Machine Learning: Performance Evaluation

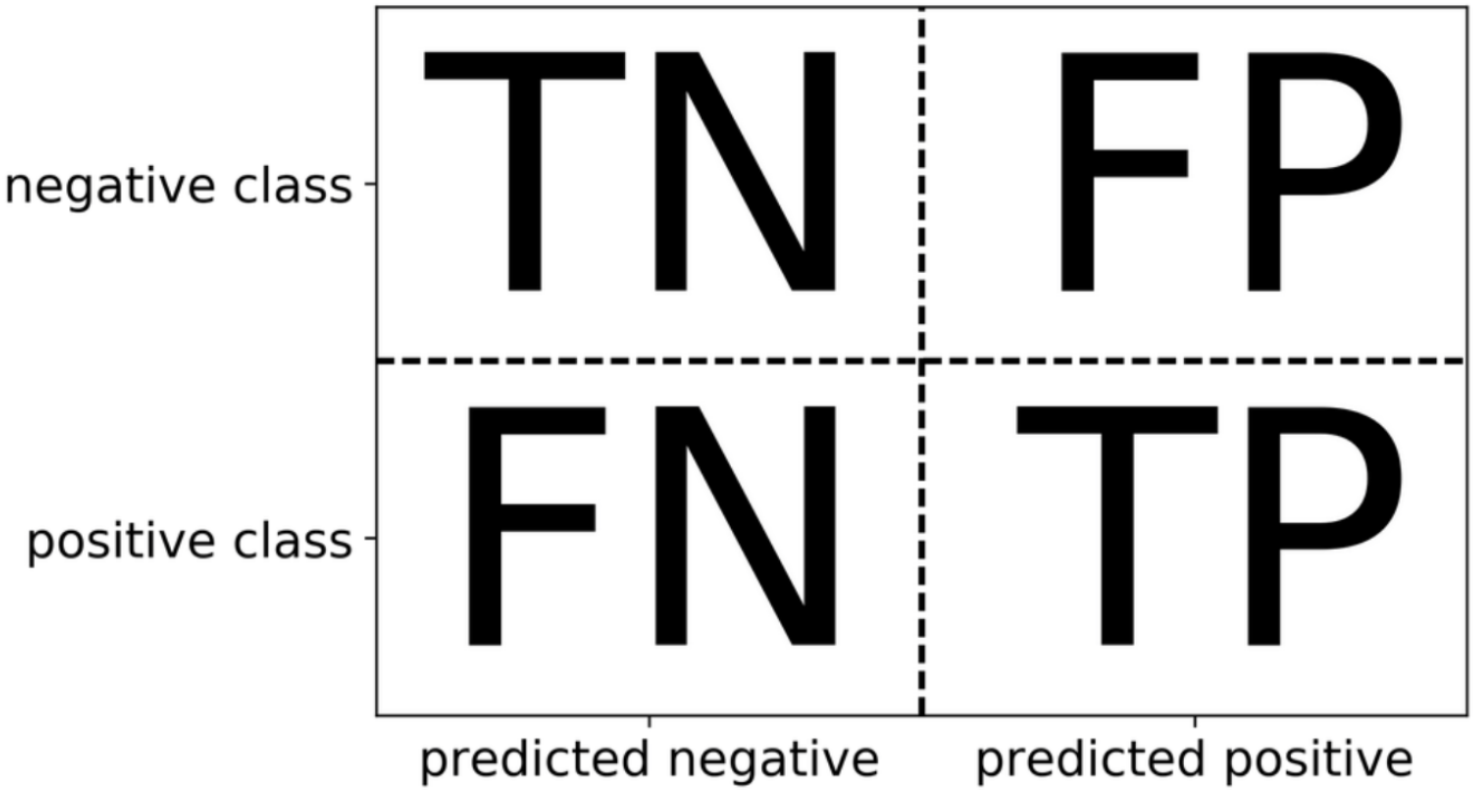
CSC 640: Advanced Software Engineering

James Walden

Northern Kentucky University

Topics

Confusion Matrix



$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Confusion Matrix Example

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
data = load_breast_cancer()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
  data.data, data.target, stratify=data.target, random_state=0)

lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
array([[49,  4],
       [ 5, 85]])

tn, fp, fn, tp = cm.ravel()
print(tn)
49
```

Accuracy

Accuracy is the percentage of correct classifications.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

```
print(round(lr.score(X_test, y_test), 3))
0.937
```

Problem: What if the data is unbalanced? If only 1% of files are malware, then a model that classifies **all** files as benign will is a 99% accurate malware detector.

Precision

Precision measures how many samples predicted as positive are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

Precision is used when the goal is to limit the number of false positives.

Problem: Precision can approach 1 if we identify only the sample we're mostly certain of as positive and classify all others as negative. Recall will be low.

Recall

Recall measures the fraction of positive samples that were identified by the model.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

Recall is used when we need to identify all positive samples, i.e. when it is important to avoid false negatives.

Problem: If model predicts all files are malware, there are zero false negatives and recall is 1. Precision will be low.

F-measure

F_1 is the harmonic mean of precision and recall

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2}$$

Provides a balanced consideration of both precision and recall, and can be a better metric of model performance than accuracy.

Performance Metrics in Scikit-learn

We can easily compute precision, recall, and F1 metrics.

```
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score
print(round(accuracy_score(y_test, y_pred), 3))
print(round(precision_score(y_test, y_pred), 3))
print(round(recall_score(y_test, y_pred), 3))
print(round(f1_score(y_test, y_pred), 3))
0.997
0.848
0.941
0.892
```

These results are for the payment fraud dataset.

Scikit-learn Classification Report

The classification report provides two sets of metrics.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
             precision    recall  f1-score   support

          0       0.91      0.92      0.92         53
          1       0.96      0.94      0.95         90

avg / total       0.94      0.94      0.94        143
```

First row of metrics is for 0 being the positive (cancer) class.
Second row is for 1 being the positive (cancer) class.

Changing Threshholds

Threshhold for Classification

By default, probabilistic classification models with a binary output classify a result as positive if the probability of a postive result based on the input data is greater than 50%.

What if we want to be more certain?

We can compute the probabilities and compare them to a user-specified threshhold (90% in the example below.)

```
y_pred = lr.predict_proba(X_test)[:,1] > 0.9

print(classification_report(y_test, y_pred))
             precision    recall  f1-score   support

          0       0.79      1.00      0.88        53
          1       1.00      0.84      0.92        90

avg / total       0.92      0.90      0.90       143
```

Precision Recall Curve

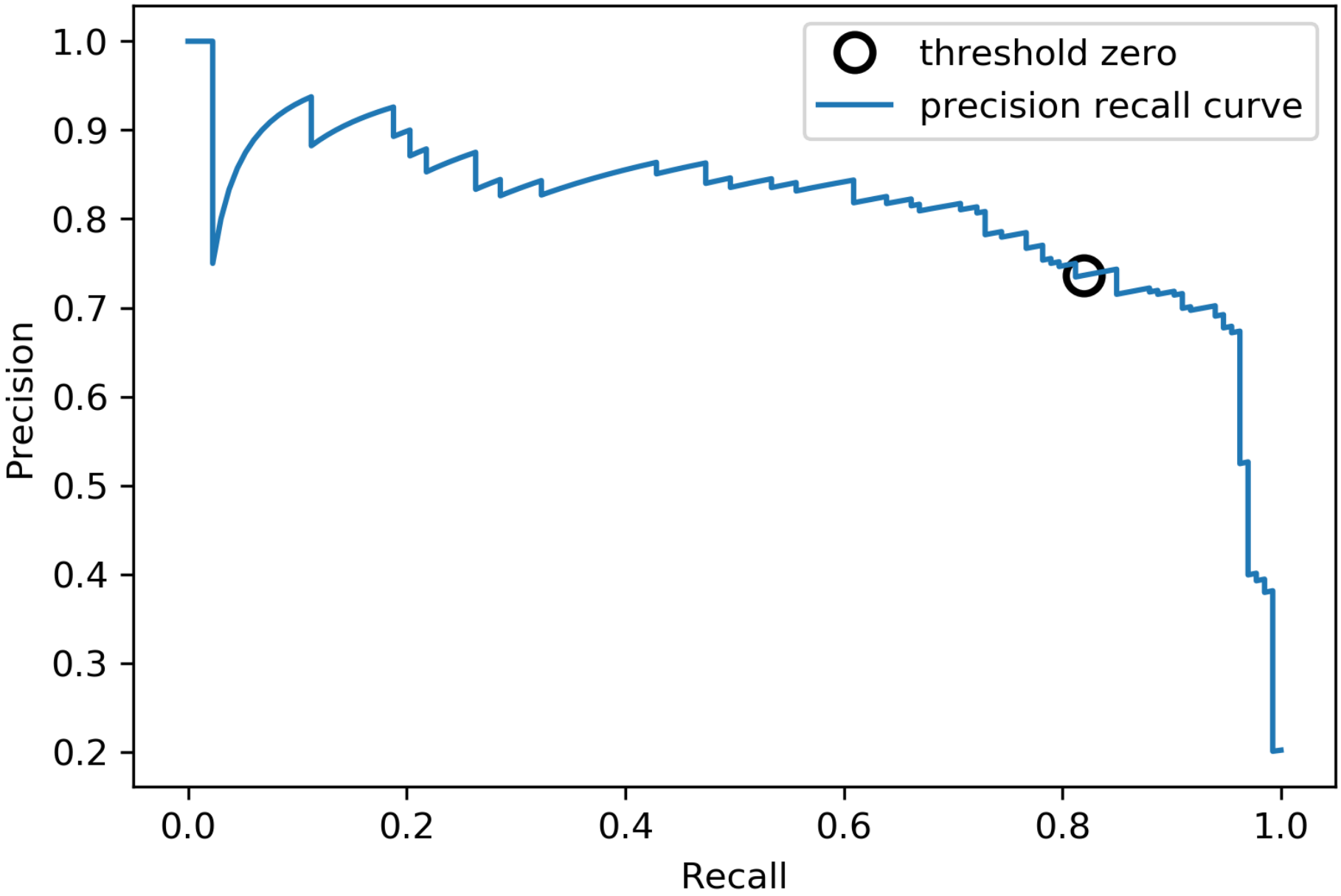
What if we don't know the best threshhold value?

We can examine model performance for all values of threshhold, computing precision and recall, and plotting them against each other.

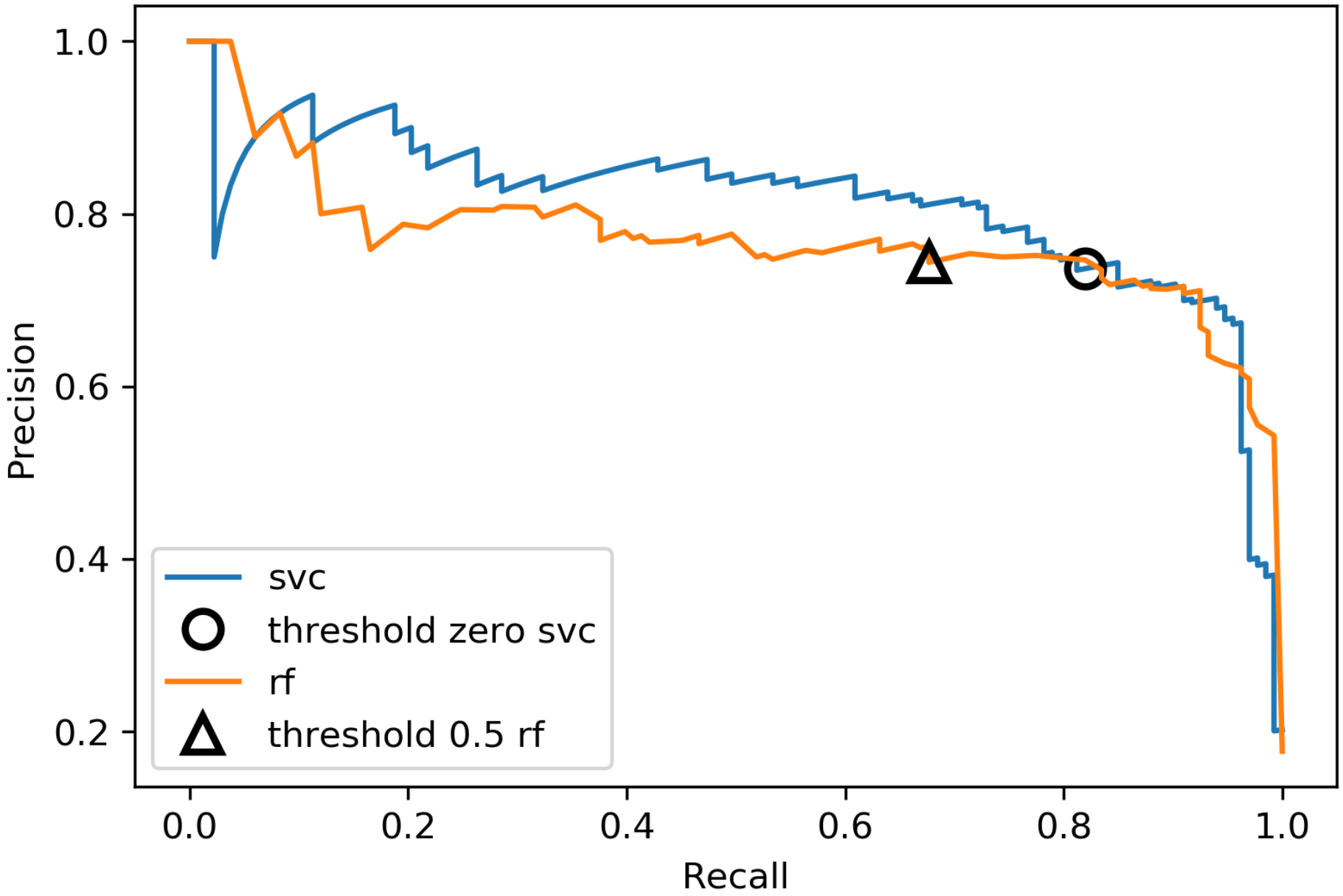
```
# obtain precision recall curve values for sample model
svc = SVC(gamma=.05).fit(X_train, y_train)
precision, recall, thresholds = precision_recall_curve(
    y_test, svc.decision_function(X_test))
plt.plot(precision, recall, label="precision recall curve")
```

Values in the upper right are best, representing models with both high precision and high recall. Threshhold zero is the default threshhold.

Precision Recall Curve



Comparing Precision Recall Curves



Area under the Precision Recall Curve

What if we want to automate model comparison?

We can compute the area under the precision recall curve, a quantity known as the **average precision**. It ranges from 0 to 1.

```
In[58]:
    from sklearn.metrics import average_precision_score
    ap_rf = average_precision_score(y_test,
        rf.predict_proba(X_test)[:, 1])
    ap_svc = average_precision_score(y_test,
        svc.decision_function(X_test))
    print("Avg precision of rf: {:.3f}".format(ap_rf))
    print("Avg precision of svc: {:.3f}".format(ap_svc))

Out[58]:
    Avg precision of rf: 0.666
    Avg precision of svc: 0.663
```

ROC Curve

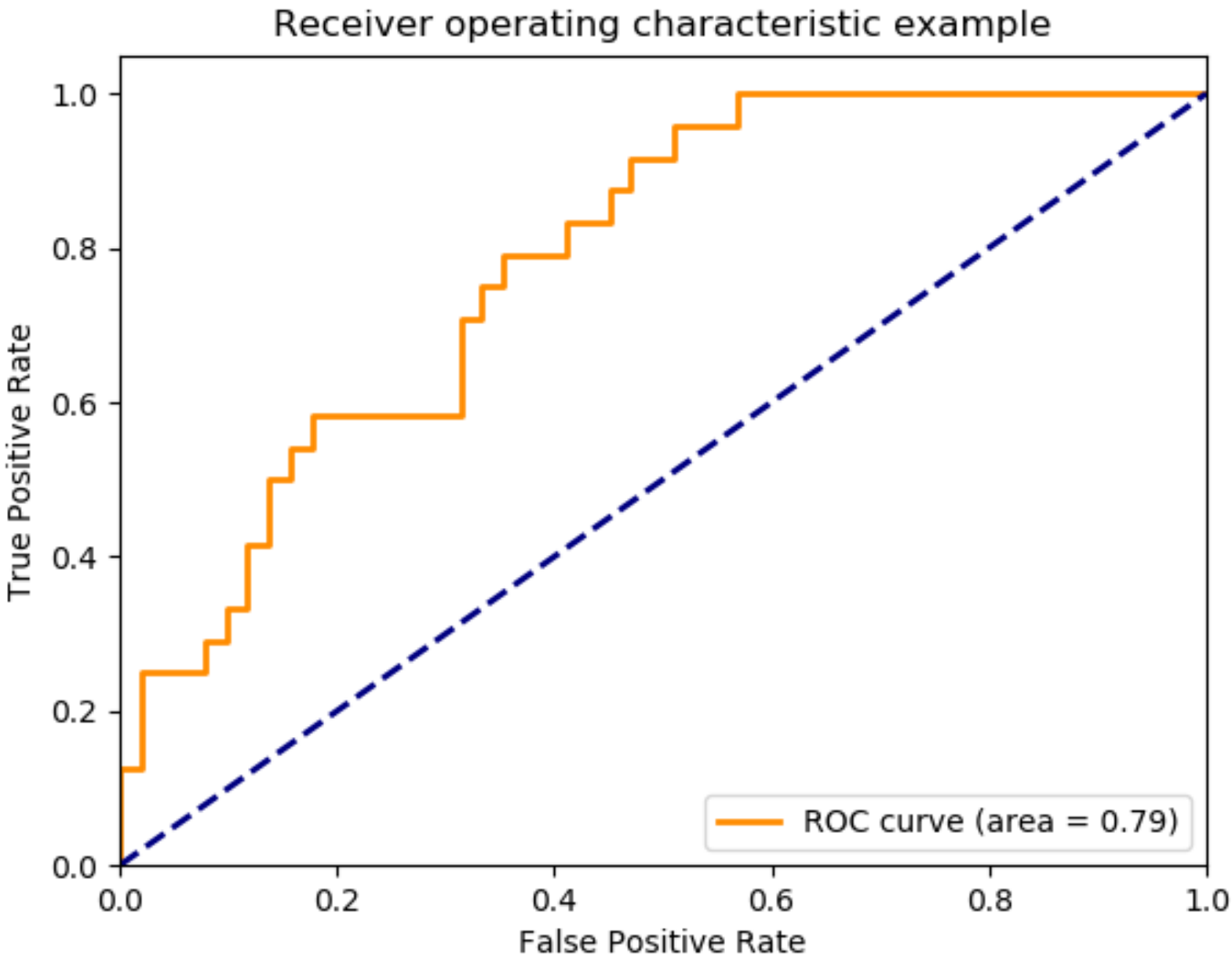
While precision-recall curves are a powerful way to compare models, especially with imbalanced classes, the older **Receiver Operating Characteristics (ROC) Curve** is more widely used.

The ROC curve compares the **False Positive Rate (FPR)** and the **True Positive Rate (TPR)**. TPR is called **sensitivity** and FPR is called **specificity** in some fields.

$$FPR = \frac{FP}{FP + TN}$$

$$TPR = \frac{TP}{TP + FN}$$

ROC Curve


Receiver operating characteristic example
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.79)

ROC Curve Interpretation

- The ideal model would be in the top left, with a false positive rate of 0 and a true positive rate of 1.
- **AUC metric**. We can compute the area under the ROC curve, which is often abbreviated to **AUC**, to provide a single number summary like average precision. It ranges from 0 to 1.
- The dashed line indicates the null model, which assigns every observation the same probability. It has an AUC of 0.5. All models we build should be above the null model line.

Random Data Splits Affect Model Performance

The effect here is measurable. Will it always be small?

```
In [13]: X_train1, X_test1, y_train1, y_test1 =
    train_test_split(X, y, test_size=0.2, random_state=1)
    ...: model1 = DecisionTreeClassifier()
    ...: model1.fit(X_train1, y_train1)
    ...: y_pred1 = model1.predict(X_test1)
    ...: round(accuracy_score(y_pred1, y_test1), 3)
Out[13]: 0.989
In [15]: X_train2, X_test2, y_train2, y_test2 =
    train_test_split(X, y, test_size=0.2, random_state=2)
    ...: model2 = DecisionTreeClassifier()
    ...: model2.fit(X_train2, y_train2)
    ...: y_pred2 = model2.predict(X_test2)
    ...: round(accuracy_score(y_pred2, y_test2), 3)
Out[15]: 0.982
```

Cross Validation

Cross-validation is a method of evaluating performance where

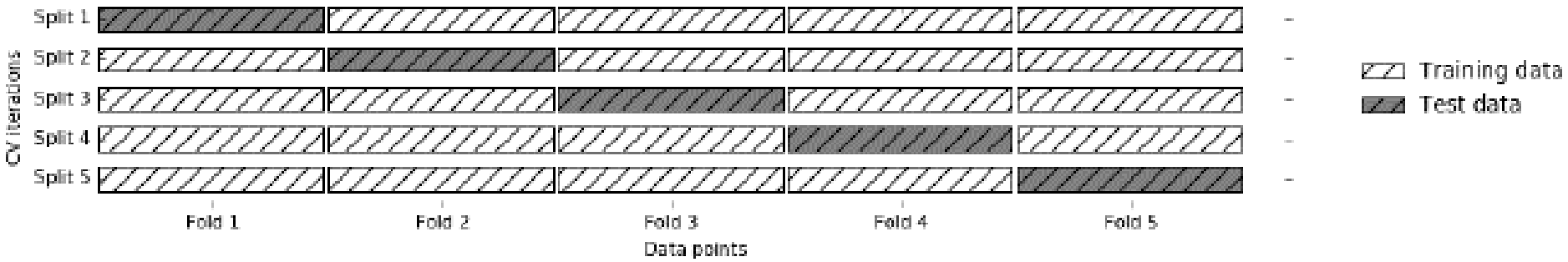
- The data set is split repeatedly and differently
- Multiple models are trained on the different splits

The method divides the dataset into k equal-sized folds, then

- Builds k models using
- Each fold once as the test dataset.
- The other folds as the training dataset.

5-fold Cross Validation

1. Randomly divide data into 5 equal sized chunks.
2. for each chunk *i* from 1 to 5
    1. Let chunk *i* be the test set.
    2. Let remaining chunks be the training set.
    3. Train model on the training set.
    4. Evaluate performance on test set (chunk *i*).

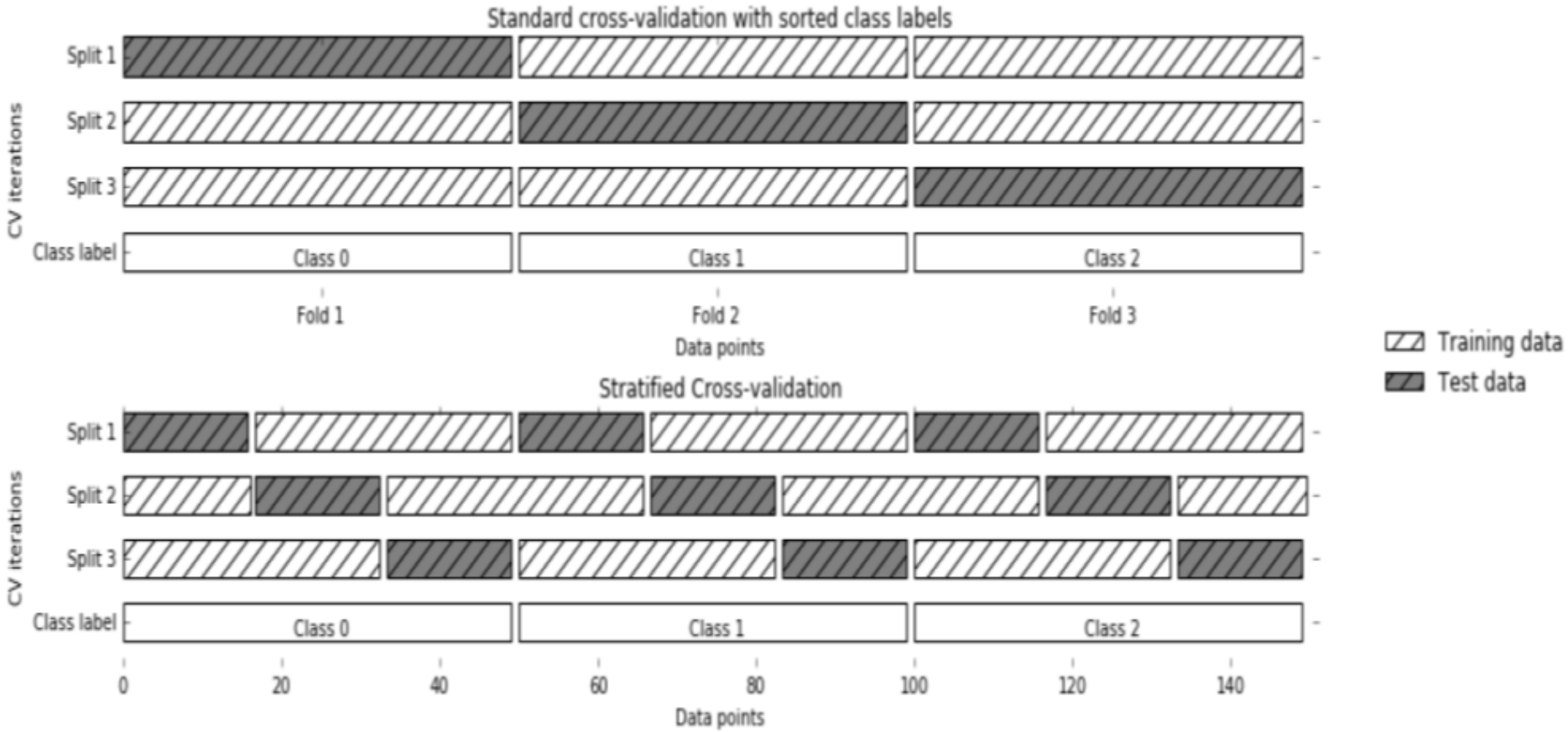


Advantages of Cross Validation

- Model performance isn't rated too high because the test/train split was "lucky" and the hard to classify samples were in the training set.
- Model performance isn't rated too low because the test/train split was "unlucky" and the hard to classify samples were in the training set.
- Multiple splits allow us to see how sensitive the model is to the selection of the training dataset.
- But it takes k times as much computation time to do cross validation.

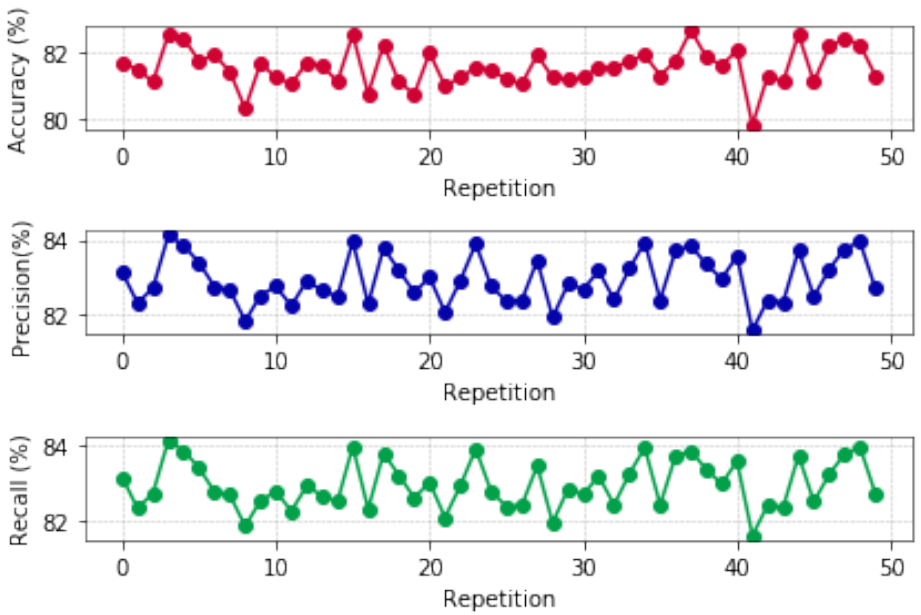
Stratified Cross Validation

Stratified cross-validation ensures label proportions are the same in each fold. Important for datasets with unbalanced classes.



Repeated Cross Validation

Different random splits of a dataset into training and test sets result in differently performing models. Repeating cross-validation multiple times and computing performance metrics as an average over all models produces a more robust measure of model performance.



Iris Dataset

```
In [1]: from sklearn.datasets import load_iris
In [2]: from sklearn.linear_model import LogisticRegression
In [3]: from sklearn.model_selection import StratifiedKFold
In [4]: from sklearn.metrics import accuracy_score
In [5]: iris = load_iris()
In [6]: X = iris.data
In [7]: y = iris.target
```

Iris Setosa

Iris Versicolor

Iris Virginica

Stratified Cross Validation

Folds vary from 90% to 100% accuracy, with mean of 95.3%.

```
In [8]: cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
In [9]: for train, test in cv.split(X, y):
    ...:     model = LogisticRegression()
    ...:     model.fit(X[train], y[train])
    ...:     y_pred = model.predict(X[test])
    ...:     print(round(accuracy_score(y[test], y_pred), 3))
0.933
1.0
0.967
0.9
0.967
In [10]: round(np.mean([0.933,1.0,0.967,0.9,0.967]),3)
Out[10]: 0.953
```

References

1. Clarence Chio and David Freeman, *Machine Learning and Security: Protecting Systems with Data and Algorithms*, O'Reilly Media, 2018.

2. Aurélien Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, 2017.

3. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer. 2014.

4. Andreas C Müller, Sarah Guido, et. Al, *Introduction to Machine Learning with Python: a Guide for Data Scientists*, O'Reilly Media, 2016.

5. Joshua Saxe and Hillary Sanders, *Malware Data Science: Attack Detection and Attribution*, No Starch Press, 2018.