Inteligencia de Negocio

Memoria Práctica 3

Competición en Kaggle.

Nuria Rodríguez Barroso
Universidad de Granada
rbnuria6@gmail.com

Índice

1. Representación de resultados obtenidos

En primer lugar, veamos una tabla donde y vemos la progresión obtenida, con la fecha y hora de subida, algunas medidas del error y la posición ocupada en la plataforma Kaggle en ese momento.

Versión	**Fecha y hora**	**Score**	**RSMLE**	**R2**	**Posición kaggle**
Versión base	26 diciembre 11:53	0.12593	0.093761	0.925365	1054
Versión 1	26 diciembre 12:21	0.12276	0.092116	0.933732	837
Versión 2	26 diciembre 12:52	0.12799	0.084268	0.944914	837
Versión 3	26 diciembre 13:25	0.12571	0.070269	0.966835	837
Versión 4	27 diciembre 19:56	0.12108	0.084329	0.947055	719
Versión 5	27 diciembre 20:03	0.13308	0.092162	0.875700	719
Versión 6	27 diciembre 21:29	0.12651	0.083975	0.948659	722
Versión 7	27 diciembre 21:48	0.12188	0.076953	0.960676	722
Versión 8	28 diciembre 10:42	0.12029	0.083910	0.946790	602
Versión 9	28 diciembre 11:23	0.12133	0.083744	0.947414	602
Versión 10	28 diciembre 11:48	0.12157	0.084471	0.945895	602
Versión 11	29 diciembre 10:02	0.12248	0.084374	0.945512	604
Versión 12	29 diciembre 11:51	0.12397	0.086304	0.942996	604
Versión 13	29 diciembre 11:58	0.12396	0.086323	0.941823	604
Versión 14	29 diciembre 12:36	0.12376	0.086527	0.942798	604
Versión 15	29 diciembre 12:48	0.12234	0.083742	0.947392	590
Versión 16	29 diciembre 19:29	0.11620	0.082165	0.950406	324
Versión 17	29 diciembre 19:41	0.11488	0.081803	0.950462	146
Versión 18	30 diciembre 16:44	0.11500	0.082334	0.948256	148
Versión 19	30 diciembre 17:00	0.11631	0.082869	0.946916	148
Versión 20	30 diciembre 17:12	0.11642	0.083116	0.948215	148
Versión 21	30 diciembre 17:50	0.11533	0.082636	0.949870	148
Versión 22	30 diciembre 18:10	0.11546	0.089738	0.936033	148
Versión 23	30 diciembre 18:27	0.11479	0.077184	0.958181	136
Versión 24	30 diciembre 18:31	0.11458	0.088620	0.939575	118
Versión 25	30 diciembre 18:38	0.11452	0.076278	0.959029	112
Versión 26	30 diciembre 18:40	0.11543	0.083389	0.949448	112
Versión 27	31 diciembre 12:46	0.11446	0.076132	0.959337	107
Versión 28	31 diciembre 14:13	0.11463	0.076038	0.959174	85
Versión 29	31 diciembre 14:30	0.11415	0.076588	0.958508	80
Versión 30	3 enero 22:24	0.11399	0.077686	0.957335	65
Versión 31	3 enero 22:45	0.11397	0.074304	0.962571	62
Versión 32 a	3 enero 23:23	0.11400	0.074355	0.962374	62
Versión 32 b	3 enero 23:25	0.11441	0.074385	0.962567	62
Versión 32 c	4 enero 00:44	0.11397	0.074341	0.962429	62
Versión 32 d	4 enero 00:46	0.11440	0.074456	0.962376	62
Versión 32 e	4 enero 00:49	0.11458	0.074520	0.962225	62
Versión 32 f	4 enero 00:50	0.11391	0.074179	0.962618	56

Versión	Fecha y hora	Score	RSMLE	R2	Posición kaggle
Versión 33 a	4 enero 19:06	0.11411	0.075897	0.959473	66
Versión 33 b	4 enero 19:11	0.11474	0.076078	0.959307	66
Versión 33 c	4 enero 19:29	0.11401	0.076566	0.958716	66
Versión 33 d	4 enero 19:51	0.11429	0.077212	0.957363	51
Versión 33 e	4 enero 19:59	0.11370	0.079222	0.953972	51
Versión 33 f	4 enero 20:30	0.11393	0.073725	0.962619	51
Versión 33 g	4 enero 21:10	0.11498	0.079520	0.953982	51
Versión 33 h	4 Enero 21:13	0.11453	0.074608	0.961425	51

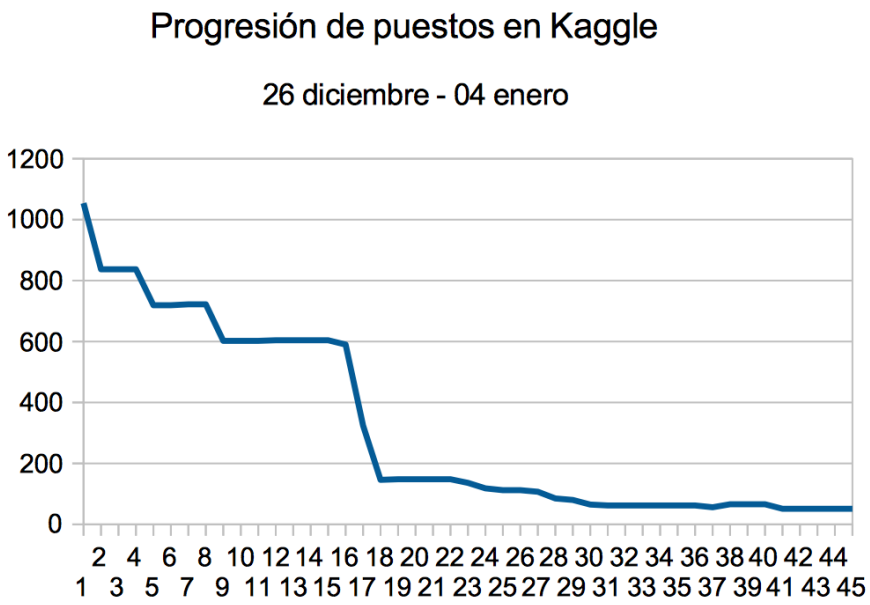


Figura 1: Progresión de los puestos ocupados en la plataforma Kaggle durante todo el proceso de la práctica.

Obersevamos en la gráfica el gran descenso que hemos conseguido a lo largo del desarrollo de la práctica. A continuación explicaremos las estrategias seguidas para conseguir posicionarnos en el **top 2 %** de esta competición.

2. Breve explicación de la estrategia seguida.

En esta sección vamos a explicar los procedimientos seguidos a lo largo de la práctica a la vez que explicamos el por qué de las estrategias elegidas. Para ello nos basaremos en diferentes técnicas estudiadas en la asignatura y asignaturas anteriores y visualizaciones de los datos de entrenamiento.

2.1. Caso base.

Como versión de partida he utilizado la solución propuesta en la web de la asignatura [1].

El preprocesado de los datos que realiza es:

- Eliminación de variables que no dice que no están correladas con *SalePrice* y las que contienen más de un 50 % de valores perdidos.
- Categorización de algunas variables numéricas, aunque en realidad representan categorías. Como *MSSubClass*, *OverallCond*, *KitchenAbvGr*, *YrSold* y *MoSold*.
- Sustitución de valores perdidos utilizando varias estrategias: sustituir por la media, la moda y por 0, en diferentes variables.
- Creación de la variable *TotalSF*, como la suma de los metros cuadrados de toda la parte de la casa y eliminación de las variables que la componen.
- Estandariza ciertas variables numéricas, *LotFrontage*, *LotArea*, *GrLivArea* y *TotalSF*.
- Creación de dummies como *Condition* con *Condition1* y *Condition2*, y *Exterior* con *Exterior1st* y *Exterior2nd* y del resto de variables categóricas.

Para resolver el problema una vez preprocesado los datos, utiliza dos algoritmos de regresión que posteriormente combina realizando la media aritmética entre las etiquetas utilizadas, estos algoritmos son:

- **Elastic Net**, en el cual utiliza *cross-validation* para estimar los parámetros *alpha* y *l1_ratio* y establece número máximo de iteraciones (*max_iter*).
- **Gradient Boosting**, con 3000 estimadores (*n_estimators)*, una tasa de aprendizaje (*learning_rate*) de 0.05, profunidad máxima (*max_depth*) 3, máximo de características utilizadas (*max_features*) raíz cuadrada del número de características del conjunto de datos, mínimo número de muestras necesarias para ser un nodo hoja (*min_samples_leaf*) a 15, mínimo número de muestras necesarias para partir un nodo interno (*min_samples_splot*) a 10 y pérdida a optimizar (*loss = 'huber'*).

2.2. Primeras mejoras en caso base.

Versión 1.

Como en el caso base utilizado no se explica por qué elimina las variables que elimina, eliminamos solo las variables que contengan más de de un 50 % de valores perdidos y ya estudiaremos más adelante la correlación de las variables. Obtenemos mejora por lo que el caso base estaba eliminando variables importantes.

Versión 2.

Dado que Gradient Boosting parece obtener mejores resultados que Elastic Net, a pesar del sobre-aprendizaje, utilizamos solo Gradient Boosting con parámetros *n_estimators = 3000* y *learning_rate = 0.05* como en el Gradient Boosting para estimar el valor de *SalePrice*, lo cual sorprendentemente es un fracaso por lo que volvemosa la versión anterior.

Versión 3.

Volviendo a la versión con el ensemble de los dos algoritmos de regresión utilizados, cambiamos Gradient Boosting por **XGBoost**, dado que por regla general obtiene mejores resultados. A pesar de que en nuestro conjunto de entrenamiento sí que obtuvo mejores resultados obtenemos peor score en la plataforma.

2.3. Preprocesado en R

A continuación, nos planteamos mejorar o cambiar el preprocesado de los datos utilizando R. Realizaremos varias versiones que planteamos a continuación:

Versión 4.

Dado que en el caso base reemplaza los valores perdidos siguiendo diferentes estrategias aparentemente aleatorias, en vez de probar con cada una de esas estrategias vamos a utilizar algoritmos de predicción para estimarlos. Nos planteamos utilizar en primer lugar vecino más cercano para estimar estos valores. Tenemos que eliminar aquella variables que tienen demasiados valores perdidos por lo que no hay suficientes muestras para estimar el valor. Vemos la parte del script de R que realiza está función a continuación. Utilizamos el número de vecinos por defecto, $k = 10$.

```
## Aunque utilicemos predicción para el resto de valores perdidos, eliminamos aquellos que tienen demasiados
## No pueden estimarse el resto de valores
train['Alley'] <- NULL
train['MiscFeature'] <- NULL
train['Fence'] <- NULL
train['PoolQC'] <- NULL

test['Alley'] <- NULL
test['MiscFeature'] <- NULL
test['Fence'] <- NULL
test['PoolQC'] <- NULL

#Estimamos el resto de valores perdidos
noNA_train <- knnImputation(train) #Valores por defecto
noNA_test <- knnImputation(test) #Valores por defecto
```

Obtenemos nos proporciona una mejora considerable.

Versión 5.

Ahora, nos planteamos sustituir la estandarización que realizamos en python por una estandarización (escalado + centrado) utilizando la función *preProcess* de R, observamos a continuación el código utilizado, aunque el resultado obtenido es decepcionante. Probablemente porque estamos normalizando variables que no se debían normalizar al no haber categorizado primero a diferencia de en python.

```
#Estimamos el resto de valores perdidos
noNA_train <- knnImputation(train) #Valores por defecto
noNA_test <- knnImputation(test) #Valores por defecto

preprocess_1 <- preProcess(noNA_train[,2:(length(noNA_train)-1)], method = c("scale", "center")) #range por defecto en 0-1
preprocess_2 <- preProcess(noNA_test[,2:(length(noNA_test))], method = c("scale", "center"))

pred_train <- predict(preprocess_1, noNA_train[,2:(length(noNA_train)-1)])
pred_test <- predict(preprocess_2, noNA_test[,2:(length(noNA_test))])

new_train <- noNA_train
new_train[,2:(length(new_train)-1)] <- pred_train

new_test <- noNA_test
new_test[,2:(length(new_test)-1)] <- pred_test
```

Versión 6.

Volvemos a trabajar sobre la versión 4 pues es la mejor obtenida hasta el momento. En esta nueva versión realizamos la eliminación de variables poco correladas con *SalePrice*, para ello imprimimos la matriz de correlación con la función *cor* de R (no la incluímos pues es demasiado grande) y eliminamos aquellos atributos cuya correlación sea baja.

Versión 7.

Volvemos a trabajar sobre la versión 4. Dado que la selección de las variables a eliminar a mano, mirando la matriz de correlaciones no nos ha proporcionado buenos resultados, procedemos a utilizar un método de reducción de las características que nos proporciona nuevas características que son combinaciones de las anteriores cumpliendo un límite de varianza que nosotros establezamos. El método es *pca* en R, y establecemos un *threshold = 0.98*. Obtenemos que los datos obtenidos son ligeramente peores a los ya obtenidos, probablemente por el sobreaprendizaje producido.

Versión 8.

En esta versión, a la vista de que no hemos conseguido ninguna mejora, nos proponemos incidir sobre la predicción de los valores perdidos que nos ha dado buenos resultados. Para ello, utilizamos 30 vecinos para la predicción de los valores perdidos, obteniendo una nueva mejora.

Dado que nos hemos conseguido una mejora, vamos a probar con diferentes valores de k (número de vecinos), en las siguientes versiones.

Versión 9.

Como al aumentar el número de vecinos más cercanos hemos mejorado, aumentamos aún más el número de vecinos. Utilizamos 100 vecinos para la predicción aunque obtenemos no obtenemos mejora.

Versión 10.

Como ha empeorado esta vez, probamos con un valor intermedio, 50, obteniendo que tampoco mejoramos por lo que dejamos la estimación de los valores perdidos con 30 vecinos.

Versión 11.

En esta versión, nos vamos a propononer utilizar otro algoritmo de predicción para los valores perdidos, utilizamos Random Forest con 15 árboles como observamos en el siguiente código:

```
## Aunque utilicemos predicción para el resto de valores perdidos, eliminamos aquellos que tienen demasiados
## No pueden estimarse el resto de valores
train['Alley'] <- NULL
train['MiscFeature'] <- NULL
train['Fence'] <- NULL
train['PoolQC'] <- NULL

test['Alley'] <- NULL
test['MiscFeature'] <- NULL
test['Fence'] <- NULL
test['PoolQC'] <- NULL


rf_train <- mice(train, meth = "rf", ntree = 15)
rf_test <- mice(test, meth = "rf", ntree = 15)
```

Como no mejora los resultados, nos quedamos como predicción definitiva de los valores perdidos el vecino más cercano con 30 vecinos.

2.4. Preprocesado en python.

Una vez hemos encontrado la mejor forma de predecir los valores perdidos en R, viendo que el resto de ideas no han funcionado, pasamos a seguir con el preprocesamiento en python.

Versión 12.

Fijándonos en la matriz de correlación anteriormente generada, eliminamos algunas variables como observamos en el siguiente código

```
#  Tenemos la variable TotalBsmtSF que recoge la suma de: BsmtFinSF1, BsmtFinSF2 y BsmtUnfSF -> Eliminamos estas variables pues son redundantes
features.drop(['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF'], axis=1, inplace=True)

#  Igualmente tenemos (BsmtFullBath y BsmtHalfBath) a la vez que tenemos (FullBath y HalfBath)
# -> Como uno está incluido en el otro -> Eliminamos estas dos primeras
features.drop(['BsmtHalfBath', 'BsmtFullBath'], axis=1, inplace=True)
```

Además, estandarizamos más variables de las que estandarizábamos en el caso base, como observamos a continuación:

```
## Standardizing numeric features
numeric_features = features.loc[:,['LotFrontage', 'LotArea', 'GrLivArea', 'TotalSF', 'MasVnrArea', 'LowQualFinSF',
'EnclosedPorch', 'X3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal']]
numeric_features_standardized = (numeric_features - numeric_features.mean())/numeric_features.std()
```

Versión 13.

Como vemos que los resultados obtenidos no son los esperados, dejamos de estandarizar más variables pero sí que eliminamos las variables anteriormente eliminadas.

Versión 14.

Los datos tampoco han sido los esperados, por eso, nos planteamos si una normalización min-max (0,1) nos proporcionaría mejores resultados, para ello utilizamos la función *MinMaxScaler()* del paquete *preprocessing* de python para normalizar las variables numericas, mientras seguimos manteniendo la eliminación de variables anterior.

```
## Standardizing numeric features
numeric_features = features.loc[:,['LotFrontage', 'LotArea', 'GrLivArea', 'TotalSF', 'MasVnrArea', 'LowQualFinSF', 'EnclosedPorch',
'X3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MSSubClass', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'FullBath', 'HalfBath',
'BedroomAbvGr', 'Fireplaces', 'GarageCars', 'GarageArea', 'GarageYrBlt']]

# Hacemos normalización min_max de todos los atributos numericos
preprocess = preprocessing.MinMaxScaler()
numeric_features_standardized = preprocess.fit_transform(numeric_features)
```

Aunque habíamos intuído de la información de los datos que iba a ser beneficioso la eliminación de las variables anteriormente comentadas, los resultados prácticos nos han demostrado lo contrario. Volvemos a incluirlas.

Versión 15.

En esta versión nos proponemos la eliminación de outliers, pues como obtenemos en la Scatter Matrix del apéndice A, hay muchos variables que contienen outliers. Por eso, implementamos una eliminación de outliers genérica, eliminando aquellas muestras que tengan algunos de sus atributos verificando que la diferencia en valor absoluto con la media sea mayor que tres veces la desviación típica. (Seguimos manteniendo la normalización min-max)

Versión 16.

Hasta ahora, los resultados obtenidos no han producido mejoras. Por tanto, nos planteamos volver a lo anterior realizando estandarización en lugar de normalización min-max (0,1) pero ahora lo aplicamos a todas las variables numéricas, esto sí nos produce notables mejoras.

Versión 17

Partiendo de la versión 16, nos proponemos aplicar transformaciones a los diferentes atributos. Igual que el caso base aplica una transformación logarítmica al las etiquetas, aplicamos una transformación logarítmica a aquellas variables numéricas que no tomen el valor 0.

Versión 18.

Como hemos obtenido mejora en la versión anterior, en esta versión queremos aplicar la misma transformación a todas las variables numéricas.

Para eso, para las variables que toman valor 0 en algún momento le aplicamos una translación al $(1, max(variable) + 1)$ y aplicamos dicha transformación.

```
#Ponemos logaritmico los que no toman valor 0
features['OverallQual'] = np.log(features['OverallQual'])
features['GrLivArea'] = np.log(features['GrLivArea'])
features['X1stFlrSF'] = np.log(features['X1stFlrSF'])
features['MSSubClass'] = np.log(features['MSSubClass'])
features['LotFrontage'] = np.log(features['LotFrontage'])
features['LotArea'] = np.log(features['LotArea'])
features['OverallCond'] = np.log(features['OverallCond'])
features['YearBuilt'] = np.log(features['YearBuilt'])
features['YearRemodAdd'] = np.log(features['YearRemodAdd'])
features['X1stFlrSF'] = np.log(features['X1stFlrSF'])
features['YrSold'] = np.log(features['YrSold'])
features['MoSold'] = np.log(features['MoSold'])
```

Versión 19.

Nos proponemos ahora estudiar la influencia de las diferentes variables en el *SalePrice* estimado obteniendo el *heatmap* (ver apéndice A). Obtenemos que hay en concreto tres variables que influyen muy poco en la predicción de *SalePrice* y son *KitchenAbvGr*, *EnclosedPorch* y *MSSubClass*. Así que los eliminamos.

Vesión 20.

Como no hemos obtenido mejora, cosa que esperábamos, repetimos la eliminación de las variables quitando la transformación logarítmica de las variables que tomaban valor 0 para no tener que realizar la translación, pero obtenemos que empeora aún más. Por lo que deshacemos la idea de eliminar estas variables.

2.5. Diferentes algoritmos de regresión y su configuración.

Nos proponemos en esta sección probar a utilizar otros algoritmos de predicción además de variar la configuración de estos algoritmos tanto para optimizar sus resultados como para evitar el sobreaprendizaje.

Versión 22.

En este versión añadimos el algoritmo de regresión **Lasso**, y utilizamos como etiquetas la media aritmética de las etiquetas estimadas por cada uno de los algoritmos.

Para la implementación del **Lasso**, dado que es un método muy perceptible a outliers, utilizamos el método del paquete *preprocessing* de python *RobustScaler()* que escala las variables sin tener en cuenta los outliers de la siguiente manera:

```
'''
    LASSO
'''
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
```

Versión 23.

Siguiendo la misma línea que en la versión anterior, añadimos un nuevo algoritmo de regresión el **Kernel Ridge Regression (KRR)**. Como este algoritmo no se ve especialmente afectado por los outliers, lo utilizamos sobre los datos anteriormente estandarizados de la forma habitual. Utilizamos como parámetros del algoritmo los que observamos en el siguiente código

```
'''
    KRR
'''

KRR = KernelRidge(alpha=0.2, kernel='polynomial', degree=2, coef0=1)
```

Las etiquetas las calculamos con la media aritmética de las cuatro etiquetas estimadas por los cuatro métodos utilizados. Conseguimos así la mejor solución hasta el momento.

Versión 24.

Nos planteamos ahora mejorar la estimación variando las configuraciones de los algoritmos. Nos centramos en KKR, pues nos ha producido una buena mejora pero es un método que sobreaprende mucho.

Hacemos varias pruebas en local utilizando cross validation y obtenemos que los parámetros que optimizan el algoritmo KRR para estos datos son *alpha = 0.2*, *degree = 2* y *coef0 = 1*.

Aumentar el valor de alpha y disminuir el grado de los polinomios utilizados nos ayuda a disminuir el sobreaprendizaje. Así, aunque el método nos ofrece una peor estimación de las etiquetas del train, vemos que mejoramos el score en la plataforma.

Versión 25.

Realizamos de forma análoga para el Gradient Boosting. Observamos que variando la tasa de aprendizaje conseguimos empeorar el algoritmo o que sobreaprenda en exceso, obteniendo que la tasa de aprendizaje óptima para estos casos es 0.05, variamos solo el valor del número de estimadores fijándolo en 5000.

Conseguimos una pequeña mejora también.

Versión 26.

Realizamos de la misma forma para el método Lasso obteniendo que el valor *alpha* que optimiza los datos es 0.001. Sin embargo, al subir esta nueva solución a Kaggle comprobamos que produce un score peor, por lo que reestablecemos el antiguo valor de *alpha.*

2.6. Otras transformaciones

Versión 27.

Pensando un poco sobre el preprocesado de los datos, nos damos cuenta que la variable *Id* no es más que una enumeración de las muestras recogidas. Por tanto quitamos esta variable y utilizamos los algoritmos con las configuraciones óptimas que hemos establecido.

Versión 28.

Repasando el resto de variables, fijamos nuestra atención en las variables temporales. Pensamos que podría ser una buena mejora si tuviésemos en cuenta los años transcurridos en vez de el año en que ocurrió.

Versión 29.

Como hemos obtenido peores resultados, deshacemos la transformación del las variables temporales.

Además, nos damos cuenta de que teníamos un error en la obtención de las etiquetas en el caso de Lasso. Así, que también lo corregimos.

2.7. Creación de nuevas variables.

Teniendo en cuenta que la eliminación de variables nos ha empeorado los resultados obtenidos, nos planteamos ahora añadir nuevas variables que sean combinación de las que ya tenemos.

Versión 30

Buscamos en la Scatter Matrix (ver apéndice B) en las gráficas del resto de variables en función de *SalePrice* figuras que sigan funciones conocidas.

Nos fijamos en que la Scatter Matrix de *SalePrice - OverallQual* y la de *SalePrice - YearBuilt* siguen una función cuadrática. Creamos las variables *QuadraticOverallQual* y *QuadraticYearBuilt* y las añadimos al conjunto de variables numéricas, siendo posteriormente estandarizadas y modificadas de forma logarítmica.

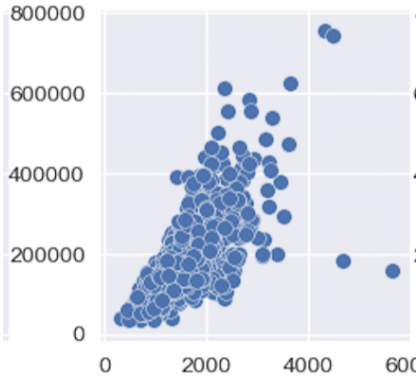
Versión 31

Como hemos conseguido una mejor solución, nos proponemos ahora encontrar variables que entre ellas sigan una relación lineal. Fijándonos nuevamente en la Scatter Matrix podemos ver que *TotalBsmtSF* y *GrLivArea* siguen una progresión lineal. Así, creamos la variable *TotalBsmtSF-GrLivArea* que se corresponde con la multiplicación de estas dos variables.

De forma análoga añadimos esta nueva variable creada al conjunto de variables numéricas siendo posteriormente estandarizada.

2.8. Tratamiento outliers

Aunque anteriormente habíamos tratado los outliers de forma genérica en función de percentiles, nos proponemos ahora estudiar un caso concreto pues en la Scatter Matrix observamos que hay varias variables con grandes outliers. Estudiamos una de estas variables en concreto, que además era una de las más relacionadas con *SalePrice*, *GrLivArea*.



Observamos que hay 5 datos que pudieran ser outleirs, aunque en concreto dos que parecen ir en contra de la progresión del resto de datos. Probamos a eliminarlos de diferentes formas pues, cuatro de ellos pertenecen a conjuntos de train y uno de ellos al conjunto de test.

2.8.1. Versión 32

- **Versión 32 a**: Eliminamos el outlier con mayor *GrLivArea* del conjunto de test y de train.
- **Versión 32 b**: Eliminado solo el outlier con mayor *GrLivArea* del conjunto de train.
- **Versión 32 c**: Eliminamos los cuatro outliers pertenecientes al conjunto de train.
- **Versión 32 d**: Eliminamos los dos outliers con mayor *GrLivArea* del conjunto de train.
- **Versión 32 e**: Quitando estos dos últimos outliers más el outlier del conjunto de test.
- **Versión 32 f:** Quitando los cinco outliers.

El que mejor resultado nos ha proporcionado en el score de Kaggle ha sido la opción c, por la que será con la que continuemos para el resto de mejoras.

2.9. Ponderación de los algoritmos

Hasta el momento, hemos utilizado las etiquetas de forma que son resultado de la media aritmética entre las etiquetas estimadas por los cuatro algoritmos. En esta sección nos proponemos realizar ponderaciones de forma que optimicemos el resultado.

Versión 33.

Nos planteamos las siguientes combinaciones:

- **Versión 33 a:** 0.3 EN + 0.2 GB + 0.2 Lasso + 0.3 KRR
- **Versión 33 b:** 0.2 EN + 0.2 GB + 0.3 Lasso + 0.3 KRR
- **Versión 33 c:** 0.3 EN + 0.3 GB + 0.2 Lasso + 0.2 KRR

  Como hemos obtenido mejoras con esta combinación, seguimos priorizando los dos primeros algoritmos.
- **Versión 33 d:** 0.4 EN + 0.4 GB + 0.1 Lasso + 0.1 KRR
- **Versión 33 e:** 0.35 EN + 0.25 GB + 0.2 Lasso + 0.2 KRR
- **Versión 33 f:** 0.25 EN + 0.35 GB + 0.2 Lasso + 0.2 KRR
- **Versión 33 g:** 0.4 EN + 0.3 GB + 0.15 Lasso + 0.15 KRR
- **Versión 33 g:** 0.3 EN + 0.4 GB + 0.15 Lasso + 0.15 KRR

La solución que mejor Score ha obtenido en Kaggle ha sio la versión 33 e, por lo que sería la solución definitiva.

Apéndice A

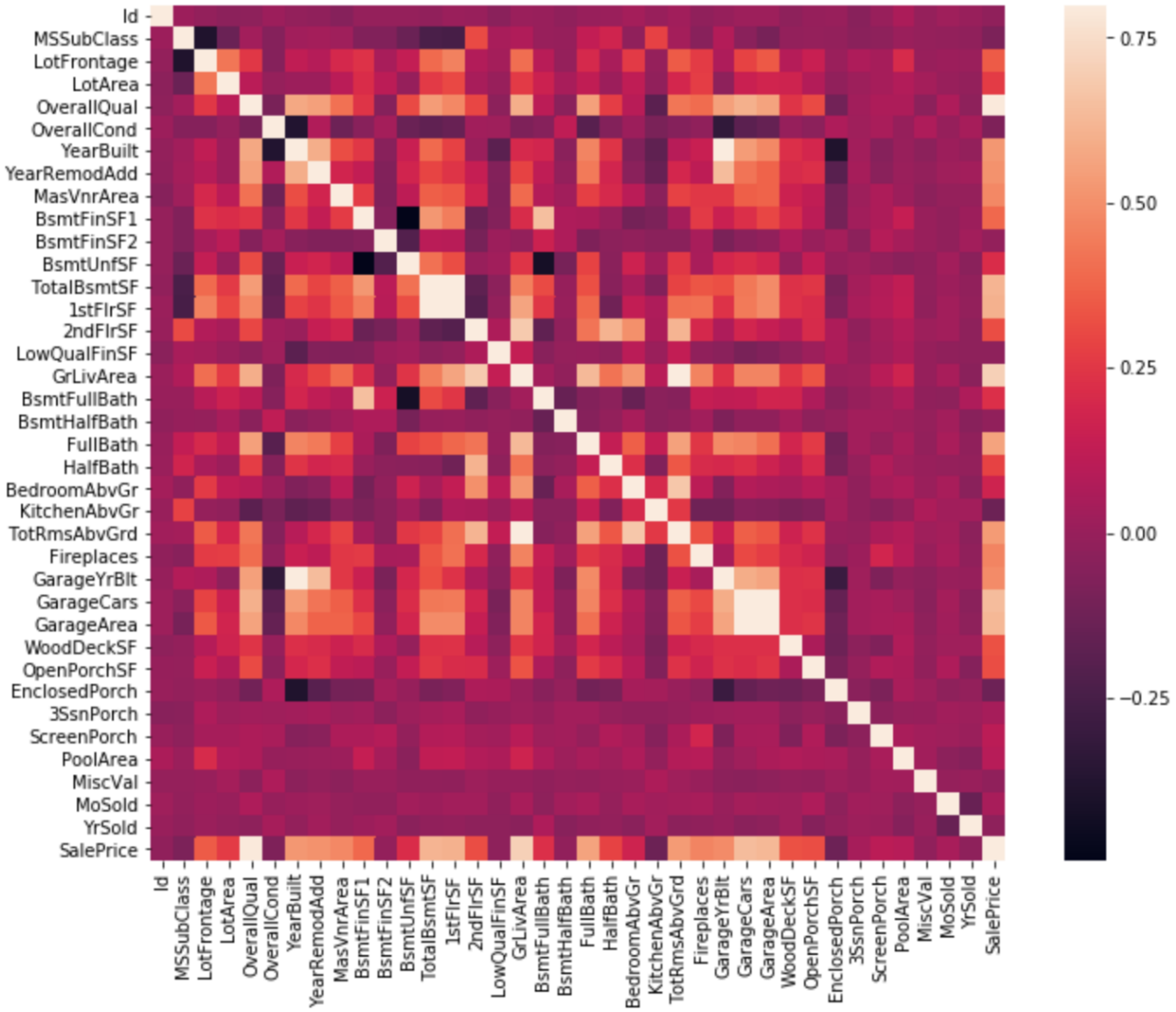


Figura 2: Heatmap

Apéndice B

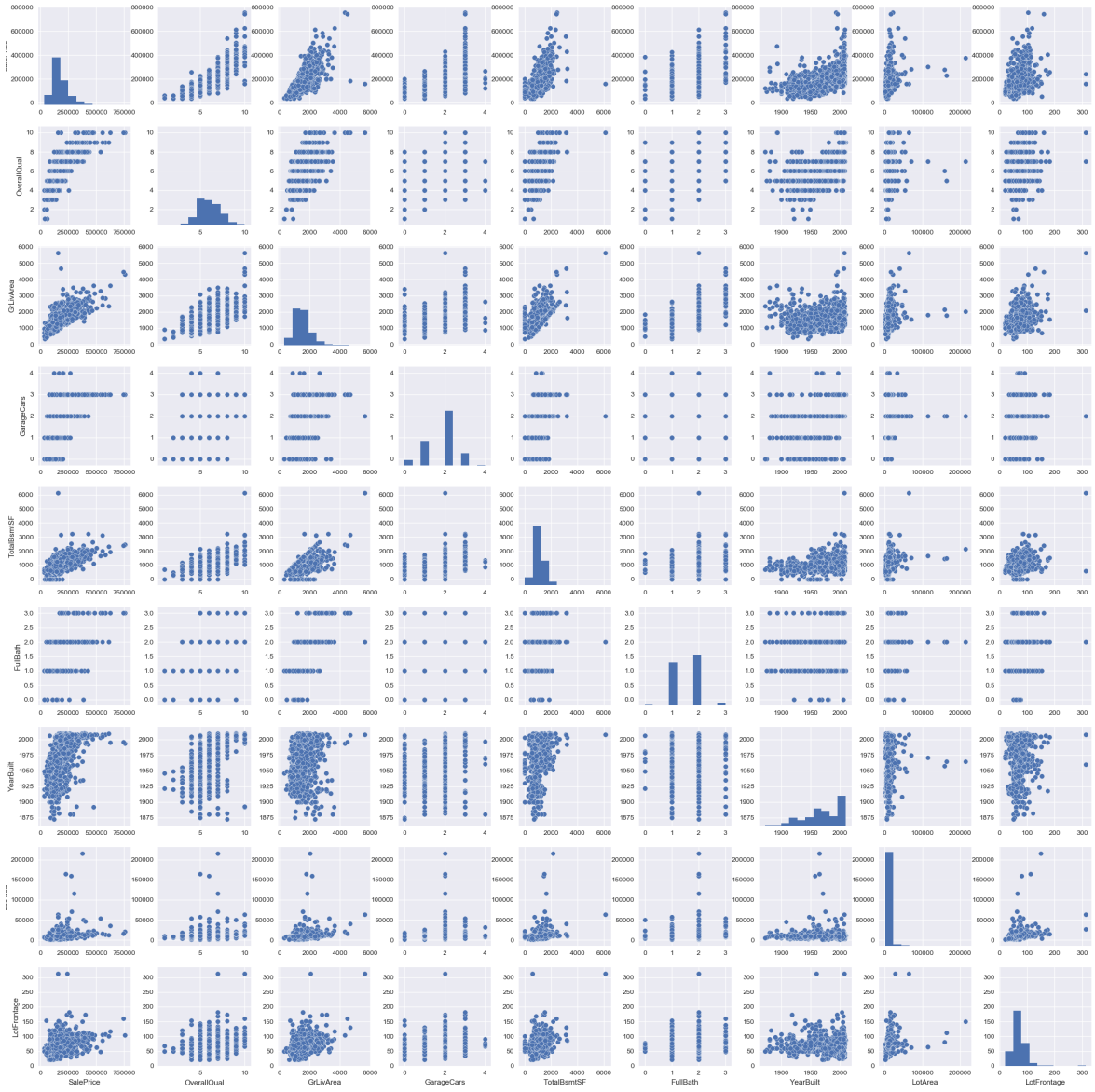


Figura 3: Scatter Matrix

Referencias

[1] Web de la asignatura http://sci2s.ugr.es/graduateCourses/in

[2] XGboost http://xgboost.readthedocs.io/en/latest/get_started/

[3] Lasso http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

[4] Elastic Net http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html

[5] Gradient Boosting Regressor http://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html

[6] Kernel Ridge Regression http://scikit-learn.org/stable/modules/kernel_ridge.html