信息与软件工程学院上机实验报告

（第 2 次）

一、实验名称

利用深度神经网络进行房价预测

二、实验目的及要求

1. 掌握深度神经网络在回归任务中的基本实现方法。
2. 掌握表格数据的预处理流程，包括缺失值处理、类别编码与数值标准化。
3. 理解 K 折交叉验证、对数误差评价指标及 Kaggle 提交流程。

三、实验环境

1. Python 3
2. Anaconda
3. PyTorch

四、实验内容

1. 参照上课例子完成 California 房价预测代码
2. 提交 Kaggle 评价自己的算法结果。

五、实验设计内容要求

1. 实验步骤

本实验的目标是使用神经网络完成加州房价预测。原始数据同时包含数值字段、类别字段、日期字符串以及房源简介等混合类型信息，因此程序不能直接将原始表输入模型，而必须先进行特征工程和统一的数值化处理。整体实现遵循“读取数据 → 构造特征 → 建立预处理器 → K 折训练 → 生成提交结果”的顺序展开。

1.1 设计思想

本实验采用较为基础的多层感知机模型，重点在于数据预处理和验证流程的规范性。

程序首先对原始数据进行特征构造。对于日期列，提取年份、月份以及挂牌时间与上次成交时间的间隔；对于文本列，保留简介长度这一简单统计特征；对于面积、价格和税额等字段，构造单位面积挂牌价和税率等比值特征。随后删除不适合直接进入模型的编号列、地址列、原始长文本以及已经拆分完成的日期字符串列。

在预处理阶段，程序自动区分数值列与类别列。数值列采用“中位数填补 + 标准化”的方式处理，类别列采用“缺失值填补 + One-Hot 编码”的方式处理，并通过设置低频类别阈值将罕见类别合并，从而避免维度过度膨胀。模型训练阶段使用 5 折交叉验证，在每一折中仅用训练折拟合预处理器，以避免信息泄漏。

1.2 实现流程

实验整体流程如图所示。

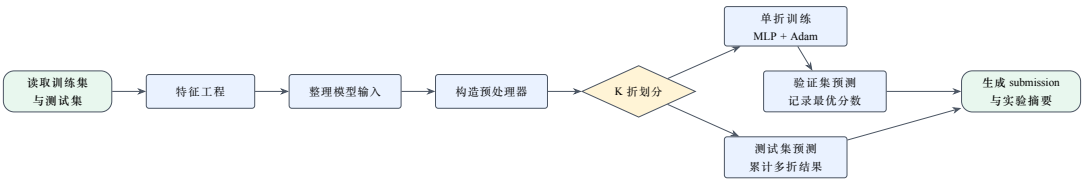


图 1: 实验二程序流程图

1.3 主要实现过程

程序的核心实现可以概括为以下几个步骤。

1. 使用 `pandas.read_csv()` 读取训练数据与测试数据，对应文件分别为 `california_train.csv` 和 `california_test.csv`。
2. 在 `add_features()` 中完成特征构造，包括卧室数量提取、日期拆分、房龄、简介长度、单位面积挂牌价和税率等。
3. 在 `build_model_input()` 中删除 `Id`、`Address`、`Summary`、原始日期列以及训练集中的标签列 `Sold Price`。

4. 在 `build_preprocessor()` 中建立预处理器：

   (a) 数值列：中位数填补缺失值，再做标准化；

   (b) 类别列：使用 `Missing` 填补缺失值，再做 One-Hot 编码；

   (c) 通过 `min_frequency=100` 合并低频类别。

5. 在 `make_fold_data()` 中，将训练折、验证折和测试集统一转换为数值矩阵，并将标签做 $\log(1+y)$ 变换。

6. 使用多层感知机作为回归模型，其结构为“输入层 $\to 256 \to 64 \to 1$”，激活函数为 ReLU，并在第一隐藏层后加入 `Dropout(0.2)`。

7. 使用 Adam 优化器和均方误差损失函数进行训练，并采用早停策略保存单折验证集分数最优的模型参数。

8. 在 `run_k_fold()` 中执行 5 折交叉验证，记录每一折验证分数，累加测试集预测结果，并最终输出平均预测。

本实验采用对数空间下的 RMSE 作为主要评价指标。设真实房价为 y_i，预测房价为 $\hat{y}_i$，则对数 RMSE 定义为：

$$\text{RMSE}_{\log} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log(1+\hat{y}_i) - \log(1+y_i)\right)^2}$$

由于程序中模型直接预测的是 $\log(1+y)$，因此实际计算时可写为：

$$\text{RMSE}_{\log} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(pred_i - target_i)^2}$$

其中 $pred_i$ 表示模型输出的对数房价，$target_i$ 表示真实房价的对数值。之所以采用该指标，是因为房价分布具有明显长尾特征，对标签做对数变换后，训练过程更加稳定，同时也与 Kaggle 评测方式保持一致。

设第 k 折对应测试集上的预测结果为 $\hat{y}_{test}^{(k)}$，则最终提交时使用 5 折平均结果：

$$\hat{y}_{test} = \frac{1}{K}\sum_{k=1}^{K}\hat{y}_{test}^{(k)}, \quad K = 5$$

2. 调试过程及实验结果

2.1 调试重点

实验调试过程中，主要集中在以下几个方面。

第一，原始数据中的字段类型较为复杂。部分列本应表示数值，但实际同时混有文本信息，例如 Bedrooms 字段既可能出现具体数量，也可能出现文字描述。为此，程序单独从该列中提取可解析的数值，形成 Bedroom count 新特征，同时保留原始列用于类别编码。

第二，类别字段数量较多，且部分字段具有较高基数。如果直接对全部类别值进行完整展开，会造成特征维度迅速增长，既增加计算量，也容易引入稀疏噪声。针对这一问题，程序删除了地址、原始简介等不适合直接编码的字段，并通过 OneHotEncoder(handle_unknown="infrequent_if_exist", min_frequency=100) 自动合并低频类别，有效控制了编码后的维度规模。

第三，多个连续值字段存在明显偏态分布，例如地块面积、室内面积、评估价、税额和挂牌价等。如果直接以原始尺度进行训练，梯度波动较大，模型收敛不够稳定。因此程序对这些高偏态字段以及标签 Sold Price 使用 $\log(1+x)$ 变换，以减轻长尾分布带来的影响。

第四，交叉验证过程中必须避免信息泄漏。若在整份训练集上拟合预处理器，再去处理某一折的验证集，则验证集信息已经间接参与训练。程序在 make_fold_data() 中严格限制预处理器只在当前训练折上 fit，再对验证折和测试集执行相同的 transform，从而保证验证结果具有参考意义。

第五，模型训练后期可能出现验证误差不再下降的情况。为减少无效训练，程序设置了 patience=5 的早停策略。当验证集分数连续 5 轮未改善时，训练提前结束，并恢复到验证集表现最好的模型状态。

2.2 实验结果

本实验的主要训练配置如下表所示。

表 1: 模型训练参数

参数项	取值
随机种子	42
K 折数	5
最大训练轮数	25
早停轮数	5
Batch Size	1024
学习率	0.002
Weight Decay	0.0001
低频类别阈值	100

5 折交叉验证结果如下表所示。

表 2: 5 折验证集 log-rmse 结果

折数	验证集 log-rmse
Fold 1	0.23275
Fold 2	0.24392
Fold 3	0.27228
Fold 4	0.22840
Fold 5	0.23461
平均值	0.24239

整体 OOF 分数为:

$$\text{OOF log-rmse} = 0.24291$$

各折验证结果可视化如图所示。

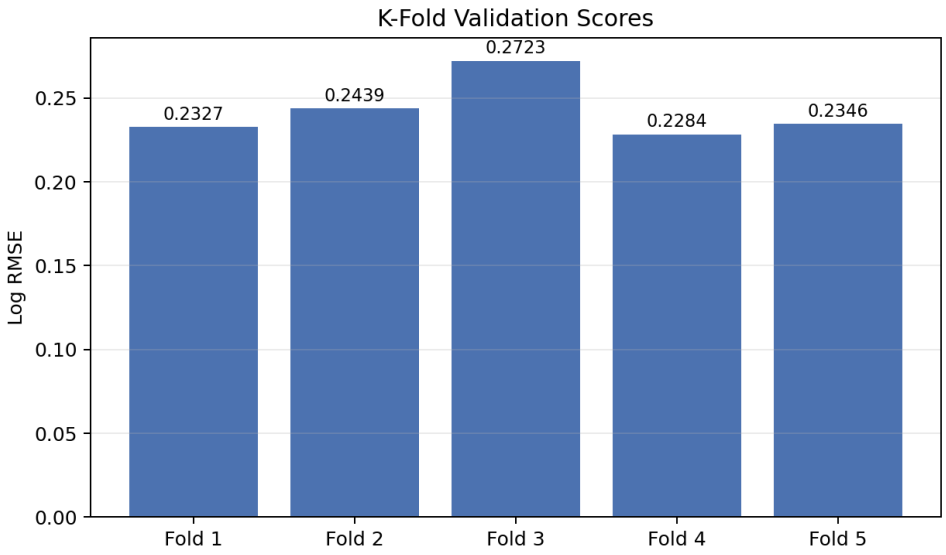


图 2: 5 折交叉验证分数统计图

此外，程序利用 Kaggle API 提交了 `submission.csv`，得到的线上评测结果如下图所示。

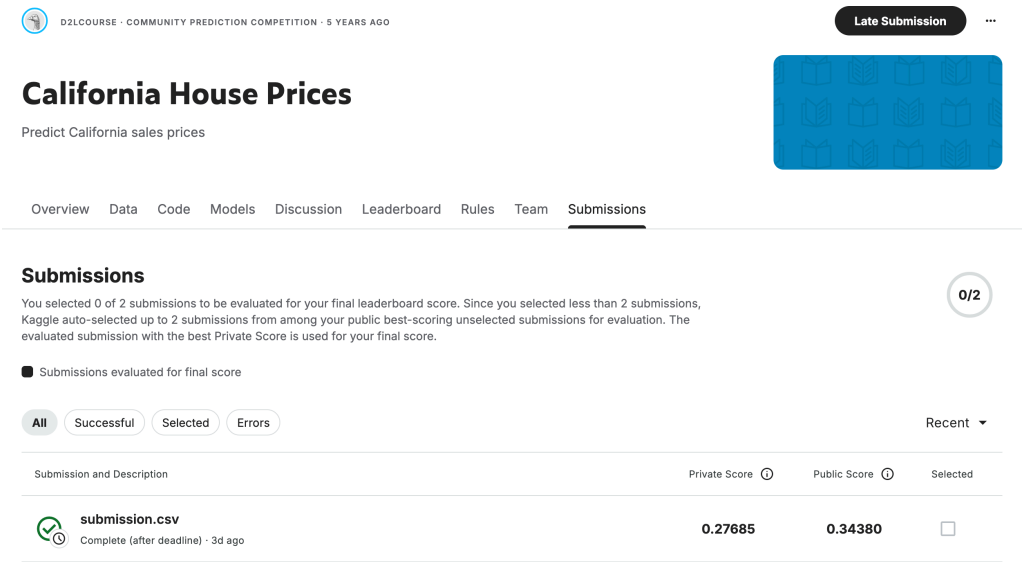


图 3: Kaggle 提交结果

2.3 结果分析

从交叉验证结果可以看出，5 折分数整体较为接近，说明模型在不同数据划分上的表现较稳定。其中第 3 折的误差略高于其他折，说明这一折的验证样本分布相对更复杂，但整体波动幅度不大，因此训练结果仍具有较好的可信度。

本地 5 折平均 log-rmse 为 0.24239，OOF 分数为 0.24291，两者十分接近，说明交叉验证评估较为稳定，没有出现明显偏离。Kaggle 公榜得分为 0.34380，私榜得分为 0.27685，与本地 OOF 分数相对接近，说明当前验证方式对线上表现具有一定参考价值。

从模型复杂度看，本实验所采用的网络结构仍属于较基础的多层感知机，没有使用特别复杂的深层网络、特征交叉网络或模型集成方法。实验结果表明，在表格回归问题中，只要预处理过程合理、特征工程有效、验证流程规范，即使使用较基础的神经网络模型，也能够取得较好的预测效果。

3. **总结**

通过本次实验，我对神经网络在表格回归任务中的完整实现流程有了较系统的认识。与实验一相比，本实验直接处理真实房价数据，数据规模更大、字段类型更复杂，因此也更能体现数据预处理的重要性。实验结果表明，在该类任务中，缺失值处理、类别编码、数值尺度变换和交叉验证策略都会直接影响模型最终性能。

在模型设计上，本实验没有刻意追求复杂结构，而是采用较为基础的多层感知机作为预测模型。通过合理构造输入特征、规范处理训练折与验证折之间的关系，并在标签空间上采用对数变换，最终得到了较稳定的本地分数和较好的 Kaggle 线上成绩。这说明对于初学阶段的回归任务学习而言，先掌握完整而规范的建模流程，比单纯追求更复杂的模型更为重要。

本次实验的主要收获包括：理解了混合类型表格数据的基本处理方法；掌握了使用 `ColumnTransformer` 与 `Pipeline` 构建预处理流程的方法；理解了 K 折交叉验证、OOF 预测和多折平均提交的意义；掌握了使用 PyTorch 构建并训练多层感知机回归模型的基本步骤。

若进一步改进本实验，还可以从以下几个方面继续展开：一是对高基数类别尝试嵌入表示或目标编码；二是对简介文本引入更细致的统计特征或文本向量化方法；三是继续调整网络宽度、正则化系数与学习率等超参数，以进一步优化线上表现。

4. 附录

完整源代码如下。

```python
from __future__ import annotations

import json
import random
from pathlib import Path

import numpy as np
import pandas as pd
import torch
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from torch import nn
from torch.utils.data import DataLoader, TensorDataset


# ============================================================
# Step 0: 集中定义实验配置
# ============================================================
# 便于统一调整训练参数与输入输出路径。

ROOT_DIR = Path(__file__).resolve().parents[1]
DATA_DIR = ROOT_DIR / "data"
OUTPUT_DIR = ROOT_DIR / "output"

TRAIN_CSV = DATA_DIR / "california_train.csv"
TEST_CSV = DATA_DIR / "california_test.csv"

SEED = 42
K_FOLDS = 5
NUM_EPOCHS = 25
PATIENCE = 5
BATCH_SIZE = 1024
LEARNING_RATE = 2e-3
WEIGHT_DECAY = 1e-4

# 控制低频类别合并阈值，避免 one-hot 维度过度膨胀。
RARE_MIN_FREQUENCY = 100

```

```python

# ============================================================
# Step 1: 固定随机种子
# ============================================================
def set_seed(seed: int) -> None:
    """
    固定随机种子，提升实验复现性。
    """
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)


def get_device() -> torch.device:
    """
    自动选择训练设备。

    优先级：CUDA -> MPS -> CPU。
    """
    if torch.cuda.is_available():
        return torch.device("cuda")
    if torch.backends.mps.is_available():
        return torch.device("mps")
    return torch.device("cpu")


# ============================================================
# Step 2: 特征工程
# ============================================================
# 原始表中包含混合格式字段、日期字符串以及强偏态数值，
# 需要先转为更适合模型学习的表示。


def add_features(df: pd.DataFrame) -> pd.DataFrame:
    """
    在原始表基础上构造少量有效特征。
    """
    df = df.copy()

    # ------------------------------------------------------------
    # 2.1 提取卧室数量
    # ------------------------------------------------------------
    # Bedrooms 同时包含数值与文本描述，因此额外提取数值列。
```

```
    df["Bedroom count"] = pd.to_numeric(
        df["Bedrooms"].astype(str).str.extract(r"(\d+(?:\.\d+)?)")[0],
        errors="coerce",
    )

    # ------------------------------------------------------------
    # 2.2 处理日期列
    # ------------------------------------------------------------
    # 日期字符串拆分为数值特征。
    listed_on = pd.to_datetime(df["Listed On"], errors="coerce")
    last_sold_on = pd.to_datetime(df["Last Sold On"], errors="coerce")

    df["Listed year"] = listed_on.dt.year.astype("float64")
    df["Listed month"] = listed_on.dt.month.astype("float64")
    df["Last sold year"] = last_sold_on.dt.year.astype("float64")
    df["Last sold month"] = last_sold_on.dt.month.astype("float64")

    # 记录挂牌时间与上次成交时间间隔。
    df["Days since last sold"] = (listed_on - last_sold_on).dt.days.astype("float64")

    # ------------------------------------------------------------
    # 2.3 构造人工特征
    # ------------------------------------------------------------
    # 房龄
    df["House age"] = df["Listed year"] - pd.to_numeric(df["Year built"], errors="coerce")

    # 保留房源简介长度信息。
    df["Summary length"] = df["Summary"].fillna("").str.len().astype("float64")

    # 构造常见比率特征。
    total_area = pd.to_numeric(df["Total interior livable area"], errors="coerce").replace(0, np.nan)
    listed_price = pd.to_numeric(df["Listed Price"], errors="coerce")
    tax_value = pd.to_numeric(df["Tax assessed value"], errors="coerce").replace(0, np.nan)
    annual_tax = pd.to_numeric(df["Annual tax amount"], errors="coerce")

```

```
    # 单位面积挂牌价
    df["Price per sqft"] = listed_price / total_area

    # 税额与评估价比值
    df["Tax rate"] = annual_tax / tax_value

    # ----------------------------------------------------------
    # 2.4 显式转换数值列
    # ----------------------------------------------------------
    # 避免因异常值导致整列被识别为字符串。
    numeric_columns = [
        "Year built",
        "Lot",
        "Bathrooms",
        "Full bathrooms",
        "Total interior livable area",
        "Total spaces",
        "Garage spaces",
        "Elementary School Score",
        "Elementary School Distance",
        "Middle School Score",
        "Middle School Distance",
        "High School Score",
        "High School Distance",
        "Tax assessed value",
        "Annual tax amount",
        "Listed Price",
        "Last Sold Price",
        "Bedroom count",
        "Listed year",
        "Listed month",
        "Last sold year",
        "Last sold month",
        "Days since last sold",
        "House age",
        "Summary length",
        "Price per sqft",
        "Tax rate",
    ]

    for column in numeric_columns:
        if column in df.columns:
            df[column] = pd.to_numeric(df[column], errors="coerce
    ")

```

```
    # ------------------------------------------------------------
    # 2.5 对高偏态连续值做对数变换
    # ------------------------------------------------------------
    # 价格、面积、税额等特征通常存在显著长尾分布。
    log_columns = [
        "Lot",
        "Total interior livable area",
        "Tax assessed value",
        "Annual tax amount",
        "Listed Price",
        "Last Sold Price",
        "Price per sqft",
    ]

    for column in log_columns:
        if column in df.columns:
            df[column] = pd.to_numeric(df[column], errors="coerce").clip(lower=0)
            df[column] = np.log1p(df[column])

    # ------------------------------------------------------------
    # 2.6 将 Zip 视为类别变量
    # ------------------------------------------------------------
    # 邮编不具备连续数值的顺序含义。
    if "Zip" in df.columns:
        df["Zip"] = df["Zip"].astype("Int64").astype("string")

    return df


# ============================================================
# Step 3: 整理模型输入
# ============================================================
def build_model_input(df: pd.DataFrame, is_train: bool) -> pd.DataFrame:
    """
    将原始 DataFrame 整理为模型输入表。
    """
    df = add_features(df)

    # 移除编号列、原始文本列和已拆分完成的日期列。
    drop_columns = ["Id", "Address", "Summary", "Listed On", "Last Sold On"]

    if is_train:
```

```
        drop_columns.append("Sold Price")

    return df.drop(columns=drop_columns)


# ============================================================
# Step 4: 构造预处理器
# ============================================================
# 将混合特征表转换为模型可直接使用的数值矩阵。


def build_preprocessor(train_x_df: pd.DataFrame) -> ColumnTransformer:
    """
    根据训练折自动识别数值列与类别列，并分别处理。
    """
    categorical_columns = train_x_df.select_dtypes(include=["object", "string"]).columns.tolist()
    numeric_columns = [column for column in train_x_df.columns if column not in categorical_columns]

    numeric_pipeline = Pipeline(
        [
            ("imputer", SimpleImputer(strategy="median")),
            ("scaler", StandardScaler()),
        ]
    )

    categorical_pipeline = Pipeline(
        [
            ("imputer", SimpleImputer(strategy="constant", fill_value="Missing")),
            (
                "onehot",
                OneHotEncoder(
                    handle_unknown="infrequent_if_exist",
                    min_frequency=RARE_MIN_FREQUENCY,
                    sparse_output=False,
                ),
            ),
        ]
    )

    preprocessor = ColumnTransformer(
        [
```

```
            ("num", numeric_pipeline, numeric_columns),
            ("cat", categorical_pipeline, categorical_columns),
        ]
    )
    return preprocessor


# ============================================================
# Step 5: 生成训练矩阵与标签
# ============================================================
def make_fold_data(
    train_df: pd.DataFrame,
    valid_df: pd.DataFrame,
    test_df: pd.DataFrame,
) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    """
    将训练折、验证折和测试集统一转换为数值矩阵。
    预处理器仅在训练折上拟合，以避免信息泄漏。
    """
    train_x_df = build_model_input(train_df, is_train=True)
    valid_x_df = build_model_input(valid_df, is_train=True)
    test_x_df = build_model_input(test_df, is_train=False)

    preprocessor = build_preprocessor(train_x_df)

    train_x = preprocessor.fit_transform(train_x_df).astype(np.float32)
    valid_x = preprocessor.transform(valid_x_df).astype(np.float32)
    test_x = preprocessor.transform(test_x_df).astype(np.float32)

    # 目标值采用对数变换，缓解房价分布的长尾问题。
    train_y = np.log1p(train_df["Sold Price"].to_numpy()).astype(np.float32)
    valid_y = np.log1p(valid_df["Sold Price"].to_numpy()).astype(np.float32)

    return train_x, train_y, valid_x, valid_y, test_x


# ============================================================
# Step 6: 定义神经网络
# ============================================================
# 采用结构较简单的多层感知机，兼顾可读性与预测效果。
```

```
289
290
291 class MLP(nn.Module):
292     def __init__(self, input_dim: int) -> None:
293         super().__init__()
294
295         self.network = nn.Sequential(
296             nn.Linear(input_dim, 256),
297             nn.ReLU(),
298             nn.Dropout(0.2),
299             nn.Linear(256, 64),
300             nn.ReLU(),
301             nn.Linear(64, 1),
302         )
303
304     def forward(self, x: torch.Tensor) -> torch.Tensor:
305         return self.network(x).squeeze(1)
306
307
308 def log_rmse(pred: torch.Tensor, target: torch.Tensor) -> float:
309     return float(torch.sqrt(torch.mean((pred - target) ** 2)).
    item())
310
311
312 # ================================================================
313 # Step 7: 训练单折模型
314 # ================================================================
315 def train_one_fold(
316     train_x: np.ndarray,
317     train_y: np.ndarray,
318     valid_x: np.ndarray,
319     valid_y: np.ndarray,
320     device: torch.device,
321 ) -> tuple[MLP, float]:
322     """
323     训练单折模型，并返回最优模型及对应验证分数。
324     """
325     train_loader = DataLoader(
326         TensorDataset(torch.tensor(train_x), torch.tensor(train_y
    )),
327         batch_size=BATCH_SIZE,
328         shuffle=True,
329     )
330
331     valid_x_tensor = torch.tensor(valid_x, device=device)
```

```
    valid_y_tensor = torch.tensor(valid_y, device=device)

    model = MLP(train_x.shape[1]).to(device)
    optimizer = torch.optim.Adam(
        model.parameters(),
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY,
    )
    loss_fn = nn.MSELoss()

    best_score = float("inf")
    best_state: dict[str, torch.Tensor] | None = None
    wait = 0

    for epoch in range(1, NUM_EPOCHS + 1):
        # 训练阶段
        model.train()
        for batch_x, batch_y in train_loader:
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device)

            optimizer.zero_grad()
            pred = model(batch_x)
            loss = loss_fn(pred, batch_y)
            loss.backward()
            optimizer.step()

        # 验证阶段
        model.eval()
        with torch.no_grad():
            valid_pred = model(valid_x_tensor)
            valid_score = log_rmse(valid_pred, valid_y_tensor)

        # 控制日志输出频率。
        if epoch <= 3 or epoch % 5 == 0 or epoch == NUM_EPOCHS:
            print(f"epoch {epoch:02d} | valid log-rmse = {
    valid_score:.5f}")

        # 早停策略：连续若干轮无提升则终止训练。
        if valid_score < best_score:
            best_score = valid_score
            best_state = {key: value.detach().cpu().clone() for
    key, value in model.state_dict().items()}
            wait = 0
        else:
```

```
            wait += 1

        if wait >= PATIENCE:
            print(f"验证集连续 {PATIENCE} 轮没有提升，提前停止。")
            break

    if best_state is None:
        raise RuntimeError("训练过程中没有保存到最优模型。")

    model.load_state_dict(best_state)
    return model, best_score


# ============================================================
# Step 8: 预测
# ============================================================
@torch.no_grad()
def predict(model: MLP, features: np.ndarray, device: torch.device) -> np.ndarray:
    """
    按批次执行预测，降低显存占用。
    """
    model.eval()
    loader = DataLoader(torch.tensor(features), batch_size=BATCH_SIZE, shuffle=False)

    preds = []
    for batch_x in loader:
        batch_x = batch_x.to(device)
        preds.append(model(batch_x).cpu().numpy())

    return np.concatenate(preds, axis=0)


# ============================================================
# Step 9: K 折训练与测试集预测
# ============================================================
def run_k_fold(
    train_df: pd.DataFrame,
    test_df: pd.DataFrame,
    device: torch.device,
) -> tuple[np.ndarray, list[float], float]:
    """
    执行 K 折训练，并对测试集预测结果取平均。
    """
```

```
418     kfold = KFold(n_splits=K_FOLDS, shuffle=True, random_state=
    SEED)
419 
420     # OOF 预测用于估计整体泛化误差。
421     oof_pred = np.zeros(len(train_df), dtype=np.float32)
422 
423     # 测试集预测采用多折平均。
424     test_pred_sum = np.zeros(len(test_df), dtype=np.float32)
425 
426     fold_scores: list[float] = []
427 
428     for fold, (train_index, valid_index) in enumerate(kfold.split
    (train_df), start=1):
429         print(f"\n===== Fold {fold} / {K_FOLDS} =====")
430 
431         fold_train_df = train_df.iloc[train_index].reset_index(
    drop=True)
432         fold_valid_df = train_df.iloc[valid_index].reset_index(
    drop=True)
433 
434         train_x, train_y, valid_x, valid_y, test_x =
    make_fold_data(
435             fold_train_df,
436             fold_valid_df,
437             test_df,
438         )
439 
440         print(f"训练矩阵形状: {train_x.shape}")
441         print(f"验证矩阵形状: {valid_x.shape}")
442 
443         model, best_score = train_one_fold(
444             train_x=train_x,
445             train_y=train_y,
446             valid_x=valid_x,
447             valid_y=valid_y,
448             device=device,
449         )
450 
451         valid_pred = predict(model, valid_x, device)
452         test_pred = predict(model, test_x, device)
453 
454         oof_pred[valid_index] = valid_pred
455         test_pred_sum += test_pred
456         fold_scores.append(best_score)
457 
```

```
458             print(f"Fold {fold} best log-rmse: {best_score:.5f}")
459
460         mean_test_pred = test_pred_sum / K_FOLDS
461         true_log_price = np.log1p(train_df["Sold Price"].to_numpy())
462         oof_score = float(np.sqrt(np.mean((oof_pred - true_log_price)
            ** 2)))
463         return mean_test_pred, fold_scores, oof_score
464
465
466 # ============================================================
467 # Step 10: 保存结果
468 # ============================================================
469 def save_results(test_df: pd.DataFrame, test_pred_log: np.ndarray
        , fold_scores: list[float], oof_score: float) -> None:
470     OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
471
472     # 将对数预测值还原为真实房价。
473     predicted_price = np.expm1(test_pred_log)
474     predicted_price = np.clip(predicted_price, a_min=1.0, a_max=
        None)
475
476     submission = pd.DataFrame(
477         {
478             "Id": test_df["Id"],
479             "Sold Price": predicted_price,
480         }
481     )
482     submission_path = OUTPUT_DIR / "submission.csv"
483     submission.to_csv(submission_path, index=False)
484
485     summary = {
486         "mean_fold_log_rmse": float(np.mean(fold_scores)),
487         "oof_log_rmse": oof_score,
488         "fold_scores": fold_scores,
489         "config": {
490             "seed": SEED,
491             "k_folds": K_FOLDS,
492             "num_epochs": NUM_EPOCHS,
493             "patience": PATIENCE,
494             "batch_size": BATCH_SIZE,
495             "learning_rate": LEARNING_RATE,
496             "weight_decay": WEIGHT_DECAY,
497             "rare_min_frequency": RARE_MIN_FREQUENCY,
498         },
499     }
```

```
    summary_path = OUTPUT_DIR / "cv_summary.json"
    with summary_path.open("w", encoding="utf-8") as f:
        json.dump(summary, f, ensure_ascii=False, indent=2)

    print("\n训练完成。")
    print(f"平均验证 log-rmse: {summary['mean_fold_log_rmse']:.5f}")
    print(f"OOF log-rmse: {summary['oof_log_rmse']:.5f}")
    print(f"提交文件已保存到: {submission_path}")
    print(f"实验摘要已保存到: {summary_path}")


# ============================================================
# Step 11: 主流程
# ============================================================
def main() -> None:
    set_seed(SEED)
    device = get_device()

    # 加载数据
    train_df = pd.read_csv(TRAIN_CSV)
    test_df = pd.read_csv(TEST_CSV)

    print(f"训练集形状: {train_df.shape}")
    print(f"测试集形状: {test_df.shape}")
    print(f"当前设备: {device}")

    # 执行 K 折训练与测试集预测。
    test_pred_log, fold_scores, oof_score = run_k_fold(
        train_df=train_df,
        test_df=test_df,
        device=device,
    )

    save_results(
        test_df=test_df,
        test_pred_log=test_pred_log,
        fold_scores=fold_scores,
        oof_score=oof_score,
    )


if __name__ == "__main__":
    main()
```