MACHINE LEARNING

A practical guide to learning patterns from data and using models with care

LEARNSAPPEAL | ORIGINAL COURSE

Start with a question. Learn from examples. Check the result.

Explore supervised and unsupervised learning, neural networks, model evaluation, and the steps that make a model useful in practice.

Before you start

Comfort with basic Python is helpful. The examples use familiar ideas such as lists, variables, and functions. You can follow the concepts first and run the scikit-learn examples when that package is installed.

Course map: Foundations (Modules 1-4), then training, evaluating, and using models (Modules 5-12). Includes explanations, code examples, self-checks with answers, projects, and a glossary.

How to use this course

For each module, write down the question a model is meant to answer. Inspect the data, run the example, and explain the result in your own words. A score is only useful when you know what data produced it and what kinds of mistakes matter.

By the end, you can: describe a machine learning workflow; prepare a dataset; train a few common model types; choose evaluation measures; recognize overfitting and data leakage; and discuss deployment, privacy, and fairness considerations.

Course map

• 1. What machine learning does
• 2. Data, features, and targets
• 3. A first prediction model
• 4. Prepare data thoughtfully
• 5. Classification and regression
• 6. Unsupervised learning
• 7. Evaluate the model fairly
• 8. Improve a model
• 9. Neural networks in plain language
• 10. Text and recommendation tasks
• 11. Put a model into use responsibly
• 12. Capstone: course discovery

FOUNDATIONS

1. What machine learning does

Machine learning uses examples to find a pattern that can help make predictions or group similar items. A typical workflow defines a question, gathers examples, prepares useful inputs, trains a model, evaluates it on held-out examples, then decides whether its results are good enough for a real use. A model learns from its training data; it does not automatically understand the world.

Example

```
# A model learns a mapping from inputs to an output
# Example: course details -> likely topic category
```

Check your understanding

- What is a model trained to find?
- Why should a model be evaluated?
- Does a prediction prove that the model understands a situation?

Suggested answers

1. Patterns that help map inputs to predictions or groups.
2. To estimate how it may perform on examples it did not train on.
3. No; it has learned patterns from data and may still be wrong.

2. Data, features, and targets

A dataset contains examples, often represented as rows. Features are the input columns used by a model. A target is the value we want to predict in supervised learning. For a course catalog, features might be title and description; the target could be a topic label. Data quality and collection choices shape the model's limits.

Example

```
features = [
    ['Python basics', 'programming'],
    ['Build a web page', 'web development'],
]
# input feature: course title
# target label: course topic
```

Check your understanding

- What is a feature?
- What is a target?
- Name a reason collected data might be incomplete.

Suggested answers

1. An input value used by a model.
2. The value a supervised model is asked to predict.
3. Some groups or examples may be missing or poorly recorded.

3. Train a first prediction model

In supervised learning, training examples pair inputs with known targets. The algorithm adjusts a model to reduce its errors on those examples. Separate training data from evaluation data before fitting. Otherwise a score can look excellent simply because the model has already seen the answers.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
# Requires: python -m pip install scikit-learn
from sklearn.tree import DecisionTreeClassifier

X = [[1, 0], [1, 1], [0, 1], [0, 0]]
y = ['programming', 'programming', 'design', 'design']
model = DecisionTreeClassifier(max_depth=2, random_state=0)
model.fit(X, y)
print(model.predict([[1, 0]]))
```

Check your understanding

- What does fit() do?
- What does predict() return?
- Why keep examples aside from training?

Suggested answers

1. It learns model parameters from examples and targets.
2. A model output for the supplied input.
3. To estimate performance on data the model has not seen.

4. Prepare data thoughtfully

Models need consistent inputs. Missing values may need a documented strategy. Text categories often need encoding; numeric columns may need scaling for some algorithms. Keep transformations inside a pipeline so they are learned from training data only. Inspect duplicates, unusual values, and whether a column accidentally reveals the answer.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

model = make_pipeline(
    SimpleImputer(), StandardScaler(),
    LogisticRegression(max_iter=1000)
)
```

Check your understanding

- Why fit preprocessing on training data only?
- What does imputation do?
- What is a suspiciously predictive feature worth checking for?

Suggested answers

1. To prevent information from evaluation data leaking into training.
2. It fills missing values using a chosen rule.
3. It may directly reveal the target or encode future information.

TRAIN, EVALUATE, AND APPLY

5. Classification and regression

Classification predicts a category, such as a course topic or whether a transaction needs review. Regression predicts a numeric value, such as expected completion time. Some models output probabilities; a decision threshold turns these into actions. Thresholds should reflect the costs of false alarms and missed cases.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=7, stratify=y
)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Check your understanding

- Which task predicts a numeric amount?
- What does a probability threshold control?
- Why stratify a split for classification?

Suggested answers

1. Regression.

2. How probabilities are converted into a class or action.
3. To keep class proportions similar across splits when possible.

6. Unsupervised learning

Unsupervised methods look for structure without a known target label. Clustering groups examples that are similar under a distance measure. Dimensionality reduction compresses data into fewer values while preserving some structure. These groups are not automatically meaningful: a person must interpret them and check whether the result is useful.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.cluster import KMeans

points = [[1, 1], [1.2, 0.9], [8, 8], [8.2, 7.9]]
model = KMeans(n_clusters=2, random_state=0, n_init='auto')
labels = model.fit_predict(points)
print(labels)
```

Check your understanding

- Does clustering require target labels?
- What can affect a distance-based grouping?
- Why inspect clusters after creating them?

Suggested answers

1. No; it explores structure without known target labels.
2. Feature scales, distance choice, and number of groups.
3. To understand whether groups have useful meaning or reflect artifacts.

7. Evaluate the model fairly

Accuracy is the fraction of predictions that are correct, but it can hide poor performance on a rare class. Precision measures how many positive predictions were correct; recall measures how many actual positives were found. A confusion matrix counts correct and mistaken predictions by class. Select metrics based on the decision and its consequences.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.metrics import (
    classification_report, confusion_matrix
)
predictions = model.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

Check your understanding

- When might accuracy be misleading?
- What does recall measure?
- Why keep a final test set separate?

Suggested answers

1. When classes are imbalanced or errors have different costs.
2. The share of actual positives the model finds.
3. To avoid tuning repeatedly to the same evaluation examples.

8. Improve a model without fooling yourself

A baseline gives a simple point of comparison. Cross-validation tests models on several training/evaluation splits. Hyperparameter search compares chosen settings. Tune using training folds and keep a final test set untouched until decisions are complete. Compare errors as well as scores, and prefer a simpler model when its performance is similar and easier to explain.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(max_depth=3, random_state=0)
scores = cross_val_score(model, X, y, cv=5)
print(scores.mean(), scores.std())
```

Check your understanding

- What is a baseline?
- Why use cross-validation?
- When should the final test set be used?

Suggested answers

1. A simple reference model or rule to beat.
2. To estimate performance across multiple data splits.
3. After model and settings are chosen, for a final estimate.

9. Neural networks in plain language

A neural network combines layers of adjustable calculations. During training, it compares predictions with targets, computes a loss, and adjusts weights to reduce that loss. More layers and parameters can learn complex patterns but often need more data, compute, and careful evaluation. Neural networks are one family of models, not the right tool for every task.

Example

```
# Conceptual training loop
for inputs, targets in batches:
    predictions = model(inputs)
    loss = loss_function(predictions, targets)
    loss.backward()       # compute gradients
    optimizer.step()      # update weights
    optimizer.zero_grad()
```

Check your understanding

- What does a loss function quantify?
- What do gradients guide?
- Why might a simpler model be preferable?

Suggested answers

1. How different predictions are from desired targets.
2. How to adjust model weights to reduce loss.
3. It may need less data and compute and be easier to inspect.

10. Text and recommendation tasks

Text models often turn words into numeric representations before learning. A simple approach counts words; more advanced representations capture context. A recommender may rank items using user-item interactions or item attributes. Watch for privacy concerns, popularity bias, and feedback loops where recommendations shape the data used for future training.

Example

Examples using scikit-learn require: python -m pip install scikit-learn

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

texts = ['learn Python coding', 'design a web page']
labels = ['programming', 'web']
vectorizer = TfidfVectorizer()
X_text = vectorizer.fit_transform(texts)
model = LogisticRegression().fit(X_text, labels)
```

Check your understanding

- Why convert text into numeric values?
- What is popularity bias?
- How can a recommendation system create a feedback loop?

Suggested answers

1. Most learning algorithms operate on numeric representations.
2. Frequently shown items can keep receiving more attention than less visible items.
3. Its suggestions change user behavior, which becomes new training data.

11. Put a model into use responsibly

A deployed model receives inputs and returns outputs through an application or service. Validate inputs, protect private data, record useful operational metrics, and keep a rollback plan. Monitor data drift and performance changes. Make clear when people should rely on human review, and provide a way to report harmful or incorrect outcomes.

Example

```
# A simple prediction service should
# 1. validate request fields
# 2. load a known model version
# 3. return prediction and useful context
# 4. log errors and latency without sensitive data
# 5. support rollback to a previous version
```

Check your understanding

- What is data drift?
- Why version a deployed model?
- Name one useful monitoring signal.

Suggested answers

1. A change in input data patterns compared with training data.
2. To reproduce behavior and roll back if needed.
3. Error rate, latency, input distribution, or outcome quality.

12. Capstone: course discovery

Build a small course-topic classifier using a modest, responsibly collected set of course titles and labels. Start with a baseline. Split the data before fitting text transformations. Compare a simple model with another option, inspect mistakes, and describe where the model should not be trusted. A human should be able to correct its suggestions.

Example

```
# Capstone outline
# 1. collect course titles and reviewed topic labels
# 2. split into train, validation, and final test sets
# 3. build TF-IDF + LogisticRegression pipeline
# 4. measure per-topic precision and recall
# 5. inspect errors and document limitations
```

Check your understanding

- Why should topic labels be reviewed?
- Why inspect individual mistakes?
- What should a course catalog do when confidence is low?

Suggested answers

1. Incorrect labels teach the model incorrect patterns.
2. Aggregate scores can hide systematic or costly errors.
3. Ask for a human choice or present the result as a suggestion.

Project workshop

1. Predict course topics

Create a labeled collection of course titles, build a text classification baseline, and inspect mistakes by topic. Document data sources, label rules, and limits.

2. Explore learner activity groups

Use a small privacy-safe, synthetic or consented dataset to group activity patterns. Scale features thoughtfully, compare cluster counts, and describe what the groups may and may not mean.

3. Compare two models

Pick a prediction question and a dataset. Define a baseline and the cost of errors, compare two models using a suitable split, inspect metrics by class, and write a short recommendation.

Project checklist
Is the question clear? Are data sources and limitations documented? Was evaluation kept separate from training? Are errors understood? Could a person review or correct the result?

Glossary

Term	Meaning
Algorithm	A procedure used to learn patterns or make predictions.
Classification	Predicting a category.
Feature	An input value used by a model.
Label	A target category attached to an example.
Model	A learned mapping from inputs to outputs.
Overfitting	Learning training examples too closely, reducing performance on new data.
Precision	The fraction of positive predictions that are correct.
Recall	The fraction of actual positive examples that are found.
Regression	Predicting a numeric value.
Target	The output a supervised model learns to predict.
Training data	Examples used to fit model parameters.
Data leakage	Information crossing into training that would not be available at prediction time.

Further study

Use current official documentation when installing or exploring libraries. Continue by reproducing a small result, inspecting the data, and writing down the assumptions behind your evaluation.

• scikit-learn. *User Guide.* https://scikit-learn.org/stable/user_guide.html
• Google. *Machine Learning Crash Course.* https://developers.google.com/machine-learning/crash-course
• NIST. *AI Risk Management Framework.* https://www.nist.gov/itl/ai-risk-management-framework

Course completion checklist
I can explain the machine learning workflow; prepare features and targets; train classification or regression models; use a suitable evaluation split and metric; identify likely leakage or overfitting; compare results with a baseline; and describe practical, privacy, and fairness limits.