Machine Learning Course: From Zero to Advanced

C Kradem

September 2025

Contents

1 Introduction **3**

2 Step 1: Foundations (Math & Programming) **3**
2.1 Why Foundations Are Important . . . 3
2.2 Mathematics Foundations . . . 3
2.2.1 Linear Algebra . . . 3
2.2.2 Calculus . . . 3
2.2.3 Probability and Statistics . . . 4
2.2.4 Linear Statistics . . . 4
2.3 Programming Foundations . . . 4
2.3.1 Python Basics . . . 4
2.3.2 Working with Data . . . 5
2.3.3 Version Control & Reproducibility . . . 5
2.4 Example Exercise . . . 5

3 Step 2: Data Handling & Preprocessing **5**
3.1 Why Preprocessing is Important . . . 5
3.2 Data Collection & Loading . . . 6
3.3 Data Cleaning . . . 6
3.4 Data Transformation . . . 6
3.5 Data Splitting . . . 7
3.6 Exploratory Data Analysis (EDA) . . . 7
3.7 Example Exercise . . . 7

4 Step 3: Model Building **8**
4.1 Introduction . . . 8
4.2 Supervised Learning Models . . . 8
4.2.1 Linear Regression (Regression Task) . . . 8
4.2.2 Logistic Regression (Classification Task) . . . 9
4.2.3 Decision Trees . . . 9
4.2.4 Support Vector Machines (SVM) . . . 10
4.3 Unsupervised Learning Models . . . 10
4.3.1 k-Means Clustering . . . 10
4.3.2 Principal Component Analysis (PCA) . . . 11
4.4 Advanced Models . . . 11

4.4.1 Ensemble Methods . . . 11
4.4.2 Neural Networks (Intro) . . . 11
4.5 Key Takeaways . . . 12

5 Step 4: Model Evaluation **12**
5.1 Why Model Evaluation Matters . . . 12
5.2 Train-Test Split . . . 12
5.3 Evaluation Metrics for Regression . . . 13
5.3.1 Mean Squared Error (MSE) . . . 13
5.3.2 Root Mean Squared Error (RMSE) . . . 13
5.3.3 Mean Absolute Error (MAE) . . . 13
5.3.4 R-squared (R^2 Score) . . . 13
5.4 Evaluation Metrics for Classification . . . 13
5.4.1 Confusion Matrix . . . 13
5.4.2 Accuracy . . . 13
5.4.3 Precision . . . 13
5.4.4 Recall (Sensitivity) . . . 14
5.4.5 F1-Score . . . 14
5.4.6 ROC Curve & AUC . . . 14
5.5 Cross-Validation . . . 14
5.6 Overfitting vs Underfitting . . . 15
5.7 Key Takeaways . . . 15

6 Step 5: Deployment & Real-World Applications **15**
6.1 Spam Detection (Classification Example) . . . 15
6.1.1 How It Works . . . 15
6.1.2 Mathematical Idea . . . 16
6.1.3 Code Example . . . 16
6.1.4 Real-World Example . . . 16
6.2 Recommendation Systems . . . 16
6.2.1 Types of Recommendation Systems . . . 16
6.2.2 Mathematical Idea (Collaborative Filtering) . . . 16
6.2.3 Code Example . . . 16
6.2.4 Real-World Example . . . 17
6.3 Deployment Techniques . . . 17
6.3.1 Steps in Deployment . . . 17
6.3.2 Real-World Platforms . . . 17

1 Introduction

Machine Learning (ML) is the science of building algorithms that allow computers to learn from data. It is a subset of Artificial Intelligence (AI) and is widely used in image recognition, natural language processing, finance, and healthcare.

This course provides a structured roadmap to learn ML step by step, with theory and practice.

2 Step 1: Foundations (Math & Programming)

2.1 Why Foundations Are Important

Before diving into advanced machine learning, you must master the essential tools:

- **Mathematics:** Gives you intuition about how algorithms work internally.
- **Programming:** Enables you to implement models and experiments in practice.

2.2 Mathematics Foundations

2.2.1 Linear Algebra

Linear algebra is the language of machine learning. It provides the framework for representing and manipulating data.

- Vectors and matrices.
- Matrix multiplication and transposition.
- Eigenvalues and eigenvectors.
- Applications in ML:
  - Representing datasets as matrices ($X \in \mathbb{R}^{n \times d}$).
  - Linear regression as $y = Xw + b$.
  - Dimensionality reduction (PCA).

2.2.2 Calculus

Calculus helps optimize models by minimizing loss functions.

- Derivatives and gradients.
- Partial derivatives for multivariable functions.
- Gradient Descent algorithm:

  $$\theta = \theta - \alpha \cdot \nabla J(\theta)$$

  where α is the learning rate and $J(\theta)$ is the cost function.
- Applications in ML:
  - Backpropagation in neural networks.
  - Optimization of regression and classification models.

2.2.3 Probability and Statistics

Machine learning models often deal with uncertainty, and probability provides the framework.

- Random variables, distributions (Normal, Bernoulli, Binomial).
- Bayes' Theorem:
$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$
- Expectation and variance.
- Applications in ML:
  - Naïve Bayes classifier.
  - Probabilistic graphical models.
  - Uncertainty estimation in predictions.

2.2.4 Linear Statistics

- Mean, median, mode.
- Standard deviation, variance.
- Covariance and correlation.
- Hypothesis testing (p-values, confidence intervals).
- Applications:
  - Feature analysis.
  - Model evaluation (statistical significance of results).

2.3 Programming Foundations

2.3.1 Python Basics

Python is the most widely used language in ML.

- Data types: lists, dictionaries, sets.
- Loops, conditionals, functions.
- Object-Oriented Programming (OOP).
- Libraries for ML:
  - **NumPy:** numerical computations.
  - **Pandas:** data manipulation.
  - **Matplotlib/Seaborn:** visualization.

2.3.2 Working with Data

- Importing datasets (CSV, Excel, JSON).
- Cleaning data (handling missing values, duplicates).
- Exploratory Data Analysis (EDA).
- Data visualization: histograms, scatter plots, boxplots.

2.3.3 Version Control & Reproducibility

- Using Git/GitHub to track code changes.
- Writing clean, modular code.
- Documenting experiments (e.g., Jupyter notebooks).

2.4 Example Exercise

Problem: Implement gradient descent for linear regression in Python.

Mathematical Formulation:

$$y = wx + b$$

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - (wx_i + b))^2$$

Python Pseudocode:

```
# Initialize parameters
w, b = 0, 0
alpha = 0.01
epochs = 1000

for epoch in range(epochs):
    y_pred = w*x + b
    dw = (-2/m) * sum(x * (y - y_pred))
    db = (-2/m) * sum(y - y_pred)
    w = w - alpha * dw
    b = b - alpha * db
```

3 Step 2: Data Handling & Preprocessing

3.1 Why Preprocessing is Important

In machine learning, raw data is rarely ready for direct use. Preprocessing transforms messy, incomplete, or inconsistent data into a clean format suitable for algorithms.

- Good data ⇒ better models and higher accuracy.
- Poor data ⇒ biased or meaningless results.

3.2 Data Collection & Loading

- Import data from multiple sources:
  - CSV, Excel, JSON files.
  - Databases (SQL).
  - Web scraping, APIs.
- Python tools:
  - `pandas.read_csv()`, `read_excel()`, `read_json()`.
  - `SQLAlchemy` for databases.
  - `requests`, `BeautifulSoup` for web data.

3.3 Data Cleaning

- **Handling Missing Values:**
  - Remove rows/columns with too many missing values.
  - Fill with mean/median/mode.
  - Use advanced imputation (e.g., k-Nearest Neighbors imputer).
- **Handling Duplicates:** Remove duplicate records.
- **Handling Outliers:**
  - Detect with boxplots or z-scores.
  - Decide whether to remove, cap, or transform them.

3.4 Data Transformation

- **Feature Scaling:**
  - Normalization: scales values to $[0, 1]$.
  - Standardization: transforms to mean 0 and standard deviation 1.
- **Encoding Categorical Variables:**
  - One-hot encoding for nominal data (e.g., colors: red, blue, green).
  - Label encoding for ordinal data (e.g., low=1, medium=2, high=3).
- **Feature Engineering:**
  - Create new features (e.g., extract "year" from a date).
  - Polynomial features for nonlinear models.

3.5 Data Splitting

- Split dataset into:
  - **Training set** (used to fit the model, e.g. 70%).
  - **Validation set** (used for tuning hyperparameters, e.g. 15%).
  - **Test set** (used for final evaluation, e.g. 15%).
- Python function:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)
```

3.6 Exploratory Data Analysis (EDA)

Before training, understand your dataset through visualizations and statistics:

- Summary statistics: `df.describe()`.
- Data distributions: histograms, density plots.
- Correlation heatmaps.
- Scatter plots to detect trends and relationships.

3.7 Example Exercise

Problem: You receive a dataset with missing values and categorical features. Preprocess it for training.

Python Pseudocode:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Load data
df = pd.read_csv("data.csv")

# Split features and labels
X = df.drop("target", axis=1)
y = df["target"]

# Define preprocessing
num_features = ["age", "salary"]
```

```
cat_features = ["gender", "city"]

num_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="mean")),
    ("scaler", StandardScaler())
])

cat_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_features),
        ("cat", cat_transformer, cat_features)
    ]
)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)

# Apply preprocessing
X_train_ready = preprocessor.fit_transform(X_train)
X_test_ready = preprocessor.transform(X_test)
```

4 Step 3: Model Building

4.1 Introduction

Once the data is cleaned and preprocessed, the next step is to build machine learning models. Model building involves choosing the right algorithm, training it on data, and preparing it for evaluation. There are three main categories of ML models:

- **Supervised Learning** (regression and classification).
- **Unsupervised Learning** (clustering, dimensionality reduction).
- **Advanced Models** (neural networks, ensemble methods).

4.2 Supervised Learning Models

4.2.1 Linear Regression (Regression Task)

- **Goal:** Predict continuous values (e.g., house price).
- **Equation:**

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

- **Loss Function (MSE):**

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

- **Python Example:**

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

4.2.2 Logistic Regression (Classification Task)

- **Goal:** Predict categories (e.g., spam vs. not spam).
- **Hypothesis:**

$$P(y = 1|x) = \sigma(wx + b), \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Loss Function (Binary Cross-Entropy):**

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \Big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \Big]$$

- **Python Example:**

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
```

4.2.3 Decision Trees

- **Goal:** Split data based on feature values to classify or predict.
- **Key Concept:** Use measures like **Gini Impurity** or **Entropy**.

$$Entropy(S) = -\sum_{c=1}^{C} p_c \log_2 p_c$$

- **Python Example:**

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=3)
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
```

4.2.4 Support Vector Machines (SVM)

- **Goal:** Find the hyperplane that maximizes the margin between classes.
- **Decision Function:**
$$f(x) = w \cdot x + b$$
- **Kernel Trick:** Allows SVM to work with nonlinear data.
- **Python Example:**

```
from sklearn.svm import SVC

svm = SVC(kernel="rbf")
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
```

4.3 Unsupervised Learning Models

4.3.1 k-Means Clustering

- **Goal:** Group data into k clusters.
- **Algorithm:**
  1. Randomly initialize k centroids.
  2. Assign each point to the nearest centroid.
  3. Update centroids by computing the mean of each cluster.
  4. Repeat until convergence.
- **Python Example:**

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

clusters = kmeans.labels_
```

4.3.2 Principal Component Analysis (PCA)

- **Goal:** Reduce dimensionality while preserving variance.
- **Mathematics:**
  - Compute covariance matrix.
  - Extract eigenvectors and eigenvalues.
  - Project data onto top k eigenvectors.
- **Python Example:**

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

4.4 Advanced Models

4.4.1 Ensemble Methods

- Combine multiple models for better accuracy.
- Examples:
  - Random Forests (multiple decision trees).
  - Gradient Boosting (e.g., XGBoost, LightGBM).
- **Python Example:**

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

4.4.2 Neural Networks (Intro)

- **Goal:** Learn complex nonlinear patterns.
- **Basic Structure:**
  - Input layer.
  - Hidden layers with activation functions (ReLU, Sigmoid, Tanh).
  - Output layer.
- **Forward Propagation:**

$$a^{[l]} = \sigma(W^{[l]}a^{[l-1]} + b^{[l]})$$

- **Python Example (using Keras):**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(16, activation="relu", input_shape=(X_train.shape[1],)),
    Dense(8, activation="relu"),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
model.fit(X_train, y_train, epochs=20, batch_size=32)
```

4.5 Key Takeaways

- Start simple (linear/logistic regression), then try advanced models.
- Always compare multiple algorithms.
- Complexity $\neq$ better results; sometimes simple models win.

5 Step 4: Model Evaluation

5.1 Why Model Evaluation Matters

Building a model is only the beginning. Without evaluation, we cannot know:

- How well the model performs.
- Whether it generalizes to new data.
- If it is overfitting or underfitting.

5.2 Train-Test Split

- Training set: used to fit the model.
- Test set: used to evaluate performance on unseen data.
- Validation set: optional, used for hyperparameter tuning.

Python:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

5.3 Evaluation Metrics for Regression

5.3.1 Mean Squared Error (MSE)

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

5.3.2 Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$

5.3.3 Mean Absolute Error (MAE)

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|$$

5.3.4 R-squared (R^2 Score)

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Python:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

5.4 Evaluation Metrics for Classification

5.4.1 Confusion Matrix

$$\text{Matrix} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

5.4.2 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

5.4.3 Precision

$$Precision = \frac{TP}{TP + FP}$$

5.4.4 Recall (Sensitivity)

$$Recall = \frac{TP}{TP + FN}$$

5.4.5 F1-Score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Python:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

5.4.6 ROC Curve & AUC

- ROC = Receiver Operating Characteristic curve.
- AUC = Area Under Curve (higher is better).

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}$$

Python:

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

y_proba = clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

plt.plot(fpr, tpr, label="ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()

auc = roc_auc_score(y_test, y_proba)
```

5.5 Cross-Validation

Instead of a single train-test split, use k-fold cross-validation.

- Split dataset into k folds.
- Train on $k - 1$ folds, test on 1 fold.
- Repeat k times and average performance.

Python:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Mean Accuracy:", scores.mean())
```

5.6 Overfitting vs Underfitting

- **Overfitting:** Model performs well on training data but poorly on test data.
- **Underfitting:** Model performs poorly on both training and test data.
- **Solution:** Regularization, cross-validation, more data.

5.7 Key Takeaways

- Use regression metrics (MSE, RMSE, R^2) for continuous outputs.
- Use classification metrics (Accuracy, Precision, Recall, F1, ROC/AUC) for categorical outputs.
- Always compare multiple models using consistent evaluation.

6 Step 5: Deployment & Real-World Applications

Machine Learning is not only about building models in notebooks, but also about **deploying** them in real-world scenarios. In this step, we focus on two key applications: **Spam Detection** and **Recommendation Systems**. We also discuss deployment techniques.

6.1 Spam Detection (Classification Example)

Spam detection is a **binary classification problem**:

$$y \in \{0, 1\}, \quad 0 = \text{Not Spam}, \quad 1 = \text{Spam}$$

6.1.1 How It Works

1. **Data Collection:** Emails or SMS messages are collected with labels (spam / not spam).
2. **Preprocessing:**
   - Convert text to lowercase.
   - Remove punctuation and stopwords.
   - Transform text into numerical vectors using Bag of Words or TF-IDF.
3. **Model Training:** Algorithms like Naïve Bayes, Logistic Regression, or even Neural Networks can be trained.
4. **Evaluation:** Precision and Recall are critical (we want to avoid missing spam).

6.1.2 Mathematical Idea

Using Naïve Bayes:

$$P(\text{Spam}|\text{Words}) = \frac{P(\text{Words}|\text{Spam}) \cdot P(\text{Spam})}{P(\text{Words})}$$

6.1.3 Code Example

from sklearn.feature$_e$*xtraction.textimportTfidfVectorizerfromsklearn.model*$_s$*electionimporttrain*$_t$*es*
Example dataset texts = ["Win a free iPhone now!", "Let's meet tomorrow at 5"]
labels = [1, 0] 1=spam, 0=not spam
Vectorization (TF-IDF) vectorizer = TfidfVectorizer() X = vectorizer.fit$_t$*ransform*(*texts*)
Train/test split X$_t$*rain*, *X*$_t$*est*, *y*$_t$*rain*, *y*$_t$*est* = *train*$_t$*est*$_s$*plit*(*X*, *labels*, *test*$_s$*ize* = 0.2)
Train classifier clf = MultinomialNB() clf.fit(X$_t$*rain*, *y*$_t$*rain*)
Evaluate y$_p$*red* = *clf.predict*(*X*$_t$*est*)*print*(*classification*$_r$*eport*(*y*$_t$*est*, *y*$_p$*red*))

6.1.4 Real-World Example

This method is used in Gmail, Outlook, and Yahoo Mail to filter spam emails. **Challenge:** Spammers constantly adapt, so models must be retrained often.

6.2 Recommendation Systems

Recommendation systems are used by Netflix, Amazon, and Spotify to suggest content.

6.2.1 Types of Recommendation Systems

- **Content-Based Filtering:** Recommends items similar to those the user liked before.
- **Collaborative Filtering:** Recommends items liked by similar users.
- **Hybrid:** A mix of both approaches.

6.2.2 Mathematical Idea (Collaborative Filtering)

We approximate the rating matrix R (users $\times$ items) as:

$$R_{u,i} \approx P_u \cdot Q_i^T$$

where P_u is the latent feature vector for user u and Q_i is for item i.

6.2.3 Code Example

import pandas as pd from surprise import Dataset, SVD from surprise.model$_s$*electionimportcross*$_v$*alida*
Load movie rating dataset data = Dataset.load$_b$*uiltin*($'ml-100k'$)
Use Matrix Factorization (SVD) model = SVD() results = cross$_v$*alidate*(*model*, *data*, *measures* = [$'RMSE'$,$'MAE'$], *cv* = 3)
print("Average RMSE:", results['test$_r$*mse*'].*mean*())

6.2.4 Real-World Example

- **Netflix:** Suggests movies/shows based on user preferences.
- **Amazon:** "Customers who bought this also bought...".
- **Spotify:** Creates personalized playlists.

6.3 Deployment Techniques

Once trained, models need to be used in production.

6.3.1 Steps in Deployment

1. **Model Export:** Save model using `joblib` or `pickle`.
2. **API Creation:** Build a web service with Flask or FastAPI.
3. **Integration:** Connect API to web apps, mobile apps, or backend systems.
4. **Monitoring:** Track model performance and retrain when accuracy drops.

6.3.2 Real-World Platforms

- **Cloud:** AWS, Google Cloud, Microsoft Azure.
- **Containers:** Docker + Kubernetes.
- **Lightweight Deployment:** Streamlit, Gradio for demos.