mnist

November 14, 2025

[1]:
```
%reload_ext autoreload
%autoreload 2
```

[2]:
```
from kret_studies import *
from kret_studies.notebook import *
from kret_studies.complex import *

logger = get_notebook_logger()
```

```
Loaded environment variables from /Users/Akseldkw/coding/kretsinger/.env.
```

```
INFO:datasets:JAX version 0.7.2 available.
```

[3]:
```
from kret_studies.kret_torch.nn_mse_ce import ClassificationNN
```

[4]:
```
class MNISTClassifier(ClassificationNN):
    def set_model(self, num_classes: int) -> None:
        self.model = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.cls = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, num_classes),  # logits
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        h = self.model(x).to(self.device)  # (B, 64, 7, 7)
        logits = self.cls(h).to(self.device)  # (B, num_classes)
        return logits.to(self.device)  # raw logits for CrossEntropyLoss
```

[5]:
```
run = start_wandb_run(MNISTClassifier.name(), project="mnist")
```

```
wandb: Currently logged in as: akseldkw
(akseldkw07) to https://api.wandb.ai. Use `wandb login
--relogin` to force relogin
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

[6]:
```
KAGGLE_DIR = DATA_DIR / "kaggle"
MNIST_DIR = KAGGLE_DIR / "mnist"
```

[7]:
```
transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
    ]
)

train_dataset = datasets.MNIST(MNIST_DIR, download=True, train=True,␣
 ↪transform=transform)
test_dataset = datasets.MNIST(MNIST_DIR, download=True, train=False,␣
 ↪transform=transform)

train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=64, shuffle=True)
```

[8]:
```
num_cats = torch.unique(train_dataset.targets).shape[0]
num_cats
```

[8]:
```
10
```

[9]:
```
model = MNISTClassifier(patience=5)
```

[10]:
```
model.set_model(num_cats)
model.post_init()
```

```
[2025-11-14 12:21:21,976 | ERROR | MNISTClassifier_v000 ] Failed to load state
from /Users/Akseldkw/coding/kretsinger/data/pytorch/MNISTClassifier_v000: [Errno
2] No such file or directory: '/Users/Akseldkw/coding/kretsinger/data/pytorch/MN
ISTClassifier_v000/weights.pt'. Continuing with fresh weights.
[2025-11-14 12:21:21,977 | INFO | MNISTClassifier_v000 ] Full State:
{'hparams': {'batchsize': 128,
             'gamma': 0.1,
                'improvement_tol': 0.0001,
                'lr': 0.001,
                'patience': 5,
                'stepsize': 7},
 'state': {'best_eval_accuracy': '-inf%',
           'best_eval_f1': '-inf%',
           'best_eval_loss': inf,
           'best_eval_r2': '-inf%',
           'epochs_trained': 0}}
```

[11]:
```
model._log = False
```

[12]:
```
model.train_model(train_dataloader, test_dataloader, epochs=5_000,␣
 ↪batch_size=320)
```

```
  0%|          | 0/5000 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[12], line 1
----> 1␣
 ↪model.train_model(train_dataloader, test_dataloader, epochs=5_000, batch_size=320)

File ~/coding/kretsinger/kret_studies/kret_torch/mixin/train_mixin.py:40, in␣
 ↪SingleVariateMixin.train_model(self, train_loader, val_loader, epochs,␣
 ↪batch_size)
     37 labels: torch.Tensor = labels.to(device)
     39 self.optimizer.zero_grad()
---> 40 outputs = self(inputs).to(device)
     42 loss = self.get_loss(outputs, labels)
     43 loss.backward()

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
 ↪module.py:1751, in Module._wrapped_call_impl(self, *args, **kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
 ↪module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
```

```
   1764 result = None
   1765 called_always_called_hooks = set()

Cell In[4], line 21, in MNISTClassifier.forward(self, x)
     19 def forward(self, x: torch.Tensor) -> torch.Tensor:
     20     h = self.model(x).to(self.device)  # (B, 64, 7, 7)
---> 21     logits = self.cls(h).to(self.device)  # (B, num_classes)
     22     return logits.to(self.device)

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
↪module.py:1751, in Module._wrapped_call_impl(self, *args, **kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
↪module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
↪_forward_hooks or self._forward_pre_hooks
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
↪container.py:240, in Sequential.forward(self, input)
    238 def forward(self, input):
    239     for module in self:
--> 240         input = module(input)
    241     return input

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
↪module.py:1751, in Module._wrapped_call_impl(self, *args, **kwargs)
   1749     return self._compiled_call_impl(*args, **kwargs)  # type:␣
↪ignore[misc]
   1750 else:
-> 1751     return self._call_impl(*args, **kwargs)

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
↪module.py:1762, in Module._call_impl(self, *args, **kwargs)
   1757 # If we don't have any hooks, we want to skip the rest of the logic in
   1758 # this function, and just call forward.
   1759 if not (self._backward_hooks or self._backward_pre_hooks or self.
↪_forward_hooks or self._forward_pre_hooks
```

```
   1760         or _global_backward_pre_hooks or _global_backward_hooks
   1761         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1762     return forward_call(*args, **kwargs)
   1764 result = None
   1765 called_always_called_hooks = set()

File ~/micromamba/envs/kret_312/lib/python3.12/site-packages/torch/nn/modules/
 ↪linear.py:125, in Linear.forward(self, input)
    124 def forward(self, input: Tensor) -> Tensor:
--> 125     return F.linear(input, self.weight, self.bias)

RuntimeError: Tensor for argument weight is on cpu but expected on mps
```

[ ]:
```
f1_score(y_true, y_pred, average="weighted")
```

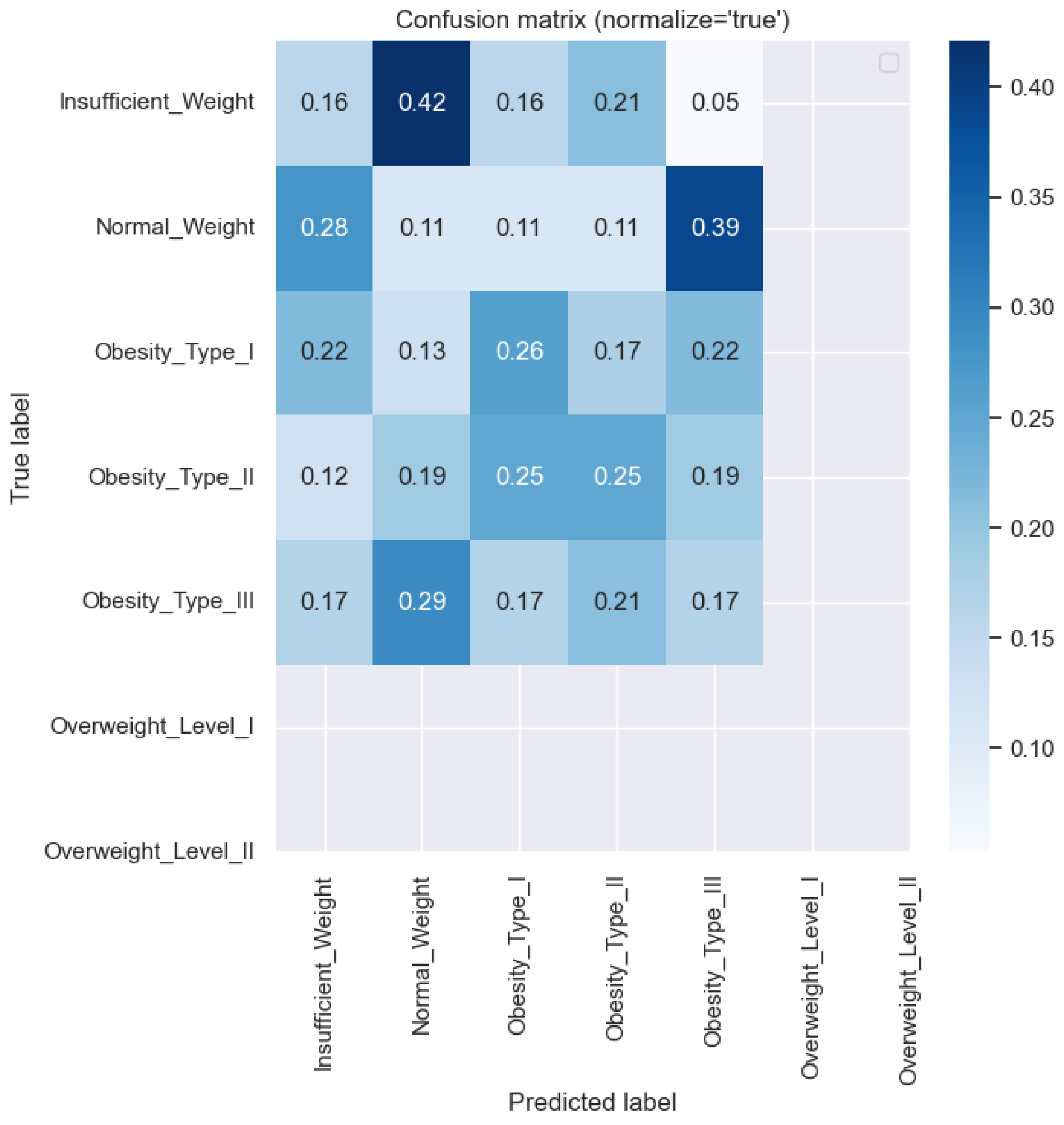
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[47], line 1
----> 1 f1_score(y_true, y_pred, average="weighted")

NameError: name 'y_true' is not defined
```

[ ]:
```
from sklearn.metrics import confusion_matrix

normalize = "true"  # or 'pred' or 'all' or None
cm = confusion_matrix(y_true, y_pred, normalize=normalize)

class_names = y.cat.categories.to_list()

fig, ax = uks_mpl.subplots(1, 1, 6, 6)
sns.heatmap(
    cm,
    ax=ax,
    annot=True,
    fmt=".2f" if normalize else "d",
    cmap="Blues",
    xticklabels=class_names,
    yticklabels=class_names,
)
ax.set_xlabel("Predicted label")
ax.set_ylabel("True label")
title = "Confusion matrix" + (f" (normalize='{normalize}')" if normalize else␣
 ↪"")
ax.set_title(title)
```

```
[ ]: Text(0.5, 1.0, "Confusion matrix (normalize='true')")
```

```
[ ]: fig
```

```
[ ]:
```



```
[ ]: # raise ValueError("STOP")
     run.finish()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

```

```