Handout 2: Comprehensive Multi-Metric Evaluation

Machine Learning for Smarter Innovation

1 Handout 2: Comprehensive Multi-Metric Evaluation

1.1 Week 9 - Intermediate Level

1.1.1 Introduction

You understand precision, recall, and F1 score. Now it's time to build production-ready validation pipelines that assess model performance across multiple dimensions, ensure statistical rigor, and enable confident deployment decisions.

Target Audience: Students with basic ML knowledge who have trained classification models and want to implement comprehensive evaluation systems.

Prerequisites: Python, pandas, sklearn basics, understanding of train/test splits, familiarity with binary classification metrics.

1.2 Part 1: Complete sklearn.metrics Tutorial

1.2.1 Classification Metrics API

```
from sklearn.metrics import (
    accuracy_score,
    precision_score, recall_score, f1_score, fbeta_score,
    roc_auc_score, average_precision_score,
    confusion_matrix, classification_report,
    roc_curve, precision_recall_curve,
    cohen_kappa_score, matthews_corrcoef
)
```

1.2.2 Binary Classification Example

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import *

# Create imbalanced dataset
X, y = make_classification(
    n_samples=10000,
    n_features=20,
```

```
    n_informative=15,
    n_redundant=5,
    weights=[0.9, 0.1],  # 10% positive class
    random_state=42
)

# Split with stratification
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

# Calculate all metrics
metrics = {
    'Accuracy': accuracy_score(y_test, y_pred),
    'Precision': precision_score(y_test, y_pred),
    'Recall': recall_score(y_test, y_pred),
    'F1': f1_score(y_test, y_pred),
    'F2 (favor recall)': fbeta_score(y_test, y_pred, beta=2),
    'F0.5 (favor precision)': fbeta_score(y_test, y_pred, beta=0.5),
    'ROC-AUC': roc_auc_score(y_test, y_proba),
    'PR-AUC': average_precision_score(y_test, y_proba),
    'Cohen Kappa': cohen_kappa_score(y_test, y_pred),
    'Matthews Corr': matthews_corrcoef(y_test, y_pred)
}

# Display
import pandas as pd
df = pd.DataFrame([metrics])
print(df.T.round(3))
```

Output:

```
                          0
Accuracy              0.950
Precision             0.857
Recall                0.750
F1                    0.800
F2 (favor recall)     0.769
F0.5 (favor precision) 0.833
ROC-AUC               0.945
PR-AUC                0.823
Cohen Kappa           0.789
Matthews Corr         0.794
```

1.2.3 Multi-Class Extensions

```
from sklearn.metrics import precision_recall_fscore_support

# Multi-class predictions
y_true_multi = [0, 1, 2, 2, 1, 0, 1, 2, 0]
y_pred_multi = [0, 1, 2, 1, 1, 0, 2, 2, 0]
```

```
# Calculate for each averaging strategy
for avg in ['macro', 'micro', 'weighted']:
    p = precision_score(y_true_multi, y_pred_multi, average=avg)
    r = recall_score(y_true_multi, y_pred_multi, average=avg)
    f = f1_score(y_true_multi, y_pred_multi, average=avg)
    print(f"{avg:8s}: P={p:.3f}, R={r:.3f}, F1={f:.3f}")
```

Averaging Strategies: - **Macro**: Unweighted mean across classes (treats all classes equally) - **Micro**: Aggregate TP/FP/FN globally (favors frequent classes) - **Weighted**: Weighted by class frequency (balances both)

1.3 Part 2: Cross-Validation Strategies

1.3.1 K-Fold Cross-Validation

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)

# Simple cross-validation (single metric)
scores = cross_val_score(model, X, y, cv=5, scoring='f1')
print(f"F1 scores: {scores}")
print(f"Mean: {scores.mean():.3f} (+/- {scores.std():.3f})")

# Multiple metrics simultaneously
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
results = cross_validate(model, X, y, cv=5, scoring=scoring)

# Organize results
import pandas as pd
cv_results = pd.DataFrame({
    metric: results[f'test_{metric}']
    for metric in scoring
})
print(cv_results.describe().loc[['mean', 'std']])
```

Output:

```
      accuracy  precision    recall        f1   roc_auc
mean     0.950      0.857     0.750     0.800     0.945
std      0.012      0.045     0.067     0.055     0.018
```

1.3.2 Stratified K-Fold (Imbalanced Data)

```
from sklearn.model_selection import StratifiedKFold

# Ensure each fold has same class distribution
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores = cross_val_score(model, X, y, cv=skf, scoring='f1')
print(f"Stratified F1: {scores.mean():.3f} (+/- {scores.std():.3f})")
```

1.3.3 Time Series Split (Temporal Data)

```
from sklearn.model_selection import TimeSeriesSplit

# For time-series data: train on past, test on future
tscv = TimeSeriesSplit(n_splits=5)

for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]
    # Train and evaluate
```

Key difference: No shuffling. Each split trains on past data, tests on future data (prevents data leakage).

1.3.4 Computing Confidence Intervals

```
import numpy as np
from scipy import stats

# 5-fold CV results
f1_scores = cross_val_score(model, X, y, cv=5, scoring='f1')

# 95% confidence interval
mean = f1_scores.mean()
std_err = f1_scores.std() / np.sqrt(len(f1_scores))
ci_95 = stats.t.interval(0.95, len(f1_scores)-1, loc=mean, scale=std_err)

print(f"F1: {mean:.3f} (95% CI: [{ci_95[0]:.3f}, {ci_95[1]:.3f}])")
```

1.4 Part 3: ROC and PR Curve Analysis

1.4.1 ROC Curve Visualization

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

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
         label='Random (AUC = 0.50)')
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')
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()
```

1.4.2 Multi-Model ROC Comparison

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.svm import SVC

models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100),
    'SVM': SVC(probability=True)
}

plt.figure(figsize=(10, 8))

for name, model in models.items():
    model.fit(X_train, y_train)
    y_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random (AUC = 0.50)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves: Model Comparison')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.show()
```

1.4.3 Precision-Recall Curve (Better for Imbalanced Data)

```
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)
pr_auc = average_precision_score(y_test, y_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkorange', lw=2,
         label=f'PR curve (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.grid(alpha=0.3)
plt.show()
```

When to use PR vs ROC: - **ROC curves**: Balanced datasets, threshold-independent evaluation - **PR curves**: Imbalanced datasets (focuses on minority class performance)

1.5 Part 4: Statistical Significance Testing

1.5.1 McNemar Test (Comparing Two Models)

```python
from statsmodels.stats.contingency_tables import mcnemar

# Train two models
model_a = LogisticRegression().fit(X_train, y_train)
model_b = RandomForestClassifier().fit(X_train, y_train)

# Predictions
pred_a = model_a.predict(X_test)
pred_b = model_b.predict(X_test)

# Contingency table
# n_01: Model A wrong, Model B correct
# n_10: Model A correct, Model B wrong
n_01 = np.sum((pred_a != y_test) & (pred_b == y_test))
n_10 = np.sum((pred_a == y_test) & (pred_b != y_test))

# McNemar test
table = np.array([[0, n_01], [n_10, 0]])
result = mcnemar(table, exact=False, correction=True)

print(f"Model A correct, Model B wrong: {n_10}")
print(f"Model A wrong, Model B correct: {n_01}")
print(f"Chi-square statistic: {result.statistic:.2f}")
print(f"p-value: {result.pvalue:.4f}")

if result.pvalue < 0.05:
    print("Significant difference (p < 0.05)")
else:
    print("No significant difference (p >= 0.05)")
```

Interpretation: - p $<$ 0.05: Models are significantly different - p $>=$ 0.05: Difference could be due to random chance

1.5.2 Permutation Test (Alternative)

```python
from sklearn.model_selection import permutation_test_score

# Test if model performs better than random
model = RandomForestClassifier(n_estimators=100, random_state=42)
score, perm_scores, pvalue = permutation_test_score(
    model, X, y, scoring='f1', cv=5, n_permutations=100, random_state=42
)

print(f"True F1 score: {score:.3f}")
print(f"Permutation F1 scores: {perm_scores.mean():.3f} (+/- {perm_scores.std
    ():.3f})")
print(f"p-value: {pvalue:.4f}")
```

1.6 Part 5: Systematic Model Comparison Framework

1.6.1 Pandas DataFrame for Organization

```python
import pandas as pd
from sklearn.metrics import *

def evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
```

```
    """Comprehensive single-model evaluation"""
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    return {
        'Model': model_name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1': f1_score(y_test, y_pred),
        'F2': fbeta_score(y_test, y_pred, beta=2),
        'ROC-AUC': roc_auc_score(y_test, y_proba),
        'PR-AUC': average_precision_score(y_test, y_proba)
    }

# Evaluate multiple models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
    ,
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100,
    random_state=42),
    'SVM': SVC(probability=True, random_state=42)
}

results = []
for name, model in models.items():
    result = evaluate_model(model, X_train, X_test, y_train, y_test, name)
    results.append(result)

# Create comparison DataFrame
df_comparison = pd.DataFrame(results)
df_comparison = df_comparison.set_index('Model')
print(df_comparison.round(3))

# Highlight best model per metric
print("\nBest model per metric:")
for col in df_comparison.columns:
    best_model = df_comparison[col].idxmax()
    best_score = df_comparison[col].max()
    print(f"{col:12s}: {best_model:20s} ({best_score:.3f})")
```

Sample Output:

```
                     Accuracy  Precision  Recall     F1     F2  ROC-AUC  PR
    -AUC
Model
Logistic Regression     0.945      0.820   0.710  0.761  0.731    0.932
    0.785
Random Forest           0.952      0.857   0.750  0.800  0.769    0.945
    0.823
Gradient Boosting       0.958      0.880   0.770  0.821  0.790    0.955
    0.847
SVM                     0.948      0.835   0.730  0.779  0.750    0.938
    0.798

Best model per metric:
Accuracy    : Gradient Boosting    (0.958)
Precision   : Gradient Boosting    (0.880)
Recall      : Gradient Boosting    (0.770)
F1          : Gradient Boosting    (0.821)
F2          : Gradient Boosting    (0.790)
```

```
ROC-AUC      : Gradient Boosting     (0.955)
PR-AUC       : Gradient Boosting     (0.847)
```

1.6.2 Visualization: Heatmap Comparison

```
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.heatmap(df_comparison, annot=True, fmt='.3f', cmap='RdYlGn', vmin=0, vmax
    =1)
plt.title('Model Comparison Heatmap')
plt.tight_layout()
plt.show()
```

1.7 Part 6: Threshold Optimization

1.7.1 Business-Driven Threshold Selection

```
def find_optimal_threshold(y_true, y_proba, cost_fn, cost_fp):
    """
    Find threshold that minimizes total cost.

    Parameters:
    - y_true: True labels
    - y_proba: Predicted probabilities
    - cost_fn: Cost of false negative
    - cost_fp: Cost of false positive
    """
    thresholds = np.linspace(0.01, 0.99, 99)
    costs = []

    for threshold in thresholds:
        y_pred = (y_proba >= threshold).astype(int)
        cm = confusion_matrix(y_true, y_pred)
        tn, fp, fn, tp = cm.ravel()

        total_cost = fn * cost_fn + fp * cost_fp
        costs.append(total_cost)

    optimal_idx = np.argmin(costs)
    optimal_threshold = thresholds[optimal_idx]
    optimal_cost = costs[optimal_idx]

    return optimal_threshold, optimal_cost, thresholds, costs

# Example: Credit risk (FN costs $50K, FP costs $5K)
optimal_thresh, optimal_cost, thresholds, costs = find_optimal_threshold(
    y_test, y_proba, cost_fn=50000, cost_fp=5000
)

print(f"Optimal threshold: {optimal_thresh:.2f}")
print(f"Optimal cost: ${optimal_cost:,.0f}")

# Visualize
plt.figure(figsize=(10, 6))
plt.plot(thresholds, costs, lw=2)
```

```
plt.axvline(optimal_thresh, color='r', linestyle='--',
            label=f'Optimal: {optimal_thresh:.2f}')
plt.axvline(0.5, color='gray', linestyle=':', label='Default: 0.50')
plt.xlabel('Decision Threshold')
plt.ylabel('Total Cost ($)')
plt.title('Threshold Optimization')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

1.7.2 Threshold Impact on Metrics

```
def threshold_analysis(y_true, y_proba, thresholds):
    """Show how metrics change with threshold"""
    results = []
    for thresh in thresholds:
        y_pred = (y_proba >= thresh).astype(int)
        results.append({
            'Threshold': thresh,
            'Precision': precision_score(y_true, y_pred),
            'Recall': recall_score(y_true, y_pred),
            'F1': f1_score(y_true, y_pred)
        })
    return pd.DataFrame(results)

thresholds = [0.3, 0.4, 0.5, 0.6, 0.7]
df_thresh = threshold_analysis(y_test, y_proba, thresholds)
print(df_thresh.round(3))

# Plot
df_thresh.plot(x='Threshold', y=['Precision', 'Recall', 'F1'],
               marker='o', figsize=(10, 6))
plt.title('Metrics vs Threshold')
plt.grid(alpha=0.3)
plt.show()
```

1.8 Part 7: Error Analysis

1.8.1 Confusion Matrix Deep Dive

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Detailed confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['Negative', 'Positive'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

# Extract components
tn, fp, fn, tp = cm.ravel()

print(f"True Negatives:  {tn} ({tn/len(y_test)*100:.1f}%)")
print(f"False Positives: {fp} ({fp/len(y_test)*100:.1f}%)")
print(f"False Negatives: {fn} ({fn/len(y_test)*100:.1f}%)")
```

```
print(f"True Positives:  {tp} ({tp/len(y_test)*100:.1f}%)")
```

1.8.2 Analyzing Misclassified Examples

```
# Find misclassified indices
misclassified_idx = np.where(y_pred != y_test)[0]

# False positives
fp_idx = np.where((y_pred == 1) & (y_test == 0))[0]
print(f"False Positives: {len(fp_idx)}")
print(f"Sample features:\n{X_test[fp_idx[:5]]}")

# False negatives
fn_idx = np.where((y_pred == 0) & (y_test == 1))[0]
print(f"False Negatives: {len(fn_idx)}")
print(f"Sample features:\n{X_test[fn_idx[:5]]}")

# Prediction confidence for errors
fp_proba = y_proba[fp_idx]
fn_proba = y_proba[fn_idx]

print(f"\nFalse Positive confidence: {fp_proba.mean():.3f} (+/- {fp_proba.std
    ():.3f})")
print(f"False Negative confidence: {fn_proba.mean():.3f} (+/- {fn_proba.std()
    :.3f})")
```

1.9 Part 8: Production Validation Pipeline

1.9.1 sklearn Pipeline Integration

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Define pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Hyperparameter grid
param_grid = {
    'classifier__n_estimators': [50, 100, 200],
    'classifier__max_depth': [10, 20, None],
    'classifier__min_samples_split': [2, 5, 10]
}

# Grid search with multiple metrics
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)
```

```
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best F1 score: {grid_search.best_score_:.3f}")

# Evaluate on test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
print(f"Test F1: {f1_score(y_test, y_pred):.3f}")
```

1.9.2 Custom Scorer for Business Metrics

```
from sklearn.metrics import make_scorer

def business_cost(y_true, y_pred, cost_fn=50000, cost_fp=5000):
    """Custom scorer: minimize business cost"""
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    return -(fn * cost_fn + fp * cost_fp)  # Negative because sklearn
    maximizes

# Use in cross-validation
cost_scorer = make_scorer(business_cost, greater_is_better=True,
                          cost_fn=50000, cost_fp=5000)

scores = cross_val_score(model, X, y, cv=5, scoring=cost_scorer)
print(f"Average business cost: ${-scores.mean():,.0f}")
```

1.10 Part 9: Common Pitfalls and Solutions

1.10.1 Pitfall 1: Data Leakage

Problem: Fitting preprocessors on entire dataset before split.

```
# WRONG
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Fitted on ALL data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)
# Test data has seen training data statistics!

# CORRECT
scaler = StandardScaler()
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train_scaled = scaler.fit_transform(X_train)  # Fit only on train
X_test_scaled = scaler.transform(X_test)  # Transform using train stats
```

1.10.2 Pitfall 2: Test Set Overfitting

Problem: Evaluating on test set multiple times during hyperparameter tuning.

Solution: Use train/validation/test split or nested cross-validation.

```
# Three-way split
from sklearn.model_selection import train_test_split

# Split 1: Separate test set (20%)
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,
                                                  stratify=y, random_state
    =42)

# Split 2: Train/validation (80% of remaining)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size
    =0.2,
                                                  stratify=y_temp,
    random_state=42)

# Use validation for hyperparameter tuning
# Use test ONLY ONCE at the end
```

1.10.3 Pitfall 3: Ignoring Class Imbalance in CV

Problem: Regular K-fold on imbalanced data creates folds with different class distributions.

Solution: Always use `StratifiedKFold`.

```
from sklearn.model_selection import StratifiedKFold

# Ensures each fold has same class distribution as original
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=skf, scoring='f1')
```

1.11 Part 10: Complete Evaluation Workflow

1.11.1 End-to-End Example

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, StratifiedKFold,
    cross_validate
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import *
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Create dataset
X, y = make_classification(n_samples=5000, n_features=20, weights=[0.9, 0.1],
                           random_state=42)

# 2. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# 3. Define models
models = {
```

```
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
    ,
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100,
    random_state=42)
}

# 4. Cross-validation
cv_results = {}
for name, model in models.items():
    scores = cross_validate(
        model, X_train, y_train,
        cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
        scoring=['accuracy', 'precision', 'recall', 'f1', 'roc_auc'],
        return_train_score=False
    )
    cv_results[name] = {
        metric: {'mean': scores[f'test_{metric}'].mean(),
                 'std': scores[f'test_{metric}'].std()}
        for metric in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
    }

# 5. Test set evaluation
test_results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    test_results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1': f1_score(y_test, y_pred),
        'ROC-AUC': roc_auc_score(y_test, y_proba)
    })

df_test = pd.DataFrame(test_results).set_index('Model')

# 6. Report
print("=== Cross-Validation Results (Mean +/- Std) ===")
for model_name, metrics in cv_results.items():
    print(f"\n{model_name}:")
    for metric, values in metrics.items():
        print(f"  {metric:12s}: {values['mean']:.3f} (+/- {values['std']:.3f})
    ")

print("\n=== Test Set Results ===")
print(df_test.round(3))

# 7. Visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Heatmap
sns.heatmap(df_test, annot=True, fmt='.3f', cmap='RdYlGn',
            vmin=0, vmax=1, ax=axes[0])
axes[0].set_title('Test Set Performance')

# ROC curves
for name, model in models.items():
    y_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
```

```
    auc_score = auc(fpr, tpr)
    axes[1].plot(fpr, tpr, lw=2, label=f'{name} (AUC={auc_score:.2f})')

axes[1].plot([0, 1], [0, 1], 'k--', lw=2)
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('ROC Curves')
axes[1].legend(loc='lower right')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

1.12 Key Takeaways

1. **Use multiple metrics**: No single metric tells the full story. Always calculate accuracy, precision, recall, F1, and AUC.
2. **Cross-validation is essential**: Single train/test split can be misleading. Use stratified 5-fold CV for robust estimates.
3. **Understand ROC vs PR curves**: ROC for balanced data, PR for imbalanced data.
4. **Test statistical significance**: Use McNemar or permutation tests to verify model differences aren't random.
5. **Optimize thresholds for business**: Default 0.5 threshold often suboptimal. Align with cost structure.
6. **Analyze errors systematically**: Understand which examples fail and why.
7. **Avoid data leakage**: Fit preprocessors only on training data. Use sklearn Pipeline.
8. **Protect test set**: Use validation set for tuning. Touch test set only once at the end.

1.13 Next Steps

- **Advanced Handout**: Custom business metrics, multi-objective optimization, Bayesian model selection, production monitoring
- **Week 9 Workshop**: Apply these techniques to credit risk validation challenge
- **Real Projects**: Implement comprehensive validation pipelines for your own ML projects

---

1.14 Resources

- sklearn documentation: https://scikit-learn.org/stable/modules/model_evaluation.html
- Cross-validation strategies: https://scikit-learn.org/stable/modules/cross_validation.html
- Threshold optimization: Elkan, C. (2001). "The Foundations of Cost-Sensitive Learning"
- Statistical tests: Dietterich, T. (1998). "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms"