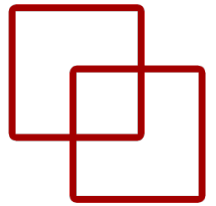
Fase de tratamiento de datos

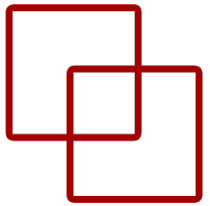
Dr. Manuel Castillo-Cara
www.manuelcastillo.eu
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Educación a Distancia (UNED)

Preliminar



-

Índice



- Evaluación de métricas.
  - Clasificación.
  - Regresión
- Feature selection.
  - Técnicas de feature selection.
- Feature Importance.
- Reducción de dimensiones.

Evaluación de métricas

Métricas básicas

- El rendimiento de un modelo de SLA refleja la diferencia entre las predicciones de éste para un conjunto de entradas X, y los valores Y esperados.
  - **Regresión**: El error cuadrático medio, que se formula como:

$$ECM(h_\theta, X) = \frac{1}{N}\sum_{i=1}^{N}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

  - **Clasificación**: La tasa de acierto/error

$$ERROR(c_\theta, X) = \frac{1}{N}\sum_{i=1}^{N} error\left(c_\theta(x^{(i)}), y^{(i)}\right)$$

$$error\left(c_\theta(x^{(i)}), y^{(i)}\right) = \begin{cases} 0 & \text{si } c_\theta(x^{(i)}) = y^{(i)} \\ 1 & \text{si } c_\theta(x^{(i)}) \neq y^{(i)} \end{cases}$$

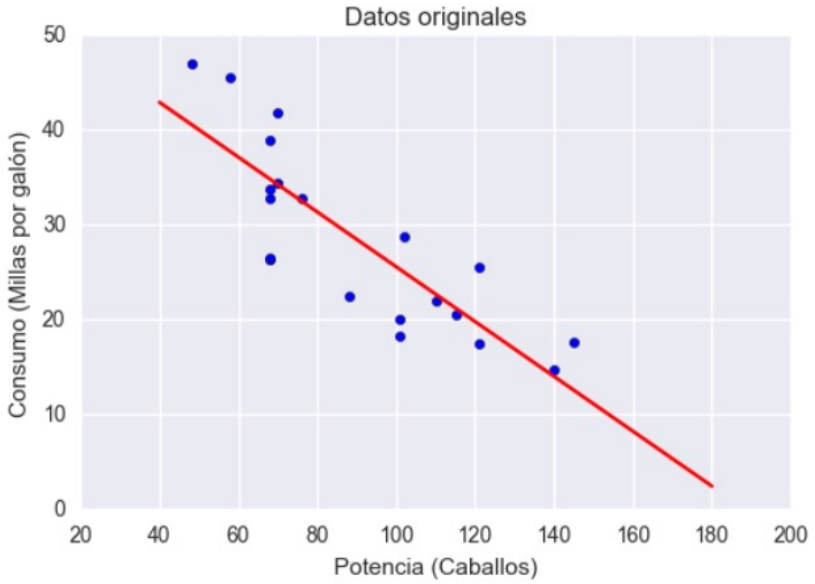
$$ACIERTO(c_\theta, X) = 1 - ERROR(c_\theta, X)$$

Donde $\boldsymbol{X}$ representa un conjunto de datos de tamaño $\boldsymbol{N}$; $\boldsymbol{h_\theta}$ y $\boldsymbol{c_\theta}$, respectivamente, los modelos de regresión y clasificación; y $\boldsymbol{\theta}$ representa los parámetros de los modelos.

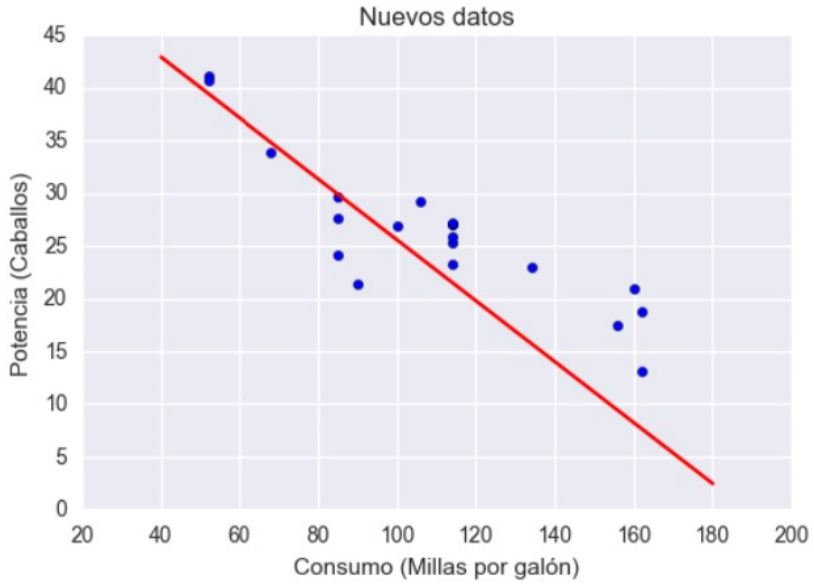
Error en LiR y LoR (I)

- Devuelven la configuración de parámetros **óptima**, es decir, aquella que **minimiza** el coste con respecto a los datos **utilizados en el aprendizaje**.
- El error del modelo **aumenta** (puede no ocurrir en algunos casos) en la predicción de nuevos casos.
  - **Error predictivo**

Ejemplo: LiR



Error cuadrático medio = 26.22

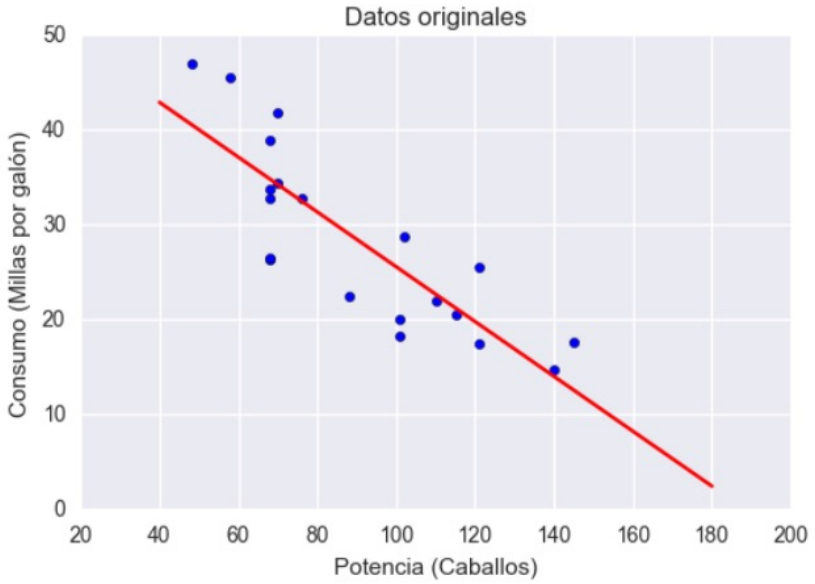


Error cuadrático medio = 34.36

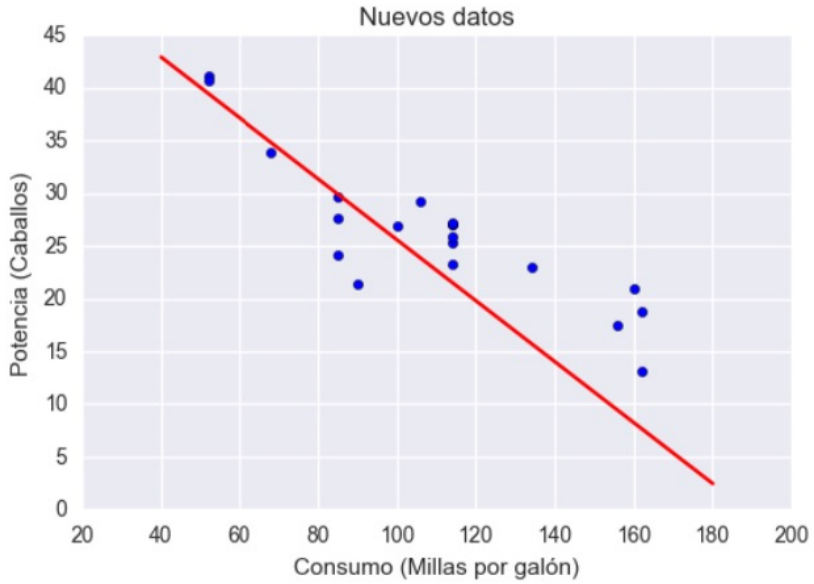
2. Error en LiR y LoR (I)

- Devuelven la configuración de parámetros **óptima**, es decir, aquella que **minimiza** el coste con respecto a los datos **utilizados en el aprendizaje**.
- El error del modelo **aumenta** (puede no ocurrir en algunos casos) en la predicción de nuevos casos.
  - **Error predictivo**

Ejemplo: LiR



Error cuadrático medio = 26.22

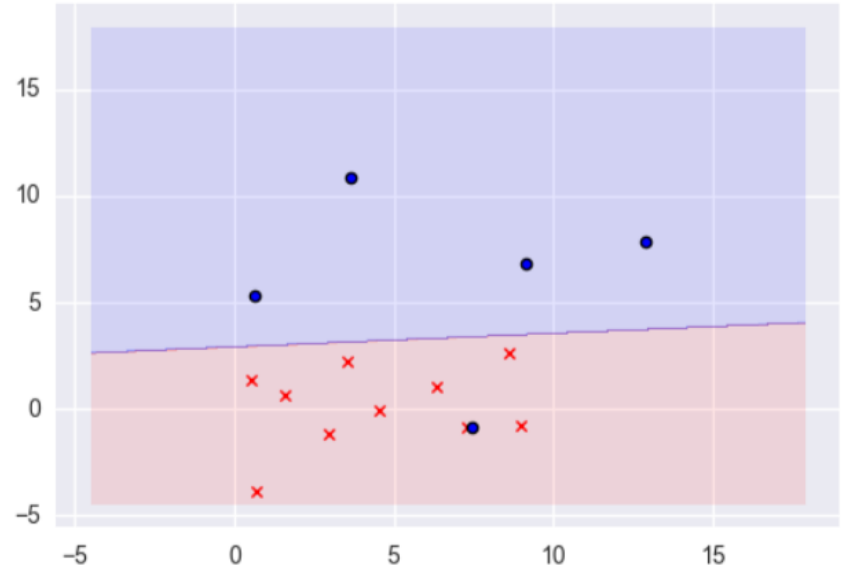


Error cuadrático medio = 34.36

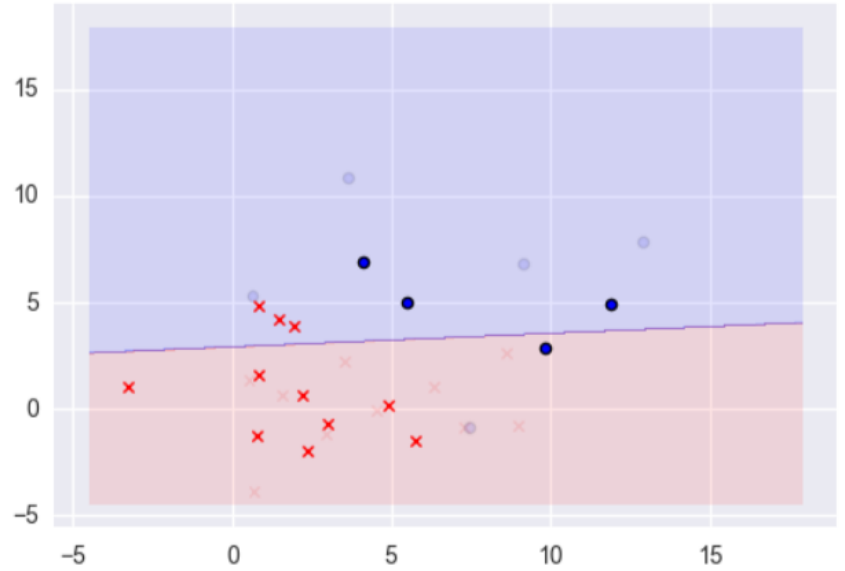
2. Error en LiR y LoR (I)

- Devuelven la configuración de parámetros **óptima**, es decir, aquella que **minimiza** el coste con respecto a los datos **utilizados en el aprendizaje**.
- El error del modelo **aumenta** (puede no ocurrir en algunos casos) en la predicción de nuevos casos.
  - **Error predictivo**

Ejemplo: LoR



Error (datos originales) = 6.6 %

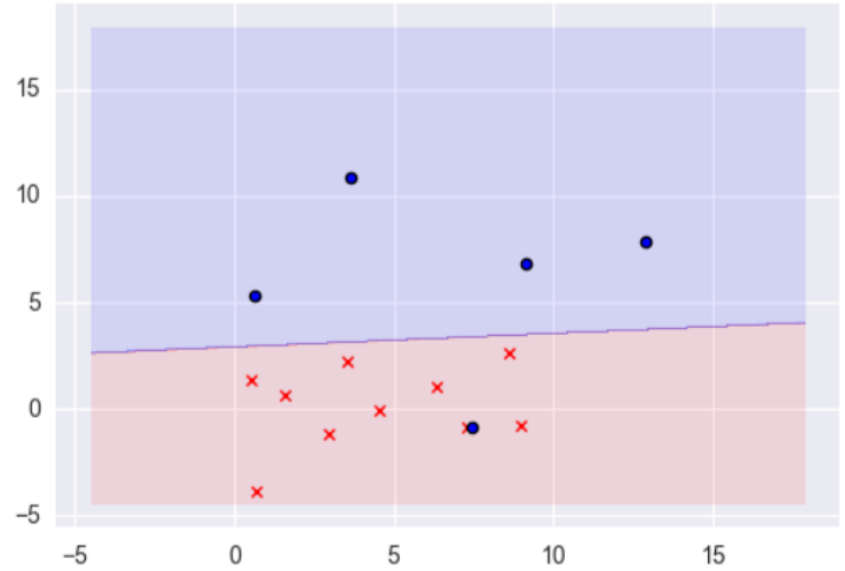


Error (nuevos datos) = 26.6 %

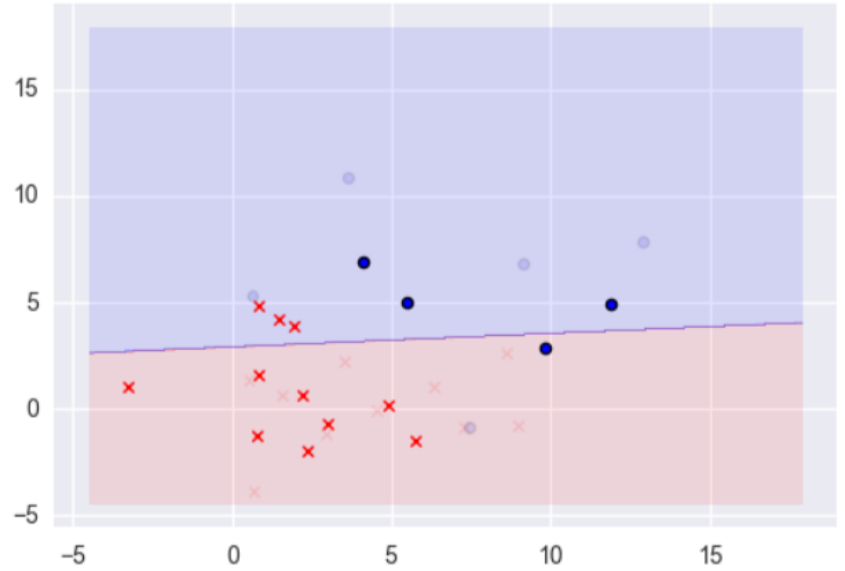
2. Error en LiR y LoR (I)

- Devuelven la configuración de parámetros **óptima**, es decir, aquella que **minimiza** el coste con respecto a los datos **utilizados en el aprendizaje**.
- El error del modelo **aumenta** (puede no ocurrir en algunos casos) en la predicción de nuevos casos.
  - **Error predictivo**

Ejemplo: LoR



Error (datos originales) = 6.6 %



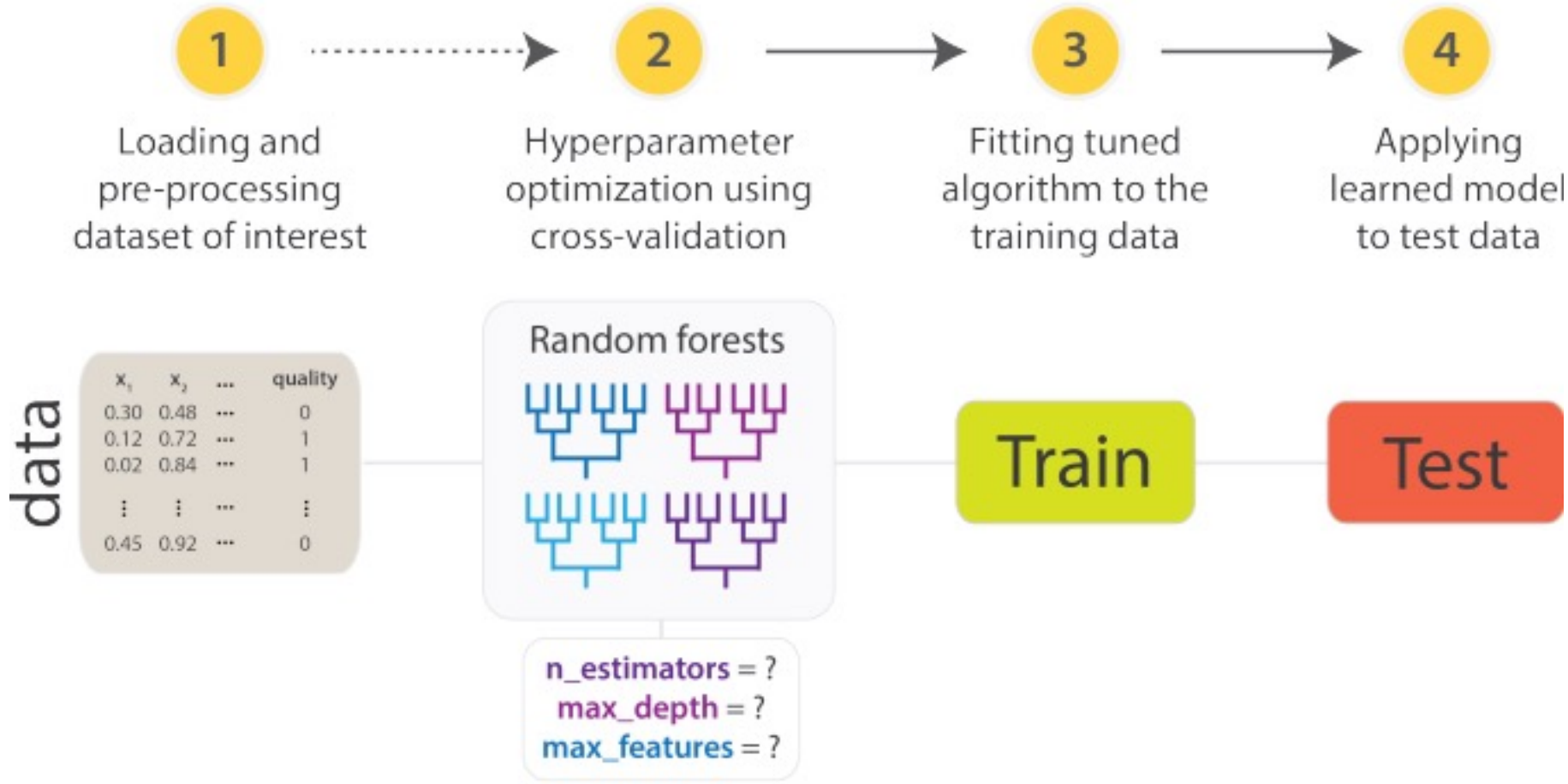
Error (nuevos datos) = 26.6 %

Métodos de remuestreo

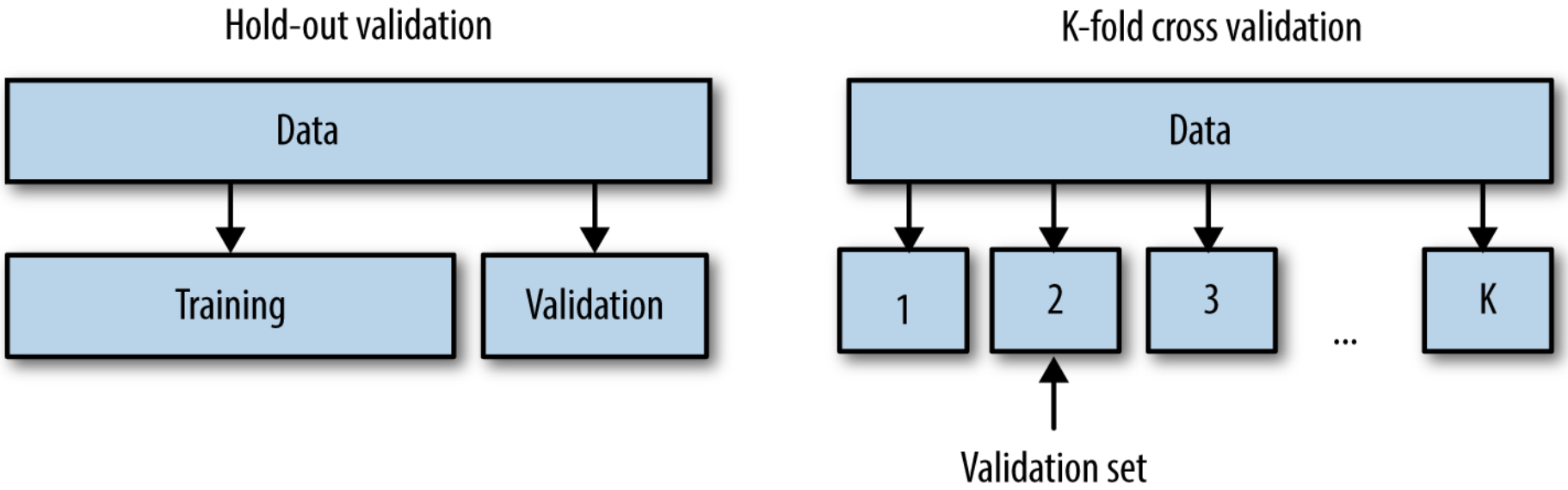
Introducción

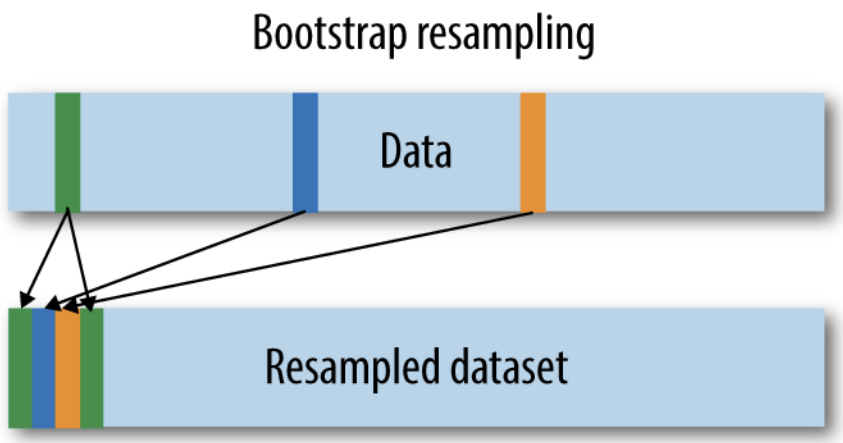
- Objetivo: Evaluar los algoritmos.
- Dividir los datos para la evaluación.
- Cuatro enfoques de división:
  - Cómo dividir un conjunto de datos en subconjuntos por porcentaje para entrenamiento/validación.
  - Cómo evaluar la robustez del modelo utilizando la validación cruzada, k-fold, con y sin repeticiones.
  - Cómo evaluar la robustez del modelo usando una validación cruzada dejando uno fuera (LOOCV).
  - División en train/test repetidos aleatoriamente.

Estimar la métrica



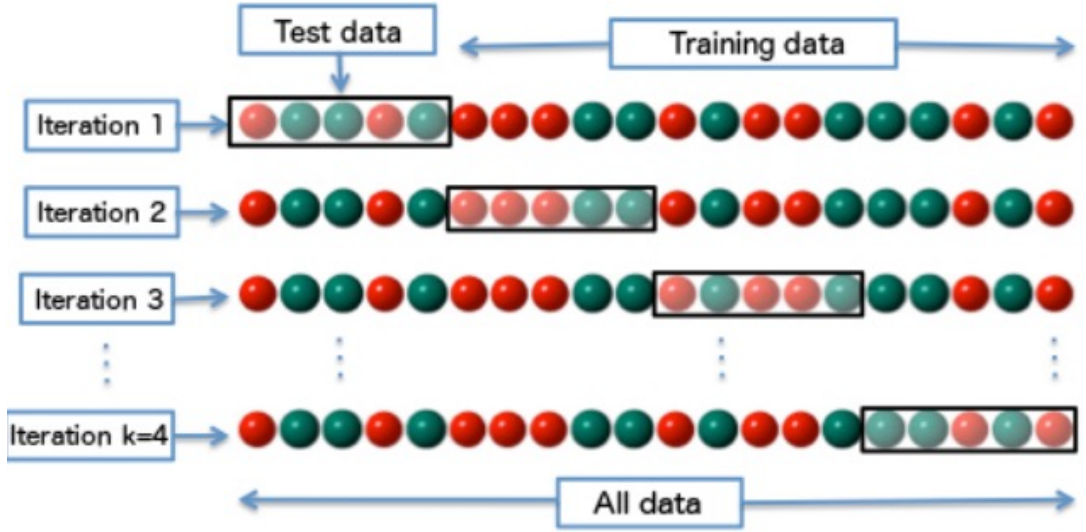
Técnicas para remuestreo





Validación cruzada

- División en k subconjuntos de datos, es decir, k-fold.
- Los valores comunes para k son de 5 y 10.




```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.60% (5.16%)
```

División por porcentaje

- Realiza una división del conjunto de datos por porcentaje directo, por ejemplo, 67% entrenamiento y 33% validación.

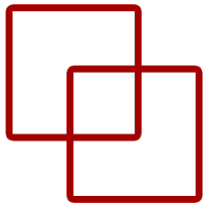
FIGURA 4.2: División de los datos de entrenamiento/validación.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                    test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print(f"Accuracy: {result*100:,.2f}%")
```

```
Accuracy: 78.74%
```

Métricas para clasificación

Accuracy y Kappa



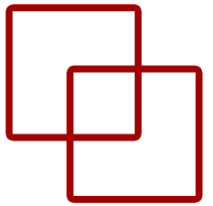
- Para problemas de clasificación
- Es el porcentaje de instancias correctamente clasificadas de todas las instancias. Muy útil para la clasificación binaria pero no para clasificación multiclase.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
scoring = 'accuracy'
results = cross_val_score(model, X_clas, Y_clas, cv=kfold, scoring=scoring)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.60% (5.16%)
```

Kappa o Cohen's Kappa



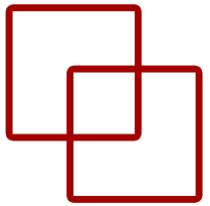
- Para problemas de clasificación.
- Útil para usar en problemas que tienen un desequilibrio en las clases
  - Por ejemplo, una división del 70 % al 30 % para las clases 0 y 1 y puede alcanzar el 70% al predecir que todas las instancias son para la clase 0, pero ninguna para la 1.
- Utilizar la clase "*cohen_kappa_score*"

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import cohen_kappa_score
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X_clas, Y_clas,
                                    test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
cohen_score = cohen_kappa_score(Y_test, predicted)
print(f"Cohens score: {cohen_score*100.0:,.2f}%")
```

```
Cohens score: 52.42%
```

Métricas para Regresión

Error medio absoluto



- El error absoluto medio (MAE) es la suma de las diferencias absolutas entre las predicciones y los valores reales.
- Da una idea de cuán erróneas fueron las predicciones.
- La medida da una idea de la magnitud del error, pero no tiene idea de la dirección (por ejemplo, sobre o por debajo de la predicción).

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression

test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X_reg, Y_reg,
                                   test_size=test_size, random_state=seed)
model = LinearRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
MAE = mean_absolute_error(Y_test, predicted)
print(MAE)
```

```
3.3286948545850623
```

R^2

- Proporciona una indicación del ajuste de un conjunto de predicciones a los valores reales.
- En la literatura estadística, esta medida se llama coeficiente de determinación.
- Este es un valor entre 0 (si no tiene ajuste) y 1 (ajuste perfecto).

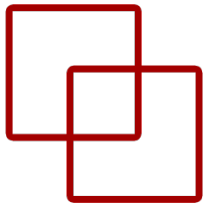
```
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X_reg, Y_reg,
                                   test_size=test_size, random_state=seed)
model = LinearRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
R2 = r2_score(Y_test, predicted)
print(R2)
```

```
0.6663089606572575
```

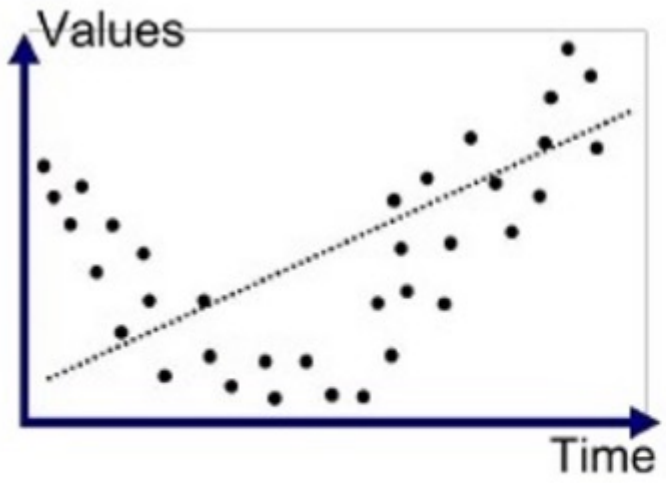
Feature Selection

Definición

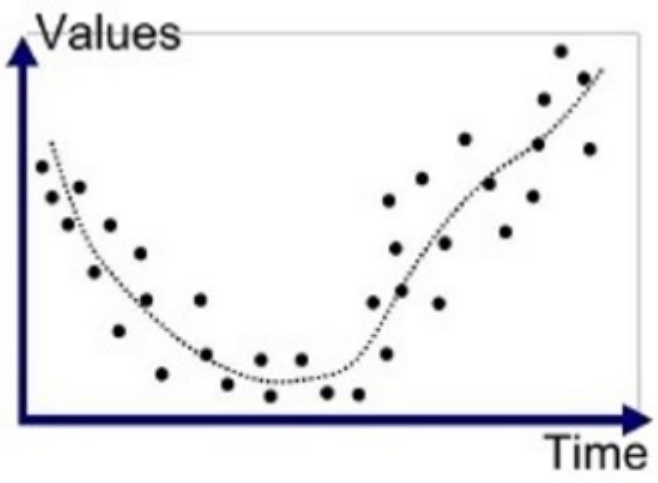


- Es el proceso de seleccionar un subconjunto de características pertinentes (variables, predictores) para su uso en construcción de modelos.
- Cuatro razones:
  - Simplificación de modelos.
  - Menor tiempo de entrenamiento.
  - Maldición de dimensionalidad (course of dimensionality).
    - ¿“Contra más datos mejor”?
  - Reducir el overfitting.
- Como dice el principio de Occam’s Razor:
  - “*Los modelos más simples son los mejores.*”

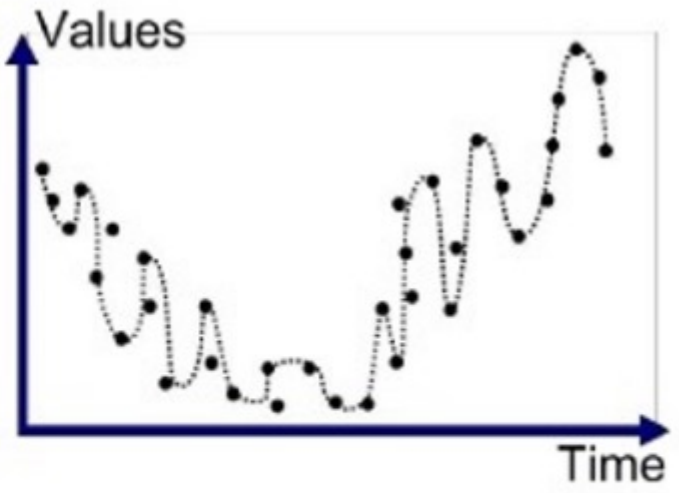
Overfitting Vs Underfitting



Underfitted

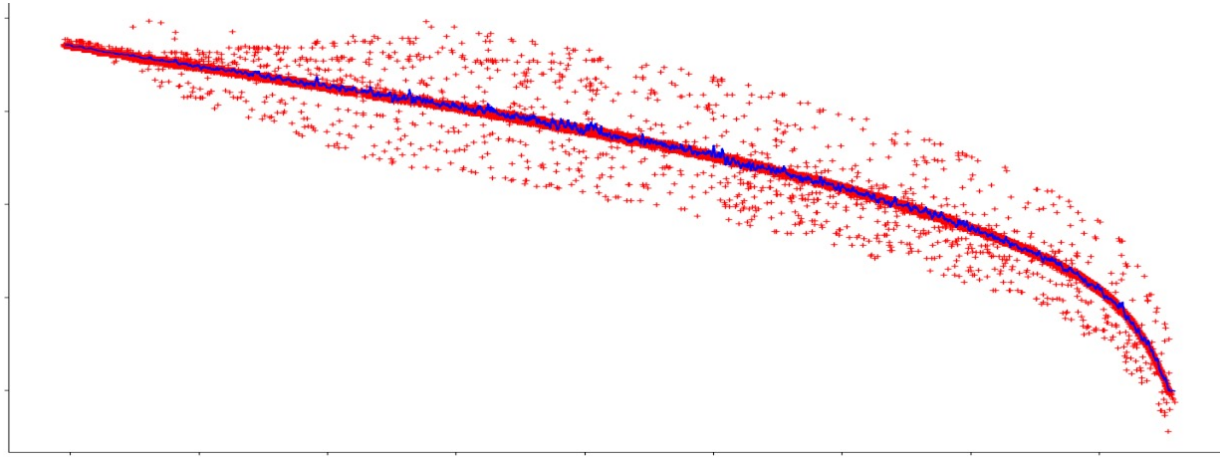


Good Fit/Robust

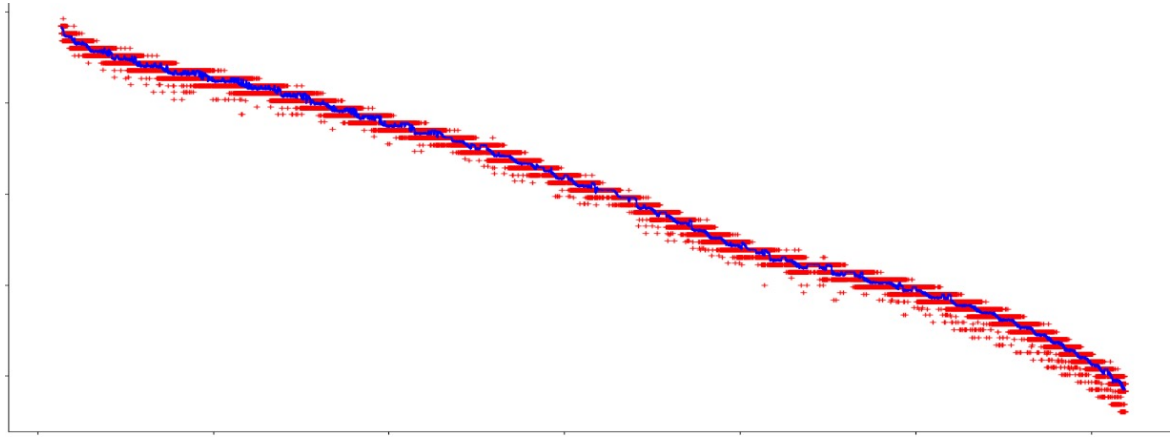


Overfitted

Course of dimensionality

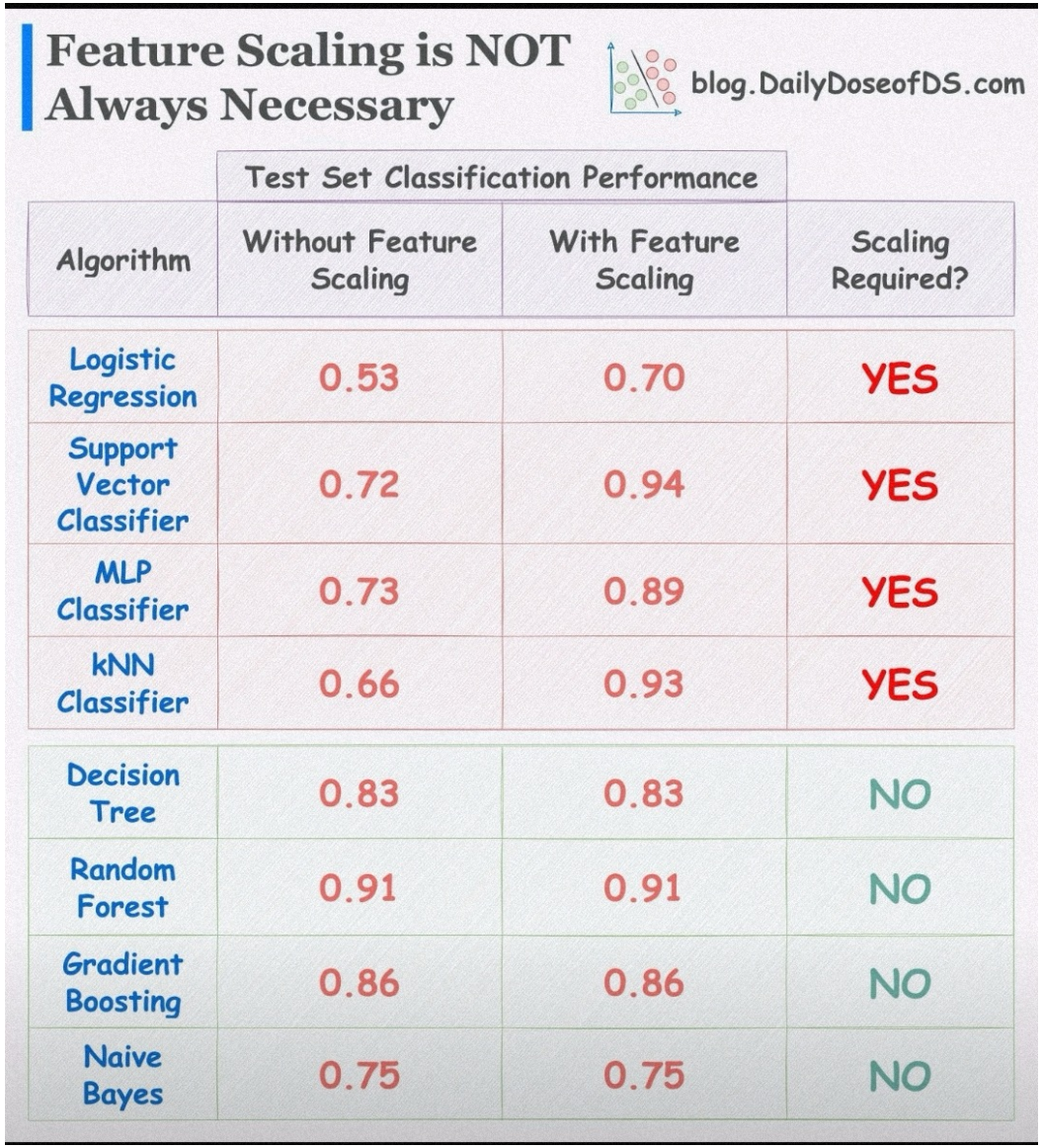


Antes de aplicar reducción de la dimensionalidad



Después de aplicar reducción de la dimensionalidad

¿Usar siempre FS?



Feature Scaling is NOT Always Necessary

blog.DailyDoseofDS.com

Algorithm	Test Set Classification Performance		
	Without Feature Scaling	With Feature Scaling	Scaling Required?
Logistic Regression	0.53	0.70	YES
Support Vector Classifier	0.72	0.94	YES
MLP Classifier	0.73	0.89	YES
kNN Classifier	0.66	0.93	YES
Decision Tree	0.83	0.83	NO
Random Forest	0.91	0.91	NO
Gradient Boosting	0.86	0.86	NO
Naive Bayes	0.75	0.75	NO

Técnicas de feature selection

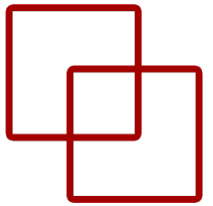
Eliminación Backward

- Procedimiento:
  - Al principio le brindamos todas las características posibles al modelo.
  - Verificamos el rendimiento del modelo y luego eliminamos iterativamente las características de peor rendimiento una por una hasta que el rendimiento general del modelo se encuentre en un rango aceptable.
- La métrica de rendimiento utilizada aquí para evaluar el rendimiento de la característica es *p–value*
  - Si el valor *p* está por encima de 0.05, eliminamos la característica, de lo contrario la conservamos.
- La variable *AGE* e *INDUS* es mayor que 0.05, por tanto, eliminaremos esta característica.

```
X_1 = sm.add_constant(X_reg) #Fitting sm.OLS model
model = sm.OLS(y_reg,X_1).fit()
model.pvalues
```

```
const      3.283438e-12
CRIM       1.086810e-03
ZN         7.781097e-04
INDUS      7.382881e-01
CHAS       1.925030e-03
NOX        4.245644e-06
RM         1.979441e-18
AGE        9.582293e-01
DIS        6.013491e-13
RAD        5.070529e-06
TAX        1.111637e-03
PTRATIO    1.308835e-12
B          5.728592e-04
LSTAT      7.776912e-23
dtype: float64
```

Selección univariable

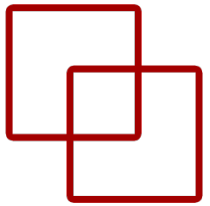


- Las pruebas estadísticas se pueden usar para seleccionar aquellas características que tienen la relación más fuerte con la variable de salida.
- La clase *SelectKBest* que se puede usar con un conjunto de diferentes pruebas estadísticas para seleccionar un número específico de características.
  - En este caso, se utiliza la prueba chi^2
- Con k se seleccionan 4 atributos que serán los que más puntaje tienen: *plas*, *test*, *mass* y *age*.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X_cla, Y_cla)
# summarize scores
np.set_printoptions(precision=3)
print(list(df_cla.columns))
print(fit.scores_)
features = fit.transform(X_cla)
# summarize selected features
print(features[0:5,:])
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
[[148.    0.   33.6  50. ]
 [ 85.    0.   26.6  31. ]
 [183.    0.   23.3  32. ]
 [ 89.   94.   28.1  21. ]
 [137.  168.   43.1  33. ]]
```

Eliminación recursiva de atributos



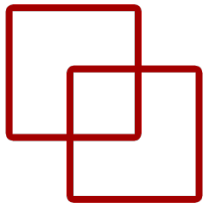
- *RFE* funciona eliminando recursivamente los atributos y construyendo un modelo sobre los atributos que quedan.
- Utiliza la precisión del modelo para identificar qué atributos (y combinación de atributos) contribuyen más a predecir el atributo objetivo.
- En este caso se seleccionan las 3 características principales como *preg*, *plas* y *mass*.
  - Estos están marcados como *True* en la matriz de soporte y marcados con una opción 1 en la matriz de clasificación.
  - Nuevamente, puede asignar manualmente los índices de características a los índices de nombres de atributos.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# feature extraction
model = LogisticRegression(solver="lbfgs", max_iter=200)
rfe = RFE(model, 3)
fit = rfe.fit(X_cla, Y_cla)
print(list(df_cla.columns))
print(f"Num Features: {fit.n_features_}")
print(f"Selected Features: {fit.support_}")
print(f"Feature Ranking: {fit.ranking_}")
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
Num Features: 3
Selected Features: [ True  True False False False  True False False]
Feature Ranking: [1 1 3 6 4 1 5 2]
```

Feature Importance

Decision Trees

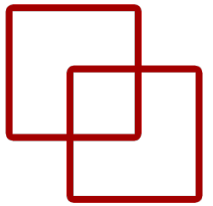


- Los árboles de decisión hacen un particionamiento del espacio de entrada mediante una estrategia voraz.
- Podemos observar que el mejor resultado se obtiene utilizando 6 de las 30 variables originales.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score
# Genera la partición
X_train, X_test, y_train, y_test = train_test_split(X_wisconsin, y_wisconsin, test_size=0.2, random_state=0)
# Aprende el modelo
depth = 3
tree = DecisionTreeRegressor(criterion='mse', max_depth=depth)
tree.fit(X_train, y_train)
# Extrae los índices de las variables utilizadas
subset = np.unique(tree.tree_.feature[tree.tree_.feature>=0])
print("Variables: ", X_wisconsin.shape[1])
print("Variables utilizadas:", subset)
print("Training: ", tree.score(X_train, y_train).round(2))
print('Test: ', tree.score(X_test, y_test).round(2))
print(df_wisconsin.columns[4], df_wisconsin.columns[8], df_wisconsin.columns[12],
      df_wisconsin.columns[23], df_wisconsin.columns[26], df_wisconsin.columns[27])
```

```
Variables:  30
Variables utilizadas: [ 4  8 10 22 23 27]
Training:  0.9
Test:  0.85
mean smoothness mean symmetry perimeter error worst area worst concavity worst concave points
```

Extra Trees



- Se pueden usar árboles de decisión tipo bagged como *Random Forest* y *Extra Trees* para estimar la importancia de las características.
- En el siguiente ejemplo, construimos un clasificador *ExtraTreesClassifier* para el inicio del conjunto de datos de diabetes de Pima Indians.
- Los puntajes sugieren la importancia de *plas, age* y *mass*.

```
from sklearn.ensemble import ExtraTreesClassifier
# feature extraction
model = ExtraTreesClassifier()
model.fit(X_cla, Y_cla)
print(list(df_cla.columns))
print(model.feature_importances_)
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[0.10869558 0.22045197 0.10747431 0.07822648 0.07324594 0.15603371
 0.11483602 0.14103599]
```

4. Random Forest

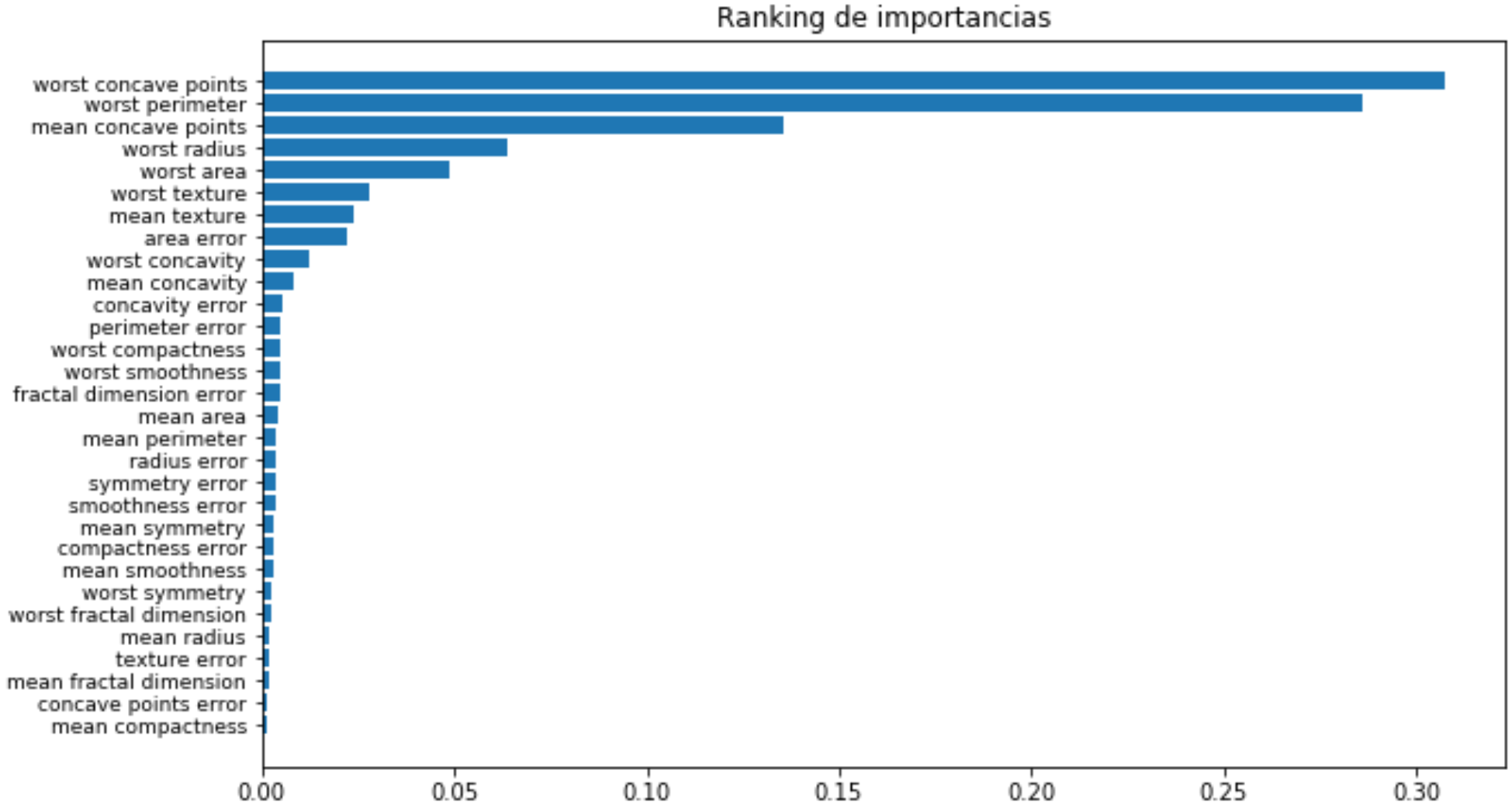
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
# Genera la partición
X_train, X_test, y_train, y_test = train_test_split(X_wisconsin, y_wisconsin, test_size=0.2, random_state=0)
# Entrenamos al algoritmo
forest = RandomForestRegressor(n_estimators=100)
forest.fit(X_train, y_train)
# Extrae las importancias
importances = forest.feature_importances_
y_pred = forest.predict(X_test)
print(f"Error (con todas las variables): {mean_squared_error(y_test, y_pred)}")
```

```
Error (con todas las variables): 0.030361403508771933
```

```
# Extrae los índices ordenados de menor a mayor
ranking = np.argsort(forest.feature_importances_)
print(wisconsin.feature_names[ranking])
```

```
['mean compactness' 'concave points error' 'mean fractal dimension'
 'texture error' 'mean radius' 'worst fractal dimension' 'worst symmetry'
 'mean smoothness' 'compactness error' 'mean symmetry' 'smoothness error'
 'symmetry error' 'radius error' 'mean perimeter' 'mean area'
 'fractal dimension error' 'worst smoothness' 'worst compactness'
 'perimeter error' 'concavity error' 'mean concavity' 'worst concavity'
 'area error' 'mean texture' 'worst texture' 'worst area' 'worst radius'
 'mean concave points' 'worst perimeter' 'worst concave points']
```

4. Random Forest

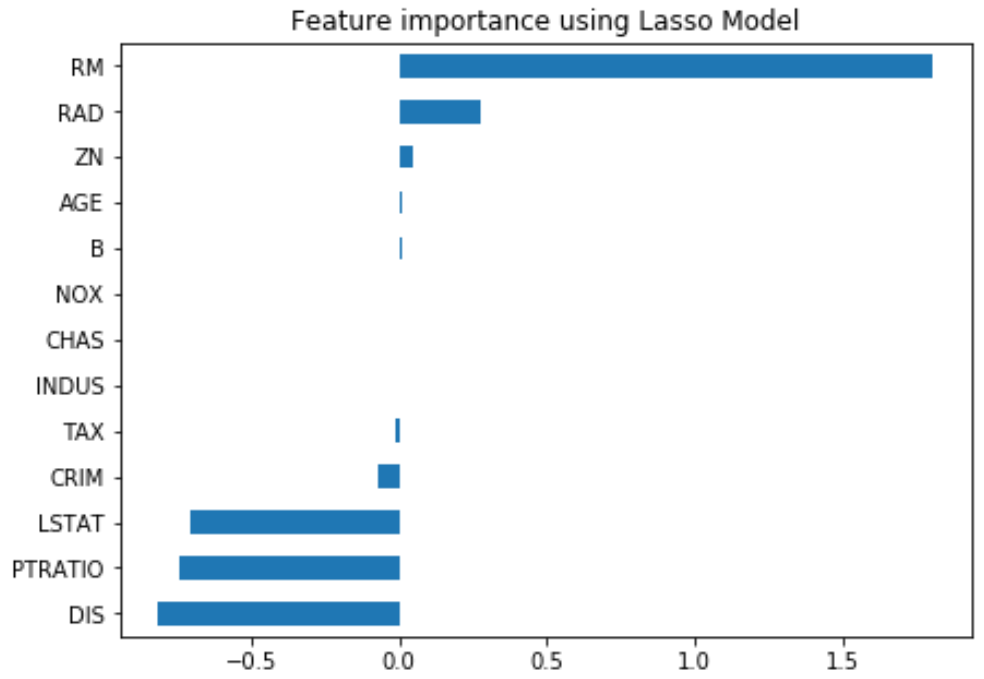


5. LASSO

- En este caso el modelo LASSO ha tomado todas las características **excepto** *NOX*, *CHAS* e *INDUS*.
- Así mismo podemos utilizar otros modelos como RIDGE o Linear Regression por ejemplo.

```
# Future imoprtance - LASSO
from sklearn.linear_model import LassoCV
# feature importance
reg = LassoCV()
reg.fit(X_reg, y_reg)
print("Best alpha using built-in LassoCV: %f" % reg.alpha_)
print("Best score using built-in LassoCV: %f" %reg.score(X_reg,y_reg))
coef = pd.Series(reg.coef_, index = X_reg.columns)
# features picked and eliminated
print("Lasso picked " + str(sum(coef != 0)) + " variables and eliminated the other " +
      str(sum(coef == 0)) + " variables")
# Show de coeficient
imp_coef = coef.sort_values()
plt.rcParams['figure.figsize'] = (7.0, 5.0)
imp_coef.plot(kind = "barh")
plt.title("Feature importance using Lasso Model")
```

```
Best alpha using built-in LassoCV: 0.724820
Best score using built-in LassoCV: 0.702444
Lasso picked 10 variables and eliminated the other 3 variables
```



¡GRACIAS!

Dr. Manuel Castillo-Cara
www.manuelcastillo.eu
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Educación a Distancia (UNED)