wtech
Teknolojide Kadın Derneği

Introduction to Machine Learning with Python

Model Evaluation and MLOps

Dr. Süha Tuna
İTÜ Informatics Institute

Standart evaluation

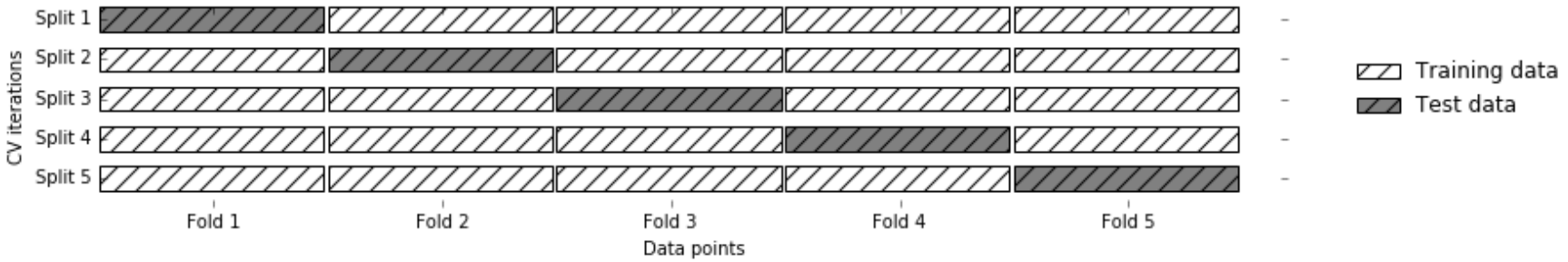
In[2]:

```
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# create a synthetic dataset
X, y = make_blobs(random_state=0)
# split data and labels into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)
# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))
```

Out[2]:

```
Test set score: 0.88
```

k-fold Cross-validation



In[4]:

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression()

scores = cross_val_score(logreg, iris.data, iris.target)
print("Cross-validation scores: {}".format(scores))
```

Out[4]:

```
Cross-validation scores: [ 0.961  0.922  0.958]
```

Cross-validation accuracy

In[5]:

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("Cross-validation scores: {}".format(scores))
```

Out[5]:

```
Cross-validation scores: [ 1.     0.967  0.933  0.9    1.   ]
```

In[6]:

```
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

Out[6]:

```
Average cross-validation score: 0.96
```

Benefits of Cross-Validation

- **More Reliable Performance Estimation:**
  - Mitigates the impact of "lucky" or "unlucky" random splits that could lead to unrealistically high or low test set accuracies.
  - Ensures every data example serves as part of the test set exactly once, requiring the model to generalize well across all samples.
- **Insights into Model Sensitivity:**
  - Provides information about how sensitive the model's performance is to different training data selections (e.g., observing a range of accuracies like 90%-100% on the Iris dataset).
  - Offers an idea of best-case and worst-case performance scenarios on new data.
- **More Effective Data Utilization:**
  - Allows a larger portion of the data to be used for training in each fold (e.g., 80% for 5-fold, 90% for 10-fold) compared to a typical 75% training split.
  - Generally leads to more accurate models due to training on more data.

Disadvantage of Cross-Validation:

- **Increased Computational Cost:**
  - Significantly slower as it requires training k models (where k is the number of folds) instead of just one, making it approximately k times slower than a single split.

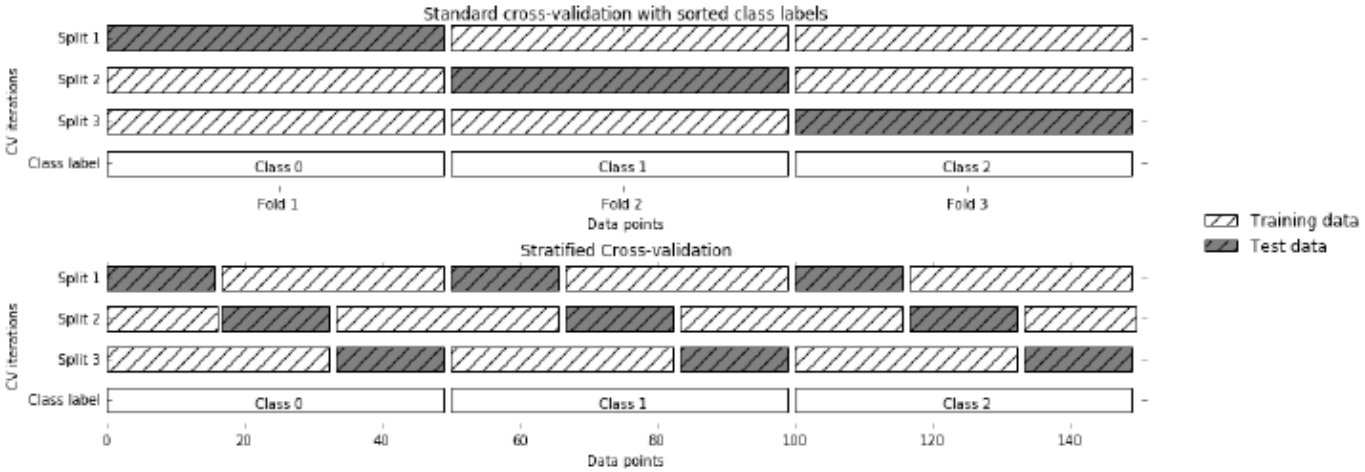
Stratified k-Fold Cross-Validation

In[7]:

```
from sklearn.datasets import load_iris
iris = load_iris()
print("Iris labels:\n{}".format(iris.target))
```

Out[7]:

```
Iris labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```



In[12]:

```
kfold = KFold(n_splits=3, shuffle=True, random_state=0)
print("Cross-validation scores:\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

Out[12]:

```
Cross-validation scores:
[ 0.9   0.96  0.96]
```

Grid Search

	C = 0.001	C = 0.01	...	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	...	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	...	SVC(C=10, gamma=0.01)
...	...	...	...	...
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	...	SVC(C=10, gamma=100)

In[18]:

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(
      X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

Out[18]:

```
Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

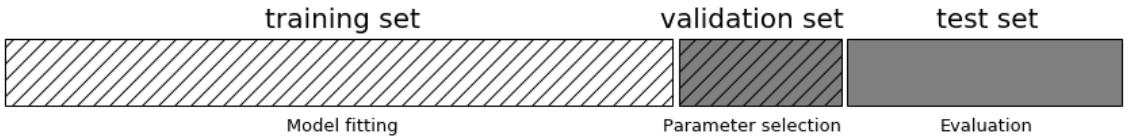
Three fold split

In[20]:

```
from sklearn.svm import SVC
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("Size of training set: {}   size of validation set: {}   size of test set:"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_valid, y_valid)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

Out[20]:

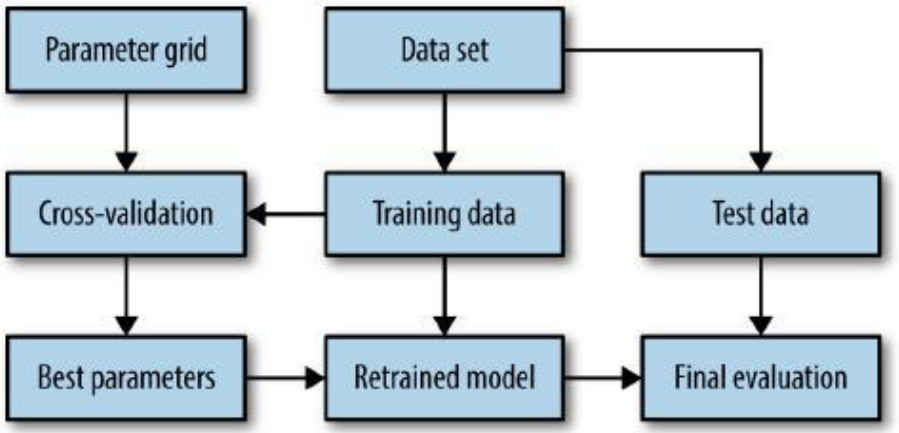
```
Size of training set: 84   size of validation set: 28   size of test set: 38

Best score on validation set: 0.96
Best parameters:  {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```



Grid search Cross-validation

In[21]:

```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```



GS with CV in sklearn

In[24]:

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_grid))
```

Out[24]:

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

In[25]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
```

In[26]:

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
```

In[27]:

```
grid_search.fit(X_train, y_train)
```

In[28]:

```
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

Out[28]:

```
Test set score: 0.97
```

In[29]:

```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

Out[29]:

```
Best parameters: {'C': 100, 'gamma': 0.01}
Best cross-validation score: 0.97
```

In[30]:

```
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

Out[30]:

```
Best estimator:
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
   max_iter=-1, probability=False, random_state=None, shrinking=True,
   tol=0.001, verbose=False)
```

Displaying GC with CV results

In[31]:

```
import pandas as pd
# convert to DataFrame
results = pd.DataFrame(grid_search.cv_results_)
# show the first 5 rows
display(results.head())
```

Out[31]:

	param_C	param_gamma	params	mean_test_score
0	0.001	0.001	{'C': 0.001, 'gamma': 0.001}	0.366
1	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	0.366
2	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	0.366
3	0.001	1	{'C': 0.001, 'gamma': 1}	0.366
4	0.001	10	{'C': 0.001, 'gamma': 10}	0.366

	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	22	0.375	0.347	0.363
1	22	0.375	0.347	0.363
2	22	0.375	0.347	0.363
3	22	0.375	0.347	0.363
4	22	0.375	0.347	0.363

	split3_test_score	split4_test_score	std_test_score
0	0.363	0.380	0.011
1	0.363	0.380	0.011
2	0.363	0.380	0.011
3	0.363	0.380	0.011
4	0.363	0.380	0.011

Confusion Matrix

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

In[41]:

```
from sklearn.metrics import confusion_matrix

confusion = confusion_matrix(y_test, pred_logreg)
print("Confusion matrix:\n{}".format(confusion))
```

Out[41]:

```
Confusion matrix:
[[401   2]
 [  8  39]]
```

Metrics

- **Accuracy**
  - **Definition:** The proportion of total predictions that were correct. It measures the overall correctness of the model.
  - **Formula:**

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{TP} + \text{TN} + \text{False Positives (FP)} + \text{False Negatives (FN)}}$$

  - **Interpretation:** A high accuracy means the model is generally good at making correct classifications.
  - **Limitations:** Can be misleading in imbalanced datasets. For example, if 95% of your data belongs to one class, a model that always predicts that class will achieve 95% accuracy, but it's not actually learning anything useful about the minority class.
- **Precision**
  - **Definition:** Of all the instances predicted as positive, how many were actually positive? It measures the quality of the positive predictions.
  - **Formula:**

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{TP} + \text{False Positives (FP)}}$$

  - **Interpretation:** High precision means a low rate of false positives. It's crucial when the cost of a false positive is high (e.g., misclassifying a non-spam email as spam, or diagnosing a healthy person with a serious disease).
  - **Analogy:** "When the model says it's positive, how often is it right?"
- **Recall (Sensitivity or True Positive Rate)**
  - **Definition:** Of all the actual positive instances, how many were correctly identified by the model? It measures the model's ability to find all relevant positive cases.
  - **Formula:**

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{TP} + \text{False Negatives (FN)}}$$

  - **Interpretation:** High recall means a low rate of false negatives. It's critical when the cost of a false negative is high (e.g., failing to detect a fraudulent transaction, or missing a cancerous tumor).
  - **Analogy:** "Of all the actual positives, how many did the model catch?"
- **F1-Score**
  - **Definition:** The harmonic mean of precision and recall. It provides a single score that balances both precision and recall.
  - **Formula:**

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

  - **Interpretation:** The F1-score is particularly useful when you need a balance between precision and recall, especially in cases of imbalanced datasets where neither high precision nor high recall alone tells the whole story. A high F1-score indicates that the model has good performance on both metrics.
  - **Why harmonic mean?** The harmonic mean penalizes extreme values. If either precision or recall is very low, the F1-score will also be low, forcing the model to perform well on both.

How to compute the metrics?

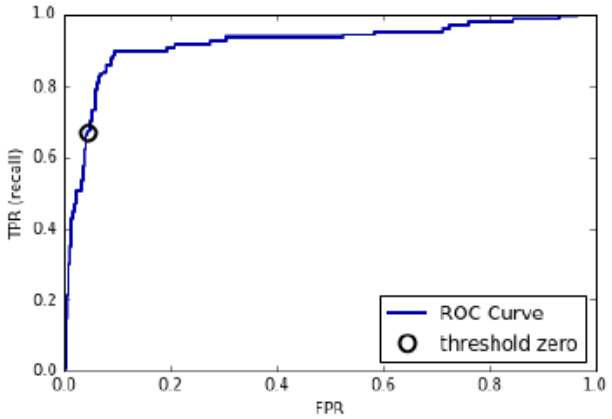
```python
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
```

```python
# True Positives (TP): Actual positive, Predicted positive
tp = np.sum((y_true == 1) & (y_pred == 1))
# True Negatives (TN): Actual negative, Predicted negative
tn = np.sum((y_true == 0) & (y_pred == 0))
# False Positives (FP): Actual negative, Predicted positive
fp = np.sum((y_true == 0) & (y_pred == 1))
# False Negatives (FN): Actual positive, Predicted negative
fn = np.sum((y_true == 1) & (y_pred == 0))
```

```python
sklearn_accuracy = accuracy_score(y_true, y_pred)
sklearn_precision = precision_score(y_true, y_pred, zero_division=0)
sklearn_recall = recall_score(y_true, y_pred, zero_division=0)
sklearn_f1 = f1_score(y_true, y_pred, zero_division=0)
```

Receiver operating characteristics (ROC) and AUC

```
In[59]:

    from sklearn.metrics import roc_curve
    fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))

    plt.plot(fpr, tpr, label="ROC Curve")
    plt.xlabel("FPR")
    plt.ylabel("TPR (recall)")
    # find threshold closest to zero
    close_zero = np.argmin(np.abs(thresholds))
    plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
             label="threshold zero", fillstyle="none", c='k', mew=2)
    plt.legend(loc=4)
```



- The ROC Curve (Receiver Operating Characteristic) is a plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied.

- It plots True Positive Rate (TPR) against False Positive Rate (FPR) at various threshold settings:
  - TPR = TP / (TP + FN) → Sensitivity or Recall
  - FPR = FP / (FP + TN)

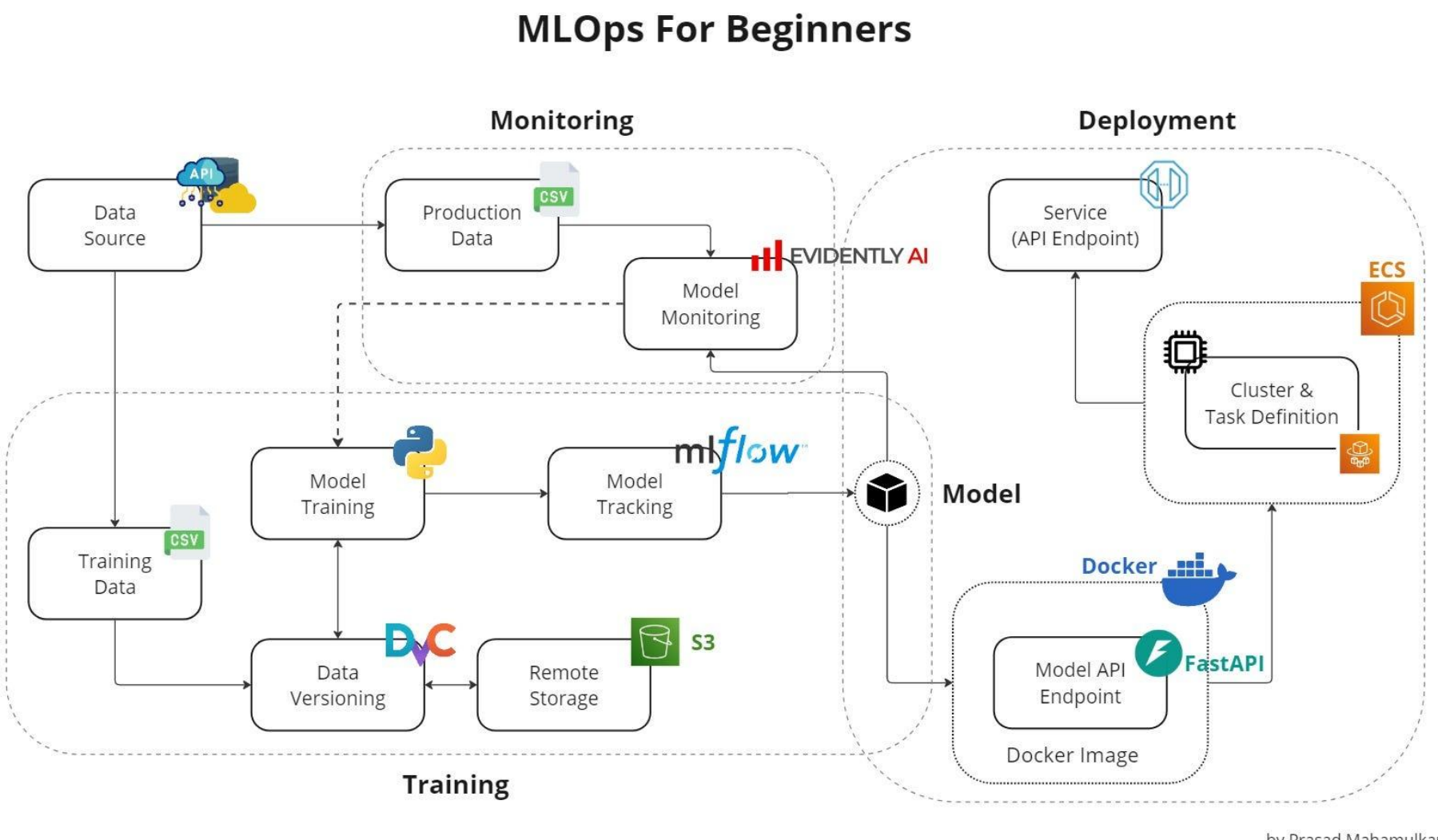
- The AUC (Area Under the Curve) quantifies the overall ability of the model to discriminate between positive and negative classes:
  - AUC = 1.0 → Perfect classifier
  - AUC = 0.5 → No discriminative power (equivalent to random guessing)

- The closer the ROC curve is to the top-left corner, the better the model's performance.

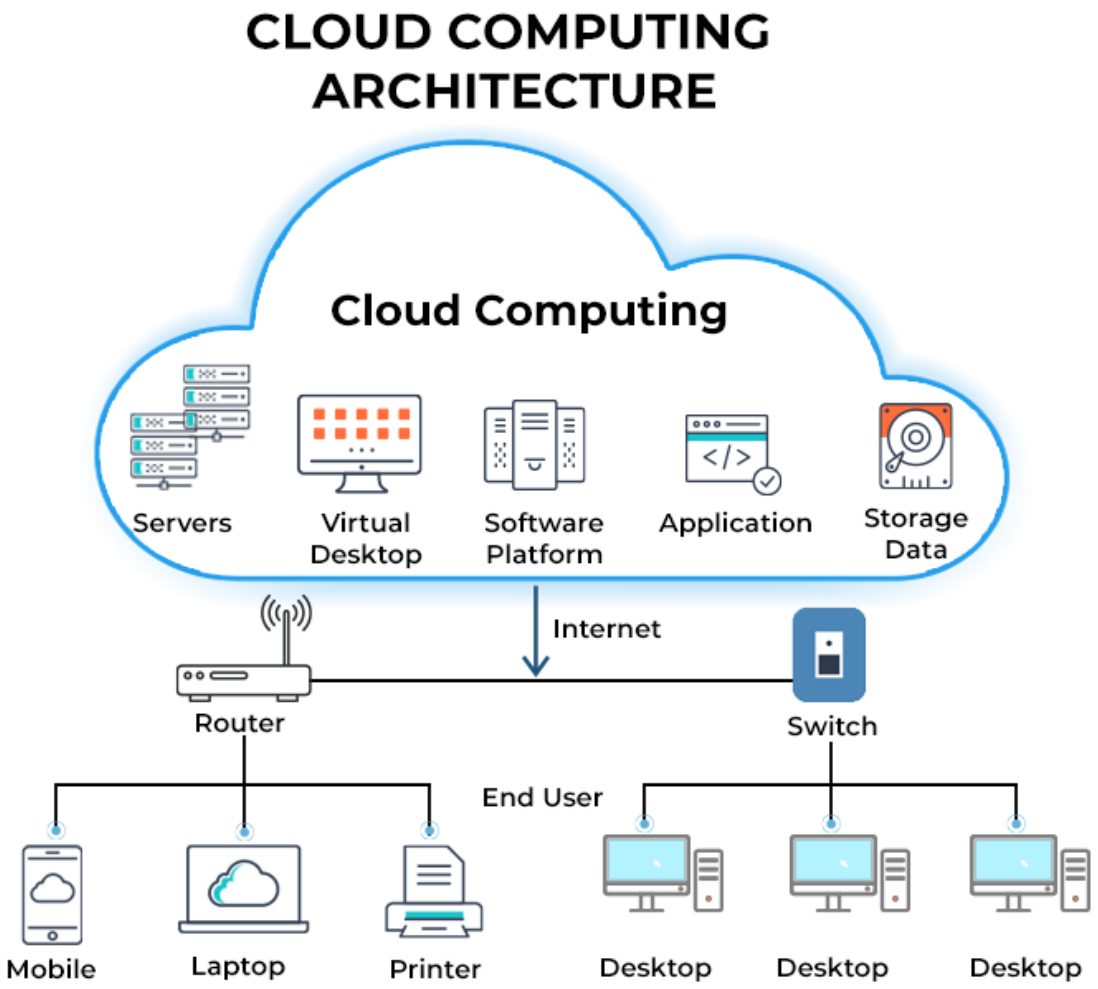
MLOps

Definition:
MLOps is the discipline of streamlining the **end-to-end machine learning lifecycle**, from data preparation and model development to deployment, monitoring, governance, and continuous integration/continuous delivery (CI/CD) of ML systems.

Core Objectives:
- **Automation** of the ML workflow
- **Versioning** of data, models, and code
- **Collaboration** between data scientists, ML engineers, and operations
- **Monitoring and maintenance** of deployed models
- **Scalability** and **reproducibility** of ML systems
- **Governance** and **compliance** for ML models



Cloud Computing



MLOps Application

1. Create a virtual environment

```bash
python -m venv venv
source venv/bin/activate    # On Windows: venv\Scripts\activate
```

2. Install dependencies

```bash
pip install scikit-learn joblib fastapi uvicorn pydantic
```

3. Train and save model: train_model.py

```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
import joblib

X, y = load_iris(return_X_y=True)
model = RandomForestClassifier()
model.fit(X, y)
joblib.dump(model, "iris_model.pkl")
```

4. Run the model

```bash
python train_model.py
```

5. Create the API service: main.py

```python
from fastapi import FastAPI
from pydantic import BaseModel
import numpy as np
import joblib

model = joblib.load("iris_model.pkl")
app = FastAPI(title="Iris Classifier API")

class IrisInput(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float

@app.post("/predict")
def predict(data: IrisInput):
    input_array = np.array([[data.sepal_length, data.sepal_width, data.petal_length, data.petal
    prediction = model.predict(input_array)[0]
    return {"prediction": int(prediction)}
```

Launching the application

6. Launch the API server

```bash
uvicorn main:app --reload
```

You should see output like:

```nginx
Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

7. Test the API

Open your browser and go to:

> http://127.0.0.1:8000/docs

Click `POST /predict` and test with:

```json
{
  "sepal_length": 5.1,
  "sepal_width": 3.5,
  "petal_length": 1.4,
  "petal_width": 0.2
}
```