CS5691 Pattern Recognition and Machine Learning

Jan – May 2021

Assignment – 3

Codes

Group 18:

BS17B033 – Shreya Nema

ME17B065 – Sahil Ansari

ME17B158 – Omkar Nath

Contents

1 Dataset 1(a) – Linearly Separable 2D Data

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix


train_data = pd.read_csv('data/Dataset 1A/train.csv')
train_data = train_data.to_numpy()

dev_data = pd.read_csv('data/Dataset 1A/dev.csv')
dev_data = dev_data.to_numpy()


def get_class_wise_data(data):
    obj = {}
    d = len(data[0]) - 1
    for point in data:
        curr_class = point[d]
        if obj.get(curr_class) is None:
            obj[curr_class] = [point]
        else:
            obj.get(curr_class).append(point)
    for class_ in obj:
        obj[class_] = np.array(obj.get(class_))
    return obj


def get_pairwise_class_data(data, first_class, second_class):
    first_class_data = data.get(first_class)
    second_class_data = data.get(second_class)
    concat_data = np.concatenate((first_class_data, second_class_data))
    np.random.shuffle(concat_data)
    _, d = concat_data.shape
    y_ = concat_data[:, d-1]
    x_ = concat_data[:, :d-1]
    return x_, y_


def perceptron_model(x_, y_, alpha_=0.0001):
    model_ = Perceptron(tol=1e-3, random_state=0, alpha=alpha_)
    model_.fit(x_, y_)
    return model_


def linear_svm_model(x_, y_, c=1.0):
    model_ = SVC(C=c)
    model_.fit(x_, y_)
    return model_


def mlp(x_, y_, hidden_layers=5):
    model_ = MLPClassifier(hidden_layer_sizes=hidden_layers)
    model_.fit(x_, y_)
    return model_
```

```
def predict(model_, x_):
    d = 0
    if len(x_.shape) == 1:
        d = len(x_)
    else:
        _, d = x_.shape
    x_ = x_.reshape(-1, d)
    y_ = model_.predict(x_)
    return y_


class_wise_data = get_class_wise_data(train_data)
val_class_wise_data = get_class_wise_data(dev_data)
x_data, y_data = get_pairwise_class_data(class_wise_data, 0, 2)
val_x_data, val_y_data = get_pairwise_class_data(val_class_wise_data, 0, 2)
# y_data = train_data[:, 2]
# x_data = train_data[:, :2]
# y_val = dev_data[:, 2]
# x_val = dev_data[:, :2]
#
# hidden_layer_sizes = [10, 15, 20]
# max_acc = 0
# best_size = 10
# for hidden_layer_size in hidden_layer_sizes:
#     model = mlp(x_data, y_data, hidden_layers=hidden_layer_size)
#     train_acc = model.score(x_data, y_data)
#     val_acc = model.score(x_val, y_val)
#     print(hidden_layer_size, 'Train Acc:', train_acc, 'Val Acc:',
val_acc)
#     if val_acc > max_acc:
#         max_acc = val_acc
#         best_size = hidden_layer_size
#
# print('Best hidden layers:', best_size)
# model = mlp(x_data, y_data, hidden_layers=best_size)
# train_pred = predict(model, x_data)
# val_pred = predict(model, x_val)
# print('Train Confusion Matrix:')
# print(confusion_matrix(y_data, train_pred))
# print('Val Confusion Matrix:')
# print(confusion_matrix(y_val, val_pred))

cs = [0.1, 0.5, 1.0]
best_acc = 0
best_alpha = 0.01
for alpha in cs:
    model = linear_svm_model(x_data, y_data, c=alpha)
    train_acc = model.score(x_data, y_data)
    val_acc = model.score(val_x_data, val_y_data)
    print('C:', alpha, 'Train acc:', train_acc, 'Val acc:', val_acc)
    if val_acc >= best_acc:
        best_acc = val_acc
        best_alpha = alpha

# model = mlp(x_data, y_data, hidden_layers=best_size)
# support_vectors = model.support_vectors_
print('Best C:', best_alpha)
model = linear_svm_model(x_data, y_data, c=best_alpha)
train_pred = predict(model, x_data)
```

```
val_pred = predict(model, val_x_data)
print('Train Confusion Matrix:')
print(confusion_matrix(y_data, train_pred))
print('Val Confusion Matrix:')
print(confusion_matrix(val_y_data, val_pred))

# fig, ax = plt.subplots()
# colors = ['r', 'g', 'b', 'y']
# legend_arr = []
# i = 0
# min_x = 999
# max_x = -999
# min_y = 999
# max_y = -999
# for point in x_data:
#     if point[0] < min_x:
#         min_x = point[0]
#     if point[0] > max_x:
#         max_x = point[0]
#     if point[1] < min_y:
#         min_y = point[1]
#     if point[1] > max_y:
#         max_y = point[1]
# # ax.scatter(x, y, color=colors[i])
#
# x_list = np.linspace(min_x, max_x, 100)
# y_list = np.linspace(min_y, max_y, 100)
#
# z = np.zeros((len(x_list), len(x_list)))
# X = []
# Y = []
# for i in range(len(x_list)):
#     temp_x = []
#     temp_y = []
#     for j in range(len(y_list)):
#         temp_x.append(x_list[i])
#         temp_y.append(y_list[j])
#         point = np.array([x_list[i], y_list[j]])
#         z[i][j] = predict(model, point)
#     X.append(temp_x)
#     Y.append(temp_y)
#
# ax.contourf(X, Y, z)
#
# x = []
# y = []
# for point in x_data:
#     x.append(point[0])
#     y.append(point[1])
#     ax.scatter(x, y, color='k')
#     i += 1

# ax.scatter(support_vectors[:, 0], support_vectors[:, 1], color='r')

plt.show()
```

2 Dataset 1(b) – Nonlinearly Separable 2D Data

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn.neural_network._base import ACTIVATIONS
from sklearn.metrics import confusion_matrix
import warnings
warnings.filterwarnings('ignore')


train_data = pd.read_csv('data/Dataset 1B/train.csv')
train_data = train_data.to_numpy()

dev_data = pd.read_csv('data/Dataset 1B/dev.csv')
dev_data = dev_data.to_numpy()


def mlp(x_, y_, hidden_layers=(5, 5), activation_='relu', max_iter=250):
    model_ = MLPClassifier(hidden_layer_sizes=hidden_layers,
random_state=0, max_iter=max_iter, activation=activation_)
    model_.fit(x_, y_)
    return model_


def svm_model(x_, y_, kernel='rbf', degree_=2, c_=1.0):
    if kernel == 'poly':
        model_ = svm.SVC(kernel=kernel, degree=degree_, random_state=0)
    else:
        model_ = svm.SVC(kernel=kernel, C=c_, random_state=0)
    model_.fit(x_, y_)
    return model_


def predict(model_, x_):
    d = 0
    if len(x_.shape) == 1:
        d = len(x_)
    else:
        _, d = x_.shape
    x_ = x_.reshape(-1, d)
    y_ = model_.predict(x_)
    return y_


_, d_ = train_data.shape
y_data = train_data[:, d_-1]
x_data = train_data[:, :d_-1]
y_val = dev_data[:, d_-1]
x_val = dev_data[:, :d_-1]

# hidden_layer_sizes = [(20, 20), (30, 30), (40, 40), (70, 70)]
# activations = ['relu', 'logistic', 'tanh']
# max_acc = 0
# best_size = (20, 20)
# best_activation = 'relu'
# for hidden_layer_size in hidden_layer_sizes:
```

```
#     for activation in activations:
#         model = mlp(x_data, y_data, hidden_layers=hidden_layer_size,
activation_=activation)
#         train_acc = model.score(x_data, y_data)
#         val_acc = model.score(x_val, y_val)
#         print(hidden_layer_size, activation, 'Train Acc:', train_acc,
'Val Acc:', val_acc)
#         if val_acc > max_acc:
#             max_acc = val_acc
#             best_size = hidden_layer_size
#             best_activation = activation
#
# print('Best size:', best_size, 'Best Activation:', best_activation)
# model = mlp(x_data, y_data, hidden_layers=best_size,
activation_=best_activation)
# train_pred = predict(model, x_data)
# val_pred = predict(model, x_val)
# print('Train Confusion Matrix:')
# print(confusion_matrix(y_data, train_pred))
# print('Val Confusion Matrix:')
# print(confusion_matrix(y_val, val_pred))

model = mlp(x_data, y_data, hidden_layers=(40, 40), activation_='relu',
max_iter=250)
weights = model.coefs_
intercepts = model.intercepts_
test_data = x_data[0]
node_num = 3
layer_num = 3
# for weight in weights[:2]:
#     test_data = np.matmul(weight.T, test_data)
#     print(weight.shape)

degrees = [2, 3, 4, 5, 6]
cs = [0.01, 0.1, 1.0, 10]
best_degree = 3
best_acc = 0
for degree in degrees:
    model = svm_model(x_data, y_data, kernel='poly', degree_=degree)
    train_acc = model.score(x_data, y_data)
    val_acc = model.score(x_val, y_val)
    print(degree, 'Train Acc:', train_acc, 'Val Acc:', val_acc)
    if val_acc >= best_acc:
        best_acc = val_acc
        best_degree = degree

print('Best degree:', best_degree)
model = svm_model(x_data, y_data, kernel='poly', degree_=best_degree)
train_pred = predict(model, x_data)
val_pred = predict(model, x_val)
print('Train Confusion Matrix:')
print(confusion_matrix(y_data, train_pred))
print('Val Confusion Matrix:')
print(confusion_matrix(y_val, val_pred))
# support_vectors = model.support_vectors_
# print(model.score(x_val, y_val))
#
# fig, ax = plt.subplots()
# fig = plt.figure()
# ax = fig.add_subplot(111, projection='3d')
# colors = ['r', 'g', 'b', 'y']
```

```
# legend_arr = []
# i = 0
# min_x = 999
# max_x = -999
# min_y = 999
# max_y = -999
# for point in x_data:
#     if point[0] < min_x:
#         min_x = point[0]
#     if point[0] > max_x:
#         max_x = point[0]
#     if point[1] < min_y:
#         min_y = point[1]
#     if point[1] > max_y:
#         max_y = point[1]
# # ax.scatter(x, y, color=colors[i])
#
# x_list = np.linspace(min_x, max_x, 100)
# y_list = np.linspace(min_y, max_y, 100)
#
# z = np.zeros((len(x_list), len(x_list)))
# X = []
# Y = []
# for i in range(len(x_list)):
#     temp_x = []
#     temp_y = []
#     for j in range(len(y_list)):
#         temp_x.append(x_list[i])
#         temp_y.append(y_list[j])
#         point = np.array([x_list[i], y_list[j]])
#         b = 0
#         for weight in weights[:layer_num]:
#             point = np.matmul(weight.T, point) + intercepts[b]
#             b += 1
#             ACTIVATIONS['relu'](point)
#         # z[i][j] = predict(model, point)
#         z[i][j] = point[node_num - 1]
#     X.append(temp_x)
#     Y.append(temp_y)
#
# ax.plot_surface(X, Y, z)

# x = []
# y = []
# for point in x_data:
#     x.append(point[0])
#     y.append(point[1])
#     ax.scatter(x, y, color='k')
#     i += 1

# ax.scatter(support_vectors[:, 0], support_vectors[:, 1], color='r')

plt.show()
```

3 Dataset 2 – Image Data Set for Static Pattern Classification

3.1 Code for Multilayer Feedforward Neural Network

```
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix



def mlp(x_, y_, hidden_layers=(5, 5), Solver='adam', Activation='relu',
Alpha=0.0001, LR='constant'):
    model_ = MLPClassifier(hidden_layer_sizes=hidden_layers, solver=Solver,
activation=Activation, alpha=Alpha, learning_rate=LR, random_state=0,
max_iter=10000)
    model_.fit(x_, y_)
    return model_


# get data for each class
classes = ['coast', 'forest', 'highway', 'mountain', 'tallbuilding']
data = None
val_data = None
cat_label = []
val_cat_label = []
for i, class_ in enumerate(classes):
    class_train_data = pd.read_csv('Dataset 2A/' + class_ + '/train.csv')
    df_class_data = class_train_data.drop(['image_names'], axis=1)
    class_data = df_class_data.to_numpy()

    m,n = class_data.shape
    cat_label += [class_]*m
    if data is None:
        data = class_data
    else:
        data = np.vstack((data, class_data))



    class_val_data = pd.read_csv('Dataset 2A/' + class_ + '/dev.csv')
    dfval_class_data = class_val_data.drop(['image_names'], axis=1)
    class_val_data = dfval_class_data.to_numpy()

    m,n = class_val_data.shape
    val_cat_label += [class_]*m
    if val_data is None:
        val_data = class_val_data
    else:
        val_data = np.vstack((val_data, class_val_data))



hidden_layer_sizes = [(100,100), (100,150), (150, 200)]
act_func = ['identity', 'logistic', 'tanh', 'relu']
Solver = ['lbfgs', 'sgd', 'adam']
alpha = [0.0001, 0.001, 0.01]
LR = ['constant', 'invscaling', 'adaptive']
max_acc = 0
```

```
best_size = (20, 20)
for hidden_layer_size in hidden_layer_sizes:
    for Sol in Solver:
        for Act in act_func:
            for alp in alpha:
                for lr in LR:
                    model = mlp(data, cat_label, 
hidden_layers=hidden_layer_size, Solver=Sol, Activation=Act, Alpha=alp, 
LR=lr)
                    train_acc = model.score(data, cat_label)
                    val_acc = model.score(val_data, val_cat_label)
                    print(hidden_layer_size, Sol, Act, alp, lr, 'Train_Acc: 
', train_acc, 'val_acc: ', val_acc)
                    if val_acc > max_acc:
                        max_acc = val_acc
                        best_model = [hidden_layer_size, Sol, Act, alp, lr]
print(best_model, 'max_acc: ', max_acc)


model = mlp(data, cat_label, hidden_layers=best_model[0], 
Solver=best_model[1], Activation=best_model[2], Alpha=best_model[3], 
LR=best_model[4])
train_acc = model.score(data, cat_label)
val_acc = model.score(val_data, val_cat_label)
print('Train_Acc: ', train_acc, 'val_acc: ', val_acc)
train_pred = model.predict(data)
val_pred = model.predict(val_data)
train_CM = confusion_matrix(cat_label, train_pred, labels=classes)
print('train confusion matrix:\n', train_CM)
val_CM = confusion_matrix(val_cat_label, val_pred, labels=classes)
print('validation confusion matrix:\n', val_CM)
```

3.2 Non-linear SVM using Gaussian Kernel

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix


def svm_model(x_, y_, kernel='rbf', C=1.0, gamma='scale', degree=2):
    if kernel == 'polynomial':
        model_ = svm.SVC(kernel=kernel, degree=degree, C=C, gamma=gamma,
random_state=0)
    else:
        model_ = svm.SVC(kernel=kernel, C=C, gamma=gamma, random_state=0)
    model_.fit(x_, y_)
    return model_


# get data for each class
classes = ['coast', 'forest', 'highway', 'mountain', 'tallbuilding']
data = None
val_data = None
cat_label = []
val_cat_label = []
for i, class_ in enumerate(classes):
    class_train_data = pd.read_csv('Dataset 2A/' + class_ + '/train.csv')
    df_class_data = class_train_data.drop(['image_names'], axis=1)
    class_data = df_class_data.to_numpy()

    m,n = class_data.shape
    cat_label += [class_]*m
    if data is None:
        data = class_data
    else:
        data = np.vstack((data, class_data))



    class_val_data = pd.read_csv('Dataset 2A/' + class_ + '/dev.csv')
    dfval_class_data = class_val_data.drop(['image_names'], axis=1)
    class_val_data = dfval_class_data.to_numpy()

    m,n = class_val_data.shape
    val_cat_label += [class_]*m
    if val_data is None:
        val_data = class_val_data
    else:
        val_data = np.vstack((val_data, class_val_data))


# fit scaler on training data
norm = MinMaxScaler().fit(data)
# transform training data
X_train_norm = norm.transform(data)

# transform testing dataabs
X_val_norm = norm.transform(val_data)
```

```
scale = StandardScaler().fit(data)
X_train_scaled = scale.transform(data)
X_val_scaled = scale.transform(val_data)



C = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
Gamma = ['scale', 'auto']
max_acc = 0
best_model = []
for c in C:
    for gamma in Gamma:
        model = svm_model(X_train_scaled, cat_label, kernel='rbf', C=c,
gamma=gamma)
        support_vectors = model.support_vectors_
        train_acc = model.score(X_train_scaled, cat_label)
        val_acc = model.score(X_val_scaled, val_cat_label)
        print('C: ', c, 'gamma: ', gamma, 'train_acc: ', train_acc,
'val_acc: ', val_acc)
        if val_acc > max_acc:
            max_acc = val_acc
            best_model = [c,gamma]

print(best_model, 'max_acc: ', max_acc)

model = svm_model(X_train_scaled, cat_label, kernel='rbf', C=best_model[0],
gamma=best_model[1])
support_vectors = model.support_vectors_
train_pred = model.predict(X_train_scaled)
val_pred = model.predict(X_val_scaled)
train_acc = model.score(X_train_scaled, cat_label)
val_acc = model.score(X_val_scaled, val_cat_label)
print('train_acc: ', train_acc, 'val_acc: ', val_acc)
train_CM = confusion_matrix(cat_label, train_pred, labels=classes)
print('train confusion matrix:\n', train_CM)
val_CM = confusion_matrix(val_cat_label, val_pred, labels=classes)
print('validation confusion matrix:\n', val_CM)
```