EE5112 Project1 Report Group7

Zhenbiao Huang A0285316B

Wu Fang

Mo Yifei

14/9/2023

Contents

1 Task 0. Recognize Facial Expressions

In this task, we built a linear logistic regression classification model, and used this model to predict the test set. Here are some key parts of the code. As requirement, we defined the Logistic Regression model with solver='sag':

```
model = linear_model.LogisticRegression(solver='sag')
model.fit(x, y)
y_test_pred = model.predict(np.vstack(dfTest['pixels'].values))
```

After training finished, there will have a warning:

```
ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
```

The first time we met this error, we tried to increase the max iteration to 1000, but we still got same warning. By searching Internet, we realized that logistic regression model may connot handle such complicated graph classification task. As a result, when the max intertion been reached, the model didn't converge. Here is our codes to visualze the accuracy:

```
accuracy = metrics.accuracy_score(dfTest.emotion, y_test_pred)
print(f'Accuracy: {accuracy}')

# get confusion matrix
matrix = metrics.confusion_matrix(dfTest.emotion, y_test_pred)
print(f'Confusion matrix:\n{matrix}')

# calculate the accuracy and print it
accuracy_each_class = metrics.classification_report(dfTest.emotion, y_test_pred,
                                                    target_names=emotion_label_to_text.values())
print(accuracy_each_class)
```

And this is how the accuracy outputs looked like:

```
Accuracy: 0.316
Confusion matrix:
[[ 26   1  16  36  16  14  26]
 [  0   0   4   2   4   1   5]
 [ 21   0  23  48  14  22  29]
 [ 24   0  18 146  30  11  18]
 [ 29   0  21  47  36  11  38]
 [ 13   0  15  24  12  42  12]
 [ 17   0  17  43  14  11  43]]
              precision    recall  f1-score   support

       anger       0.20      0.19      0.20       135
     disgust       0.00      0.00      0.00        16
        fear       0.20      0.15      0.17       157
```

```
   happiness       0.42      0.59      0.49       247
     sadness       0.29      0.20      0.23       182
    surprise       0.38      0.36      0.37       118
     neutral       0.25      0.30      0.27       145

    accuracy                           0.32      1000
   macro avg       0.25      0.25      0.25      1000
weighted avg       0.30      0.32      0.30      1000
```

Beacuse the model didn't converge, the accuracy is quiet low, but still better than guess randomly, which accuracy would be 0.14. We also try different parameters to train the model:

```
    for penalty in [ 'l2', 'none']:
    for c in [0.1, 0.2, 0.5, 1]:
        # TODO: calculate the accuracy
        model = linear_model.LogisticRegression(solver='sag', penalty=penalty, C=c)
        model.fit(x, y)
        y_test_pred = model.predict(np.vstack(dfTest['pixels'].values))
        print(penalty, c, metrics.accuracy_score(dfTest['emotion'].values, y_test_pred))
```

But seems these parameters cannot improve the model.

```
    l2 0.1 0.316
    l2 0.2 0.316
    l2 0.5 0.315
    l2 1 0.317
    None 0.1 0.316
    None 0.2 0.315
    None 0.5 0.315
    None 1 0.317
```

In conclusion, even if we tried different hyper-parameters, logistic regression model still cannot got a satisfying result. And we thought that's why in task1 and task2 we need to try neuron networks to solve this problem.

Task 1. Understand body language by gesture recognition with fully connected neural network

Body language is a type of nonverbal communication in which physical behaviors, as opposed to words, are used to express or convey the information, and gestures can be some of the most direct and obvious body language signals. By using gestures, people can use the virtual-manual modality to convey meaning. If robot can understand human body language, for example, gestures, then robot can receieving more data from their human co-workers, which would be beneficial to human-robot cooperation.

In task1, we built a fully connected neural network to recognize hand gesture. Here is the model part of the code:

```
import torch.nn as nn

class FCNNModel(nn.Module):
    def __init__(self, input_layer_size, hidden_layer_size, num_classes):
        super(FCNNModel, self).__init__()
        self.fc1 = nn.Linear(input_layer_size, hidden_layer_size)
        self.fc2 = nn.Linear(hidden_layer_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

```
model = FCNNModel(input_layer_size=H*W, hidden_layer_size=int(H*W/2), num_classes=
                                                num_classes)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
loss_func = nn.CrossEntropyLoss()
```

We notied that the model accuracy finally stop around 0.938

```
epoch=28  train loss=26.858157  train accuracy=0.935  test accuracy=0.812
epoch=29  train loss=4.186286 train accuracy=0.968  test accuracy=0.938
epoch=30  train loss=1.278437 train accuracy=0.984  test accuracy=0.938
epoch=31  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=32  train loss=0.011071 train accuracy=0.984  test accuracy=0.938
epoch=33  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=34  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=35  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=36  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=37  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=38  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
```

```
epoch=39  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=40  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
epoch=41  train loss=0.000000 train accuracy=1.000  test accuracy=0.938
```