Lecture 4 - Module 1.3 Model evaluation (part 2)

COMP 551 Applied machine learning

Yue Li
Assistant Professor
School of Computer Science
McGill University

January 16, 2025

Outline

Learning objectives

Understanding the following concepts

- Cross-validation
- Method comparison
- Precision-recall curve

Outline

Objectives

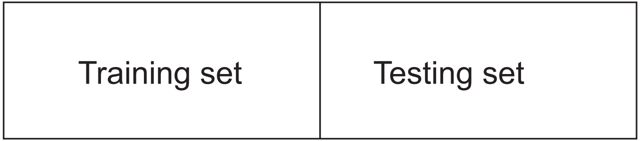
Cross-validation

Method comparisons

Precision-recall and F1-score

K-fold Cross Validation

- In the example from the last lecture, we split the data into training and testing
- Suppose the split is half-half, we train the model using only half of the data and evaluate the model using the other half:



- This is quite wasteful. How can we evaluate our model on *every data point* while training on the rest of the data points?
- Answer: K-fold cross-validation

Five-fold cross validation

Step 1. Randomly split the data $\mathcal{D}$ into 5 folds

$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

Training

Step 2. Training and prediction

Fold 1	$\mathcal{F}1$					Train on $\mathcal{D}$ - $\mathcal{F}1$, predict on $\mathcal{F}1$
Fold 2		$\mathcal{F}2$				Train on $\mathcal{D}$ - $\mathcal{F}2$, predict on $\mathcal{F}2$
Fold 3			$\mathcal{F}3$			Train on $\mathcal{D}$ - $\mathcal{F}3$, predict on $\mathcal{F}3$
Fold 4				$\mathcal{F}4$		Train on $\mathcal{D}$ - $\mathcal{F}4$, predict on $\mathcal{F}4$
Fold 5					$\mathcal{F}5$	Train on $\mathcal{D}$ - $\mathcal{F}5$, predict on $\mathcal{F}5$

testing

- How many times each data point is trained?
- Answer: 4 times
- How many times each data point is predicted?
- Answer: 1

* only 1 data pt used for testing

Evaluate on all K folds

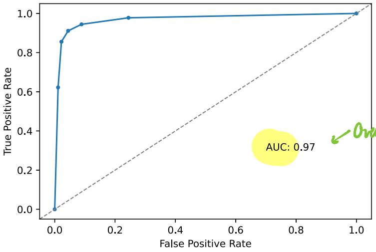
Step 3. Evaluate predictions on all 5 folds by ROC

Predicted probabilities	$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

versus

True labels	$\mathcal{F}1$	$\mathcal{F}2$	$\mathcal{F}3$	$\mathcal{F}4$	$\mathcal{F}5$

ROC curve of KNN predicted on ALL data points



Overall performance

Cross validation in Python scikit-learn (Colab)

```python
def cross_validate(model, X_input, Y_output):
    kf = KFold(n_splits=5, random_state=1, shuffle=True)
    true_labels = np.array([0] * X_input.shape[0])
    pred_scores = np.array([0.0] * X_input.shape[0])
    for train_index, test_index in kf.split(X_input):
        model.fit(X_input[train_index], Y_output[train_index])
        true_labels[test_index] = Y_output[test_index]
        pred_scores[test_index] =
        ↪  model.predict_proba(X_input[test_index])[:,1]
    return true_labels, pred_scores
true_labels,pred_scores = cross_validate(model, X, y)
```

↳ take second column and store in pred-scores

Outline

Method comparisons

- There are many machine learning methods implemented in scikit-learn
- How do we know which one performs the best on *our data set*?
- To get the answer, we will need to compare these methods using cross validation
- Let's compare three machine learning methods namely
  - K-nearest neighbours (KNN)
  - Decision tree classifier (DT) (Module 3)
  - Logistic regression (LR) (Module 4.2)
- Note: for each method (or class), we create an *object* of the method using their initializer method defined under that class
- Training and prediction follows the *generic* syntax

Method comparisons using scikit-learn (Colab)

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

models = [LogisticRegression(),
          KNeighborsClassifier(),
          DecisionTreeClassifier()]

perf = {}

for model in models:
    model_name = type(model).__name__
    print(model_name)
    label,pred = cross_validate(model, X, y)
    fpr, tpr, thresholds = roc_curve(label, pred)
    auc = roc_auc_score(label, pred)
    perf[model_name] = {'fpr':fpr,'tpr':tpr,'auc':auc}
```

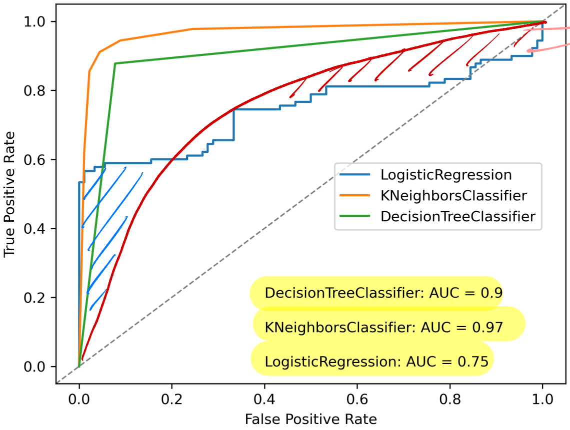
Plot the ROC curves for all method in one plot

```
import matplotlib.pyplot as plt

i = 0
for model_name, model_perf in perf.items():
    plt.plot(model_perf['fpr'], model_perf['tpr'], label=model_name)
    plt.text(0.4, i, model_name + ': AUC = '+
                str(round(model_perf['auc'],2)))
    i += 0.1

plt.legend(loc='upper center',
              bbox_to_anchor=(0.75, 0.5))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

plt.savefig('roc_multimethods.eps')
```

make the model_name are on diff lines

ROC curves and AUC for all of the four methods

- KNN (K=5) performs the best with 0.97 AUC
- DT achieves 0.85 AUC
- LR did worse (AUC = 0.73) because our data are not linearly separable
- In contrast, DT and KNN are non-linear methods



When you have the same Area under ROC?

→ You care more about better TPR among top predictions

→ >

↳ Precision ↑

① KN
② Decis.
} Non-linear, so perf higher

③ Log. →

The current data is non-linearly separable

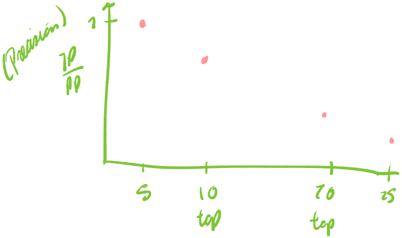
Outline

Objectives

Cross-validation

Method comparisons

Precision-recall and F1-score



⇒ Top K precision curve decreases

Sensitivity/Recall, Specificity, and Precision

Sensitivity or Recall: Proportion of true positive example among ALL positive (P)

$$TPR = Sensitivity = Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \quad (1)$$

Precision: Proportion of true positive example among **the predicted positive** (PP)

$$Precision = \frac{TP}{PP} \quad (2)$$

→ Important when you have limited tests
↳ and you want the top

F1-score: F1 = 2 × (precision × recall) / (precision + recall)

Precision is very important in many circumstances, e.g.,

- We can only afford testing 5 drugs among 100 predicted drugs
- We can admit a small number of high-risk patients among all patients

```python
from sklearn.metrics import precision_recall_curve
precision, recall, thres = precision_recall_curve(label, pred)
auprc = auc(recall, precision)
```

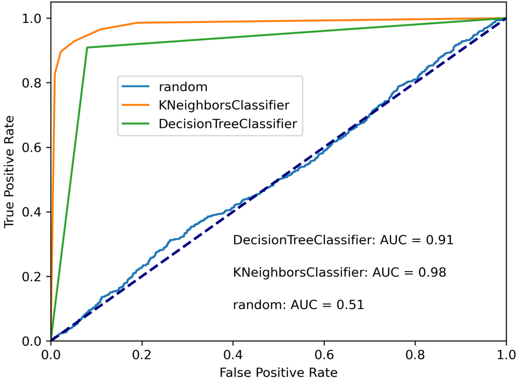
Constructing ROC and PR curve by varying the thresholds

TPR	FPR	Threshold
0.0	0.000000	1.999820
0.0	0.001111	0.999820
0.0	0.020000	0.984463
...	...	...
0.3	0.067778	0.936420
0.3	0.090000	0.918972
0.4	0.090000	0.918953
0.4	0.291111	0.719385
0.5	0.291111	0.717308
...	...	...
0.9	0.946667	0.058096
1.0	0.946667	0.056398
1.0	1.000000	0.000270

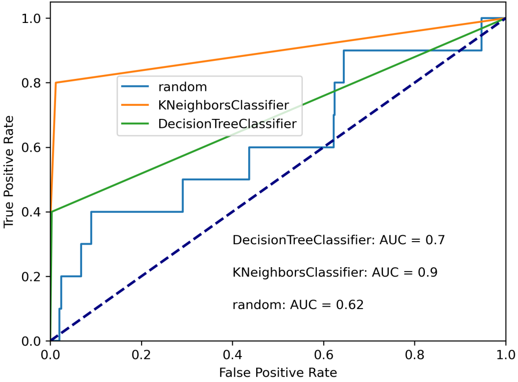
Precision	Recall	Threshold
0.010216	0.9	0.037235
0.010227	0.9	0.037284
0.010239	0.9	0.038246
...	...	...
0.010870	0.8	0.206335
0.010884	0.8	0.206341
...	...	...
0.010870	0.7	0.288363
...	...	...
1.000000	0.0	0.999557

ROC is designed for class-balanced data (Colab)

When we have 50% positive and 50% negative labels, a line that goes along the diagonal indicates random guess (P=900,N=900).

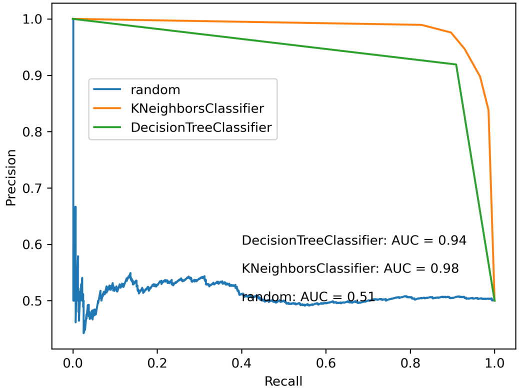


However, suppose we have 1% positive and 99% negative labels, random prediction will no longer follow the diagonal line (P=10,N=900).

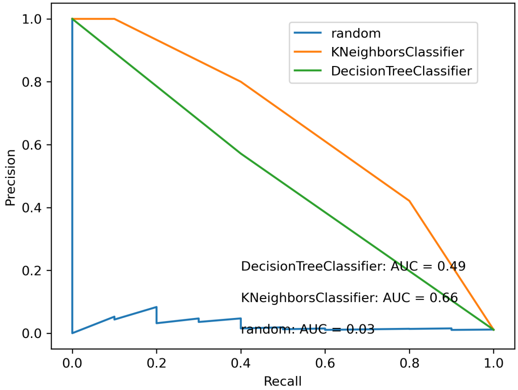


Precision-recall curve is a better choice for imbalanced data (Colab)

When we have 50% positive and 50% negative labels (P=900,N=900).



When we have 1% positive and 99% negative labels (P=10,N=900).



Summary

- Approximate generalization performance using test data
- ROC is an effective way to test overall model performance using all thresholds
- Cross-validation makes use of the full data for both training and evaluation
- Generic model implementation in Scikit-learn enables efficient method comparison
- Precision-recall is an alternative metric to ROC and it is better suited to measure performance on imbalanced data and circumstance where precision is important.