Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Proyecto

Clasificador de gestos de manos

Integrantes: Lukas Pavez
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudantes: Francisco Leiva C.
Gabriel Andrés Azócar Cárcamo
Giovanni Pais L.
Nicolas Cruz Brunet

Fecha de entrega: 12 de Julio de 2019
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

Se tiene como objetivo implementar un detector de gestos de manos, escogiendo uno o más clasificadores que se hayan estudiado durante el semestre.
El conjunto de datos a utilizar es el *EMG data for gestures Data Set* (link), y corresponde capturas de ocho señales, tomadas en el antebrazo usando electromiografía (EMG) a 36 personas, donde se realizaron 7 gestos, con pausas intermedias.

Para la implementación, se ignorarán los datos con label de gesto 7, ya que no todas las personas lo realizaron, también, de las 36 personas, se considerará a los datos correspondientes a las últimas 8 para el set de test, y el resto para entrenamiento (80 %) y validación (20 %), luego de dividir los datos se tomaran ventanas de datos y se les calculará un número de características y se les asignará un label, después se escogerán clasificadores para entrenar y se entrenará un detector de pausas (2 labels: gesto y pausa) y un clasificador de gestos (6 labels: del 1 al 6), antes de entrenar se utilizará un selector de características. Se realizarán pruebas variando 3 parámetros: el ancho de las ventanas, el salto entre las ventanas y el número de características seleccionadas.

Al finalizar el entrenamiento se evaluarán los resultados con el conjunto de test, probando cada clasificador por separado y luego ambos juntos (el detector de pausa seguido por el detector de gestos).

2. Desarrollo

2.1. Lectura del dataset

Se utilizo la librería *os* para listar todos los archivos, y se creo la función load_data(subjects, path, data, drop7=True) que recibe una lista de sujetos para cargar, el path de las carpetas de los datos, un DataFrame data, que se le va agregando los datos cargados, y un bool que indica si se ignora el label 7 o no. La función lista los archivos de cada carpeta y los carga con la función de pandas read_csv(). Luego de cargar los datos, borra los datos con label 7 (en caso de que drop7 sea True), y retorna una tupla que contiene los datos sin label y los labels. El código de la función es el siguiente:

```
def load_data(subjects, path, data, drop7=True):
    ndata = data.copy()
    for subject in subjects:
        new_path = path + subject + "/"
        for file in os.listdir(new_path):
            ndata = ndata.append(pd.read_csv(new_path + file, sep="\t"), sort=True)
    if drop7:
        ndata = ndata[ndata["class"] != 7]
    ndata = ndata.drop("time", 1)
    return ndata.drop("class", 1), ndata["class"]
```

2.2. Generación de ventanas y calculo de características

Para crear las ventanas se creo la función load_and_save_csv(width=200, step=200), que carga los datos, genera las ventanas, calcula las características y guarda los datasets generados en un csv, esto para ahorrarse el tiempo que demora el crear todas las ventanas en cada prueba, que puede tomar hasta 1 hora.
La función parte por separar a las personas en 2 conjuntos: test y train-valid, y utiliza la función load_data() para crear los DataFrames. Luego, para los 2 conjuntos, calcula las ventanas según el width y step dado, para esto utiliza la función windows(data, width, step) que recibe los datos y el ancho y paso de las ventanas, esta función parte con una lista vacía y tiene un loop que dura hasta que no pueda seguir tomando ventanas, dentro del loop toma el ancho de la ventana en datos y ve si el label del primer dato es igual al del último, esto para dejar solo las ventanas donde sus datos tengan el mismo label, los datos tomados corresponden a un DataFrame que se agrega a la lista creada al inicio, al terminar el loop se retorna la lista de ventanas. El código de la función es el siguiente:

```
def windows(data, width, step):
    start = 0
    windows_list = []
    while (start + width) < len(data):
        data_slice = data[start:start + width].reset_index()
        if data_slice["class"][0] == data_slice["class"][width - 1]:
            windows_list.append(data_slice)
        start += step
    return windows_list
```

Después de calcular la lista de ventanas para los conjuntos de train-valid y test, se hace un shuffle a las ventanas de train-valid y el conjunto se separa en train_windows y valid_windows (las ventanas calculadas del conjunto test se llaman test_windows). Luego de generar las ventanas se escogieron 8 características de las 13 propuestas en el paper referenciado en el enunciado del proyecto: promedio, varianza, mínimo, máximo, rango, media cuadrática, skew y kurtosis, las otras se descartaron por falta tiempo de implementación o resultados infinitos o nan al calcularlas en las ventanas.
Ya escogidas las características, se calculan para cada uno de los 8 canales de las ventanas calculadas con la función get_features(window) que recibe un DataFrame que representa a una ventana, la función crea un diccionario donde las llaves son el nombre de las característica calculada y un número al final que representa el canal de la ventana. Al terminar de calcular las 64 características (8 características por cada uno de los 8 canales), se agrega al diccionario el label de la ventana, que es cualquier dato de la columna *class* de la ventana, en este caso se toma el primero. Se finaliza retornando el diccionario. El código de la función es el siguiente:

```
def get_features(window):
    features = {}
    for i in range(1, 9):
        max_value = np.min(window["channel" + str(i)])
        min_value = np.max(window["channel" + str(i)])
        features["mean" + str(i)] = np.mean(window["channel" + str(i)])
        features["variance" + str(i)] = np.var(window["channel" + str(i)])
        features["min" + str(i)] = min_value
        features["max" + str(i)] = max_value
        features["range" + str(i)] = max_value - min_value
        features["RMS" + str(i)] = root_mean_square(window["channel" + str(i)])
        features["skew" + str(i)] = skew(window["channel" + str(i)])
        features["kurtosis" + str(i)] = kurtosis(window["channel" + str(i)])
    features["label"] = window["class"][0]
    return features
```

Los diccionarios generados se van agregando a un DataFrame que representa las características del conjunto al que se esta calculando, por lo que se generan 3 DataFrames: train_features, validation_features y test_features, la función termina guardando cada DataFrame en un csv, donde el nombre es el nombre del DataFrame, seguido por el ancho de la ventana y después el salto entre las ventanas, por ejemplo el csv de los datos de entrenamiento con ancho de ventana 200 y salto 400 es *train_features_200_400.csv*.

El código de la función load_and_save_csv() es el siguiente:

```
def load_and_save_csv(width=200, step=200):
    path = "EMG_data/"
    subjects = os.listdir(path)
    subjects.sort()

    test_len = len(subjects) - 8

    test_subjects = subjects[test_len:]
    train_validation_subjects = subjects[:test_len]
```

```
10
11  print("train and validation subjects: {}".format(train_validation_subjects))
12  print("test subjects: {}".format(test_subjects))
13
14  columns = ["channel1", "channel2", "channel3", "channel4", "channel5", "channel6", "channel7", "
     ↪ channel8", "class"]
15  empty_data = pd.DataFrame(columns=columns)
16
17  print(100 * '-')
18  print(44 * '-' + "loading data" + 44 * '-')
19  t = time()
20  x_train_valid, y_train_valid = load_data(train_validation_subjects, path, empty_data)
21  x_test, y_test = load_data(test_subjects, path, empty_data)
22
23  print("train_validation samples: {}".format(len(x_train_valid)))
24  print("test_samples: {}".format(len(x_test)))
25  print("time: {:.2f}".format(time() - t))
26  print(100 * '-')
27
28  print(40 * '-' + "calculating windows" + 41 * '-')
29  t = time()
30  train_valid_values = np.hstack((x_train_valid, y_train_valid.to_numpy().reshape(len(
     ↪ x_train_valid), 1)))
31  test_values = np.hstack((x_test, y_test.to_numpy().reshape(len(x_test), 1)))
32  train_valid = pd.DataFrame(columns=columns, data=train_valid_values)
33  test = pd.DataFrame(columns=columns, data=test_values)
34
35  train_valid_windows_list = windows(train_valid, width, step)
36  test_windows_list = windows(test, width, step)
37
38  if width == 100:  # error al calcular caracteristicas a ventanas de largo 100 al incluir label 0
39      train_valid_windows_list = [x for x in train_valid_windows_list if x["class"][0] != 0]
40      test_windows_list = [x for x in test_windows_list if x["class"][0] != 0]
41
42  np.random.shuffle(train_valid_windows_list)
43  train_len = int(len(train_valid_windows_list) * 0.8)
44  train_windows = train_valid_windows_list[:train_len]
45  valid_windows = train_valid_windows_list[train_len:]
46  test_windows = test_windows_list
47
48  print("train_windows: {}".format(len(train_windows)))
49  print("validation_windows: {}".format(len(valid_windows)))
50  print("test_windows: {}".format(len(test_windows)))
51  print("total_windows: {}".format(len(train_windows) + len(valid_windows) + len(test_windows)
     ↪ ))
52  print("time: {:.2f}".format(time() - t))
53  print(100 * '-')
54
55  print(40 * '-' + "calculating features" + 40 * '-')
56
57  features = ["mean", "variance", "min", "max", "range", "RMS", "skew", "kurtosis"]
58  print("features: {}".format(features))
```

```
59  cols = []
60  for i in range(1, 9):
61      for feature in features:
62          cols.append(feature + str(i))
63  cols.append("label")
64
65  print("calculating train features")
66  train_features = pd.DataFrame(columns=cols)
67  for window in train_windows:
68      window_features = get_features(window)
69      train_features = train_features.append(window_features, ignore_index=True)
70
71  print("calculating validation features")
72  validation_features = pd.DataFrame(columns=cols)
73  for window in valid_windows:
74      window_features = get_features(window)
75      validation_features = validation_features.append(window_features, ignore_index=True)
76
77  print("calculating test features")
78  test_features = pd.DataFrame(columns=cols)
79  for window in test_windows:
80      window_features = get_features(window)
81      test_features = test_features.append(window_features, ignore_index=True)
82
83  train_name = "train_features_" + str(width) + "_" + str(step) + ".csv"
84  validation_name = "validation_features_" + str(width) + "_" + str(step) + ".csv"
85  test_name = "test_features_" + str(width) + "_" + str(step) + ".csv"
86
87  train_features.to_csv(train_name, index=None, header=True)
88  validation_features.to_csv(validation_name, index=None, header=True)
89  test_features.to_csv(test_name, index=None, header=True)
```

2.3. Clasificadores

Para las pruebas de clasificación, se escogieron 2 clasificadores: svm y perceptrón multicapa (mlp), para ambos se utilizaron las librerías correspondientes de scikit-learn. Para el clasificador svm se creó la función svm_classifier(x_train, y_train, x_data, y_data), la función crea una grilla para encontrar el mejor parámetro C, buscando entre 0.1, 1 y 10, y como clasificador base se utiliza LinearSVC, que utiliza ova (one-vs-all) como método de clasificación, además, para cross-validation la grilla utiliza 5 folds. La función obtiene el mejor clasificador de la grilla, entrenando con los datos x_train e y_train, para luego mostrar la matriz de confusión, accuracy, tiempo y el mejor clasificador encontrado con los datos x_data e y_data. Finalmente se retorna el valor del accuracy y el clasificador encontrado. El código de la función es el siguiente:

```
def svm_classifier(x_train, y_train, x_data, y_data):
    svc = svm.LinearSVC()
    t1 = time()
    parameters = {'C': [0.1, 1, 10]}
    grid = GridSearchCV(svc, parameters, cv=5)
    grid.fit(x_train, y_train)

    classifier = grid.best_estimator_
    predictions = classifier.predict(x_data)
    confm = metrics.confusion_matrix(y_data, predictions)
    confm_diagonal = np.diag(confm)
    accuracy = confm_diagonal.sum() / confm.sum()
    t2 = time()
    print(100 * "-")
    print(confm)
    print("Accuracy: {:.4f}".format(accuracy))
    print("training time = {:.2f}".format(t2 - t1))
    print(100 * "-")
    print("classifier: {}".format(classifier))
    print(100 * "-")
    return [accuracy, classifier]
```

Para el clasificador mlp, se creó la función mlp_classifier(x_train, y_train, x_data, y_data), donde se crea el clasificador MLPClassifier de scikit-learn, se utilizaron los parámetros por default, donde se tiene una capa oculta con 100 neuronas y la función de activación es relu. Después de entrenarlo se muestra la matriz de confusión, accuracy, tiempo y el clasificador con los datos x_data e y_data. Finalmente se retorna el valor del accuracy y el clasificador encontrado. El código de la función es el siguiente:

```
def mlp_classifier(x_train, y_train, x_data, y_data):
    t1 = time()

    classifier = MLPClassifier()
    classifier.fit(x_train, y_train)
    predictions = classifier.predict(x_data)
    confm = metrics.confusion_matrix(y_data, predictions)
    confm_diagonal = np.diag(confm)
```

```
    accuracy = confm_diagonal.sum() / confm.sum()
    t2 = time()
    print(100 * "-")
    print(confm)
    print("Accuracy: {:.4f}".format(accuracy))
    print("training time = {:.2f}".format(t2 - t1))
    print(100 * "-")
    print("classifier: {}".format(classifier))
    print(100 * "-")
    return [accuracy, classifier]
```

2.4. Pruebas

Para ambos clasificadores escogidos, se realizaron pruebas variando el ancho y paso de las ventanas y el número de características seleccionadas k. Para las pruebas se crearon 2 funciones: gesture_classifier_test() para el clasificador de gestos y pause_detector_test() para el detector de pausas.

Para el clasificador de gestos, se utilizaron los siguientes anchos de ventana: 100, 200 y 400, se utilizaron los mismos valores para los saltos, por lo que se tienen 9 pruebas, además, se utilizaron los siguientes valores para k: 20, 30, 40, 50 y 64, por lo que en total se tienen 45 pruebas por clasificador. La función itera sobre las listas de anchos y pasos, y en cada iteración se cargan los csv de características correspondientes de entrenamiento y validación, descartando los datos con label 0 (las pausas), luego, para normalizar los datos, se crea un objeto StandardScaler() de scikit-learn y se hace fit con el conjunto de entrenamiento, y luego se hace transform a los conjuntos de entrenamiento y validación, esto normaliza ambos conjuntos según el conjunto de entrenamiento. Después se itera sobre la lista de k, donde para cada k se crea un selector SelectKBest de scikit-learn, que recibe una score_funcion y el valor de k. Para la función se le entrega *mutual_info_classif*, que mide la dependencia entre variables, y para k se le entrega el valor sobre el que se esta iterando, luego se repiten los mismos pasos que para el StandardScaler, se hace fit con el conjunto de entrenamiento y transform a ambos conjuntos, esto deja en ambos conjuntos las mejores k características encontradas por el selector, luego se llaman a las funciones mlp_classifier y svm_classifier y se guardan los valores de accuracy obtenidos en un diccionario de resultados por cada clasificador, svm_results y mlp_results, los diccionarios tienen llave k y el valor es otro diccionario, donde la llave es un string con la información de ancho y salto de las ventanas, separadas por una x, al final se entrega una tupla con 2 elementos, el primero una lista con los 2 diccionarios de resultados, y el segundo una lista con la cantidad de anchos y saltos probados. El código de la función es el siguiente:

```
def gesture_classifier_test():
    sizes_list = ["100", "200", "400"]
    print(100 * "-")
    print(100 * "-")
    print(100 * "-")
    print("gesture classifier")
    svm_results = {}
    mlp_results = {}
    ks = [20, 30, 40, 50, 64]
```

```
    for k in ks:
        svm_results[k] = {}
        mlp_results[k] = {}
    size_w = len(sizes_list)
    size_s = len(sizes_list)
    for w in sizes_list:
        for s in sizes_list:
            print(100 * '-')
            print("window width: {}".format(w))
            print("window step: {}".format(s))
            train_file = "train_features_" + w + "_" + s + ".csv"
            train_features = pd.read_csv(train_file, sep=",")
            train_features = train_features.loc[train_features['label'] != 0].to_numpy()

            validation_file = "validation_features_" + w + "_" + s + ".csv"
            validation_features = pd.read_csv(validation_file, sep=",")
            validation_features = validation_features.loc[validation_features['label'] != 0].to_numpy()

            print(100 * '-')
            print(36 * '-' + "Standardization of datasets" + 37 * '-')
            t = time()
            train_nc = train_features.shape[1] - 1
            x_train = train_features[:, :train_nc]
            y_train = train_features[:, train_nc]

            valid_nc = validation_features.shape[1] - 1
            x_valid = validation_features[:, :valid_nc]
            y_valid = validation_features[:, valid_nc]

            scaler = StandardScaler()
            scaler.fit(x_train)
            x_train = scaler.transform(x_train)
            x_valid = scaler.transform(x_valid)
            print("time: {:.2f}".format(time() - t))
            print(100 * '-')

            print(41 * '-' + "feature selection" + 42 * '-')

            for k in ks:
                print("{} features".format(k))
                selector = SelectKBest(mutual_info_classif, k=k)
                selector.fit(x_train, y_train)
                train = selector.transform(x_train)
                valid = selector.transform(x_valid)
                print(100 * "-")
                print("MLP classifier")
                mlp_results[k][w + 'x' + s] = int(round(mlp_classifier(train, y_train, valid, y_valid)[0] *
    ↪  100))
                print(100 * "-")
                print("SVM classifier")
                svm_results[k][w + 'x' + s] = int(round(svm_classifier(train, y_train, valid, y_valid)[0]
    ↪ * 100))
```

```
60            print(100 * "-")
61    return [svm_results, mlp_results], [size_w, size_s]
```

Luego de realizar las pruebas con el clasificador de gestos, los resultados de accuracy son los siguientes:

2.4.1. Resultados clasificador de gestos con clasificador MLP

Tabla 1: Resultados clasificador de gestos con clasificador MLP

	w/s	100	200	400
k=20	100	87 %	83 %	81 %
	200	89 %	89 %	83 %
	400	90 %	88 %	89 %
k=30	100	87 %	87 %	83 %
	200	93 %	90 %	88 %
	400	96 %	94 %	89 %
k=40	100	90 %	90 %	88 %
	200	96 %	94 %	91 %
	400	99 %	96 %	93 %
k=50	100	88 %	89 %	87 %
	200	96 %	92 %	91 %
	400	98 %	95 %	91 %
k=64	100	86 %	87 %	86 %
	200	94 %	90 %	88 %
	400	98 %	93 %	91 %

2.4.2. Resultados clasificador de gestos con clasificador SVM

Tabla 2: Resultados clasificador de gestos con clasificador SVM

	w/s	100	200	400
k=20	100	80 %	78 %	78 %
	200	80 %	82 %	78 %
	400	80 %	84 %	82 %
k=30	100	81 %	82 %	81 %
	200	84 %	83 %	81 %
	400	85 %	82 %	86 %
k=40	100	83 %	84 %	82 %
	200	86 %	84 %	84 %
	400	88 %	88 %	85 %
k=50	100	82 %	83 %	83 %
	200	86 %	84 %	83 %
	400	88 %	87 %	84 %
k=64	100	82 %	83 %	81 %
	200	86 %	84 %	82 %
	400	89 %	87 %	86 %

Para el detector de pausas se siguieron los mismos pasos que para el clasificador de gestos, se utilizaron anchos de ventana 200 y 400, y saltos 100, 200 y 400, se descarto ancho de ventana 100 por error en los cálculos de características. Se utilizaron valores de k 20, 30, 40, 50 y 64. Al cargar los datos de entrenamiento y validación se cambiaron los labels diferentes de 0 a 1, así todos los gestos quedan con el mismo label, y se tienen 2 clases: gesto y no gesto. Luego se siguieron los mismos pasos que en la parte anterior, se normalizaron los datos según el conjunto de entrenamiento con StandardScaler y para cada k se seleccionaron las mejores k características y se realizaron las pruebas con ambos clasificadores, y se retornan los diccionarios con resultados. El código de la función es el siguiente:

```
def pause_detector_test():
    print(100 * "-")
    print(100 * "-")
    print(100 * "-")
    print("pause vs gesture classifier")
    sizes_list = ["100", "200", "400"]
    svm_results = {}
    mlp_results = {}
    ks = [20, 30, 40, 50, 64]
    for k in ks:
        svm_results[k] = {}
        mlp_results[k] = {}
```

```
    size_w = len(sizes_list[1:])
    size_s = len(sizes_list)
    for w in sizes_list[1:]:
        for s in sizes_list:
            print(100 * '-')
            print("window width: {}".format(w))
            print("window step: {}".format(s))
            train_file = "train_features_" + w + "_" + s + ".csv"
            train_features = pd.read_csv(train_file, sep=",")
            train_features.loc[train_features['label'] != 0, 'label'] = 1
            train_features = train_features.to_numpy()

            validation_file = "validation_features_" + w + "_" + s + ".csv"
            validation_features = pd.read_csv(validation_file, sep=",")
            validation_features.loc[validation_features['label'] != 0, 'label'] = 1
            validation_features = validation_features.to_numpy()

            print(36 * '-' + "Standardization of datasets" + 37 * '-')
            t = time()
            train_nc = train_features.shape[1] - 1
            x_train = train_features[:, :train_nc]
            y_train = train_features[:, train_nc]

            valid_nc = validation_features.shape[1] - 1
            x_valid = validation_features[:, :valid_nc]
            y_valid = validation_features[:, valid_nc]

            scaler = StandardScaler()
            scaler.fit(x_train)
            x_train = scaler.transform(x_train)
            x_valid = scaler.transform(x_valid)
            print("time: {:.2f}".format(time() - t))
            print(100 * '-')

            print(41 * '-' + "feature selection" + 42 * '-')
            for k in ks:

                print("{} features".format(k))
                selector = SelectKBest(mutual_info_classif, k=k)
                selector.fit(x_train, y_train)
                train = selector.transform(x_train)
                valid = selector.transform(x_valid)
                print(100 * "-")
                print("MLP classifier")
                mlp_results[k][w + 'x' + s] = int(round(mlp_classifier(train, y_train, valid, y_valid)[0] *
    ↪  100))
                print(100 * "-")
                print("SVM classifier")
                svm_results[k][w + 'x' + s] = int(round(svm_classifier(train, y_train, valid, y_valid)[0]
    ↪ * 100))
                print(100 * "-")
    return [svm_results, mlp_results], [size_w, size_s]
```

Luego de realizar las pruebas con el detector de pausas, los resultados de accuracy son los siguientes:

2.4.3. Resultados detector de pausas con clasificador MLP

Tabla 3: Resultados detector de pausas con clasificador MLP

	w/s	100	200	400
k=20	200	71 %	71 %	69 %
	400	77 %	74 %	75 %
k=30	200	73 %	71 %	70 %
	400	81 %	77 %	76 %
k=40	200	73 %	71 %	71 %
	400	81 %	76 %	76 %
k=50	200	72 %	70 %	69 %
	400	81 %	78 %	74 %
k=64	200	71 %	70 %	68 %
	400	82 %	77 %	74 %

2.4.4. Resultados detector de pausas con clasificador SVM

Tabla 4: Resultados detector de pausas con clasificador SVM

	w/s	100	200	400
k=20	200	68 %	68 %	68 %
	400	72 %	72 %	72 %
k=30	200	68 %	69 %	70 %
	400	74 %	73 %	72 %
k=40	200	68 %	69 %	70 %
	400	74 %	74 %	73 %
k=50	200	69 %	70 %	70 %
	400	75 %	75 %	73 %
k=64	200	69 %	70 %	71 %
	400	75 %	76 %	73 %

3. Pruebas con conjunto de test

3.1. Cada clasificador por separado

Al ver los resultados con en conjunto de validación, para el clasificador de gestos se eligió ancho de ventana 400, salto 100 y k 40, que obtuvo un accuracy de 99 %. Para el detector de pausas se eligió ancho de ventana 400, salto 100 y k 64, que obtuvo un accuracy de 82 %. Para esto se creó la función classifiers_test(), que realiza las pruebas de ambos clasificadores con los mejores parámetros encontrados e imprime los resultados, además de hacer una figura con la matriz de confusión, y se guardada en la carpeta del programa, los resultados obtenidos son los siguientes:

- Accuracy clasificador de gestos: 82 %
- Accuracy detector de pausas: 72 %

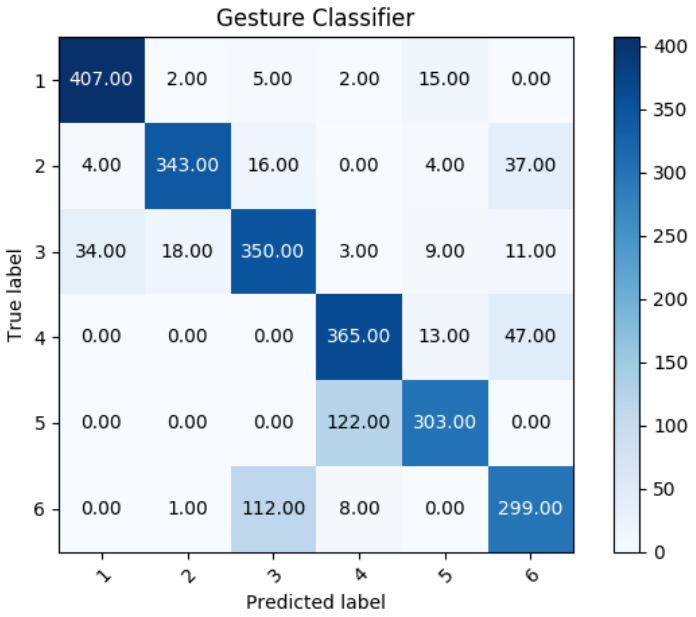

Figura 1: Matriz de confusión clasificador de gestos

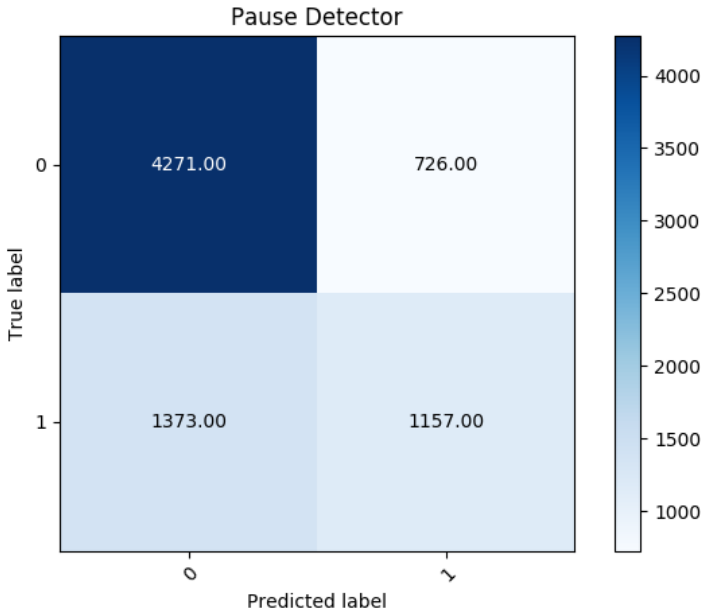

Figura 2: Matriz de confusión detector de pausas

El código de la función es el siguiente:

```
def classifiers_test():
    print("gesture classifier")
    w = '400'
    s = '100'
    k = 40

    print(100 * '-')
    print("window width: {}".format(w))
    print("window step: {}".format(s))
    train_file = "train_features_" + w + "_" + s + ".csv"
    train_features = pd.read_csv(train_file, sep=",")
    train_features = train_features.loc[train_features['label'] != 0].to_numpy()

    test_file = "test_features_" + w + "_" + s + ".csv"
    test_features = pd.read_csv(test_file, sep=",")
    test_features = test_features.loc[test_features['label'] != 0].to_numpy()

    print(100 * '-')
    print(36 * '-' + "Standardization of datasets" + 37 * '-')
    t = time()
    train_nc = train_features.shape[1] - 1
    x_train = train_features[:, :train_nc]
    y_train = train_features[:, train_nc]

    test_nc = test_features.shape[1] - 1
    x_test = test_features[:, :test_nc]
```

```
27  y_test = test_features[:, test_nc]
28
29  scaler = StandardScaler()
30  scaler.fit(x_train)
31  x_train = scaler.transform(x_train)
32  x_test = scaler.transform(x_test)
33  print("time: {:.2f}".format(time() - t))
34  print(100 * '-')
35
36  print(41 * '-' + "feature selection" + 42 * '-')
37
38  print("{} features".format(k))
39  selector = SelectKBest(mutual_info_classif, k=k)
40  selector.fit(x_train, y_train)
41  x_train = selector.transform(x_train)
42  x_test = selector.transform(x_test)
43  print(100 * "-")
44  print("MLP classifier")
45  acc, gesture_classifier = mlp_classifier(x_train, y_train, x_test, y_test)
46  acc = int(round(acc * 100))
47  print("accuracy: {}".format(acc))
48  confm = metrics.confusion_matrix(y_test, gesture_classifier.predict(x_test))
49  plot_confussion_matrix(confm, title="Gesture Classifier", classes=range(1, 7))
50  plt.savefig("gesture_classifier_test")
51  print(100 * "-")
52
53  print(100 * "-")
54  print(100 * "-")
55  print(100 * "-")
56  print("pause vs gesture classifier")
57
58  k = 64
59
60  print(100 * '-')
61  print("window width: {}".format(w))
62  print("window step: {}".format(s))
63  train_file = "train_features_" + w + "_" + s + ".csv"
64  train_features = pd.read_csv(train_file, sep=",")
65  train_features.loc[train_features['label'] != 0, 'label'] = 1
66  train_features = train_features.to_numpy()
67
68  test_file = "test_features_" + w + "_" + s + ".csv"
69  test_features = pd.read_csv(test_file, sep=",")
70  test_features.loc[test_features['label'] != 0, 'label'] = 1
71  test_features = test_features.to_numpy()
72
73  print(36 * '-' + "Standardization of datasets" + 37 * '-')
74  t = time()
75  train_nc = train_features.shape[1] - 1
76  x_train = train_features[:, :train_nc]
77  y_train = train_features[:, train_nc]
78
```

```
79    test_nc = test_features.shape[1] - 1
80    x_test = test_features[:, :test_nc]
81    y_test = test_features[:, test_nc]
82
83    scaler = StandardScaler()
84    scaler.fit(x_train)
85    x_train = scaler.transform(x_train)
86    x_test = scaler.transform(x_test)
87    print("time: {:.2f}".format(time() - t))
88    print(100 * '-')
89
90    print(41 * '-' + "feature selection" + 42 * '-')
91
92    print("{} features".format(k))
93    selector = SelectKBest(mutual_info_classif, k=k)
94    selector.fit(x_train, y_train)
95    x_train = selector.transform(x_train)
96    x_test = selector.transform(x_test)
97    print(100 * "-")
98    print("MLP classifier")
99    acc, pause_detector = mlp_classifier(x_train, y_train, x_test, y_test)
100   acc = int(round(acc * 100))
101   print("accuracy: {}".format(acc))
102   confm = metrics.confusion_matrix(y_test, pause_detector.predict(x_test))
103   plot_confussion_matrix(confm, title="Pause Detector", classes=range(2))
104   plt.savefig("pause_detector_test")
105   print(100 * "-")
```

3.2. Ambos clasificadores juntos

Para ambos clasificadores se utilizó ancho de ventana 400, con salto de 100, y k 40, utilizando el clasificador mlp, ya que con el clasificador de gestos y detector de pausas esos parámetros entregaron buenos resultados. Para esta sección se creó la función both_classifiers_test(), donde se parte por cargar los datos de entrenamiento y de test, y para ambos se hace una copia de la columna de labels (train_gesture_y, test_gesture_y), y después se cambian todos los labels distintos de 0 por 1 para entrenar al detector de pausas. Después de cargar los datos se normalizan según el set de entrenamiento con StandardScaler(), se eligen las k mejores características y se entrena el detector de pausas. Con las predicciones del detector de pausas se eligen las que se predijeron como 1 y se cambia el valor por su label real (que esta guardado en test_gesture_y), también se guardan los datos que corresponden a esa etiqueta para las predicciones finales en la lista x_pred. Luego se entrena el clasificador de gestos y se clasifican los datos x_pred, y se muestra la matriz de confusión en una figura, además de guardarla en la carpeta del programa. Se obtiene un accuracy de 54 %, la figura es la siguiente:

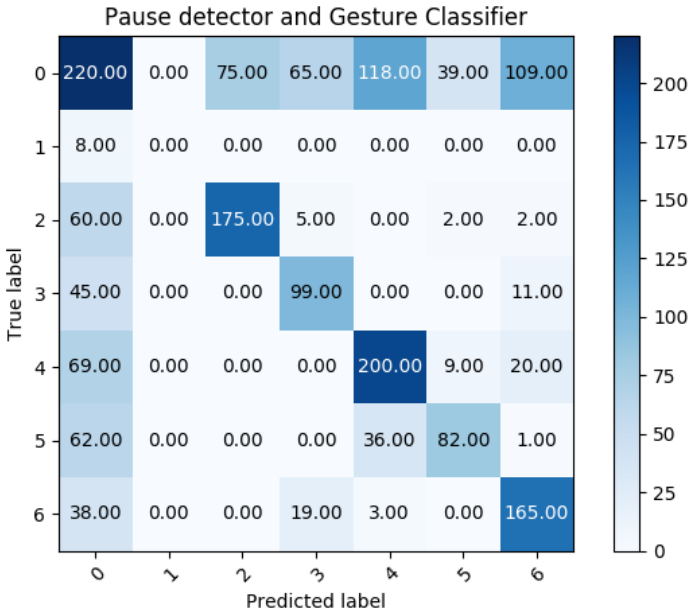

Figura 3: Matriz de confusión ambos clasificadores juntos

El código del programa es el siguiente:

```
def both_classifiers_test():
    np.set_printoptions(threshold=sys.maxsize)
    np.random.seed(42)
    w = '400'
    s = '100'
    k = 40
    print(100 * '-')
    print("window width: {}".format(w))
    print("window step: {}".format(s))

    train_file = "train_features_" + w + "_" + s + ".csv"
    train_features = pd.read_csv(train_file, sep=",")
    train_gesture_y = train_features['label'].copy().to_numpy()
    train_features.loc[train_features['label'] != 0, 'label'] = 1
    train_features = train_features.to_numpy()

    test_file = "test_features_" + w + "_" + s + ".csv"
    test_features = pd.read_csv(test_file, sep=",")
    test_gesture_y = test_features['label'].copy().to_numpy()
    test_features.loc[test_features['label'] != 0, 'label'] = 1
    test_features = test_features.to_numpy()

    print(36 * '-' + "Standardization of datasets" + 37 * '-')
    t = time()
    train_nc = train_features.shape[1] - 1
    x_train = train_features[:, :train_nc]
```

```
27    y_train = train_features[:, train_nc]
28
29    test_nc = test_features.shape[1] - 1
30    x_test = test_features[:, :test_nc]
31    y_test = test_features[:, test_nc]
32
33    scaler = StandardScaler()
34    scaler.fit(x_train)
35    x_train = scaler.transform(x_train)
36    x_test = scaler.transform(x_test)
37    print("time: {:.2f}".format(time() - t))
38    print(100 * '-')
39    print(41 * '-' + "feature selection" + 42 * '-')
40    print("{} features".format(k))
41    selector = SelectKBest(mutual_info_classif, k=k)
42    selector.fit(x_train, y_train)
43    train = selector.transform(x_train)
44    test = selector.transform(x_test)
45    print(100 * "-")
46    print("MLP classifier")
47    acc, pause_classifier = mlp_classifier(train, y_train, test, y_test)
48    pause_predictions = pause_classifier.predict(test)
49    real_y_pred = []
50    x_pred = []
51    for i in range(len(pause_predictions)):
52        if pause_predictions[i] == 1:
53            real_y_pred.append(test_gesture_y[i])
54            x_pred.append(test[i])
55
56    acc, gesture_classifier = mlp_classifier(train, train_gesture_y, x_pred, real_y_pred)
57    gesture_predictions = gesture_classifier.predict(x_pred)
58    print(np.unique(gesture_predictions))
59    confm = metrics.confusion_matrix(real_y_pred, gesture_predictions)
60    confm_diagonal = np.diag(confm)
61    accuracy = confm_diagonal.sum() / confm.sum()
62    print(100 * "-")
63    print(confm)
64    print("Accuracy: {:.4f}".format(accuracy))
65    print(100 * "-")
66    plot_confussion_matrix(confm, title="Pause detector and Gesture Classifier", classes=range(7))
67    plt.savefig("pause_detector_and_gesture_classifier_test")
```

4. Conclusiones

Analizando los resultados obtenidos, se puede ver que con cada clasificador por separado se obtuvieron en general buenos resultados, al utilizar el conjunto de validación se puede ver que se obtuvo un 99 % de accuracy con los mejores parámetros utilizando el clasificador perceptrón multicapa, esto puede significar que hubo un overfitting a los datos de validación, de igual manera se obtuvieron buenos resultados con los otros parámetros de tamaño de ventana y k, donde con el clasificador mlp los valores promedian alrededor de 90 %, y con el conjunto de test se llegó a un 82 %.

Con respecto al detector de pausas, los valores fueron más bajos, con los mejores parámetros encontrado se llegó a un 82 % en validación, y al probar con el conjunto de test bajó a un 72 %, pero al analizar la matriz de confusión en la Figura 2, se puede ver que aunque se clasifico bien la mayor parte de las pausas, mas de la mitad de los gestos se clasificaron como pausa, por lo que si bien se clasifica bien las pausas, los gestos también se tiende a clasificarlos como pausa.

Al utilizar ambos clasificadores juntos, se puede ver que el valor del accuracy disminuye bastante (54 %), esto se puede deber a que gran parte de los gestos se clasificaron como pausa en el primer clasificador, y luego se intento clasificar pausas que se predijeron como gesto, estas 2 cosas pueden ser la causa del mal resultado, donde el clasificar a que gesto corresponde una señal es casi como tirar una moneda.

Se puede concluir que se puede clasificar de manera aceptable las señales que corresponden solo a gestos, mientras que el detectar pausas es medianamente aceptable, y con ambos juntos no se puede asegurar buenos resultados.

Se logró el objetivo de la implementación del clasificador de gestos, aunque si se tuviera un mayor conocimiento de que características utilizar en trabajos con señales del tipo que se utilizaron (electromiografía) probablemente se hubieran tenido mejores resultados, por lo que se podría realizar una implementación con de un clasificador de gestos utilizando deep learning, donde no es necesario un análisis de características, ya que lo hace la red, por lo que puede ser un posible trabajo futuro.