Ames Housing Data:
Machine Learning Applications

Victoria Baker, Jan Forslow, Alice Lam, Josh Lee

Overview

The data, and our approach

The Data

- Feature counts:
  - 72 => 212 after processing
- Row counts:
  - 1460 test, 1459 train
- Features cover:
  - Size
  - Conditions rating
  - Utilities
  - Construction features
  - Community and neighborhood
- Data split for assessment:
  - Fit/Predict: mini_train (80%), dev (20%)
  - Cross validation on mini_train
  - Scoring: Root mean squared log error (RMSLE)

Our Approach

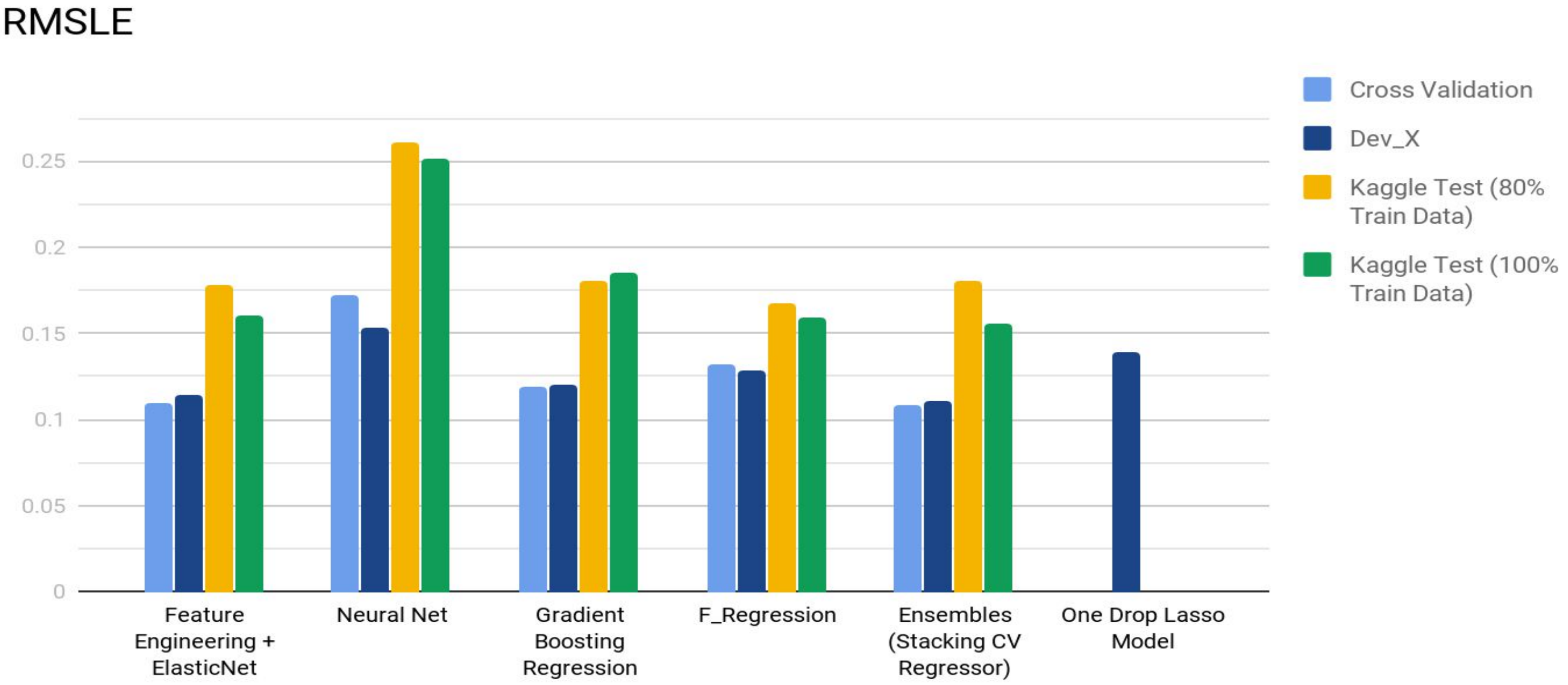
Looking for the best model

EDA → Data Cleansing → Initial Diagnostic → Feature Engineering + GSCV

Experimenting with other models

Neural Network → GDBoost → F Regression → Ensembles → One Drop Model → Decision Tree Regressor

Our Best Results (RMSLE)



What have we learned from the process?

Why some work while some don't?

?

1

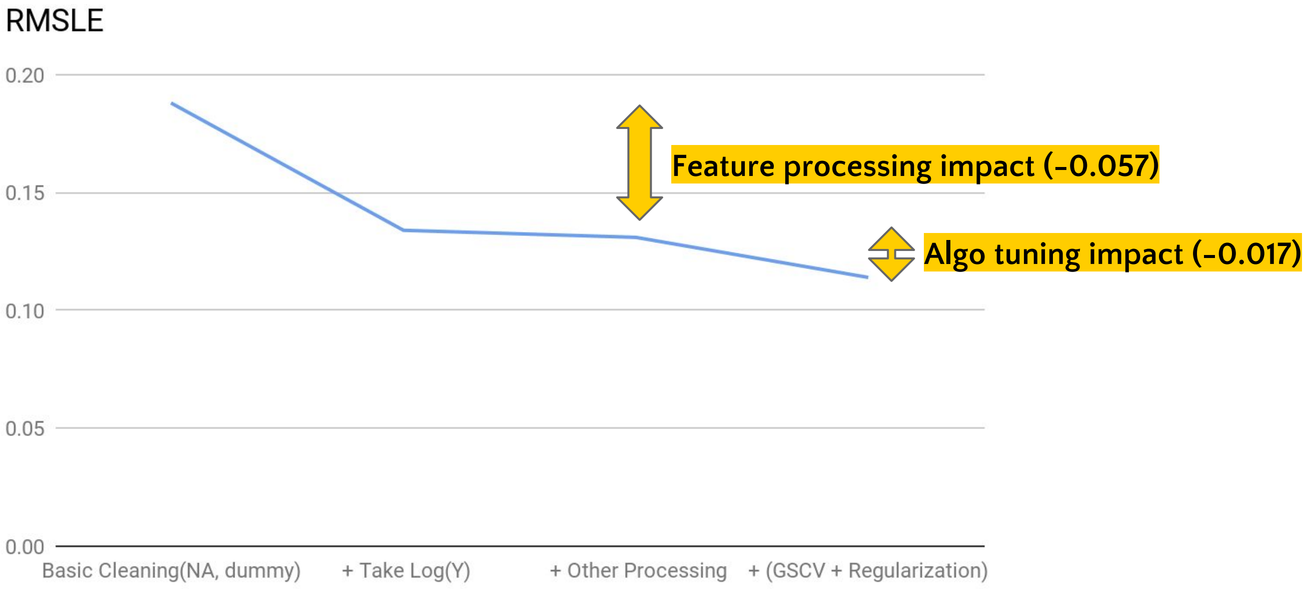
Key Learnings

From: EDA, Feature Processing

Impact of feature processing

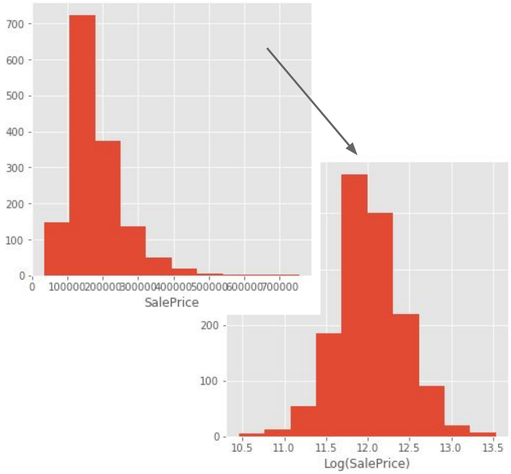
Based on Basic LinearRegression() Performance:

Fit mini_train (N=1162)
Pred dev (N = 291)

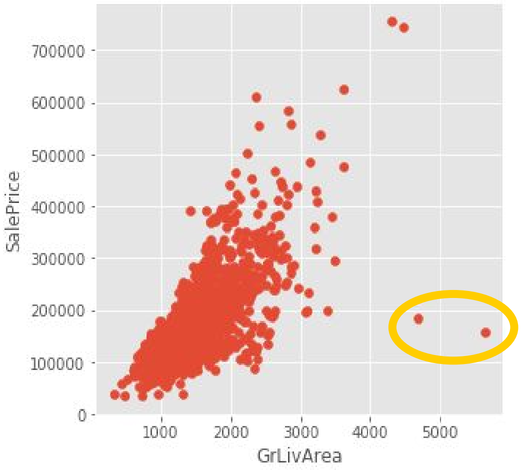


Feature processing

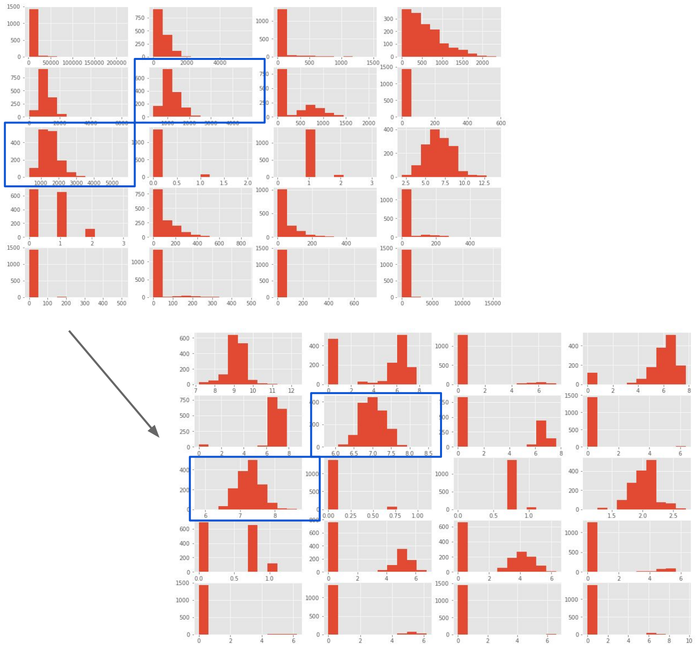
Taking Log of Y (SalePrice)



Outliers



Normalize skewness



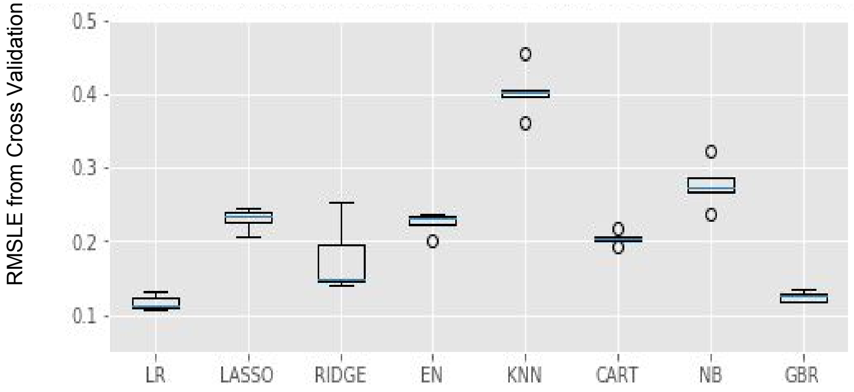
2

Key Learnings

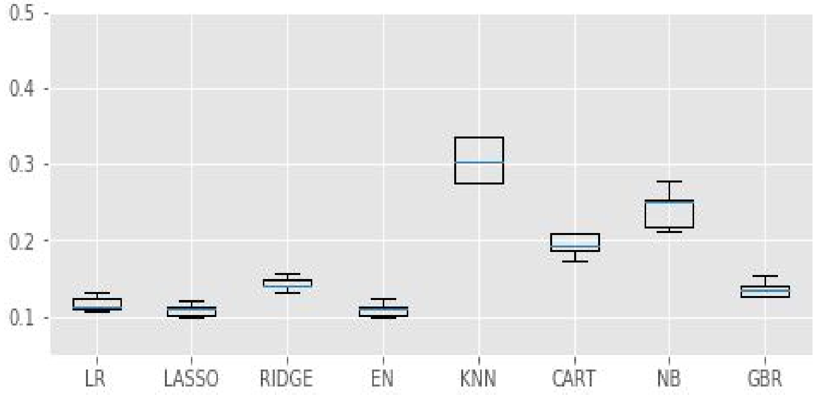
From: Feature Engineering and GSCV

GSCV: Comparing across models

Default Param



Best Param from GridsearchCV



Best Param:

LR: Normalize = False

Lasso: Normalize = True, Alpha = 0.0001

Ridge: Normalize = True, Alpha = 1.0

EN (Elastic Net): Fit_Intercept= True, Normalize = True, Alpha = 0.0001, L1_ratio = 0.9

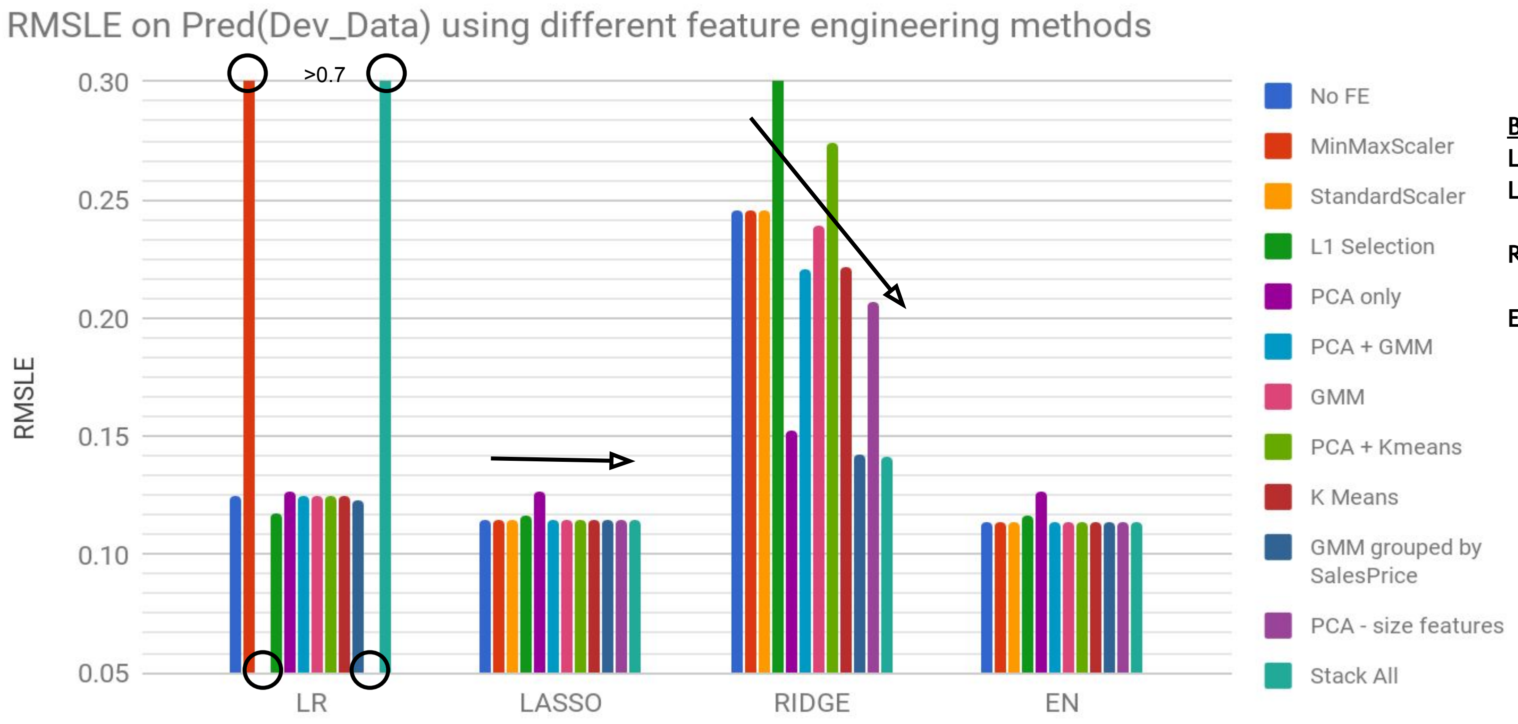
KNN: N = 1

CART (Decision Tree Regressor): Max_features = 10, max_depth = 11, min_samples_split = 60

NB: alpha = 0.0001

GBR (Gradient Boosting Regressor): Learning_rate = 0.04, n_estimators = 100

Feature Engineering



Best Param:

LR: Normalize = False

Lasso: Normalize = True, Alpha = 0.0001

Ridge: Normalize = True, Alpha = 1.0

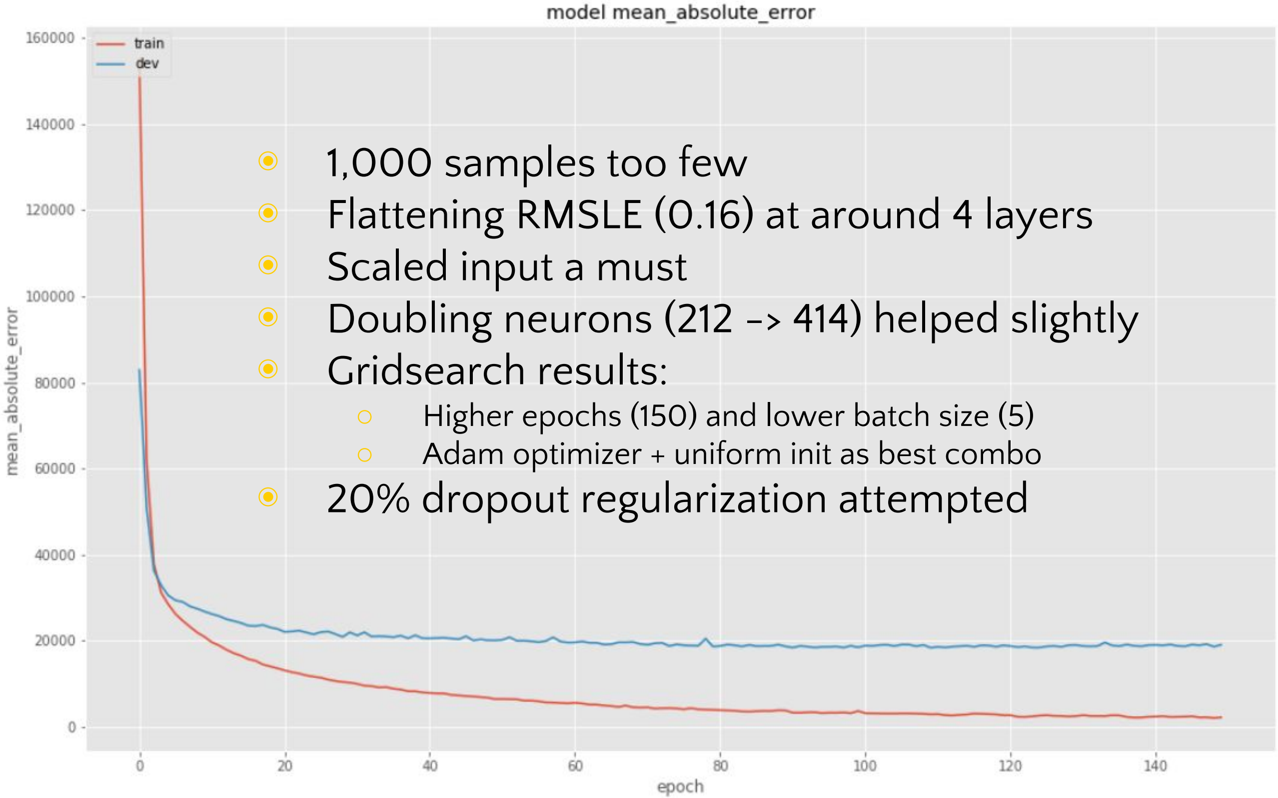
EN: Fit_Intercept= True, Normalize = True, Alpha = 0.0001, L1_ratio = 0.9

3

Key Learnings

From: Neural Network

NN on Regression Problem



- 1,000 samples too few
- Flattening RMSLE (0.16) at around 4 layers
- Scaled input a must
- Doubling neurons (212 -> 414) helped slightly
- Gridsearch results:
  - Higher epochs (150) and lower batch size (5)
  - Adam optimizer + uniform init as best combo
- 20% dropout regularization attempted

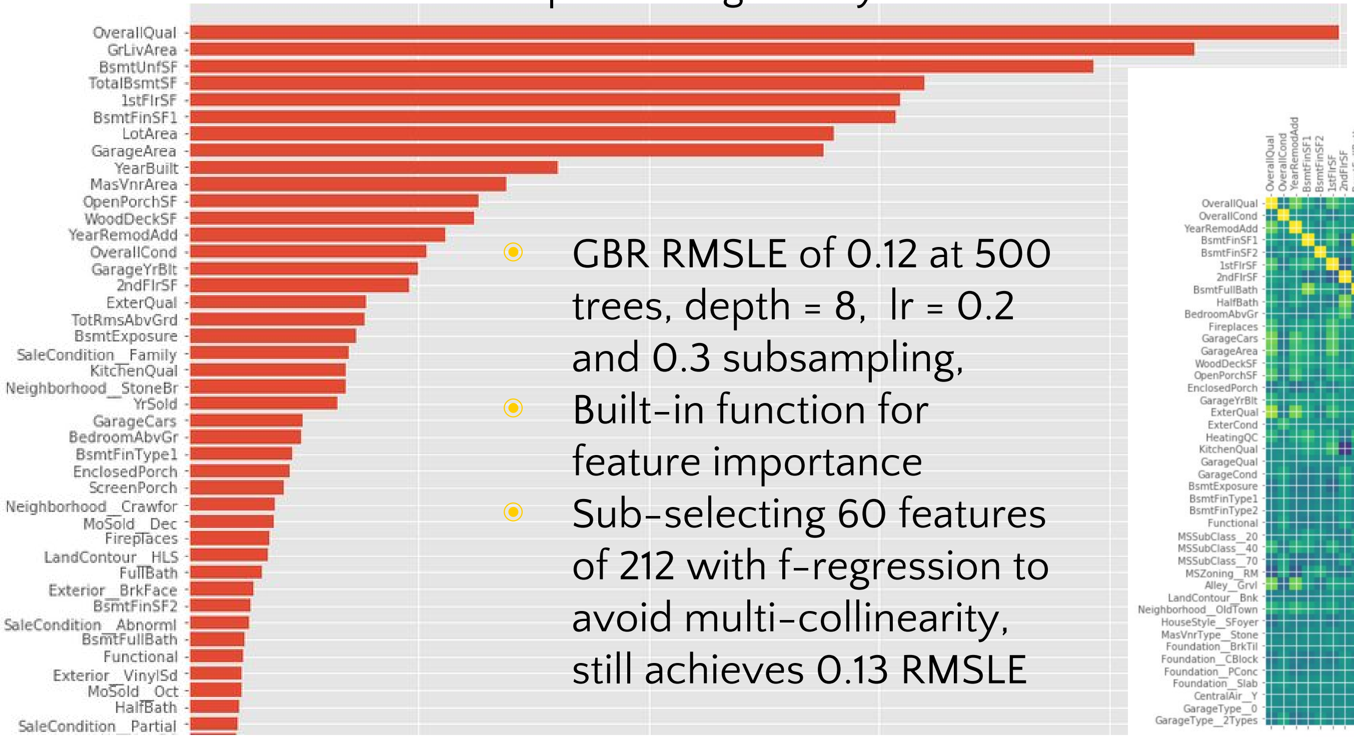
4

Key Learnings

From: Gradient Boosting & F Regression

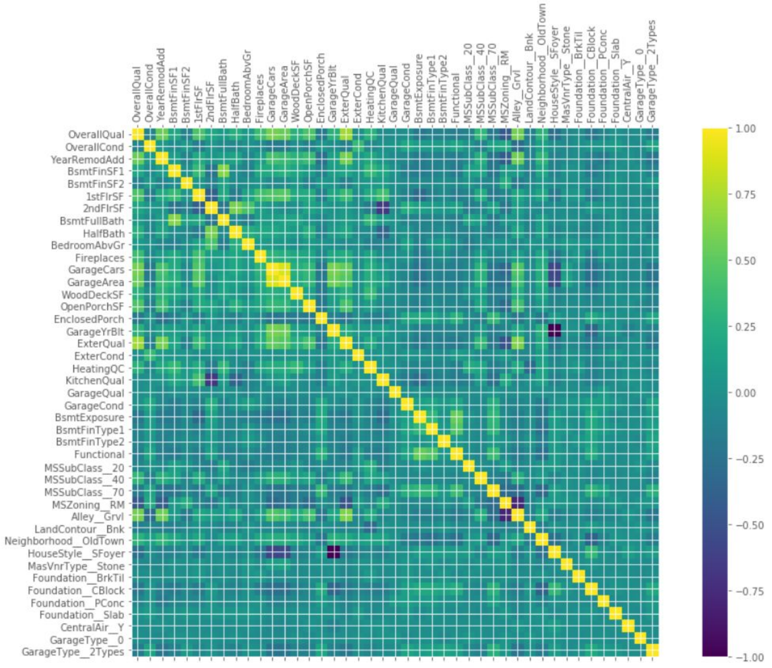
Helpful for Feature Learning

Feature Importance given by GBR



- GBR RMSLE of 0.12 at 500 trees, depth = 8, lr = 0.2 and 0.3 subsampling,
- Built-in function for feature importance
- Sub-selecting 60 features of 212 with f-regression to avoid multi-collinearity, still achieves 0.13 RMSLE

Correlation Matrix based on F-Regression, n=60



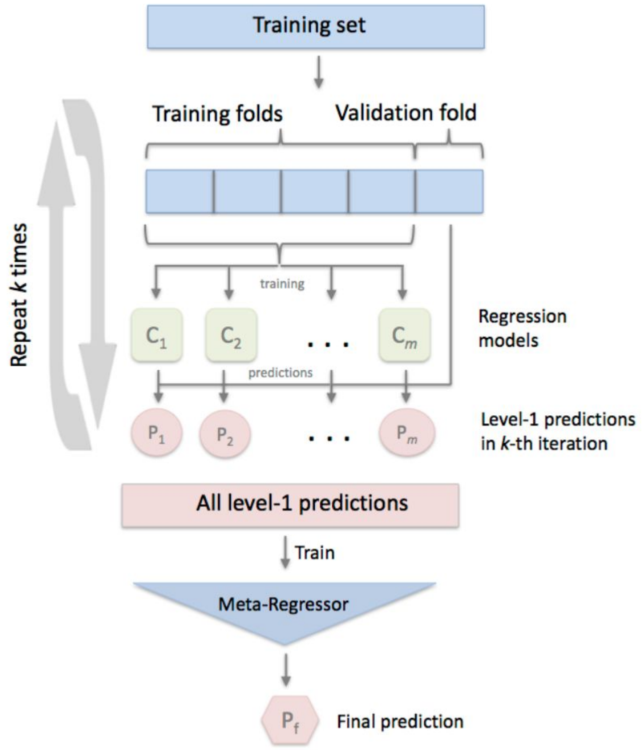
5

Key Learnings

From: Ensembles

Ensembles Hard to Find

- Tested three off-the-shelf ensemble modules
- Scikit-learn ensemble.VotingClassifier
  - Wraps models and averages the predictions
  - Not working for LR models
- mixtend.StackingRegressor
  - Individual regression models trained on complete training set; Meta-regressor LR fitted on the outputs
  - Combined LR, LASSO, EN, Ridge, CART and GBR
- mixtend.StackingCVRegressor
  - Using out-of-fold predictions to prepare the input data for the level-2 classifier; Avoids over-fitting
  - Achieved RMSLE of 0.11 on dev-set



6

Key Learnings

From: One Drop Model & Decision Trees

One Drop Model: What We Learned

Key Points

1. Feature processing is very important

2. Use “system” level not “unit” level metrics

RR Before FP

	0
count	212.000000
mean	0.418923
std	0.000114
min	0.418449
25%	0.418919
50%	0.418919
75%	0.418919
max	0.420471

RR After FP

	0
count	2.1e+02
mean	2.4e-01
std	3.7e-02
min	1.4e-01
25%	2.4e-01
50%	2.5e-01
75%	2.5e-01
max	3.3e-01

Decision Tree: Information Gain & f-regression

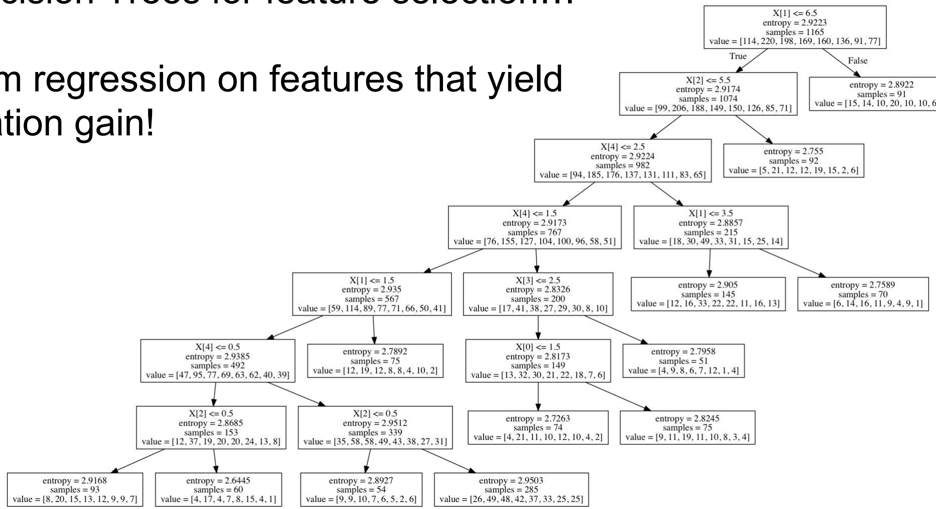
DT Similar to GBR Feature Importance

GBR	DT
OverallQual	OverallQual
GrLivArea	TotalBsmtSF
BsmtUnfSF	GrLivArea
TotalBsmtSF	GarageArea
1stFlrSF	GarageFinish
BsmtFinSF1	GarageType__Detchd

Idea

Why not use Decision Trees for feature selection…

And then perform regression on features that yield greatest information gain!



Conclusion

Ways to improve further

Regression Diagnostics

- Problems meeting multicollinearity and homoscedasticity assumptions
- Difficult to combine/remove enough features to make an impact without losing potentially important data
- Feature engineering work should be informed by regression diagnostics, not limited by them

Further Improvements

- Use regression diagnostics to optimize One Drop and f-regression models
- Run One Drop model with more features being dropped
- Try using Gradient Descent and see if it improves basic linear regression
- Design our own ensemble function that combines all model types
- Use Independent Component Analysis to decompose features with high mutual information
- Adding jitter to the features in training data to test stability of OLS, Lasso, Ridge, and Elastic Net models

Thanks!

Any ***questions*** *?*

Reference:

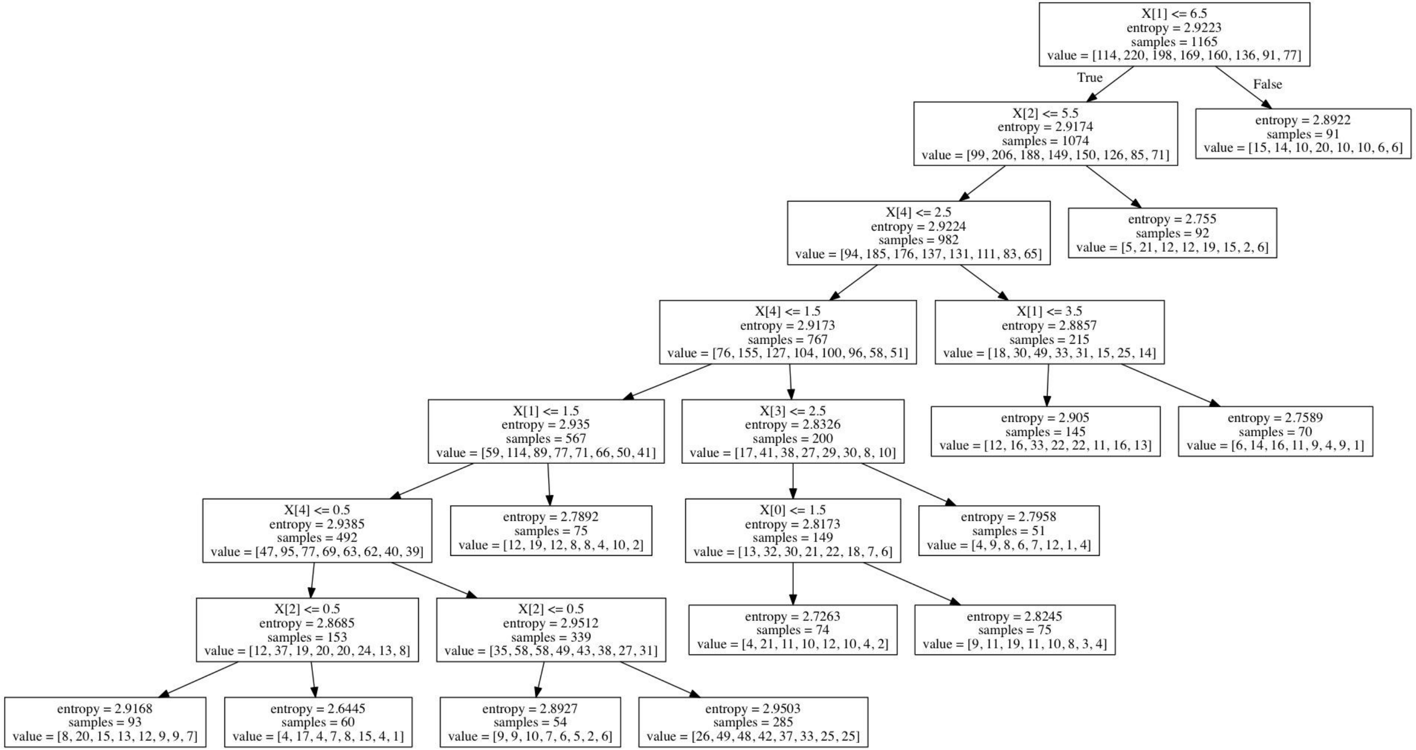
- Effects of multicollinearity: http://blog.minitab.com/blog/adventures-in-statistics-2/what-are-the-effects-of-multicollinearity-and-when-can-i-ignore-them

Ames Housing Data:
Machine Learning Applications

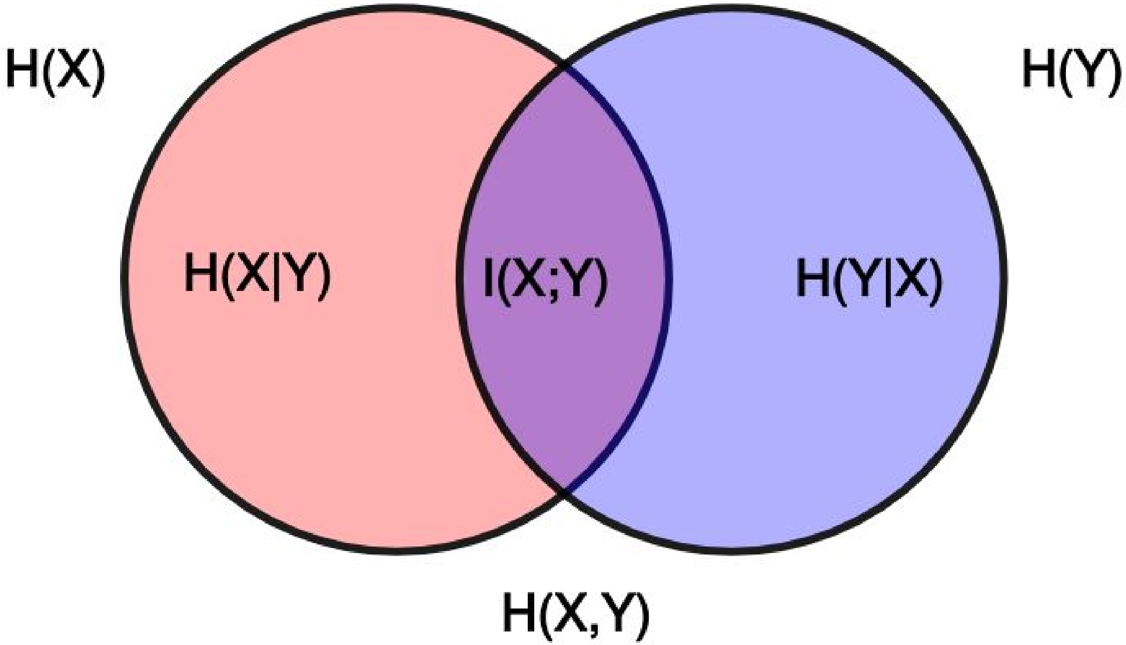
Victoria Baker, Jan Forslow, Alice Lam, Josh Lee

Additional Slides

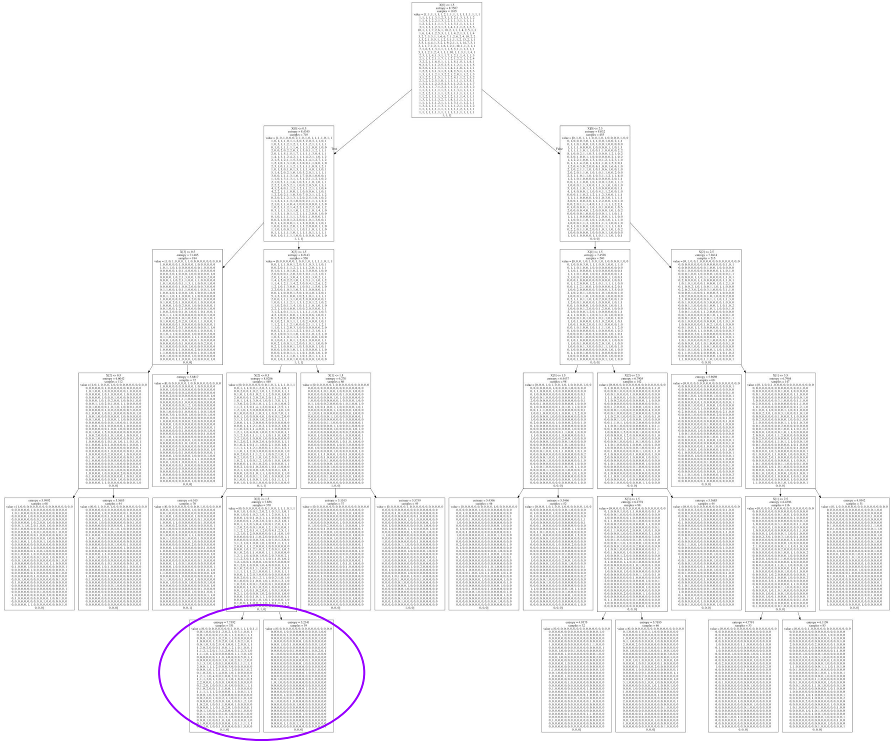
Decision Trees: 8 price bins min sample leaf size 30

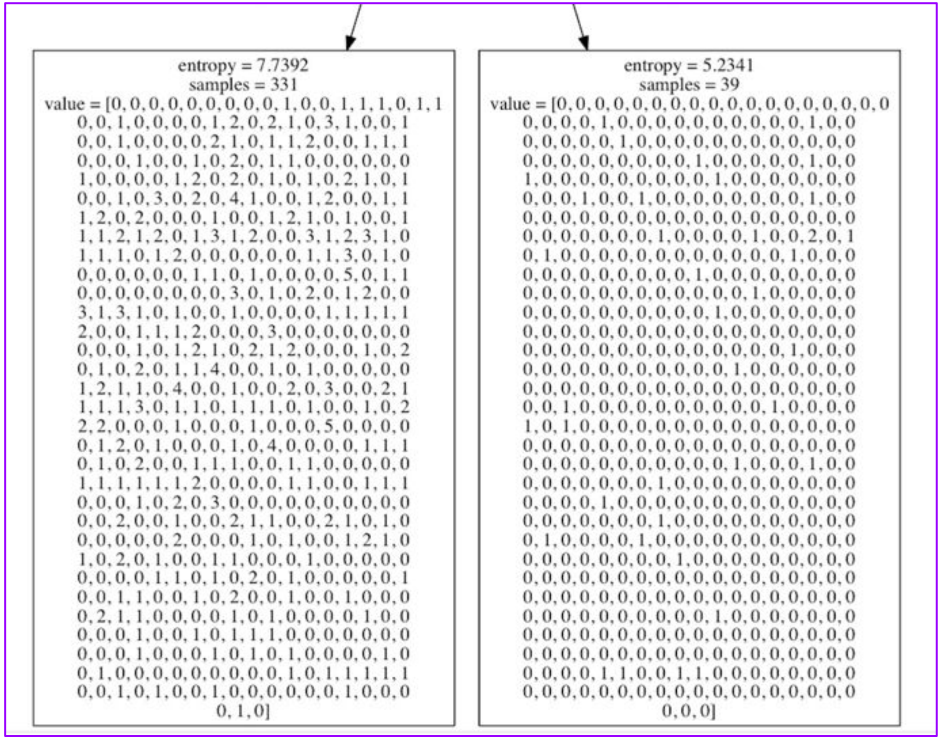


Mutual Information

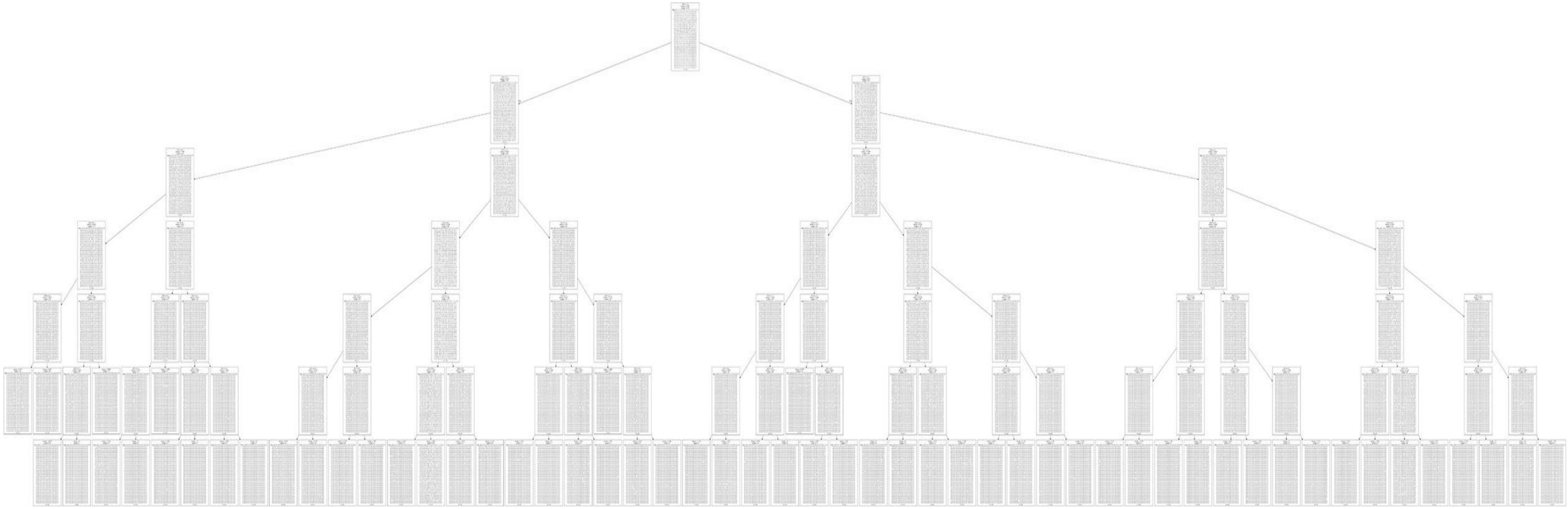


Decision Tree: Classifier

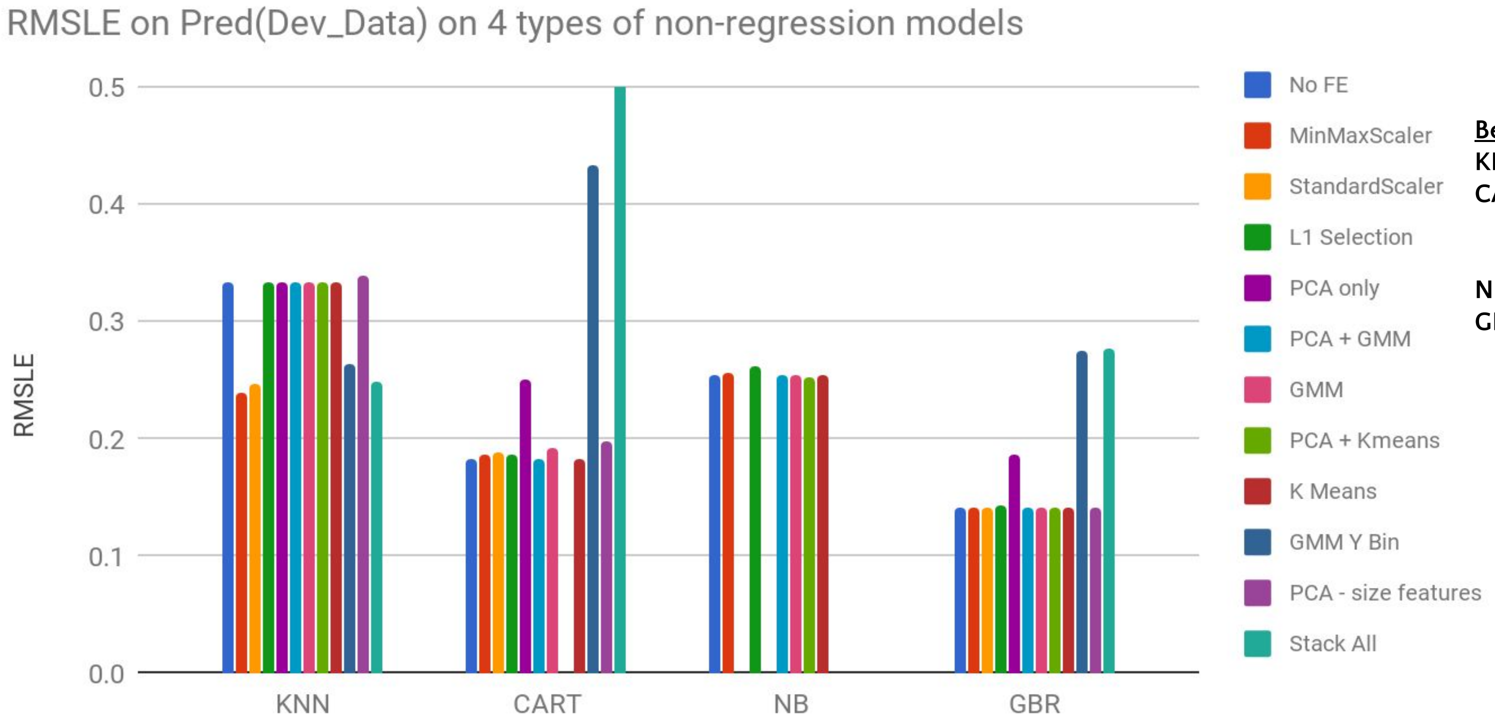




Decision Trees: what happens when you use the default settings



Feature Engineering



Best Param:

KNN: N = 1

CART: Max_features = 10, max_depth = 11, min_samples_split = 60

NB: alpha = 0.0001

GBR: Learning_rate = 0.04, n_estimators = 100

Our Best Results (RMSLE)

	Feature Engineering + GSCV ElasticNet	Neural Net	Gradient Boosting Regression	F_Regression	Ensembles (Stacking CV Regressor)	One Drop Lasso Model
Cross Validation	0.110	0.172	0.119	0.132	0.108	/
Dev_X	0.114	0.153	0.120	0.129	0.111	0.139
Kaggle Test (80% Train data)	0.178	0.261	0.181	0.167	0.181	
Kaggle Test (100% Train data)	0.160	0.252	0.185	0.159	0.156	