Programming assignment 03 – Learning

CII-2M3 Introduction to Artificial Intelligence

Even Semester 2021/2022

Report by:
Muhammad Daffa Khairi (1301203150)
Muhammad Fadli Ramadhan (1301203533)
Shinta Dewi Lestari Soleman (1301203567)

Problem Statement

We want to predict the ‘y’ value in the test data, but we don't have any kind of knowledge to predict what the value will be. Then, we need to analyze and make a model based on train data, so we can predict the ‘y’ value in the test data accurately. We used kNN because it is easier to implement and has better accuracy for continuous data.

The Description of the Program Design

- Read training/test data

  This program is processing the file input, which is “traintest.xlsx”.
- Model training

  We trained the model until get the best one
- Save the trained model

  To find the optimal k value.
- Model testing

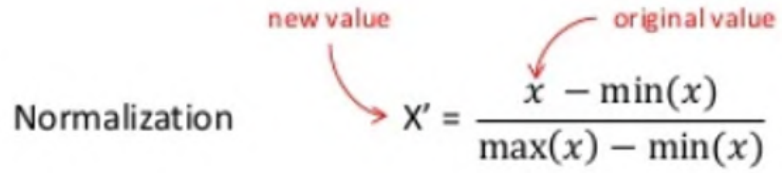
  To predict the truth of validation data, then it will be tested with the confusion matrix.
- Model evaluation

  To predict the data test.
- Save output to a file

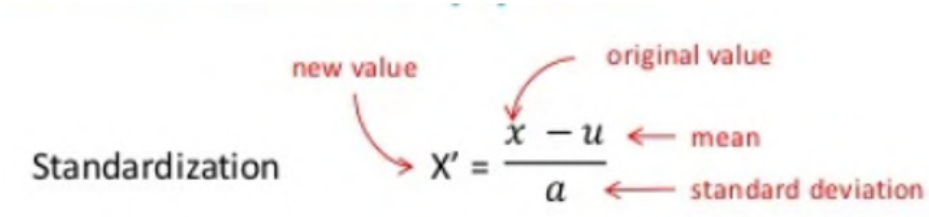
The Description of The Decoding Function

- read_excel(path, sheet_target) → to read the file "traintest.xlsx"

- reScale(data, method) → to check the data, are they within the same range or not by using this method of scaling,



or



- cross_data(train_data, total) → fold the data based on the cross validation based on each index.

- euclidean(X_train, X_test, i, j) → find the distance with the Euclidean method.
  Euclidean formula

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{ip} - x_{jp})^2}$$

- manhattanDist(X_train, X_test, i, j) → find the distance with the manhattan method.
  Manhattan formula

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}|$$

- minkowski(X_train, X_test, i, j, p = 3) → find the distance with the Minkowski method.

  Minkowski formula

$$d(i,j) = \sqrt[h]{\left|x_{i1} - x_{j1}\right|^h + \left|x_{i2} - x_{j2}\right|^h + \ldots + \left|x_{ip} - x_{jp}\right|^h}$$

- supremum(X_train, X_test, i, j) → find the distance with the Supremum method.

  Supremum formula

$$d(i,j) = \lim_{h\to\infty}\left(\sum_{f=1}^{p}\left|x_{if} - x_{jf}\right|^h\right)^{\frac{1}{h}} = \max_{f}^{p}\left|x_{if} - x_{jf}\right|$$

- closest_neighbor(train_data, validate_data,k, method, debug) → to find the closest neighbor.

  How it works:

  1. Choose the method.
  2. While it is still looping, sort the value. The results that are side by side are the neighbors.

- kNN(train_data, validate_data, k, method, debug, run_as) → process the kNN algorithm.

  How it works:

  1. Search for the y value in the neighborhood array.
  2. If the ‘run_as’ is ‘train’, then the prediction column is filled by the prediction value that previously appended.

     If the ‘run_as’ is ‘predict’, then the y column is filled by the prediction value that previously appended.

- confussion_matrix(trained) → confusion matrix to evaluate the model (k value).

  How it works:

  Make this model.

		Actual Class	
		+	-
Predicted Class	+	TP	FP
	-	FN	TN

Accuracy

$$\text{ACC} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Error Rate} = 1 - \text{ACC}$$

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Specificity

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

F1

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Miss Rate

$$Miss\ rate = \frac{FN}{FN + TP + 1}$$

Mathew

$$Matthew = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN) + 1}}$$

- training_model(data, method, debug, run_as, reScale_method, fold_total) → making a model from kNN

  How it works:

  1. Get the data from the reScale function.
  2. Start to train the data.
  3. Make a confusion matrix model from the trained data.
  4. Return the training conclusion value.

- k_optimal(model conclusion, split_index) → find the most optimal "k"

  How it works:

  1. Find the "k" value in every situation.
  2. Count the modus of "k".
  3. Inside the loop, if j is equal to the modus of k, then we count the accuracy, precision, specificity, recall, F1 score, miss rate, and mathew values.
  4. Determine the best index.

- model_optimal(model_conclusion, split_index) → find the most optimal model or “k” value based on accuracy.

  How it works :

  1. Find the locations that have maximum value.
  2. Find the most optimal index.

- test_prediction(data, test_data, k_value, method, debug, run_as, reScale_method, model_index,via) → predicting the ‘test’ database (to predict y value)

  How it works:

  1. Get the data from the reScale.
  2. Get the test data from the reScale function.
  3. Return predict value between data and test data.

- model_report(knn_method, optimal_k, reScale_method, fold_total, end_time, start_time, model_grade) → To show the model report.

The order

Determine the index for K-Fold Cross Validation

```
def cross_data(train_data, total):
    fold = train_data.shape[0]//total
    split_loc = []
    a = 0
    b = fold-1
    for i in range(total):
        split_loc.append([a,b])
        a += fold
        b += fold
    return split_loc, fold
```

Find the closest neighbor

```
def closest_neighbor(train_data, validate_data,k, method, debug):
    closest = []
    for i in range(len(validate_data)):
        temp = []
        for j in range(len(train_data)):
            if(method == 'eu'):
                temp.append([train_data['id'][j], euclidean(train_data, validate_data, i, j)])
            elif(method == 'manhat'):
                temp.append([train_data['id'][j], manhattanDist(train_data, validate_data, i, j)])
            elif(method == 'mink'):
                temp.append([train_data['id'][j], minkowski(train_data, validate_data, i, j)])
            elif(method == 'supre'):
                temp.append([train_data['id'][j], supremum(train_data, validate_data, i, j)])
        temp = sorted(temp, key=lambda l: l[1])[0:k]
        closest.append({'vali_id': validate_data['id'][i], 'closest': list(map(lambda c : c[0], [c for c in temp]))})
    if(debug):
        print(closest)
    return closest
```

kNN algorithm

```
def kNN(train_data, validate_data, k, method, debug, run_as):
    neighbor = closest_neighbor(train_data, validate_data,k, method, debug)
    prediction_arr = []
    for i in range(len(validate_data)):
        truth_value = []
        for j in neighbor[i]['closest']:
            truth_value.append(int(train_data.iloc[np.where(train_data['id'] == j)]['y']))
        prediction_arr.append(mode(truth_value))
    if(run_as == 'train'):
        validate_data['prediction'] = prediction_arr
    elif(run_as == 'predict'):
        validate_data['y'] = prediction_arr
        if(debug):
            print(validate_data)

    return validate_data
```

Confussion matrix to evaluate the model (k value)

```
def confussion_matrix(trained):
    tp, fp, fn, tn = 0,0,0,0
    for i in range(trained.shape[0]):
        if((trained['y'][i] == 1) and (trained['prediction'][i] == 1)): tp += 1
        elif((trained['y'][i] == 0) and (trained['prediction'][i] == 1)): fp += 1
        elif((trained['y'][i] == 1) and (trained['prediction'][i] == 0)): fn += 1
        elif((trained['y'][i] == 0) and (trained['prediction'][i] == 0)): tn += 1

    l = 10**(-10)
    acc =  ((tp+tn)/(tp+fp+tn+fn+l))*100
    prec = (tp/(tp+fp+l))*100
    spec = (tn/(tn+fp+l))*100
    recc = (tp/(tp+fn+l))*100
    f1 = 2*((prec*recc)/(prec+recc+l))
    miss = fn / (fn + tp+l)
    mathew = ((tp*tn) - (fp*fn)) / math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn)+l)
    acc_ball = (recc+spec)/2
    return {'Accuracy' : acc, 'Precision' : prec, 'Specificity' : spec, 'Recall' : recc, 'F1':f1, 'Miss_Rate':miss, 'Accuracy_Balance':acc_ball, 'Mathew_C
```

Training driver

```
def training_model(data, method, debug, run_as, reScale_method, fold_total):
    data = reScale(data, reScale_method)
    training_conclusion = []
    acc_array = []
    split_loc = cross_data(data, fold_total)
    for i in split_loc:
        acc_array = []
        validate_data = data[i[0]:i[1]]
        train_data = data
        for k in range(3, 20, 2):
            trained = kNN(train_data.reset_index(), validate_data.reset_index(), k, method, debug, run_as)
            cof_matrix = confussion_matrix(trained)
            acc_array.append([k, cof_matrix['Accuracy'],cof_matrix])
        training_conclusion.append(acc_array)
    return (training_conclusion, split_loc)
```

```
def k_optimal(model_conclusion, split_index):
    optimal_k = []
    for i in range(len(model_conclusion)):
        optimal_k.append(max(model_conclusion[i], key=lambda l: l[1])[0])

    modus = mode(optimal_k)
    acc =  0
    prec = 0
    spec = 0
    recc = 0
    f1 = 0
    miss = 0
    mathew = 0
    acc_ball = 0
    tp, fp, fn, tn = 0,0,0,0
    n = 0
    for i in model_conclusion:
        for j in i:
            if(j[0] == modus):
                acc += j[2]['Accuracy']
                prec += j[2]['Precision']
                spec += j[2]['Specificity']
                recc += j[2]['Recall']
                f1 += j[2]['F1 Score']
                miss += j[2]['Miss Rate']
                mathew += j[2]['Mathew Correlation']
                acc_ball += j[2]['Accuracy Balance']
                tp += j[2]['Matrix'][0]
                fp += j[2]['Matrix'][1]
                fn += j[2]['Matrix'][2]
                tn += j[2]['Matrix'][3]
                n += 1

    model_report = {'Accuracy' : acc/n, 'Precision' : prec/n, 'Specificity' : spec/n, 'Recall' : recc/n, 'F1 Score':f1/n, 'Miss Rate':miss/n, 'Accuracy Balance':acc_ball/n, 'Mathev
    best_index = [0,0]
    for x in range(len(model_conclusion)):
        for y in range(len(model_conclusion[x])):
            if((model_conclusion[x][y][0] == modus) and (model_conclusion[x][y][1] > model_conclusion[best_index[0]][best_index[1]][1])):
                best_index = [x,y]

    return modus, model_report, split_index[best_index[0]]
```

based on the model and its k

```
def model_optimal(model_conclusion, split_index):
    c = []
    for j in range(len(split_index)):
        a = max(model_conclusion[j], key=lambda l: l[1])[0]
        loc = [i for i, t in enumerate(model_conclusion[j]) if t[0]==a]
        c.append([j,loc[0]])

    print(c)
    high_index = c[0]
    for x in c:
        if(model_conclusion[x[0]][x[1]][1] >= model_conclusion[high_index[0]][high_index[1]][1]):
            high_index = x

    return (split_index[high_index[0]], model_conclusion[high_index[0]][high_index[1]][0], model_conclusion[high_index[0]][high_index[1]][2])
```

Predict the 'test' dataset

```
def test_prediction(data, test_data, k_value, method, debug, run_as, reScale_method, model_index ,via):
    data = reScale(data, reScale_method)
    if(via == 'model'):
        data = data.drop(data.index[model_index[0]:model_index[1]])
        data = data.reset_index()
    print(data.shape[0])
    test_data = reScale(test_data, reScale_method)
    prediction = kNN(data, test_data, k_value, method, debug, run_as)
    return prediction
```

OUTPUT PROGRAM.

The description of the data.

	id	x1	x2	x3	y
count	296.000000	296.000000	296.000000	296.000000	296.000000
mean	148.500000	52.462838	62.881757	4.111486	0.736486
std	85.592056	10.896367	3.233753	7.291816	0.441285
min	1.000000	30.000000	58.000000	0.000000	0.000000
25%	74.750000	44.000000	60.000000	0.000000	0.000000
50%	148.500000	52.000000	63.000000	1.000000	1.000000
75%	222.250000	61.000000	65.250000	5.000000	1.000000
max	296.000000	83.000000	69.000000	52.000000	1.000000

The detail of ‘y’ values in ‘train’ data.

```
1    218
0     78
Name: y, dtype: int64
```

Standardization scaling

	id	x1	x2	x3	y
0	1	0.691713	0.345804	-0.563849	1
1	2	0.141071	-0.891149	0.944691	0
2	3	1.150582	-0.272673	2.453232	0
3	4	-1.694403	-0.891149	-0.563849	1
4	5	-1.327308	1.891995	2.316092	0
...	...	...	...	...	...
291	292	0.599940	0.345804	-0.426709	1
292	293	1.150582	1.273518	-0.563849	1
293	294	0.049297	0.655042	1.081831	0
294	295	0.416392	0.345804	-0.426709	0
295	296	0.141071	-1.200388	0.396131	1

296 rows × 5 columns

Data describing after we do standardization scaling

Model training

The output of the model training is on the ‘Training Log.txt’ file that is attached.

Model Configuration

```
<Model configuration>
Algorithm : KNN
Distance Method : eu
Average K Value : 9
Scaling Method : std
K-Fold Cross-Validation : 9 folds
Model Optimizer : k_only
Best Fold Index : [64, 95]
Training Time : 30.34144377708435s
Model Quality :
<-> Accuracy Average : 73.8351254477905
<-> Precision Average : 76.99058677599106
<-> Specificity Average : 21.822991822717803
<-> Recall Average : 91.60555273046536
<-> F1 Score Average : 83.33460366715704
<-> Miss Rate Average : 0.08394447269089245
<-> Accuracy Balance Average : 56.71427227659158
<-> Mathew Correlation Average : 0.18038255998384498
<-> Confussion Matrix
    21.0        6.222222222222222
    1.8888888888888888  1.8888888888888888
```

Prediction Result

	id	x1	x2	x3	y
0	297	43	59	2	1
1	298	67	66	0	1
2	299	58	60	3	0
3	300	49	63	3	1
4	301	45	60	0	1
5	302	54	58	1	1
6	303	56	66	3	0
7	304	42	69	1	1
8	305	50	59	2	1
9	306	59	60	0	1

Team Member Contribution:

1. Muhammad Daffa Khairi (1301203150)
2. Muhammad Fadli Ramadhan (1301203533)
3. Shinta Dewi Lestari Soleman (1301203567)

REFERENCES

Nahzat, S. & Yağanoğlu, M. (2021). Diabetes Prediction Using Machine Learning Classification Algorithms. European Journal of Science and Technology, (24), 53-59.

Zhangjie Chen , Ya Wang, "Sleep monitoring using an infrared thermal array sensor," Proc. SPIE 10970, Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2019, 109701D (27 March 2019)

Puspita, R., & Widodo, A. (2021). Perbandingan Metode KNN, Decision Tree, dan Naïve Bayes Terhadap Analisis Sentimen Pengguna Layanan BPJS. *Jurnal Informatika Universitas Pamulang*, *5*(4), 646. https://doi.org/10.32493/informatika.v5i4.7622