Predicting Trends in Housing Prices Using Ensemble Learning

Team Members:
16BCB0062 (VAASU GUPTA)
16BCB0129 (SIDDHARTH SHAILENDRA)
16BCB0029 (PAVAN PRAJAPATI)

Report submitted for the Third Project Review of

Course Code: CSE3019 – Data Mining

Slot: G1 + TG1

Professor: Rishin Haldar

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)

Abstract

Usually, House price index represents the summarized price changes of residential housing. While for a single family house price prediction, it needs more accurate method based on location, house type, size, build year, local amenities, and some other factors which could affect house demand and supply. With this dataset and data features, a practical and composite data pre-processing, creative feature engineering method is examined in this paper. The paper also proposes ensemble learning techniques for boosting regression model to predict individual house price. The proposed approach has recently been deployed as the key kernel for Kaggle Challenge “House Prices: Advanced Regression Techniques”. The performance is promising

Introduction

Ensemble learning

In statistics and Data science ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone

Supervised learning algorithms are most commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem.

Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis. The term ensemble is usually reserved for methods that generate multiple hypotheses using the same base learner

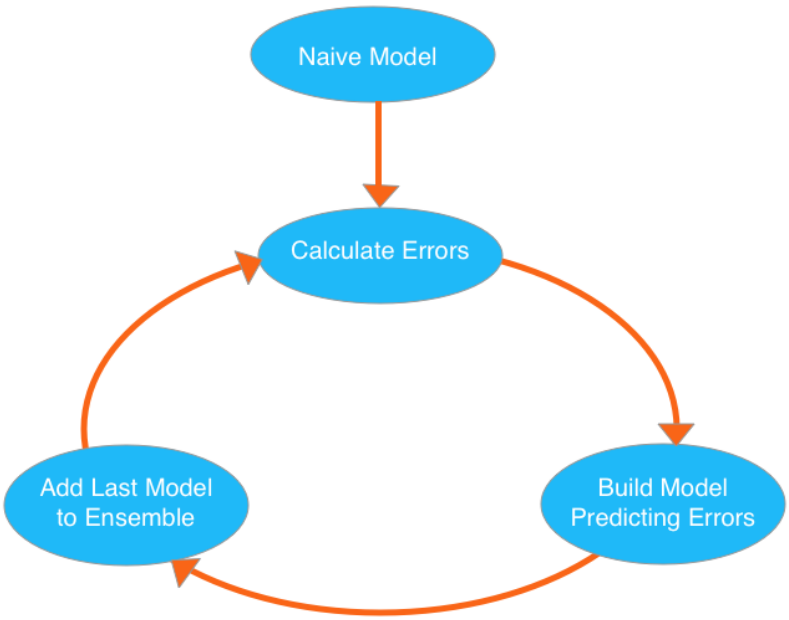
An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, some ensemble techniques (especially bagging) tend to reduce problems related to over-fitting of the training data

XGBoost

XGBoost is the leading model for working with standard tabular data(the type of data stored in Pandas DataFrames, as opposed to more exotic types of data like images and videos).

To reach peak accuracy, XGBoost models require more knowledge and *model tuning* than techniques like Random Forest.

XGBoost is an implementation of the Gradient Boosted Decision Trees algorithm (scikit-learn has another version of this algorithm, but XGBoost has some technical advantages.)



It goes through cycles that repeatedly builds new models and combines them into an **ensemble** model. We start the cycle by calculating the errors for each observation in the dataset. We then build a new model to predict those. We add predictions from this error-predicting model to the "ensemble of models.

To make a prediction, we add the predictions from all previous models. We can use these predictions to calculate new errors, build the next model, and add it to the ensemble.

There's one piece outside that cycle. We need some base prediction to start the cycle. In practice, the initial predictions can be pretty naive. Even if it's

predictions are wildly inaccurate, subsequent additions to the ensemble will address those errors.

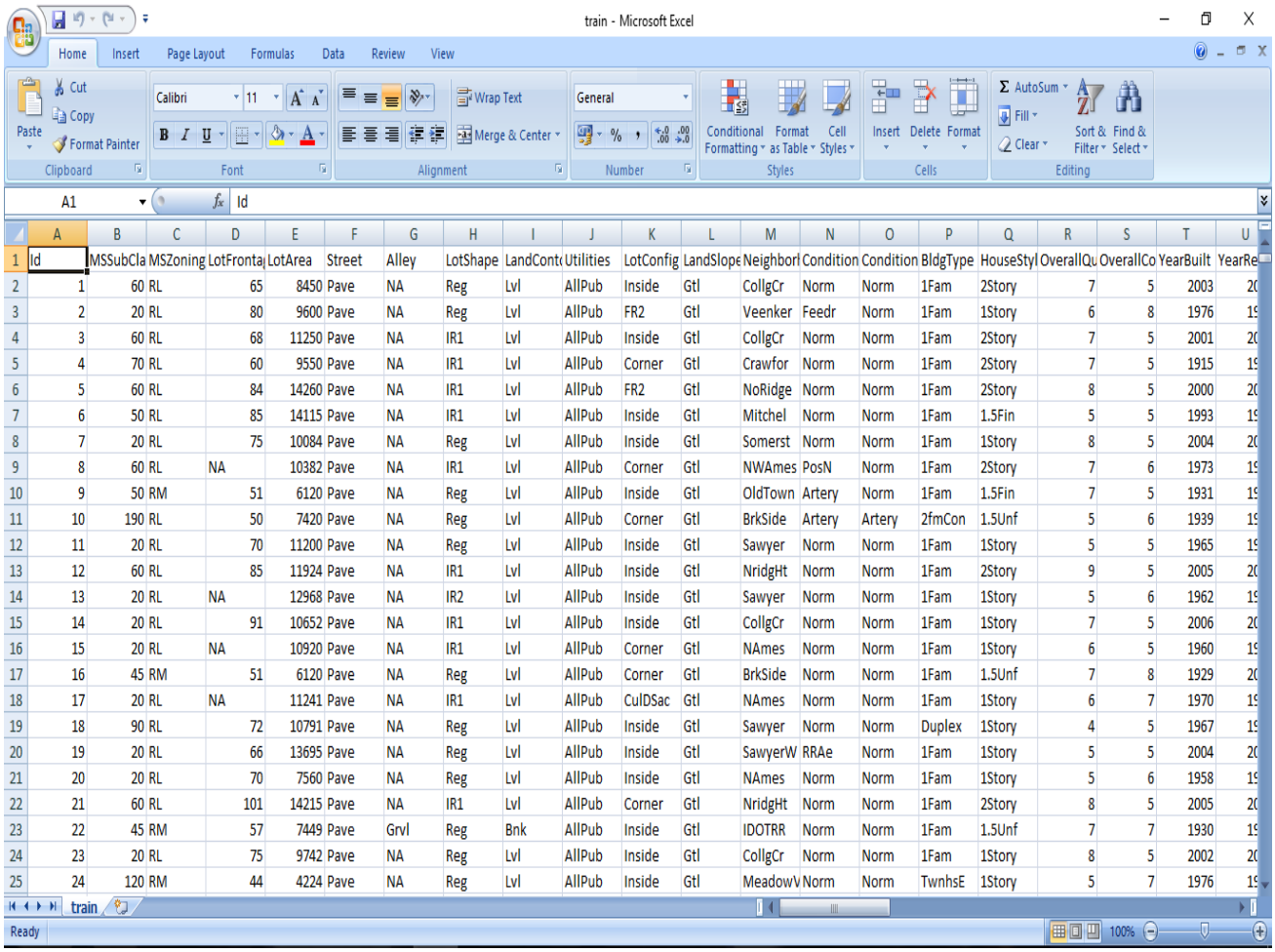
Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

The idea of gradient boosting can be interpreted as an optimization algorithm on a suitable cost function. Boosting algorithms as iterative *functional gradient descent* algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

DATASET

Here we use the Housing Price dataset provided by kaggle. In this the train file has 1460 observations and the test file has 1459 observations. Both datasets contain 79 explanatory variables composed of 46 categorical and 33 continuous variables that describe house features such as neighborhood, square footage, number of full bathrooms, and many more. The train file contains a response variable column, SalePrice, which is what we will predict in the test set. There is also a unique ID for each house sold, but were not used in fitting the models.



Id	MSSubCla	MSZoning	LotFronta	LotArea	Street	Alley	LotShape	LandCont	Utilities	LotConfig	LandSlope	Neighborh	Condition	Condition	BldgType	HouseStyl	OverallQu	OverallCo	YearBuilt	YearRe
1	60	RL	65	8450	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	7	5	2003	20
2	20	RL	80	9600	Pave	NA	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	1Fam	1Story	6	8	1976	19
3	60	RL	68	11250	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	7	5	2001	20
4	70	RL	60	9550	Pave	NA	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	1Fam	2Story	7	5	1915	19
5	60	RL	84	14260	Pave	NA	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	1Fam	2Story	8	5	2000	20
6	50	RL	85	14115	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	Norm	1Fam	1.5Fin	5	5	1993	19
7	20	RL	75	10084	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	Norm	1Fam	1Story	8	5	2004	20
8	60	RL	NA	10382	Pave	NA	IR1	Lvl	AllPub	Corner	Gtl	NWAmes	PosN	Norm	1Fam	2Story	7	6	1973	19
9	50	RM	51	6120	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	OldTown	Artery	Norm	1Fam	1.5Fin	7	5	1931	19
10	190	RL	50	7420	Pave	NA	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Artery	Artery	2fmCon	1.5Unf	5	6	1939	19
11	20	RL	70	11200	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	Norm	1Fam	1Story	5	5	1965	19
12	60	RL	85	11924	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	NridgHt	Norm	Norm	1Fam	2Story	9	5	2005	20
13	20	RL	NA	12968	Pave	NA	IR2	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	Norm	1Fam	1Story	5	6	1962	19
14	20	RL	91	10652	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	1Story	7	5	2006	20
15	20	RL	NA	10920	Pave	NA	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam	1Story	6	5	1960	19
16	45	RM	51	6120	Pave	NA	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Norm	Norm	1Fam	1.5Unf	7	8	1929	20
17	20	RL	NA	11241	Pave	NA	IR1	Lvl	AllPub	CulDSac	Gtl	NAmes	Norm	Norm	1Fam	1Story	6	7	1970	19
18	90	RL	72	10791	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	Norm	Duplex	1Story	4	5	1967	19
19	20	RL	66	13695	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	SawyerW	RRAe	Norm	1Fam	1Story	5	5	2004	20
20	20	RL	70	7560	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	Norm	1Fam	1Story	5	6	1958	19
21	60	RL	101	14215	Pave	NA	IR1	Lvl	AllPub	Corner	Gtl	NridgHt	Norm	Norm	1Fam	2Story	8	5	2005	20
22	45	RM	57	7449	Pave	Grvl	Reg	Bnk	AllPub	Inside	Gtl	IDOTRR	Norm	Norm	1Fam	1.5Unf	7	7	1930	19
23	20	RL	75	9742	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	1Story	8	5	2002	20
24	120	RM	44	4224	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	MeadowV	Norm	Norm	TwnhsE	1Story	5	7	1976	19

train - Microsoft Excel

A1 Id

	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO
1	YearRemc	RoofStyle	RoofMatl	Exterior1s	Exterior2r	MasVnrTy	MasVnrAr	ExterQual	ExterConc	Foundatic	BsmtQual	BsmtConc	BsmtExpo	BsmtFinTy	BsmtFinSf	BsmtFinTy	BsmtFinSf	BsmtUnfS	TotalBsmt	Heating	Heatin
2	2003	Gable	CompShg	VinylSd	VinylSd	BrkFace	196	Gd	TA	PConc	Gd	TA	No	GLQ	706	Unf	0	150	856	GasA	Ex
3	1976	Gable	CompShg	MetalSd	MetalSd	None	0	TA	TA	CBlock	Gd	TA	Gd	ALQ	978	Unf	0	284	1262	GasA	Ex
4	2002	Gable	CompShg	VinylSd	VinylSd	BrkFace	162	Gd	TA	PConc	Gd	TA	Mn	GLQ	486	Unf	0	434	920	GasA	Ex
5	1970	Gable	CompShg	Wd Sdng	Wd Shng	None	0	TA	TA	BrkTil	TA	Gd	No	ALQ	216	Unf	0	540	756	GasA	Gd
6	2000	Gable	CompShg	VinylSd	VinylSd	BrkFace	350	Gd	TA	PConc	Gd	TA	Av	GLQ	655	Unf	0	490	1145	GasA	Ex
7	1995	Gable	CompShg	VinylSd	VinylSd	None	0	TA	TA	Wood	Gd	TA	No	GLQ	732	Unf	0	64	796	GasA	Ex
8	2005	Gable	CompShg	VinylSd	VinylSd	Stone	186	Gd	TA	PConc	Ex	TA	Av	GLQ	1369	Unf	0	317	1686	GasA	Ex
9	1973	Gable	CompShg	HdBoard	HdBoard	Stone	240	TA	TA	CBlock	Gd	TA	Mn	ALQ	859	BLQ	32	216	1107	GasA	Ex
10	1950	Gable	CompShg	BrkFace	Wd Shng	None	0	TA	TA	BrkTil	TA	TA	No	Unf	0	Unf	0	952	952	GasA	Gd
11	1950	Gable	CompShg	MetalSd	MetalSd	None	0	TA	TA	BrkTil	TA	TA	No	GLQ	851	Unf	0	140	991	GasA	Ex
12	1965	Hip	CompShg	HdBoard	HdBoard	None	0	TA	TA	CBlock	TA	TA	No	Rec	906	Unf	0	134	1040	GasA	Ex
13	2006	Hip	CompShg	WdShing	Wd Shng	Stone	286	Ex	TA	PConc	Ex	TA	No	GLQ	998	Unf	0	177	1175	GasA	Ex
14	1962	Hip	CompShg	HdBoard	Plywood	None	0	TA	TA	CBlock	TA	TA	No	ALQ	737	Unf	0	175	912	GasA	TA
15	2007	Gable	CompShg	VinylSd	VinylSd	Stone	306	Gd	TA	PConc	Gd	TA	Av	Unf	0	Unf	0	1494	1494	GasA	Ex
16	1960	Hip	CompShg	MetalSd	MetalSd	BrkFace	212	TA	TA	CBlock	TA	TA	No	BLQ	733	Unf	0	520	1253	GasA	TA
17	2001	Gable	CompShg	Wd Sdng	Wd Sdng	None	0	TA	TA	BrkTil	TA	TA	No	Unf	0	Unf	0	832	832	GasA	Ex
18	1970	Gable	CompShg	Wd Sdng	Wd Sdng	BrkFace	180	TA	TA	CBlock	TA	TA	No	ALQ	578	Unf	0	426	1004	GasA	Ex
19	1967	Gable	CompShg	MetalSd	MetalSd	None	0	TA	TA	Slab	NA	NA	NA	NA	0	NA	0	0	0	GasA	TA
20	2004	Gable	CompShg	VinylSd	VinylSd	None	0	TA	TA	PConc	TA	TA	No	GLQ	646	Unf	0	468	1114	GasA	Ex
21	1965	Hip	CompShg	BrkFace	Plywood	None	0	TA	TA	CBlock	TA	TA	No	LwQ	504	Unf	0	525	1029	GasA	TA
22	2006	Gable	CompShg	VinylSd	VinylSd	BrkFace	380	Gd	TA	PConc	Ex	TA	Av	Unf	0	Unf	0	1158	1158	GasA	Ex
23	1950	Gable	CompShg	Wd Sdng	Wd Sdng	None	0	TA	TA	PConc	TA	TA	No	Unf	0	Unf	0	637	637	GasA	Ex
24	2002	Hip	CompShg	VinylSd	VinylSd	BrkFace	281	Gd	TA	PConc	Gd	TA	No	Unf	0	Unf	0	1777	1777	GasA	Ex
25	1976	Gable	CompShg	CemntBd	CmentBd	None	0	TA	TA	PConc	Gd	TA	No	GLQ	840	Unf	0	200	1040	GasA	TA

train

Ready

train - Microsoft Excel

A1 | Id

	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI
1	HeatingQ(	CentralAir	Electrical	1stFlrSF	2ndFlrSF	LowQualF	GrLivArea	BsmtFullB	BsmtHalfB	FullBath	HalfBath	Bedroom/	KitchenAb	KitchenQu	TotRmsAb	Functiona	Fireplaces	Fireplace(	GarageTyp	GarageYrB	Garage
2	Ex	Y	SBrkr	856	854	0	1710	1	0	2	1	3	1	Gd	8	Typ	0	NA	Attchd	2003	RFn
3	Ex	Y	SBrkr	1262	0	0	1262	0	1	2	0	3	1	TA	6	Typ	1	TA	Attchd	1976	RFn
4	Ex	Y	SBrkr	920	866	0	1786	1	0	2	1	3	1	Gd	6	Typ	1	TA	Attchd	2001	RFn
5	Gd	Y	SBrkr	961	756	0	1717	1	0	1	0	3	1	Gd	7	Typ	1	Gd	Detchd	1998	Unf
6	Ex	Y	SBrkr	1145	1053	0	2198	1	0	2	1	4	1	Gd	9	Typ	1	TA	Attchd	2000	RFn
7	Ex	Y	SBrkr	796	566	0	1362	1	0	1	1	1	1	TA	5	Typ	0	NA	Attchd	1993	Unf
8	Ex	Y	SBrkr	1694	0	0	1694	1	0	2	0	3	1	Gd	7	Typ	1	Gd	Attchd	2004	RFn
9	Ex	Y	SBrkr	1107	983	0	2090	1	0	2	1	3	1	TA	7	Typ	2	TA	Attchd	1973	RFn
10	Gd	Y	FuseF	1022	752	0	1774	0	0	2	0	2	2	TA	8	Min1	2	TA	Detchd	1931	Unf
11	Ex	Y	SBrkr	1077	0	0	1077	1	0	1	0	2	2	TA	5	Typ	2	TA	Attchd	1939	RFn
12	Ex	Y	SBrkr	1040	0	0	1040	1	0	1	0	3	1	TA	5	Typ	0	NA	Detchd	1965	Unf
13	Ex	Y	SBrkr	1182	1142	0	2324	1	0	3	0	4	1	Ex	11	Typ	2	Gd	BuiltIn	2005	Fin
14	TA	Y	SBrkr	912	0	0	912	1	0	1	0	2	1	TA	4	Typ	0	NA	Detchd	1962	Unf
15	Ex	Y	SBrkr	1494	0	0	1494	0	0	2	0	3	1	Gd	7	Typ	1	Gd	Attchd	2006	RFn
16	TA	Y	SBrkr	1253	0	0	1253	1	0	1	1	2	1	TA	5	Typ	1	Fa	Attchd	1960	RFn
17	Ex	Y	FuseA	854	0	0	854	0	0	1	0	2	1	TA	5	Typ	0	NA	Detchd	1991	Unf
18	Ex	Y	SBrkr	1004	0	0	1004	1	0	1	0	2	1	TA	5	Typ	1	TA	Attchd	1970	Fin
19	TA	Y	SBrkr	1296	0	0	1296	0	0	2	0	2	2	TA	6	Typ	0	NA	CarPort	1967	Unf
20	Ex	Y	SBrkr	1114	0	0	1114	1	0	1	1	3	1	Gd	6	Typ	0	NA	Detchd	2004	Unf
21	TA	Y	SBrkr	1339	0	0	1339	0	0	1	0	3	1	TA	6	Min1	0	NA	Attchd	1958	Unf
22	Ex	Y	SBrkr	1158	1218	0	2376	0	0	3	1	4	1	Gd	9	Typ	1	Gd	BuiltIn	2005	RFn
23	Ex	Y	FuseF	1108	0	0	1108	0	0	1	0	3	1	Gd	6	Typ	1	Gd	Attchd	1930	Unf
24	Ex	Y	SBrkr	1795	0	0	1795	0	0	2	0	3	1	Gd	7	Typ	1	Gd	Attchd	2002	RFn
25	TA	Y	SBrkr	1060	0	0	1060	1	0	1	0	3	1	TA	6	Typ	1	TA	Attchd	1976	Unf

housing_train.csv - Excel (Product Activation Failed)

BR15 | 0

	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ	CA	CB	CC	CD
1	GarageYrB	GarageFini	GarageCar	GarageAre	GarageQu	GarageCo	PavedDriv	WoodDecl	OpenPorcl	EnclosedP	3SsnPorch	ScreenPor	PoolArea	PoolQC	Fence	MiscFeatu	MiscVal	MoSold	YrSold	SaleType	SaleCondit	SalePrice	
2	2003	RFn	2	548	TA	TA	Y	0	61	0	0	0	0	NA	NA	NA	0	2	2008	WD	Normal	208500	
3	1976	RFn	2	460	TA	TA	Y	298	0	0	0	0	0	NA	NA	NA	0	5	2007	WD	Normal	181500	
4	2001	RFn	2	608	TA	TA	Y	0	42	0	0	0	0	NA	NA	NA	0	9	2008	WD	Normal	223500	
5	1998	Unf	3	642	TA	TA	Y	0	35	272	0	0	0	NA	NA	NA	0	2	2006	WD	Abnorml	140000	
6	2000	RFn	3	836	TA	TA	Y	192	84	0	0	0	0	NA	NA	NA	0	12	2008	WD	Normal	250000	
7	1993	Unf	2	480	TA	TA	Y	40	30	0	320	0	0	NA	MnPrv	Shed	700	10	2009	WD	Normal	143000	
8	2004	RFn	2	636	TA	TA	Y	255	57	0	0	0	0	NA	NA	NA	0	8	2007	WD	Normal	307000	
9	1973	RFn	2	484	TA	TA	Y	235	204	228	0	0	0	NA	NA	Shed	350	11	2009	WD	Normal	200000	
10	1931	Unf	2	468	Fa	TA	Y	90	0	205	0	0	0	NA	NA	NA	0	4	2008	WD	Abnorml	129900	
11	1939	RFn	1	205	Gd	TA	Y	0	4	0	0	0	0	NA	NA	NA	0	1	2008	WD	Normal	118000	
12	1965	Unf	1	384	TA	TA	Y	0	0	0	0	0	0	NA	NA	NA	0	2	2008	WD	Normal	129500	
13	2005	Fin	3	736	TA	TA	Y	147	21	0	0	0	0	NA	NA	NA	0	7	2006	New	Partial	345000	
14	1962	Unf	1	352	TA	TA	Y	140	0	0	0	176	0	NA	NA	NA	0	9	2008	WD	Normal	144000	
15	2006	RFn	3	840	TA	TA	Y	160	33	0	0	0	0	NA	NA	NA	0	8	2007	New	Partial	279500	
16	1960	RFn	1	352	TA	TA	Y	0	213	176	0	0	0	NA	GdWo	NA	0	5	2008	WD	Normal	157000	
17	1991	Unf	2	576	TA	TA	Y	48	112	0	0	0	0	NA	GdPrv	NA	0	7	2007	WD	Normal	132000	
18	1970	Fin	2	480	TA	TA	Y	0	0	0	0	0	0	NA	NA	Shed	700	3	2010	WD	Normal	149000	
19	1967	Unf	2	516	TA	TA	Y	0	0	0	0	0	0	NA	NA	Shed	500	10	2006	WD	Normal	90000	
20	2004	Unf	2	576	TA	TA	Y	0	102	0	0	0	0	NA	NA	NA	0	6	2008	WD	Normal	159000	
21	1958	Unf	1	294	TA	TA	Y	0	0	0	0	0	0	NA	MnPrv	NA	0	5	2009	COD	Abnorml	139000	
22	2005	RFn	3	853	TA	TA	Y	240	154	0	0	0	0	NA	NA	NA	0	11	2006	New	Partial	325300	
23	1930	Unf	1	280	TA	TA	N	0	0	205	0	0	0	NA	GdPrv	NA	0	6	2007	WD	Normal	139400	
24	2002	RFn	2	534	TA	TA	Y	171	159	0	0	0	0	NA	NA	NA	0	9	2008	WD	Normal	230000	
25	1976	Unf	2	572	TA	TA	Y	100	110	0	0	0	0	NA	NA	NA	0	6	2007	WD	Normal	129900	
26	1968	Unf	1	270	TA	TA	Y	406	90	0	0	0	0	NA	MnPrv	NA	0	5	2010	WD	Normal	154000	
27	2007	RFn	3	890	TA	TA	Y	0	56	0	0	0	0	NA	NA	NA	0	7	2009	WD	Normal	256300	
28	2005	Unf	2	576	TA	TA	Y	222	32	0	0	0	0	NA	NA	NA	0	5	2010	WD	Normal	134800	
29	2008	RFn	3	772	TA	TA	Y	0	50	0	0	0	0	NA	NA	NA	0	5	2010	WD	Normal	306000	
30	1957	RFn	1	319	TA	TA	Y	288	258	0	0	0	0	NA	NA	NA	0	12	2006	WD	Normal	207500	
31	1920	Unf	1	240	Fa	TA	Y	49	0	87	0	0	0	NA	NA	NA	0	5	2008	WD	Normal	68500	
32	1920	Unf	1	250	TA	Fa	N	0	54	172	0	0	0	NA	MnPrv	NA	0	7	2008	WD	Normal	40000	

train - Microsoft Excel

A1 Id

	CA	CB	CC
4	WD	Normal	223500
5	WD	Abnorml	140000
6	WD	Normal	250000
7	WD	Normal	143000
8	WD	Normal	307000
9	WD	Normal	200000
10	WD	Abnorml	129900
11	WD	Normal	118000
12	WD	Normal	129500
13	New	Partial	345000
14	WD	Normal	144000
15	New	Partial	279500
16	WD	Normal	157000
17	WD	Normal	132000
18	WD	Normal	149000
19	WD	Normal	90000
20	WD	Normal	159000
21	COD	Abnorml	139000
22	New	Partial	325300
23	WD	Normal	139400
24	WD	Normal	230000
25	WD	Normal	129900
26	WD	Normal	154000
27	WD	Normal	256300
28	WD	Normal	134800

Research kernels

Our first Kernel (made by Dan Beck)

Model Tuning

XGBoost has a few parameters that can dramatically affect model's accuracy and training speed. The first parameters are:

n_estimators and early_stopping_rounds

n_estimators specifies how many times to go through the modeling cycle described above.

In the underfitting vs overfitting graph, n_estimators moves further to the right. Too low a value causes underfitting, which is inaccurate predictions on both training data and new data. Too large a value causes overfitting, which is accurate predictions on training data, but inaccurate predictions on new data (which is what we care about). Typical values range from 100-1000, though this depends a lot on the **learning rate** discussed below.

The argument **early_stopping_rounds** offers a way to automatically find the ideal value. Early stopping causes the model to stop iterating when the validation score stops improving, even if we aren't at the hard stop for n_estimators. It's smart to set a high value for **n_estimators** and then use **early_stopping_rounds** to find the optimal time to stop iterating.

Since random chance sometimes causes a single round where validation scores don't improve, you need to specify a number for how many rounds of straight deterioration to allow before stopping. **early_stopping_rounds = 5** is a reasonable value. Thus we stop after 5 straight rounds of deteriorating validation scores.

Learning rate

Here's a subtle but important trick for better XGBoost models:

Instead of getting predictions by simply adding up the predictions from each component model, we will multiply the predictions from each model by a small number before adding them in. This means each tree we add to the ensemble helps us less. In practice, this reduces the model's propensity to overfit.

So, you can use a higher value of **n_estimators** without overfitting. If you use early stopping, the appropriate number of trees will be set automatically.

In general, a small learning rate (and large number of estimators) will yield more accurate XGBoost models, though it will also take the model longer to train since it does more iterations through the cycle.

n_jobs

On larger datasets where runtime is a consideration, you can use parallelism to build your models faster. It's common to set the parameter **n_jobs** equal to the number of cores on your machine. On smaller datasets, this won't help.

The resulting model won't be any better, so micro-optimizing for fitting time is typically nothing but a distraction. But, it's useful in large datasets where you would otherwise spend a long time waiting during the fit command.

XGBoost has a multitude of other parameters, but these will go a very long way in helping you fine-tune your XGBoost model for optimal performance.

Conclusion

XGBoost is currently the dominant algorithm for building accurate models on conventional data (also called tabular or strutured data).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Imputer

data = pd.read_csv('../input/train.csv')
data.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = data.SalePrice
X = data.drop(['SalePrice'], axis=1).select_dtypes(exclude=['object'])
train_X, test_X, train_y, test_y = train_test_split(X.as_matrix(), y.as_mat
rix(), test_size=0.25)

my_imputer = Imputer()
train_X = my_imputer.fit_transform(train_X)
test_X = my_imputer.transform(test_X)
```

We build and fit a model just as we would in scikit-learn.

```
In [2]:
from xgboost import XGBRegressor

my_model = XGBRegressor()
# Add silent=True to avoid printing out updates with each cycle
my_model.fit(train_X, train_y, verbose=False)
```

We similarly evaluate a model and make predictions as we would do in scikit-learn.

```
In [3]:
# make predictions
predictions = my_model.predict(test_X)

from sklearn.metrics import mean_absolute_error
```

```
print("Mean Absolute Error : " + str(mean_absolute_error(predictions, test_
y)))
```

In [4]:

```
my_model = XGBRegressor(n_estimators=1000)
my_model.fit(train_X, train_y, early_stopping_rounds=5,
             eval_set=[(test_X, test_y)], verbose=False)
```

In [5]:

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)
my_model.fit(train_X, train_y, early_stopping_rounds=5,
             eval_set=[(test_X, test_y)], verbose=False)
```

Model Accuracy -85%

Our Second kernel

Regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting

Data pre-processing:

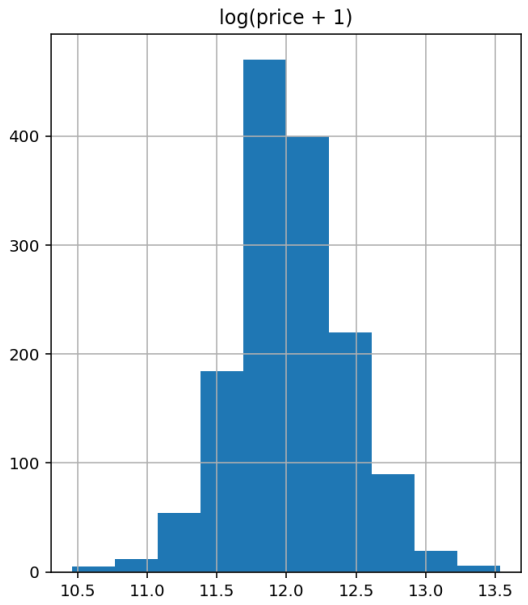
We're not going to do anything fancy here:

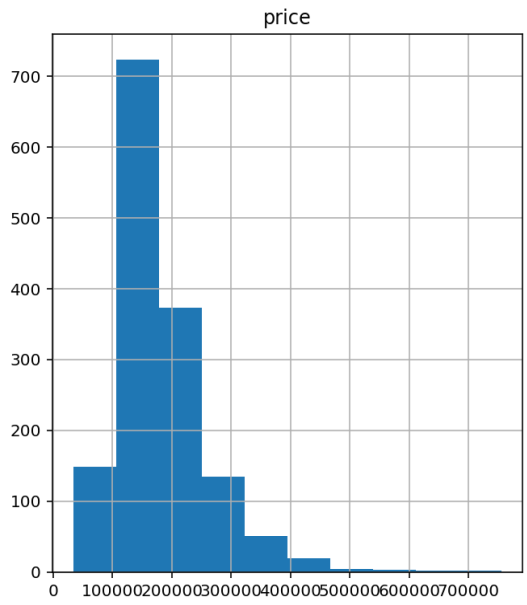
- First we'll transform the skewed numeric features by taking log(feature + 1) - this will make the features more normal
- Create Dummy variables for the categorical features
- Replace the numeric missing values (NaN's) with the mean of their respective columns

Models

After that we are going to use regularized linear regression models from the scikit learn module. I'm going to try both l_1(Lasso) and l_2(Ridge) regularization. We'll also define a function that returns the cross-validation rmse error so we can evaluate our models and pick the best tuning par

The main tuning parameter for the Ridge model is alpha - a regularization parameter that measures how flexible our model is. The higher the regularization the less prone our model will be to overfit. However it will also lose flexibility and might not capture all of the signal in the data.





Implementation

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)

my_model.fit(train_X, train_y, early_stopping_rounds=5,

             eval_set=[(test_X, test_y)], verbose=False)

all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],

                      test.loc[:,'MSSubClass':'SaleCondition']))

matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)

prices = pd.DataFrame({"price":train["SalePrice"], "log(price +
1)":np.log1p(train["SalePrice"])})

prices.hist()

#log transform the target:

train["SalePrice"] = np.log1p(train["SalePrice"])



#log transform skewed numeric features:

numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index



skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
#compute skewness

skewed_feats = skewed_feats[skewed_feats > 0.75]

skewed_feats = skewed_feats.index



all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

all_data = pd.get_dummies(all_data)

In [8]:
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())

In [9]:
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice

from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, Lasso
LarsCV
from sklearn.model_selection import cross_val_score

def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y, scoring="neg_mean_squ
ared_error", cv = 5))
```

```
    return(rmse)
```

In [11]:

```
model_ridge = Ridge()

alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]
```

In [13]:

```
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation - Just Do It")
plt.xlabel("alpha")
plt.ylabel("rmse")

cv_ridge.min()

model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)

rmse_cv(model_lasso).mean()

coef = pd.Series(model_lasso.coef_, index = X_train.columns)
```

In [18]:

```
print("Lasso picked " + str(sum(coef != 0)) + " variables and eliminated th
e other " +  str(sum(coef == 0)) + " variables")

imp_coef = pd.concat([coef.sort_values().head(10),
                      coef.sort_values().tail(10)])
```

In [20]:

```
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = "barh")
plt.title("Coefficients in the Lasso Model")

#let's look at the residuals as well:

matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)


preds = pd.DataFrame({"preds":model_lasso.predict(X_train), "true":y})

preds["residuals"] = preds["true"] - preds["preds"]

preds.plot(x = "preds", y = "residuals",kind = "scatter")

import xgboost as xgb
```

In [23]:

```
dtrain = xgb.DMatrix(X_train, label = y)
dtest = xgb.DMatrix(X_test)

params = {"max_depth":2, "eta":0.1}
model = xgb.cv(params, dtrain,  num_boost_round=500, early_stopping_rounds=
100)
```

In [24]:

```
model.loc[30:,["test-rmse-mean", "train-rmse-mean"]].plot()

model_xgb = xgb.XGBRegressor(n_estimators=360, max_depth=2, learning_rate=0
.1) #the params were tuned using xgb.cv

model_xgb.fit(X_train, y)

xgb_preds = np.expm1(model_xgb.predict(X_test))
lasso_preds = np.expm1(model_lasso.predict(X_test))
```

In [27]:

```
predictions = pd.DataFrame({"xgb":xgb_preds, "lasso":lasso_preds})
predictions.plot(x = "xgb", y = "lasso", kind = "scatter")
```

```
reds = 0.7*lasso_preds + 0.3*xgb_preds
```

In [29]:

```
solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
solution.to_csv("ridge_sol.csv", index = False)
```

```
from keras.layers import Dense
```

```
from keras.models import Sequential
```

```
from keras.regularizers import l1
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split
```

```
X_train = StandardScaler().fit_transform(X_train)
```

In [32]:

```
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y, random_state = 3)
```

In [33]:

```
X_tr.shape
```

```
X_tr
```

```
model = Sequential()
```

```
#model.add(Dense(256, activation="relu", input_dim = X_train.shape[1]))
```

```
model.add(Dense(1, input_dim = X_train.shape[1], W_regularizer=l1(0.001)))


model.compile(loss = "mse", optimizer = "adam")
```

```
hist = model.fit(X_tr, y_tr, validation_data = (X_val, y_val))
pd.Series(model.predict(X_val)[:,0]).hist()
```

Model Accuracy - 88%

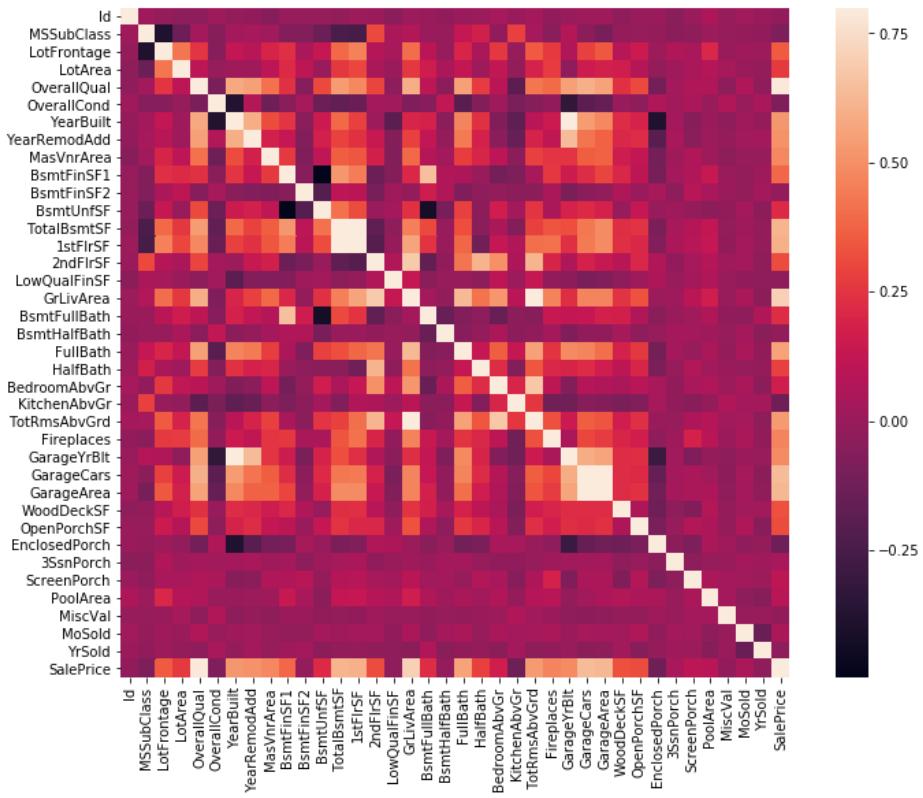
Our Proposed Solution

In both the above kernels the importance of feature selection and attribute correlation is being ignored. So we thought of finding how each attribute is being related to the other attribute in the dataset and plotted a heat map of every numeric attribute in the dataset.

At first sight, there are two red colored squares that get our attention. The first one refers to the 'TotalBsmtSF' and '1stFlrSF' variables, and the second one refers to the 'Garage*X*' variables. Both cases show how significant the correlation is between these variables. Actually, this correlation is so strong that it can indicate a situation of multicollinearity. If we think about these variables, we can conclude that they give almost the same information so multicollinearity really occurs. Heatmaps are great to detect this kind of situations and in problems dominated by feature selection, like ours, they are an essential tool.

We observed the following trends in our dataset:

- 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'.
- 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. However, as we discussed in the last sub-point, the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are like twin brothers. You'll never be able to distinguish them. Therefore, we just need one of these variables in our analysis (we can keep 'GarageCars' since its correlation with 'SalePrice' is higher).
- 'TotalBsmtSF' and '1stFloor' also seem to be twin brothers
- 'TotRmsAbvGrd' and 'GrLivArea', twin brothers again.
- It seems that 'YearBuilt' is slightly correlated with 'SalePrice'.



Implementation

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from sklearn.preprocessing import Imputer

import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

%config InlineBackend.figure_format = 'r

train = pd.read_csv("train.csv") test = pd.read_csv("test.csv")

all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.
log1p(train["SalePrice"])})
prices.hist()

train["SalePrice"] = np.log1p(train["SalePrice"])
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #
compute skewness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
```

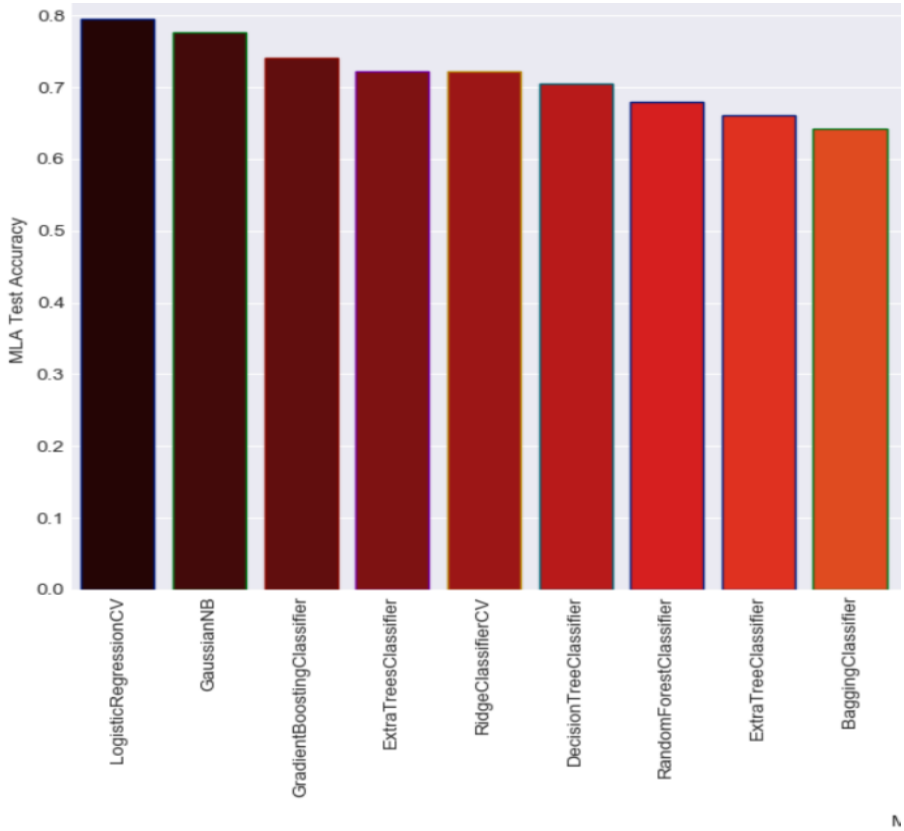
```
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

all_data = pd.get_dummies(all_data)
all_data = all_data.fillna(all_data.mean())

X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice

all_data_2 = X_train
all_data_2['SalePrice'] = y
corrmat = all_data_2.corr()
cols = corrmat.nlargest(200,'SalePrice')['SalePrice'].index
cols=cols[1:]
X_train = X_train[cols]
X_test = X_test[cols]

from sklearn.model_selection import train_test_split
X_train_train, X_train_test, y_train, y_test = train_test_split(X_train
.as_matrix(), y.as_matrix(), test_size=0.2)

from sklearn.cross_validation import cross_val_score
from xgboost import XGBRegressor

model_xgb = XGBRegressor(n_estimators=360, max_depth=2, learning_rate=0
.1)
scores = cross_val_score(estimator=model_xgb,X=X_train_train,y=y_train,
cv=20,n_jobs=-1)

model_xgb.fit(X_train_train, y_train)
model_xgb.score(X_train_test,y_test)
my_imputer = Imputer()
X_test = my_imputer.fit_transform(X_test)

xgb_preds = np.expm1(model_xgb.predict(X_test))

Id = test['Id']
sub = pd.DataFrame()
sub['Id'] =Id
sub['SalePrice']=xgb_preds
sub.to_csv("Model3.csv",index=False)
```

Model Accuracy - 91%

Result

As we can see the First kernel gives us an accuracy of 85% whereas the second kernel is better with accuracy of 88%

We however created an even more efficient kernel which gives us an efficiency of 91%



Here we have compared different classifiers on our dataset and as we can see the XGBoost comes in at 3ed in terem's of accuracy.

Behind Gaussian Naïve Bayes and Logistic regression.

Now as we know we can try our kernel with these classifiers and further improve our accuracy