9. Metrics: Precision and recall
LING 471

Learning outcomes

- Discuss and give examples of **classes** and **type** hierarchies
- Instantiate a class through a **constructor**
- Access **class variables** and **methods**
- Manipulate filenames and paths with the **pathlib** library
- Define and calculate **accuracy**, **precision**, and **recall** for simple examples

Assignment 3

- Posted on class GitHub
- Due May 12 at 11:59 PM
- General goal: Extend the system you made in assignment 2 to perform a classification on many files instead of just one

Class presentations

- Check GitHub for more detailed information, but in summary:

1. Choose a computational linguistics paper (peer-reviewed) such that involves statistical analysis of language data and that you understand well enough
2. Prepare a 10-minute presentation and basically, answer who, what, how, and why of the paper
3. Reply with your group members and paper as a reply to discussion board (okay to change until May 26)

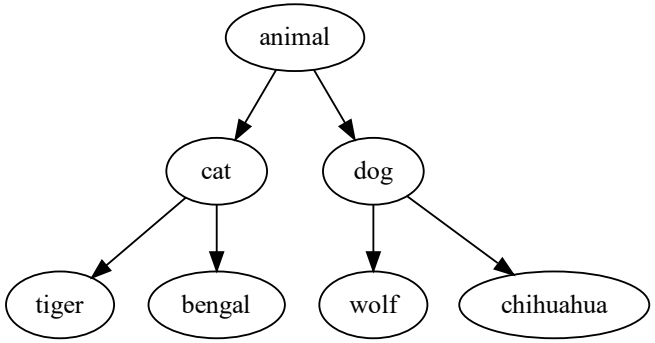
Midterm survey

- Take this optional survey via this Google Form
- It's anonymous and helps me prepare the second half of the course for you to get the most out of it

Classes and objects

Classes

- A **class** is a specific type or category of object
  - Real life examples: animal, cat, dog, plant
  - Basic Python examples: int, float, str, list, dict
- Classes are often parts of **type hierarchies**
- Any particular value in Python belongs to one or more classes
- Some types of classes can be used through Python syntax
  - Like making a string with quote marks, e.g., 'hello'



Instantiating a class

- If we want a value that belongs to a particular class, we need to **instantiate** the value from the class using a **constructor**
- Constructors often start with a capital letter
- Constructors often require some parameters and/or assume default values

```
from collections import
OrderedDict
d = collections.OrderedDict()

from datetime import date
d = date(2026, 4, 28)
print(d)
# ‘2026-04-28'
```

Class attributes

- Classes can have variables and methods associated with them
- They are accessed using the dot operator
- We have seen this already with methods like `str.split` and `list.sort`
- Same principle applies for class variables

```
from datetime import date
d = date(2026, 4, 28)
d.year # 2026
d.month # 4
d.day # 28
```

Working with filenames and pathnames

Paths

- A **path** on your computer is a location on your hard drive
- An **absolute path** specifies the exact location of the file
  - E.g., "C:\Users\matt\file.txt" or "/home/siyu/file.txt"
- A **relative path** specifies the location of a file **relative** to the folder you are in
  - E.g., ”ling471\file.txt", ”ling471/file.txt", or "file.txt"

Paths and filenames

- A filename is the name of some sort of collection of data on your computer, e.g., "slides.pptx", "main.py", or "README"
  - Usually, but not always, has an extension like ".txt", ".py"
- A path with a filename at the end is a **file path**
- A path without a filename at the end is a **directory path** or **folder path**

Manipulating paths in Python

- Python has a lot of different options for manipulating paths and finding files
- We are going to focus on a particular module called pathlib
  - Newer, more abstract way of working with paths
  - Involves instantiating a `Path` object from the `Path` class
- Other options are found in the `os` module
  - Older, sometimes more convenient, but perhaps a bit harder to work with

Instantiating a Path object

- If we want to look at paths in Python, we need a `Path` object from the pathlib module
- When we create a `Path` object, we can give it a string to identify the path
- If we don't give it an argument, it defaults to '.', which means "the current directory"
- A `Path` object does not need to point to a path that exists

```
# Examples may give
# "WindowsPath" on Windows

from pathlib import Path
p = Path()
# PosixPath('.')

p = Path('C:/Users/matt')
# PosixPath('C:/Users/matt')
```

Changing a Path object

- A `Path` object can have additional information added to it using the / operator
- `Path` objects can also be joined within another call to the `Path` constructor

```
From pathlib import Path

p = Path()
# PosixPath('.')

p = p / 'Documents' / 'essay.pdf'
# PosixPath('Documents/essay.pdf')

p1 = Path('/home/matt/Documents')
p2 = Path('ENG101', 'essay.pdf')
p_full = Path(p1, p2)
#
PosixPath('/home/matt/Documents/ENG101/e
ssay.pdf')
# same as p1 / p2
```

Getting lists of files in a path

- A `Path` object has a method called `glob`
- Calling `glob` will find files and directories in that path that match a certain pattern
- `glob` returns a generator, so you need to call list on it if you want a list
- Can use a list comprehension to only get files

```
from pathlib import Path

p = Path()
p.glob('*')

txt_files = list(p.glob('*.txt'))

files = [x for x in p.glob('*') if
x.is_file()]
```

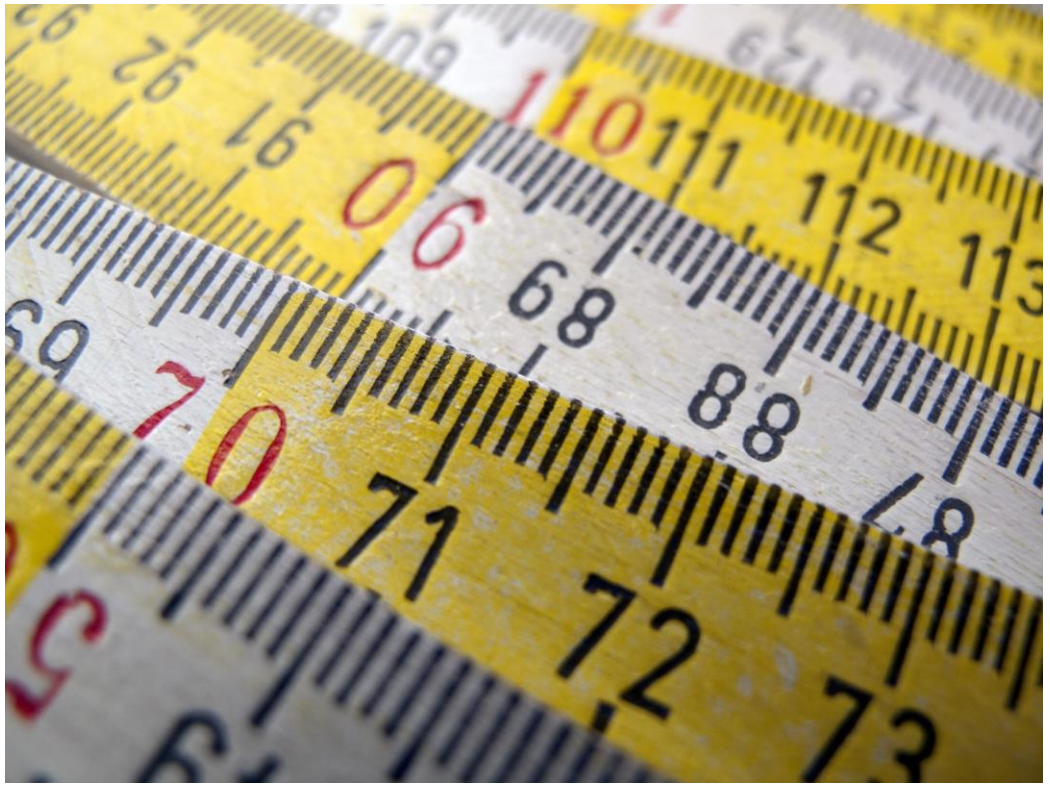
Programming activities

- Create a Path object to a folder on your computer and store it in a variable
- Add a subdirectory to that Path object
- Create a list of all files and folders in the path your object refers to
- Create a list of only the files in the path your object refers to
- Create a list of only the files that have a docx file extension in the path your object refers to
  - Do the same but for pdf, and then py, and then jpg

Metrics

Evaluating NLP and data science systems

- Recall from earlier that we usually evaluate NLP systems using some kind of metric
  - We are "measuring" the error
- Some of the most common are accuracy, precision, and recall
- You will often need to choose which one is most relevant for your system



Accuracy

- How many data points out of the total were correctly identified/classified?

$$acc = \frac{n_{corr}}{n_{tot}}$$

- n_{corr} – number of correct predictions
- n_{tot} – total number of predictions

```
import random as r
r.seed(20220426)
gold = r.choices([0, 1], k=1000)
preds = r.choices([0, 1], k=1000)
corr = sum(1 for g, p in zip(gold, preds) if g == p)
acc = corr / len(gold)
# or
acc = sum(g == p for g, p in zip(gold, preds))
# or
corr = 0
for i in range(len(gold)):
    g = gold[i]
    p =preds[i]
    if g == p:
        corr += 1
acc = corr / len(gold)
```

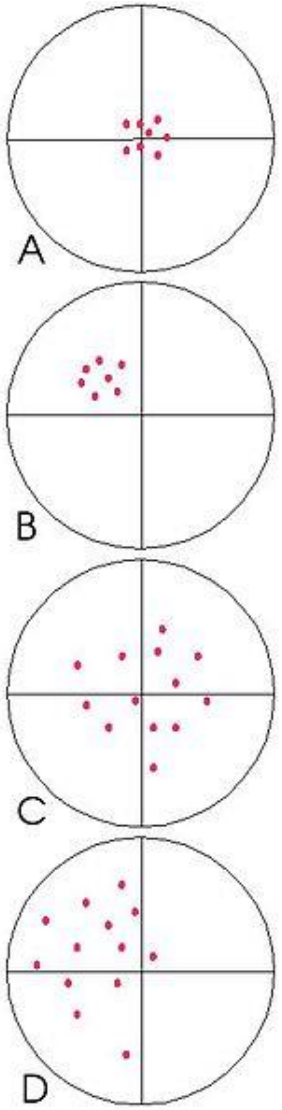


Image from Guam-common-swiki under CC BY-SA 3.0

Notes on accuracy

- Can be computed when you don't care about classes of items
  - E.g., You don't care how good the system does at recognizing bad reviews in particular
  - You only care how well it did overall
- Circles A and C are accurate
  - C is not precise

Accuracy paradox

- Suppose you have two systems: System A and System B
- Both systems output predicted medical diagnoses
  - POS – has the disease
  - NEG – does not have the disease
- Which system has higher accuracy?

		Predictions			
		System A		System B	
		neg	pos	neg	Pos
True value	neg	999	0	990	9
	pos	1	0	0	1

Accuracy paradox

- System A accuracy:
  - 999/1000 = 0.999 = 99.9%
- System B accuracy:
  - 991 / 1000 = 0.991 = 99.1%
- Which system would you prefer in your hospital?
  - System A did not identify the person who had a disease
  - System B correctly identified the person who had the disease

		Predictions			
		System A		System B	
		neg	pos	neg	Pos
True value	neg	999	0	990	9
	pos	1	0	0	1

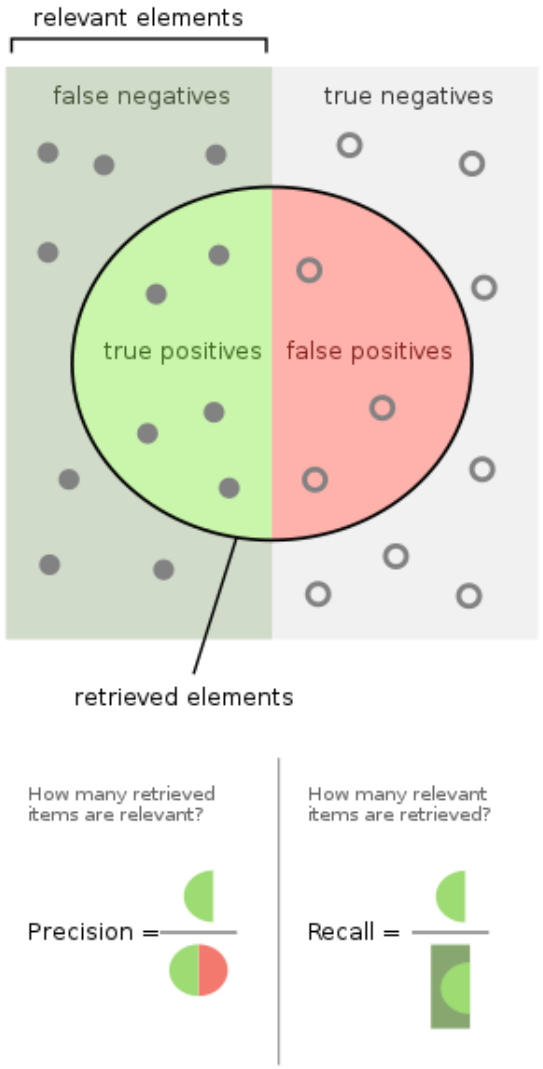


Image from Walber under CC BY-SA 4.0

Precision and recall

- Some systems need to be accurate, e.g., part of speech tagger
- Some systems need more precision, e.g., voice assistants
  - Want to understand speaker correctly but care less if it doesn't react sometimes
- Some systems need better recall, e.g., medical diagnoses
  - Want to **correctly** diagnose **ALL** sick patients
  - Care **less** about those who it says are sick but are actually healthy
  - But, we also don't want to give chemo to a healthy patient…

Precision and recall

- How well does the system do with respect to **relevant** items?
- I.e., you have **different types of items** and want to know how well the system does with respect to **each** type separately
- E.g., how good is the system at recognizing **bad** reviews?
  - May be important for business considerations
  - Ethical considerations, e.g., is the system biased to do well on one class of items?

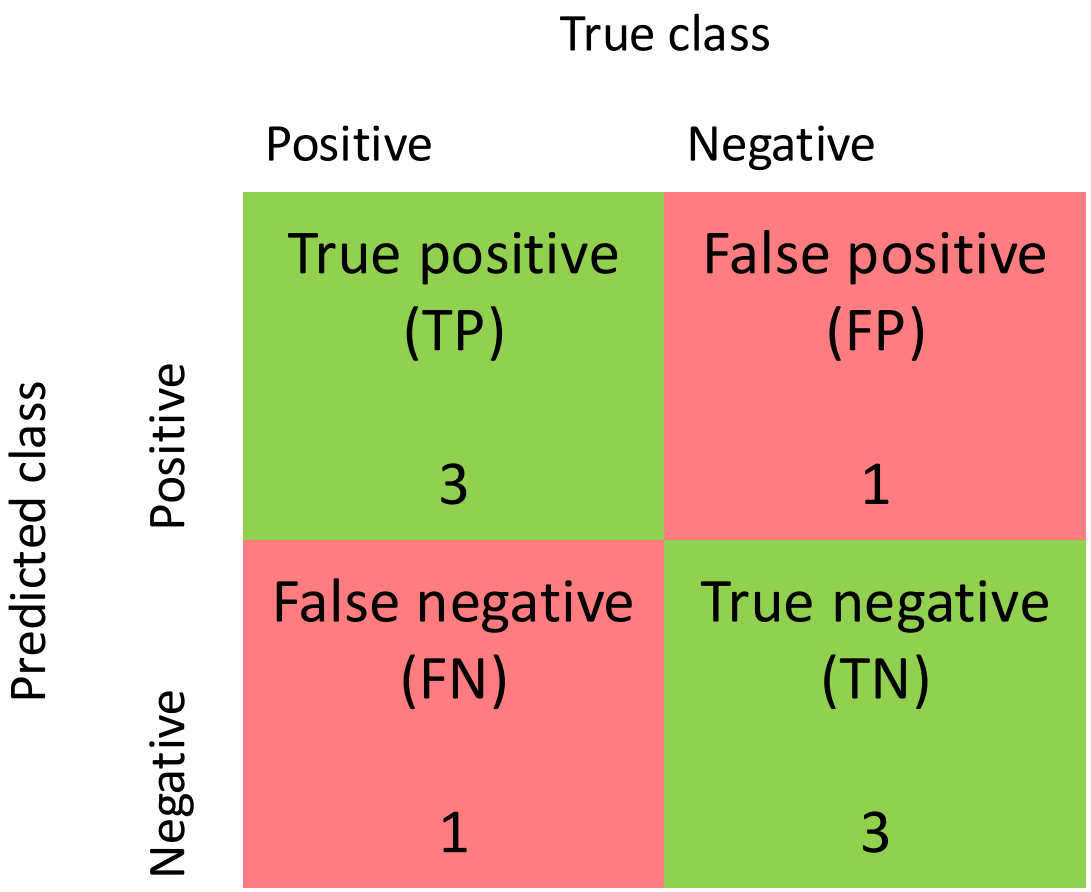
Tensions between metrics

- An ideal system is accurate and precise and has high recall
- In reality, precision and recall are in tension with each other
- E.g., it is **trivial** to have a system which has 100% recall and unacceptably low precision
  - 100% recall: Always predict always predict TRUE/POSITIVE
- Accuracy and precision can be in tension as well
  - See bias-variance tradeoff

Precision and recall example

- Context: apple 🍎 **retrieval**
  - Positive means retrieving apple; negative means retrieving lemon
- Ground truth: [🍎🍎🍎🍎🍋🍋🍋🍋]
- Our system: [🍎🍋🍎🍎🍋🍋🍎🍋]
- True positive: 0, 2, 3 (total of 3)
- False positive: 6 (total of 1)
- True negative: 4, 5, 7 (total of 3)
- False negative: 1 (total of 1)
- Can now compute precision and recall per definitions

Visualizing classifications: Confusion matrix



- We can see how often an object of a particular class got classified correctly or as another class
- Can look down upper-left to lower-right diagonal to see accuracy

Precision and recall calculation

PRECISION

$$\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\frac{3}{3+1} = \frac{3}{4} = 0.75$$

RECALL

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\frac{3}{3+1} = \frac{3}{4} = 0.75$$

Precision and recall for assignment 3

- The use of "positive" and "negative" can be confusing
- The name of the **review type** ("positive" and "negative") has **nothing** to do with **error types** (e.g., false positive, false negative)
- Try thinking of reviews as "good" and "bad" instead
  - You can rename variables accordingly
  - Compute carefully with respect to each class
- Test on the tiny data sets provided
  - If you don't test, you will likely get **incorrect** results

Tiny data set

- Comes with assignment 3
- **Do** use it instead of the main data set at first
  - You do **not** want to manually debug thousands of files if your numbers are wrong
- Use tiny data set to **make sure** you **understand** how to compute precision and recall

Iterating through two lists at once: The zip function

- It's sometimes helpful to be able to iterate through two (or more) lists at once
  - Perhaps very soon…
- `for x, y in zip(a, b)`
  - `x` – item in a
  - `,` - separates each variable
  - `t` – item in b
  - `zip` – function that "zips" lists togther
  - `a` – an iterable
  - `b` – an iterable
- Can be extended with more variables in the "x, y" part and more lists in zip
- Should work on all iterables

```
a = [0, 1, 2, 3, 4]
b = [5, 6, 7, 8, 9]
for x, y in zip(a, b):
    print(f'{a}, {b}')
# 0, 5
# 1, 6
# 2, 7
# 3, 8
# 4, 9
```

Error analysis

Error analysis in a nutshell

- Evaluate your system
- Inspect the data points the system gets wrong
- Make conclusions for further development
  - And possibly for further deployment

Error analysis and computer science

- There is not much error analysis in NLP. Why?
- Recall from earlier lectures:
  - NLP emphasizes "raw" data
- Computer science often aspires to have full automation
- Error analysis requires manual work and domain expertise
- Computer science likes systems that work for "generic" data or that are "general solutions" and tends to assume all systems are like that
  - Should not matter what the data are like
  - Should solve all similar problems at once (e.g., sorting)

Error analysis and class presentations

- If the authors of the system/paper/project you are looking at did any error analysis, talk about what they did
- If they did not do error analysis, talk about why they could have done

Programming activity

- Consider a task of classifying tweets as political or not
  - 1 = political
  - 0 = not political
- You have 10 tweets, and the ground truth is [1, 1, 0, 1, 0, 0, 1, 1, 0, 1]
- Calculate accuracy, precision, and recall if your system outputs:
  - [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]
  - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  - [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
  - [1, 1, 0, 1, 0, 0, 1, 1, 0, 1]