Deeper Code Samples with Definitions

1. Fundamentals of Machine Learning

Definitions:

- Machine learning teaches models to find patterns in data.
- Supervised learning uses labeled examples.
- Unsupervised learning finds structure in unlabeled inputs.
- Reinforcement learning learns by trial and error with feedback.

Supervised Classification Example (Python + scikit-learn)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train model
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)

# Evaluate
preds = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, preds))
```

Unsupervised Clustering Example (Python + scikit-learn)

```
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load data
X, _ = load_wine(return_X_y=True)

# Reduce dimensions for visualization
pca = PCA(n_components=2)
X2 = pca.fit_transform(X)

# Cluster
km = KMeans(n_clusters=3)
labels = km.fit_predict(X2)
```

```
# Plot clusters
plt.scatter(X2[:,0], X2[:,1], c=labels)
plt.title("Wine Data Clusters")
plt.show()
```

2. Core ML Architecture & Workflow

Definition:

Core ML enables on-device inference by converting models into a unified **.mlmodel** format. The workflow goes from model training in Python to conversion, integration in Xcode, and on-device prediction.

Python Conversion with coremltools

```
import coremltools as ct
from sklearn.ensemble import RandomForestClassifier

# Train a simple model
clf = RandomForestClassifier(n_estimators=50)
clf.fit(X_train, y_train)

# Convert to Core ML format
mlmodel = ct.converters.sklearn.convert(clf,
                                        input_features=["sepal_length", "sepal_width",
"petal_length", "petal_width"],
                                        output_feature_names=["species"])
mlmodel.save("IrisClassifier.mlmodel")
```

Swift Integration & Inference

```
import CoreML

guard let model = try? IrisClassifier(configuration: MLModelConfiguration()) else {
    fatalError("Failed to load model")
}

let input = IrisClassifierInput(sepal_length: 5.1, sepal_width: 3.5, petal_length: 1.4,
petal_width: 0.2)
guard let prediction = try? model.prediction(input: input) else {
    fatalError("Prediction failed")
}

print("Predicted species:", prediction.species)
```

3. Model Types & Formats

Definition:

Core ML supports neural networks, tree ensembles, generalized linear models, and custom pipeline formats. Models must be packaged as **.mlmodel** or **.mlpackage** for app use.

PyTorch to Core ML Conversion

```
import torch
import coremltools as ct
from torchvision import models

# Load pretrained ResNet
model = models.resnet18(pretrained=True)
model.eval()

# Trace a sample input
example_input = torch.rand(1, 3, 224, 224)
traced_model = torch.jit.trace(model, example_input)

# Convert to Core ML
mlmodel = ct.convert(traced_model,
                     inputs=[ct.ImageType(name="input_image", shape=example_input.shape)])
mlmodel.save("ResNet18.mlmodel")
XGBoost to Core ML Conversion
import xgboost as xgb
import coremltools as ct

# Train a simple XGBoost model
dtrain = xgb.DMatrix(X_train, label=y_train)
params = {"objective": "multi:softprob", "num_class": 3}
bst = xgb.train(params, dtrain, num_boost_round=10)

# Convert
mlmodel = ct.converters.xgboost.convert(bst,
                                        feature_names=["f1", "f2", "f3", "f4"],
                                        target="species")
mlmodel.save("WineXGBoost.mlmodel")
```

4. Evaluation & Metrics

Definition:

Evaluating model performance ensures reliability. Common metrics include accuracy, precision, recall, F1 score, and AUC-ROC. Each metric highlights different aspects of correctness.

Computing Metrics (Python + scikit-learn)

```
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

# Assume y_test, preds, and pred_probs available
precision = precision_score(y_test, preds, average="macro")
recall = recall_score(y_test, preds, average="macro")
f1 = f1_score(y_test, preds, average="macro")
auc = roc_auc_score(y_test, pred_probs, multi_class="ovo")

print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1: {f1:.2f}, AUC: {auc:.2f}")
```

5. Optimization Techniques

Definition:

Optimizations shrink model size and speed up inference. Quantization reduces bit-width, pruning removes redundant weights, and distillation teaches a small model to mimic a large one.

Quantization with coremltools

```
import coremltools as ct

# Load existing .mlmodel
mlmodel = ct.models.MLModel("ResNet18.mlmodel")

# Apply 8-bit quantization
quantized = ct.models.neural_network.quantization_utils.quantize_weights(mlmodel, nbits=8)
quantized.save("ResNet18_Quantized.mlmodel")
```

6. Advanced Core ML Features

Definition:

Advanced features include on-device personalization, federated learning, and differential privacy. They enable models to adapt while preserving user data confidentiality.

On-Device Personalization (Swift)

```
import CoreML

// Load updatable model
let config = MLModelConfiguration()
config.parameters[.allowLowPrecisionAccumulationOnGPU] = true
guard let updatableModel = try? PersonalizedModel(configuration: config) else { return }

// Update with new sample
let updateInput = PersonalizedModelUpdateInput(feature: 0.75, label: 1)
try updatableModel.update(using: [updateInput])

// Use updated model
let result = try updatableModel.prediction(from: updateInput)
print("Updated prediction:", result.featureValue(for: "output")!)
```

7. Tooling & Integration

Definition:

Core ML tooling spans command-line and GUI. Create ML offers a macOS app for drag-and-drop training. coremltools provides Python APIs. Xcode's profiler helps debug performance.

Create ML Training (Swift Playground)

```
import CreateML
import Foundation

let dataURL = URL(fileURLWithPath: "/path/to/dataset.csv")
let data = try MLDataTable(contentsOf: dataURL)

let classifier = try MLTextClassifier(trainingData: data, textColumn: "text", labelColumn:
"label")
try classifier.write(to: URL(fileURLWithPath: "TextSentimentClassifier.mlmodel"))
```

8. Deployment & Versioning

Definition:

Over-the-air updates let apps fetch new models. Version checks and A/B tests ensure safe rollouts. Rollback strategies protect against performance regressions.

Swift OTA Model Update

```
import Foundation
import CoreML

func fetchModel(from url: URL, completion: @escaping (MLModel?) -> Void) {
    URLSession.shared.dataTask(with: url) { data, _, error in
        guard let data = data,
              let tempURL = try? FileManager.default
                .url(for: .documentDirectory, in: .userDomainMask, appropriateFor: nil,
create: true)
                .appendingPathComponent("UpdatedModel.mlmodelc"),
              error == nil else {
            completion(nil)
            return
        }
        try? data.write(to: tempURL)
        let model = try? MLModel(contentsOf: tempURL)
        completion(model)
    }.resume()
}
```

Python on-air Update

function using **`requests`** for downloading, and **`coremltools`** to load the model assuming working with **`.mlmodel`** format:

```
import requests
import os
import coremltools as ct

def fetch_model(url, save_dir=".", filename="UpdatedModel.mlmodel"):
    try:
        # Download the model file
        response = requests.get(url)
        response.raise_for_status()

        # Save to disk
        file_path = os.path.join(save_dir, filename)
        with open(file_path, "wb") as f:
            f.write(response.content)

        # Convert and load CoreML model
        model = ct.models.MLModel(file_path)
        return model

    except Exception as e:
        print(f"Error occurred: {e}")
        return None
```

🔍 **Notes:**

- This example assumes you're using the **`coremltools`** library for Python, which can handle **`.mlmodel`** files.
- Unlike Swift, Python doesn't compile models to **`.mlmodelc`** so we load the **`.mlmodel`** directly.
- You'll want to run this in a macOS environment for compatibility with CoreML tools.

These examples assume you're downloading a pre-trained model file (like `.h5` for TensorFlow or `.pt` for PyTorch) and loading it dynamically.

🧠 TensorFlow on-air Update

```
import requests
import os
import tensorflow as tf

def fetch_tf_model(url, save_dir=".", filename="updated_model.h5"):
    try:
        # Download the model file
        response = requests.get(url)
        response.raise_for_status()

        # Save to disk
        file_path = os.path.join(save_dir, filename)
        with open(file_path, "wb") as f:
            f.write(response.content)

        # Load the model
        model = tf.keras.models.load_model(file_path)
        return model

    except Exception as e:
        print(f"TensorFlow Error: {e}")
        return None
```

☑ This example uses TensorFlow's `load_model()` for HDF5 models (`.h5`). You can adjust if you're using `SavedModel` format instead.

🔥 **PyTorch on-air Update**

```python
import requests
import os
import torch

def fetch_torch_model(url, save_dir=".", filename="updated_model.pt", model_class=None):
    try:
        # Download the model file
        response = requests.get(url)
        response.raise_for_status()

        # Save to disk
        file_path = os.path.join(save_dir, filename)
        with open(file_path, "wb") as f:
            f.write(response.content)

        # Load the model weights
        if model_class is None:
            raise ValueError("You must pass a model class to load weights into.")

        model = model_class()  # Create instance
        model.load_state_dict(torch.load(file_path))
        model.eval()
        return model

    except Exception as e:
        print(f"PyTorch Error: {e}")
        return None
```

🛠️ This PyTorch version expects a `model_class` that defines the architecture to which the weights are applied. You can wrap this in a larger module to load specific model variants (e.g., ResNet, LSTM, etc.).