Gizli Markov Modellərindən istifadə etməklə sözlərin avtomatik olaraq aşkar edilməsi

Xülasə **– Bu report Gizli Markov Modellərindən istifadə etməklə müxtəlif uzunluqlu 41 fərqli söz və söz birləşmələrinin avtomatik olaraq tanınması üçün method qurmaq və qurulan metodun nəticəsini optimallaşdırmağı əhatə edir.**

1 Giriş

Səs yazılarının müxtəlif vasitələrdən istifadə etməklə avtomatik olaraq aşkar edilməsi kompüter texnoogiyaları sahəsinin ən aktual mövzularından biridir. Avtomatik Səs Tanımasını *(AST)* həyata keçirmək üçün müxtəlif konsepsiyalar mövcudur. Bunlardan ən geniş istifadə edilən Gizli Markov Modeli *(GMM)* və Təkrarlanan Neyron Şəbəkə *(TNŞ)* modelləridir. Hər iki model dildəki müxtəlif strukturları (hecaların işlənmə sırası və ya hası səslər toplusunun hası xarakterik keyfiyyətləri göstərməsi kimi) öyrənmək üçün istifadə edilə bilər.

Biz modeli qurmaq üçün GMM-in xüsusi növü olan *Gauss GMM*-dən istifadə edəcəyik. Bundan əlavə oaraq, model qurarkən hər bir söz və söz birləşməsinin fonetik səviyyədə xarakteristikasını müəyyənləşdirmək üçün səsin qısa müddətli güc spektrinin təmsil edilməsi metodundan *(MFCC)* istidadə ediləcəkdir.

2 Metod

Bu layihədə *python* proqramlaşdırma dili istiadə edilərək tətbiq olunmuşdur. və Gauss GMM qurulması üçün *hmmlearn* və MFCC metodu üçün isə *python speech features* kitabxanalarından istifadə edilmişdir.

2.1 MFCC Tətbiqi

Verilmiş səs yazılarını (.wav) oxuyub hər birini ayrı-ayrı qruplaşdırdıqdan sonra əladə edilən hər bir səs yazısının tezlik aralıqlarına MFCC metodunu tətbiq etməklə hər bir söz üçün qısa müddətli güc spektri tapılır.

```
from scipy.io import wavfile
from python_speech_features import mfcc, logfbank

#ses yazisini oxu
sampling_freq, audio = wavfile.read("input.wav")

#mfcc deyerlerini oxu
mfcc_features = mfcc(audio, sampling_freq)
```

Listing 1: Səs yazısından güc spektri dəyərlərinin tapılması

MFCC metodunun tətbiqi alınan $Nx13$ ölçülü vektor qısa müddətli güc spektri dəyərləri özündə saxlayır və sözlərin ağız boşluğundan çıxarkən əmələ gətirdiyi səsin əsas keyfiyyətlərini xarakterizə edir. Bu dəyərlərdən istifadə etməklə biz uyğun modeli qura bilərik.

2.2 Dataset Qurulması və Miqyaslama

Hər bir söz və söz birləşməsi üçün MFCC dəyərlərini hsabladıqdan sonra yekunda ümumi uzunluğu 21433 olan data-set əldə edirik.

MFCC-dən alınan dəyərlərin x∈[-200,200] aralığında dəyişdiyini nəzərə alaraq optimallaşdırma algoritmasının daha sürətli və dəqiq olaraq optimal göstəriciyə yaxınlaşması üçün StandartScalar usulundan istifadə etməklə dəyərlərin miqyasını kiçildirik.

```
from sklearn.preprocessing import StandardScaler

# datani miqyasla
scaler = StandardScaler()
scaler.fit(data)
train_data = scaler.transform(data)
```

Listing 2: StandartScalar ilə miqyaslama

Növbəti mərhələdə, əlimizdə olan dataları təsadüfü olaraq qarışdırdıqdan sonra onları 0.15 nisbətindən *Təlim* və *Test* qruplarına ayırırıq.

Tabelle 1: Data qruplarının ölçüsü

Təlim	Test
18218	3215

Daha sonra *Təlim* qrupunda olan dataları, həmçinin, daha model optimalaşdırması zamanı k-qat çapraz doğrulama üsulu ilə modelin özünü test etmək üçün istifadə ediləcəkdir.

```
from sklearn.cross_validation import StratifiedShuffleSplit

#bolucunun teyin edilmesi
ShuffleSplit = StratifiedShuffleSplit(all_labels, test_size=0.1, random_state=0)

#datalarin bolunmesi
for telim_index, test_index in ShuffleSplit:
    X_telim,X_test = data[telim_index, ...], data[test_index, ...]
    y_telim,y_test = all_labels[telim_index],all_labels[test_index]
```

Listing 3: Təlim və Test qurplarının bölünməsi

2.3 Model və Təsnifatı

Hər bir 41 söz və söz birləşməsi üçün əlavə olaraq model qurulmuşdur. Qurulmuş model k-qatlama üsulu ilə opimizasiya edildikdən sonra, görülməmiş test datalarının üzərində yoxlanılır. Test zamanı hər bir söz sinf üçün bütün modelər yoxlanılır və ən yüksək dəyəri verən modelin nəticəsi həmin sinfin yekun çıxış nəticəsi kimi hesab olunur.

```
from hmmlearn import hmm

#modelin teyin olunmasi
speech_clf = hmm.GaussianHMM(n_components=3,
             n_iter=1000,
             algorith="viterbi"
             covariaance_type="diag"
             init_params="smtc",
             params="smtc")
```

Listing 4: İlkin model və parametrləri

- *n_components*: gizli vəziyyətlərin sayı
- *n_iter*: algoritmanın maksimum iterasiya sayı
- *algorith*: dekoder algoritması

- *covariaance_type*: optimalaşma yaxınlaşmasının dəyəri
- *params*: təlim prosesində hansı parametrlrin yeniləndiyini nəzarət edir edir.
- *init_params*: təlim prosesindən əvvəl hansı parametrlrin yeniləndiyini nəzarət edir edir.

3 Optimizasiya

İlkin göstəricilərlə öyrətdiyimiz *GMM* modelinin nəticəsi yaxşı olsada bizim istədiyimiz dəyəri vermədi, lakin qurduğumuz *GMM* modelinə ən uyğun olan parmetirləri tapmaqla yekun dəyəri istədiyimiz səviyyə catdıra bilərik. Bunun üçün biz k-qatlamalı çarpaz doğrulamalı optimizasiya üsulundan istifadə edəcəyik.

İlk olaraq axtarmaq istədiyimiz paramterləri təyin edirik.Daha sonra təyin olunan modellər üçün parametrlərin bütün mümkün kombinasiyaları optimizasiya zamanı yoxlanılacaqdır.

```
param_grid = [
    {
            'n_components': [3, 5, 10,15],
            'covariance_type': ['diag','full'],
            'algorithm': ['viterbi', 'map']
    }
]
```

Listing 5: Axtarmaq istədiyin parametrləri təyin et

```
for label in unique_labels:


    #datalari sinfine gore ayir
    data=seperate_l(X_train[y_tain==label],k_fold)
    data=np.array(data)

    #ayrilmis datani k-qatlama deyerini nezere alaraq hisselere bol
    select=[x for x in range(k_fold)]
    select=np.array(select)

    max_result=-10000000000
    best_model= None #en yaxs modeli saxlamaq ucun obyekt

    #her bir parametr kombinasiyasini yoxla
    for param_i,_ in enumerate(param_list):
        #K-qatlama hesablamasi burada olacaq
```

Listing 6: Bütün sinf sözlər üzrə parametr konbinasiyasının aparılması

Yekunda, k-qatlı çapraz doğrulama ilə tapılan ən yaxşı deyərlərlə modelimizi yenidən həmin dəyərlərl görə öyrədirik.

```
k_fold_result=[] # list for stroing result of k_folds

# her bir parametr kombinasiyasinda k_fold boyunca yoxla
for k in range(0,k_fold):

    # word_data-ni k-folda esasen bol
    train  = word_data[select!=k] # k_fold-1 train ucun
    test = word_data[select==k]   # 1-i test ucun

    # CONCATENATE TRAIN AND TEST word_data

    # do first word_data array concatenate
    train_word_data=np.array(train[0][0])
    test_word_data=np.array(test[0][0])


    # get lenght for each test&train word_data
```

```
    lengths_train=[train[0][0].shape[0]] #get first length
    lengths_test=[test[0][0].shape[0]] #get first length


    # do the rest of concatenation for train
    for i in range(0,train.shape[0]):
        for j in range(0,train[i].shape[0]):
            if (i!=0 or j!=0):
                train_word_data=np.append(train_word_data,train[i][j],axis=0)
                lengths_train.append(train[i][j].shape[0])

    # do the rest of concatenation for test
    for i in range(0,test.shape[0]):
        for j in range(0,test[i].shape[0]):
            if (i!=0 or j!=0):
                test_word_data=np.append(test_word_data,test[i][j],axis=0)
                lengths_test.append(test[i][j].shape[0])


    # telim datasini miqyasla
    scaler = StandardScaler()
    scaler.fit(train_word_data)
    train_word_data = scaler.transform(train_word_data)

    # test datasini miqyasla
    scaler = StandardScaler()
    scaler.fit(test_word_data)
    test_word_data = scaler.transform(test_word_data)

    # modelin parameterlerini qur
    hmm_trainer = HMMTrainer(
            n_components = param_list[param_i]['n_components'],
            algorithm = param_list[param_i]['algorithm'],
            covariance_type = param_list[param_i]['covariance_type'])


    # modeli oyret
    hmm_trainer.train(train_word_data, lengths_train)
    # oyredilmis modelin neticesini test datasinda yoxla ve listde saxla
    k_fold_result.append(hmm_trainer.get_score(test_word_data,lengths_test))

    hmm_trainer=None

if (sum(k_fold_result)>max_result):
    max_result=sum(k_fold_result)
    best_model=hmm_trainer

# en yaxsi modeli yaddasa yaz
with open('{}_grid_search_cv.pkl'.format(label), 'wb') as f:
pickle.dump(best_model, f)
print(best_model)
```

Listing 7: K-qatlama hesablaması və ən yaxşı modelin seçilməsi

4 Nəticə

Yekun olaraq *Təlim* və *Test* modelləri üçün log argmax üsulu ilə doğruluq payı tapılmışdır. Bu dəyər *Təlim* modeli üçün 96.24% *Test* modeli üçün 95.74% təşkil edir. Hər iki modelin göstəricilərinin yüksək olması və bir-birinə yaxın dəyərlər alması modelin sistemi doğru şəkildə ümumiləşdirə bildiyinin göstərgəsidir. Bütün kod tətbiqi github qovluğunda açıq olaraq verilmişdir. [1]

[1] https://github.com/anaramirli/Single-Word-ASR/blob/master/HMM_speech_recognizer.ipynb