Introduction to Machine Learning

Day 1: Setup, Git, and the Landscape

Joanna Bieri DATA301

Welcome

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours: Duke 209, Click Here for Joanna's Schedule
- Class Website

Today's Big Ideas

- Getting your computer ready to train models
- Getting your GitHub repository working
- What machine learning actually is
- Where we are going for fourteen weeks

How This Class Runs

- **Flipped.** Videos before class, hard problems in class.
- **Individual work.** Your own private repo, your own Pull Requests.
- **Two take-home exams**, released Thursday, due Monday.
- **Team final project**, and that is where the LLM material gets assessed.

Computer Set Up

What You Need

- Anaconda Python and JupyterLab
- Git
- Everything runs on **CPU**. You do not need a graphics card.

One Command

From inside your repository folder:

```
pip install -r requirements.txt
```

Installs into your regular Anaconda Python. Nothing to turn on later.

Getting to a Terminal

`jupyterlab-git` comes with the install. After you restart JupyterLab you get:

- A **Git** panel in the left sidebar for commit, push, and pull
- A **Terminal** in the Launcher, which is the easy way to get one on Windows

For the very first install: **Anaconda Prompt** on Windows, **Terminal** on Mac.

The One Trap

On **Linux**, plain `pip install torch` downloads 2.5 GB of CUDA libraries you cannot use. Install the CPU build first:

```
pip install torch==2.13.0 torchvision==0.28.0 \
  --index-url https://download.pytorch.org/whl/cpu
```

Check Your Understanding

- **Q1.** Why does `requirements.txt` say `transformers<6.0` instead of just `transformers`?
- **Q2.** `import torch` fails in a notebook but works in the terminal. What is the most likely reason?

Your Git Workflow

Two Repositories

Repo	What	Write?
`MachineLearningUoR/FALL26`	Course materials	No
`DATA301-F26-<you>`	**Yours**, private	Yes

`FALL26` is your `upstream`. Your repo is your `origin`. Material flows one way.

Getting New Material

```
git fetch upstream
git merge upstream/main
git push origin main
```

Turning In Work

```
git checkout -b hw/week-01
git add . && git commit -m "..."
git push origin hw/week-01
```

Then open a **Pull Request** into your own `main`. **That PR is the submission.** My comments on it are your feedback.

Check Your Understanding

- **Q3.** Why branch instead of committing straight to `main`?
- **Q4.** Peer reviewers may comment but may not commit. Why that line?

What Is Machine Learning?

Two Ways to Write a Spam Filter

By hand. Notice spam says "free money". Write a rule. Spammers write "fr33 m0ney". Write another rule. Forever.

By learning. Collect labeled email. Let the program find the patterns. When spammers change, retrain instead of rewriting.

The second is not smarter than you. It just does what you would do, across more examples than you have patience for.

A More Precise Definition

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*
> *Tom Mitchell*

For the spam filter:

- **T** = classify whether something is spam or not
- **E** = trying to classify thousands of emails that already have labels
- **P** = accuracy, how many did we get right

It starts out bad, and it gets better with experience. That is supervised learning.

Kinds of Learning

Kind	Data	Example
Supervised	Examples **with** answers	Photos labeled cat/dog
Unsupervised	Examples, no answers	Customer segments
Self-supervised	Answers from the data itself	Predict a hidden word
Reinforcement	Rewards, not answers	Agent plays a game

Keep **self-supervised** in mind. It is the trick behind large language models, and we get there in November.

Two Flavors of Supervised Learning

- **Classification** predicts a category. Spam or not spam. Which of ten digits. Whether a patient gets readmitted.
- **Regression** predicts a number. The price of a house. Tomorrow's temperature. How many minutes until you arrive.

Quick test: a label is classification, a quantity you can do arithmetic on is regression.

The same situation can go either way. "Will this customer leave?" is classification. "How many months until this customer leaves?" is regression.

You Try

Classification or regression?

Real Estate

- **A.** The exact selling price of a house in dollars
- **B.** Whether a house sells in under 30 days or longer

Email

- **A.** Sorting mail into "Work," "Personal," or "Spam"
- **B.** How many minutes someone spends reading an email

Farming

- **A.** The total weight of tomatoes one plant will harvest
- **B.** Labeling a plant "Healthy," "Nutrient Deficient," or "Diseased"

Check Your Understanding

- **Q5.** A hospital wants to predict which patients will be readmitted within 30 days. Which kind of learning, and what would the training data need to look like?

The Cycle That Runs This Whole Course

The Loop

Frame the question → split the data → train → tune on the validation set → check the test set **once** → report what you got

Three Piles of Data

- **Training set.** The model learns from this one.
- **Validation set.** For while you are still making choices. Check it as often as you want.
- **Test set.** Used once, at the end, to report how you did.

The validation set is the one that saves you. Try fifteen models against validation, pick one, then check the test set a single time.

The One Rule

You only get to use your test set once.

Every time you check your test performance and then go change something, a little information about the test set leaks into your model, and your reported number gets less honest.

The Rule in the Real World

In practice you sometimes have to look more than once: to debug broken code, fix a data pipeline error, or check the model is predicting anything at all.

The golden rule still holds. **Every time you peek at the test set to make a design choice, you are allowing data leakage.**

Looking to see whether your code runs is fine. Looking to decide what to change is not.

What This Course Is Not

If you took DATA 201, you have met kNN, logistic regression, naive Bayes, trees, SVMs, PCA, k-means. **We are not re-teaching those.**

We start from the harder question: *how do you know if a model is any good?* Three weeks on that, because most people get it wrong.

Being Wrong Has a Cost

Every model is wrong sometimes. The question is what happens when it is, and who it happens to.

- Navigation app says 12 minutes, real answer 14. Costs you almost nothing.
- Fraud model gets it wrong one way: a real purchase is declined at the grocery store.
- Fraud model gets it wrong the other way: a stolen card keeps working.

Not the same size mistake, and not the same person paying. Plain "accuracy" hides all of this. More on Day 5.

Check Your Understanding

- **Q6.** You tune your model fifteen times against the test set and reach 91%. What is your honest estimate of how it performs on new data? Why?
- **Q7.** Pick any model you use. What does being wrong look like, and who pays for it?

The Importance of Data

Getting Good Data Is the Hard Part

Most of you are data science majors or minors, so you already know how hard it is to find, wrangle, and clean data. It is the first and probably the most important part of building a good model.

1. Not Enough Data

To train a good model you need a lot of data. Modern LLMs were trained on essentially the whole internet.

Banko and Brill (2001) found that almost any model, simple or complicated, can perform comparably well if you just give it enough data.

2. Data That Does Not Represent the Real Thing

Train on data that differs from what the model will actually see, and it will never perform well.

Build a fruit classifier where only 0.001% of your images are figs. It will not identify figs. Worse, when you split into three piles, there is a real chance no figs land in the training set at all.

3. Poor Quality Data

If your data is wrong, your model is wrong. Missing features, outliers, or just a bad save.

4. Irrelevant Features

More features do not always make a better model. Sending in variables unrelated to the task makes models behave strangely.

Training a breast cancer detector and feeding it patients' initials will not help, and could make it worse.

Feature selection, extraction, and engineering are data science skills that feed straight into machine learning.

Check Your Understanding

- **Q8.** You have 10,000 images for a fruit classifier, and 8 of them are figs. What happens, and what would you do about it?
- **Q9.** Give an example of a feature that would be irrelevant for predicting house prices. Why might including it actually hurt?

Where We Are Going

Five Units, Fourteen Weeks

1. **Evaluation done right** (Days 1-5)
2. **Ensembles**, the classifiers 201 skips (Days 6-8)
3. **Neural networks** (Days 9-13)
4. **Representations**, CNNs and embeddings (Days 14-16)
5. **Transformers and LLMs** (Days 17-22)

By December

You will **build a small language model from scratch** and train it on your own laptop. Attention implemented by you. A transformer block assembled by you. Trained until it produces text.

It will be small and it will not be very good. That is the point. Once you have built one, the large ones stop being magic and become an engineering story about scale.

Wrap-Up

Before Thursday

1. Finish setup. Every line of the check cell prints a version.
2. Clone your repo, add `upstream`, run `nbstripout --install`.
3. Read **Geron chapter 1**.
4. Do `HW_day1.ipynb`.

If Git Fought You Today

Submit on Canvas this week and come see me in office hours. Nobody loses points for a week-one setup problem!

Bring your laptop Thursday. We start training models immediately.