Nanodegree Engenheiro de Machine Learning

Projeto final

Márcio Alexandre Costa
13 de maio de 2019

I. Definição

Visão geral do projeto

Construir um modelo que possa prever o valor de venda das casas com base na localização e característica da casa ideal.

Descrição do problema

O objetivo é criar um modelo de Aprendizagem de Máquinas para prever o valor de venda de casa em Ames / Iowa nos Estados Unidos.

Com base na descrição da casa ideal, como localização, quantidade de quartos, banheiros, tamanho da casa e muitas outras descrições (79 descrições estão disponíveis no dataset deste projeto) que o comprador pode usar para descrever e ser usado pelo modelo para prever o valor de venda destas casas.

O modelo poderá servir para balizar o sonho da casa ideal verso a valor que se deseja ou pode pagar por esta "casa dos sonhos".

Métricas

O modelo será avaliado pelo RMSE (Root Mean Squared Error) entre o logaritmo do valor previsto e o logaritmo do valor de venda observado.

$$\mathrm{RMSE}_{fo} = [\sum_{i=1}^{N} (z_{f_i} - z_{o_i})^2 / N]^{1/2}$$

Σ = summation ("add up")
(zfi – Zoi)Sup>2 = differences, squared
N = sample size.

II. Análise

Exploração dos dados

O dataset deste projeto está disponibilizado no Kaggle, no link https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data.

O conjunto de dados contém 79 características e 1.460 entradas, sendo 43 características com valores categóricas (não numéricas) e 36 características com valores numéricas. O conjunto de dados contém 19 características com valores ausentes (não preenchidos).

Segue abaixo uma pequena amostra do dataset que será trabalhado.

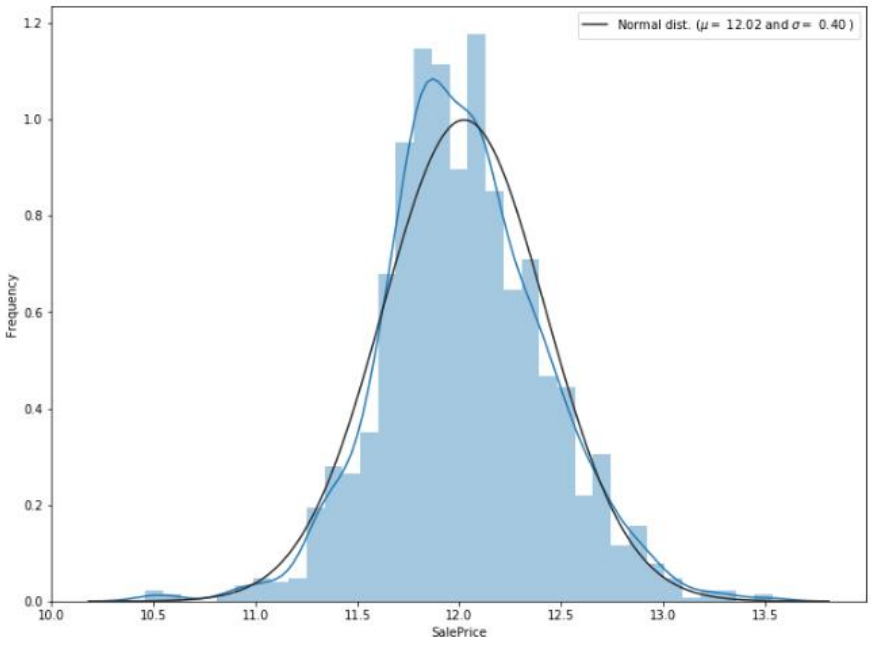
```
[ ] df_train = pd.read_csv('gdrive/My Drive/Colab Notebooks/train.csv')
    df_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

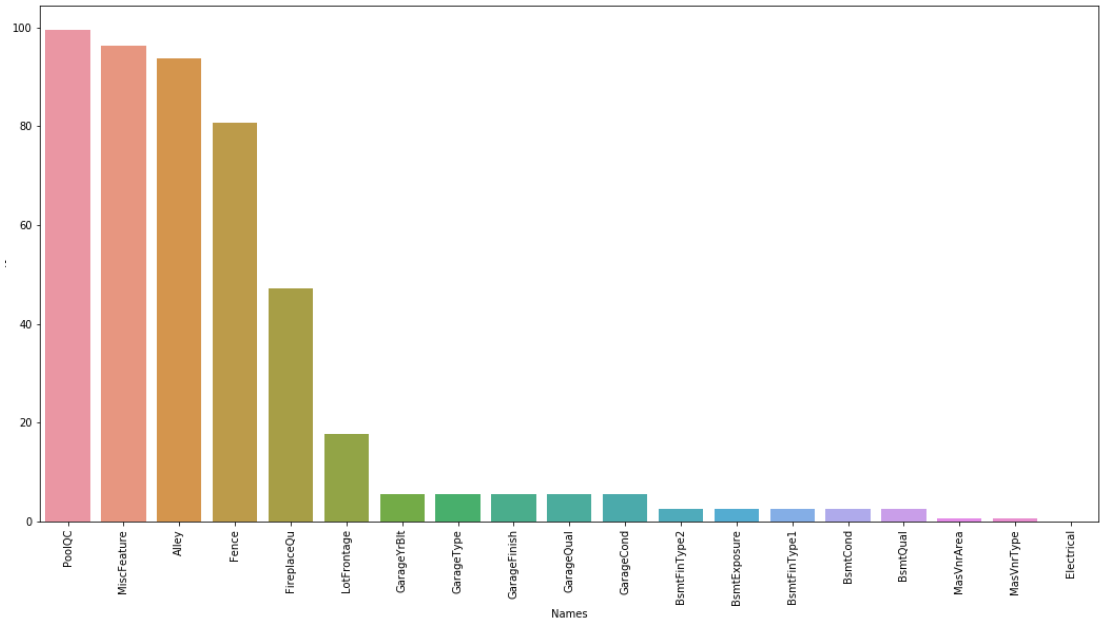
5 rows × 81 columns

Visualização exploratória

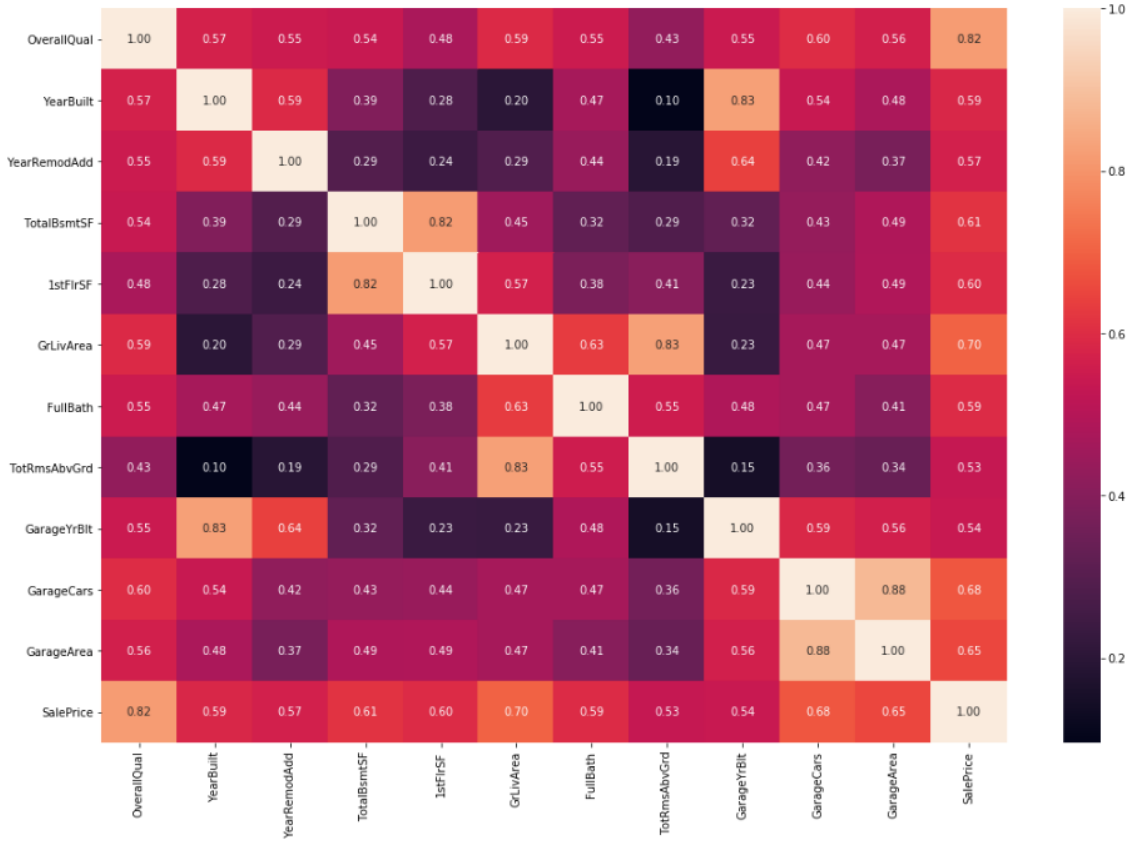
O gráfico da variável SalePrice do dataset (variável alvo, o modelo usará as características da casa para chegar no valor de venda da casa) após transformação logaritmo.



Estas são as variáveis (características da casa) do dataset que tem dados não preenchidos e com as porcentagens de dados ausentes em cada uma destas variáveis.



Abaixo um gráfico onde mostra as correlações das características da casa que tenham correlação acima de 0.5 (50%).



Algoritmos e técnicas

Utilizados algoritmos de Regressão da aprendizagem de máquina supervisionada. Os algoritmos de regressão trabalhados nesta proposta são:

- Lasso;
- Simple e/ou Multiple Linear Regression;
- Support Vector Regression (SVR);
- Decision Tree Regression;
- Random Forest regression.

Na primeira abordagem, os parâmetros utilizados nestes algoritmos são parâmetros padrões, sem nenhuma customização, para atingir um melhor resultado.

Após a definição de qual é o melhor algoritmo, irei usar o GridSearch, para chegar nos melhores parâmetros deste algoritmo.

Benchmark

Iremos criar um modelo com o algoritmo 'Naive' de Regressão para utilizar como modelo de benchmark comparando com os algoritmos de regressão mais avançados.

O resultado do algoritmo DummyRegressor que foi utilizado como algoritmo de regressão 'Naive' teve o RMSE de:

```
RMSE on Training set : 0.13071262536971964
RMSE on Test set : 0.1269396054397447
```

III. Metodologia

Pré-processamento de dados

Após entendimentos dos dados, foram feitos os seguintes pré-processamentos:

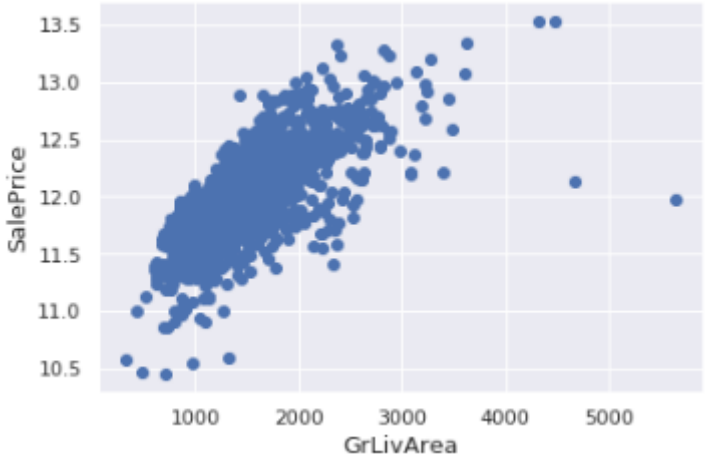
- Tratamento dos dados ausentes dos dados.

MiscFeature = Preenchido com None (No Misc Feature).
Alley = Preenchido com None (No Alley access).
Fence = Preenchido com None (No Fence).
FireplaceQu = Preenchido com None (No FireplaceQu).
LotFrontage = Preenchido com o valor mediano do bairro.
GarageType | GarageFinish | GarageQual | GarageCond = Todas foram preenchidas com None (No Garage).
GarageYrBlt | GarageArea | GarageCars = Todas foram preenchidas com 0.
BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath' = Preenchido com 0.
'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2' = Preenchido com None (No basement).
MasVnrType = Preenchido com None (masonry veneer).
MasVnrArea = Preenchido com 0 (masonry veneer).
MSZoning = Preenchido com RL.
Utilities = Eliminado do dataset (Todos os dados no dataset tem o mesmo valor – AllPub).
Functional = Preenchido com "Typ" (significa típico).
Electrical = Preenchido com o valor mais frequente ‘SBrkr’.
KitchenQual = Preenchido com o valor mais frequente ‘TA’.
Exterior1st = Preenchido com o valor mais frequente.
Exterior2nd = Preenchido com o valor mais frequente.
SaleType = Preenchido com o valor mais frequente ‘WD’.
MSSubClass = Preenchido com None (No building class).
PoolQC = Eliminado do dataset.

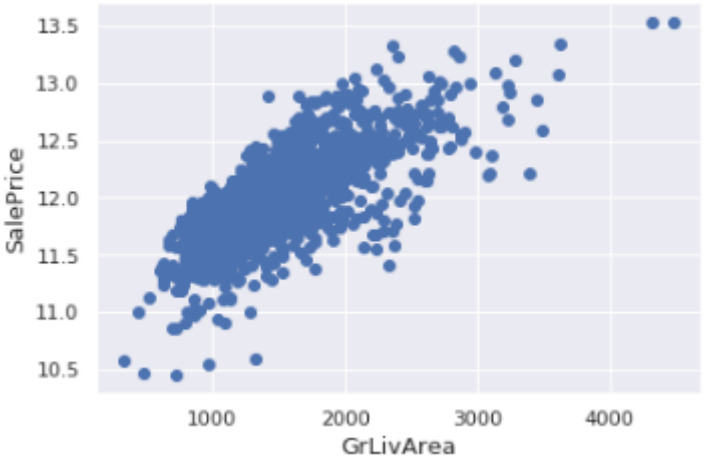
- Eliminação de Outliers observando a variável SalePrice.

Correlacionando os valores de SalePrice com GrLiveArea que são 2 variáveis numérica com maior correlação.

Antes



Depois de eliminado os 2 registros onde GrLiveArea maior que 4000 e SalePrice maior que 11.5.



- Normalização dos dados numéricos utilizando o MinMaxScaler da biblioteca Scikit Learn.
- Convertendo algumas variáveis pseudo numéricas para variáveis categóricas.

O dataset passou a ter a dimensão de 1458 registros e 333 variáveis.

- Por último, convertendo as variáveis categóricas em variáveis ‘dummy’, com valor 0 ou 1.

```
[27] df_train.head()
```

	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	...	SaleType_ConLw	SaleType_New	SaleType_Oth	SaleType_WD	SaleCondition_Abnorml	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCo
0	0.150685	0.033420	0.666667	0.949275	0.883333	0.12250	0.322669	0.0	0.064212	0.266999	...	0	0	0	1	0	0	0	
1	0.202055	0.038795	0.555556	0.753623	0.433333	0.00000	0.446984	0.0	0.121575	0.393637	...	0	0	0	1	0	0	0	
2	0.160959	0.046507	0.666667	0.934783	0.866667	0.10125	0.222121	0.0	0.185788	0.286962	...	0	0	0	1	0	0	0	
3	0.133562	0.038561	0.666667	0.311594	0.333333	0.00000	0.098720	0.0	0.231164	0.235808	...	0	0	0	1	1	0	0	
4	0.215753	0.060576	0.777778	0.927536	0.833333	0.21875	0.299360	0.0	0.209760	0.357143	...	0	0	0	1	0	0	0	

5 rows × 333 columns

Implementação

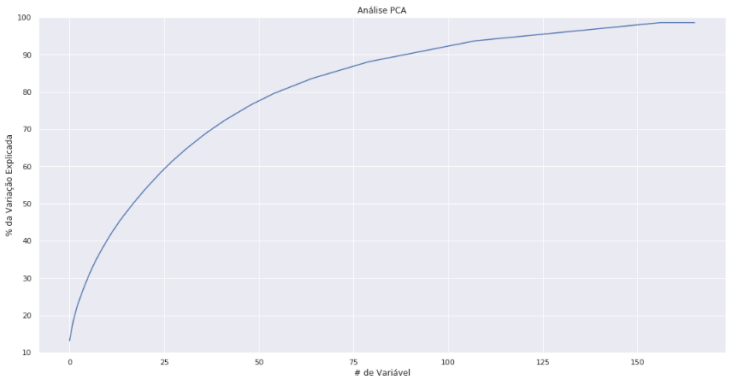
Foi criado uma função ‘train_predict’ para servi de pipeline dos modelos.

```
[70] # Criação do pipeline de treino dos modelos

     def train_predict(learner, X_train, y_train, X_test, y_test):
         '''
         inputs:
            - learner: the learning algorithm to be trained and predicted on
            - X_train: features training set
            - y_train: income training set
            - X_test: features testing set
            - y_test: income testing set
         '''

         results = {}

         # Medindo o tempo de criação do modelo
         start = time()
         learner = learner.fit(X_train, y_train)
         end = time()

         # Calculando tempo de treino
         results['train_time'] = end - start

         # medindo o tempo de prever
         start = time() # Get start time
         predictions_train = learner.predict(X_train)
         predictions_test = learner.predict(X_test)
         end = time() # Get end time

         # Calculando o tempo de prever
         results['pred_time'] = end - start

         # RMSE de Treino
         results['rmse_train'] = rmse_cv_train(learner).mean()

         # RMSE de Teste
         results['rmse_test'] = rmse_cv_test(learner).mean()

         # Success
         print("Modelo {} demorou {} de treino e {} para prever com o RMSE de treino de {} e o RMSE de teste {} samples.".format(learner.__class__.__name__, results['train_time'], results['pred_time'], results['rmse_train'], results['rmse_test']))

         # Return do resultados
         return results
```

A função acima tem como objetivo rodar todos os algoritmos definidos e medir o tempo de criação (fit), o tempo de predição, além de gerar o RMSE do dataset de treino e o de teste.

Apenas para o rodar o algoritmo “Linear Regression Simple”, com um única variável, foi preciso rodar fora do pipeline (função ‘train_predict’), devido a necessidade de extrair apenas um variável do dataset de treino e teste, que foi escolhida a variável 'GrLivArea' por ter uma correlação mais forte com a variável SalePrice.

Os modelos foram testados utilizado o dataset todo e com o dataset reduzido utilizando a técnica de PCA (Principal Component Analysis). Para explicar 85% do preço de venda de casa, são necessários utilizar no mínimo 75 características.



Refinamento

Comparando o tempo e o RMSE dos algoritmos com seus parâmetros padrões entre o dataset completo (df_desempenho) e o dataset reduzido (df_desempenho_pca com PCA), concluo que não há necessidade de utilização do PCA, como pode ver abaixo. [1]

```
[78] resultado = df_desempenho - df_desempenho_pca
     resultado
```

	DummyRegressor	Lasso	LinearRegression	SVR	DecisionTreeRegressor	RandomForestRegressor
pred_time	0.000021	0.001795	0.001669	0.015383	0.001900	0.001714
rmse_test	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
rmse_train	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
train_time	-0.000359	0.004143	0.049516	0.012399	-0.021146	-0.516016

[1] O jupyter notebook foi computado utilizando o serviço da Google Colab - https://colab.research.google.com. Pode ser que rodando este notebook em notebooks, possam obter outros resultados e outra conclusão.

Utilizado o GridSearch para definir os melhores parâmetros do **algoritmo que mais desempenho no dataset de teste.**

IV. Resultados

Modelo de avaliação e validação

Os parâmetros padrões utilizado para cada algoritmo foram:

```
# Inicialize os modelos

clf_A = DummyRegressor()
clf_B = linear_model.Lasso(alpha=0.1, random_state = 40)
clf_C = LinearRegression()
clf_D = SVR(gamma='scale', C=1.0, epsilon=0.2)
clf_E = DecisionTreeRegressor(random_state=40)
clf_F = RandomForestRegressor(max_depth=2, n_estimators=100, random_state=40)
```

O resultado dos desempenhos dos algoritmos com seus parâmetros padrões foram:

```
[73] df_desempenho = pd.DataFrame(desempenho)
     df_desempenho
```

	DummyRegressor	Lasso	LinearRegression	SVR	DecisionTreeRegressor	RandomForestRegressor
pred_time	0.000161	0.003680	4.600763e-03	0.024902	0.004576	0.016091
rmse_test	0.126940	0.126940	1.936900e+10	0.085973	0.076008	0.074165
rmse_train	0.130713	0.130713	1.174931e+09	0.074627	0.067612	0.075746
train_time	0.000899	0.006837	5.665088e-02	0.021402	0.043694	0.422792

Após a utilizando com GridSearch para automatização dos melhores parâmetros utilização do Randon Forest Regression o RMSE ficou:

```
Unoptimized model
------
Score on Test set : 0.6495593292427022
RMSE on Test set : 0.07416484852022152

Optimized Model
------
Score on Test set : 0.859131653368277
RMSE on Test set : 0.053563052292514804
```

Uma redução de ~28% do RMSE.

Justificativa

Comparando com o RMSE do benchmark, o qual foi utilizado o 'DummyRegressor' que se tinha os seguintes resultados:

```
RMSE on Training set : 0.13071262536971964
RMSE on Test set : 0.1269396054397447
```

Com os resultados obtidos com Random Forest Regression

```
RMSE on Test set : 0.053563052292514804
```

Temos uma melhora de ~58% em redução do erro calculado pelo RMSE.

Recomendamos a utilização do modelo de aprendizagem de máquina com o algoritmo Random Forest Regression para prever o valor de venda de casa em Ames / Iowa nos Estados Unidos com base nas 79 características da casa.

V. Conclusão

Forma livre de visualização

```
[78] resultado = df_desempenho - df_desempenho_pca
     resultado
```

	DummyRegressor	Lasso	LinearRegression	SVR	DecisionTreeRegressor	RandomForestRegressor
pred_time	0.000085	0.001439	-0.003647	0.016572	0.002398	0.001805
rmse_test	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
rmse_train	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
train_time	-0.000666	0.004936	0.041267	0.012957	-0.024661	-0.514149

Na visualização acima é uma comparação de desempenho (tempo) e eficiência (RMSE), como pode ver, com 75 características utilizando o PCA, pode se checar ao mesmo nível de eficiência quando utilizados as 333 características.

Para treinar os algoritmos DummyRegressor (Benchmark), DecisionRegressor e RandomForestRegressor, o tempo de ficou ligeiramente maior. E somente o Linear Regression que o tempo de prever maior com o dataset reduzido (PCA), lembrando que o LinearRegression foi o pior algoritmo para este dataset.

Reflexão

O processo seguindo neste projeto pode ser resumido assim:

- Definição do problema.
- Coleta de dados sobre o problema.
- Entendimento dos dados.
- Definição do benchmark para avaliação do modelo proposto posteriormente e qual seria a métrica de comparação.
- Pré processamento dos dados, trabalhar com os ausentes, outliers, correlação e até a redução de características.
- Pré Treinar o modelo e avaliar o resultado.
- Comparar o resultado e definir o melhor algoritmo de regressão para resolver este problema.
- Trabalhar com os melhores parâmetros do algoritmo escolhido.
- Comparação final.

Melhorias

Entendo que este projeto poderia ser enriquecido o treino de outros algoritmos de Regressão, tais como: Ridge, Elastic-Net, SGD (Stochastic Gradiente Descent) e outros.

O uso de modelo de Classificação onde transformaria a variável alvo em categórica, ou seja, preço de venda entre 100K-150K, 150K-200K, 200K-250K.... e utilizar modelos de classificação ao invés de Regressão poderia ser verificado. Como já realizado no estudo de Hujia Yu.[2]

[2] De Hujia Yu, Jiafu Wu. "Real Estate Price Prediction with Regression and Classification." CS 229 Autumn 2016 Project Final Report - http://cs229.stanford.edu/proj2016/report/WuYu_HousingPrice_report.pdf