PROGRAM ELM DAN LVQ DENGAN IRIS DATASET

LAPORAN
Disusun untuk Memenuhi Tugas Kelompok
Mata Kuliah Komputasi Lunak Semester VII
yang diampu oleh Rismiyati, B.Eng, M.Cs

DISUSUN OLEH:
JOHANADI SANTOSO (24060117120001)
LINGGAR MARETVA CENDANI (24060117120031)

PROGRAM STUDI STRATA 1 INFORMATIKA
DEPARTEMEN ILMU KOMPUTER/INFORMATIKA
FAKULTAS SAINS DAN MATEMATIKA
UNIVERSITAS DIPONEGORO
SEMARANG
2020

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Permasalahan

1.1.1. Bagaimana cara kerja program LVQ dengan Iris dataset?

1.1.2. Bagaimana cara kerja program ELM dengan Iris dataset?

1.2 Tujuan

1.1.1. Mampu menjelaskan program LVQ dengan Iris dataset?

1.1.2. Mampu menjelaskan program ELM dengan Iris dataset?

BAB II

PEMBAHASAN

2.1 LVQ

a. Fungsi train_lvq

```
import numpy as np

# train_lvq: trains an lvq system using the given training data an
d
# corresponding labels. Run the desired number of epochs using the
# given learning rate. Optional validation set to monitor performa
nce.
def train_lvq(data, labels, num_epochs, learning_rate, validation_
data=None, validation_labels=None):
    # Get unique class labels.
    num_dims = data.shape[1]
    labels = labels.astype(int)
    unique_labels = list(set(labels))

    # Initialize prototypes using class means.
    num_protos = len(unique_labels)
    prototypes = np.empty((num_protos, num_dims))
    proto_labels = []
    prototypes = np.array([[5.1, 3.5, 1.4, 0.2],[7.0, 3.2, 4.7, 1.
4],[6.3, 3.3, 6.0, 2.5]])
    proto_labels = np.array([1,2,3])

    # Loop through data set.
    learning_rate_new = learning_rate
    for epoch in range(0, num_epochs):
        print("Epoch : ", epoch+1)
        print("Learning Rate : ", learning_rate_new, "\n")
        count = 1;
        for fvec, lbl in zip(data, labels):
            #As Counting
            print("Data", count, ": ")
            count+=1;

            print("Bobot Awal : \n", prototypes, "\n")
            # Compute distance from each prototype to this point
            distances = list(np.sqrt(np.sum(np.subtract(fvec, p)**
2)) for p in prototypes)
            print("Jarak ke vektor bobot 1 : ", distances[0])
            print("Jarak ke vektor bobot 2 : ", distances[1])
```

```
            min_dist_index = distances.index(min(distances))

            # Determine winner prototype.
            winner = prototypes[min_dist_index]
            winner_label = proto_labels[min_dist_index]

            # Push or repel the prototype based on the label.
            if winner_label == lbl:
                sign = 1
            else:
                sign = -1

            # Update winner prototype
            prototypes[min_dist_index] = np.add(prototypes[min_dis
t_index], np.subtract(fvec, winner) * learning_rate_new * sign)
            print("Kelas pemenang : ",min_dist_index+1,"\n")
            print("Bobot baru ",min_dist_index+1," : ", prototypes
[min_dist_index], "\n")
        learning_rate_new = learning_rate_new -
 learning_rate_new*0.1
        print("\n\n")

    print("Bobot Akhir :")
    return (prototypes, proto_labels)
```

Fungsi train_lvq secara umum merupakan fungsi yang berguna untuk mentraining atau melatih dataset Iris. Proses training dataset Iris adalah dengan menggunakan Learning Vector Quantization atau LVQ. Terdapat beberapa masukan/ input untuk menjalankan fungsi train_lvq, yaitu dataset, label data atau kelas, jumlah epoch yang diinginkan, dan learning rate yang diinginkan.

Di dalam fungsi train_lvq ini, terdapat beberapa blok kode. Pertama ada blok kode # Get unique class labels. yang berguna untuk melakukan inisialisasi dengan mendapatkan nilai dimensi dari dataset Iris, membuat label bertipe integer, dan memasukkan dataset labels ke sebuah list dengan variabel *unique_labels*. Intinya blok kode ini untuk mendapatkan set berupa daftar kelas atau label dari data yang ada.

Kemudian ada blok kode # Initialize prototypes using class means. Yang berguna untuk melakukan inisialisasi list *prototype* atau *vector references*. Disini nilai dari *prototype* atau *vector references* awal ini didapat dari 3 data

pertama dari masing - masing kelas di dataset Iris, namun inisialisasinya dilakukan secara manual. Dilakukan input *3 vector references* awal karena dataset Iris terdiri dari 3 kelas, maka satu bobot *vector references* untuk masing - masing kelas.

Selanjutnya ada blok kode # Loop through data set. Pada blok kode ini dilakukan iterasi sejumlah epoch yang didapat dari masukkan fungsi atau parameter. Setiap iterasi epoch, akan ditampilkan urutan epoch dan learning rate yang digunakan. Didalam setiap iterasi epoch ini terdapat iterasi tiap data yang dibandingkan dengan vektor bobot.

#As Counting digunakan untuk menandai data keberapa yang sedang diproses pada iterasi, kerena pada tiap iterasi data yang dilakukan, ditampilkan data keberapa yang sedang diproses dan bobot awal sebelum diupdate.

Kemudian masih di dalam iterasi data, terdapat blok kode # Compute distance from each prototype to this point. Yang berguna untuk menghitung jarak data ke tiap vektor bobot dengan menggunakan *Euclidean distance*. Hasilnya disimpan di variabel *distances*. Index dari vector dengan jarak paling kecil kemudian disimpan di variabel *min_dist_index*.

Kemudian ada blok kode # Determine winner prototype. yang berguna untuk mendapatkan nilai dari bobot vector yang terpilih beserta labelnya yang kemudian dimasukkan ke variabel *winner* dan *winner_label*.

Selanjutnya ada blok kode # Push or repel the prototype based on the label. yang mengecek apakan label dari bobot vector sama dengan label dari data yang sedang diproses. Jika iya, maka mendapat sign 1, karena jika sama dengan target, maka bobot vector didekatkan, begitupula sebaliknya. Nantinya hasil ini digunakan untuk komputasi perbaruan bobot vektor.

Kemudian terdapat blok kode # Update winner prototype yang berguna untuk memperbarui nilai dari bobot vektor. Ditampilkan juga kelas pemenang dan bobot vektor setelah diperbarui.

Setelah melakukan iterasi pada setiap data yang ada, dilakukan update learning rate dengan rumus **learning rate baru = learning rate - learning rate * 0.1**.

Setelah semua epoch berhasil dijalankan, maka ditampilkan hasil bobot akhir.

b. Inisialisasi dataset dan labels

```
#Inisialisasi dataset dan labels
data = np.genfromtxt(r'iris.csv',delimiter=';')
labels = np.genfromtxt(r'iris_labels.csv',delimiter=';')
train_lvq(data, labels, num_epochs=3, learning_rate=0.1)
```

Kode blok ini digunakan untuk melakukan inisialisasi dataset yang ingin digunakan. Disini saya menggunakan dataset Iris dengan bentuk file dengan nama **iris.csv** dan labelnya disimpan di **iris_labels.csv**.

Kemudian setelah ini sialisasi dataset, dilakukan training dengan memanggil fungsi *train_lvq* dengan masukan data, labelnya, jumlah epoch, dan learning rate.

Hasil dari eksekusi kode diatas adalah sebagai berikut :

```
Epoch :  1
Learning Rate :  0.1

Data 1 :
Bobot Awal :
 [[5.1 3.5 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.3 3.3 6.  2.5]]

Jarak ke vektor bobot 1 :  0.5385164807134502
Jarak ke vektor bobot 2 :  4.096339829652808
Kelas pemenang :  1

Bobot baru  1  :  [5.08 3.45 1.4  0.2 ]

Data 2 :
Bobot Awal :
 [[5.08 3.45 1.4  0.2 ]
 [7.   3.2  4.7  1.4 ]
 [6.3  3.3  6.   2.5 ]]

Jarak ke vektor bobot 1 :  0.4657252408878006
Jarak ke vektor bobot 2 :  4.27668095606862
Kelas pemenang :  1

Bobot baru  1  :  [5.042 3.425 1.39  0.2  ]
```

.

.

```
Jarak ke vektor bobot 1 :  4.0772969180840235
Jarak ke vektor bobot 2 :  1.2951509407749828
Kelas pemenang :  2

Bobot baru  2  :  [6.03421574 2.80291496 3.82263125 1.04251937]



Bobot Akhir :
(array([[5.15391313, 3.55624398, 1.46469489, 0.24672142],
       [6.03421574, 2.80291496, 3.82263125, 1.04251937],
       [6.85405629, 3.03395299, 5.8879436 , 2.13242263]]), array([1, 2, 3]))
```

Pada contoh di atas, eksekusi dilakukan sebanyak 3 epoch, dengan masing - masing epoch memproses 58 data Iris. Bobot Akhir yang dihasilkan adalah [5.15391313, 3.55624398, 1.46469489, 0.24672142] untuk **kelas 1**, [6.03421574, 2.80291496, 3.82263125, 1.04251937] untuk **kelas 2**, dan [6.85405629, 3.03395299, 5.8879436 , 2.13242263] untuk **kelas 3**.

c. Testing LVQ

```
#Testing LVQ
import math as mt
def test_data(weight_lvq, data_test):
    print(weight_lvq)
    comparison_mat = []
    cl_A = 0
    cl_B = 0
    cl_C = 0
    for q in range(len(data_test)):
        for g in range(len(weight_lvq)):
            for w in range(len(weight_lvq[g]) - 1):
                cl_A += (data_test[q][w] - weight_lvq[0][w])
                cl_B += (data_test[q][w] - weight_lvq[1][w])
                cl_C += (data_test[q][w] - weight_lvq[2][w])
                w += 1
            g += 1
        plant_class = data_test[q][4]
        xw1 = mt.sqrt(cl_A ** 2)
        xw2 = mt.sqrt(cl_B ** 2)
        xw3 = mt.sqrt(cl_C ** 2)
        res = np.array([xw1, xw2, xw3])
        val = np.amin(res)
        if(val == xw1):
            tanaman_type = "IRIS SETOSA"
            winner = weight_lvq[0]
        elif (val == xw2):
            tanaman_type = "IRIS VESICOLOR"
            winner = weight_lvq[1]
        elif (val == xw3):
            tanaman_type = "IRIS VIRGINICA"
            winner = weight_lvq[2]
        print("Plant Type " + str(q + 1)+" = "+tanaman_type+".")
        result = [plant_class,winner[4]]
        comparison_mat.append(result)
        q++1
    right=0
```

```
    wrong=0
    for x in range(len(comparison_mat)):
            if(comparison_mat[x][0]==comparison_mat[x][1]):
                right+=1
            else:
                wrong+=1
    print("Models Accuracy = "+ str((float(right)/(right+wrong))*1
00) +"%")
```

Kode blok di atas adalah kode untuk melakukan testing terhadap hasil pembaruan bobot dengan algoritma LVQ yang telah dilakukan. Testing menggunakan dataset Iris juga.

Testing dilakukan dengan mendapatkan hasil dari data testing, dan kemudian dikomparasikan atau dibandingkan dengan label yang sebenarnya. Hasil keduanya dimasukkan dalam sebuah matrix bernama *comparison_mat*. Dibuat dua variabel yaitu *right* dan *wrong*. Setiap hasil benar, maka right bertambah satu, setiap hasil salah maka wrong bertambah satu. Hasilnya total right dibagi penjumlahan right dan wrong. Maka didapatkan akurasi dari LVQ. Pada contoh ini, didapatkan hasil sebagai berikut :

```
Plant Type 55 = IRIS VESICOLOR.
Plant Type 56 = IRIS VESICOLOR.
Plant Type 57 = IRIS VESICOLOR.
Plant Type 58 = IRIS VESICOLOR.
Plant Type 59 = IRIS VESICOLOR.
Plant Type 60 = IRIS VESICOLOR.
Plant Type 61 = IRIS VESICOLOR.
Models Accuracy = 40.98360655737705%
```

Didapatkan akurasi sebesar 40 %.

2.2 ELM

a. Fungsi elm_fit

```
import numpy as np
def elm_fit(x,target,h,W = None):
    if W is None:
       W = np.random.uniform(-1.,1.,(h,len(x[0])))

   Hinit = x@W.T
   print(Hinit)
   H = 1/(1+np.exp(-Hinit))

   Ht = H.T
   Hp = np.linalg.inv(Ht@H)@Ht
   beta = Hp@target

   y = H@beta
   mape = sum(abs(y-target)/target)*1000 / len(target)

   return W, beta, mape
```

Fungsi elm_fit merupakan fungsi extrme learning machine (ELM) dengan parameter inputan yaitu bobot (W), jumlah hidden layer (h), inputdata, dan target. Fungsi elm_fit berfungsi untuk melakukan training terhadap data training yang disediakan. fungsi elm diawali dengan inisialisasi nilai bobot secara random dengan rentang nilai antara 1 sampai -1.

Kemudian dilanjutkan dengan menghitung output dari hidden layer menggunakan fungsi aktivasi. Langkah pertama adalah menghitung keluaran hidden layer (Hinit), setelah nilai Hinit didapatkan kemudian dihitung menggunakan fungsi aktivasi.

Selanjutnya menghitung moore-penrose generalized inverse dari hasil keluaran hidden layer yang menggunakan fungsi aktivasi, dengan mencari nilai invers dari hasil perkalinjan Htranspose dikali H. kemudian hasil invers

tersebut dikalikan dengan Htranspose. Kemudian melakukan perhitungan nilai beta dengan cara mengalikan moore-penrose generalized dengan matriks target.

b. Fungsi elm_predict

```
def elm_predict(X,W,b,round_output = False):
    Hinit   =   X@W.T
    print(Hinit)
    H = 1/(1+np.exp(-Hinit))
    print(H)
    print("beta {}".format(beta))
    y = H@b
    print(y)
    if round_output:
        y = [int(round(x)) for x in y]

    return y
```

Fungsi elm_predict yaitu fungsi untuk memprediksi hasil extreme learning machine (ELM) dari input data test yang disediakan. Fungsi ini akan melakukan testing menggunakan fungsi aktivasi dan menghasilkan kelas prediksi.

c. Inisialisasi input data training dan target

```
inputdata =
np.array([[1,1,0,0],[0,0,0,1],[0,0,1,1],[1,0,0,0],[0,1,1,0]])
target = np.array([1,2,2,1,2])
```

Kode di atas adalah data input untuk elm_fit yang digunakan sebagai data training. Sedangkan target yaitu sasaran dalam fungsi elm_fit.

d. Memanggil fungsi elm_fit

```
bobot, beta, mape = elm_fit(inputdata,target,2,W = None)
```

Kode diatas akan menghasilkan keluaran seperti gambar di bawah, yaitu melakuan output/print nilai bobot, nilai beta dan memanggil nilai fungsi elm_fit dengan parameter inputdata sebagai data training, target, 2 lapis hidden layer dan bobot yang belum diinisialisasi.

```
[[ 0.38702188 -1.24838554]
 [ 0.46272998  0.54141981]
 [ 0.75922369  1.22293667]
 [ 0.11367446 -0.29012916]
 [ 0.56984114 -0.27673951]]

[[0.59556557 0.22297974]
 [0.6136616  0.63214264]
 [0.68118517 0.77257994]
 [0.52838805 0.42797225]
 [0.63872652 0.43125331]]

[[ 1.86918443 -0.41461076 -0.87252447  0.31965838  0.88716508]
 [-1.88759814  0.90027615  1.50052109 -0.04736944 -0.66598096]]

[1.38890251 1.53466499]
```

```
output_dat =
np.array([[0,1,0,0],[1,1,1,0],[0,1,1,1],[0,0,1,0]])
```

e. Inisialisasi data testing

Output_dat adalah input data untuk fungsi elm_predict sebagai data testing.

f. Memanggil fungsi elm_predict

```
predict = elm_predict(output_dat,bobot,beta,round_output =
True)
```

```
[[-0.18171641 -0.72908689]
 [ 0.38994839 -1.39187244]
 [ 1.11356321 -0.05394903]
 [ 0.81778198 -0.14275846]]
[[0.45469549 0.32539513]
 [0.59627027 0.199109  ]
 [0.75279281 0.48651601]
 [0.69376531 0.46437088]]
beta [2.68600294 0.13162471]
[1.26414348 1.62779138 2.08604123 1.92457836]
```

```
print(predict)
```

```
[1, 2, 2, 2]
```

Memanggil fungsi elm_predict dengan parameter output_dat, bobot hasil dari elm_fit, nilai beta yang dihasilkan elm_fit dan round_output = True. round_output bernilai True agar melakukan pembulatan pada output yang

dihasilkan. Keluarna akhir dari fungsi predict tersebut adalah [1, 2, 2, 1] yang berarti bahwa [0,1,0,0] berada di kelas 1, [1,1,1,0] di kelas 2, [0,1,1,1] di kelas 2, dan [0,0,1,0] di kelas 1.

g. Elm dengan mengguakan iris datasets

```
from sklearn import datasets
from sklearn.preprocessing import minmax_scale
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = iris.target
Y+ = 1

X_train, X_test, Y_train,
Y_test=train_test_split(X,Y,test_size=.3)
W,beta,mape = elm_fit(X_train,Y_train,5)
print("mape {}".format(mape))

output = elm_predict(X_test,W,beta,round_output = True)
accuracy = accuracy_score(output,Y_test)

print("output = {}".format(output))
print("label = {}".format(Y_test))
print("accuracy = {}".format(accuracy))
```

Pada baris pertama dilakukan pemanggilan library sklearn yang diperlukan untuk keperluan pemrograman. Baris kode selanjutnya yaitu malakukan import iris datasets, selanjutnya dilakukan normalisasi data menggunakan minmax_scale dan inisialisasi target. Kemudian iris datasets dibagi menjadi X_train, X_test, Y_train dan Y_test. Setelah itu adalah memanggil fungsi elm_fit dengan parameter input X_train, Y_train dan 5 lapis hidden layer untuk melakukan training. Setelah dilakukan training, fungsi elm_predict dipanggil untuk melakukan prediksi dan terakhir diukur akurasi prediksinya menggunakan accuracy_score yaitu melakukan pencocokan hasil prediksi dengan Y_test. Kode tersebut akan menghasilkan keluaran seperti gambar di

bawah ini, terlihat bahwa algoritma ini menghasilkan prediksi dengan akurasi sebesar 0.9777777777777777 %.

```
output= [2, 1, 2, 3, 3, 1, 3, 3, 2, 2, 2, 3, 1, 3, 3, 2, 3, 2, 1, 2, 3, 3, 2, 2, 3
label= [2 1 2 3 3 1 3 3 2 2 2 3 1 2 3 2 3 2 1 2 3 3 2 2 3 3 2 2 1 1 1 1 2 3 1 3 1
 1 1 1 3 1 3 3 1]
accuracy= 0.9777777777777777
```

BAB III

PENUTUP

3.1 Kesimpulan

LVQ atau Learning Vector Quantization merupakan jaringan syaraf yang menggunakan aturan winner takes all, dimana dilakukan perbaruan bobot vektor references setiap kali iterasi dengan sejumlah epoch dan learning rate yang ditentukan.

Pada program terdapat beberapa fungsi seperti train_lvq yang digunakan untuk mentraining data, dan Testing LVQ yang digunakan untuk melakukan pengujian data. Hasil pengujian pada program ini didapatkan akurasi sebesar 40.98360655737705 %. Hasil ini dapat dikatakan tidak terlalu baik, sehingga algoritma LVQ memerlukan penyesuaian pada epoch dan learning rate, serta jumlah data training yang digunakan.

Fungsi elm_fit merupakan fungsi extrme learning machine (ELM) dengan parameter inputan yaitu bobot (W), jumlah hidden layer (h), inputdata, dan target. Fungsi elm_fit berfungsi untuk melakukan training terhadap data training yang disediakan. Fungsi elm_predict yaitu fungsi untuk memprediksi hasil extreme learning machine (ELM) dari input data test yang disediakan

Berdasarkan hasil dari extreme learning machine (ELM) menunjukan akurasi yang tinggi, yaitu sebesar 0.9777777777777777 %.