In-Class Exercises: Supervised Learning

Work through these during the lecture. Ask if you get stuck.

Data Science with Python – BSc Course

Exercise 1: OLS by Hand

8 min | Pen & Paper

Five training points:

Point	x	y
1	1	2.0
2	2	3.5
3	3	5.0
4	4	6.2
5	5	7.8

Fit an ordinary least squares (OLS) line $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ using the closed-form formulas:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

(a) Compute $\bar{x}$ and $\bar{y}$.

(b) Fill in the working table:

x_i	y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$
1	2.0			
2	3.5			
3	5.0			
4	6.2			
5	7.8			

(c) Compute the numerator $\sum(x_i - \bar{x})(y_i - \bar{y})$ and the denominator $\sum(x_i - \bar{x})^2$, then obtain $\hat{\beta}_1$ and $\hat{\beta}_0$.

(d) Predict $\hat{y}$ at $x = 6$. What is the residual if the true $y_6 = 9.2$?

Exercise 2: Read the Coefficient Path

5 min | Conceptual

Refer to the ridge regression coefficient path shown in **lecture slide 22** (Coefficient Shrinkage: Ridge vs. Lasso). The plot shows four coefficients $\beta_1, \beta_2, \beta_3, \beta_4$ on the vertical axis and the regularization strength λ on the horizontal axis (increasing to the right).

(a) As λ increases, in which direction do the coefficients move — toward zero or away from zero? Why?

(b) Which feature has the *largest absolute coefficient* at small λ? In practical terms, what does this tell us about the feature?

(c) At what λ do all four coefficients become essentially indistinguishable from zero? What is the resulting model at that point?

(d) In a **Lasso** (L1) path plot, some coefficients would become *exactly* zero at finite λ, while in a Ridge (L2) path they only approach zero asymptotically. Why does this difference matter in practice?

Exercise 3: Compute Precision, Recall, F1

7 min | Pen & Paper

A fraud-detection classifier produced the following confusion matrix on a test set of 1,000 transactions. The positive class is "fraudulent."

	Predicted Fraud	**Predicted Legit**
Actual Fraud	TP = 80	FN = 15
Actual Legit	FP = 20	TN = 885

Recall the definitions:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN},$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \qquad \text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}.$$

(a) Compute precision, recall, F1, and accuracy. Show your working.

(b) A baseline that labels *every* transaction as legitimate would achieve what accuracy? What would its precision and recall on the fraud class be?

(c) In fraud detection, investigators pay a cost for each false positive (wasted investigation time), and the bank loses money on each false negative (fraud goes through). If false negatives are 10× more costly than false positives, would you prefer a model with high precision or high recall?

Exercise 4: Which Model?

5 min | Discussion

Match each scenario to the most appropriate supervised learning method. Write your reasoning — more than one method can be defensible.

#	Scenario	Best Method?
A	Predict a continuous stock return (real-valued target) given a small set of clean, roughly linear factors.	
B	Classify whether a loan applicant will default. The bank's risk committee requires that each decision be *explainable* to regulators.	
C	Classify credit-card transactions as fraud vs. legitimate. The training data has only 1% fraud, and the bank wants top performance.	
D	Predict quarterly revenue using 50 candidate features, many of which are strongly correlated with each other.	

Methods to choose from: Linear Regression, Logistic Regression, Decision Tree, Random Forest, Lasso, SMOTE + XGBoost.

Bonus: For scenario C, why is plain logistic regression a poor choice? What does SMOTE fix, and why add XGBoost on top?

Exercise 5: Build a Classifier Pipeline

10 min | Python (Optional)

Complete the blanks in this `sklearn` pipeline that predicts loan default from three borrower features. The pipeline should: (1) standardize features, (2) generate polynomial interactions, (3) fit a logistic regression.

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import ________, ________    # (1)(2)
from sklearn.linear_model import ________              # (3)
from sklearn.model_selection import train_test_split
import numpy as np

# Synthetic loan data: 500 applicants, 3 features (income,
# debt_ratio, credit_score); target y is 0 = no default, 1 = default.
np.random.seed(42)
X = np.random.randn(500, 3)
y = (X[:, 0] + X[:, 1] - X[:, 2] + 0.3 * np.random.randn(500) > 0).
    astype(int)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Build the pipeline
pipe = Pipeline([
    ('scaler', ________(________)),       # (4) standard scaler
    ('poly',   ________(________)),       # (5) degree-2 polynomial
    ('clf',    ________(________)),       # (6) logistic reg., C=1.0
])

# Fit on training data
pipe.________(X_train, y_train)            # (7) which method?

# Evaluate
train_acc = pipe.score(X_train, y_train)
test_acc  = pipe.score(X_test,  y_test)
print(f"Train accuracy: {train_acc:.3f}")
print(f"Test accuracy : {test_acc:.3f}")
```

(a) Fill in blanks (1)–(7).

(b) Why is the order (scaler → polynomial features → classifier) important? What would happen if you applied polynomial features *before* standardization?

(c) The regularization strength in `LogisticRegression` is parameterized as `C` (inverse of λ). Does `C = 0.01` or `C = 100` correspond to stronger regularization?

Exercise 6: Random Forest Feature Importance 10 min | Python (Bonus)

Use a Random Forest to identify which features drive loan defaults, then visualize the importances.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

# Synthetic loan data: 1000 applicants, 6 features
n = 1000
loans = pd.DataFrame({
    'income':       np.random.normal(60_000, 20_000, n),
    'debt_ratio':   np.random.uniform(0.0, 1.0, n),
    'credit_score': np.random.normal(700, 80, n),
    'age':          np.random.normal(40, 12, n),
    'employ_yrs':   np.random.exponential(5, n),
    'dependents':   np.random.poisson(1.5, n),
})

# Target: default depends mostly on debt_ratio and credit_score
logits = (2.5 * loans['debt_ratio']
          - 0.01 * (loans['credit_score'] - 700)
          + 0.3 * np.random.randn(n))
loans['default'] = (logits > 0.5).astype(int)

X = loans.drop(columns='default')
y = loans['default']

# Step 1: Fit a Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X, y)

# Step 2: Extract feature importances
importances = pd.Series(rf.feature_importances_, index=X.columns)
importances = importances.sort_values(ascending=True)
print(importances.round(3))

# Step 3: Bar chart
importances.plot(kind='barh', color='#1e3a5f')
plt.xlabel('Feature importance')
plt.title('Random Forest feature importance')
plt.tight_layout()
plt.savefig('feature_importance.pdf', dpi=300)
```

(a) Run the code. Which two features does the Random Forest rank highest? Does this match the way the target was generated?

(b) Why do features like `dependents` or `age` end up with low importance even though the

Random Forest split on them many times during training?

(c) Feature importance in Random Forests is known to *inflate* the importance of high-cardinality numeric features (e.g., income, credit score) relative to low-cardinality ones (e.g., dependents). What alternative method (introduced in L26 or L32) would give a fairer ranking?