UNIVERSITY of WEST FLORIDA

Machine Learning

Module 6:

Learning Best Practices for Model Evaluation and Hyperparameter Tuning

An overview of advanced techniques for model optimization

Brian Jalaian, Ph.D.

Associate Professor

Intelligent Systems & Robotics Department

Outline

UNIVERSITY of WEST FLORIDA

- Introduction to Model Evaluation & Hyperparameter Tuning
- Streamlining workflows with Pipelines
- K-fold Cross-validation
- Diagnosing Bias and Variance Problems with Learning Curves
- Fine-tuning Machine Learning Models via Grid Search
- Evaluating Models with Different Performance Metrics
- Dealing with Class Imbalance
- Summary & Next Steps

Introduction

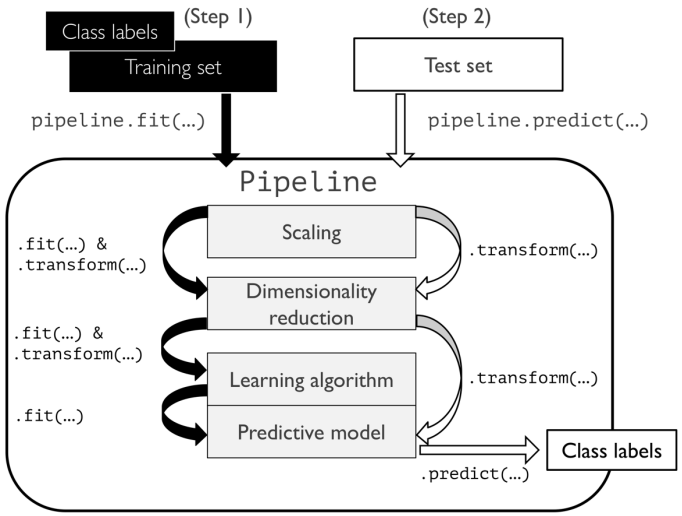
UNIVERSITY of WEST FLORIDA

- Importance of Model Evaluation and Hyperparameter Tuning
- Overview of Overfitting and Underfitting
- Significance of Model Evaluation, Hyperparameter Tuning, and Handling Class Imbalance in Machine Learning

Streamlining workflows with Pipelines

UNIVERSITY of WEST FLORIDA

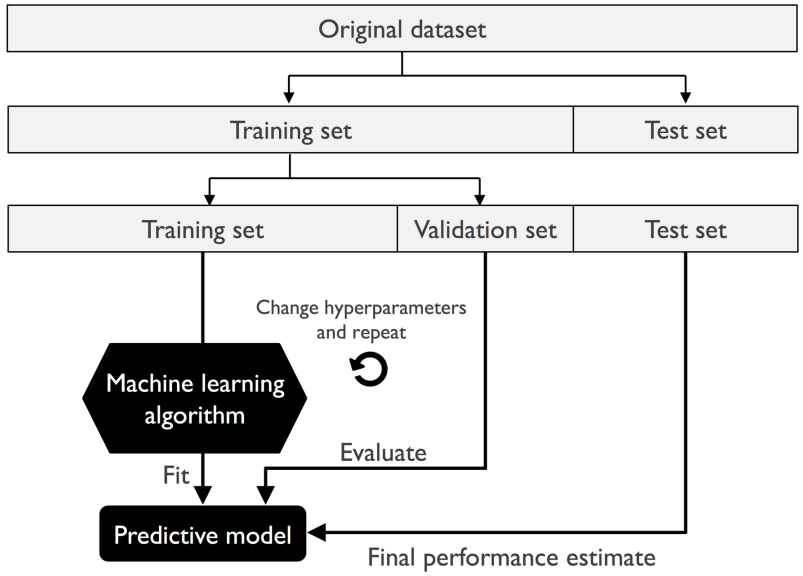
- Definition of Pipelines
- Benefits of Using Pipelines
- Implementing Pipelines in Scikit-Learn




```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

# Make pipeline
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1, solver='lbfgs'))

pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
```

Holdout Method

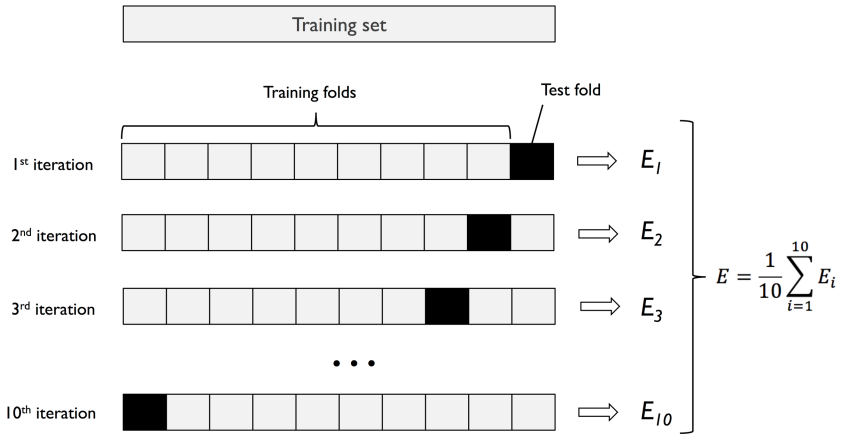
- Explanation of the Holdout Method
- Show how it's used for creating training and test sets
- Discuss its limitations leading to the need for K-fold Cross-validation



K-fold Cross-validation

UNIVERSITY of WEST FLORIDA

- What is K-fold Cross-validation?
- The Significance of K-fold Cross-validation in Model Selection and Evaluation
- Visualization and Example of K-fold Cross-validation




```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_lr,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)

print('Cross-validation accuracy scores: %s' % scores)
```

Stratified Cross-validation

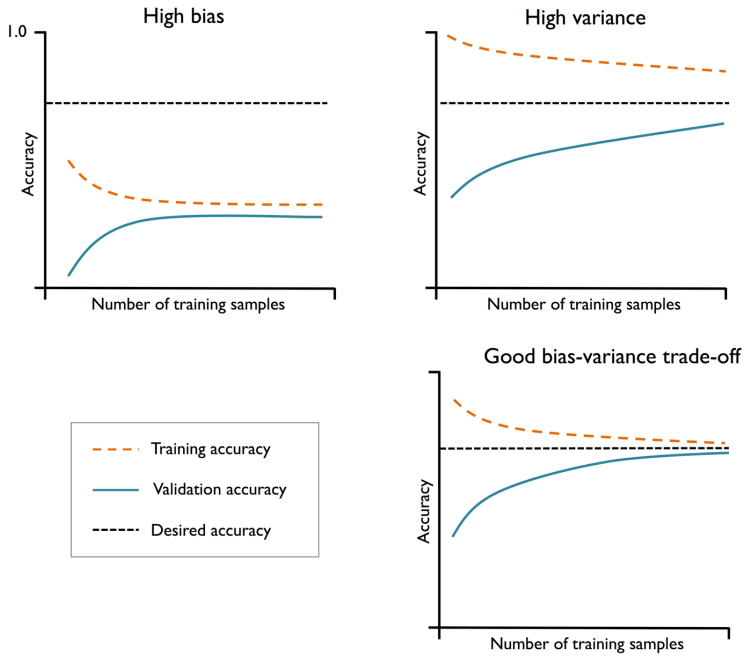
UNIVERSITY of WEST FLORIDA

- Explanation of Stratified Cross-validation
- Discuss how it helps to preserve the class proportions in each fold
- Example of how to implement Stratified Cross-validation with scikit-learn

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=10)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # fit and evaluate your model
```

Diagnosing Bias and Variance Problems with Learning Curves

- **Bias** is the error due to incorrect assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- **Variance** is the error due to high sensitivity to small fluctuations in the training set. High variance can cause overfitting: modeling the random noise in the training data, rather than the intended outputs.
- Learning Curves provide a graphical representation of how an algorithm learns as it is trained on more data.

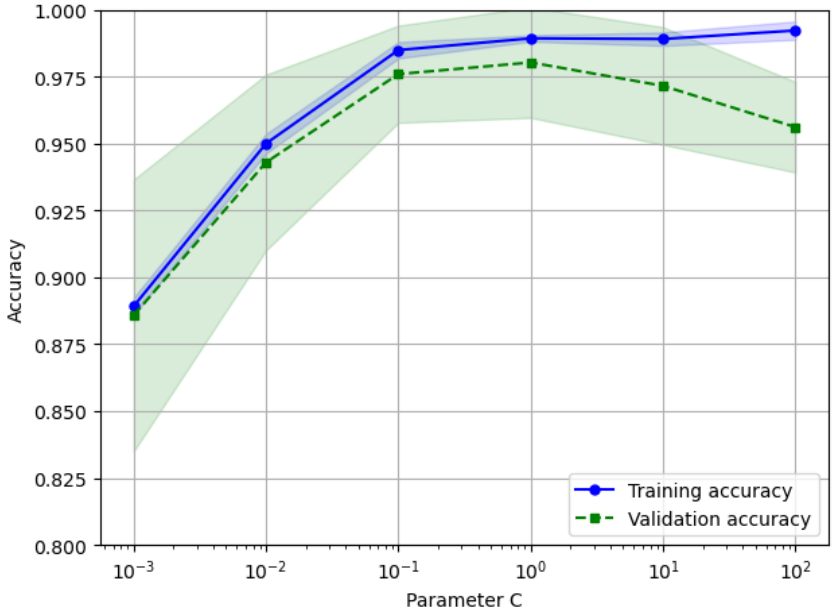



```
from sklearn.model_selection import learning_curve
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipeline,
    X=X_train,
    y=y_train,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=10,
    n_jobs=1)
```

Addressing over- and underfitting with Validation Curves

UNIVERSITY of WEST FLORIDA

- Explanation of what validation curves are
- Discuss how validation curves help in determining the model's complexity and addressing overfitting and underfitting issues
- Show an example of validation curve plots




```
from sklearn.model_selection import validation_curve
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
train_scores, test_scores = validation_curve(
                estimator=pipe_lr,
                X=X_train,
                y=y_train,
                param_name='logisticregression__C',
                param_range=param_range,
                cv=10)
```

Fine-tuning Machine Learning Models via Grid Search

UNIVERSITY of WEST FLORIDA

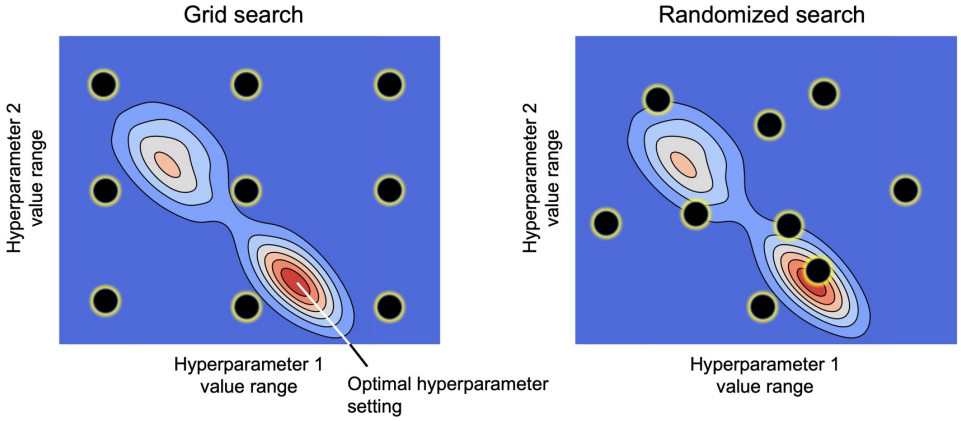
- Explanation of what Grid Search is
- Discuss how Grid Search aids in model hyperparameter tuning
- Show an example of Grid Search implementation

```
from sklearn.model_selection import GridSearchCV
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'clf__C': param_range,
               'clf__kernel': ['linear']},
              {'clf__C': param_range,
               'clf__gamma': param_range,
               'clf__kernel': ['rbf']}]

gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  refit=True,
                  cv=10,
                  n_jobs=-1)
gs = gs.fit(X_train, y_train)
```

Exploring Hyperparameter Configuration More Widely with Randomized Search

UNIVERSITY of WEST FLORIDA

- Discuss the idea of randomized search
- Talk about the advantages and trade-offs of using randomized search over grid search
- Show a code example of how to perform randomized search with scikit-learn




```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC

pipe_svc = make_pipeline(
    StandardScaler(),
    SVC(random_state=1)
)

param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

param_grid = [{'svc__C': param_range,
               'svc__kernel': ['linear']},
              {'svc__C': param_range,
               'svc__gamma': param_range,
               'svc__kernel': ['rbf']}
]

rs = RandomizedSearchCV(estimator=pipe_svc,
                        param_distributions=param_grid,
                        scoring='accuracy',
                        refit=True,
                        n_iter=20,
                        cv=10,
                        random_state=1,
                        n_jobs=-1)
```

More Resource-Efficient Hyperparameter Search with Successive Halving

UNIVERSITY of WEST FLORIDA

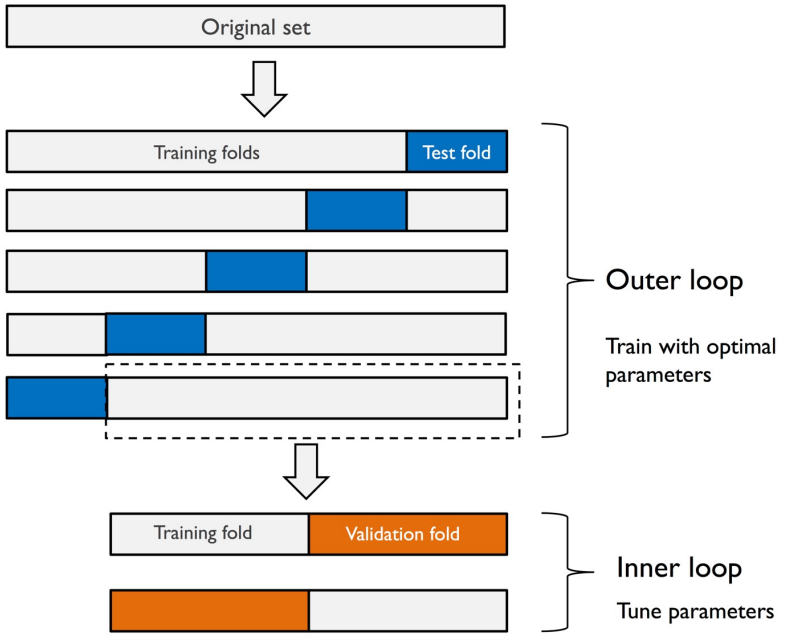
- Explanation of Successive Halving
- Discuss the benefits and practical uses of Successive Halving
- Show a code example using scikit-learn's `HalvingRandomSearchCV` function

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV

hs = HalvingRandomSearchCV(
    pipe_svc,
    param_distributions=param_grid,
    n_candidates='exhaust',
    resource='n_samples',
    factor=1.5,
    random_state=1,
    n_jobs=-1)

hs = hs.fit(X_train, y_train)
print(hs.best_score_)
print(hs.best_params_)

clf = hs.best_estimator_
print(f'Test accuracy: {hs.score(X_test, y_test):.3f}')
```

Algorithm Selection with Nested Cross-Validation

UNIVERSITY of WEST FLORIDA

- Explanation of Nested Cross-Validation
- Compare and contrast with standard k-fold cross-validation
- Code example of nested cross-validation using scikit-learn

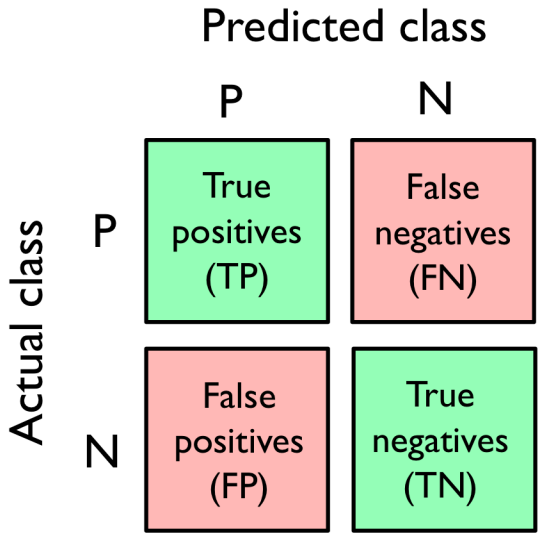



```
from sklearn.model_selection import cross_val_score

gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='accuracy', cv=5)
print(f'CV accuracy: {np.mean(scores):.3f} +/- {np.std(scores):.3f}')
```

Understanding Confusion Matrix

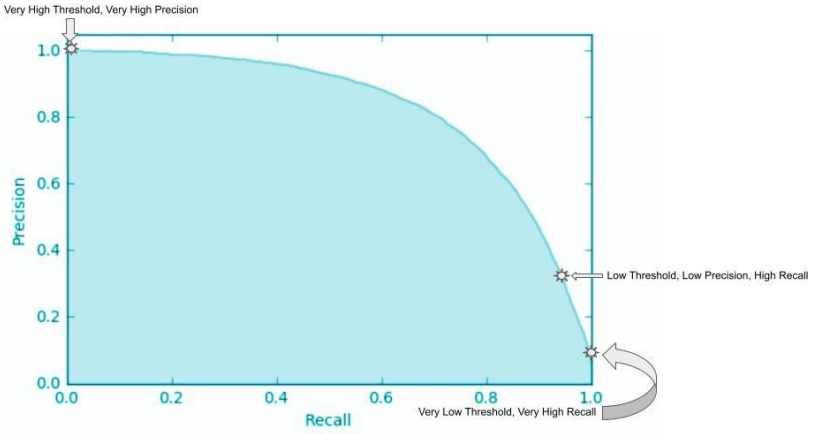
- Confusion Matrix: A tool for summarizing the performance of a classification algorithm.
- True Positives (TP) : Predicted class = Actual class = 1
- True Negatives (TN) : Predicted class = Actual class = 0
- False Positives (FP) (Type I Error): Predicted class = 1, but Actual class = 0
- False Negatives (FN) (Type II Error) : Predicted class = 0, but Actual class = 1

```
from sklearn.metrics import confusion_matrix
y_true = [1, 0, 1, 1, 0, 1]
y_pred = [0, 0, 1, 1, 0, 1]
conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)
```



Optimizing the Precision and Recall of a Classification Model

- Precision: Measures the accuracy of the positive predictions. Formula: $Precision = \frac{TP}{TP+FP}$
- Recall (or Sensitivity or True Positive Rate): Measures the fraction of positives that were correctly identified. Formula: $Recall = \frac{TP}{TP+FN}$
- Precision-Recall Trade-off: Increasing precision reduces recall, and vice versa. This is often visualized with a Precision-Recall Curve.




```
from sklearn.metrics import precision_recall_curve

precision, recall, _ = precision_recall_curve(y_test, y_scores)

plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```

The F1 Score and MCC: Balancing Precision and Recall

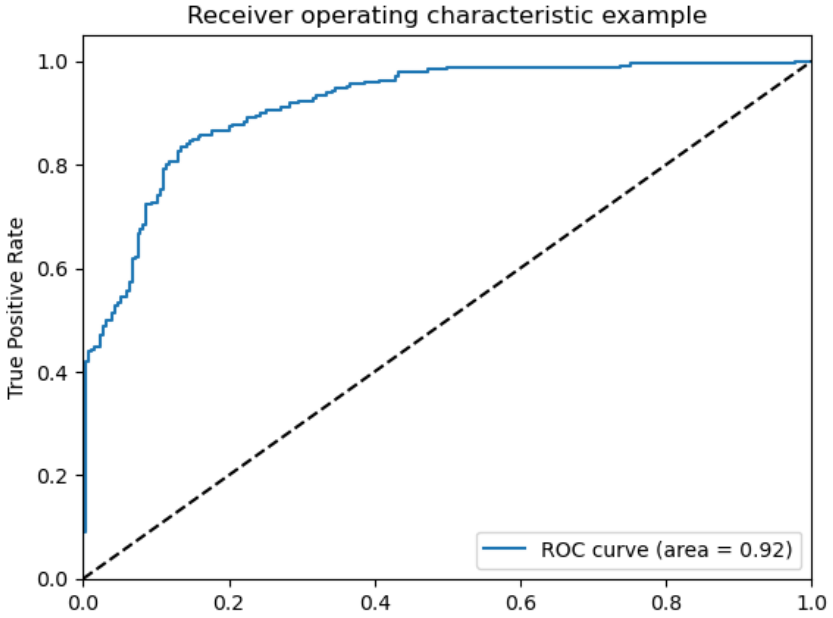
UNIVERSITY of WEST FLORIDA

- F1 Score: Harmonic mean of Precision and Recall
  - $F1Score = \frac{2\times(Precision\times Recall)}{Precision+Recall}$
- Matthews Correlation Coefficient (MCC): Balanced measure even when classes are of different sizes
  - $MCC = \frac{TP\times TN - FP\times FN}{\sqrt{(TP+FP)\times(TP+FN)\times(TN+FP)\times(TN+FN)}}$
- F1 score is best when you seek a balance between Precision and Recall and there is an uneven class distribution. MCC is used in binary classification problems and is regarded as a balanced measure even when the classes are of different sizes.

```
from sklearn.metrics import f1_score, matthews_corrcoef

# assuming y_test are the true labels and y_pred are the predicted labels
f1 = f1_score(y_test, y_pred)
mcc = matthews_corrcoef(y_test, y_pred)

print(f'F1 Score: {f1}')
print(f'Matthews Correlation Coefficient: {mcc}')
```

Receiver Operating Characteristic (ROC) and Area Under Curve (AUC)

- Receiver Operating Characteristic (ROC): Plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.
- Area Under Curve (AUC): The area underneath the ROC. AUC provides an aggregate measure of performance across all possible classification thresholds.
- ROC and AUC are used to evaluate the performance of a binary classifier, where a higher AUC indicates a better model.




```python
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# assuming y_test are the true labels and y_scores are the predicted scores
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = roc_auc_score(y_test, y_scores)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

Performance Metrics for Multiclass Classification

UNIVERSITY of WEST FLORIDA

- Discussing the extension of binary classification metrics to multi-class problems.
- Macro-Averaged Precision: It is the average precision calculated for each class, then averaged over all classes.
- The importance of choosing the right metric based on the problem at hand and the costs of different types of errors.

```
from sklearn.metrics import precision_score

# assuming y_test are the true labels and y_pred are the predicted labels
macro_precision = precision_score(y_test, y_pred, average='macro')

print(f'Macro-Averaged Precision: {macro_precision:.2f}')
```

Dealing with Class Imbalance

UNIVERSITY of WEST FLORIDA

- Explanation of the issue of class imbalance: when one class has significantly more examples than the other class.
- Strategies to handle class imbalance: Adjusting class_weight, upsampling, downsampling, generating synthetic training examples (SMOTE).
- Importance of handling class imbalance in the right manner to improve model performance.

```
from sklearn.utils import resample

# assuming X_train and y_train are the training dataset and labels

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_fraud = X[X.target==0]
fraud = X[X.target==1]

# upsample minority
fraud_upsampled = resample(fraud,
      replace=True, # sample with replacement
      n_samples=len(not_fraud), # match number in majority class
      random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([not_fraud, fraud_upsampled])

y_train_upsampled = upsampled.target
X_train_upsampled = upsampled.drop('target', axis=1)
```

Summary

UNIVERSITY of WEST FLORIDA

- Recap of the importance of model evaluation and hyperparameter tuning
- Revisiting the use of pipelines in streamlining workflows
- Emphasizing the importance of validation techniques like K-fold Cross-validation, Holdout Method, and Stratified Cross-validation
- Discussing bias-variance trade-off and methods to diagnose them with learning curves and validation curves
- Reflecting on the use of Grid Search, Randomized Search, Successive Halving for hyperparameter tuning
- Importance of different performance metrics: Confusion matrix, Precision, Recall, F1 Score, MCC, ROC Curve, and AUC
- Reiterating strategies to handle class imbalance for improved model performance