Week 10 Assignment#10

Name : Sneetha Sree

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Assignment No.1a

```
# create training function

def train(features, target):
    '''
    Training model architecture and returns the model and losses
    :
    :
    losses = []
    model = model_architecture(features)
    for epoch in range:
        pred_y = model(features)
        loss = loss_fn(pred_y, target)
        losses.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    return model, losses
```

```
# predict function
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

Assignment No.1b

```
# create cv input function that splits tensors into groups

def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.1c

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.1d

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.2

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.3

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.4

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.5

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.6

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.7

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.8

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.9

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.10

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.11

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

Assignment No.12

```
# import numpy as np
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# create cv input function
def kcross(features, target):
    '''
    :
    :
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    for train_idx, test_idx in kfold.split(features):
        train_f = features[train_idx]
        train_t = target[train_idx]
        test_f = features[test_idx]
        test_t = target[test_idx]
    return train_f, train_t, test_f, test_t
```

```
# predict
predictions = f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.predict(features, verbose=0).flatten()"
print(f"{predictions}")
```

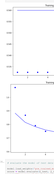
```
'this model'.predict(features, verbose=0).flatten()
```

```
# create cv
cv = kcross(features, target)
print(f"{'this matrix' if isinstance(features, torch.Tensor) else 'this model'}.shape, {'this matrix' if isinstance(target, torch.Tensor) else 'this model'}.shape")
```

```
'this model'.shape, 'this model'.shape
```

To CSV

```
# plot the curve
plt.clf()
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.savefig('./acc.jpg', dpi=300)
plt.clf()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.savefig('./loss.jpg', dpi=300)
plt.clf()
plt.title('Training and validation lr')
plt.plot(epochs, lr, 'bo', label='Training lr')
plt.plot(epochs, val_lr, 'b', label='Validation lr')
plt.savefig('./lr.jpg', dpi=300)
plt.clf()
```



To CSV

```
# Analysis the model of val_acc
model_name = model.name.replace('_', '')
model.save('./{}.h5'.format(model_name))
print('Saved trained model at %s ', './{}.h5'.format(model_name))
print('Training time: %s', (t1 - t0))
```

```
Saved trained model at %s  ./Resnet50.h5
Training time: %s  1.7e-05
```

```
Model of the Resnet: 50
```

END