DnA1

AMAZON 제품 가격 예측 모델

202210888 허현승
202310861 이선역
202410777 심민진
202510729 손태영

목차...

01 전처리

피처 엔지니어링 | 인코딩&수치형 변수 정리 | SOTA 임베딩+PCA+클러스터링

1.1. 기본 전처리

```
print("1. 데이터 로드 및 기본 전처리...")
df_train = pd.read_csv('/content/drive/MyDrive/amazonbamboo01/amazonproduct_train.csv')
df_test = pd.read_csv('/content/drive/MyDrive/amazonbamboo01/amazonproduct_test.csv')
submission = pd.read_csv('/content/drive/MyDrive/amazonbamboo01/submission01.csv')
```

```
#가격 전처리: Selling Price를 문자열로 변환 -> $와 , 제거, 변환하지 않으면 NaN(결측치)로 바뀜
df_train['Selling Price'] = df_train['Selling Price'].astype(str).str.replace(r'[$,]', '',
regex=True)
df_train['Selling Price'] = pd.to_numeric(df_train['Selling Price'], errors='coerce')
df_train = df_train.dropna(subset=['Selling Price']) #가격이 NaN인 행 dropna로 제거
y_tr = df_train['Selling Price'].values #정답(Y) 벡터를 numpy 배열로 추출
```

```
# 결측치 채우기: 결측치를 처리할 텍스트 컬럼 목록 지정
text_cols = ['Category', 'Product Specification', 'Product Name', 'Description']
for col in text_cols:
  #train/test 둘 다 결측치는 문자열 "unKnown"으로 채움
    df_train[col] = df_train[col].fillna('Unknown').astype(str)
    df_test[col] = df_test[col].fillna('Unknown').astype(str)
#.astype(str)로 다양한 타입이 섞여 있어도 이후 텍스트 처리 함수가 안정적으로 동작
#-> 에러 방지, 일관성 유지, 모델이 결측을 학습
```

```
import pandas as pd
import numpy as np
import re
import joblib
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')
```

1.2. 전처리 - 피처 엔지니어링

```
print("2. 피처 엔지니어링 (One-Pass)...")

def process_all_features_in_one_pass(row):
  #각 행(row=상품 1개)을 입력으로 받아, 7개 피처를 만들어서 반환
    spec = str(row['Product Specification']) #스펙 텍스트
    name = str(row['Product Name']) #상품명
    # 소문자 버전도 제작해서 정규식 검색을 쉽게!
    spec_lower = spec.lower()
    name_lower = name.lower()
```

이후에 크게 두 단계로 구분!

수치 정보 추출	BERT용 텍스트 정보 구성
01. Item Weight (제품 무게)	Brand 추출
02. Shipping Weight (배송 무게)	스펙 텍스트 조립 (텍스트 정제)
03. Product Volume (부피)	
04. Age (연령)	
05. Pack(수량)	

1.2.1. 피처 엔지니어링 - 수치 정보 추출

*** 제품 무게와 배송 무게 -> 무게 추출 함수 사용**

무게 추출 함수

```
# 함수: 텍스트에서 숫자+단위 찾아서 lb(파운드)로 통일해 변환
    def extract_weight(text, pattern_prefix=''):
        w_lb = 0.0
        # 이걸로 item weight 뒤의 무게만 찾거나
        #shipping weight 뒤의 무게만 찾아 문맥 기반 추출을 할 수 있게 만듦
        regex_base = pattern_prefix + r':?\s*(\d+\.?\d*)\s*'

        m_lb = re.search(regex_base + r'(pound|lb)', text)
        m_oz = re.search(regex_base + r'(ounce|oz)', text)
        m_kg = re.search(regex_base + r'(kg|kilogram)', text)

        if m_lb: w_lb = float(m_lb.group(1))
        elif m_oz: w_lb = float(m_oz.group(1)) / 16.0
        elif m_kg: w_lb = float(m_kg.group(1)) * 2.20462
        return w_lb
```

제품 무게, 배송 무게 추출

```
# 1. Item Weight (제품 무게)
    #item weight 문맥에서 무게 찾고, 없으면 스펙 전체에서 일반 무게 패턴 다시 탐색
    #제품 자체 무게는 가격(재료,크기,배송비)과 상관 큼
    item_weight_lb = extract_weight(spec_lower, r'item\s*weight')
    if item_weight_lb == 0: # item weight 명시가 없으면 일반 패턴 검색
        item_weight_lb = extract_weight(spec_lower, r'')
```

```
# 2. Shipping Weight (배송 무게)
    #shipping weight 문맥에서만 무게 탐색
    #배송무게는 포장/박스 포함이라 가격과 또 다른 상관이 있을 수 있음
    ship_weight_lb = extract_weight(spec_lower, r'shipping\s*weight')
```

1.2.1. 피처 엔지니어링 - 수치 정보 추출

* 부피, 연령, 수량

```
# 3. Product Volume (크기 -> 부피 계산)
    #스펙에서 숫자x숫자x숫자 패턴을 찾아서 가로x세로x높이를 곱해 부피(상대적 크기)생성
    #크기가 크면 가격이 비쌀 확률이 높음(특히 장난감/가전/가구류)
    vol = 0.0
    dim_m = re.search(r'(\d+\.?\d*)\s*x\s*(\d+\.?\d*)\s*x\s*(\d+\.?\d*)', spec_lower)
    if dim_m:
        try:
            vol = float(dim_m.group(1)) * float(dim_m.group(2)) * float(dim_m.group(3))
        except:
            vol = 0.0
```

```
# 4. Age (연령)
    # (\d+) (year|yr|month) 패턴으로 찾음
    # month면 12로 나눠 years로 통일
    #유아용/아동용/성인용 구분에 도움, 가격대도 달라질 수 있음
    min_age = -1.0
    age_str = None
    age_m = re.search(r'(\d+)\s*(year|yr|month)', spec_lower)
    if age_m:
        val = float(age_m.group(1))
        unit = age_m.group(2)
        if 'month' in unit: min_age = val / 12.0
        else: min_age = val
        age_str = f"{min_age:.1f} years"
```

```
# 5. Pack (수량)
    #상품명에서 3pack, 10 pcs, 2 count 같은 패턴을 찾음
    #동일 제품도 수량이 많으면 가격이 올라가기 때문에 강한 피처
    pack_qty = 1.0 #묶음 수량
    pack_m = re.search(r'(\d+)\s*(pack|pk|pcs|set|count)', name_lower)
    if pack_m: pack_qty = float(pack_m.group(1))
```

1.2.2. 피처 엔지니어링 - BERT용 텍스트 정보 구성

```
extracted_parts = []
    brand_col = "Unknown"

    # Brand 추출
    # 스펙에서 Manufacturer: 또는 Brand: 뒤의 값을 찾으면 그걸 브랜드로 사용
    # 없으면 첫 상품명의 첫 단어를 브랜드로 추정
    #브랜드는 가격에 매우 큰 영향(프리미엄/저가)
    brand_match = re.search(r'(Manufacturer|Brand):?\s*([^|]+)', spec, re.I)
    if brand_match:
        found_brand = brand_match.group(2).strip()
        extracted_parts.append(f"Brand: {found_brand}")
        brand_col = found_brand
    else:
        if len(name) > 0: brand_col = name.split()[0]
```

```
# 스펙 텍스트 조립(텍스트 정제)
    # 위에서 뽑은 정보들을 사람이 읽기 좋은 요약 스펙 테스트로 만듦
    if item_weight_lb > 0: extracted_parts.append(f"Weight: {item_weight_lb:.2f} lb")
    if ship_weight_lb > 0: extracted_parts.append(f"Ship Weight: {ship_weight_lb:.2f} lb")
    if vol > 0: extracted_parts.append(f"Vol: {vol:.2f}")
    if age_str: extracted_parts.append(f"Age: {age_str}")

    cleaned_spec = " | ".join(extracted_parts) if extracted_parts else "Unknown"
    # -> | 기준으로 Brand: LEGO | Weight: 2.30 lb | Ship Weight: 2.80 lb | Vol: 120.00 | Age: 6.0
years

    # 반환값에 새로 만든 수치형 변수들(item_weight, ship_weight, vol) 추가
    return pd.Series([cleaned_spec, brand_col, min_age, item_weight_lb, ship_weight_lb, vol,
pack_qty],
                     index=['Cleaned_Spec', 'Brand', 'min_age', 'item_weight_lb',
'ship_weight_lb', 'product_vol', 'pack_qty'])
```

1.2.2. 피처 엔지니어링 - BERT용 텍스트 정보 구성

```
# 적용
#train/test의 각 행마다 함수 실행, 결과(7개 값)를 새로운 컬럼으로 저장
#이후 모델에서 숫자 피처: min_age, item_weight_lb, ship_weight_lb, product_vol, pack_qty
#범주 피처: Brand, 텍스트 피처/임베딩용으로는 Cleaned_Spec 활용
new_cols = ['Cleaned_Spec', 'Brand', 'min_age', 'item_weight_lb', 'ship_weight_lb', 'product_vol', 'pack_qty']
df_train[new_cols] = df_train.apply(process_all_features_in_one_pass, axis=1)
df_test[new_cols] = df_test.apply(process_all_features_in_one_pass, axis=1)
```

1.3. 인코딩 및 수치형 변수 정리

```
print("3. 인코딩 (Category Splitting & Label/Target Encoding)...")

# 1) 카테고리 세분화
'''
| 기준으로 3단계로 쪼개기
Main_cat = 1단계(대분류), 가격대가 크게 갈림
Sub_cat = 2단계(중분류), 소분류와 함께 세부 차이를 잡아줌(가격과 상관 있는 구조 더 잘 반영)
Deep_cat = 3단계(소분류)
'''
def split_category(df):
    split_data = df['Category'].str.split('|', n=2, expand=True)
    df['Main_Cat'] = split_data[0].fillna('Unknown').str.strip()
    df['Sub_Cat'] = split_data[1].fillna('Unknown').str.strip()
    df['Deep_Cat'] = split_data[2].fillna('Unknown').str.strip() if split_data.shape[1] > 2 else 'Unknown'
    return df

df_train = split_category(df_train)
df_test = split_category(df_test)
```

```
# 2) 레이블 인코딩
#문자열 카테고리를 정수ID로 변환#train에는 없고 test에만 있는 브랜드가 존재할 수 있으므로 train과 test를 합쳐 처음 보는 값 때문에 터지는 일 예방
print("   -> Label Encoding...")
label_cols = ['Brand', 'Main_Cat', 'Sub_Cat', 'Deep_Cat']
for col in label_cols:
    le = LabelEncoder()
    all_values = pd.concat([df_train[col], df_test[col]]).astype(str).unique()
    le.fit(all_values)
    df_train[f'{col}_label'] = le.transform(df_train[col].astype(str))
    df_test[f'{col}_label'] = le.transform(df_test[col].astype(str))
```

```
# 3) 타겟 인코딩 (Deep Cat 포함)
#각 범주마다 학습 데이터에서의 평균 가격(직접 관련된 숫자 특징)을 구해서 피처로 만듦
#ex) Brand = "LEGO" 인 행들의 평균 가격이 32.5라면 Brand_target_enc = 32.5
print("   -> Target Encoding (Adding Deep_Cat)...")
target_cols = ['Brand', 'Main_Cat', 'Sub_Cat', 'Deep_Cat'] # Deep_Cat 추가됨!
global_mean = y_tr.mean()

for col in target_cols:
    mean_map = df_train.groupby(col)['Selling Price'].mean()
    df_train[f'{col}_target_enc'] = df_train[col].map(mean_map)
    df_test[f'{col}_target_enc'] = df_test[col].map(mean_map).fillna(global_mean)
```

1.3. 인코딩 및 수치형 변수 정리

```
# 최종 수치형 데이터셋
'''
포함되는 피처 종류
정규식으로 뽑은 수치 피처: 권장 연령(년 단위), 묶음 수량, 제품 무게, 배송 무게, 부피
타겟 인코딩 피처(가격 평균 기반): Brand_target_enc, Main_Cat_target_enc, Sub_Cat_target_enc, Deep_Cat_target_enc
레이블 인코딩 피처(정수ID): Brand_label, Main_Cat_label, Sub_Cat_label, Deep_Cat_label

'''
final_num_cols = ['min_age', 'pack_qty', 'item_weight_lb', 'ship_weight_lb', 'product_vol'] + \
                 [f'{c}_target_enc' for c in target_cols] + \
                 [f'{c}_label' for c in label_cols]

print(f"   -> Numerical Features ({len(final_num_cols)}): {final_num_cols}")

# [중요] 나중에 병합을 위해 변수명 X_num_tr, X_num_te로 생성
#위에서 만든 숫자 피처들을 순서대로 뽑아 numpy array형태의 행렬로 변환
X_num_tr = df_train[final_num_cols].values
X_num_te = df_test[final_num_cols].values
```

1.4. SOTA임베딩 + PCA + 클러스터링

```
# ========================================
# 4. SOTA 임베딩 + PCA + 클러스터링
#원본 텍스트는 모델이 바로 못 먹음-> 숫자 벡터(임베딩)로 변환
#임베딩 768차원은 너무 큼 -> PCA로 64차원까지 압축
#비슷한 상품 텍스트끼리 그룹이 존재할 수 있음 -> KMeans로 클러스터 번호(1차원) 추가
#최종적으로 텍스트 1개당 65차원(=64PCA + 1 cluster) 피처가 만들어짐
# ========================================
```

```
print("4. SOTA 모델(all-mpnet-base-v2) 임베딩 및 차원축소...")
bert = SentenceTransformer('all-mpnet-base-v2') #문장/텍스트를 768차원 벡터로 변환
#텍스트 의미를 숫자 공간으로 옮겨서 비슷한 텍스트끼리 가까운 벡터에 위치하게 함

def get_advanced_emb(text_list_tr, text_list_te, n_comp, n_cluster):
  #train 텍스트 리스트& test 텍스트 리스트를 받아
  #각각을 임베딩+PCA+클러스터한 결과 반환
    #임베딩 생성
    emb_tr = bert.encode(text_list_tr, show_progress_bar=True, batch_size=64)
    emb_te = bert.encode(text_list_te, show_progress_bar=True, batch_size=64)
    #PCA로 차원 축소
    pca = PCA(n_components=n_comp, random_state=42)
    pca_tr = pca.fit_transform(emb_tr)
    pca_te = pca.transform(emb_te)
  #KMeans로 군집(클러스터) 생성
  #PCA된 64차원 벡터들을 비슷한 것끼리 20개 그룹으로 묶음
    kmeans = KMeans(n_clusters=n_cluster, random_state=42, n_init=10)
    cluster_tr = kmeans.fit_predict(pca_tr).reshape(-1, 1)
    cluster_te = kmeans.predict(pca_te).reshape(-1, 1)
  #PCA 64차원+ 클러스터 1차원 결합해서 반환
    return np.hstack([pca_tr, cluster_tr]), np.hstack([pca_te, cluster_te])
```

1.4. SOTA임베딩 + PCA + 클러스터링

```
# 1. Product Name
#상품명 텍스트 기반 픽처, 가격과 강한 상관
#브랜드/모델명/세트/구성 등이 상품명에 있기 때문
print("    -> Processing Product Name...")
tr_n, te_n = get_advanced_emb(df_train['Product
Name'].tolist(),
                              df_test['Product
Name'].tolist(), 64, 20)
```

```
# 2. Description
#상세 설명 기반 피처, 품질/구성/재질 같은 가격 결정
단서 포함
print("    -> Processing Description...")
tr_d, te_d =
get_advanced_emb(df_train['Description'].to
list(),

df_test['Description'].tolist(), 64, 20)
```

```
# 3. Category + Cleaned Spec
#카테고리 문장 자체 임베딩, 카테고리 의미 유사성을
숫자로 반영
print("    -> Processing Category only...")
tr_c, te_c =
get_advanced_emb(df_train['Category'].tolis
t(),

df_test['Category'].tolist(),

n_comp=64, n_cluster=20)
```

```
#Cleand_Spec
#정규식으로 정제해둔 스펙 요약 텍스트
#원본 스펙보다 노이즈가 적어 임베딩 성능이 더 잘 나올
가능성이 큼.
print("    -> Processing Cleaned_Spec only...")
tr_s, te_s =
get_advanced_emb(df_train['Cleaned_Spec'].tolist(
),
df_test['Cleaned_Spec'].tolist(),
                                    n_comp=64,
n_cluster=20)
```

1.5. 저장

```
print("5. 데이터 병합 및 저장...")

# [수정] tr_manual -> X_num_tr, te_manual ->
X_num_te (변수명 매칭)
X_tr = np.hstack([tr_n, tr_d, tr_c, tr_s,
X_num_tr])
X_test = np.hstack([te_n, te_d, te_c, te_s,
X_num_te])

print(f"최종 피처 개수: {X_tr.shape[1]}")

package = {
    'X_tr': X_tr,
    'y_tr': y_tr,
    'X_test': X_test,
    'submission': submission,
    'feature_names': num_cols + ['emb_pca_...']
}

# 경로는 사용자 환경에 맞게 유지
save_path =
"/content/drive/MyDrive/amazonbamboo01/data7.pkl"
joblib.dump(package, save_path, compress=3)
print(f"데이터셋 저장 완료: {save_path}")
```

02 모델

LightGBM

2. 모델 - LightGBM

```
import lightgbm as lgb
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
import numpy as np
import pandas as pd
import joblib
import os

# ==========================================================
# 0. 데이터 로드 (NameError 방지)
# ==========================================================
if 'X_tr' not in globals():
    try:
        print("📦 'data6.pkl'에서 데이터를 불러오는 중...")
        data = joblib.load("data6.pkl")
        X_tr, y_tr = data['X_tr'], data['y_tr']
        X_test, submission = data['X_test'],
data['submission']
        print(f"✅ 데이터 로드 성공! (X_tr shape:
{X_tr.shape})")
    except Exception as e:
        print(f"❌ 데이터 로드 실패: {e}. 'data6.pkl' 파일이
있는지 확인해 주세요.")
```

```
import lightgbm as lgb
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
import numpy as np
import pandas as pd

# ==========================================================
# 1. 사용자 정의 평가 함수 (Rounded MAE)
# ==========================================================
def eval_rounded_mae_lgb(preds, train_data):
    labels = train_data.get_label()
    preds = np.round(np.expm1(preds), 2)
    actual = np.expm1(labels)
    return 'rounded_mae', mean_absolute_error(actual, preds),
False
```

2. 모델 - LightGBM

```python
def run_lgbm_final(X, y, X_test, submission_df, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

    # 결과 저장용 배열
    oof_preds = np.zeros(X.shape[0])
    test_preds = np.zeros(X_test.shape[0])
    scores = []
    print(f"\n🔥 LightGBM 학습 시작 (Log Scale + Rounded MAE Evaluation)")

    # 파라미터 설정 (Metric을 None으로 설정하여 커스텀 함수만 출력)
    #하이퍼 파라미터는 인간이 직접 노가다ㅠㅠ
    params = {
        'objective': 'regression',
        'metric': 'None',
        'boosting_type': 'gbdt',
        'num_leaves': 64,
        'learning_rate': 0.008,
        'feature_fraction': 0.7,
        'bagging_fraction': 0.8,
        'bagging_freq': 3,
        'verbose': -1,
        'n_jobs': -1,
        'random_state': 42
    }
    # Log 변환 (타겟)
    y_log = np.log1p(y)
```

```python
# 피처 중요도 출력을 위한 이름 리스트
    feat_names = [f'Feature {i}' for i in range(X.shape[1])]

    for fold, (train_idx, val_idx) in enumerate(kf.split(X, y)):
        # 데이터 분할 (get_fold_data가 없으면 기본 슬라이싱 사용)
        try:
            X_t, X_v, y_t, y_v = get_fold_data(X, y_log, train_idx, val_idx)
        except NameError:
            X_t, X_v = X[train_idx], X[val_idx]
            y_t, y_v = y_log[train_idx], y_log[val_idx]
               train_ds = lgb.Dataset(X_t, label=y_t)
        valid_ds = lgb.Dataset(X_v, label=y_v, reference=train_ds)
        # 모델 학습
        model = lgb.train(
            params,
            train_ds,
            num_boost_round=100000,
            valid_sets=[valid_ds],
            feval=eval_rounded_mae_lgb,  # [핵심] 커스텀 평가 함수 적용
            callbacks=[
                lgb.early_stopping(2000, verbose=False),
                lgb.log_evaluation(1000)
            ]
        )
    # 예측 및 역변환
        val_pred_log = model.predict(X_v)
        test_pred_log = model.predict(X_test)
        val_pred = np.expm1(val_pred_log)
        test_pred = np.expm1(test_pred_log)
        oof_preds[val_idx] = val_pred
        test_preds += test_pred / n_splits
```

2. 모델 - LightGBM + 후처리

```
# 점수 계산 (반올림 적용된 점수로 기록)
        # eval_rounded_mae_lgb에서 계산된 최적 점수는
model.best_score에 있음
        best_score =
model.best_score['valid_0']['rounded_mae']
        scores.append(best_score)

        print(f"    -> Fold {fold+1} Best Rounded MAE:
{best_score:.4f}")

        # --- 피처 중요도 Top 10 출력 ---
        imp_df = pd.DataFrame({
            'Feature': feat_names,
            'Importance':
model.feature_importance(importance_type='split')
        })
        top_10 = imp_df.sort_values('Importance',
ascending=False).head(10)
        print(f"       [Top 10 Important Features]")
        print(top_10.to_string(index=False,
header=False))
        print("-" * 40)
```

```
    # 6. 최종 결과 평가 및 저장 로직 (통합됨)

    # 1) 원본 OOF 점수
    score_final = mean_absolute_error(y, oof_preds)
    # 2) 반올림 적용 OOF 점수
    score_rounded = mean_absolute_error(y,
np.round(oof_preds, 2))
    print("\n" + "="*50)
    print(f"🏆 LightGBM Global MAE (Raw):
{score_final:.5f}")
    print(f"👀 (참고) 반올림 적용 시 MAE :
{score_rounded:.5f}")
    use_rounding = False
    if score_rounded < score_final:
        print("✨ 반올림(소수점 2자리)이 더 유리합니다! (최종
제출에 적용함)")
        use_rounding = True
    else:
        print("🤔 반올림하지 않는 것이 더 좋습니다.")
    print("="*50)
```

```
    # 최종 예측값 후처리
    final_preds = np.maximum(0.01, test_preds) # 음수 방지
    if use_rounding:
        final_preds = np.round(final_preds, 2)
        final_score_str = f"{score_rounded:.4f}"
    else:
        final_score_str = f"{score_final:.4f}"
```

03 결과

3. 결과

YOUR RECENT SUBMISSION

LGBM_Best_MAE_2.4288.csv

Submitted by SEONYEOK LEE · Submitted 36 minutes ago

Score: 1.75449

Private score:

↓ Jump to your leaderboard position

감사합니다