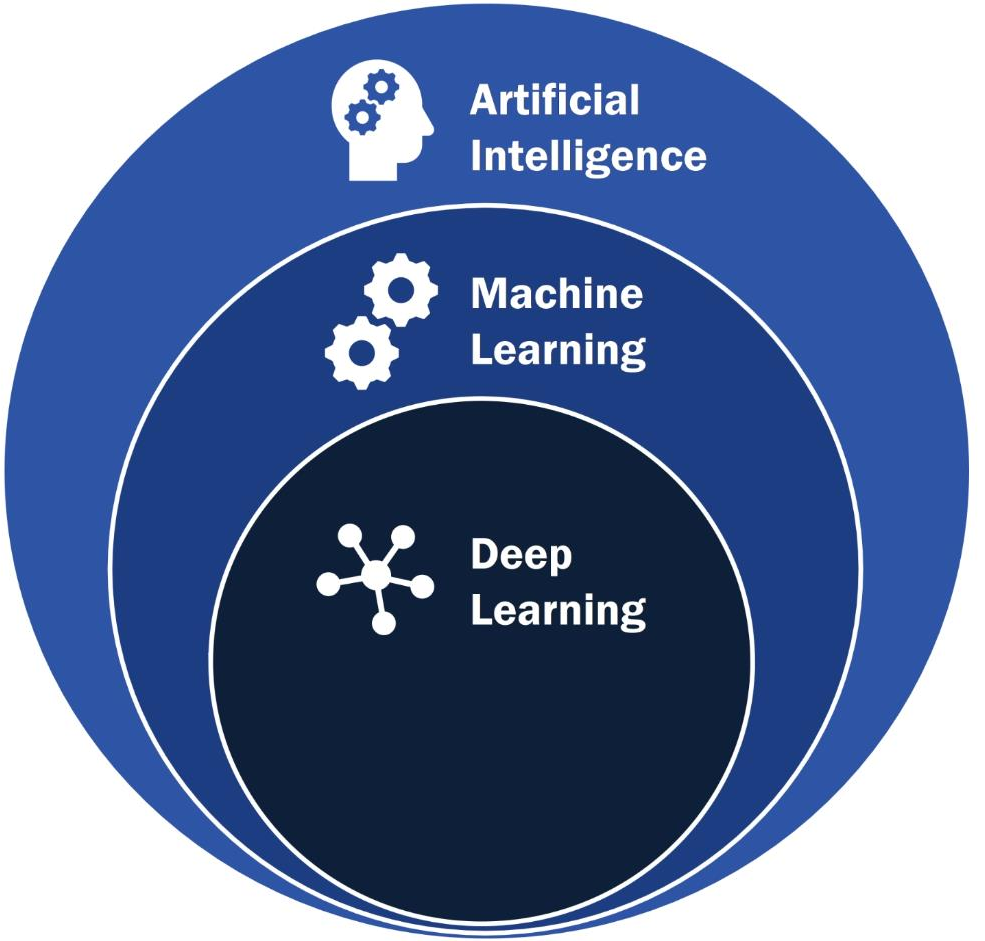
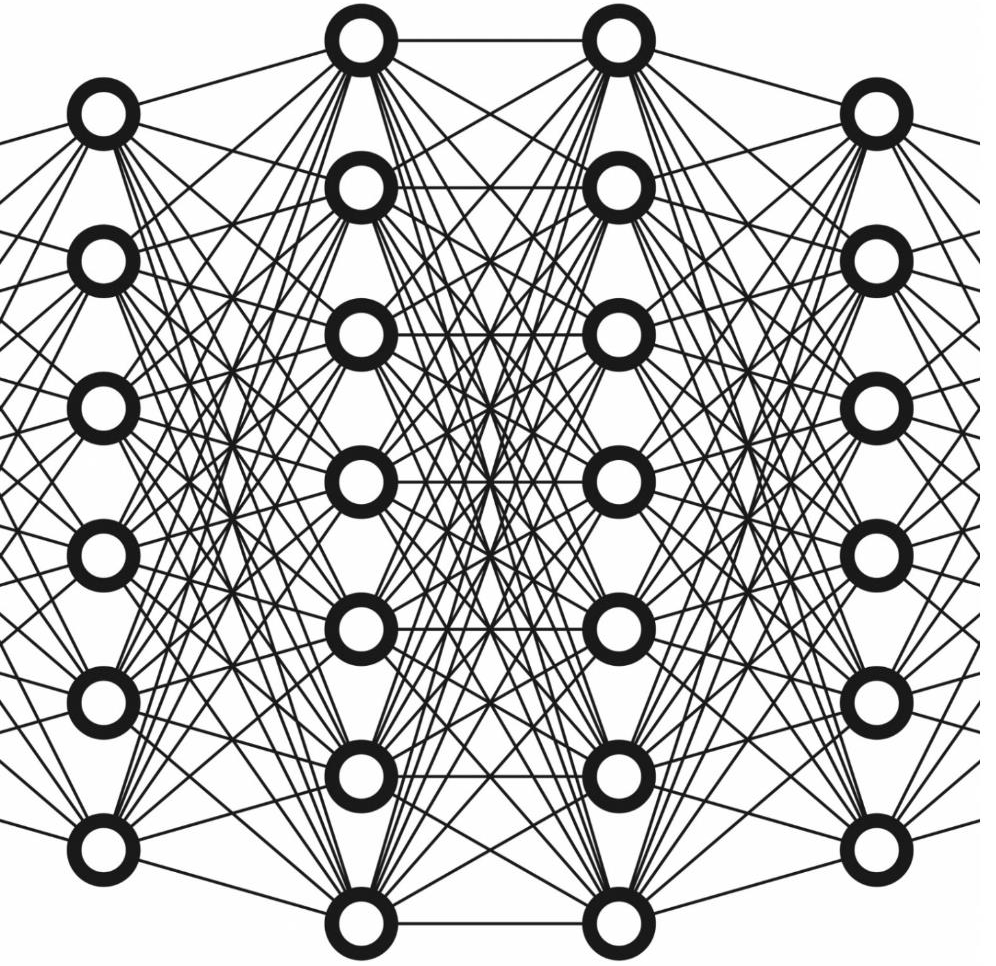
Intro to Machine Learning Tools & Applications for Geoscience

Nathalie Redick

What is Machine Learning?

It's ~fancy~ stats, not magic


Artificial
Intelligence
Machine
Learning
Deep
Learning



Overview

1. Goals
2. What can ML do?
3. Some basics
4. Demo: building a model from start to finish
5. What's a decision tree?
6. Cutting edge applications in geoscience
7. (A very incomplete) selection of ML-based research tools

Goals

- Build a basic understanding of ML
- Gain some familiarity with ML vocabulary
- Learn about some of the tools available for ML in geoscience applications/research

What can ML do?

- Process huge amounts of large, multidimensional data
- Speed up data analysis & pattern recognition
- Automate repetitive workflows
- Explore complex relationships between parameters

What can’t ML do?

- Replace the need for domain expertise
- Provide absolute certainty on its predictions (probabilistic)
- Fully comprehend the context & implications of findings
- Be creative

Some basics

Learning paradigms

- supervised
- unsupervised
- reinforcement learning

Algorithms

- regression
- dimensionality reduction
- clustering
- classification

Demo: building a model from start to finish

1. Data Preprocessing
2. Model Selection
3. Model Training
4. Model Evaluation
5. ~~Model Deployment~~

Our goal

Use volcano data from the Smithsonian database to **predict** the major rock type of a volcano based on other **features** like volcano type and tectonic setting.

Disclaimer: this is *not* a cutting edge model!

Data Preprocessing

- Data cleaning
- Data transformation
- Feature selection
- Data normalization/standardization & encoding
- Train/test split

First things first: We need some data to play with 🌋

Download Holocene and Pleistocene volcano data from the Smithsonian database using `geopandas`

```
# request data from Smithsonian database
server = 'https://webservices.volcano.si.edu/geoserver/GVP-VOTW/ows?'
queries = {
    'holocene':
        'service=WFS&request=GetFeature&typeName=GVP-VOTW:Smithsonian_VOTW_Holocene_Volcanoes&outputFormat=json',
    'pleistocene':
        'service=WFS&request=GetFeature&typeName=GVP-VOTW:Smithsonian_VOTW_Pleistocene_Volcanoes&outputFormat=json'
}

# download data using geopandas
holocene_volcanoes = gpd.read_file(server + queries['holocene'])
pleistocene_volcanoes = gpd.read_file(server + queries['pleistocene'])

# combine dataframes
volcanoes = gpd.GeoDataFrame(pd.concat([holocene_volcanoes, pleistocene_volcanoes], ignore_index=True))
```

What does the data look like?

Major_Rock_Type	Tectonic_Setting	Primary_Volcano_Type	Latitude	Longitude	Volcanic_Landform
Basalt / Picro-Basalt	Rift zone / Continental crust (> 25 km)	Lava dome(s)	45.7860	2.9810	Cluster
Basalt / Picro-Basalt	Intraplate / Continental crust (> 25 km)	Volcanic field	38.8700	-4.0200	Cluster
Andesite / Basaltic Andesite	Subduction zone / Continental crust (> 25 km)	Stratovolcano	38.6380	15.0640	Composite
…					

933 rows × 6 columns

Data cleaning & transformation

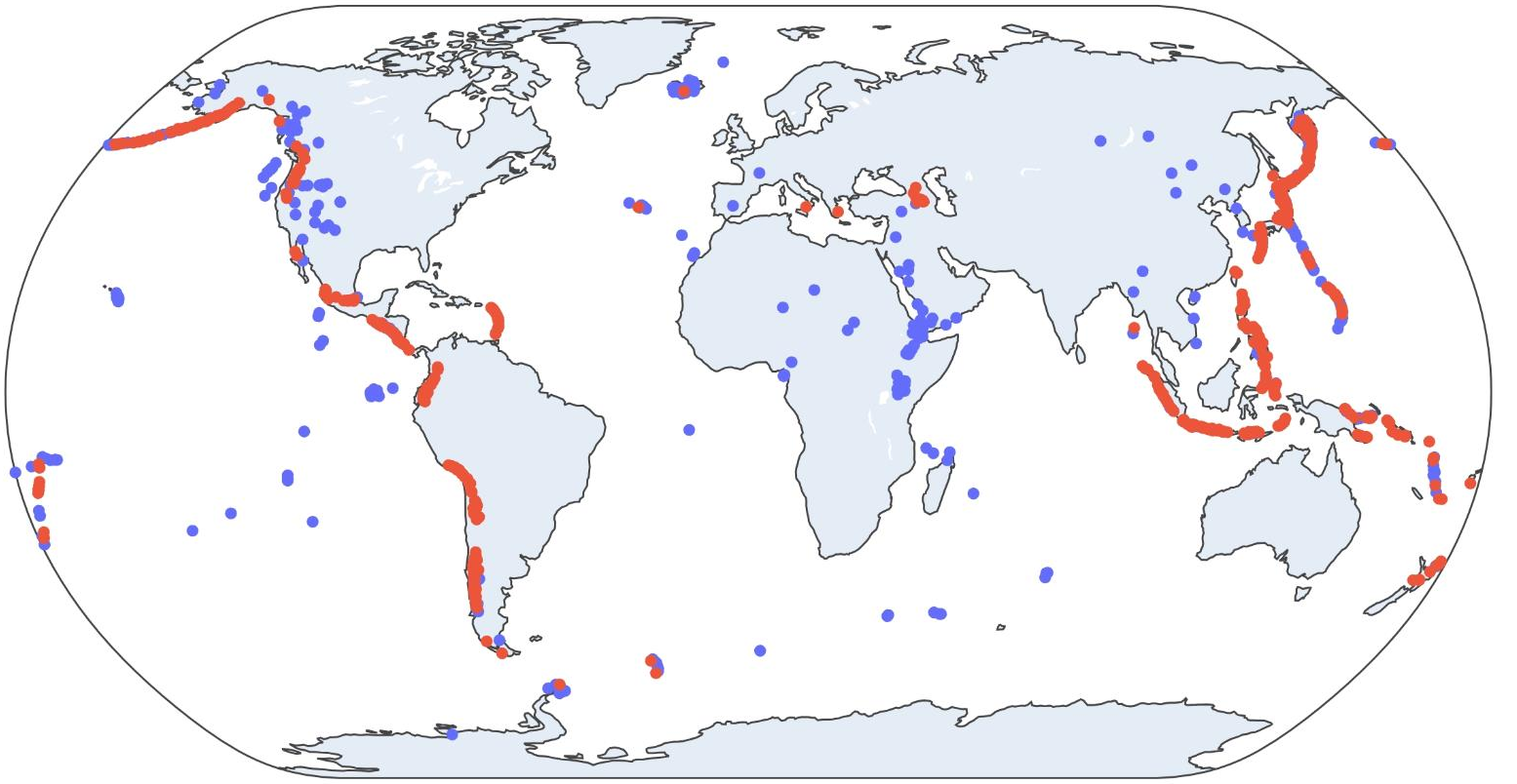
```
# remove rows with missing values
volcanoes.dropna(subset=['Latitude', 'Longitude', 'Primary_Volcano_Type', 'Tectonic_Setting', 'Major_Rock_Type'],
inplace=True)

# check out the range of values in the 'Major_Rock_Type' column
volcanoes['Major_Rock_Type'].value_counts()
```

Output:

```
Major_Rock_Type
Andesite / Basaltic Andesite               533
Basalt / Picro-Basalt                      400
Dacite                                      85
Trachybasalt / Tephrite Basanite            62
Rhyolite                                    53
No Data (checked)                           39
Trachyte / Trachydacite                     35
Trachyandesite / Basaltic Trachyandesite    24
Foidite                                     14
Phonolite                                    8
Phono-tephrite /  Tephri-phonolite           3
Name: count, dtype: int64
```

```
# filter for two most common rock types
volcanoes = volcanoes[volcanoes['Major_Rock_Type'].isin(['Andesite / Basaltic Andesite', 'Basalt / Picro-Basalt'])]
```

Pleistocene & Holocene Volcanoes by Rock Type



Feature selection, standardization, & encoding

What **features** (`X`) are we going to use to predict the major rock type (our target, `y`) of a volcano?

```
# separate features (X) and target (y)
X = volcanoes[['Latitude', 'Longitude', 'Tectonic_Setting', 'Volcanic_Landform', 'Primary_Volcano_Type']]
y = volcanoes['Major_Rock_Type']
```

Data transformation

```
# define a preprocessor to normalize numerical features and encode categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Latitude', 'Longitude']), # standardize
        ('cat', OneHotEncoder(handle_unknown='ignore'), # one-hot encoding
         ['Tectonic_Setting', 'Volcanic_Landform', 'Primary_Volcano_Type'])
    ])

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

What does "encoded" data really look like?

`y_encoded` : our **label-encoded** target tensor

```
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
...
```

`X_train` : our **feature** tensor

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
  with 3730 stored elements and shape (746, 42)>
  Coords  Values
  (0, 0)  -1.6456388799023938
  (0, 1)  1.3690578375972304
  (0, 11) 1.0
  (0, 15) 1.0
  (0, 37) 1.0
  (1, 0)  1.6163339408037167
...
```

Train/test split

1. **train**
2. **test**

```
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, shuffle=True)
```

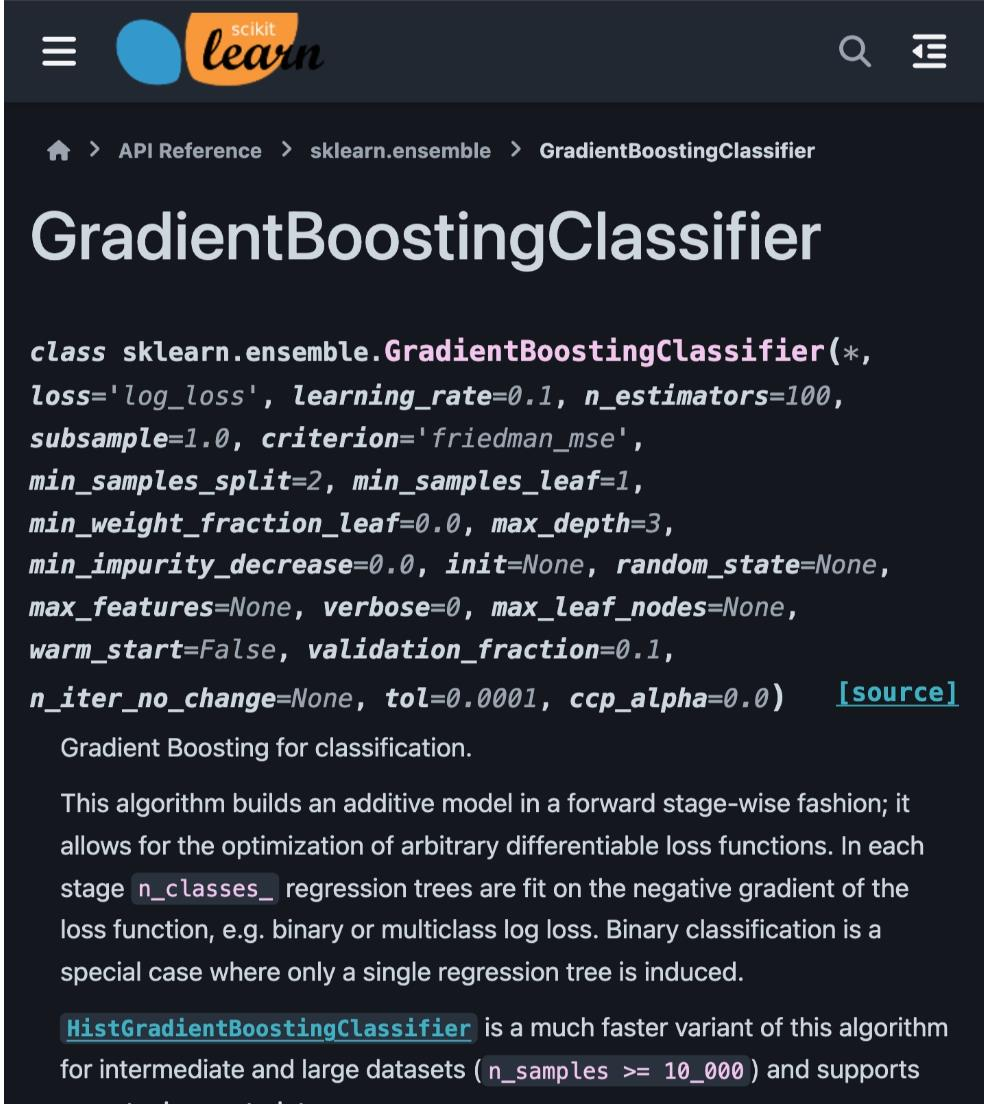
It’s important to shuffle the data before splitting to avoid any **bias** in the data.

Demo: building a model from start to finish

1. ~~Data Preprocessing~~
2. Model Selection
3. Model Training
4. Model Evaluation

Supervised classification…?

We want to **classify** the major rock type of a volcano based its other features. So, we're going to use a **supervised** learning algorithm because we have **labeled** data.



scikit learn

API Reference > sklearn.ensemble > GradientBoostingClassifier

GradientBoostingClassifier

```
class sklearn.ensemble.GradientBoostingClassifier(*,
loss='log_loss', learning_rate=0.1, n_estimators=100,
subsample=1.0, criterion='friedman_mse',
min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0, init=None, random_state=None,
max_features=None, verbose=0, max_leaf_nodes=None,
warm_start=False, validation_fraction=0.1,
n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

[source]

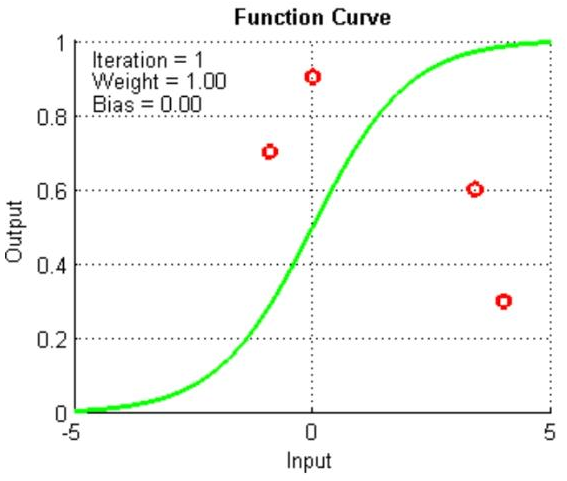
Gradient Boosting for classification.

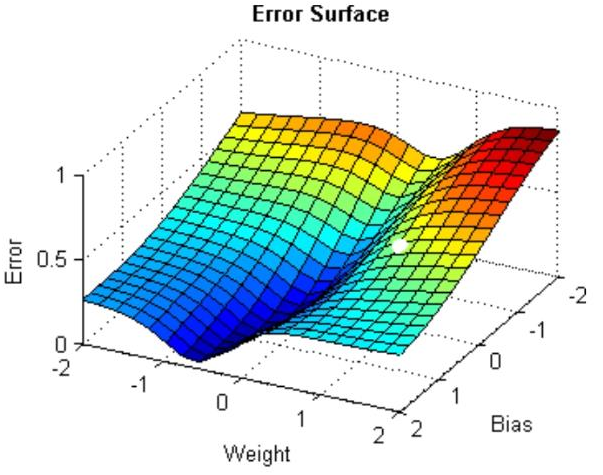
This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced.

`HistGradientBoostingClassifier` is a much faster variant of this algorithm for intermediate and large datasets (`n_samples >= 10_000`) and supports

`GradientBoostingClassifier`

Gradient $\Longrightarrow$ using **gradient descent** to minimize the loss function by fitting a series of **decision trees** (weak learners) to the **residual losses** of the previous tree.

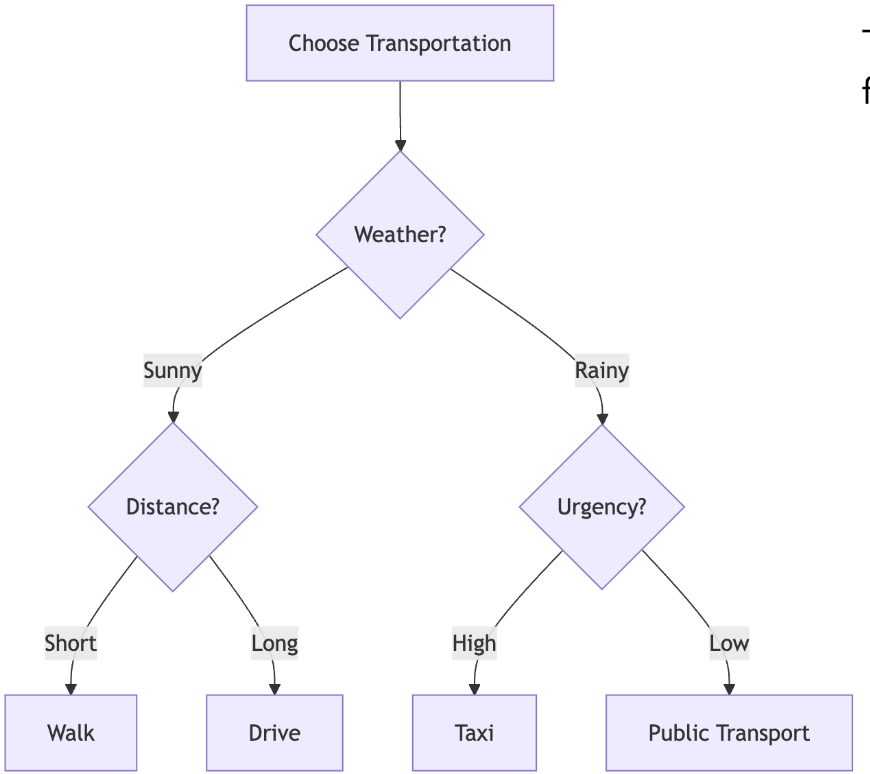




Boosting $\Longrightarrow$ combining multiple weak learners (shallow trees) to create a strong learner; this is an **ensemble** method.

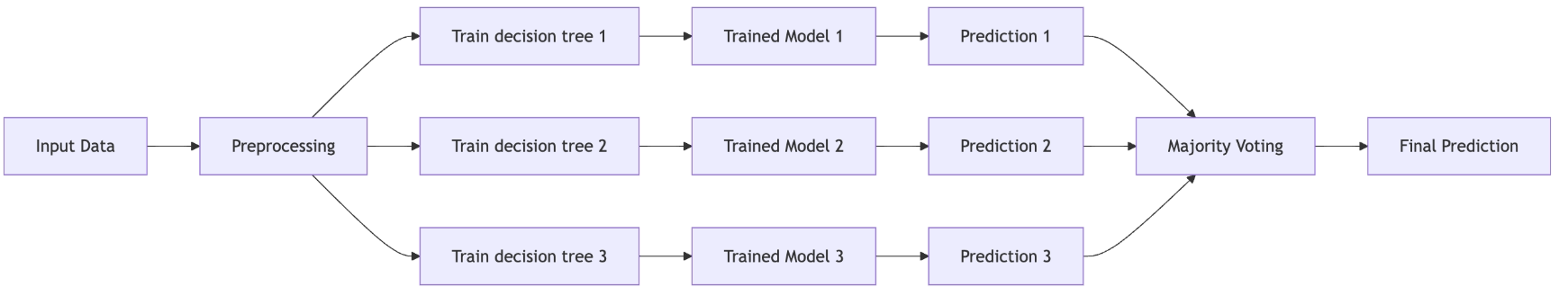
Classifier $\Longrightarrow$ we're doing **classification** to predict **categorical** outputs.

What's a decision tree?



This tree has a `depth` of 2 (since we start counting from 0 at the root node).

Ensemble methodology



Using several trained models **boosts** the overall performance of the model.

Seems kinda complex… what does the code look like?

```
# construct the model pipeline
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(
        verbose=1,
        n_estimators=100,
        validation_fraction=0.15,
        learning_rate=0.075,
        max_depth=3))
])
```

The pipeline has two steps:

1. Apply the preprocessor to the data before it is input to the classifier
    - the one-hot encoding & standardization functions that were defined earlier
2. Use the `GradientBoostingClassifier` to train the model

...that's it.

Model hyperparameters

(Some) values you can tweak to improve the performance of your model:

- `n_estimators` : the number of trees in the forest
- `learning_rate` : the rate at which the model learns from the data
- `max_depth` : the maximum depth of the tree
- `validation_fraction` : the fraction of the data to use for validation

```
# construct the model pipeline
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(
        verbose=1,
        n_estimators=100,
        validation_fraction=0.15,
        learning_rate=0.075,
        max_depth=3))
])
```

(not the same as model parameters!)

Demo: building a model from start to finish

1. ~~Data Preprocessing~~
2. ~~Model Selection~~
3. Model Training
4. Model Evaluation

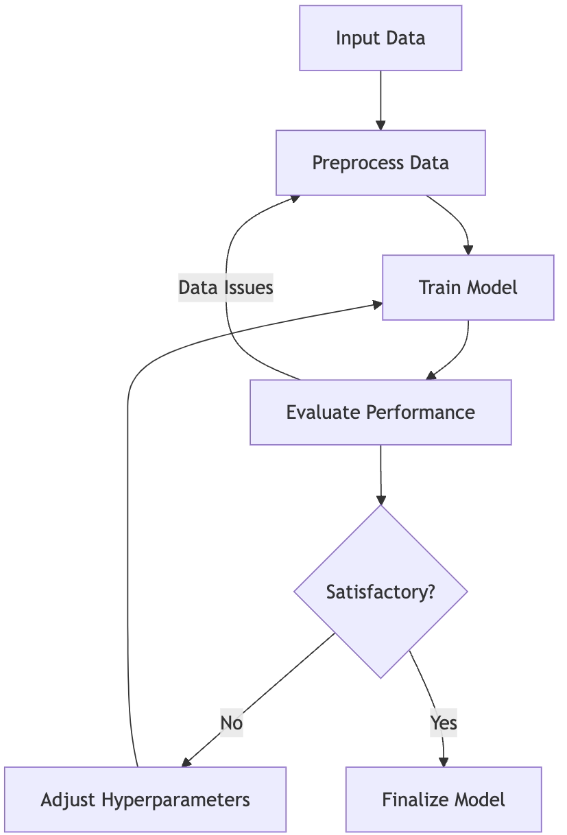
Model training

```
model.fit(X_train, y_train)
```

Output:

```
      Iter       Train Loss   Remaining Time
         1           1.3102            0.21s
         2           1.2640            0.19s
         3           1.2242            0.27s
         4           1.1895            0.24s
         5           1.1590            0.21s
         6           1.1318            0.20s
        ...
        50           0.7840            0.06s
        60           0.7564            0.04s
        70           0.7349            0.03s
        80           0.7155            0.02s
        90           0.6875            0.01s
       100           0.6723            0.00s

Accuracy on Test Data: 82.89%
Log Loss on Test Data: 0.4393
```

An iterative process



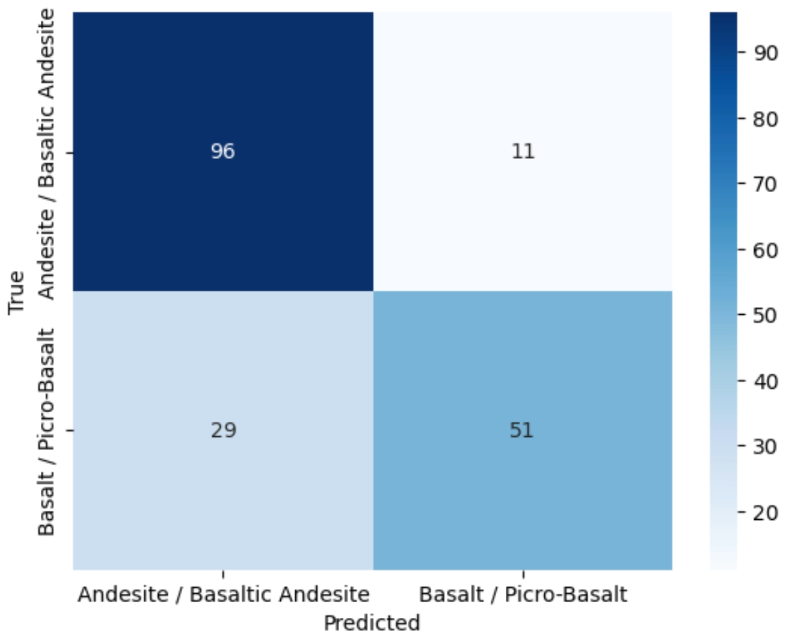
Demo: building a model from start to finish

1. ~~Data Preprocessing~~
2. ~~Model Selection~~
3. ~~Model Training~~
4. Model Evaluation

Model evaluation

Confusion matrix

```
cm = confusion_matrix(y_test, y_pred)
```



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Model evaluation

Accuracy & loss

We evaluate the model's performance on the test data, which it has never seen before.

```
Accuracy on Test Data: 82.89%
Log Loss on Test Data: 0.4393
```

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

For a single sample with true label and a probability estimate, the **log loss** is:

$$L_{\log}(y, p) = -y \log(p_i) + (1 - y) \log(1 - p)$$

scikit learn

API Reference > sklearn.metrics

sklearn.metrics

Score functions, performance metrics, pairwise metrics and distance computations.

User guide. See the Metrics and scoring: quantifying the quality of predictions and Pairwise metrics, Affinities and Kernels sections for further details.

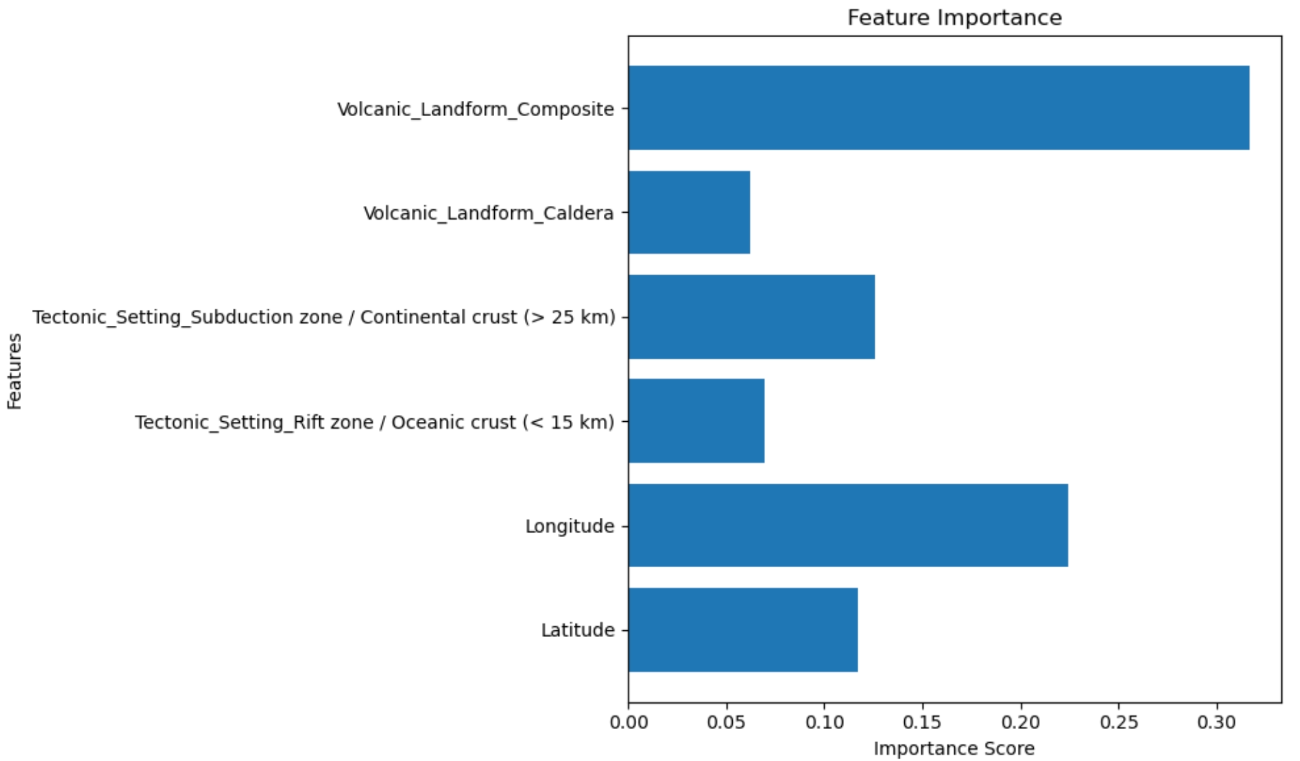
Model selection interface

User guide. See the The scoring parameter: defining model evaluation rules section for further details.

check_scoring	Determine scorer from user options

Model evaluation

Feature importance



Interpretation

The model has learned how to predict the major rock type of a volcano based on its other features. Knowing which features are most important gives us insight into the model's decision-making process and point to interesting scientific questions.

Applications?

Predicting the major rock-type en masse for volcanoes that we don't have direct rock type data on (foreign planetary bodies, etc.)

Other stuff?

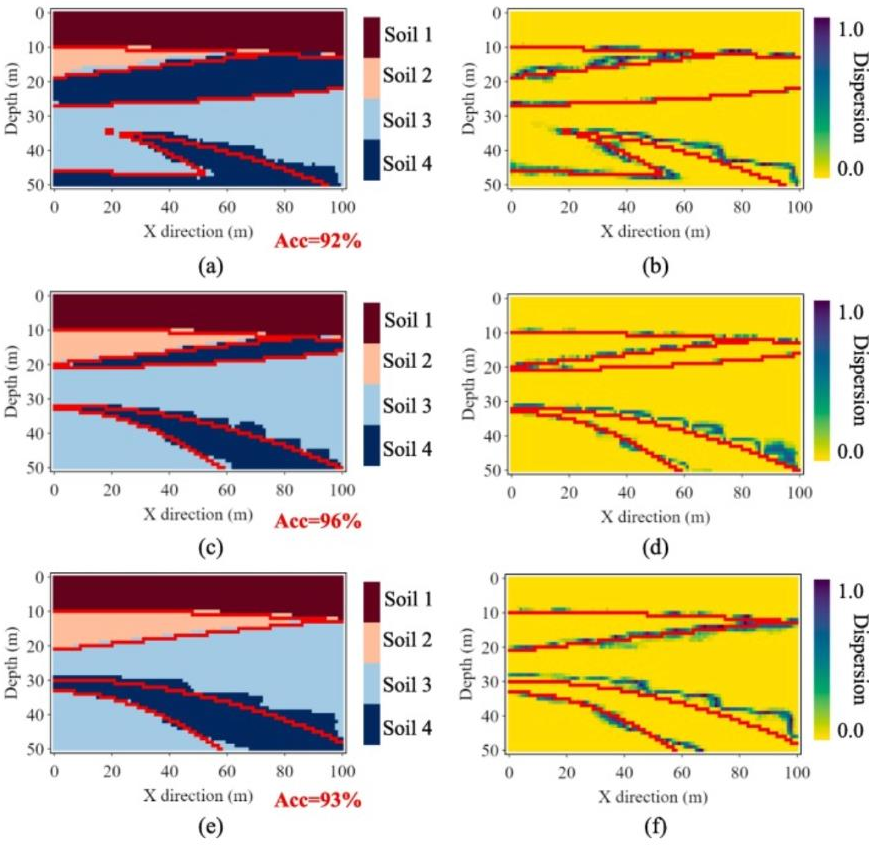
> This model is not meant to have novel scientific insight, but rather to demonstrate the process of building a model from start to finish.

Cutting edge applications in geoscience

Generative Adversarial Networks (GANs) for subsurface geological models from limited borehole data and prior geological knowledge

- can generate realistic 3D geological models from limited borehole data

(B Lyu, Y Wang, C Shi - Computers and Geotechnics, Elsevier, 2024)

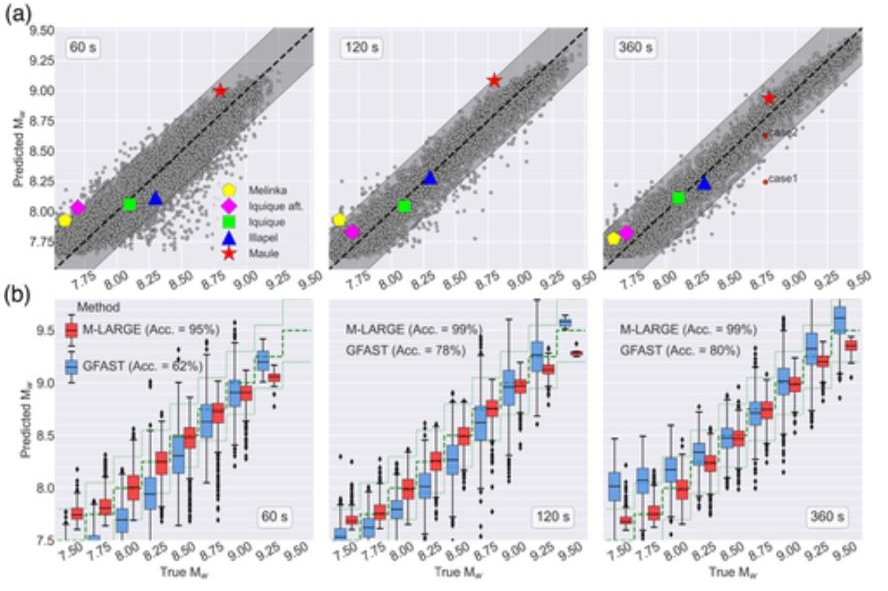


Cutting edge applications in geoscience

Early warning for great earthquakes from characterization of crustal deformation patterns with deep learning

- synthetic data to train a deep learning model to predict magnitude from crustal deformation patterns in simulated real-time
- model has an accuracy of 99% and accurately estimates the magnitude of five real large events

*(JT Lin, D Melgar, **AM Thomas**, J Searcy - Journal of Geophysical Research: Solid Earth, 2021)*



(A very incomplete) selection of ML-based research tools

1. ChatGPT, Perplexity AI (LLMs)
2. Github Copilot (for coding)
3. Segment Anything (for image segmentation)
4. Tools for discovering & dissecting papers
   - SciSpace
   - Elicit
   - Consensus

LLMs: large language models

> These are a type of **deep learning** model that can generate human-like text. They are trained on vast amounts of text data and can generate text that is nearly indistinguishable from human-written text.

Tips for working with LLMs:

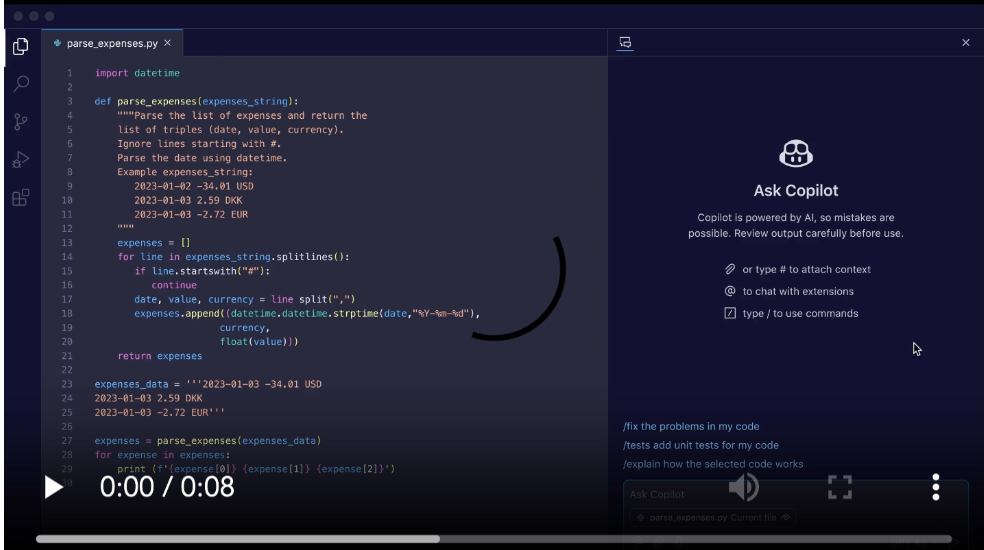
- In the settings, you can adjust the "context" of the model; for example, you can tell it that you are a researcher in geodynamics and prefer concise answers
- When you get to a result you like, you can ask the LLM to tell you what prompt would have generated that result (and save for later) ⟵ (thanks Haoyuan)

Github Copilot

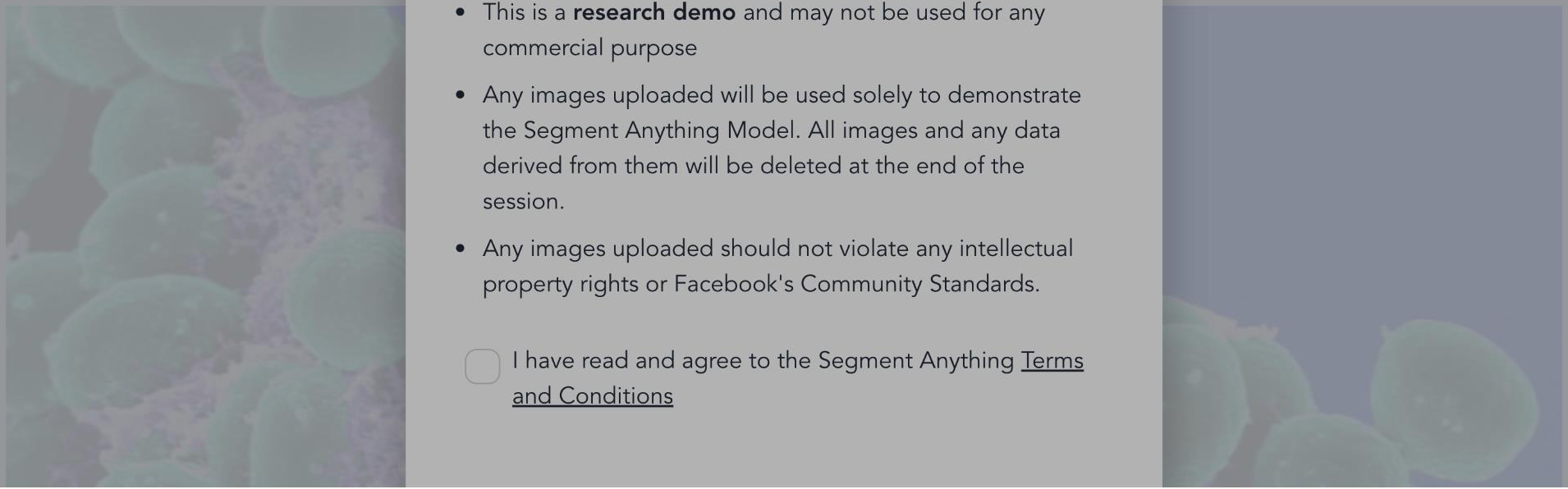
> Trained on Github's public codebase.

Features:

- Pro is **free** for students/educators
- LLM tailored specific to coding
- Real-time suggestions
- Integrated chat in your code editor
- Can generate tests, docs, etc.



↓ Find a photo in the gallery,



Before you begin

Close

- This is a **research demo** and may not be used for any commercial purpose
- Any images uploaded will be used solely to demonstrate the Segment Anything Model. All images and any data derived from them will be deleted at the end of the session.
- Any images uploaded should not violate any intellectual property rights or Facebook's Community Standards.

☐ I have read and agree to the Segment Anything Terms and Conditions