Nutrient Analysis

Alex Viller, 45375325

Contents

1 Data Cleaning and Preliminary Analysis

The Nutrient information file is a spreadsheet of different nutritional information. For this report I have analyzed the ”All solids & liquids per 100g” sheet. In this sheet, there are 293 features and 291 classes. Upon looking through the features I can see that there are features looking at the same things, but with different units, i.e. C4 (%T) vs C4 (g). I chose to drop the %T versions of these data points as they are just a function of the grams.

2 Appendix

2.1 PyTorch CNN

```python
class CNNClassifier(pl.LightningModule):
    def __init__(self, input_size, hidden_size, num_classes, lr, dropout_rate=0.5):
        super(CNNClassifier, self).__init__()
        # Calculate the number of channels for each convolutional layer
        conv1_channels = max(1, hidden_size // 8)
        conv2_channels = max(1, hidden_size // 4)
        conv3_channels = max(1, hidden_size // 2)
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=conv1_channels,
                kernel_size=2, stride=2)
        self.conv2 = nn.Conv1d(in_channels=conv1_channels,
                out_channels=conv2_channels, kernel_size=2, stride=2)
        self.conv3 = nn.Conv1d(in_channels=conv2_channels,
                out_channels=conv3_channels, kernel_size=2, stride=2)
        self.pool = nn.AdaptiveMaxPool1d(output_size=1)
        self.dropout = nn.Dropout(dropout_rate)
        self.batchnorm1 = nn.BatchNorm1d(conv1_channels)
        self.batchnorm2 = nn.BatchNorm1d(conv2_channels)
        self.batchnorm3 = nn.BatchNorm1d(conv3_channels)
        self.fc1 = nn.Linear(conv3_channels, conv1_channels)
        self.fc2 = nn.Linear(conv1_channels, num_classes)
        self.lossfn = nn.CrossEntropyLoss()
        nn.init.xavier_uniform_(self.fc1.weight) # Initialize fc1 weights
        nn.init.xavier_uniform_(self.fc2.weight) # Initialize fc2 weights
        self.learning_rate = lr

    def forward(self, x):
        x = x.unsqueeze(1)  # Add a channel dimension for 1D convolution
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = self.batchnorm1(x)
        x = self.dropout(x)
        x = x.expand(x.shape[0], x.shape[1], 2) if x.shape[-1] < 2 else x
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = x.expand(x.shape[0], x.shape[1], 2) if x.shape[-1] < 2 else x
        x = F.relu(self.conv3(x))
        x = self.pool(x)
        x = self.batchnorm3(x)
        x = self.dropout(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

2.2 CNN train, val, test loops and optimisation setup

```python

    def training_step(self, batch, batch_idx):
        x, y = batch
        # Forward pass
        y_hat = self(x)
        # Calculate loss
        loss = self.lossfn(y_hat, y)
        # Log accuracy and loss (optional)
        self.log("train_loss", loss, prog_bar=True)

        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = self.lossfn(y_hat, y)
        _, predicted = torch.max(y_hat, dim=1)
        # Calculate accuracy
        accuracy = torch.sum(predicted == y).item() / len(y)
        self.log("val_acc", accuracy, prog_bar=True, on_epoch=True)
        self.log("val_loss", loss, on_step=True, on_epoch=True, prog_bar=True, logger=True)

    def test_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = self.lossfn(y_hat, y)
        self.log("test_loss", loss, on_step=True, on_epoch=True, prog_bar=True,
                logger=True)
        _, predicted_labels = torch.max(y_hat, 1)
        accuracy = torch.sum(predicted_labels == y).item() / len(y)
        self.log("test_accuracy", accuracy, on_step=True, on_epoch=True, prog_bar=True,
                logger=True)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer=optimizer, step_size=35,
                gamma=0.1, verbose=True)
        scheduler_dict = {
            "scheduler": scheduler,
            "interval": "epoch",
            "monitor": "val_loss",
        }

        return [optimizer], [scheduler_dict]

```