39                 return outputlog

```

Rys. 4. Kod klasy GmmObject.

```

1  """ this is main commander of a program"""
2
3  import GmmObject
4  import WaveReader
5  import WaveToMfcc
6
7  reader = WaveReader.WaveReader("train")
8  (signals, rate) = reader.read_all()
9  converter = WaveToMfcc.WaveToMfcc(signals, rate)
10 mfcc_array = converter.create_mfcc()
11
12 table = []
13 for each in range(0, 9):
14     table.append(GmmObject.GmmObject(16, 20, mfcc_array[each]))
15
16 for each in table:
17     each.train_data()
18
19 print("YAY")
20
21

```

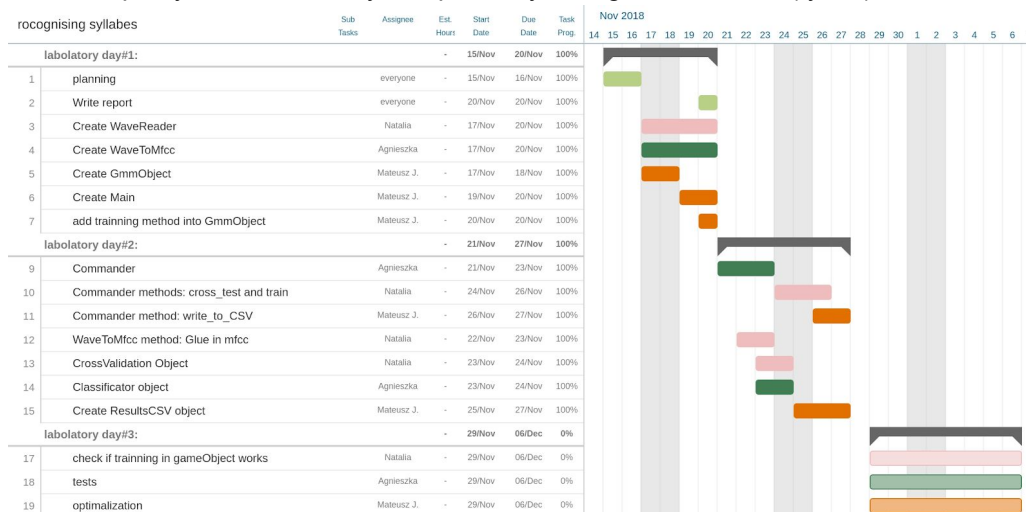
Rys. 5. Kod klasy Main.

tydzień #2

W drugim tygodniu wykonano:

1. walidację krzyżową
2. stworzenie klasy zarządzającej całym programem
3. serię testów różnych wariantów walidacji krzyżowej
4. zapis rezultatów do pliku results.csv

Rozkład pracy można zobaczyć na poniższym diagramie Gantt'a (rys. 6)



Rys. 6 Diagram Gantt'a dla drugiego tygodnia pracy

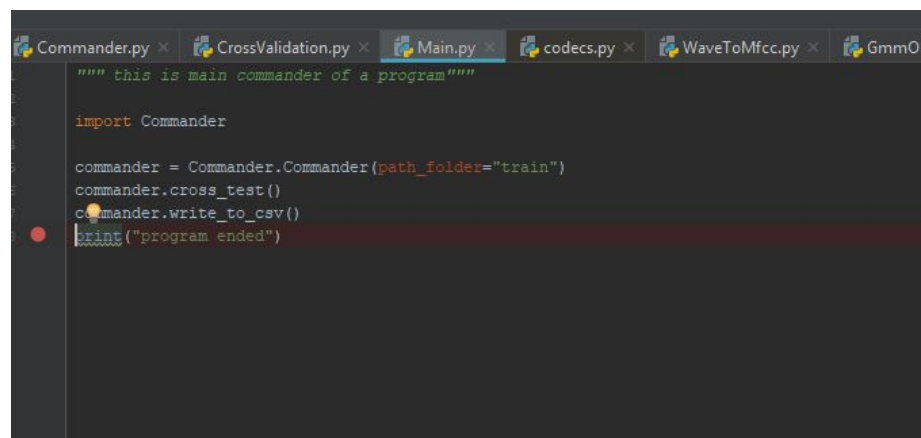
Do projektu zostały dodane 4 nowe klasy: Commander, Classifier, CrossValidation oraz ResultsCSV. Modyfikacji uległa cała struktura projektu jak i sposób wywoływania poszczególnych metod.

1. Podgląd części wyników:

```
1 Crossvalidation type: ,0
2
3 ADIMI_0_wav, recognised: ,0.0
4
5 ADIMI_1_wav, recognised: ,1.0
6
7 ADIMI_2_wav, recognised: ,2.0
8
9 ADIMI_3_wav, recognised: ,3.0
10
11 ADIMI_4_wav, recognised: ,4.0
12
13 ADIMI_5_wav, recognised: ,5.0
14
15 ADIMI_6_wav, recognised: ,6.0
16
17 ADIMI_7_wav, recognised: ,7.0
18
19 ADIMI_8_wav, recognised: ,8.0
20
21 ADIMI_9_wav, recognised: ,5.0
22
23 BCIMI_0_wav, recognised: ,0.0
24
25 BCIMI_1_wav, recognised: ,1.0
26
27 BCIMI_2_wav, recognised: ,2.0
28
29 BCIMI_3_wav, recognised: ,3.0
30
31 BCIMI_4_wav, recognised: ,4.0
32
33 BCIMI_5_wav, recognised: ,5.0
34
35 BCIMI_6_wav, recognised: ,6.0
36
37 BCIMI_7_wav, recognised: ,7.0
38
39 BCIMI_8_wav, recognised: ,8.0
40
41 BCIMI_9_wav, recognised: ,5.0
42
43 Recognition ratio: ,0.9
44
45 Crossvalidation type: ,1
46
47 DGIMI_0_wav, recognised: ,0.0
48
49 DGIMI_1_wav, recognised: ,1.0
50
51 DGIMI_2_wav, recognised: ,2.0
52
53 DGIMI_3_wav, recognised: ,4.0
54
55 DGIMI_4_wav, recognised: ,4.0
56
57 DGIMI_5_wav, recognised: ,5.0
58
59 DGIMI_6_wav, recognised: ,6.0
60
61 DGIMI_7_wav, recognised: ,7.0
62
63 DGIMI_8_wav, recognised: ,8.0
64
```

Rys. 7. podgląd Results.csv

2. Kod źródłowy przedstawiają poniższe rysunki:

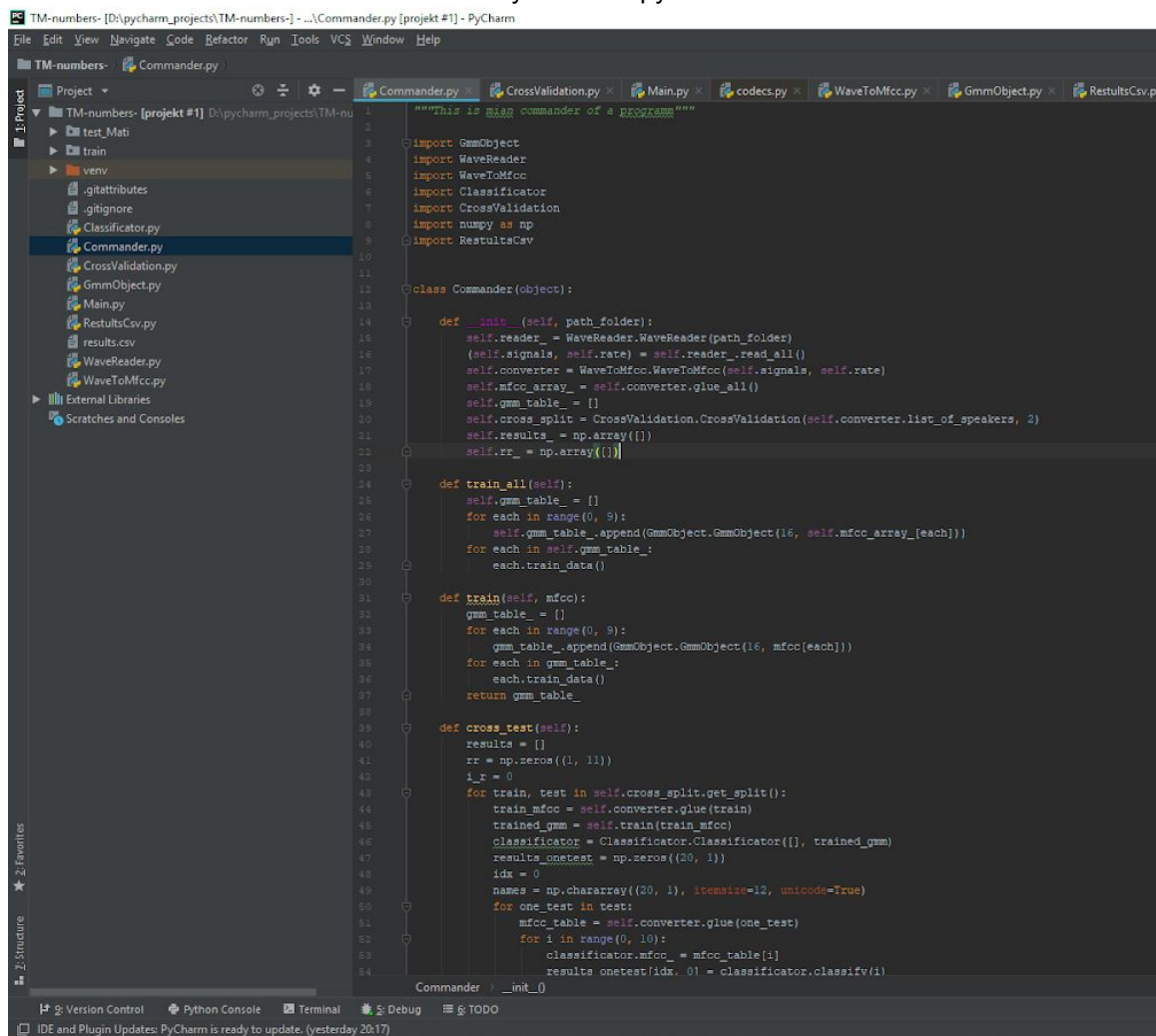


```
""" this is main commander of a program """

import Commander

commander = Commander.Commander(path_folder="train")
commander.cross_test()
commander.write_to_csv()
print("program ended")
```

Rys. 8 main.py



```
"""This is main commander of a program"""

import GmmObject
import WaveReader
import WaveToMfcc
import Classifier
import CrossValidation
import numpy as np
import ResultsCsv

class Commander(object):

    def __init__(self, path_folder):
        self.reader_ = WaveReader.WaveReader(path_folder)
        (self.signals, self.rate) = self.reader_.read_all()
        self.converter = WaveToMfcc.WaveToMfcc(self.signals, self.rate)
        self.mfcc_array_ = self.converter.glue_all()
        self.gmm_table_ = []
        self.cross_split_ = CrossValidation.CrossValidation(self.converter.list_of_speakers, 2)
        self.results_ = np.array([])
        self.tr_ = np.array([])

    def train_all(self):
        self.gmm_table_ = []
        for each in range(0, 9):
            self.gmm_table_.append(GmmObject.GmmObject(16, self.mfcc_array_[each]))
        for each in self.gmm_table_:
            each.train_data()

    def train(self, mfcc):
        gmm_table_ = []
        for each in range(0, 9):
            gmm_table_.append(GmmObject.GmmObject(16, mfcc[each]))
        for each in gmm_table_:
            each.train_data()
        return gmm_table_

    def cross_test(self):
        results = []
        rr = np.zeros((1, 11))
        i_r = 0
        for train, test in self.cross_split_.get_split():
            train_mfcc = self.converter.glue(train)
            trained_gmm = self.train(train_mfcc)
            classifier = Classifier.Classifier([], trained_gmm)
            results_onetest = np.zeros((20, 1))
            idx = 0
            names = np.chararray((20, 1), itemsize=12, unicode=True)
            for one_test in test:
                mfcc_table = self.converter.glue(one_test)
                for i in range(0, 10):
                    classifier.mfcc_ = mfcc_table[i]
                    results_onetest[idx, 0] = classifier.classify(i)
```

Rys. 9 Commander.py część 1


```

30
31 def train(self, mfcc):
32     gmm_table = []
33     for each in range(0, 9):
34         gmm_table.append(GmmObject.GmmObject(16, mfcc[each]))
35     for each in gmm_table:
36         each.train_data()
37     return gmm_table
38
39 def cross_test(self):
40     results = []
41     rr = np.zeros((1, 11))
42     i_r = 0
43     for train, test in self.cross_split.get_split():
44         train_mfcc = self.converter.glue(train)
45         trained_gmm = self.train(train_mfcc)
46         classifier = Classifier.Classifier([], trained_gmm)
47         results_onetest = np.zeros((20, 1))
48         idx = 0
49         names = np.chararray((20, 1), itemsize=12, unicode=True)
50         for one_test in test:
51             mfcc_table = self.converter.glue(one_test)
52             for i in range(0, 10):
53                 classifier.mfcc = mfcc_table[i]
54                 results_onetest[idx, 0] = classifier.classify(i)
55                 names[idx, 0] = self.converter.list_of_speakers[one_test] + "_" + str(i) + '._wav'
56                 idx += 1
57             results_onetest = np.concatenate((names, results_onetest), axis=1)
58             results.append(results_onetest)
59             rr[i_r, 0] = classifier.get_RR()
60             i_r += 1
61
62     self.results_ = results
63     self.rr_ = rr
64     return results, rr
65
66 def write_to_csv(self):
67     temp = np.array([])
68
69     if self.results_ == temp or self.rr_ == temp:
70         print("NOTHING TO WRITE")
71     else:
72         writer = ResultsCsv.ResultsCsv(self.results_, self.rr_)
73         writer.write_to_csv()
74
75

```

Rys. 10 Commander.py część 2

```

1 import numpy as np
2
3
4 class Classifier(object):
5     def __init__(self, mfcc_input, gmm_list_input):
6         self.mfcc_ = mfcc_input
7         self.gmm_list = gmm_list_input
8         self.n_correct = 0
9         self.n_iterations = 0
10
11     # methods:
12     def classify(self, mfcc_digit):
13         self.n_iterations += 1
14         scores_list = []
15         for gmm in self.gmm_list:
16             gmm = gmm.gmm_score(self.mfcc_)
17             scores_list.append(gmm)
18         max_likelihood = np.max(scores_list)
19         if scores_list.index(max_likelihood) == mfcc_digit:
20             self.n_correct += 1
21         return scores_list.index(max_likelihood)
22
23     def get_RR(self):
24         return self.n_correct/self.n_iterations
25

```

Rys. 11. Classifier.py

```

1 from sklearn.model_selection import KFold
2
3
4 class CrossValidation(object):
5     def __init__(self, x, n_test):
6         self.n_test_ = n_test
7         self.x_ = x.T
8
9     def get_split(self):
10        kf = KFold(n_splits=int(len(self.x_)/self.n_test_))
11        return kf.split(self.x_)
12

```

Rys. 12. CrossValidation.py

```

1 """this program is going to train data from MFCC input"""
2
3 import sklearn.mixture.gaussian_mixture
4
5
6 class GmmObject(object):
7
8     # constructor
9     def __init__(self, n_components, mfcc_input):
10        self.gmm = sklearn.mixture.gaussian_mixture.GaussianMixture(n_components=n_components, random_state=20)
11        self.untrained = True
12        self.mfcc_ = mfcc_input
13
14     # methods:
15     def train_data(self):
16        self.gmm.fit(self.mfcc_)
17        self.untrained = False
18        print("Training successful")
19        print("number of iterations: " + str(self.gmm.n_iter_))
20
21     def recognize(self, mfcc_input):
22        if self.untrained:
23            print("GMM object wasn't trained yet!")
24            return -99999
25        else:
26            return self.gmm.score(mfcc_input)
27

```

Rys. 13. GmmObject.py

```

1 """ this program is going to write results to csv"""
2
3 import csv
4
5
6 class ResultsCsv(object):
7
8     # constructor
9
10    def __init__(self, results_matrix, rr):
11        self.results_ = results_matrix
12        self.rr_ = rr
13
14    # methods
15
16    def write_to_csv(self):
17
18        with open('results.csv', mode='w') as results_file:
19            results_writer = csv.writer(results_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
20
21            for site in range(0, 10):
22                site_file = self.results_[site]
23                results_writer.writerow(['Crossvalidation type: ' + str(site)])
24
25                for each in site_file:
26                    results_writer.writerow([str(each[0]) + ' recognised: ' + str(each[1])])
27
28                results_writer.writerow(['Recognition ratio: ' + str(self.rr_[0, site])])
29
30        results_file.close()
31

```

Rys. 14. ResultsCSV.py


```

1  """this is wave file connector and reader"""
2
3  import scipy.io.wavfile as wav
4  import numpy as np
5  import os
6
7
8  class WaveReader(object):
9
10     def __init__(self, path_folder):
11         self.path_folder_ = path_folder
12
13     def read_all(self):
14         matrix = np.zeros((22, 10), dtype=object)
15         names = np.chararray((22, 1), itemsize=5, unicode=True)
16         rate = 0
17         for i in range(0, 10):
18             i2 = 0
19             for file in os.listdir(self.path_folder_):
20                 if file.endswith(" + str(i) + '.wav')":
21                     path = os.path.join(self.path_folder_, file)
22                     (rate, number_1) = wav.read(path)
23                     matrix[i2][i] = number_1
24                     i2 = i2 + 1
25                 if i == 9:
26                     filename, file_extension = os.path.splitext(os.path.basename(path))
27                     name = filename[0:-3]
28                     names[i2 - 1, 0] = name
29             matrix = np.concatenate((matrix, names), axis=1)
30             return (matrix, rate)
31
32

```

Rys. 15. WaveReader.py

```

1  """ This program will convert wave files to MFCC"""
2
3  import numpy as np
4  from python_speech_features import mfcc
5
6
7  class WaveToMfcc(object):
8
9     def __init__(self, array, rate):
10         self.wave_array_ = array
11         self.rate_ = rate
12         self.mfcc_array_ = self.create_mfcc()
13         self.list_of_speakers = self.mfcc_array_[1, 10]
14
15     def create_mfcc(self):
16         mfcc_array = self.wave_array_
17         for row in range(0, len(self.wave_array_)):
18             for index in range(0, len(self.wave_array_[0]) - 1):
19                 mfcc_array[row][index] = mfcc(self.wave_array_[row][index], self.rate_, appendEnergy=False)
20         return mfcc_array
21
22     def glue(self, indexes):
23         output = []
24         for i in range(0, 10):
25             if not isinstance(indexes, np.int32):
26                 mfcc_i = self.mfcc_array_[indexes[0], i]
27                 indexes2 = np.delete(indexes, 0)
28                 for row in range(1, len(self.mfcc_array_)):
29                     if row in indexes2:
30                         mfcc_i = np.concatenate((mfcc_i, self.mfcc_array_[row][i]), axis=0)
31             else:
32                 mfcc_i = self.mfcc_array_[indexes, i]
33             output.append(mfcc_i)
34         return output
35
36     def glue_all(self):
37         output = []
38         for i in range(0, 10):
39             mfcc_i = self.mfcc_array_[0, i]
40             for row in range(1, len(self.mfcc_array_)):
41                 mfcc_i = np.concatenate((mfcc_i, self.mfcc_array_[row][i]), axis=0)
42             output.append(mfcc_i)
43         return output
44

```

Rys. 16. WaveToMfcc.py

tydzień #3

W trzecim tygodniu wykonano:

1. Oszacowanie empiryczne najlepszej architektury systemu klasyfikacji cyfr

a) Cechy MFCC

- Długość ramki FFT
- Liczba filtrów melowych
- Liczba współczynników cepstralnych
- Wyłączenie filtra preemfazy
- Wyłączenie lifteringu
- Normalizacja cech:
 - Cepstral Mean Subtraction
 - Cepstral Mean Variance Normalisation

b) Model GMM

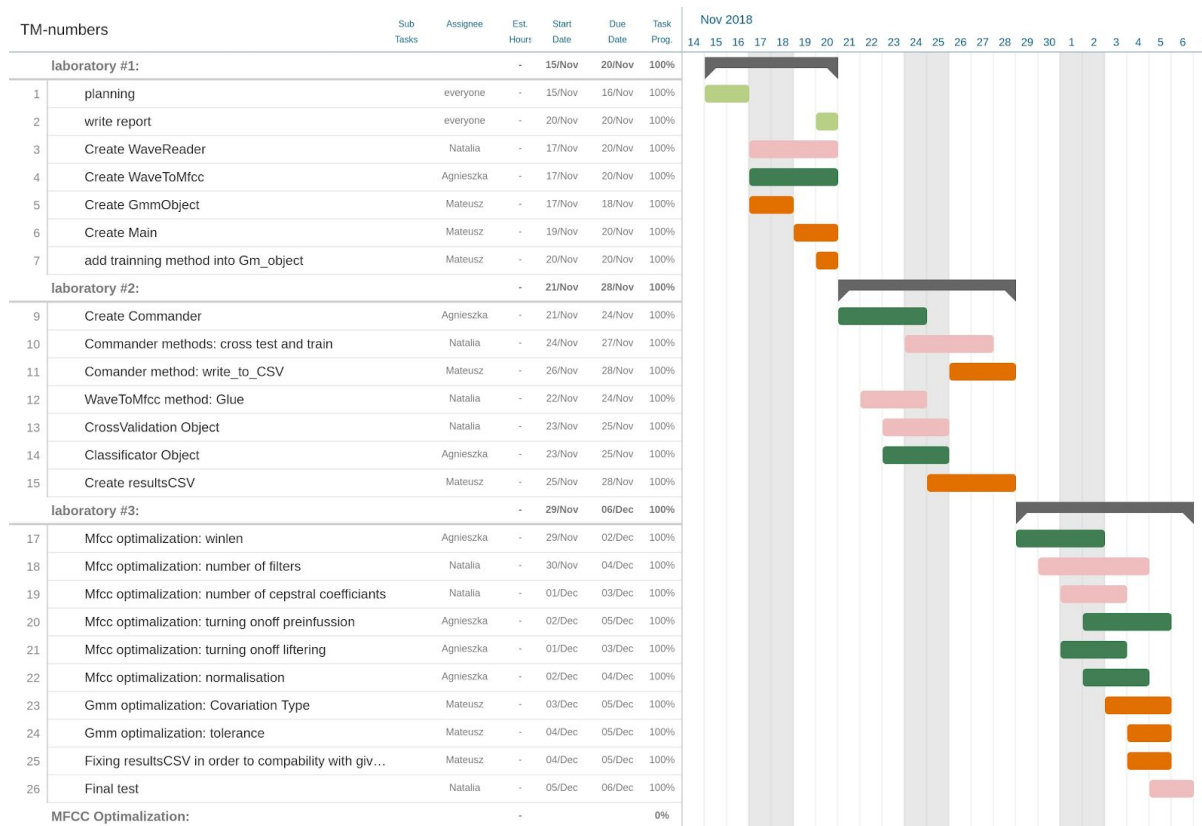
- Pełna lub diagonalna macierz kowariancji
- Modyfikacja "stopu" treningu - zmiana tolerancji

c) Normalizacja wyników

- Log-likelihood ratio

2. Ewaluacja systemu na "nieznanym" zbiorze

Rozkład pracy można zobaczyć na poniższym diagramie Gantt'a (rys. 17)



Rys. 17. Diagram Gantt'a dla trzeciego tygodnia pracy.

Dla każdego ze sposobów zmiany parametrów (i innych sposobów ulepszenia modelu) z punktu nr 1 wyznaczono RR (recognition ratio). Testy przeprowadzono niezależnie, to znaczy dla każdej zmiany resztę parametrów, funkcji itd. pozostawiano jak na początku z domyślnymi wartościami.

RR w teście z wartościami początkowymi wyniosło 0.79.

Dla normalizacji log-likelihood wynik pozostał bez zmian.

Cechy *Mfcc*:

Tabela 1. RR w zależności od długości ramki FFT.

długość ramki [ms]	RR
5	0.82
10	0.76
15	0.78
20	0.75
25	0.79
30	0.74
35	0.79
40	0.77

Tabela 2. RR w zależności od liczby filtrów melowych.

liczba filtrów melowych	RR
28	0.76
29	0.77
30	0.8
31	0.8
32	0.79
35	0.78
37	0.78
40	0.8
45	0.77

Tabela 3. RR w zależności od liczby współczynników cepstralnych.

liczba współczynników cepstralnych	RR
5	0.8
6	0.82
7	0.84

8	0.8
9	0.8
10	0.8
11	0.8
12	0.8
14	0.77
15	0.75
20	0.68

Tabela 4. RR w zależności od pozostałych zmian MFCC.

zmieniony parametr	RR
wyłączony filtr preemfazy	0.74
wyłączony liftering	0.82
Cepstral Mean Subtraction	0.87
Cepstral Mean Variance Normalisation	0.89
liczba współczynników cepstralnych: 7, liczba filtrów melowych: 30	0.85

Cechy Gaussian mixture:

Tabela 5. RR w zależności od tolerancji

tolerance:	RR
1e-1	0.74
1e-2	0.78
1e-3	0.76

Tabela 6. RR w zależności od typu kowariancji

Covariance type:	RR
“full”	0.78
“tied	0.76
“diag”	0.57
“spherical”	0.57

Tabela 7. RR w zależności od n_init

n_init:	RR
1	0.78
2	0.79
3	0.79
4	0.79
5	0.79

Tabela 8. RR w zależności od init_params

init_params:	RR
'kmeans'	0.79
'random'	0.71

warm_start	RR
True	0.79
False	0.79

Ostatecznie najlepsze RR uzyskano dla następujących ustawień, wyniosło ono:

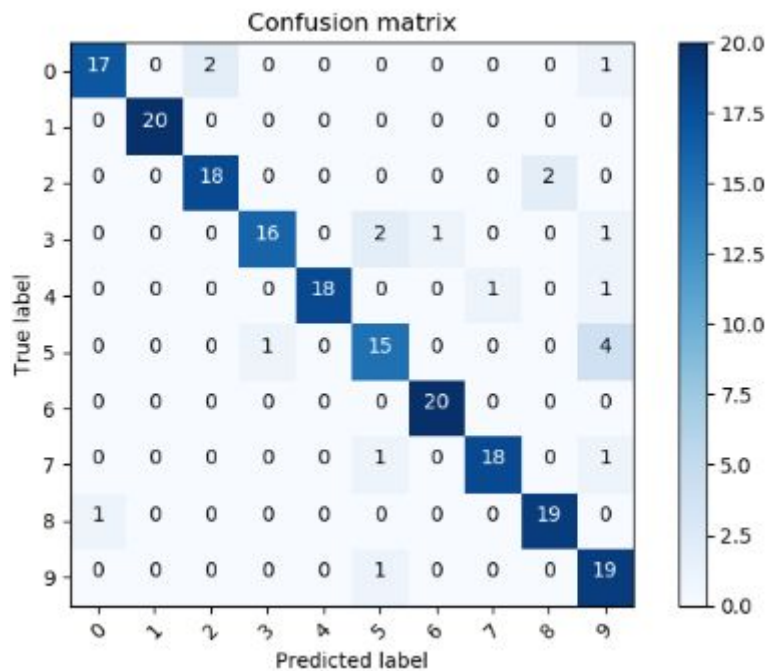
RR = 0.95

Tabela 9. Podsumowanie najlepszych parametrów (RR największe).

MFCC	
długość ramki [ms]	15
liczba filtrów melowych	30
liczba współczynników cepstralnych	7
wyłączony liftering	
Cepstral Mean Variance Normalisation	
GMM	
Covariance type:	"full"

test na zbiorze ewaluacyjnym:

RR = 0.9



Rys. 18. Macierz otrzymana po uruchomieniu dołączonego skryptu

Zmiany w kodzie źródłowym względem poprzedniego tygodnia:

```
external_evaluator.py x Classifier.py x Commander.py x ResultsCsv.py x GmmObject.py x Main.py x
1 """ this program is going to write results to csv"""
2
3 import csv
4
5
6 class ResultsCsv(object):
7     # constructor
8
9     def __init__(self, results_matrix, rr, name):
10         self.name_ = name
11         self.results_ = results_matrix
12         self.rr_ = rr
13     # methods
14
15     def write_to_csv(self):
16
17         with open(str(self.name_), mode='w', newline='') as results_file:
18             results_writer = csv.writer(results_file, delimiter=',')
19             for site in range(0, 11):
20                 site_file = self.results_[site]
21
22                 for each in site_file:
23                     results_writer.writerow([str(each[0])] + [str((each[1][0]))] + [each[2][0:5]])
24
25         results_file.close()
26         print("Recognition ratio: " + str(self.rr_))
27
28
```

Rys. 19. Poprawione resultsCsv.py

```

def cross_test(self):
    results = []
    rr = np.zeros((1, 11))
    i_r = 0
    for train, test in self.cross_split.get_split():
        train_mfcc = self.converter.glue(train)
        trained_gmm = self.train(train_mfcc)
        classifier = Classifier.Classifier([], trained_gmm)
        results_onetest = np.zeros((20, 1))
        results_likelyhoods = np.zeros((20, 1))
        names = np.chararray((20, 1), itemsize=12, unicode=True)
        for one_test in test:
            mfcc_table = self.converter.glue(one_test)
            for i in range(0, 10):
                classifier.mfcc_ = mfcc_table[i]
                results_onetest[idx, 0] = classifier.classify(i)[0]
                results_likelyhoods[idx, 0] = classifier.classify(i)[1]
                names[idx, 0] = self.converter.list_of_speakers[one_test] + "_" + str(i) + '_.wav'
                idx += 1
            results_onetest = np.concatenate((names, results_onetest, results_likelyhoods), axis=1)
            results.append(results_onetest)
            rr[0, i_r] = classifier.get_RR()
            i_r += 1

    self.results_ = results
    rr_i = np.mean(rr)
    self.rr_ = rr_i
    return results, rr_i

```

Typo: In word 'onetest' more... (Ctrl+F1)

Rys. 20. Zmiana w metodzie cross_test w Commanderze

```

# methods:
def classify(self, mfcc_digit):
    self.n_iterations += 1
    scores_list = []
    for gmm in self.gmm_list:
        gmm = gmm.gmm_.score(self.mfcc_)
        scores_list.append(gmm)
    max_likelihood = np.max(scores_list)
    if scores_list.index(max_likelihood) == mfcc_digit:
        self.n_correct += 1
    return scores_list.index(max_likelihood), max_likelihood

```

Rys. 21. Zmiana w metodzie Classify w Classifier.py

Przystosowanie kodu do typu danych w zbiorze ewaluacyjnym :

```

def eval_test(self):
    results = []
    train_mfcc = self.converter.glue_all()
    trained_gmm = self.train(train_mfcc)
    classifier = Classifier.Classificator([], trained_gmm)
    idx = 0
    path_ = "eval"
    keys = EvaluateObject.load_keys()
    for file in os.listdir(path_):
        path = os.path.join(path_, file)
        (rate, number_1) = wav.read(path)
        mfcc_table = mfcc(number_1, rate, appendEnergy=False, winlen=0.015, nfilt=30, numcep=7, ceplifter=0)
        mfcc_table -= np.mean(mfcc_table, axis=0)
        mfcc_table = mfcc_table / np.std(mfcc_table)
        classifier.mfcc_ = mfcc_table
        name = os.path.basename(path)
        (result, results_likelyhoods) = classifier.classify(keys[os.path.basename(path)])
        idx += 1
        result = np.array((name, result, results_likelyhoods), dtype=object)
        results.append(result)
    rr = classifier.get_RR()
    self.results_ = results
    self.rr_ = rr
    return results, rr

def write_to_csv_else(self, file_name):
    temp = np.array([])
    if self.results_ == temp or self.rr_ == temp:
        print("NOTHING TO WRITE")
    else:
        writer = ResultsCsv.ResultsCsv(self.results_, self.rr_, file_name)
        writer.write_to_csv_else()
    EvaluateObject.evaluate()

```

Rys. 22. Dodane nowe metody do testu i zapisu, oraz wywołania funkcji evaluate w Commander.py

```

def write_to_csv_else(self):
    with open(str(self.name_), mode='w', newline='') as results_file:
        results_writer = csv.writer(results_file, delimiter=',')
        for site in range(0, 200):
            each = self.results_[site]
            results_writer.writerow(each)

    results_file.close()
    print("Recognition ratio: " + str(self.rr_))

```

Rys. 23. Dodana nowa metoda do zapisu w ResultsCsv.py

1	001.wav,4,-4.048214485170623
2	002.wav,9,-6.706395122381126
3	003.wav,0,-4.533058120860486
4	004.wav,3,-5.048576692877666
5	005.wav,7,-3.183948231272118
6	006.wav,9,-3.6330536844957333
7	007.wav,0,-6.667659667536857
8	008.wav,2,-3.581027143363664
9	009.wav,9,-4.20916682052034
10	010.wav,2,-4.101715208196968
11	011.wav,1,-3.8030384891554982
12	012.wav,2,-1.4758171423721844
13	013.wav,5,-2.362041718738132
14	014.wav,7,-4.17103108991303
15	015.wav,8,-6.425261967657905
16	016.wav,3,-5.671238310414938
17	017.wav,2,-6.782352854258278
18	018.wav,6,-2.4371567329830413
19	019.wav,9,-2.9619929506611418
20	020.wav,4,-0.3922361897174838
21	021.wav,6,-5.5579257158259265

Rys. 24. Fragment wyników w pliku "results.csv".