Proyecto 1

November 5, 2020

1 Proyecto 1 - Edgar Gerardo Alarcón González

1.1 Predicción de precio de ventas por regresiones

1.1.1 Carga de librerías

[1]:
```
from scipy import stats
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import math
import seaborn as sns
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
```

1.1.2 A. Carga de datos

Entrenamiento

[2]:
```
train = pd.read_csv(r"C:
 ↪\Users\alarc\OneDrive\Documents\Programming\Python\Jupyter\Inteligencia␣
 ↪Artificial\Proyecto 1\house-prices-advanced-regression-techniques/train.csv",
                    index_col="Id")
```

Prueba

[3]:
```
test = pd.read_csv(r"C:
 ↪\Users\alarc\OneDrive\Documents\Programming\Python\Jupyter\Inteligencia␣
 ↪Artificial\Proyecto 1\house-prices-advanced-regression-techniques/test.csv",
                    index_col="Id")
```

Sample Submission

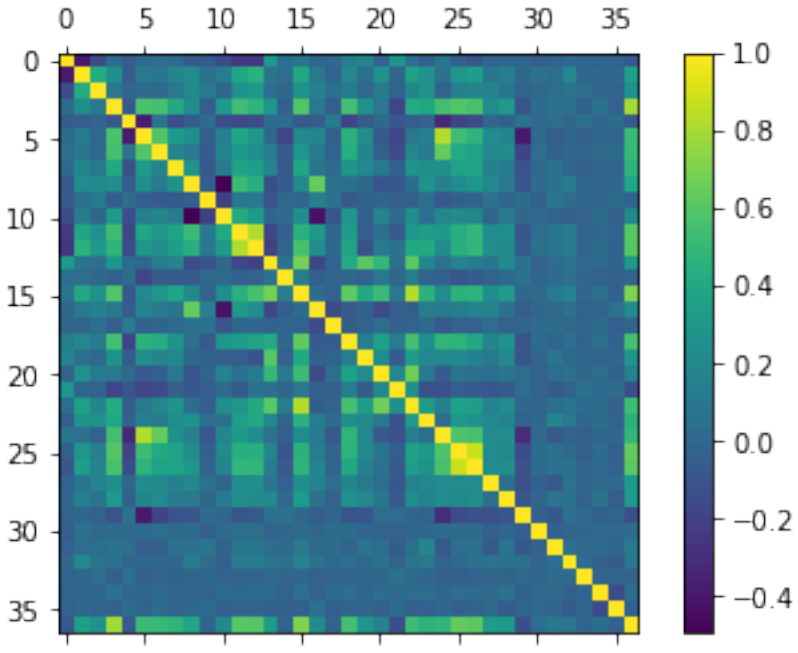
[4]:
```
sample = pd.read_csv(r"C:
 ↪\Users\alarc\OneDrive\Documents\Programming\Python\Jupyter\Inteligencia␣
 ↪Artificial\Proyecto 1\house-prices-advanced-regression-techniques/
 ↪sample_submission.csv",
```

```
                    index_col="Id")
```

1.1.3 B. Correlación de los datos

Primeramente vamos a hacer énfasis que nosotros trabajaremos con variables explicativas que sean totalmente numéricas, existe una gran cantidad de variables categóricas que presenta la información, sin embargo, con la finalidad de mantener un tanto de parsimonía en el modelo, simplemente ubicaremos a las variables que sean numéricas. Vamos a ver la correlación que tiene toda la tabla de entrenamiento, que es con la que alimentaremos el modelo.

[5]:
```
# Gráfico con leyendas
figure = plt.figure()
axes = figure.add_subplot(111)
# using the matshow() function
caxes = axes.matshow(train.corr())
figure.colorbar(caxes)
# Hacemos el gráfico
plt.show()
```



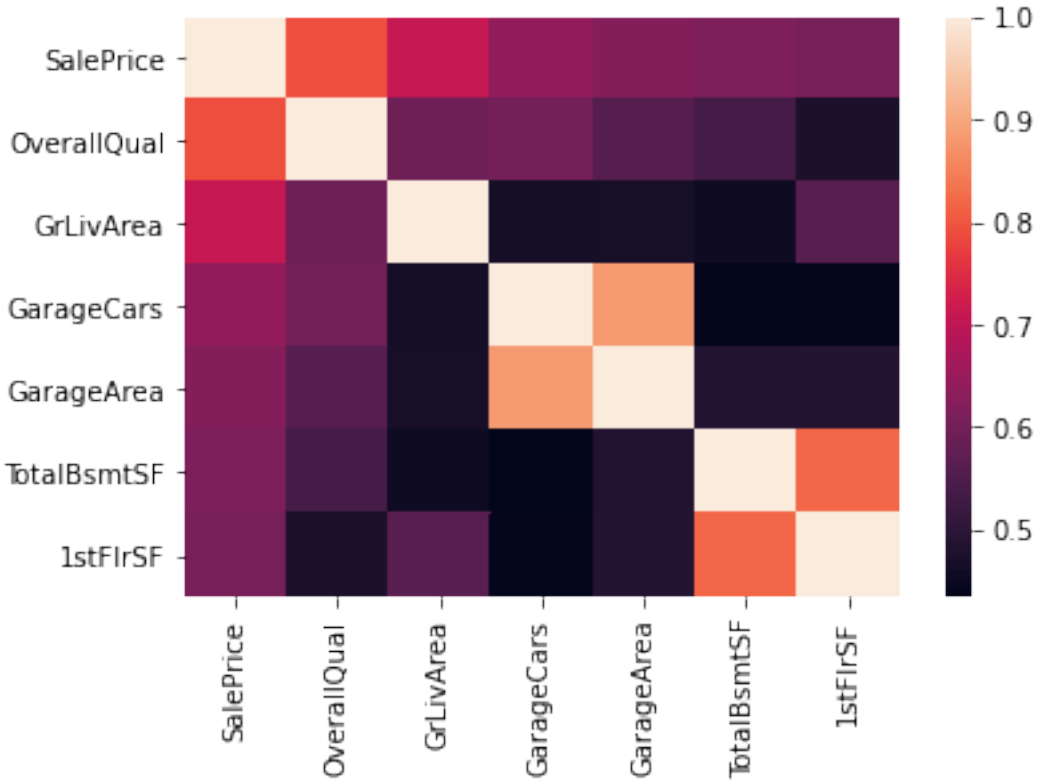
Notemos que en general la mayoría de las variables que estamos trabajando no guardan multicolinealidad. Lo que haremos a continuación es buscar las variables explicativas que guarden una correlación mayor con la respuesta, `"PriceSales"`. Para esto, filtraremos las variables explicativas tales que guarden una correlación mayor que 0.6 con la respuesta.

[6]:
```
MC=train.corr()
SaleCorr=abs(MC.loc[:,"SalePrice"])
aux=SaleCorr.sort_values(ascending=False)
Explicativas=aux.index[aux>0.6]
train_selec = train[Explicativas]
```

Veamos un gráfico de la correlación con las variables que hemos extraido.

[7]:
```
sns.heatmap(train_selec.corr())
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4bd5c0700>
```



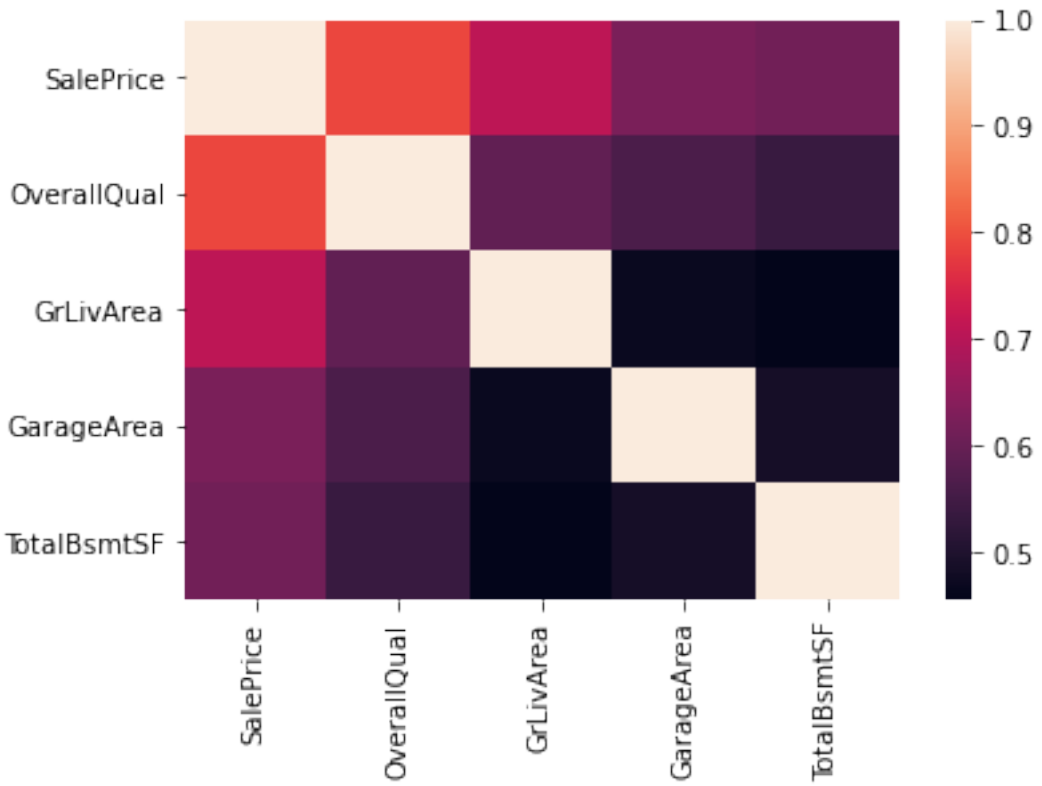
Vemos que hay mucha multicolinealidad entre algunas de las parejas de variables `"GarageCars"-"GarageArea"` y `"TotalBsmtSF"-"1stFlrSF"`. Para evitar la multicolinealidad vamos a extraer estas variables.

[8]:
```
train_selec=train_selec.drop(columns=['GarageCars', '1stFlrSF'])
```

Y veamos entonces el gráfico resultante con las variables seleccionadas.

[9]:
```
sns.heatmap(train_selec.corr())
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4bd681e50>
```



Vamos a guardar en los objetos `X` y `y` las variables explicativas y la respuesta, respectivamente.

[10]:
```
X = train_selec.drop(columns=['SalePrice'])
y = train_selec['SalePrice']
```

1.1.4 C. Creación de Modelos

Aquí vamos a hacer la creación de los modelos, los cuales vamos a alimentar con los objetos anteriores. Adicionalmente, vamos a mostrar, los coeficientes del modelo y estadísticas de las variables respuesta obtenidas.

C.1. Modelo Lineal

[11]:
```
linear_model = LinearRegression(normalize=False,fit_intercept=True)
linear_model.fit(X,y)
y_linear=linear_model.predict(X)
aux=stats.describe(y_linear)
df = pd.DataFrame([aux], columns=aux._fields)
df.round(3)
```

[11]:

	nobs	minmax	mean	variance	\
0	1460	(-59644.12391656727, 673805.9903350528)	180921.196	4.779432e+09	

```
   skewness  kurtosis
0      0.75     2.277
```

[12]:
```
aux=linear_model.coef_
df = pd.DataFrame([aux],columns=X.columns)
print(df.round(3))
```

```
   OverallQual  GrLivArea  GarageArea  TotalBsmtSF
0    24954.711     45.628      58.246       30.126
```

C.2. Modelo Lasso

[13]:
```
lasso_model = Lasso(alpha=0.5,normalize=True, max_iter=1e6)
lasso_model.fit(X,y)
y_lasso=lasso_model.predict(X)
aux=stats.describe(y_lasso)
df = pd.DataFrame([aux], columns=aux._fields)
df.round(3)
```

[13]:
```
   nobs                                  minmax        mean      variance  \
0  1460  (-59572.62003024666, 673583.0638959827)  180921.196  4.776324e+09

   skewness  kurtosis
0      0.75     2.276
```

[14]:
```
aux=lasso_model.coef_
df = pd.DataFrame([aux],columns=X.columns)
df.round(3)
```

[14]:
```
   OverallQual  GrLivArea  GarageArea  TotalBsmtSF
0    24952.937     45.611      58.202       30.105
```

C.3. Modelo Ridge

[15]:
```
ridge_model = Ridge(alpha=0.5)
ridge_model.fit(X,y)
y_ridge=ridge_model.predict(X)
aux=stats.describe(y_ridge)
df = pd.DataFrame([aux], columns=aux._fields)
df.round(3)
```

[15]:
```
   nobs                                  minmax        mean      variance  \
0  1460  (-59623.34441928893, 673854.5705534369)  180921.196  4.779005e+09

   skewness  kurtosis
0     0.751     2.278
```

[16]:
```
aux=ridge_model.coef_
df = pd.DataFrame([aux],columns=X.columns)
df.round(3)
```

[16]:

	OverallQual	GrLivArea	GarageArea	TotalBsmtSF
0	24945.847	45.636	58.262	30.133

De donde vemos que los resultados son bastante similares entre todos los modelos.

1.1.5 D. Preparando la información

En esta parte, vamos a procesar la tabla `test` para que que podamos utilizar las variables explicativas que estamos manejando y también quitando los datos que no sean aceptables para realizar un modelo como los que hicimos arriba.

[17]:
```
# Filtramos las covariables que vamos a utilizar
test_selec = test[X.columns]
# Procedemos a quitar algunos datos faltantes.
is_NaN = test_selec.isnull()
row_has_NaN = is_NaN.any(axis=1)
# Nos quedamos con las que NO son NA
test_selec = test_selec[~row_has_NaN]
Xs = test_selec[~row_has_NaN]
ys = sample[~row_has_NaN]
ysd=ys.describe()
```

```
<ipython-input-17-4f8374e6896f>:8: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  Xs = test_selec[~row_has_NaN]
```

1.1.6 E. Implementación de los modelos

En esta sección lo que haremos será poner a prueba los modelos propuestos.

En las secciones E.X.A vamos a estar probando los modelos con el conjunto de entrenamiento, esto significa que vamos a pronosticar las respuestas con base en el modelo lineal apoyado por las variables explicativas y las vamos a comparar con las respuestas originales en un gráfico, donde esperaríamos ver la identidad lo más similarmente posible, esto estaría indicando un pronóstico adecuado de la respuesta. Recordemos que esta información viene del archivo `train.csv`

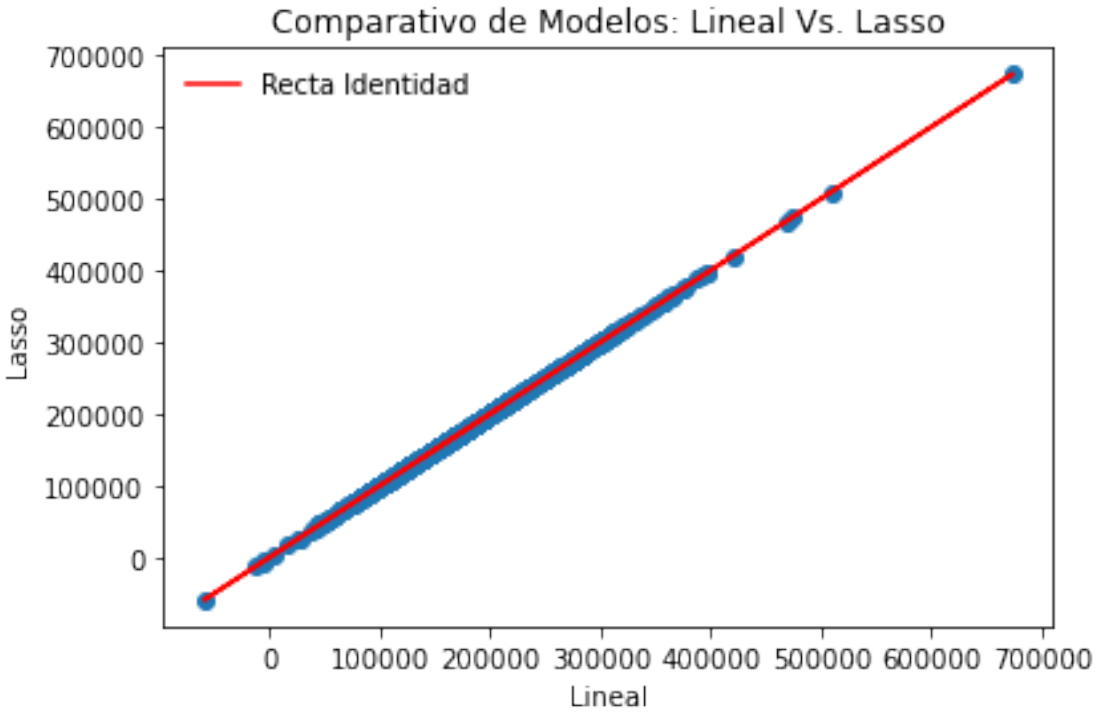
Análogamente en las secciones E.X.B vamos a pronosticar las variables respuesta apoyados del archivo `test.csv` donde vienen las variables explicativas y posteriormente vamos a comparar contra sus respuestas que se encuentran en el archivo

Ciertamente los modelos guardan parsimonía al tener pocos parámetros para estimar además de ser explicados únicamente en términos de variables que son totalmente numéricas. Sin embargo, aunque en el conjunto de entrenamiento parece funcionar de forma relativamente adecuada, parece que no se llega a modelar bien el comportamiento subyacente de los datos con este modelo, ya que presenta grandes diferencias a comparación de los precios que se encuentran en

la tabla `sample_submission`. Además, los modelos parecen no diferir mucho entre sí, de hecho cambian realmente muy poco, lo cual podemos ver en los siguientes gráficos.

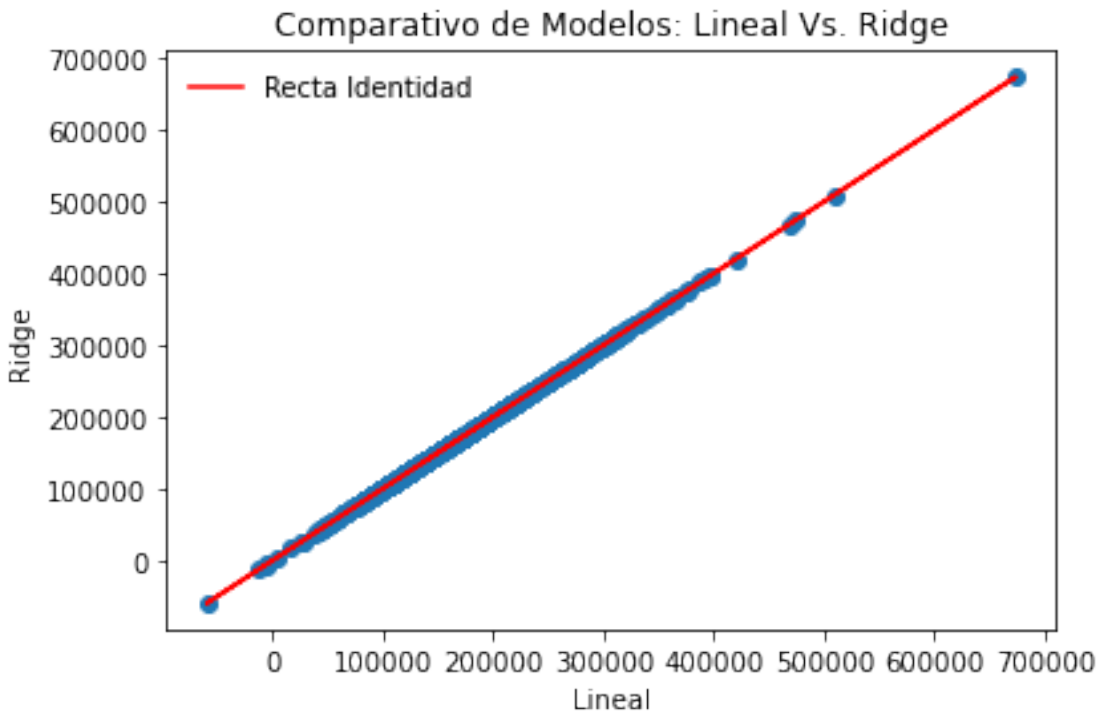
F.1. Lineal Vs Lasso

[30]:
```
# Graficamos
plt.scatter(y_linear,y_lasso)
plt.plot(y_linear, y_linear, '-',color="red",label="Recta Identidad")
plt.title('Comparativo de Modelos: Lineal Vs. Lasso')
plt.xlabel('Lineal')
plt.ylabel('Lasso')
plt.legend(loc='best', frameon=False)
plt.show()
```



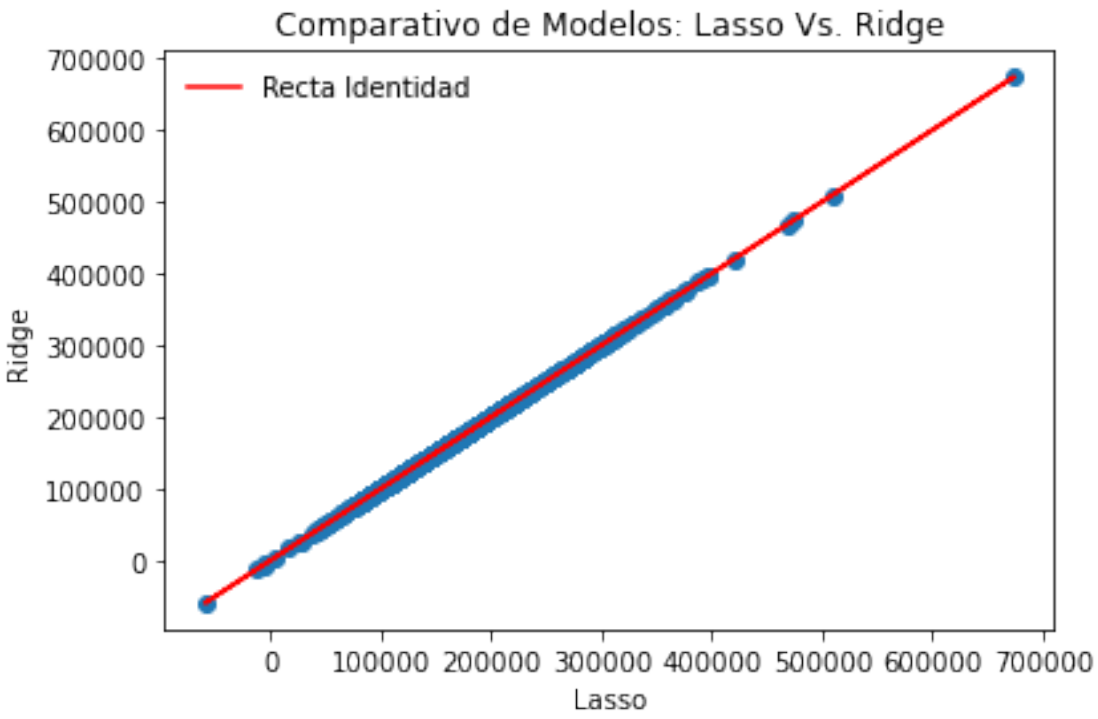
F.1. Lineal Vs Ridge

[31]:
```
plt.scatter(y_linear,y_ridge)
plt.plot(y_linear, y_linear, '-',color="red",label="Recta Identidad")
plt.title('Comparativo de Modelos: Lineal Vs. Ridge')
plt.xlabel('Lineal')
plt.ylabel('Ridge')
plt.legend(loc='best', frameon=False)
plt.show()
```



F.1. Lasso Vs Ridge

[32]:
```
plt.scatter(y_lasso,y_ridge)
plt.plot(y_lasso, y_lasso, '-',color="red",label="Recta Identidad")
plt.title('Comparativo de Modelos: Lasso Vs. Ridge')
plt.xlabel('Lasso')
plt.ylabel('Ridge')
plt.legend(loc='best', frameon=False)
plt.show()
```



En conclusión, para obtener resultados más precisos, lo ideal sería experimentar con más variables respuesta y de esta manera buscar modelos que ajusten mejor desde el conjunto de entrenamiento.