

The Prediction Machine

Teaching machines with examples

Day 02 · Week 1

Unlocks  Prediction Machine

1 Mission briefing

A field station just radioed in a spreadsheet of penguin measurements — and **no species labels**. Their expert is out sick. Your job: build a machine that names a penguin's species from its measurements alone, so the station can label the whole backlog automatically. *Previously:* as the Data Detective you cleaned this data and watched three species form clear clusters. This is your first real model.

- ☐ Learn **supervised learning**: features, labels, the train/test split
- ☐ See decision boundaries and a readable **decision tree**
- ☐ Spring the **feature-scaling trap** — and fix it
- ☐ Build **k-Nearest Neighbors** from scratch, then in 3 lines
- ☐ Diagnose mistakes with a **confusion matrix**
- ☐ Ship the champion and **unlock your first Studio tab**

2 Field guide the concepts

Supervised learning means learning from labeled examples. Show a machine hundreds of (X, y) pairs — **features** in, correct **label** out — and it learns the mapping, then guesses the label for a brand-new example. But how do you know it really learned, versus just memorized? You hide part of the data. Split into a **training set** (it studies these) and a **test set** (locked in a drawer), and grade only on the test set. It's the difference between studying past papers and sitting a fresh exam.

All 333 penguins

TRAINING SET · 249

the machine studies these

TEST · 84

 locked away
the honest exam

The golden rule: never test on data you trained on.

A 75 / 25 split. `stratify=y` keeps each species at the same proportion in both halves.

Two classifiers. **k-NN** labels a new penguin by a majority vote of its **k** nearest neighbors — small k can overfit; very large k can underfit. A **decision tree** writes itself a flowchart of yes/no questions you can read out loud. And one trap: k-NN measures "nearest" by distance, so a big-numbered feature like body mass can drown out the rest — `StandardScaler` gives every feature a fair vote.

A precise reading

Choose k and other settings using validation data or cross-validation within the training set, then evaluate once on an untouched test set. Fit scaling on training data only. A larger k makes a smoother boundary; too large can underfit. Comparing many settings on the same test set makes that score optimistic.

3 Lab missions check each off in the notebook

1 Split the data honestly

```
train_test_split(..., test_size=0.25, stratify=y).
```

Expect: 249 training penguins, 84 held out for testing.

2 Build k-NN from scratch

Fill in `knn_predict`: distance to all points → k nearest → majority vote.

Expect: it labels the first test penguin correctly.

3 Grade it on the test set

Run your k-NN on every test penguin and measure accuracy.

Expect: around 0.95 (the self-check insists on beating 0.80).

4 The same thing in 3 lines

```
KNeighborsClassifier — .fit then .score.
```

Expect: almost exactly your from-scratch score. Same algorithm.

5 Grow a tree

```
DecisionTreeClassifier(max_depth=3), then plot it.
```

Expect: a shallow flowchart you can trace as a sentence.

6 Build the confusion matrix

```
confusion_matrix(y_test, y_pred) as a heatmap.
```

Expect: a bright diagonal; the few mix-ups cluster on Chinstrap.

7 Run the tournament (raw features)

Try every k from 1–15 on all four *unscaled* features.

Expect: four raw features score *worse* than two — the trap.

8 Scale, then rerun

Same tournament on `StandardScaler`-scaled features.

Expect: accuracy jumps above 0.95 at the best k.

4 Cheat sheet supervised learning with scikit-learn

<code>train_test_split(X, y, test_size=0.25, stratify=y)</code>	Split into train/test, keeping class ratios
<code>KNeighborsClassifier(n_neighbors=5).fit(Xtr, ytr)</code>	k-NN: vote of the 5 nearest neighbors
<code>model.score(Xte, yte)</code>	Fraction correct on the test set (accuracy)
<code>model.predict(X_new)</code>	Predict labels for new examples
<code>DecisionTreeClassifier(max_depth=3)</code>	A readable if/else flowchart classifier
<code>confusion_matrix(y_test, y_pred)</code>	Rows = truth, cols = guesses; diagonal is correct
<code>StandardScaler().fit_transform(X_train)</code>	Rescale every feature to a fair, comparable spread
<code>make_pipeline(StandardScaler(), KNeighbors...)</code>	Bundle scaling + model into one object
<code>joblib.dump(bundle, path)</code>	Save the fitted model to disk

5 Unlock your Studio module

Prediction Machine

The unlock cell saves your fitted pipeline as a bundle (`model` + `feature_names` + `target_names`). It must write exactly:

```
ai_studio/models/prediction_machine.joblib
```

Then run the Studio and drag the four sliders — your model predicts live:

```
$ python ai_studio/app.py
```

6 Mini-glossary

supervised learning

Learning a mapping from labeled examples (features → known answer).

k-NN

Classify a point by the majority label of its k nearest neighbors.

accuracy

The fraction of predictions that are correct.

overfitting

Memorizing the training data (noise and all) instead of the general pattern.

train/test split

Hiding part of the data to grade the model honestly on unseen examples.

decision tree

A learned flowchart of yes/no questions ending in a prediction.

confusion matrix

A grid of true vs predicted labels — shows *where* mistakes happen.

7 Show & tell

1. What accuracy did your **Prediction Machine** reach — and which two species did it mix up most?
2. Compare the $k = 1$ and $k = 15$ boundaries: which would you trust on a brand-new penguin, and why?
3. The scaling trap made *more* data hurt. Explain to a friend why **body mass** was drowning out bill length.