AI 534 Machine Learning HW3

Submitted by: Apoorv Malik (malikap@oregonstate.edu)

1. Understanding the Evaluation Metric

1. The Root Mean Squared Logarithmic Error (**RMSLE**) is a metric used to evaluate the performance of regression models, especially when the predicting quantities are of a large scale. It is useful when you want to penalize under-predictions more than over predictions.

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

2. The Root Mean Squared Error **RMSE** computes the square root of the average of the squared differences between predicted and actual values. **RMSLE** first takes the natural logarithm of each predicted and actual value (plus one to avoid log of zero), then computes the square root of the average of these squared log-differences.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

3. Real estate prices often have a skewed distribution; there are far more moderately priced homes than extremely expensive ones. **RMSLE**, by applying a logarithmic transformation, helps in normalizing this skewness,

   RMSE can disproportionately penalize large errors, as it squares the difference between predicted and actual values. **RMSLE** reduces the relative impact of large errors compared to **RMSE**.

   Also **RMSLE** has an advantage of penalizing underestimates more than overestimates. This is a good feature because underestimating the value of a property can be more problematic than overestimating it.

4. An **RMSLE** of **0.11** indicates that, on average, the logarithmic error of the predictions is relatively small. A **28th rank** in the competition indicates a strong performance relative to other models. It also indicates a balanced approach to underestimation and overestimation, leaning slightly towards avoiding underestimation.

   Also, $e^{0.11} - 1 \approx 0.116$. This means that the predicted prices are, on average, within about $11.6\,\%$ of the actual prices.

5. Kaggle $\text{RMSLE} = 0.40613$
   Kaggle $\text{Rank} = 4530$

6. My Team Name is **Apoorv Malik #2**

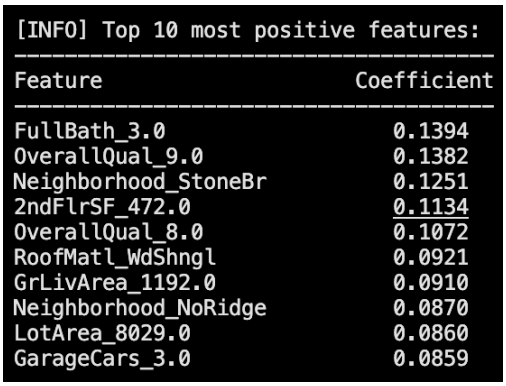
[EXTRA CREDIT]: Housing prices often have a skewed distribution. There are a few very high values and many low-to-moderate values properties. Taking the log of the target variable (SalePrice) can help normalizing the distribution, making it more symmetric and closer to the normal distribution. Taking the log also stabilizes the variance across the range of data, and reduces the impact of outliers by compressing the scale (thus preventing them from having a high influence on the model). By addressing issues like skewness, irregular variance, and outliers, log transformation can lead to an improvement in the overall fit of the model and the accuracy of predictions.

Table 01: Number of Features for each Field

Field No.	No. of Features		Fields No.	No. of Features		Fields No.	No. of Features		Fields No.	No. of Features
-	-		**Field 21**	61		**Field 41**	4		**Field 61**	4
Field 2	15		**Field 22**	6		**Field 42**	2		**Field 62**	5
Field 3	5		**Field 23**	8		**Field 43**	6		**Field 63**	422
Field 4	108		**Field 24**	15		**Field 44**	721		**Field 64**	6
Field 5	989		**Field 25**	16		**Field 45**	390		**Field 65**	6
Field 6	2		**Field 26**	5		**Field 46**	21		**Field 66**	3
Field 7	3		**Field 27**	305		**Field 47**	810		**Field 67**	253
Field 8	4		**Field 28**	4		**Field 48**	4		**Field 68**	193
Field 9	4		**Field 29**	5		**Field 49**	3		**Field 69**	116
Field 10	2		**Field 30**	6		**Field 50**	4		**Field 70**	17
Field 11	5		**Field 31**	5		**Field 51**	3		**Field 71**	72
Field 12	3		**Field 32**	5		**Field 52**	8		**Field 72**	8
Field 13	25		**Field 33**	5		**Field 53**	4		**Field 73**	4
Field 14	9		**Field 34**	7		**Field 54**	4		**Field 74**	5
Field 15	8		**Field 35**	601		**Field 55**	12		**Field 75**	5
Field 16	5		**Field 36**	7		**Field 56**	7		**Field 76**	21
Field 17	8		**Field 37**	131		**Field 57**	4		**Field 77**	12
Field 18	10		**Field 38**	730		**Field 58**	6		**Field 78**	5
Field 19	9		**Field 39**	686		**Field 59**	7		**Field 79**	9
Field 20	110		**Field 40**	6		**Field 60**	97		**Field 80**	6
TOTAL FEATURES										7227

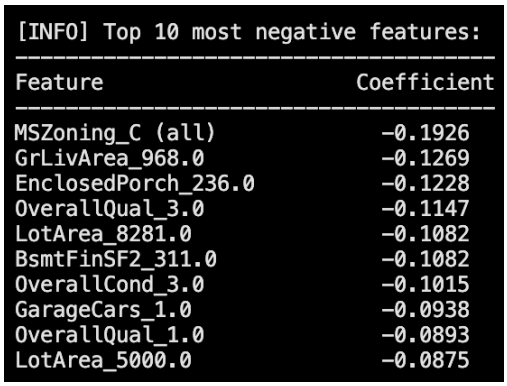
2. Naive data processing: binarizing all fields

1. We have 7227 total features
2. Refer to **Table 01** above.
3. Dev $\text{RMSLE} = 0.15190$
4. Please refer to the image below.



[INFO] Top 10 most positive features:

Feature	Coefficient
FullBath_3.0	0.1394
OverallQual_9.0	0.1382
Neighborhood_StoneBr	0.1251
2ndFlrSF_472.0	0.1134
OverallQual_8.0	0.1072
RoofMatl_WdShngl	0.0921
GrLivArea_1192.0	0.0910
Neighborhood_NoRidge	0.0870
LotArea_8029.0	0.0860
GarageCars_3.0	0.0859



[INFO] Top 10 most negative features:

Feature	Coefficient
MSZoning_C (all)	-0.1926
GrLivArea_968.0	-0.1269
EnclosedPorch_236.0	-0.1228
OverallQual_3.0	-0.1147
LotArea_8281.0	-0.1082
BsmtFinSF2_311.0	-0.1082
OverallCond_3.0	-0.1015
GarageCars_1.0	-0.0938
OverallQual_1.0	-0.0893
LotArea_5000.0	-0.0875

Yes, these features make sense, as they seem very critical for affecting the price of a given house.

5. **[Q5 + Extra Credit Question]** Our regression tool automatically handles the addition of bias (intercept) dimension for us. The feature weight for the bias dimension (or intercept) is 12.1748. In the original scale this intercept translates to $e^{12.1748} \approx 160{,}000$. Yes, this value makes sense.

   The intercept represents the predicted value of the target variable when all other feature values are zero. In house pricing, it's the baseline value before accounting for any specific features of the house.

   Looking at the intercept, we can assume that the base price of a house (when it lacks all the additional characteristics) would be around **USD 160k**.

6. Kaggle $\text{RMSLE} = 0.15737$
   Kaggle $\text{Rank} = 3155$

3. Smarter binarization: Only binarizing categorical features

1. The main drawback of Naive Binarization is the loss of information. It might perhaps be the most significant one. Binarization reduces the complexity of the data to just two states, which can oversimplify the nuances and gradations in the original data. This is particularly critical for continuous variables where the range of values conveys important information.

   In datasets with a large number of features, binarization can lead to increased data sparsity. Sparse data can be challenging for certain machine learning algorithms.

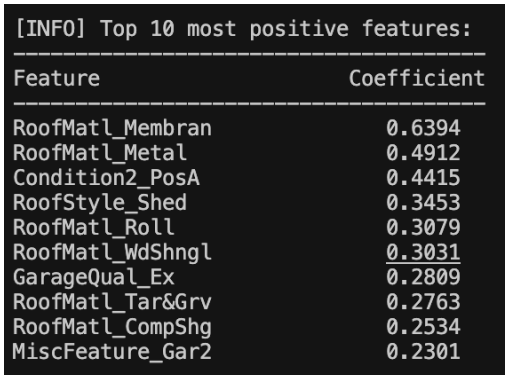
   It may also lead to decreased efficiency of the model, as the input space may become extremely high dimensional after Naive Binarization.

2. I only binarized the categorical features and scaled the numerical ones. I have 286 features now. For the mixed features such as $\mathrm{LotFrontage}$ and $\mathrm{GarageYrBlt}$, I used **SimpleImputer** in my preprocessing pipeline which replaces the NA values with the mean value in the column.

3. I have 286 features now.

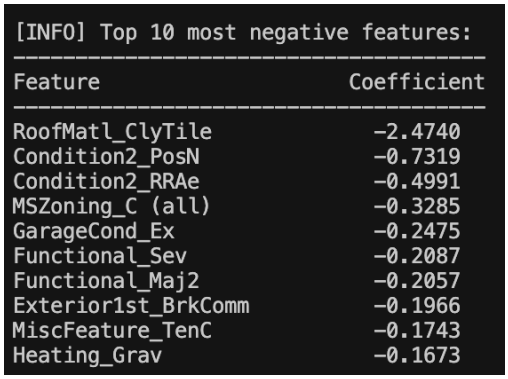
4. Dev $\mathrm{RMSLE} = 0.12394$

5. Please refer to the image below.



[INFO] Top 10 most positive features:

Feature	Coefficient
RoofMatl_Membran	0.6394
RoofMatl_Metal	0.4912
Condition2_PosA	0.4415
RoofStyle_Shed	0.3453
RoofMatl_Roll	0.3079
RoofMatl_WdShngl	0.3031
GarageQual_Ex	0.2809
RoofMatl_Tar&Grv	0.2763
RoofMatl_CompShg	0.2534
MiscFeature_Gar2	0.2301



[INFO] Top 10 most negative features:

Feature	Coefficient
RoofMatl_ClyTile	-2.4740
Condition2_PosN	-0.7319
Condition2_RRAe	-0.4991
MSZoning_C (all)	-0.3285
GarageCond_Ex	-0.2475
Functional_Sev	-0.2087
Functional_Maj2	-0.2057
Exterior1st_BrkComm	-0.1966
MiscFeature_TenC	-0.1743
Heating_Grav	-0.1673

Yes, these features are completely different from the previous section.

6. Kaggle $\mathrm{RMSLE} = 0.15038$
   Kaggle $\mathrm{Rank} = 2750$

4. Experimentation

1. I used $\text{sklearn.linear\_model.Ridge}$ and after tuning α on dev, I got $\alpha = 0.01$. These are the results:

   - **Naive Binarization**: Kaggle $\text{RMSLE} = 0.15725$ (minor improvement)
   - **Smart Binarization**: Kaggle $\text{RMSLE} = 0.15086$ (minor deterioration)

2. I used non-linear regression using $\text{PolynomialFeatures}(\text{degree} = 2)$. I used these columns $(\text{LotArea}, \text{LotFrontage}, \text{TotalBsmtSF})$ as candidates for non-linear transformations. I am using **Smart Binarization + Regularized Linear Regression** ($\alpha = 0.01$). These are the results:

   - Dev $\text{RMSLE} = 0.12242$ (best till now)
   - Kaggle $\text{RMSLE} = 0.14394$ (best till now)
   - Kaggle $\text{Rank} = 2045$ (best till now)

3. Transforming the input space with a non-linear feature map enables linear models to effectively capture non-linear relationships.

   The perceptron was not able to linearly separate the **XOR** dataset. Because for **XOR** dataset, (true) $y = x_1 x_2$, which is a non-linear relation. Therefore, we created a feature map that consisted of a non-linear feature (using feature combination), this allowed us to linearly separate the dataset in the higher dimension.

   The same principle applies here. Using a feature map consisting of non-linear features (including feature combinations), we can linearly fit the model in higher dimensions, which then creates a non-linear decision boundary in the original input space dimension. The transformation into a higher-dimensional space (via non-linear feature mapping) allows linear models to fit more complex patterns.

4. I tried **XGBRegressor**, and tuned parameter using grid search (n_estimators = 500, learning_rate = 0.1, max_depth = 3). I am using **Smart Binarization + PolynomialFeatures**. These are the results that got:

   I don't have a proper dev error, because I used the whole dataset for training and used cross validation (k = 5).

   - Kaggle $\text{RMSLE} = 0.13581$ (best result)
   - Kaggle $\text{Rank} = 1468$ (best result)

5. Debriefing

1. 12 hours.
2. I would rate this as moderate.
3. I worked on this alone.
4. I feel I understand 99%.
5. Really liked trying hard to get a good score on Kaggle and tune custom models.