**Feature Engineering – Target Encoding**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import warnings

from category\_encoders import MEstimateEncoder

from sklearn.model\_selection import cross\_val\_score

from xgboost import XGBRegressor

*# Set Matplotlib defaults*

plt.style.use("seaborn-whitegrid")

plt.rc("figure", autolayout=True)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=14,

titlepad=10,

)

warnings.filterwarnings('ignore')

def score\_dataset(X, y, model=XGBRegressor()):

*# Label encoding for categoricals*

for colname **in** X.select\_dtypes(["category", "object"]):

X[colname], \_ = X[colname].factorize()

*# Metric for Housing competition is RMSLE (Root Mean Squared Log Error)*

score = cross\_val\_score(

model, X, y, cv=5, scoring="neg\_mean\_squared\_log\_error",

)

score = -1 \* score.mean()

score = np.sqrt(score)

return score

df = pd.read\_csv("../input/fe-course-data/ames.csv")

First you'll need to choose which features you want to apply a target encoding to. Categorical features with a large number of categories are often good candidates. Run this cell to see how many categories each categorical feature in the Ames dataset has.

df.select\_dtypes(["object"]).nunique()

MSSubClass 16

MSZoning 7

Street 2

Alley 3

LotShape 4

LandContour 4

Utilities 3

LotConfig 5

LandSlope 3

Neighborhood 28

Condition1 9

Condition2 8

BldgType 5

HouseStyle 8

OverallQual 10

OverallCond 9

RoofStyle 6

RoofMatl 8

Exterior1st 16

Exterior2nd 17

MasVnrType 5

ExterQual 4

ExterCond 5

Foundation 6

BsmtQual 6

BsmtCond 6

BsmtExposure 5

BsmtFinType1 7

BsmtFinType2 7

Heating 6

HeatingQC 5

CentralAir 2

Electrical 6

KitchenQual 5

Functional 8

We talked about how the M-estimate encoding uses smoothing to improve estimates for rare categories. To see how many times a category occurs in the dataset, you can use the value\_counts method. This cell shows the counts for SaleType, but you might want to consider others as well.

df["SaleType"].value\_counts()

WD 2536

New 239

COD 87

ConLD 26

CWD 12

ConLI 9

ConLw 8

Oth 7

Con 5

VWD 1

Name: SaleType, dtype: int64

1) Choose Features for Encoding

Which features did you identify for target encoding? After you've thought about your answer, run the next cell for some discussion.

The Neighborhood feature looks promising. It has the most categories of any feature, and several categories are rare. Others that could be worth considering are SaleType, MSSubClass, Exterior1st, Exterior2nd. In fact, almost any of the nominal features would be worth trying because of the prevalence of rare categories.

Now you'll apply a target encoding to your choice of feature. As we discussed in the tutorial, to avoid overfitting, we need to fit the encoder on data heldout from the training set. Run this cell to create the encoding and training splits:

*# Encoding split*

X\_encode = df.sample(frac=0.20, random\_state=0)

y\_encode = X\_encode.pop("SalePrice")

*# Training split*

X\_pretrain = df.drop(X\_encode.index)

y\_train = X\_pretrain.pop("SalePrice")

# 2) Apply M-Estimate Encoding

Apply a target encoding to your choice of categorical features. Also choose a value for the smoothing parameter m (any value is okay for a correct answer).

*# YOUR CODE HERE: Create the MEstimateEncoder*

*# Choose a set of features to encode and a value for m*

encoder = MEstimateEncoder(

cols=["Neighborhood"],

m=1.0,

)

*# Fit the encoder on the encoding split*

encoder.fit(X\_encode, y\_encode)

*# Encode the training split*

X\_train = encoder.transform(X\_pretrain, y\_train)

see how the encoded feature compares to the target

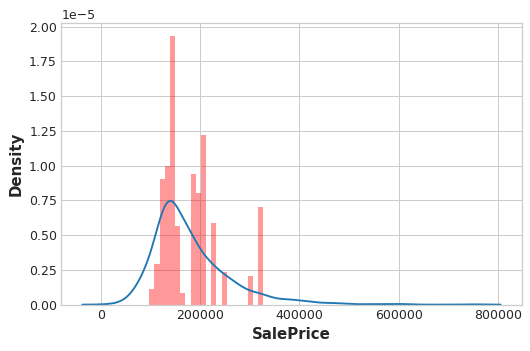
feature = encoder.cols

plt.figure(dpi=90)

ax = sns.distplot(y\_train, kde=True, hist=False)

ax = sns.distplot(X\_train[feature], color='r', ax=ax, hist=True, kde=False, norm\_hist=True)

ax.set\_xlabel("SalePrice");



From the distribution plots, does it seem like the encoding is informative?

And this cell will show you the score of the encoded set compared to the original set:

X = df.copy()

y = X.pop("SalePrice")

score\_base = score\_dataset(X, y)

score\_new = score\_dataset(X\_train, y\_train)

print(f"Baseline Score: **{**score\_base**:**.4f**}** RMSLE")

print(f"Score with Encoding: **{**score\_new**:**.4f**}** RMSLE")

Baseline Score: 0.1428 RMSLE

Score with Encoding: 0.1402 RMSLE

Do you think that target encoding was worthwhile in this case? Depending on which feature or features you chose, you may have ended up with a score significantly worse than the baseline. In that case, it's likely the extra information gained by the encoding couldn't make up for the loss of data used for the encoding.

In this question, you'll explore the problem of overfitting with target encodings. This will illustrate this importance of training fitting target encoders on data held-out from the training set.

So let's see what happens when we fit the encoder and the model on the same dataset. To emphasize how dramatic the overfitting can be, we'll mean-encode a feature that should have no relationship with SalePrice, a count: 0, 1, 2, 3, 4, 5, ....

*# Try experimenting with the smoothing parameter m*

*# Try 0, 1, 5, 50*

m = 0

X = df.copy()

y = X.pop('SalePrice')

*# Create an uninformative feature*

X["Count"] = range(len(X))

X["Count"][1] = 0 *# actually need one duplicate value to circumvent error-checking in MEstimateEncoder*

*# fit and transform on the same dataset*

encoder = MEstimateEncoder(cols="Count", m=m)

X = encoder.fit\_transform(X, y)

*# Results*

score = score\_dataset(X, y)

print(f"Score: **{**score**:**.4f**}** RMSLE")

Score: 0.0293 RMSLE

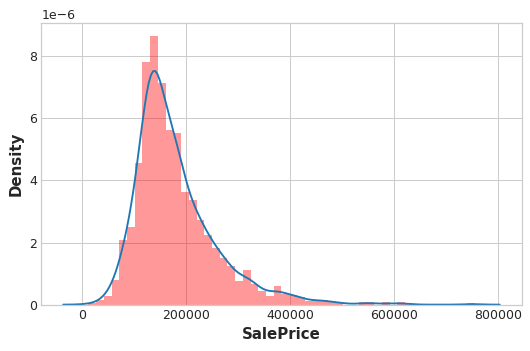
Almost a perfect score

plt.figure(dpi=90)

ax = sns.distplot(y, kde=True, hist=False)

ax = sns.distplot(X["Count"], color='r', ax=ax, hist=True, kde=False, norm\_hist=True)

ax.set\_xlabel("SalePrice");



# 3) Overfitting with Target Encoders

Based on your understanding of how mean-encoding works, can you explain how XGBoost was able to get an almost a perfect fit after mean-encoding the count feature?

Since Count never has any duplicate values, the mean-encoded Count is essentially an exact copy of the target. In other words, mean-encoding turned a completely meaningless feature into a perfect feature.

Now, the only reason this worked is because we trained XGBoost on the same set we used to train the encoder. If we had used a hold-out set instead, none of this "fake" encoding would have transferred to the training data.

The lesson is that when using a target encoder it's very important to use separate data sets for training the encoder and training the model. Otherwise the results can be very disappointing!