CS-498-AML HW5

Chao Xu, ZHENBANG WU

TOTAL POINTS

100 / 100

QUESTION 1

1 Experiments Table **40 / 40**

✓ **- 0 pts** **Minimum 4 values for k and window length (combined)**

✓ **+ 10 pts** **Extra credit if more that 4 values provided**

✓ **- 0 pts** **Accuracy greater than 60%**

- 2 pts Maximum Accuracy between 55-60%

- 5 pts Maximum Accuracy between 50%-55%

- 10 pts Maximum Accuracy between 40%-50%

- 15 pts Maximum Accuracy between 30%-40%

- 20 pts Maximum Accuracy between 20%-30%

- 25 pts Maximum Accuracy less than 20%

QUESTION 2

2 Histograms **28 / 28**

✓ **+ 28 pts** **14 histograms (2pts per activity)**

QUESTION 3

3 Confusion Matrix **22 / 22**

✓ **+ 22 pts** **Correct - Diagonal Entries should be large | Possible confusion between "climb stairs-descend stairs", "eat meat-eat soup" (similar pairs)**

- 12 pts Seems incorrect/uninterpretable/confusing

QUESTION 4

4 Important Code Snippets **10 / 10**

✓ **+ 10 pts** **All correct**

- 3 pts Segmentation/Window length sample code not available

- 2 pts k-means code not available

- 3 pts conversion to histogram features code not available

- 2 pts classifier training code not available

QUESTION 5

5 Relevant Code **0 / 0**

✓ **- 0 pts** **Correct**

Zhenbang Wu / Chao Xu

Prof. D.A. Forsyth

CS498 Applied Machine Learning

16 October 2018

Homework 5

Page 1: Experiment Results

#	Fixed Length	%Overlap	#Clusters	K-means	Accuracy	#	Fixed Length	%Overlap	#Clusters	K-means	Accuracy
1	20	0.0	200	standard	0.784200385	28	30	0.3	300	standard	0.759152216
2	20	0.1	200	standard	0.782273603	29	30	0.4	300	standard	0.78805395
3	20	0.2	200	standard	0.789980732	30	30	0.5	300	standard	0.809248555
4	20	0.3	200	standard	0.780346821	31	30	0.0	400	standard	0.772639692
5	20	0.4	200	standard	0.784200385	32	30	0.1	400	standard	0.761078998
6	20	0.5	200	standard	0.770712909	33	30	0.2	400	standard	0.772639692
7	20	0.0	300	standard	0.80539499	34	30	0.3	400	standard	0.774566474
8	20	0.1	300	standard	0.786127168	35	30	0.4	400	standard	0.799614644
9	20	0.2	300	standard	0.791907514	36	30	0.5	400	standard	0.78805395
10	20	0.3	300	standard	0.799614644	37	40	0.0	200	standard	0.716763006

#	Fixed Length	%Overlap	#Clusters	K-means	Accuracy	#	Fixed Length	%Overlap	#Clusters	K-means	Accuracy
11	20	0.4	300	standard	0.764932563	38	40	0.1	200	standard	0.737957611
12	20	0.5	300	standard	0.751445087	39	40	0.2	200	standard	0.770712909
13	20	0.0	400	standard	0.764932563	40	40	0.3	200	standard	0.751445087
14	20	0.1	400	standard	0.78805395	41	40	0.4	200	standard	0.716763006
15	20	0.2	400	standard	0.770712909	42	40	0.5	200	standard	0.747591522
16	20	0.3	400	standard	0.764932563	43	40	0.0	300	standard	0.743737958
17	20	0.4	400	standard	0.757225434	44	40	0.1	300	standard	0.736030829
18	20	0.5	400	standard	0.759152216	45	40	0.2	300	standard	0.736030829
19	30	0.0	200	standard	0.799614644	46	40	0.3	300	standard	0.72061657
20	30	0.1	200	standard	0.818882466	47	40	0.4	300	standard	0.753371869
21	30	0.2	200	standard	0.780346821	48	40	0.5	300	standard	0.74566474
22	30	0.3	200	standard	0.764932563	49	40	0.0	400	standard	0.768786127
23	30	0.4	200	standard	0.786127168	50	40	0.1	400	standard	0.739884393
24	30	0.5	200	standard	0.776493256	51	40	0.2	400	standard	0.718689788
25	30	0.0	300	standard	0.786127168	52	40	0.3	400	standard	0.776493256
26	30	0.1	300	standard	0.791907514	53	40	0.4	400	standard	0.716763006
27	30	0.2	300	standard	0.789980732	54	40	0.5	400	standard	0.770712909

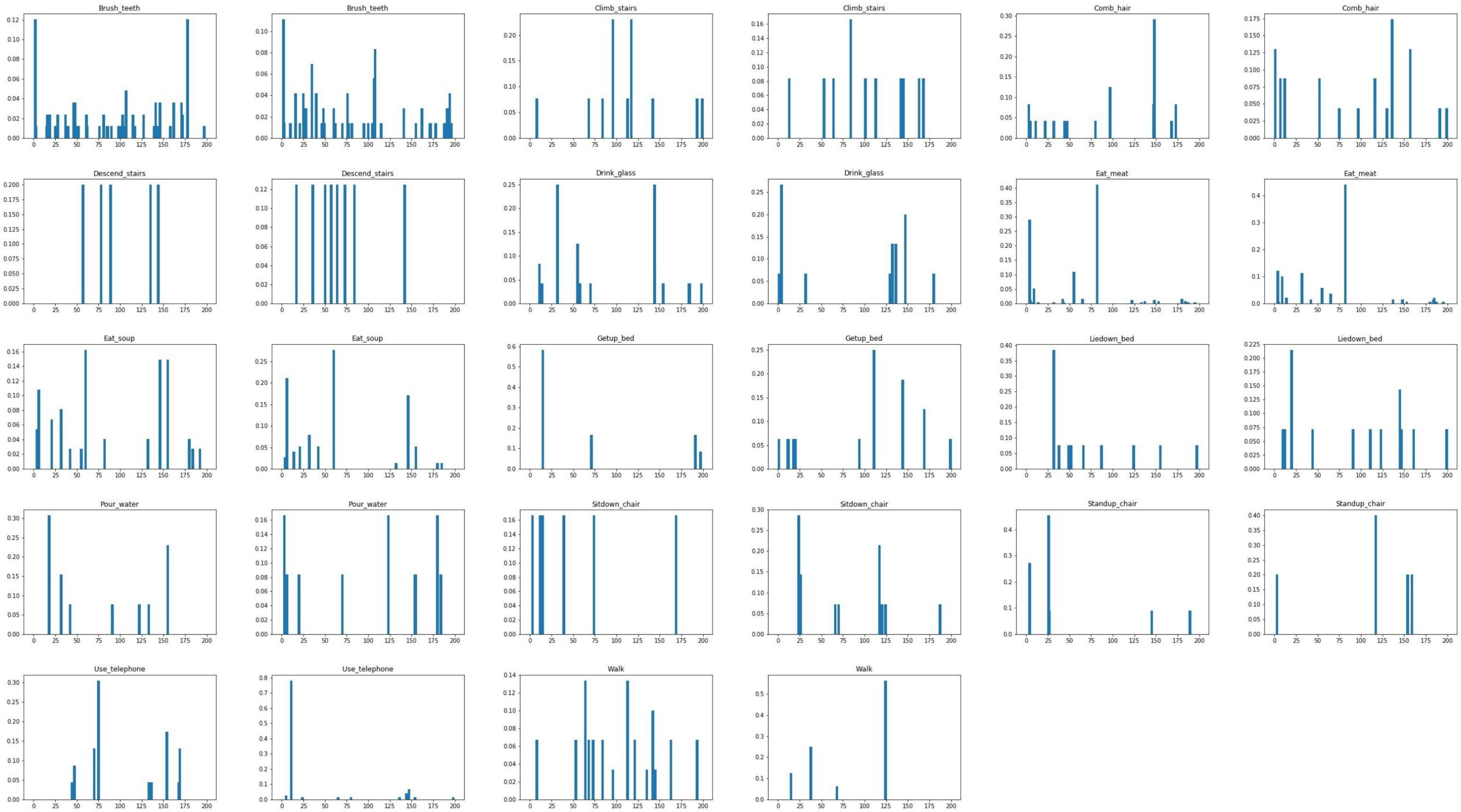
Remarks: i) fixing fixed length and number of clusters, we can see that changing the overlap percent will not significantly influence the accuracy (i.e. larger number of sliced pieces do not always bring us higher accuracy); ii) using a ‘moderate’ (i.e. neither too small or too large) fixed length will (to some extent) raise the accuracy; iii) varying the number of clusters among [200, 400] do not significantly change the accuracy (i.e. higher number of clusters do not always bring us higher accuracy).

1 Experiments Table **40 / 40**

- ✓ **- 0 pts Minimum 4 values for k and window length (combined)**
- ✓ **+ 10 pts Extra credit if more that 4 values provided**
- ✓ **- 0 pts Accuracy greater than 60%**
- **- 2 pts** Maximum Accuracy between 55-60%
- **- 5 pts** Maximum Accuracy between 50%-55%
- **- 10 pts** Maximum Accuracy between 40%-50%
- **- 15 pts** Maximum Accuracy between 30%-40%
- **- 20 pts** Maximum Accuracy between 20%-30%
- **- 25 pts** Maximum Accuracy less than 20%

Page 2: Plots

Histograms of the mean quantized vector for each activity (Fixed Length = 30, %Overlap = 0.1, #Clusters = 200)



Remarks: i) the histograms for different activities do look different to each other; ii) for the same activity, though they are performed by different people, they are somewhat similar to each other; iii) from the histograms, we can actually see that some activities are similar while some activities are different from each other (at least in terms of wrist-worn accelerometer; iv) we can also verify the ideas above from the class confusion matrix: for example, from the histograms we can see that “walk” and “climb stairs” are somehow similar, and the classifier do predict 4 “climb stairs” activities to be “walk”.

2 Histograms 28 / 28

✓ **+ 28 pts** **14 histograms (2pts per activity)**

Class confusion matrix

	Brush Teeth	Climb Stairs	Comb Hair	Descend Stairs	Drink Glass	Eat Meat	Eat Soup	Getup Bed	Lie Down Bed	Pour Water	Sit Down Chair	Standup Chair	Use Telephone	Walk
Brush Teeth	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Climb Stairs	0	14	0	4	0	0	0	0	1	0	0	0	0	0
Comb Hair	0	0	6	0	0	0	0	0	0	0	0	0	0	0
Descend Stairs	0	0	0	4	0	0	0	0	0	0	0	0	0	0
Drink Glass	0	2	0	0	19	0	0	1	0	0	0	0	1	0
Eat Meat	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Eat Soup	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Getup Bed	0	0	0	0	0	0	0	14	3	0	1	1	0	0
Lie Down Bed	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pour Water	1	0	1	0	1	0	1	0	1	20	0	1	0	0
Sit Down Chair	1	1	0	1	0	0	0	1	1	0	19	4	0	1
Standup Chair	0	0	0	0	0	0	0	5	0	0	0	14	0	1
Use Telephone	0	0	0	0	0	0	0	0	0	0	0	0	2	0
Walk	0	4	0	0	0	0	0	0	0	0	0	1	0	18

Remarks: i) the histograms for different activities do look different to each other; ii) for the same activity, though they are performed by different people, they are somewhat similar to each other; iii) from the histograms, we can actually see that some activities are similar while some activities are different from each other (at least in terms of wrist-worn accelerometer; iv) we can also verify the ideas above from the class confusion matrix: for example, from the histograms we can see that "walk" and "climb stairs" are somehow similar, and the classifier do predict 4 "climb stairs" activities to be "walk".

3 Confusion Matrix 22 / 22

✓ **+ 22 pts Correct - Diagonal Entries should be large | Possible confusion between "climb stairs-descend stairs", "eat meat-eat soup" (similar pairs)**

- 12 pts Seems incorrect/uninterpretable/confusing

Page 3: Screenshot of Code

Segmentation of the Vector

```
##########################################################################
#
# slice: function used to slice data into several non-overlapping
#        equal length pieces
# input:
#       @data: the data to be sliced
#       @fixed_len: length of each sliced piece
# output:
#       @segments: a 2-demension list that contains each file in its
#                  first dimension, and sliced pieces for each file
#                  in the corresponding second dimension
#
##########################################################################
def slice(data, fixed_len, overlap_percent):
  step = floor(fixed_len * (1 -overlap_percent))
  segments = []
  # for each folder
  for i in range(len(data)):
    # for each file
    for j in range(len(data[i])):
      # perform slicing
      for x in range(fixed_len, len(data[i][j]), step):
        tmp = []
        # convert the matrix into one dimension vector
        # by joining each row
        for n in range(fixed_len):
          tmp.extend(data[i][j][x-fixed_len:x][n])
        segments.append(tmp)
  # print("Obtain ", len(segments), " pieces from slicing")
  return segments
```

K-means

```
##########################################################################
#
# _kmeans_classify: helper function called by make_histograms,
#                   used to classify each pieces of data into
#                   given clusters
# input:
#       @data: the input data, whose structure is:
#             data[ADL][FileNumber]
#       @fixed_len: length of each sliced piece
#       @kmeans_model: kmeans model used to apply cluster
# output:
#       @classification: a 2-demension list that contains each
#                        file in its first dimension,
#                        and sliced pieces classification for
#                        each file in the corresponding
#                        second dimension
#
##########################################################################
def _kmeans_classify(data, fixed_len, kmeans_model):
  classification = []
  file_count = 0
  # for each folder
  for i in range(len(data)):
    # for each file
    for j in range(len(data[i])):
      file_count += 1
      file_classification = []
      # for each sliced piece
      for x in range(fixed_len, len(data[i][j]),fixed_len):
        # due to the constraint, sklearn kmeans,
        # we have to make sliced_piece into 2 dimensions
        sliced_piece = []
        tmp = []
        for n in range(fixed_len):
          tmp.extend(data[i][j][x-fixed_len:x][n])
        sliced_piece.append(tmp)
        # classify
        file_classification.extend(kmeans_model.predict(np.array(sliced_piece)))
      classification.append(file_classification)
  # print("Total number of test files is:", file_count)
  # print("Total number of prediction is:", len(classification))
  return classification
```

Generating the Histogram

```
###################################################################
#
# make_histograms: function used to convert given data into
#                  histograms (namely, extract equal length feature
#                  out of given data)
# input:
#       @data: the input data, whose structure is:
#             data[ADL][FileNumber]
#       @fixed_len: length of each sliced piece
#       @kmeans_model: kmeans model used to apply cluster
#       @clusterNum: total number of clusters
# output:
#       @histograms: a 2-demension list that contains each
#                    file in its first dimension,
#                    and corresponding features in its
#                    second dimension
#
###################################################################
def make_histograms(data, fixed_len, kmeans_model, clusterNum):
  # call _kmeans_classify to slice and classify
  prediction = _kmeans_classify(data, fixed_len, kmeans_model)
  histograms = []
  # for each file
  for fileNum in range(len(prediction)):
    tmp_histograms = [0] * clusterNum
    # count the number of each cluster and build histograms based on that
    for cluster in prediction[fileNum]:
      tmp_histograms[cluster] += 1
    histograms.append(tmp_histograms)
  return histograms
```

Classification

```
###################################################################
#
# classifier: function to classify based on vector quantization and k-means
# input:
#       @fixed_len: length of each sliced piece
#       @ncluster: total number of clusters
# output:
#       @confusionMatrix: the confusion matrix of the classifier
#       @accuracy: the accuracy on the held out test dataset
#
###################################################################
def classifier(fixed_len, ncluster, overlap_percent, plot = False):
    # VECTOR QUANTIZE & PREPARE FEATURES + LABELS
    # break signals into sample segments
    segments = slice(train, fixed_len, overlap_percent)
    # normal k-means (480 cluster centers)
    kmeans = KMeans(n_clusters = ncluster, random_state = 0).fit(np.array(segments))
    # making features using histogram of cluster centers
    trainFeatures = make_histograms(train, fixed_len, kmeans, ncluster)
    testFeatures = make_histograms(test, fixed_len, kmeans, ncluster)
    # normalize the histograms to get rid of the influence caused by the length of files
    trainFeatures_normalized = normalize(trainFeatures)
    testFeatures_normalized = normalize(testFeatures)
    # get the ground truth labels for both train & test dataset
    trainLabel = get_label(train, ADL_Names)
    testLabel = get_label(test, ADL_Names)

    if (plot == True):
      plot_activity_histogram(trainFeatures_normalized, trainLabel)

    # CLASSIFICATION
    # Random Forest Prediction
    RandomForestModel = RandomForestClassifier(n_estimators = 100)
    RandomForestModel.fit(trainFeatures_normalized, trainLabel)
    test_pred = RandomForestModel.predict(testFeatures_normalized)

    # confusion matrix
    confusionMatrix = confusion_matrix(test_pred, testLabel)
    # accuracy of prediction
    accuracy = accuracy_score(test_pred, testLabel)
    return (confusionMatrix, accuracy)
```

4 Important Code Snippets **10 / 10**

- ✓ **+ 10 pts** **All correct**
- **- 3 pts** Segmentation/Window length sample code not available
- **- 2 pts** k-means code not available
- **- 3 pts** conversion to histogram features code not available
- **- 2 pts** classifier training code not available

Page 3: Screenshot of Code

Segmentation of the Vector

```
##############################################################################
#
# slice: function used to slice data into several non-overlapping
#        equal length pieces
# input:
#       @data: the data to be sliced
#       @fixed_len: length of each sliced piece
# output:
#       @segments: a 2-demension list that contains each file in its
#              first dimension, and sliced pieces for each file
#              in the corresponding second dimension
#
##############################################################################
def slice(data, fixed_len, overlap_percent):
  step = floor(fixed_len * (1 -overlap_percent))
  segments = []
  # for each folder
  for i in range(len(data)):
    # for each file
    for j in range(len(data[i])):
      # perform slicing
      for x in range(fixed_len, len(data[i][j]), step):
        tmp = []
        # convert the matrix into one dimension vector
        # by joining each row
        for n in range(fixed_len):
          tmp.extend(data[i][j][x-fixed_len:x][n])
        segments.append(tmp)
  # print("Obtain ", len(segments), " pieces from slicing")
  return segments
```

K-means

```
##############################################################################
#
# _kmeans_classify: helper function called by make_histograms,
#                   used to classify each pieces of data into
#                   given clusters
# input:
#       @data: the input data, whose structure is:
#            data[ADL][FileNumber]
#       @fixed_len: length of each sliced piece
#       @kmeans_model: kmeans model used to apply cluster
# output:
#       @classification: a 2-demension list that contains each
#                        file in its first dimension,
#                        and sliced pieces classification for
#                        each file in the corresponding
#                        second dimension
#
##############################################################################
def _kmeans_classify(data, fixed_len, kmeans_model):
  classification = []
  file_count = 0
  # for each folder
  for i in range(len(data)):
    # for each file
    for j in range(len(data[i])):
      file_count += 1
      file_classification = []
      # for each sliced piece
      for x in range(fixed_len, len(data[i][j]),fixed_len):
        # due to the constraint, sklearn kmeans,
        # we have to make sliced_piece into 2 dimensions
        sliced_piece = []
        tmp = []
        for n in range(fixed_len):
          tmp.extend(data[i][j][x-fixed_len:x][n])
        sliced_piece.append(tmp)
        # classify
        file_classification.extend(kmeans_model.predict(np.array(sliced_piece)))
      classification.append(file_classification)
  # print("Total number of test files is:", file_count)
  # print("Total number of prediction is:", len(classification))
  return classification
```

Generating the Histogram

```
###########################################################################
#
# make_histograms: function used to convert given data into
#                  histograms (namely, extract equal length feature
#                  out of given data)
# input:
#       @data: the input data, whose structure is:
#              data[ADL][FileNumber]
#       @fixed_len: length of each sliced piece
#       @kmeans_model: kmeans model used to apply cluster
#       @clusterNum: total number of clusters
# output:
#       @histograms: a 2-demension list that contains each
#                    file in its first dimension,
#                    and corresponding features in its
#                    second dimension
#
###########################################################################
def make_histograms(data, fixed_len, kmeans_model, clusterNum):
  # call _kmeans_classify to slice and classify
  prediction = _kmeans_classify(data, fixed_len, kmeans_model)
  histograms = []
  # for each file
  for fileNum in range(len(prediction)):
    tmp_histograms = [0] * clusterNum
    # count the number of each cluster and build histograms based on that
    for cluster in prediction[fileNum]:
      tmp_histograms[cluster] += 1
    histograms.append(tmp_histograms)
  return histograms
```

Classification

```
###########################################################################
#
# classifier: function to classify based on vector quantization and k-means
# input:
#       @fixed_len: length of each sliced piece
#       @ncluster: total number of clusters
# output:
#       @confusionMatrix: the confusion matrix of the classifier
#       @accuracy: the accuracy on the held out test dataset
#
###########################################################################
def classifier(fixed_len, ncluster, overlap_percent, plot = False):
    # VECTOR QUANTIZE & PREPARE FEATURES + LABELS
    # break signals into sample segments
    segments = slice(train, fixed_len, overlap_percent)
    # normal k-means (480 cluster centers)
    kmeans = KMeans(n_clusters = ncluster, random_state = 0).fit(np.array(segments))
    # making features using histogram of cluster centers
    trainFeatures = make_histograms(train, fixed_len, kmeans, ncluster)
    testFeatures = make_histograms(test, fixed_len, kmeans, ncluster)
    # normalize the histograms to get rid of the influence caused by the length of files
    trainFeatures_normalized = normalize(trainFeatures)
    testFeatures_normalized = normalize(testFeatures)
    # get the ground truth labels for both train & test dataset
    trainLabel = get_label(train, ADL_Names)
    testLabel = get_label(test, ADL_Names)

    if (plot == True):
      plot_activity_histogram(trainFeatures_normalized, trainLabel)

    # CLASSIFICATION
    # Random Forest Prediction
    RandomForestModel = RandomForestClassifier(n_estimators = 100)
    RandomForestModel.fit(trainFeatures_normalized, trainLabel)
    test_pred = RandomForestModel.predict(testFeatures_normalized)

    # confusion matrix
    confusionMatrix = confusion_matrix(test_pred, testLabel)
    # accuracy of prediction
    accuracy = accuracy_score(test_pred, testLabel)
    return (confusionMatrix, accuracy)
```

5 Relevant Code 0 / 0

✓ **- 0 pts** Correct