Principles of Machine Learning

Lecture 7: Model Evaluation and Validation

Sharif University of Technology
Dept. of Aerospace Engineering

April 20, 2025

Table of Contents

Table of Contents

Model Evaluation

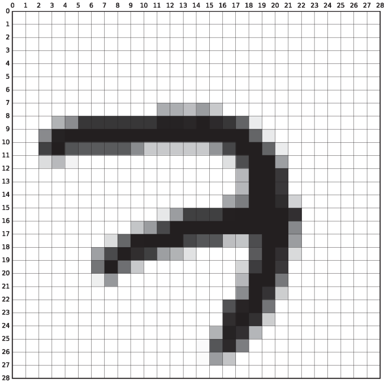
- Some performance measures/metrics $\longrightarrow$ effectiveness of the model
- For regression tasks:
  - **M**ean **A**bsolute **E**rror
  - **M**ean **S**quared **E**rror
  - **R**oot **M**ean **S**quared **E**rror
  - R-squared (R^2)
- For classification tasks:
  - ???
  - $\vdots$
  - ???

MNIST Dataset

Before continuing on evaluating classifiers, let's get to know about the MNIST dataset, the "Hello World!" of the AI/ML!

- Modified National Institute of Standards and Technology dataset was created in 1998
- A set of $70,000$ small images of digits handwritten by high school students and employees of the US Census Bureau
- Each image is labeled with the digit it represents and has 784 features (pixels)

MNIST Dataset



(**a**) MNIST sample belonging to the digit '7'.

(**b**) 100 samples from the MNIST training set.

Table of Contents

Accuracy

- The most basic performance measure
- Defined as *the ratio of correct predictions*

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{All Predictions}}$$

Example 1. Implementing a classifier for MNIST

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', as_frame=False)

X, y = mnist.data, mnist.target
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

y_train_5 = (y_train == '5') # True for all 5s, False for all other digits
y_test_5 = (y_test == '5')
```

Example 1 contd. Implementing a classifier for MNIST

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)

>>> sgd_clf.predict([some_digit])
array([ True])
```

Performance Measures – Accuracy using Cross Validation

Example 1 contd. Implementing a classifier for MNIST

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.95035, 0.96035, 0.9604 ])

from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train_5)
print(any(dummy_clf.predict(X_train))) # prints False: no 5s detected

>>> cross_val_score(dummy_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.90965, 0.90965, 0.90965])
```

Cross Validation

- A model validation technique
- Splitting to training data and validation data
- Better generalization $\rightarrow$ reduce the chance of overfitting

k-**fold Cross-validation**

- Splitting the training set into k folds
- Training the model k times
- Holding out a different fold each time for evaluation

Performance Measures – Confusion Matrix

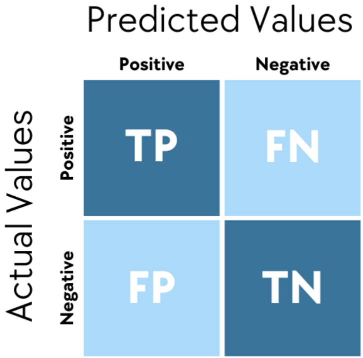
Why not "accuracy"?

- From Example 1. $\longrightarrow$ *accuracy* is usually not a good performance measure for classifiers
- Especially for *skewed* datasets (i.e., some classes are more frequent than others)

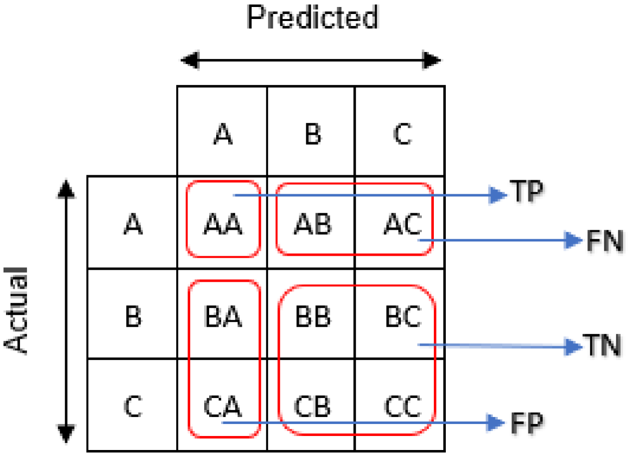
Confusion Matrices

- General idea: count the number of times instances of class A are classified as class B, for all A/B pairs
- First need to have a set of predictions so that they can be compared to the actual targets
- Could make predictions on the test set, but it's best to keep that untouched for now
- Each row: an actual class
- Each column: a predicted class

A Confusion MatriX



A 3 $\times$ 3 Confusion Matrix example



Performance Measures – Confusion Matrix

Implementing CM for Example 1.

```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)

>>> from sklearn.metrics import confusion_matrix
>>> cm = confusion_matrix(y_train_5, y_train_pred)
>>> cm
array([[53892, 687],
[ 1891, 3530]]

>>> y_train_perfect_predictions = y_train_5 # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579, 0],
[ 0, 5421]])

```

Performance Measures – Confusion Matrix

The confusion matrix gives useful metrics:

- **Precision**: the accuracy of the positive predictions

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

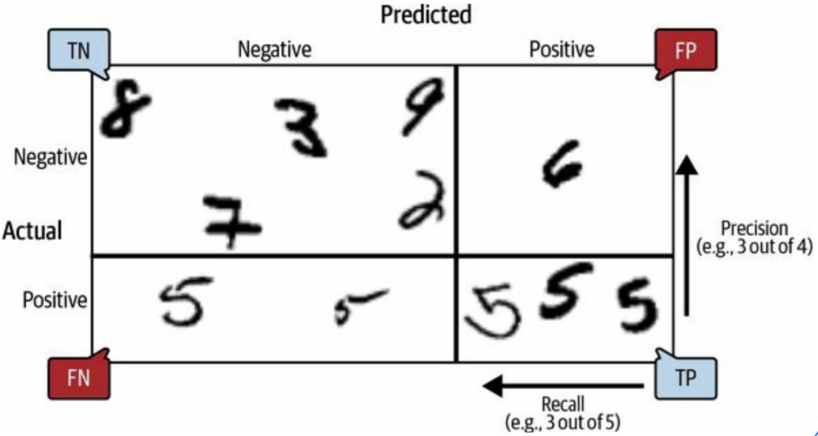
- **Recall**, sensitivity or the true positive rate (TPR): the ratio of positive instances that are correctly detected by the classifier

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

- TP: # true positives
- FP: # false positives
- TN: # true negatives
- FN: # false negatives

Performance Measures – Confusion Matrix

The confusion matrix for the example 1 (predicting the digit 5)



Performance Measures – Confusion Matrix

Implementing Precision, Recall, & F-1 for Example 1.

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 3530 / (687 + 3530)
0.8370879772350012
>>> recall_score(y_train_5, y_train_pred) # == 3530 / (1891 + 3530)
0.6511713705958311

>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7325171197343846
```

F-1 Score

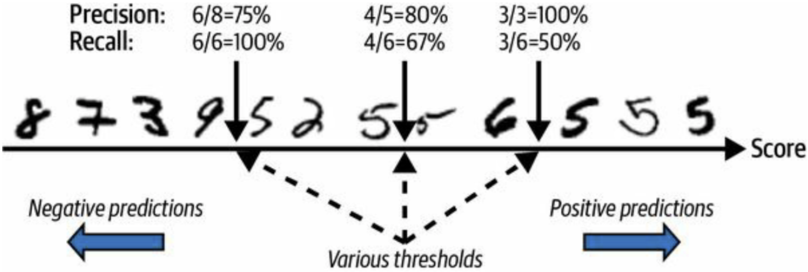
- A combination of precision and recall
- Defined as the harmonic mean of the precision and recall

$$\text{F-1} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Useful especially when comparing two classifiers
- Both precision and recall high $\longrightarrow$ high F-1 score

Performance Measures – The Precision/Recall Trade-off

The precision/recall trade-off is carried out by a *decision function* that determines a *threshold*:



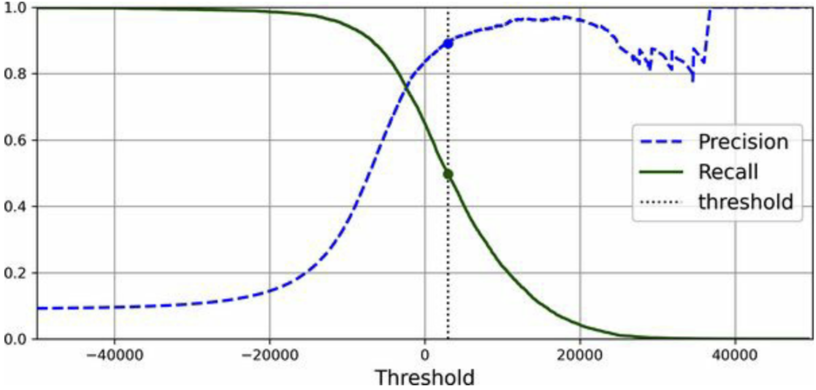
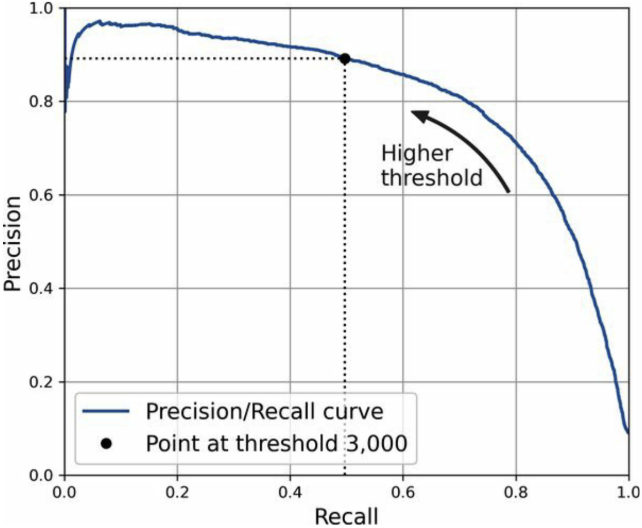


Figure: Precision and recall versus the decision threshold

Performance Measures – The Precision/Recall Trade-off

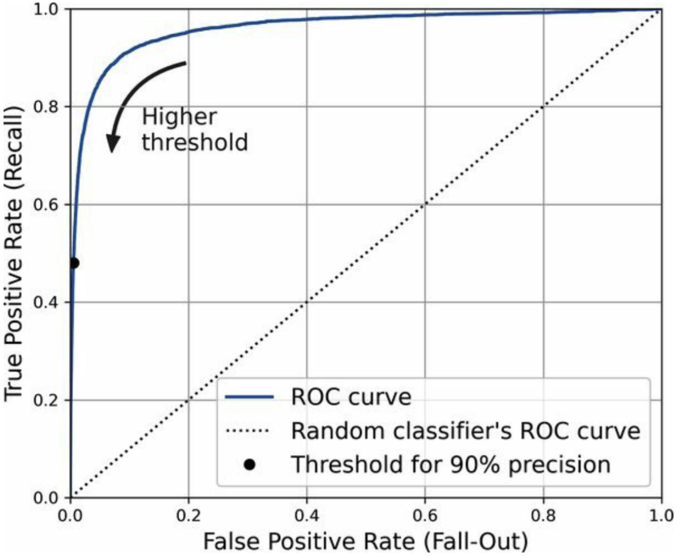
Another way of selecting a good precision/recall trade-off:



The receiver operating characteristic (ROC) curve

- A common evaluation tool for binary classifiers
- Plots TPR (true positive rate) against FPR (false positive rate)
- TPR = recall: *sensitivity*, FPR = 1 - TNR: 1 - *specificity*

The ROC curve



For **The ROC curve**

- A good classifier: stay toward the top-left corner
- To compare two classifiers: measure the *area under the curve* (AUC)
  - Perfect: AUC=1
  - A purely random classifier: AUC=0.5

Performance Measures – The Precision/Recall Trade-off

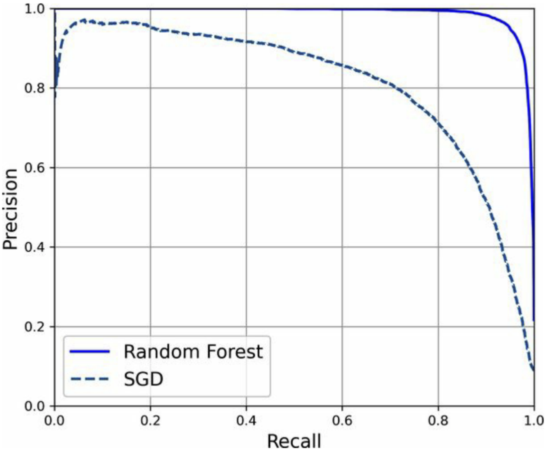


Figure: Comparing PR curves: the random forest classifier is superior to the SGD classifier because its PR curve is much closer to the top-right corner, and it has a greater AUC

Multiclass Classification

- distinguish between more than two classes
- multiclass classifiers:
  `LogisticRegression`, `RandomForestClassifier`, `GaussianNBanNB`
- strictly binary classifiers: `SGDClassifier`, `SVC`

Performance Measures

To perform multiclass classification with multiple binary classifiers

1. **one-versus-the-rest (OvR)** or **one-versus-all (OvA)** strategy

   Example. classify the digit images into 10 classes (from 0 to 9)

   - train 10 binary classifiers, one for each digit
     - a 0-detector
     - a 1-detector
     - a 2-detector
     - ⋮
   - get the decision score from each classifier for that image
   - select the class whose classifier outputs the highest score

To perform multiclass classification with multiple binary classifiers

2. **one-versus-one (OvO)**

   Example. classify the digit images into 10 classes (from 0 to 9)

   - train a binary classifier for every pair of digits
     - one to distinguish 0s and 1s
     - another to distinguish 0s and 2s
     - another for 1s and 2s
     - $\vdots$
   - For N classes, train $\frac{N(N-1)}{2}$ classifiers
     - For the MNIST problem, this means training 45 binary classifiers
     - run the image through all 45 classifiers
     - see which class wins the most duels

   - Main advantage: each classifier only needs to be trained on part of the training set containing the two classes that it must distinguish

OvR vs. OvO

- Some algorithms (like SVM classifiers) scale poorly with the size of the training set $\longrightarrow$ **OvO** $\longrightarrow$ faster to train many classifiers on small training sets than to train few classifiers on large training sets
- For most binary classification algorithms $\longrightarrow$ **OvR**

• Note that
`scikit-learn` automatically detects when you try to use a binary classification algorithm for a multiclass classification task
and
automatically runs OvR or OvO depending on the algorithm

Performance Measures

Multilabel Classification

- outputs multiple binary tags
  - face-recognition classifier: multiple faces in the same picture
  - one tag per person
- multiple classes for each instance

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)

```

Evaluating a Multilabel Classifier

(Note that selecting the right metric really depends on the project)

1. Measure F_1 score for each individual label (or any other binary classifier metric)
    - then compute the average score
    - assumes that labels are equally important (have same weight)
2. For a non-natively multilabel classifier (such as SVC): train one model per label
    - cannot capture the dependencies between the labels
        - to solve the above issue: oraganize the models in a chain
        - `ClassifierChain`