Lecture 10: Model Evaluation

(textbook chapter 6)

Dr. Huiping Cao

Goals

- Obtain unbiased estimates of a model's performance
- Evaluate predictive models using different performance metrics

Cross validation

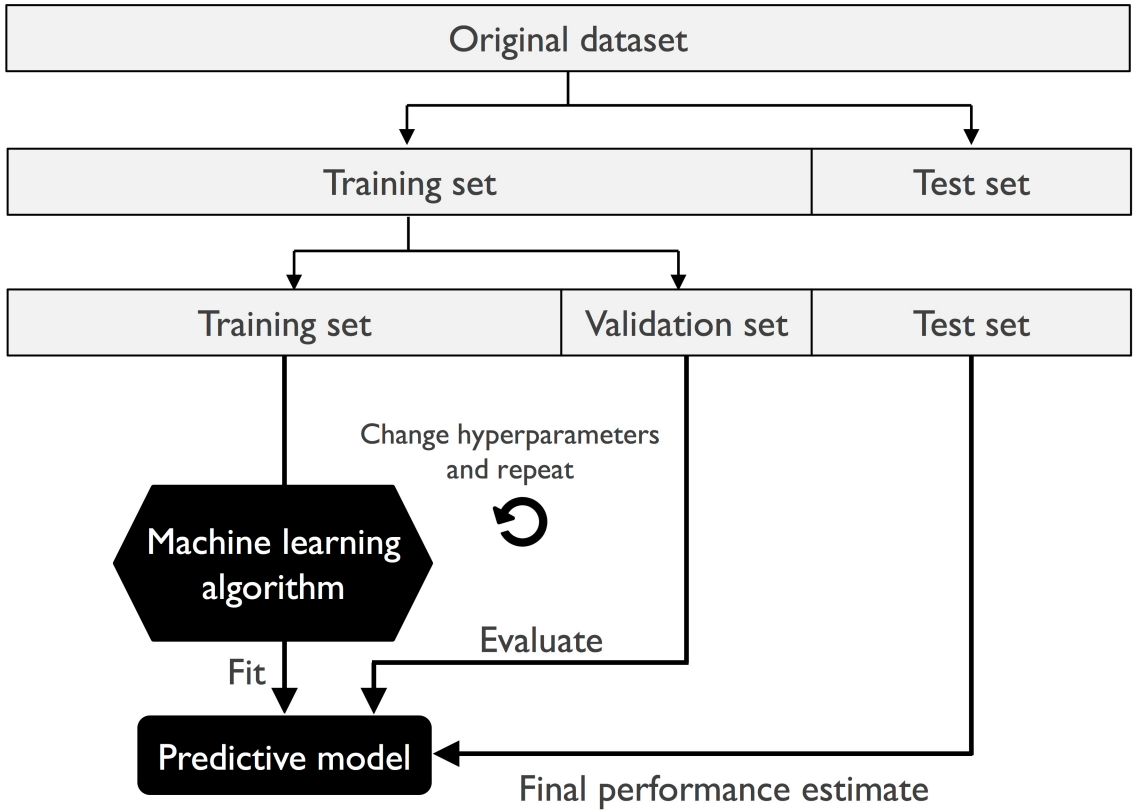
- **Generalization performance**: one of the key steps in building a machine learning model is to estimate its generalization performance (i.e., performance on data that the model has not seen before).
- A model can suffer from **underfitting** (high bias) if the model is too simple, or it can **overfit** the training data (high variance) if the model is too complex .
- It is critical to find an acceptable **bias-variance tradeoff** through careful model evaluation**.**
- **Cross-validation techniques** to obtain reliable estimates of the model's generalization performance (or how well a model performs on unseen data).

Model selection

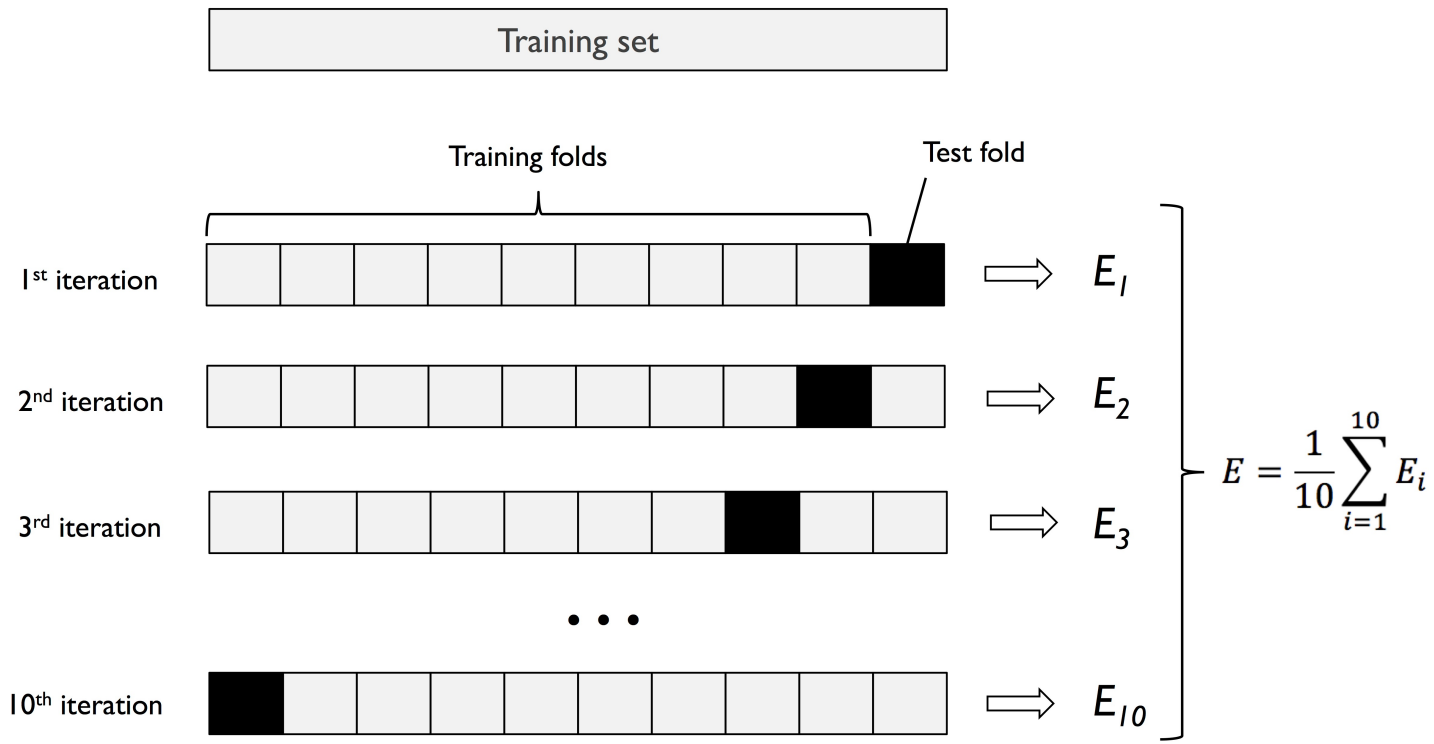
- **Model selection**: tune and compare different parameters (hyperparameters) and find their optimal values for a model.
  - Hyperparameter examples: learning rate in perceptron and Adaline, C and γ in SVM
- **Supervised learning**: split our initial dataset into separate training and test datasets
- **Issue** with using the same test data to conduct model selection: the model may tend to overfit.

Holdout cross-validation

- Split the data to training, validation, and test dataset.
- **Training set**: fit different models
- **Validation set**: model selection using different hyperparameter values
- **Test set**: estimate the model's generalization performance.
- **Advantage**: having a test dataset that the model hasn't seen before during the training and model selection can help us obtain a less biased estimate of its ability to generalize to new data.
- **Disadvantage**: the performance estimate may be very sensitive to how we partition the training dataset into the three subsets.



K-fold cross-validation



- **Basic idea**: repeat holdout methods on subsets of training data
- The training set is divided into *k* folds.
- During *k* iterations, *k* − 1 folds are used for training and one fold is used as the test set for the model evaluation.
- The estimated performances E_i from all the iterations are used to calculate the averaged performance of the model.
- Typical *k* is set to 10 (empirical evidence suggests this).
- When a dataset is small, typically we use a larger *k*. More training data is used in each iteration. Larger running time.
- When a dataset is very big, we can use a smaller *k*.

K-fold cross-validation

- **Leave-one-out cross-validation (LOOCV):** the number of folds equal to the number of training examples (i.e., k=n)
  - One training example is used to testing in each iteration.
  - For very small datasets.
- **Stratified K-fold cross-validation**: the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset.
  - Better for datasets with unequal class proportions.
- Code example

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=10).split(X_train, y_train)
```

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=10).split(X_train, y_train)

for k, (train, test) in enumerate(kfold):
    print("K", k, "train:", train[:10], "test", test[:10])

    c1 = mysvm(X_train[train], y_train[train], other parameters)      # Create your model
    c1.fit                                                            # Train your model
    score = c1.predict                                                # Make predictions using your model
```

Pseudo code

```
K 0 train: [45 46 47 48 50 51 52 53 54 55] test [0 1 2 3 4 5 6 7 8 9]
K 1 train: [0 1 2 3 4 5 6 7 8 9] test [45 46 47 48 50 51 52 53 54 55]
K 2 train: [0 1 2 3 4 5 6 7 8 9] test [85 87 88 89 90 93 94 95 97 98]
K 3 train: [0 1 2 3 4 5 6 7 8 9] test [132 133 134 135 136 139 140 141 143 145]
K 4 train: [0 1 2 3 4 5 6 7 8 9] test [177 178 182 183 184 185 188 189 191 193]
K 5 train: [0 1 2 3 4 5 6 7 8 9] test [227 229 230 231 233 234 235 237 238 239]
K 6 train: [0 1 2 3 4 5 6 7 8 9] test [271 272 273 274 275 276 277 279 280 284]
K 7 train: [0 1 2 3 4 5 6 7 8 9] test [315 316 317 318 320 322 326 327 328 329]
K 8 train: [0 1 2 3 4 5 6 7 8 9] test [364 366 367 368 369 370 371 372 373 374]
K 9 train: [0 1 2 3 4 5 6 7 8 9] test [409 411 412 413 414 415 416 417 418 419]
```

Pipelines

```
for k, (train, test) in enumerate(kfold):
    print("K", k, "train:", train[:10], "test", test[:10])

    c1 = mysvm(X_train[train], y_train[train], other parameters)      # Create your model
    c1.fit                                                            # Train your model
    score = c1.predict                                                # Make predictions using your model
```

```
from sklearn.svm import SVC

pipe_svc = make_pipeline(StandardScaler(), SVC(random_state=1,probability=True))
                                                                      # Create the model

pipe_svc.fit(X_train[train], y_train[train])                          # Train the model

y_pred = pipe_svc.predict(X_test[test])                               # Get predictions
score = pipe_svc.score(X_train[test], y_train[test])                  # Get prediction score
```

Performance evaluation metrics

- Misclassification error
- Classification accuracy

- Others?

Evaluation metric – confusion matrix

- Confusion matrix

		Predicted class	
		Class 1 (positive)	Class 0 (negative)
Actual class	Class 1 (positive)	f_{11} (TP)	f_{10} (FN)
	Class 0 (negative)	f_{01}(FP)	f_{00} (TN)

- f_{11}: True Positive (TP)
- f_{01}: False Positive (FP)
- f_{10}: False Negative (FN)
- f_{00}: True Negative (TN)

Evaluation metric – confusion matrix

- confusion_matrix function

```
from sklearn.metrics import confusion_matrix

pipe_svc.fit(X_train, y_train)
y_pred = pipe_svc.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

```
[[71 1]
 [ 2 40]]
```

Evaluation metric – accuracy and error

- Desirable classifier: high accuracy, low error rate

		Predicted class	
		Class 1 (positive)	Class 0 (negative)
Actual class	Class 1 (positive)	f_{11} (TP)	f_{10} (FN)
	Class 0 (negative)	f_{01}(FP)	f_{00} (TN)

$$Accuracy = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$Error\ rate = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

Limitations of accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If a model predicts everything to be class 0, accuracy is 9990/10000 = 99.9%
- Accuracy is misleading because the model does not detect any class 1 example
- Existence of data sets with imbalanced class distributions. E.g., defective products and non-defective products.
- Analyzing imbalanced data sets, where the rare class is considered more interesting than the majority class. Binary classification, the rare class is denoted as the positive class.
- The accuracy measure may not be well suited for evaluating models derived from imbalanced data sets.

Precision, Recall, F_1

- Desirable classifier: high precision, high recall, high F_1

		Predicted class	
		Class 1 (positive)	Class 0 (negative)
Actual class	Class 1 (positive)	f_{11} (TP)	f_{10} (FN)
	Class 0 (negative)	f_{01} (FP)	f_{00} (TN)

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

$$Recall = r = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN} = \frac{2}{\frac{1}{r} + \frac{1}{p}}$$

Code - Precision, Recall, F_1

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

print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
Precision: 0.976
Recall: 0.952
F1: 0.964
```

Alternative metrics

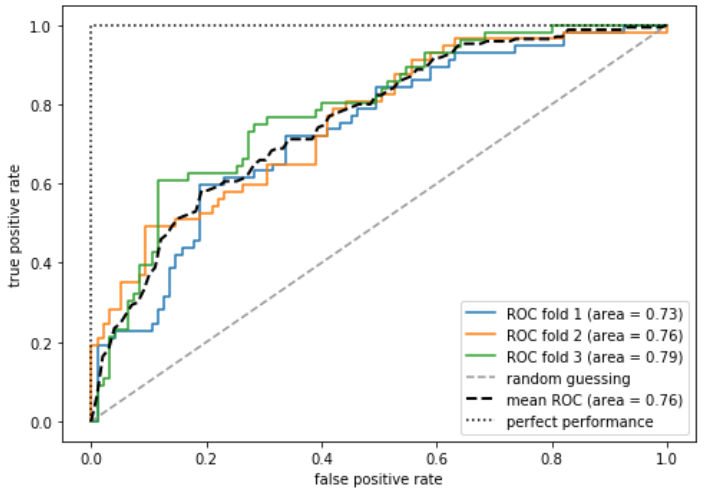
- **True positive rate (TPR)**, also called recall or **sensitivity**: $TPR = \frac{TP}{TP + FN}$
- True negative rate, or **specificity**: $TNR = SPC = \frac{TN}{N} = \frac{TN}{TN + FP}$
- **False positive rate (FPR)**: $FPR = \frac{FP}{FP + TN} = 1 - SPC$
- False negative rate: $FNR = \frac{FN}{TP + FN}$

ROC (Receiver Operating Characteristic)

- Developed for signal detection theory
- Characterize the trade-off between true positive rate (TPR) and false positive rate (FPR)
- ROC curve plots TPR (on the y-axis) against FPR (on the x-axis)
- The performance of each possible prediction criterion is represented as a point on the ROC curve

ROC properties

- TPR=0, FPR=0: every instance is predicted to be a negative class
- TPR=1, FPR=1: every instance is predicted to be a positive class
- TPR=1, FPR=0: the ideal model
- Diagonal line: random guessing
- A good classification model should be located as close as possible to the upper-left corner of the diagram.



Construct a ROC curve

- Sort the instances according to $P(+|A)$ in decreasing order
- Apply threshold at each unique value of $P(+|A)$
- Count the number of TP, FP, TN, FN at each threshold δ
  - Assign the selected instances with $p \geq \delta$ to be positive class.
  - Assign those instances with with $p < \delta$ as negative class

Instance	P(+|instance)	True class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.86	+
5	0.85	-
6	0.84	-
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

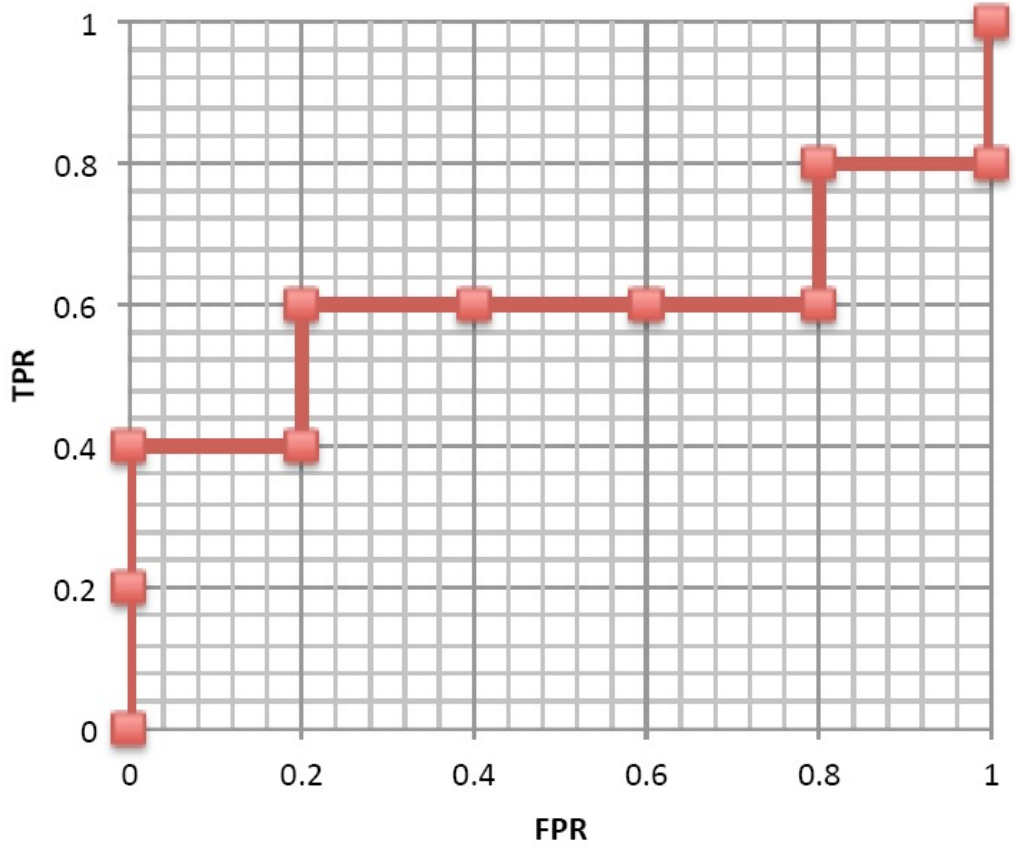
Example

id	10	9	8	7	6	5	4	3	2	1	
class	+	-	+	-	-	-	+	-	+	+	
	0.25	0.43	0.53	0.76	0.84	0.85	0.86	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

Recall $TPR = \frac{TP}{TP+FN}$, $FPR = \frac{FP}{FP+TN}$

Example

TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0



Appendix - construct a ROC curve - Tie

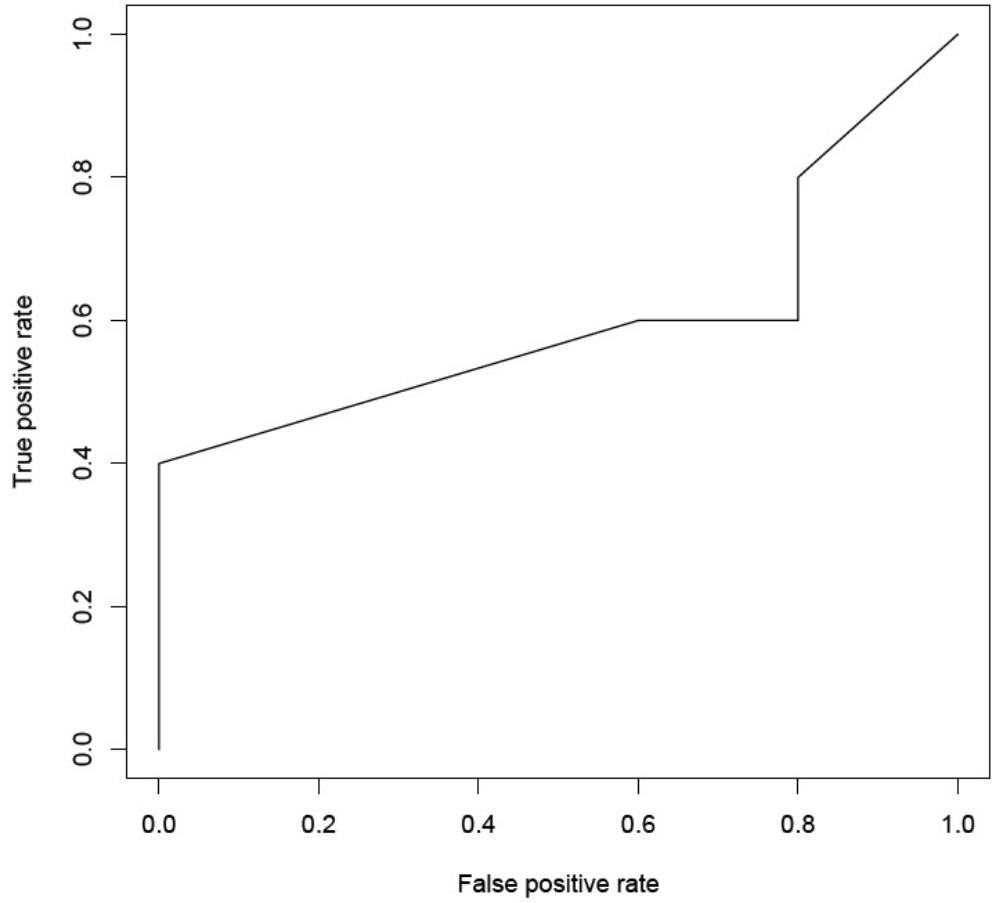
- ROC curve when some probability values are duplicated?
- Use only distinct probabilities

Recall $TPR = \frac{TP}{TP+FN}$, $FPR = \frac{FP}{FP+TN}$

class	+	-	+	-	+	-	-	-	+	+	
	0	0	0.7	0.76	0.85	0.85	0.85	0.85	0.95	0.95	1.00
Boundary	0		0.7	0.76	0.85				0.95		
TP	5		4	3	3				2		0
FP	5		4	4	3				0		0
TN	0		1	1	2				5		5
FN	0		1	2	2				3		5
TPR	1		0.8	0.6	0.6				0.4		0
FPR	1		0.8	0.8	0.6				0		0

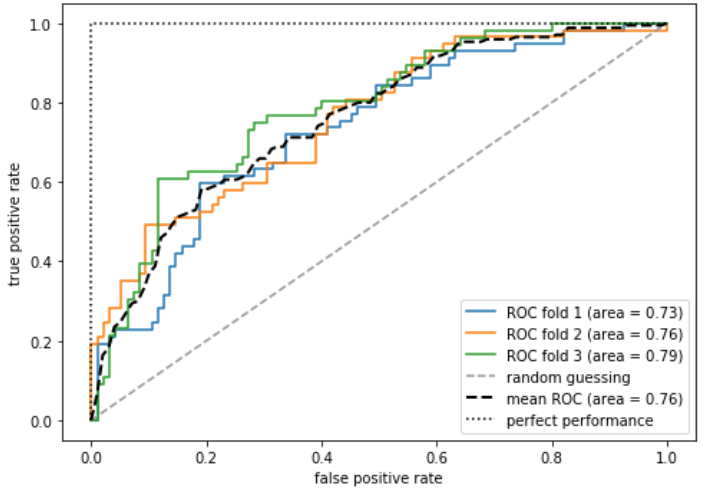
Example - tie

TPR	1	0.8	0.6	0.6	0.4	0
FPR	1	0.8	0.8	0.6	0	0



AUC (Area under the ROC Curve)

- AUC can evaluate which model is better on average.
  - AUC = 1: the model is perfect
  - AUC = 0.5: random guess



ROC and AUC calculation

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

X_train2 = X_train[:, [4, 14]]
kfold = StratifiedKFold(n_splits=3).split(X_train, y_train)

for k, (train, test) in enumerate(kfold):
    pipe_svc.fit(X_train2[train],y_train[train])
    probas = pipe_svc.predict_proba(X_train2[test])

    fpr, tpr, thresholds = roc_curve(y_train[test],
                                     probas[:, 1],
                                     pos_label=1)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr,
            label='ROC fold %d (area = %0.2f)'
                % (k, roc_auc))
    print("auc=", roc_auc)
```

```
auc= 0.7268698060941828
auc= 0.7488457987072946
auc= 0.774248120300752
```