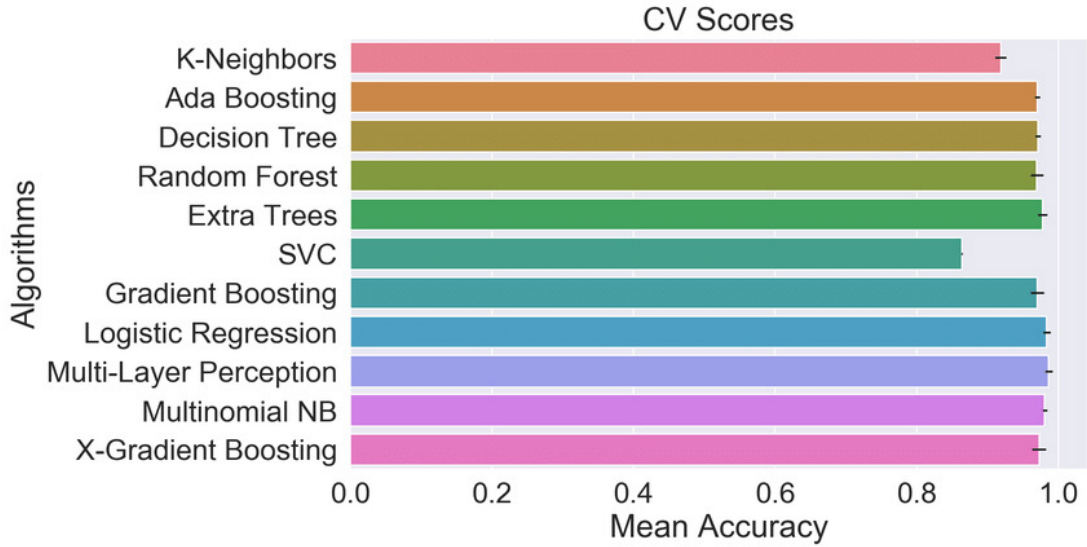
Lecture 15
Evaluating Classification Models

Dennis Sun (modified by Jonathan Taylor)
Stanford University

DATASCI 112

May 8, 2026



Source: "A robust system for message filtering using an ensemble machine learning supervised approach"

Case Study: Credit Card Fraud

Data set of credit card transactions from Vesta.

```
import pandas as pd

df_fraud = pd.read_csv("|\courseurl|data/fraud.csv")
df_fraud
```

	card4	card6	P_emaildomain	TransactionAmt	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	isFraud
0	visa	debit	gmail.com	62.950	139.0	110.0	0.0	0.0	135.0	93.0	0.0	0.0	93.0	0.0	93.0	0.0	637.0	114.0	0
1	visa	debit		35.950	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	3.0	1.0	0
2	visa	debit	yahoo.com	117.000	1.0	1.0	0.0	0.0	0.0	2.0	0.0	0.0	1.0	0.0	1.0	0.0	4.0	1.0	1
3	visa	debit	hotmail.com	54.500	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0
4	visa	debit	gmail.com	255.000	1.0	3.0	0.0	0.0	0.0	4.0	0.0	0.0	2.0	0.0	3.0	1.0	20.0	1.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59049	mastercard	debit	gmail.com	20.522	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0
59050	mastercard	credit	yahoo.com	50.000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0
59051	mastercard	debit	icloud.com	97.950	2.0	2.0	0.0	0.0	1.0	2.0	0.0	0.0	1.0	0.0	2.0	0.0	15.0	2.0	0
59052	visa	debit	gmail.com	16.723	1.0	2.0	0.0	1.0	0.0	1.0	1.0	2.0	0.0	1.0	1.0	1.0	1.0	1.0	1
59053	mastercard	debit		107.950	24.0	21.0	0.0	0.0	14.0	13.0	0.0	0.0	17.0	0.0	18.0	0.0	84.0	17.0	0

59054 rows × 19 columns

Goal: Predict **`isFraud`**, where 1 indicates a fraudulent transaction.

1 Recap

2 Precision and Recall

Classification Model

We can use k-nearest neighbors for classification:

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer

pipeline = make_pipeline(
    make_column_transformer(
        (OneHotEncoder(handle_unknown="ignore", sparse_output=False),
         ["card4", "card6", "P_emaildomain"]),
        remainder="passthrough"),
    StandardScaler(),
    KNeighborsClassifier(n_neighbors=5))
```

Training a Classifier

```
X_train = df_fraud.drop("isFraud", axis="columns")
y_train = df_fraud["isFraud"]
```

```
from sklearn.model_selection import cross_val_score

cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="accuracy",
    cv=10
).mean()
```

```
0.9681647131621484
```

How is the accuracy so high?

A Closer Look

Let's take a closer look at the labels.

```
y_train.value_counts()
```

```
0    56935
1     2119
Name: isFraud, dtype: int64
```

The vast majority of transactions are normal!

If we just predicted that every transaction is normal, the accuracy would be $\frac{56935}{59054} = .964$.

Even though such predictions would be accurate *overall*, it is inaccurate for fraudulent transactions. A good model is "accurate for every class".

1 Recap

2 Precision and Recall

Precision and Recall

We need a score that measures "accuracy for class c".

There are at least two reasonable definitions:

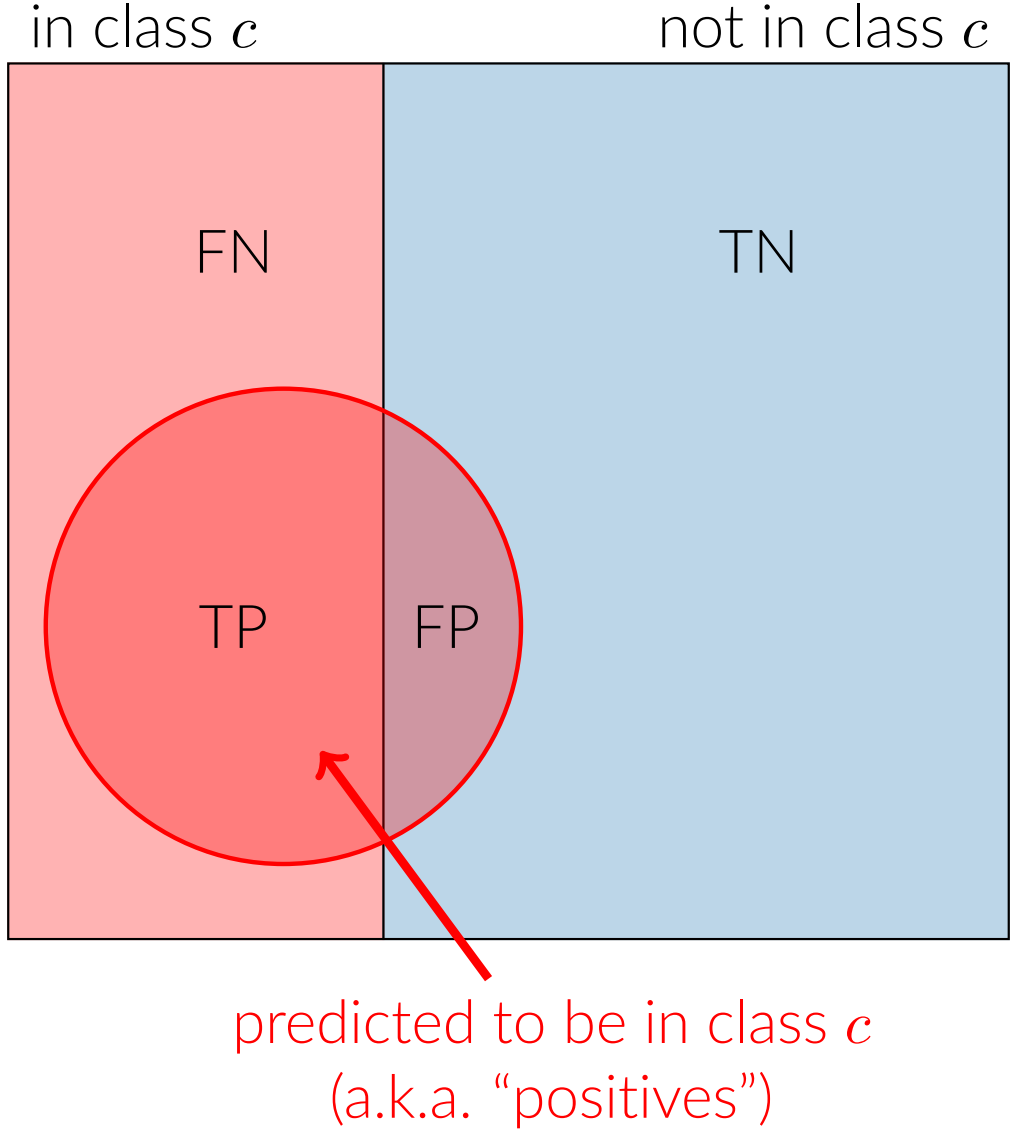
- **precision**: $P(\text{correct}|\text{predicted class } c)$
  Among the observations that were predicted to be in class c, what proportion actually were?

  High precision for fraud means: when a transaction is identified as fraud, there's a high probability it actually is fraud.
- **recall**: $P(\text{correct}|\text{actual class } c)$.
  Among the observations that were actually in class c, what proportion were predicted to be?

  High recall for fraud means: when a transaction is fraudulent, there's a high probability it is caught.

  (This is also called **sensitivity** in epidemiology.)

A Geometric Look at Precision and Recall



$$\textbf{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \textbf{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Precision and Recall by Hand

To check our understanding of these definitions, let's calculate some precisions and recalls by hand.

First, summarize the results by the **confusion matrix**

```
from sklearn.metrics import confusion_matrix
pipeline.fit(X_train, y_train)
y_train_ = pipeline.predict(X_train)
confusion_matrix(y_train, y_train_)
```

```
array([[56817,   118],      ← actually in class 0
       [ 1524,   595]])     ← actually in class 1
```

- What's the precision for fraudulent transactions? 83.5%
- What's the recall for fraudulent transactions? 28.1%
- What's the precision for normal transactions? 97.4%
- What's the recall for normal transactions? 99.8%
- What is the accuracy? 97.2%

Note: Each class has its own precision and recall!

Tradeoff between Precision and Recall

Can you imagine a classifier that always has 100% recall for class c, no matter the data?

In general, if the model classifies more observations as c,

- recall (for class c) ↑
- precision (for class c) ↓

How do we compare two classifiers, if one has higher precision and the other has higher recall?

The **F1 score** combines precision and recall into a single score:

$$\text{F1 score} = \text{harmonic mean of precision and recall}$$

$$= 1 \Big/ \frac{1}{2}\left(\frac{1}{\text{precision}} + \frac{1}{\text{recall}}\right)$$

So the F1 score of the classifier for fraudulent transactions is

$$1 \Big/ \frac{1}{2}\left(\frac{1}{.835} + \frac{1}{.281}\right) \approx 42.0\%.$$

To achieve a high F1 score, both precision and recall have to be high. If either is low, then the harmonic mean will be low.

Estimating Test Precision, Recall, and F1

Remember that each class has its own precision, recall, and F1.

But Scikit-Learn requires that the `scoring=` parameter be a single number.
For this, we can use

- `"precision_macro"`
- `"recall_macro"`
- `"f1_macro"`

which average the score over the classes.

Example:

```
cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="f1_macro",
    cv=10
).mean()
```

```
0.6475574801118277
```

Precision-Recall Curve

Another way to illustrate the tradeoff between precision and recall is to graph the **precision-recall curve**.

First, we need the predicted probabilities.

```
y_train_probs_ = pipeline.predict_proba(X_train)
y_train_probs_
```

```
array([[1. , 0. ],
       [1. , 0. ],
       [0.6, 0.4],
       ...,
       [1. , 0. ],
       [0.8, 0.2],
       [1. , 0. ]])
```

By default, Scikit-Learn classifies a transaction as fraud if this probability is > 0.5.

What if we instead used a threshold t other than 0.5?

Depending on which t we pick, we'll get a different precision and recall. We can graph this tradeoff.

Precision-Recall Curve

Let's graph the precision-recall curve in a Colab.