Wireless Networks

Assignment-4

Submitted By

Arnav Singh
2021019

Part A

Preprocess the CSI For each subcarrier, there are 2 values real and imaginary indexed at even and odd indices, respectively. Read the CSI dataset files, remove the first 128 columns (CSI data from the legacy header), and remove the pilot and null subcarrier.

```
In [8]: def preprocess(dataset):
            #print(dataset)
            len_=dataset.shape[0]
            len_=int(abs(0.03*len_))
            # print(len_)
            dataset=(dataset).iloc[len_:-len_, 128:]
            label=pd.DataFrame((dataset)['headlabel'])
            (dataset)=(dataset).drop(columns=['timestamp', 'headlabel'], axis=1)
            (dataset).columns=range(len((dataset).columns))
            # remove null and pilot subcarriers
            delete_idxs= np.asarray([0,1,2,3,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,
            (dataset)=(dataset).drop((dataset).columns[delete_idxs], axis=1)
            (dataset).columns=range(len((dataset).columns))
            # print(amp_result.shape)
            return dataset,label
```

```
In [9]: #Fiber

        P26_fiber_4,label1=preprocess(P26_fiber_4)
        P27_fiber_4,label2=preprocess(P27_fiber_4)
        P28_fiber_4,label3=preprocess(P28_fiber_4)
        P29_fiber_4,label4=preprocess(P29_fiber_4)
        P30_fiber_4,label5=preprocess(P30_fiber_4)

        #Wood

        P26_wood_4,label6=preprocess(P26_wood_4)
        P27_wood_4,label7=preprocess(P27_wood_4)
        P28_wood_4,label8=preprocess(P28_wood_4)
        P29_wood_4,label9=preprocess(P29_wood_4)
        P30_wood_4,label10=preprocess(P30_wood_4)

        #Glass
        P26_glass_4,label11=preprocess(P26_glass_4)
        P27_glass_4,label12=preprocess(P27_glass_4)
        P28_glass_4,label13=preprocess(P28_glass_4)
        P29_glass_4,label14=preprocess(P29_glass_4)
        P30_glass_4,label15=preprocess(P30_glass_4)
```

Part B & C

Compute the amplitude for each subcarrier using the equation amplitudes=sqrt(imaginary2+real2) + Denoise the CSI samples with Hampel filter, 1-D wavelet transform filter, and savgol filter.

```python
def remove_nan(matrix_):
    temp=matrix_.copy()
    temp=temp.dropna()
    temp=temp.reset_index(drop=True)
    return  temp

def amplitude(df):
    amp=[]
    d=np.array(df)
    for j in range(len(d)):
        imaginary=[]
        real=[]
        amplitudes=[]
        # phases=[]
        for i in range(len(d[j])):
            if i%2==0:
                imaginary.append(d[j][i])
            else:
                real.append(d[j][i])
        for i in range(int(len(d[0])/2)):
            amplitudes.append(sqrt(imaginary[i]**2+real[i]**2))
            # phases.append(atan2(imaginary[i], real[i]))
        amp.append(amplitudes)
    amp=pd.DataFrame(amp)
    amp=amp.reset_index(drop=True)
    print("amplitude completed")
    return  amp

def hampel_filter(input_matrix, window_size, n_sigmas=3):
    n_rows, n_cols=input_matrix.shape
    new_matrix=np.zeros_like(input_matrix)
    std_dev=np.std(input_matrix)
    mad=np.median(np.abs(input_matrix - np.median(input_matrix)))
    k=std_dev / (mad)
    for col_idx in range(n_cols):
        for ti in range(n_rows):
            start_idx=max(0, ti - window_size)
            end_idx=min(n_rows, ti + window_size)
            x0=np.nanmedian(input_matrix[start_idx:end_idx, col_idx])
            s0=k * np.nanmedian(np.abs(input_matrix[start_idx:end_idx, col_idx] - x0))
            if np.abs(input_matrix[ti, col_idx] - x0) > n_sigmas * s0:
                new_matrix[ti, col_idx]=x0
            else:
                new_matrix[ti, col_idx]=input_matrix[ti, col_idx]
    print('hampel')
    return new_matrix

# 1-D wavelet transform
def denoise(df,wavelt):
    dwt=pd.DataFrame()
    # wevelt='bior1.1'
    for i in range(len(df.columns)):
        signal=df.iloc[:, i]
        coeff=pywt.wavedec(signal, wavelet=wavelt, mode="per")
        d=np.std(coeff[-1])
        sigma=4 * d
        uthresh=sigma * np.sqrt(2 * np.log(len(signal)))
        denoised_coeff=[coeff[0]]
        for c in coeff[1:]:
            denoised_coeff.append(pywt.threshold(c, value=uthresh, mode='soft'))
        denoised_signal=pywt.waverec(denoised_coeff, wavelet=wavelt, mode='per')
        dwt[i]=denoised_signal

    print("Denoising completed")
    return dwt



# savgol_filter
def smooth(df):
    from scipy.signal import savgol_filter
    window_length=5
    poly_order=2
    smoothed_data=savgol_filter(df, window_length, poly_order)
    smoothed_data= pd.DataFrame(smoothed_data)
    print('smooth')
    return smoothed_data
```

```python
 def ampdeno(dataset,label):
    amp_result= amplitude(P26_fiber_4)
    print(amp_result.shape)
    amp_result= hampel_filter(np.asarray(amp_result), 1000)
    amp_result= denoise(pd.DataFrame(amp_result), 'db4')
    amp_result= smooth(amp_result)
    amp_result= pd.concat([amp_result, label], axis=1)
    # amp_result= amp_result
    amp_result= remove_nan(amp_result)
    return amp_result
```

Part D

Create folders with filenames, for example, P 26 glass 4, and then create separate files for each head gesticulation and save them within the respective folder.

```
main_folder_path='C:/Users/lbjki/Downloads/gesticulation_data'
amp_list=[P26_fiber_4_amp, P26_wood_4_amp, P26_glass_4_amp, P27_fiber_4_amp, P27_wood_4_amp, P27_glass_4_amp, P28
participants=[f"P{i}" for i in range(26,31)]
obstacles=["fiber_4","wood_4","glass_4"]
idx=0
for pid in participants:
    for oid in obstacles:
        folder=pid+"_"+oid+"_amp"
        file_path=os.path.join(main_folder_path,folder)
        os.makedirs(file_path,exist_ok=True)
        gamp=amp_list[idx].groupby('headlabel')
        for i,j in gamp:
            i.strip()
            file=i+".csv"
            final_file_path=os.path.join(file_path,file)
            j.to_csv(final_file_path,index=False)
            print("saved "+file+" in "+folder)
        idx+=1
```

Part E & F

Preprocessing the data for training with few settings out strategy. Combine the CSI files of two materials for training and the CSI files of the remaining materials for testing. For example, combine “wood” and “fiber” for training and “glass” for testing + Prepare the input data for training.

```
import os
import pandas as pd
training_materials=['fiber_4','wood_4']
testing_materials=['glass_4']
training_all_files=[]
testing_all_files=[]
main_folder_path='C:/Users/lbjki/Downloads/gesticulation_data'
participants=[f"P{i}" for i in range(26,31)]
for pid in participants:
    for oid in training_materials:
        folder=pid+"_"+oid+"_amp"
        file_path=os.path.join(main_folder_path,folder)
        for headlabel_data in os.listdir(file_path):
            final_file_path=os.path.join(file_path,headlabel_data)
            td=pd.read_csv(final_file_path)
            training_all_files.append(td)

training_data=pd.concat(training_all_files,axis=0)
training_data=training_data.reset_index(drop=True)
Y=training_data['headlabel']
X=training_data.iloc[0:,0:-1]

#Select 100 best
selector=SelectKBest(f_classif,k=100)
csi_feature_100=selector.fit_transform(X,Y)

Y=pd.DataFrame(Y)
onehot_encoder=OneHotEncoder()
csi_headlabel=onehot_encoder.fit_transform(Y)

csi_feature_10X10=np.reshape(csi_feature_100,(csi_feature_100.shape[0],10,10))
csi_feature_10X10.shape

X_train=csi_feature_10X10
Y_train=csi_headlabel

for pid in participants:
    for oid in testing_materials:
        folder=pid+"_"+oid+"_amp"
        file_path=os.path.join(main_folder_path,folder)
        for headlabel_data in os.listdir(file_path):
            final_file_path=os.path.join(file_path,headlabel_data)
            td=pd.read_csv(final_file_path)
            testing_all_files.append(td)

testing_data=pd.concat(testing_all_files,axis=0)
testing_data=testing_data.reset_index(drop=True)
Y=testing_data['headlabel']
X=testing_data.iloc[0:,0:-1]

#Select 100 best
selector=SelectKBest(f_classif,k=100)
csi_feature_100=selector.fit_transform(X,Y)

Y=pd.DataFrame(Y)
onehot_encoder=OneHotEncoder()
csi_headlabel=onehot_encoder.fit_transform(Y)

csi_feature_10X10=np.reshape(csi_feature_100,(csi_feature_100.shape[0],10,10))
csi_feature_10X10.shape
X_test,X_val,Y_test,Y_val=train_test_split(csi_feature_10X10,csi_headlabel,test_size=0.2,random_state=42)
Y_train=Y_train.toarray()
Y_val=Y_val.toarray()
```

Part G

Train the model.

In [20]:

```
def csi_network_inc_res(input_sh, output_sh):
    nb_filters=64
    x_input=Input(input_sh)
    tcn1=TCN(
        nb_filters=nb_filters,
        nb_stacks=1,
        kernel_size=3,
        dilations=(1, 2, 4, 8, 16),
        use_skip_connections=True,
        use_layer_norm=True,
        kernel_initializer='glorot_uniform'
    )
    x=tcn1(x_input)
    print(f'TCN.receptive_field: {tcn1.receptive_field}.')
    x=tf.keras.layers.Flatten()(x)
    x=tf.keras.layers.Dense(output_sh, activation='relu', name='dense')(x)
    model=Model(inputs=x_input, outputs=x, name='csi_model')
    return model
```

In [21]:

```
model=csi_network_inc_res((10, 10), 7)
# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
model.fit(
    X_train,
    Y_train,
    batch_size=64,
    epochs=5,
    validation_data=[X_val, Y_val],
    callbacks=[
        tf.keras.callbacks.ModelCheckpoint('checkpointmodel.keras',
                                           monitor='val_accuracy',
                                           save_best_only=True,
                                           save_weights_only=False),
    ]
)
```

```
WARNING:tensorflow:From C:\Users\lbjki\PycharmProjects\pythonProject\.venv\Lib\site-packages\keras\src\backend\tensorflow\cor
e.py:204: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.
```

Part H

Few shot learning

```
WARNING:tensorflow:From C:\Users\lbjki\PycharmProjects\pythonProject\.venv\Lib\site-packages\keras\src\backend\tensorflow\cor
e.py:204: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

TCN.receptive_field: 48.
Epoch 1/5
963/963 ━━━━━━━━━━━━━━━━━━━━ 53s 39ms/step - accuracy: 0.1475 - loss: 10.1556 - val_accuracy: 0.1016 - val_loss: 8.0907
Epoch 2/5
963/963 ━━━━━━━━━━━━━━━━━━━━ 35s 36ms/step - accuracy: 0.1668 - loss: 10.0106 - val_accuracy: 0.1954 - val_loss: 5.4123
Epoch 3/5
963/963 ━━━━━━━━━━━━━━━━━━━━ 37s 38ms/step - accuracy: 0.2835 - loss: 5.9509 - val_accuracy: 0.1974 - val_loss: 5.4153
Epoch 4/5
963/963 ━━━━━━━━━━━━━━━━━━━━ 44s 42ms/step - accuracy: 0.2822 - loss: 5.9717 - val_accuracy: 0.1954 - val_loss: 5.3959
Epoch 5/5
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 43ms/step - accuracy: 0.2850 - loss: 5.9721 - val_accuracy: 0.1959 - val_loss: 5.4987
```

Out[21]: <keras.src.callbacks.history.History at 0x1f3147dbb30>

In [22]:

```
model.evaluate(X_test, Y_test, verbose=0)
```

Out[22]: [5.54888391494751, 0.19157542288303375]

In [25]:

```
from sklearn.model_selection import StratifiedShuffleSplit
import time
shot_sizes=[10,50,100,200,250,300,350,400,500,1000,2000,5000,10000]
results=[]

for numshots in shot_sizes:
    retrained_model=tf.keras.models.clone_model(model)
    retrained_model.set_weights(model.get_weights())
    retrained_model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    stratified_split=StratifiedShuffleSplit(test_size=len(X_test)-numshots,random_state=42)
    for train_idx,test_idx in stratified_split.split(X_test,Y_test):
        few_shot_indices=train_idx
        test_indices=test_idx

    X_FS=X_test[few_shot_indices]
    Y_FS=Y_test[few_shot_indices]
    X_test_new=X_test[test_indices]
    retrain_start_time=time.time()
    retrained_model.fit(X_FS,Y_FS,batch_size=64,epochs=5,verbose=0)
    retrain_time=time.time()-retrain_start_time
    test_start_time=time.time()
    test_loss,test_accuracy=retrained_model.evaluate(X_test,Y_test,verbose=0)
    test_time=time.time()-test_start_time
```

Part I

Plotting changes in accuracy with the number of shots

```
print(f"Retraining Time:{result['retrain_time']} seconds")
print(f"Testing Time:{result['test_time']} seconds")
print(f"Accuracy:{result['accuracy']}\n")
```