Nanodegree Engenheiro de Machine Learning – Udacity

Proposta de projeto final - 09 de setembro de 2018

Autor: Marcos Paulo da Silva Falcirolli

I. Definição

Proposta do projeto

A proposta deste projeto é criar um modelo estatístico de capaz de estimar o valor de venda de um imóvel, este é um problema conhecido da área de Engenharia de Avaliação de Bens.

Em 1977 foi criada a NB-502, que foi a primeira norma brasileira para avaliação de imóveis urbanos, este foi um marco para o mercado imobiliário brasileiro, pois antes disso o valor de um imóvel era obtido de maneira subjetiva, o avaliador estimava o valor baseados em sua experiencia. Com esta norma foi determinado a necessidade de um modelo de regressão linear com níveis de precisão definidos para as avaliações, assim substituindo a subjetividade pela ciência.

Em 1989 surge a NBR 5676 que é uma revisão da NB-502. Em 2001 surge a NBR 14653 que tem sua versão de 2011 vigente até os dias de hoje. Em todas essas normas é imprescindível que os avaliadores de imóveis utilizem inferência estatística a partir de um modelo de regressão linear para avaliação de imóveis urbanos. (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2011)

Nos dias de hoje normalmente um engenheiro de avaliação de bens ao realizar um laudo coleta os dados quase dos imóveis quase que manualmente, porém com a informatização do setor esperamos que em breve os engenheiros tenham grandes bases de bigdata com esses dados assim causando uma possibilidade de mudança no setor. Com grandes volumes de dados nos permitirá aplicar técnicas estatísticas mais avançadas trazendo possivelmente resultados mais precisos e de maneira mais prática. Portanto a exploração de técnicas de regressão mais avançadas pode ser de grande contribuição para esta mudança.

O laudo de avalição de um imóvel que determina o seu valor tem inúmeras aplicações, como por exemplo: negociação de venda, financiamento imobiliário, hipotecas, separação de bens e outras.

Descrição do problema

O problema a ser resolvido é obter o preço de venda de um imóvel.

A solução proposta neste trabalho é criar um modelo utilizando técnicas de regressão, que terá com input diversas características do imóvel, assim poderemos estimar/inferir os preços de venda.

Métricas

A performance do modelo poderá ser medida através do Root-Mean-Squared-Error (RMSE) entre os valores de venda apresentados no conjunto de dados e os valores de venda previstos ou do coeficiente de determinação (R^2).

II. Análise dos dados

Exploração dos dados

O conjunto de dados neste trabalho será o conjunto da competição Kaggle “House Prices: Advanced Regression Techniques” que deve permitir uma boa aplicação de diversas técnicas de regressão e assim realizar um bom estudo e a aplicação da biblioteca do Scikit-learn.

Este conjunto é composto por setenta e nove variáveis dependentes que são características relacionados ao imóvel e uma variável dependente que é o valor de venda do imóvel ('SalePrice'). As *features* podem ser vistas na tabela abaixo ou no acessadas no site https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

Tabela 1 – Descrição das *features* (variáveis):

Feature	Descrição	Feature	Descrição
SalePrice	the property's sale price in dollars.	HeatingQC	Heating quality and condition
MSSubClass	The building class	CentralAir	Central air conditioning
MSZoning	The general zoning classification	Electrical	Electrical system
LotFrontage	Linear feet of street connected to property	1stFlrSF	First Floor square feet
LotArea	Lot size in square feet	2ndFlrSF	Second floor square feet
Street	Type of road access	LowQualFinSF	Low quality finished square feet (all floors)
Alley	Type of alley access	GrLivArea	Above grade (ground) living area square feet
LotShape	General shape of property	BsmtFullBath	Basement full bathrooms
LandContour	Flatness of the property	BsmtHalfBath	Basement half bathrooms
Utilities	Type of utilities available	FullBath	Full bathrooms above grade
LotConfig	Lot configuration	HalfBath	Half baths above grade
LandSlope	Slope of property	Bedroom	Number of bedrooms above basement level
Neighborhood	Physical locations within Ames city limits	Kitchen	Number of kitchens
Condition1	Proximity to main road or railroad	KitchenQual	Kitchen quality
Condition2	Proximity to main road or railroad (if a second is present)	TotRmsAbvGrd	Rooms above grade (not include bathrooms)
BldgType	Type of dwelling	Functional	Home functionality rating
HouseStyle	Style of dwelling	Fireplaces	Number of fireplaces
OverallQual	Overall material and finish quality	FireplaceQu	Fireplace quality
OverallCond	Overall condition rating	GarageType	Garage location
YearBuilt	Original construction date	GarageYrBlt	Year garage was built
YearRemodAdd	Remodel date	GarageFinish	Interior finish of the garage
RoofStyle	Type of roof	GarageCars	Size of garage in car capacity
RoofMatl	Roof material	GarageArea	Size of garage in square feet
Exterior1st	Exterior covering on house	GarageQual	Garage quality
Exterior2nd	Exterior covering on house (if more than one material)	GarageCond	Garage condition
MasVnrType	Masonry veneer type	PavedDrive	Paved driveway
MasVnrArea	Masonry veneer area in square feet	WoodDeckSF	Wood deck area in square feet
ExterQual	Exterior material quality	OpenPorchSF	Open porch area in square feet
ExterCond	Present condition of the material on the exterior	EnclosedPorch	Enclosed porch area in square feet
Foundation	Type of foundation	3SsnPorch	Three season porch area in square feet
BsmtQual	Height of the basement	ScreenPorch	Screen porch area in square feet
BsmtCond	General condition of the basement	PoolArea	Pool area in square feet
BsmtExposure	Walkout or garden level basement walls	PoolQC	Pool quality
BsmtFinType1	Quality of basement finished area	Fence	Fence quality
BsmtFinSF1	Type 1 finished square feet	MiscFeature	Miscellaneous feature not covered in other categories
BsmtFinType2	Quality of second finished area (if present)	MiscVal	$Value of miscellaneous feature
BsmtFinSF2	Type 2 finished square feet	MoSold	Month Sold
BsmtUnfSF	Unfinished square feet of basement area	YrSold	Year Sold
TotalBsmtSF	Total square feet of basement area	SaleType	Type of sale
Heating	Type of heating	SaleCondition	Condition of sale

Características do *target*:

```
#Verificando algumas caracteristicas do nosso target
data_set['SalePrice'].describe()
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Figura 1 – Características do target. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Outra característica importante para uma melhor compreensão do conjunto de dados é saber como quais tipos de dados estamos lidando, sendo assim vamos separar as variáveis que contem somente dados numéricos ['int16', 'int32', 'int64', 'float16', 'float32', 'float64'].

Tabela 2 – Variáveis numéricas:

Variáveis numéricas			
Id	BsmtFinSF2	HalfBath	EnclosedPorch
MSSubClass	BsmtUnfSF	BedroomAbvGr	3SsnPorch
LotFrontage	TotalBsmtSF	KitchenAbvGr	ScreenPorch
LotArea	1stFlrSF	TotRmsAbvGrd	PoolArea
OverallQual	2ndFlrSF	Fireplaces	MiscVal
OverallCond	LowQualFinSF	GarageYrBlt	MoSold
YearBuilt	GrLivArea	GarageCars	YrSold
YearRemodAdd	BsmtFullBath	GarageArea	SalePrice
MasVnrArea	BsmtHalfBath	WoodDeckSF	
BsmtFinSF1	FullBath	OpenPorchSF	

Em seguida separamos as variáveis categóricas ['object'].
Tabela 3 – Variáveis categóricas:

Variáveis categóricas			
MSZoning	BldgType	BsmtCond	GarageType
Street	HouseStyle	BsmtExposure	GarageFinish
Alley	RoofStyle	BsmtFinType1	GarageQual
LotShape	RoofMatl	BsmtFinType2	GarageCond
LandContour	Exterior1st	Heating	PavedDrive
Utilities	Exterior2nd	HeatingQC	PoolQC
LotConfig	MasVnrType	CentralAir	Fence
LandSlope	ExterQual	Electrical	MiscFeature
Neighborhood	ExterCond	KitchenQual	SaleType
Condition1	Foundation	Functional	SaleCondition
Condition2	BsmtQual	FireplaceQu	

Verificando a ausência de dados.

Tabela 4 – Dados ausêntes:

Feature	Total de dados ausentes	Percentual de dados ausentes %
PoolQC	2909	99.6574%
MiscFeature	2814	96.4029%
Alley	2721	93.2169%
Fence	2348	80.4385%
SalePrice	1459	49.9829%
FireplaceQu	1420	48.6468%
LotFrontage	486	16.6495%
GarageCond	159	5.4471%
GarageYrBlt	159	5.4471%
GarageQual	159	5.4471%
GarageFinish	159	5.4471%
GarageType	157	5.3786%
BsmtCond	82	2.8092%
BsmtExposure	82	2.8092%
BsmtQual	81	2.7749%
BsmtFinType2	80	2.7407%
BsmtFinType1	79	2.7064%
MasVnrType	24	0.8222%
MasVnrArea	23	0.7879%
MSZoning	4	0.1370%
Utilities	2	0.0685%
Functional	2	0.0685%
BsmtFullBath	2	0.0685%
BsmtHalfBath	2	0.0685%
GarageCars	1	0.0343%
BsmtFinSF2	1	0.0343%
Exterior2nd	1	0.0343%
GarageArea	1	0.0343%
TotalBsmtSF	1	0.0343%
BsmtUnfSF	1	0.0343%
BsmtFinSF1	1	0.0343%
Exterior1st	1	0.0343%
KitchenQual	1	0.0343%
SaleType	1	0.0343%
Electrical	1	0.0343%

Exploração visual

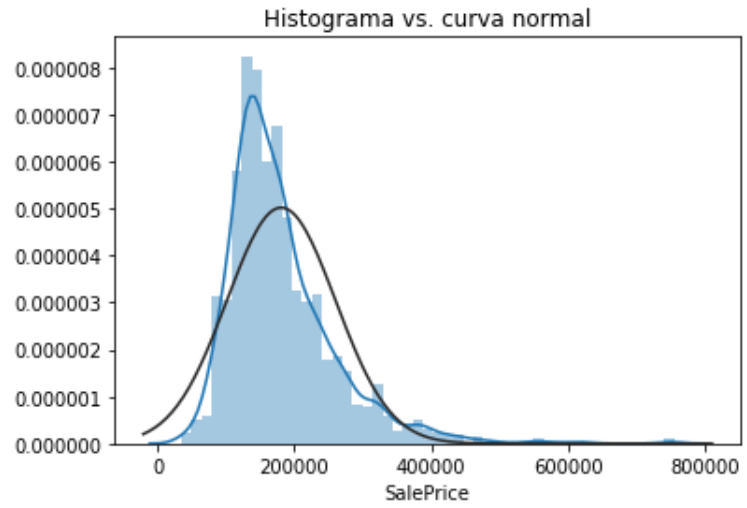


Figura 2 – Histograma SalePrice vs. curva normal. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Podemos observar uma calda longa para a direita, no pré-processamento podemos aplicar o logaritmo nos valores para conseguir uma maior normalidade dos dados.

Como se trata de um problema de regressão é importante observarmos a correlação entre as variáveis, uma das formas é utilizar uma matriz de correlação, veja abaixo.

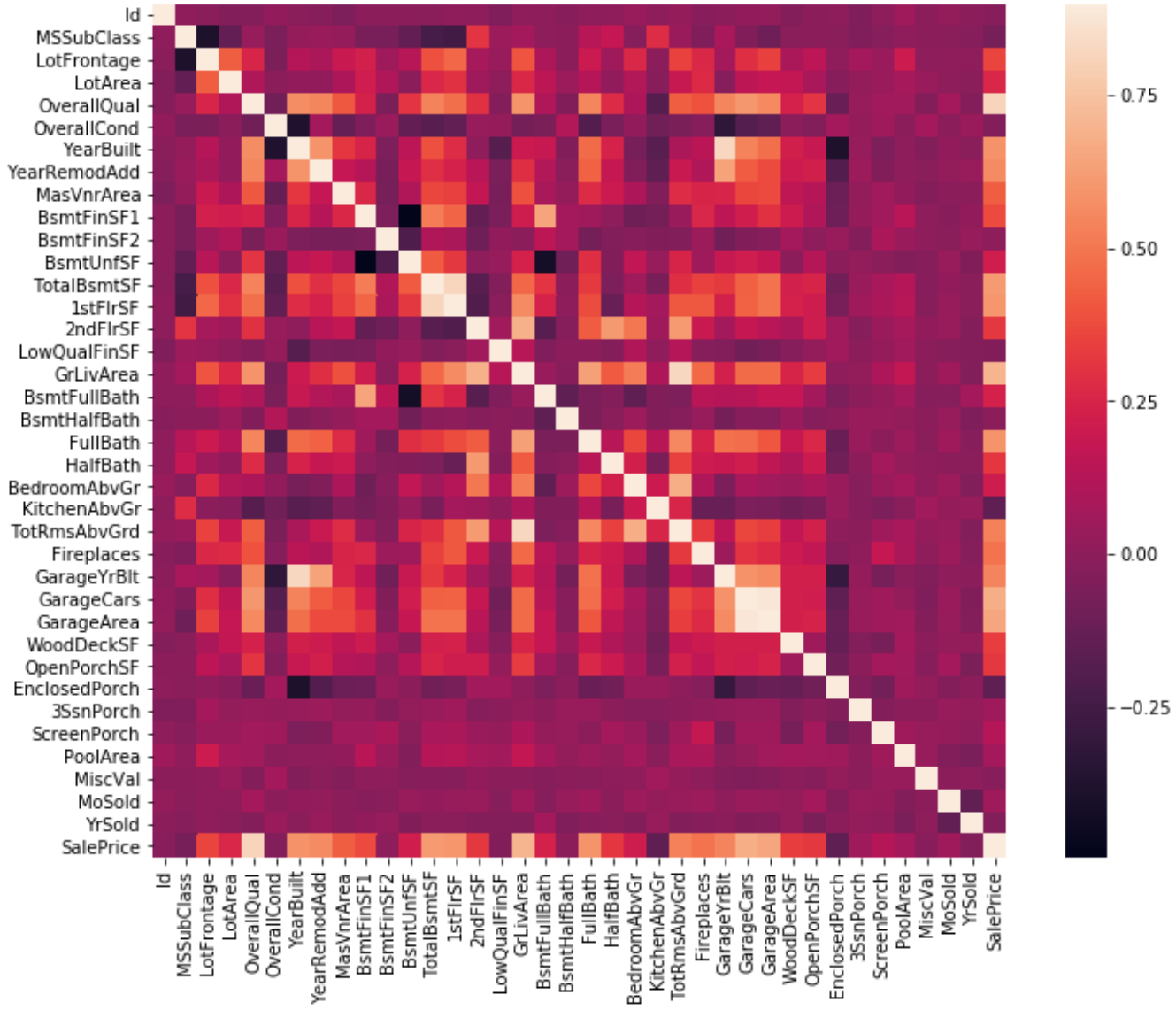


Figura 3 – Matriz de correlação. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Infelizmente não podemos mensurar a correlação com variáveis categóricas com esta matriz, mas podemos observar uma correlação mais forte do nosso *target* SalePrice em cores mais claras na matriz como por exemplo OverallQual, GrLivArea e Garage Cars. As *features* com maior correlação com o *target* tendem a ter um maior poder explicativo da sua variação, observe os gráficos a seguir:

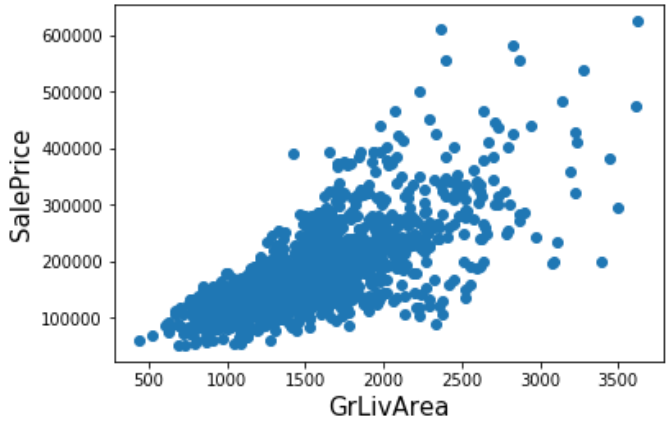


Figura 4 – SalePrice vs GrLivArea. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Analisando o gráfico podemos observar que a medida que os valores em SalePrice aumentam os valores de GrLivArea também aumentam. Note o mesmo comportamento com a variável OverallQual.

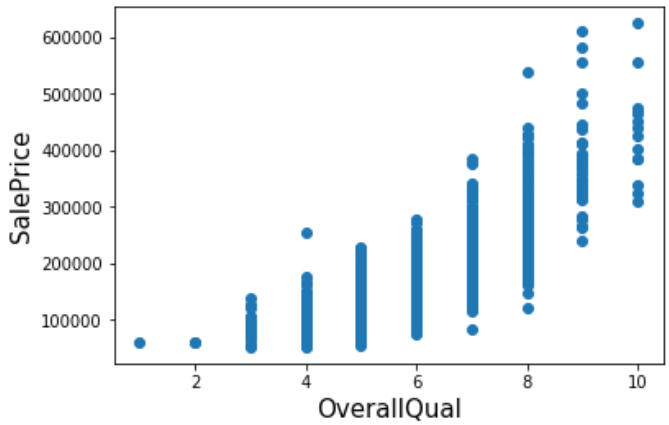


Figura 5 – SalePrice vs OverallQual. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Agora em para ver a diferença podemos observar uma variável com baixa correlação, EnclosedPorch. Podemos ver que não é possível notar uma tendência entre os SalePrice e EnclosedPorch.

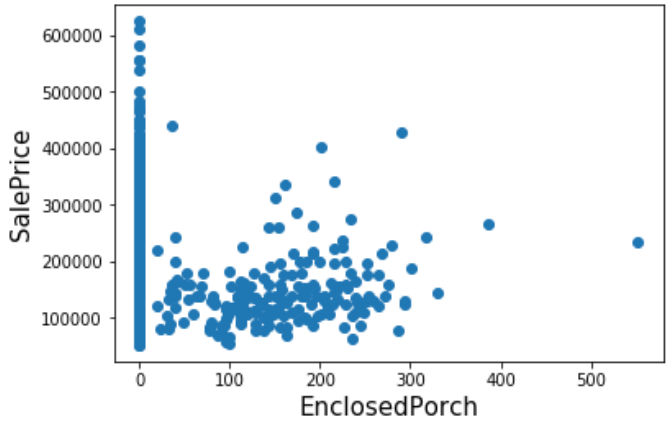


Figura 6 – SalePrice vs EnclosedPorch. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Algoritmos e técnicas a serem utilizadas

Para resolver o problema de determinação do SalePrice vamos utilizar técnicas de aprendizado de máquina.

Por definição aprendizado de máquina é:

"Um programa de computador é dito para aprender com a experiência E com a relação a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho T, medida pelo P, melhora com a experiência E" (Tom Mitchell, 1997)

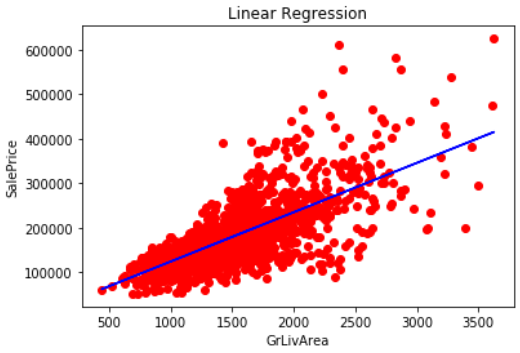
No nosso caso temos:

- Experiência E: Conjunto de dados do Kaggle.
- Tarefa T: Determinar o valor de venda o imóvel.
- Desempenho P: RMSE ou R^2

Outra característica importante dos algoritmos que vamos testar é que eles são considerados supervisionados, isto quer dizer que, as nossas amostrar já tem o *label* do nosso *target* definido que é o próprio SalePrice, mas nosso algoritmo uma vez treinado deverá ser capaz de determinar o SalePrice de novas amostras.

Neste trabalho trabalharemos com alguns algoritmos de regressão da biblioteca Scikit-Learn.

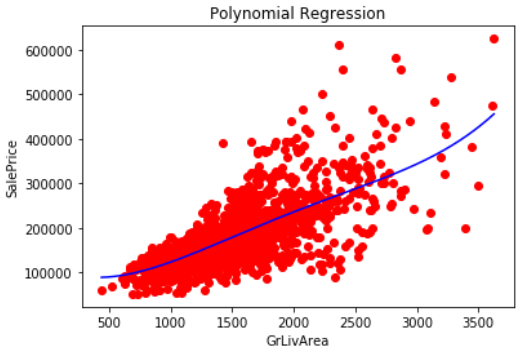
- LinearRegression:



Para exemplificar este algoritmo, a linha azul no gráfico abaixo foi definida pelo algoritmo Linear Regression, ela representa a linha reta que melhor se ajusta aos pontos que tem suas coordenadas definidas por SalePrice e GrLivArea.

Figura 7 – Exemplo de Linear Regression. Fonte: Final_Project_Udacity_Nanodregree.ipynb

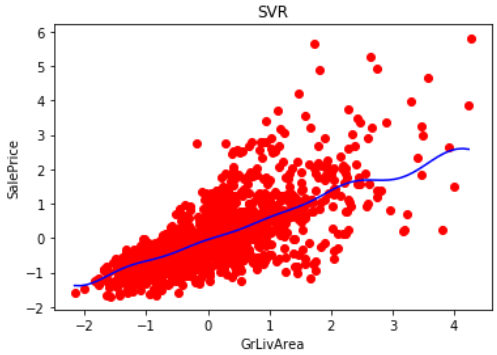
- LinearRegression com PolynomialFeatures ou Polynomial Regression:



Com a aplicação do PolynomialFeatures nossa regressão consegue ajustar nossa linha em uma função polinomial, assim ela pode se ajustar em formas mais complexas.

Figura 8 – Exemplo de Polynomial Regression. Fonte: Final_Project_Udacity_Nanodregree.ipynb

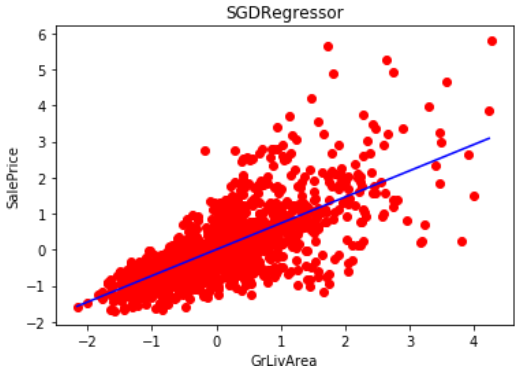
- SVR (Support Vector Regression):



O Support Vector Regression também consegue "desenhar" linhas bem complexas. Porem este algoritmo tem uma desvantagem, para ele funcionar com uma performance aceitável é totalmente necessário que os dados estejam na mesma estala, portanto é imprescindível a aplicação de um *scaller.*

Figura 9 – Exemplo SVR. Fonte: Final_Project_Udacity_Nanodregree.ipynb

- SGDRegressor:



Este algoritmo é parecido com o LinearRegression, porém, a maior diferença é que ele apreende com uma amostra por vez enquanto o Linear Regression recebe como *input o batch* (pacote do conjunto de dados de uma só vez). Está é uma vangatem em relação aos demais pois ele pode ser usado para *online learning*, melhorando o algoritmo a medida que novas amostras vão se tornando disponíveis. Igual ao SVR, o *input* também deve ser escalado.

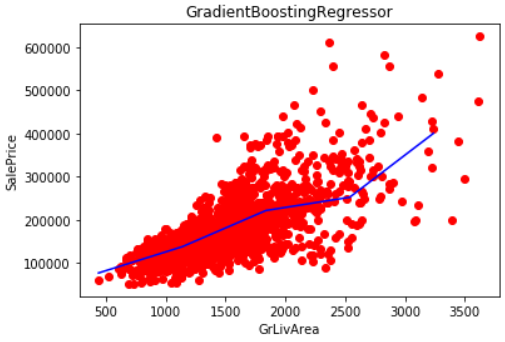
Figura 10 – Exemplo SDGRegressor. Fonte: Final_Project_Udacity_Nanodregree.ipynb

- DecisionTreeRegressor:

Diferente dos algoritmos vistos acima, ele não é considerado um algoritmo "linear" que tenta ajustar os parâmetros a uma linha, ele faz parte dos *Tree Based Models,* nestes o valor previsto do *target* deve ser determinado por simples tomadas de decisão inferidas pelas variáveis (http://scikit-learn.org/stable/modules/tree.html)

Figura 11 – Exemplo DecisionTreeRegressor. Fonte: Final_Project_Udacity_Nanodregree.ipynb

- GradientBoostingRegressor:



Este é outro algotirmo *Tree Based Model,* ele consiste na combinação de diversas arvores de decisão através da técnica de Boosting.

Figura 12 – Exemplo GradientBoostingRegressor Fonte: Final_Project_Udacity_Nanodregree.ipynb

Durante o trabalho aplicaremos o GridSearch para identificar os hiper-parâmetros que otimizam o resultado dos algoritmos utilizados.

Também aplicaremos uma técnica de ensemble nos dois algoritmos com os melhores resultados. O método a ser utilizado é chamado de *Weighted Averaging,* este consiste em atribuir um peso a cada um dos resultados obtidos e realizar uma média ponderada, desta forma criando uma combinação do resultado dos dois algoritmos.

Observação: Para a aplicação do PolynomialFeatures será necessário aplicar uma redução de dimensão no conjunto de dados, por isso utilizaremos o StandardScaler e na sequência o algoritmo/técnica PCA. O motivo pelo qual a aplicação do PCA é necessária é por que o PolynomialFeatures cria novas *features* a partir das combinações das *features* de *input* e o número de *features* criadas é tão grande que torna o uso da memória RAM inviável.

Modelo de referência (benchmark)

Utilizaremos dois modelos de referência, o primeiro visa validar a proposta do trabalho, o segundo verificar se o melhor modelo encontrado apresenta uma performance satisfatória.

Modelo 1 – Regressão Linear Simples:

Para validar a proposta do projeto iremos comparar o modelo gerado a partir da técnica de regressão linear simples levando em consideração as métricas de avaliação Root-MeanSquared-Error e R^2 com os outros modelos gerados a partir das outras técnicas de regressão utilizadas neste trabalho.

Modelo 2 - Kaggle leaderboard:

#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲2302	DSXL			0.06628	3	5d
2	▼1	Igor S			0.06946	8	8d
3	▼1	Zheng Pan			0.08021	3	1mo
4	▼1	Javale			0.08397	1	2mo
5	▼1	mohammed khamis			0.10567	1	12d

Figura 4 - Kaggle Leaderboard. Fonte: https://www.kaggle.com/c/house-prices-advancedregression-techniques/leaderboard

O melhor modelo apresentado neste trabalho será submetido na competição, assim o modelo poderá ser comparado com o de outros competidores através do score fornecido pelo Kaggle.

III. Metodologia

Pré-processamento dos dados

Observamos na exploração dos dados que existem diversos dados com valores ausentes nas amostras por conta disso iremos preencher esses valores faltantes seguindo algumas estratégias:

Remover a *feature*:

```
total_data = total_data.drop(["MiscFeature"], axis=1)
total_data = total_data.drop("Utilities", axis=1)
```

Preencher com o valor correto para o valor “nulo” para padronizar o conjunto de dados:

```
total_data["PoolQC"].fillna("NA", inplace=True)
total_data["Alley"].fillna("NA", inplace=True)
total_data["Fence"].fillna("NA", inplace=True)
total_data["FireplaceQu"].fillna("NA", inplace=True)
total_data["GarageCond"].fillna("NA", inplace=True)
total_data["GarageFinish"].fillna("NA", inplace=True)
total_data["GarageQual"].fillna("NA", inplace=True)
total_data["GarageType"].fillna("NA", inplace=True)
total_data["BsmtExposure"].fillna("NA", inplace=True)
total_data["BsmtCond"].fillna("NA", inplace=True)
total_data["BsmtQual"].fillna("NA", inplace=True)
total_data["BsmtFinType2"].fillna("NA", inplace=True)
total_data["BsmtFinType1"].fillna("NA", inplace=True)
total_data["MasVnrType"].fillna("NA", inplace=True)
total_data["BsmtFullBath"].fillna(0, inplace=True)
total_data["BsmtHalfBath"].fillna(0, inplace=True)
total_data["Functional"].fillna("Typ", inplace=True)
total_data["GarageArea"].fillna(0, inplace=True)
total_data["BsmtFinSF2"].fillna(0, inplace=True)
total_data["BsmtUnfSF"].fillna(0, inplace=True)
total_data["TotalBsmtSF"].fillna(0, inplace=True)
total_data["BsmtFinSF1"].fillna(0, inplace=True)
total_data["MasVnrArea"].fillna(0, inplace=True)
total_data.at[2576,'GarageCars']=0
```

Preencher com o valor que mais se repete (moda):

```
total_data['Exterior1st'].fillna("VinylSd", inplace=True)
total_data['Exterior2nd'].fillna("VinylSd", inplace=True)
total_data["KitchenQual"].fillna("TA", inplace=True)
total_data["SaleType"].fillna("WD", inplace=True)
total_data["MSZoning"].fillna("RL", inplace=True)
total_data["Electrical"].fillna("SBrkr", inplace=True)
```

Os únicos valores ausentes que foram preenchidos de outra forma foram os da *feature* LotFrontage, neste caso os valores ausentes foram determinados a partir de uma regressão linear das *features* 'LotArea' e 'LotConfig'

```
total_data["LotFrontage"]=predictLotFrontage()
```

Também iremos remover as *features* com índices

```
total_data = total_data.drop("Id", axis=1)
total_data = total_data.drop("index", axis=1)
```

Aplicaremos *log transformation* no nossos target.

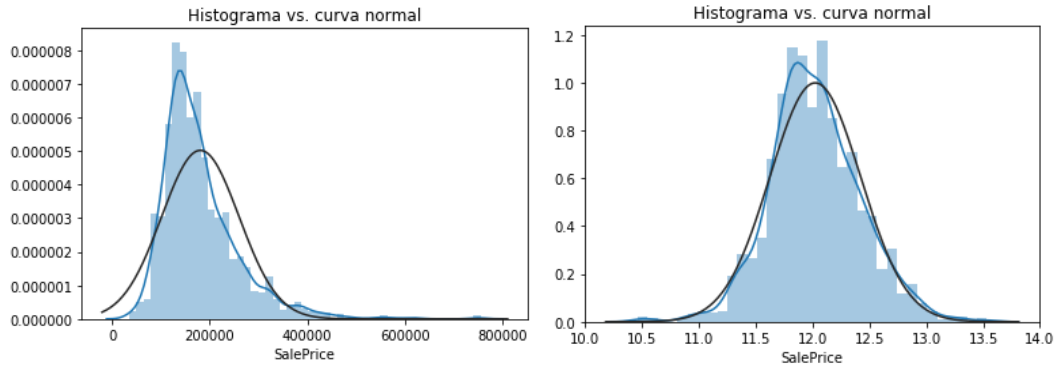


Figura 5 - Histogramas SalePrice vs. curva normal. Fonte: Final_Project_Udacity_Nanodregree.ipynb

Aplicando log temos um novo histograma, podemos observar como a linha de distribuição dos dados agora se aproxima da curva normal (figura da direita).

Iremos transformar algumas variáveis numéricas ([YrSold', 'MoSold' e 'MSSubClass') em categóricas com a finalidade de obter uma maior sensibilidade na variação das mesmas

```
total_data['YrSold']=total_data['YrSold'].astype(str)
```

```
total_data['MoSold']=total_data['MoSold'].astype(str)
```

```
total_data['MSSubClass']=total_data['MSSubClass'].astype(str)
```

Por fim aplicamos o método do pandas get_dummies para transformar as variáveis categóricas.

Para os algoritmos SVR (Support Vector Regression) e SGDRegressor será aplicamos o StandardScaler pois uma característica nestes algoritmos é que os dados de entrada devem estar em uma ordem de grandeza próxima.

Já para a aplicação do PolynomialFeatures será necessário aplicar uma redução de dimensão no conjunto de dados, por isso utilizaremos o StandardScaler e na sequência o algoritmo/técnica PCA. O motivo pelo qual a aplicação do PCA é necessária é por que o PolynomialFeatures cria novas *features* a partir das combinações das *features* de *input* e o número de *features* criadas é tão grande que torna o uso da memória RAM inviável.

Implementação dos algoritmos

Implementamos os seguintes algoritmos da biblioteca do Scikit-Learn no conjunto de dados:

- LinearRegression
- LinearRegression com PolynomialFeatures
- SVR (Support Vector Regression)
- SGDRegressor
- DecisionTreeRegressor
- GradientBoostingRegressor

A estratégia adotada para mensurar a performance foi o K-fold com (k=3) para as duas métricas:

```
def RMSE_score(model, X, y):
    kfold = KFold(3, shuffle=True, random_state=42).get_n_splits(X.values)
    RMSE = np.sqrt(-cross_val_score(model, X.values, y,
scoring="neg_mean_squared_error", cv = kfold))
    return(RMSE)

def R2_score(model, X, y):
    kfold = KFold(3, shuffle=True, random_state=42).get_n_splits(X.values)
    R2 = (cross_val_score(model, X.values, y, scoring="r2", cv = kfold))
    return(R2)
```

Refinamento

Para o *tunning* dos algoritmos a estratégia utilizada foi utilizar o GridSearch dos principais parâmetros de cada algoritmo (sklearn.model_selection.GridSearchCV).

Assim uma vez que os melhores parâmetros encontrados foram estabelecidos eles foram aplicados na implementação dos algoritmos.

IV. Resultados

Avaliação dos modelos e validação

Tabela 5 – Score RMSE e R^2 por modelo:

	RMSE	Desvio Padrão	R^2	Desvio Padrão
LinearRegression	0.1353	0.0087	0.8775	0.0053
PolynomialRegression	0.1364	0.0115	0.8753	0.0082
SVR	0.1171	0.0088	0.9080	0.0067
SGDRegressor	0.1127	0.0098	0.9147	0.0083
DecisionTreeRegressor	0.1759	0.0049	0.7922	0.0168
GradientBoostingRegressor	0.1148	0.0055	0.9112	0.0118

Analisando os resultados dos modelos obtidos pelo CrossValidation, observamos que:

- em MSE o melhor modelo é o GradientBoostingRegressor com o RMSE score de 0.1148 modelo:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=2, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=3,
             min_samples_split=7, min_weight_fraction_leaf=0.0,
             n_estimators=300, presort='auto', random_state=42,
             subsample=0.7, verbose=0, warm_start=False)
```

- em R^2 o melhor modelo é o SDGRegressor com R2 score de 0.9147 modelo:

```
SGDRegressor(alpha=0.3, average=False, epsilon=0.1, eta0=0.01,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', max_iter=1000, n_iter=None, penalty='l2',
       power_t=0.25, random_state=42, shuffle=True, tol=None, verbose=0,
       warm_start=False)
```

Sendo assim vamos aplicar o GradientBoostingRegressor e o SDGRegressor no test_set do Kaggle, e seguindo a proposta vamos aplicar o *ensemble method Weighted Averaging* nesses modelos, os demais não serão combinados.

```
#Realizando previsões com o GradientBoostingRegressor
regressorGradientBoostingRegressor.fit(X, y)
pred_GradientBoostingRegressor =
np.exp(regressorGradientBoostingRegressor.predict(X_preprocessed_submission_set))

#Realizando previsões com o SGDRegressor
regressorSGDRegressor.fit(X_scaled,y)
pred_SGDRegressor =
np.exp(regressorSGDRegressor.predict(X_preprocessed_submission_set_scaled))

# Weighted Average
esemble_pred = (0.30*pred_SGDRegressor)+(0.70*pred_GradientBoostingRegressor)
```

Submetendo as predições do GradientBoostingRegressor, SGDRegressor e Weighted Average temos os seguintes resultados. (Quando menor o score melhor)

Submission and Description	Public Score
predicts_esemble.csv 2 minutes ago by Marcos Falcirolli predicts_esemble.csv	0.12280
predicts_SGDRegressor.csv 4 minutes ago by Marcos Falcirolli predicts_SGDRegressor.csv	0.12930
predicts_GradientBoostingRegressor.csv 7 minutes ago by Marcos Falcirolli predicts_GradientBoostingRegressor.csv	0.12585

Figura 6 - Kaggle submissions.

Justificativa

Modelo 1 - Regressão Linear Simples

Os scores gerados pela Regressão linear simples foram RMSE 0.1536 e R^2 0.851, portanto observando a tabela apresentada nos resultados todos os modelos propostos gerados tiveram desempenho superior ao deste benchmark com exceção do DecisionTreeRegressor que ficou abaixo tanto em RMSE e R^2.

Modelo 2 - Kaggle leaderboard

O *Esemble Weighted Averaging* gerou um Kaggle Score de 0.12280 que é melhor do que os scores gerados pelos modelos GradientBoostingRegressor e SGDRegressor individualmente.

V. Conclusão

Foi possível aplicar todos os algoritmos propostos neste trabalho para a previsão do valor de venda de um imóvel. São eles:

- LinearRegression
- LinearRegression com PolynomialFeatures
- SVR (Support Vector Regression)
- SGDRegressor
- DecisionTreeRegressor
- GradientBoostingRegressor

No entanto dentre os modelos observamos que alguns performaram mais do que outros, em destaque o GradientBoostingRegressor no critério RMSE e SGDRegressor no R^2.

A combinação dos modelos através de média ponderada também se mostrou ser uma técnica que ajuda a melhorar a previsão dos modelos.

Veja os exemplos abaixo, quando aplicamos a média ponderada o desvio é atenuado:

Exemplo 1		
Amostra id: 538	Resultado Previsto	Módulo do Desvio
SDG	204906.01	8343.99
GradientBoost	217131.72	3881.72
Weighted Avegaring	213464.01	**214.01**
Valor Verdadeiro	213250.00	

Exemplo 2		
Amostra id: 1447	Resultado Previsto	Módulo do Desvio
SDG	200203.24	9796.76
GradientBoost	222741.19	12741.19
Weighted Avegaring	215979.81	**5979.81**
Valor Verdadeiro	210000.00	

Podemos concluir então que para realizar avaliações de imóveis também é possível a utilização de outras técnicas de regressão além da regressão linear como sugerido pela NBR 14653, norma que regulamenta avaliação de bens no Brasil.

Reflexões

Foi interessante notar como os modelos se comportam com diferenças na aplicação do data set, por exemplo os modelos SVR e SGDRegressor simplesmente não funcionam se o data set não estiver escalado.

Outro ponto interessante foi observar que mesmo com a aplicação de uma técnica *esemble* tão simples quanto o *Weighted Averaging* pode gerar resultados ligeiramente superiores.

Mesmo com o modelo combinado com a técnica *Weighted Averaging* o resultado obtido rendeu uma posição entre os 29% melhores nesta competição (Kaggle "House Prices: Advanced Regression Techniques") o que é abaixo das expectativas para este projeto.

Melhorias futuras

Acredito que este trabalho pode ser melhorado, após pesquisar bastante no fórum do Kaggle sobre esta competição vi que existem outros algoritmos de regressão que desempenham muito bem e são amplamente utilizados neste tipo de competição como LASSO Regression, Elastic Net Regression, Kernel Ridge Regression, LightGBM e XGBoost.

Realizando um estudo também no curso "How to Win a Data Science Competition" do Coursera notei que para este tipo de competição é muito utilizado técnicas mais avançadas de *esemble* que ajudam a melhorar o score.

Então para um trabalho futuro focaria na aplicação dos algoritmos citados acima e em ensembles mais avançados como *Stacking* e *Boosting*.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2011, NBR 14653-1: Avaliação de bens. São Paulo. 2001

MITCHELL THOMAS. Machine Learning: 1997.

https://pt.coursera.org/learn/competitive-data-science

http://scikit-learn.org/stable/modules/sgd.html#regression

http://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting

http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python