# Handout 1: Understanding Model Performance Beyond Accuracy

Machine Learning for Smarter Innovation

## 1 Handout 1: Understanding Model Performance Beyond Accuracy

### 1.1 Week 9 - Basic Level

#### 1.1.1 Introduction

You've trained your first machine learning model and it shows "95% accuracy." Great, right? Not necessarily. This handout explains why accuracy alone can be misleading and introduces essential metrics for evaluating model performance.

**Target Audience**: Beginners with basic Python knowledge, no advanced ML background required.

---

### 1.2 Part 1: Why 95% Accuracy Can Be Terrible

#### 1.2.1 The Spam Filter Example

Imagine you build an email spam filter. Your dataset contains: - 9,500 legitimate emails (95%) - 500 spam emails (5%)

You create a "model" that simply predicts every email as "not spam." What happens?

**Accuracy = 9,500 correct / 10,000 total = 95%**

Sounds impressive! But your filter catches **zero spam emails**. It's completely useless despite high accuracy.

**Key Lesson**: When classes are imbalanced (one outcome much rarer than others), accuracy hides critical failures.

#### 1.2.2 Real-World Examples Where Accuracy Fails

**Medical Diagnosis**: - Disease prevalence: 1% (99 healthy, 1 sick per 100 people) - Model predicts everyone as "healthy" - Accuracy: 99% - Problem: Misses every sick patient (catastrophic)

**Fraud Detection**: - Fraud rate: 0.1% (1 in 1,000 transactions) - Model predicts all transactions as "legitimate" - Accuracy: 99.9% - Problem: Catches no fraud, loses millions

**Hiring AI**: - Qualified candidates: 10% - Model rejects everyone - Accuracy: 90% - Problem: No one gets hired

**Common Pattern**: Rare positive class + naive model = high accuracy but zero value.

---

## 1.3 Part 2: The Confusion Matrix - Four Numbers That Tell the Truth

### 1.3.1 Understanding the 2x2 Grid

Every binary classification model's predictions fall into four categories:

```
                      Actual  Condition
                   Positive    Negative
Predicted   Pos    TP (   )      FP (X)
            Neg    FN (X)      TN (   )
```

**Definitions**: - **TP (True Positive)**: Model said "yes," reality is "yes" (correct) - **TN (True Negative)**: Model said "no," reality is "no" (correct) - **FP (False Positive)**: Model said "yes," reality is "no" (wrong - false alarm) - **FN (False Negative)**: Model said "no," reality is "yes" (wrong - missed case)

### 1.3.2 Medical Diagnosis Example

You test 100 people for a disease: - 10 actually have the disease - 90 are healthy

Your model's results: - TP = 8 (correctly identified 8 sick patients) - FN = 2 (missed 2 sick patients) - TN = 85 (correctly identified 85 healthy people) - FP = 5 (false alarm: told 5 healthy people they're sick)

**Accuracy** = (TP + TN) / Total = (8 + 85) / 100 = 93%

But this doesn't tell you: - You missed 2 sick patients (could be fatal) - You gave 5 healthy people false alarms (unnecessary stress/treatment)

---

## 1.4 Part 3: Precision and Recall - The Essential Trade-off

### 1.4.1 Precision: "Of my positive predictions, how many were correct?"

**Formula**: Precision = TP / (TP + FP)

**Plain English**: When the model says "yes," how often is it right?

**Example**: Your fraud detector flags 100 transactions as fraudulent: - 80 are actually fraud (TP = 80) - 20 are legitimate (FP = 20) - Precision = 80 / 100 = 80%

**High Precision Means**: Few false alarms. When the model raises an alert, trust it.

**When to prioritize precision**: - Spam filtering (don't delete real emails) - Content moderation (don't ban innocent users) - Precision medicine (don't give unnecessary treatment)

### 1.4.2 Recall: "Of all actual positives, how many did I catch?"

**Formula**: Recall = TP / (TP + FN)

**Plain English**: Of all the "yes" cases, how many did the model find?

**Example**: There are 100 actual fraud cases in your dataset: - Model catches 80 (TP = 80) - Model misses 20 (FN = 20) - Recall = 80 / 100 = 80%

**High Recall Means**: Few missed cases. The model is thorough.

**When to prioritize recall**: - Cancer screening (don't miss any cases) - Security systems (catch all threats) - Fraud detection (find all fraud, even with false alarms)

### 1.4.3 The Fundamental Trade-off

**You cannot maximize both precision and recall simultaneously.**

**Example**: Airport Security

**High Recall Strategy** (catch every threat): - Search everyone thoroughly - Many false alarms (precision drops) - Long lines, but no threats slip through

**High Precision Strategy** (minimize false alarms): - Only search obvious suspects - Fast lines, fewer innocent people searched - But some threats might pass (recall drops)

**Key Insight**: Every problem requires balancing these two based on which error is more costly.

---

## 1.5 Part 4: When Is Your Model "Good Enough"?

### 1.5.1 Decision Framework

**Ask these questions**:

1. **What's the class balance?**
   - Balanced (50/50): Accuracy is reasonable
   - Imbalanced (1/99): Ignore accuracy, focus on precision/recall
2. **Which error is more costly?**
   - FP costs more: Optimize for precision
   - FN costs more: Optimize for recall
   - Both costly: Balance with F1 score (harmonic mean)
3. **What's the baseline?**
   - Always compare to a simple baseline (e.g., "always predict majority class")
   - If your model isn't better than the baseline, it's useless

### 1.5.2 Example Decision Trees

**Email Spam Filter**: - Class balance: 95% real, 5% spam (imbalanced) - Costly error: FP (deleting real email) - Metric: Precision (aim for 95%+) - Acceptable: Recall 70% (some spam gets through)

**Cancer Screening**: - Class balance: 99% healthy, 1% cancer (very imbalanced) - Costly error: FN (missing cancer) - Metric: Recall (aim for 99%+) - Acceptable: Precision 20% (many false alarms, but caught all cancer)

**Credit Scoring**: - Class balance: 95% repay, 5% default - Costly error: Both (lose money from defaults, lose customers from false rejections) - Metric: F1 score (balance precision and recall) - Target: F1 $>$ 0.75

---

## 1.6 Part 5: Using sklearn.metrics - Simple Examples

### 1.6.1 Basic Code Template

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix

# Assuming you have:
# y_true: actual labels (e.g., [0, 1, 0, 1, 1])
# y_pred: model predictions (e.g., [0, 1, 0, 0, 1])

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred)
```

```
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print(f"Accuracy:  {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall:    {recall:.2f}")
print(f"F1 Score:  {f1:.2f}")

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("\nConfusion Matrix:")
print(cm)
```

### 1.6.2 Complete Example

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Create imbalanced dataset (5% positive class)
X, y = make_classification(n_samples=1000, n_features=20,
                           weights=[0.95, 0.05], random_state=42)

# Split data
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 = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Comprehensive report
print(classification_report(y_test, y_pred))

# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

**Output Interpretation**:

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       190
           1       0.75      0.60      0.67        10

    accuracy                           0.97       200
```

This tells you: - Class 0 (negative): 98% precision, 99% recall (excellent) - Class 1 (positive): 75%

precision, 60% recall (room for improvement) - Overall accuracy 97% (misleading due to imbalance)

---

## 1.7 Part 6: Evaluation Checklist for Beginners

### 1.7.1 Before Training

☐ Understand the class distribution (balanced vs imbalanced)
☐ Define which error is more costly (FP vs FN)
☐ Choose primary metric (precision, recall, or F1)
☐ Establish baseline performance (e.g., majority class)

### 1.7.2 After Training

☐ Calculate accuracy (but don't trust it alone)
☐ Calculate precision and recall
☐ Create confusion matrix
☐ Compare to baseline
☐ Check performance on both classes separately

### 1.7.3 Before Deployment

☐ Test on held-out test set (never used in training)
☐ Analyze failure cases (which examples does it miss?)
☐ Validate on recent data (not just old historical data)
☐ Get stakeholder sign-off on acceptable performance

---

## 1.8 Part 7: Common Mistakes and How to Avoid Them

### 1.8.1 Mistake 1: Only Looking at Accuracy

**Problem**: Accuracy hides class imbalance issues.

**Solution**: Always calculate precision, recall, and F1. If classes are imbalanced, ignore accuracy entirely.

### 1.8.2 Mistake 2: Not Using Stratified Splits

**Problem**: Random train/test split might put all positive examples in training set.

**Solution**: Use `stratify=y` in `train\_test\_split()`:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

### 1.8.3 Mistake 3: Testing on Training Data

**Problem**: Model has "seen" the test data, giving falsely high performance.

**Solution**: Always keep test set completely separate. Never fit any preprocessing on test data.

### 1.8.4 Mistake 4: Forgetting to Evaluate Both Classes

**Problem**: "95% precision" might only be for the majority class.

**Solution**: Use `classification\_report()` which shows per-class metrics:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

### 1.8.5 Mistake 5: Choosing Wrong Metric for Problem

**Problem**: Optimizing precision when recall matters (or vice versa).

**Solution**: Talk to domain experts: - Medical: Usually recall (don't miss cases) - Finance: Usually precision (false positives costly) - Safety: Usually recall (catch all threats)

---

## 1.9 Part 8: Quick Reference

### 1.9.1 Metric Cheat Sheet

| Metric | Formula | When to Use | Good Value |
|---|---|---|---|
| Accuracy | (TP+TN)/Total | Balanced classes only | $> 0.80$ |
| Precision | TP/(TP+FP) | False positives costly | $> 0.90$ |
| Recall | TP/(TP+FN) | False negatives costly | $> 0.90$ |
| F1 Score | $2\times(P\times R)/(P+R)$ | Balance both errors | $> 0.75$ |

### 1.9.2 Code Snippets

**Calculate all metrics**:

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

acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred)
rec = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
```

**Confusion matrix**:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
```

**Classification report (all metrics)**:

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

---

## 1.10 Practice Exercise

### 1.10.1 Problem: Email Spam Filter

You have 1,000 emails: - 950 legitimate - 50 spam

Your model's predictions: - Correctly identified 40 spam emails (TP) - Missed 10 spam emails (FN) - Flagged 30 legitimate emails as spam (FP) - Correctly identified 920 legitimate emails (TN)

**Questions**:

1. What is the accuracy?
2. What is the precision?
3. What is the recall?
4. Is this a good model? Why or why not?
5. Which metric matters most for a spam filter?

### 1.10.2 Solutions

1. **Accuracy** = (TP + TN) / Total = (40 + 920) / 1000 = 0.96 (96%)
2. **Precision** = TP / (TP + FP) = 40 / (40 + 30) = 0.57 (57%)
3. **Recall** = TP / (TP + FN) = 40 / (40 + 10) = 0.80 (80%)
4. **Assessment**: Accuracy looks great (96%), but precision is poor (57%). This means:
   - The model catches 80% of spam (good recall)
   - But 43% of "spam" flags are false alarms (poor precision)
   - Users will see 30 legitimate emails incorrectly flagged
   - This could delete important emails (unacceptable)
5. **Most important metric**: Precision. For spam filters, false positives (deleting real emails) are worse than false negatives (letting some spam through). Aim for precision > 95%, even if recall drops to 60-70%.

---

## 1.11 Key Takeaways

1. **Accuracy is not enough**: It hides class imbalance and cost asymmetries.
2. **Confusion matrix is fundamental**: Always start by understanding TP, TN, FP, FN.
3. **Precision vs Recall trade-off**: You can't maximize both. Choose based on which error is costlier.
4. **Context determines metrics**: Medical (recall), Finance (precision), General (F1).
5. **Always use test data**: Never evaluate on training data. Use stratified splits for imbalanced classes.
6. **Compare to baseline**: If your model doesn't beat "always predict majority class," it's useless.

---

## 1.12 Next Steps

- **Intermediate Handout**: Learn cross-validation, ROC curves, and statistical testing
- **Advanced Handout**: Custom business metrics, multi-objective optimization, deployment strategies
- **Week 9 Workshop**: Apply these concepts to real credit risk validation challenge

---

## 1.13 Resources

**sklearn Documentation**: - Metrics: https://scikit-learn.org/stable/modules/model_evaluation.html - Classification metrics: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

**Tutorials**: - Classification metrics explained: https://developers.google.com/machine-learning/crash-course/classification - Precision-Recall trade-off: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

**Books**: - “Hands-On Machine Learning” by Aurelien Geron (Chapter 3: Classification) - “Introduction to Machine Learning with Python” by Andreas Muller (Chapter 5: Model Evaluation)