İrem Zeynep Dündar
63931

COMP 448/548 – Medical Image Analysis HW2 Report

Part 1: Implementations

1. M = calculateCooccurrenceMatrix(grayImg, binNumber, di, dj)

This method takes gray image of input image, bin number and the distance (di, dj) values to calculate cooccurance matrix. Firstly the image is put into given number of bins by findBin method. For every row of the image, np.digitize method classifies the values according to

```python
def calculateCooccurrenceMatrix(binned_img, binNumber, di, dj):
    binned_img = findBin(binned_img, binNumber)
    max_i = binNumber

    co_occur = [[0 for _ in range(max_i+1)] for _ in range(max_i+1)]
    co_occur = np.array(co_occur)

    for x1 in range(len(binned_img)):
        for y1 in range(len(binned_img[0])):

            if di + x1 >= 0 and dj + y1 >=0 and di+x1 < len(binned_img) and dj+y1 < len(binned_img[0]):
                x2 = x1+di
                y2 = y1+dj
                co_occur[int(binned_img[x1,y1]), int(binned_img[x2,y2])] += 1
            else:
                # switch other column
                if dj+y1<0:
                    continue
                # break column loop, continue with row loop
                if di+x1<0:
                    break
                if di+x1>max_i:
                    break
                if dj+y1>max_i:
                    break

    return co_occur
```

```python
def findBin(gray_img, binno):
    temp = gray_img.copy()
    bins = np.arange(0, 1, 1/binno, dtype=float)
    for row_ind in range(len(temp)):
        an_array = temp[row_ind]
        bin_indices = np.digitize(an_array, bins)
        temp[row_ind] = bin_indices

    return temp
```

number of bins. It returns a row including the bin number of every corresponding value in that row. In every iteration that returned row is assigned to every row of image. After binned image is obtained, cooccurance matrix is calculated as given in lecture notes slide 5.

For cooccurance matrix every element pair in the of 2D image is traversed by given di, dj values. For every occurrence of the pairs the frequency of that pair is incremented on cooccurrence matrix, which was initialized as zero matrix in the beginning.

İrem Zeynep Dündar
63931

2. **accM = calculateAccumulatedCooccurrenceMatrix(grayImg, binNumber, d)**

CalculateCooccurrenceMatrix method is called for the given arrangement of (di,dj) values. **After** that all the obtained values summed and returned.

```
def calculateAccumulatedCooccurrenceMatrix(grayImg, binNumber, d):
    dlist = [(d, 0), (d, d), (0, d),(-d, d), (-d, 0), (-d, -d), (0, -d), (d, -d)]
    shape = calculateCooccurrenceMatrix(grayImg,binNumber,dlist[0][0],dlist[0][1]).shape
    sum_co_occur = np.zeros(shape, dtype=int)
    for i in range(1,len(dlist)):
        sum_co_occur = sum_co_occur + calculateCooccurrenceMatrix(grayImg,binNumber,dlist[i][0],dlist[i][1])
    return sum_co_occur
```

3. **feat = calculateCooccurrenceFeatures(accM)**

For calculating 6 features given in the homework handout, I created a class called Calculate_Features.

In calculateCooccurrenceFeatures method I created and returned an array for the values calculated for each feature. For the normalizing option, I added an additional input "normalized."

```
def calculateCooccurrenceFeatures(norm_co_occ, normalized):
    all_features = [angular_second_moment(norm_co_occ), max_prob(norm_co_occ), inv_diff_mom(norm_co_occ), contrast(norm_co_occ),
                    entropy(norm_co_occ), correlation(norm_co_occ)]
    if not normalized:
        return all_features
    else:
        norm = np.linalg.norm(all_features)
        return all_features / norm
```

PART 2: Classification

In order to make class-based classification for each balanced and imbalanced version of dataset test, train data directories are obtained for 3 classes. Moreover, corresponding labels are loaded as text file and each of them is read before the test and train begin.

By the chosen class and the type of balanced or imbalanced data, getFeature method is called to extract 2D feature array of corresponding data.

X_train and X_test 2D feature arrays are obtained by getFeatures method. getFeatures method takes folder path and dataset name as an input, so it decides start, end, size and filepath of the images in a given folder path.

İrem Zeynep Dündar
63931

By the values between start and end, sequence number of images are obtained normalized by processImg. processImg method returns a normalized image obtained by calculateAccumulatedCooccurrenceMatrix method. Afterwards the returned image from processImg is given to calculateCooccurrenceFeatures method and 2D feature array is obtained**.**

To point out, there is an imbalance problem which may cause bias an unhealthy result in while testing. Therefore, data and is split to three classes. Maximum size among 3 classes (class 2, size = 88) is selected and the classes having lower number of images are oversampled till they reach maximum class size. Corresponding y_labels are modified accordingly. As a result, a database2 is created by having balanced and imbalanced data.

Train and test method are created. Train method finds a best parameter where kernel may be specified by user as “rbf” or “linear”. After that a classifier is trained and returned.

The test method calculates train-test set accuracy for each class in addition to overall accuracy.

In test output 0 represents all data where 1, 2, 3 defines the classes. “i” or “b” added represents whether the data is imbalanced and balanced respectively.

Balancing made better results obviously as can be understood from the table below.

	Selected parameters	Training set accuracies				Test set accuracies			
		Class 1	Class 2	Class 3	**<u>Overall</u>**	Class 1	Class 2	Class 3	**<u>Overall</u>**
Linear kernel	C = 50000000	-	-	-	-	0.812	0.825	0.436	0.715
Linear kernel (balanced)	C = 50000	0.636	0.830	0.739	0.735	0.708	0.877	0.692	0.771
RBF kernel	C = 50000000 gamma = 1	0.750	0.864	0.684	0.790	0.542	0.895	0.487	0.667

İrem Zeynep Dündar
63931

RBF kernel (balanced)	C = 5000000 gamma = 10	0.977	0.943	0.966	0.962	0.792	0.860	0.410	0.715
Statistically different?									

PART 3: Grid-based approach

In part 3, in for loop every iimage is sliced into N pieces and saved into a folder. After that, this folders' path and "sub" flag is passed to getFeatures method. In getFeatures, again a dataset is created by the features array as in part 2, but this time return value is different for the same function.

The mean of this dataset is returned and this returned value assigned to row of another database initialized in getSlicedFeatures. This process is repeated for the all pictures in the database.

For mc_nemar statistics, two methods are written: mc_nemar and eval_mec_nemar. Creating a statistics for grid (sliced) and entire image (not sliced) versions of linear and rbf kernels, rbf and linear train data prediction results and test results are obtained for their 3 classes and whole set of classes. linear and rbf grid – entire image prediction results put into pairs respectively in order to be passed as an input to eval_mc_nemar method.

After that in a for loop that iterates all the pairs are passed into eval_mc_nemar. In eval_mc_nemar if mc_nemar method return is checked. If mc_nemar's result is 0, it means that there is not a statistical result to signify the difference between to methods else there is a statistical result to signify the difference. So, this information is printed by each case.

Another array is created

İrem Zeynep Dündar
63931

	Selected parameters	Training set accuracies				Test set accuracies			
		Class 1	Class 2	Class 3	Overall	Class 1	Class 2	Class 3	**<u>Overall</u>**
Linear kernel (grid-based)	C = 10	0.011	0.000	0.989	0.337	0.021	0.000	0.974	0.278
Linear kernel (entire image)	C =								
Statistically different?		**<u>y</u>**	**<u>y</u>**	**<u>y</u>**	**<u>y</u>**	**<u>y</u>**	**<u>y</u>**	**<u>n</u>**	**<u>y</u>**
RBF kernel (grid-based)	C = 1 gamma = 10	0.011	0.000	0.989	0.337	0.021	0.000	0.974	0.278
RBF kernel (entire image)	C = gamma =								
Statistically different?		**<u>n</u>**	**<u>y</u>**	**<u>y</u>**	**<u>y</u>**	**<u>n</u>**	**<u>y</u>**	**<u>y</u>**	**<u>y</u>**

İrem Zeynep Dündar
63931

Linear kernel and RBF kernel entire image is same with (entire image) table in part two, therefore they can be checked from table in part 2. It is obvious that results are not improved compared to part 2 by grid method.

Statistically significant and insignificant results are marked as y and n (yes, no).

The Outcomes of part 2 and part 3 evaluations and eval_mc_nemar results are added to appendix.

References

https://machinelearningmastery.com/mcnemars-test-for-machine-learning/

https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html

İrem Zeynep Dündar
63931

APPENDIX

Part 2

Selected kernel is: linear
Best parameter is {'C': 50000, 'kernel': 'linear'}
y_pred test len is: 144
y_test len is: 144
Class: 0b - linear
Accuracy train: 0.735
Accuracy test: 0.771
----
y_pred test len is: 48
y_test len is: 48
Class: 1b - linear
Accuracy train: 0.636
Accuracy test: 0.708
----
y_pred test len is: 57
y_test len is: 57
Class: 2b - linear
Accuracy train: 0.830
Accuracy test: 0.877
----
y_pred test len is: 39
y_test len is: 39
Class: 3b - linear
Accuracy train: 0.739
Accuracy test: 0.692

Selected kernel is: linear
Best parameter is {'C': 50000, 'kernel': 'linear'}
y_pred test len is: 144
y_test len is: 144
Class: 0b - linear
Accuracy train: 0.735
Accuracy test: 0.771
----
y_pred test len is: 48
y_test len is: 48
Class: 1b - linear
Accuracy train: 0.636
Accuracy test: 0.708
----
y_pred test len is: 57

y_test len is: 57
Class: 2b - linear
Accuracy train: 0.830
Accuracy test: 0.877
----
y_pred test len is: 39
y_test len is: 39
Class: 3b - linear
Accuracy train: 0.739
Accuracy test: 0.692

Selected kernel is: rbf
Best parameter is {'C': 5000000, 'gamma': 10, 'kernel': 'rbf'}
y_pred test len is: 144
y_test len is: 144
Class: 0b - rbf
Accuracy train: 0.962
Accuracy test: 0.715
----
y_pred test len is: 48
y_test len is: 48
Class: 1b - rbf
Accuracy train: 0.977
Accuracy test: 0.792
----
y_pred test len is: 57
y_test len is: 57
Class: 2b - rbf
Accuracy train: 0.943
Accuracy test: 0.860
----
y_pred test len is: 39
y_test len is: 39
Class: 3b - rbf
Accuracy train: 0.966
Accuracy test: 0.410

Selected kernel is: linear
Best parameter is {'C': 50000000, 'kernel': 'linear'}
y_pred test len is: 144
y_test len is: 144
Class: 0i' - linear
Accuracy train: 0.758
Accuracy test: 0.736
----
y_pred test len is: 48

y_test len is: 48
Class: 1i - linear
Accuracy train: 0.717
Accuracy test: 0.667
----
y_pred test len is: 57
y_test len is: 57
Class: 2i - linear
Accuracy train: 0.841
Accuracy test: 0.912
----
y_pred test len is: 39
y_test len is: 39
Class: 3i - linear
Accuracy train: 0.632
Accuracy test: 0.564

Part 3

Selected kernel is: linear_s
Best parameter is {'C': 1, 'gamma': 10, 'kernel': 'rbf'}
y_pred test len is: 144
y_test len is: 144
Class: 0b - linear_s
Accuracy train: 0.337
Accuracy test: 0.278
----
y_pred test len is: 48
y_test len is: 48
Class: 1b - linear_s
Accuracy train: 0.011
Accuracy test: 0.021
----
y_pred test len is: 57
y_test len is: 57
Class: 2b - linear_s
Accuracy train: 0.000
Accuracy test: 0.000
----
y_pred test len is: 39
y_test len is: 39
Class: 3b - linear_s
Accuracy train: 0.989

Accuracy test: 0.974

Selected kernel is: rbf_s
Best parameter is {'C': 1, 'gamma': 10, 'kernel': 'rbf'}
y_pred test len is: 144
y_test len is: 144
Class: 0b - rbf_s
Accuracy train: 0.337
Accuracy test: 0.278
----
y_pred test len is: 48
y_test len is: 48
Class: 1b - rbf_s
Accuracy train: 0.011
Accuracy test: 0.021
----
y_pred test len is: 57
y_test len is: 57
Class: 2b - rbf_s
Accuracy train: 0.000
Accuracy test: 0.000
----
y_pred test len is: 39
y_test len is: 39
Class: 3b - rbf_s
Accuracy train: 0.989
Accuracy test: 0.974

Mc Nemar Statistics Results

Class: tr1 - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: tr2 - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

İrem Zeynep Dündar
63931

Class: tr3 - linear
statistic=1.000, p-value=0.000
statistically insignificant results in train prediction.
statistic=1.000, p-value=0.000
statistically insignificant results in test prediction.

Class: tr - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts1 - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts2 - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts3 - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

İrem Zeynep Dündar
63931

Class: ts - linear
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: tr1 - rbf
statistic=0.000, p-value=0.000
statistically insignificant results in train prediction.
statistic=0.000, p-value=0.000
statistically insignificant results in test prediction.

Class: tr2 - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: tr3 - rbf
statistic=1.000, p-value=0.625
statistically significant results in train prediction.
statistic=1.000, p-value=0.625
statistically significant results in test prediction.

Class: tr - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts1 - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.

statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts2 - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts3 - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.

Class: ts - rbf
statistic=0.000, p-value=1.000
statistically significant results in train prediction.
statistic=0.000, p-value=1.000
statistically significant results in test prediction.