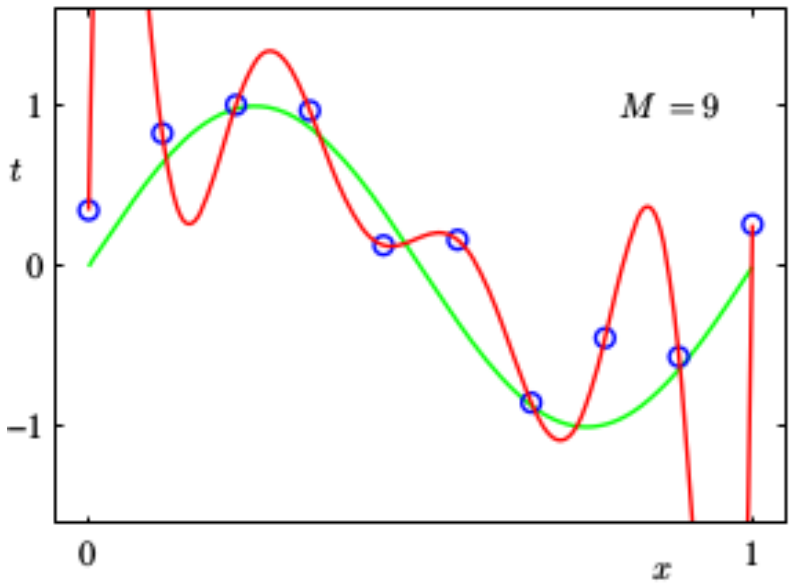
Selección del modelo y su evaluación

July 28, 2018

Outline

Introducción

Supongamos que hemos ajustado de manera satisfactoria un modelo de machine learning. Sin embargo, vemos que sobre nuevos datos, los errores en las predicciones son muy grandes.



generalización

Nuestros modelo tiene que ser generalizables a nuevo datos

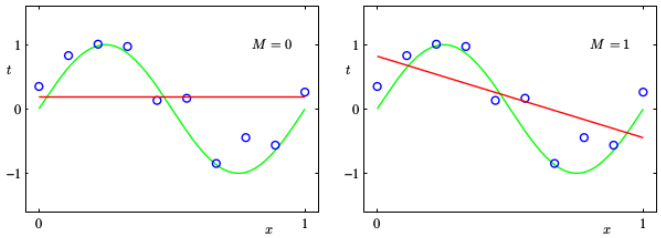
- La generalización es uno de los conceptos claves en machine learning.
- La busqueda de dicha generalización guia la elección del algoritmo.
- Proporciona una medida de la cualidad del último modelo escogido
- Tenemos que buscar una medida, de tal manera que reduzcamos el error cuando intentamos generalizar los resultados. El error se suele medir de la siguiente manera:

$$Error = 1 - ACC \text{ (clasificación)} \tag{1}$$

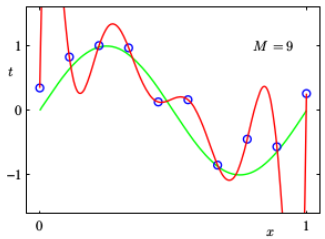
$$Error = RSS \text{ (regresión)} \tag{2}$$

Bias y Variance

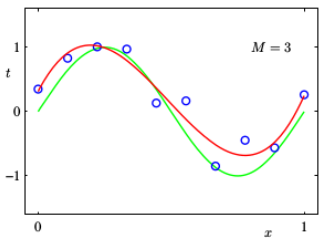
- Pocas potencias (variables) hace que no ajusten bien la curva a las observaciones. Se dice que en este caso, el modelo sufre de mucho **BIAS**



- Muchas potencias (variables) hacen la curva se ajuste demasiado bien a los puntos, de manera muy compleja. Se dice que en este caso, el modelo sufre de mucho **OVERFITTING**



- Lo ideal es siempre encontrar un equilibrio entre bias y overfitting



Método Holdout

Objetivos

- Estimar la accuracy de generalización, esto es, el rendimiento predictivo de nuestro modelo en una futura data.
- Aumentar el poder predictivo de nuestro modelo ajustando sus hiperparámetros y seleccionando aquellos que den mejor rendimiento.
- Elegir el mejor modelo para el problema en estudio. Por ello, nos interesa comparar el rendimiento de diferentes algoritmos y seleccionar aquél con mejor comportamiento

Todo esto se puede llevar a cabo aplicando el método **holdout**.

Holdout method

- Dividimos nuestros datos en dos datasets: *training* y *holdout* data.
- Usamos el training data para ajustar el modelo. El error cometido sobre esta dataset se puede escribir como

$$Err^{train} = \sum_{i=1}^{N_{train}} (y_i^{pred} - y_i^{teor})^2 \tag{3}$$

$$Err^{train} = 1 - ACC^{train} \tag{4}$$

- Usamos el holdout data para generalizar el modelo (En estos casos, se conoce como **test set**). El error cometido sobre esta dataset se puede escribir como

$$Err^{holdout} = \sum_{i=1}^{N_{holdout}} (y_i^{pred} - y_i^{teor})^2 \tag{5}$$

$$Err^{holdout} = 1 - ACC^{holdout} \tag{6}$$

- Normalmente $Err^{holdout} > Err^{training}$
  En scikit, ya hemos visto que esto se puede hacer mediante **model_ selection.train_ test_ split**

Estratificación

Cuando partimos los datos, pueden ocurrir dos situaciones que pueden empeorar los resultados dados por el clasificador:

- Aunque nuestra data está balanceada, la partición, al ser aleatoria, coloca más datos de una clase que de otra en el training set
- Si la data no está balanceada, seguramente el training set estará formando por la clase de mayor presencia, por lo que el clasificador "aprenderá" sólo esta clase
- En ambos casos, las predicciones pueden verse afectadas negativamente.
- Por ello, es aconsejable que la partición se realice de manera estratificada, es decir, manteniendo la proporción entre clases

En scikit, esto se puede lograr mediante **model_ selection.StratifiedShuffleSplit**

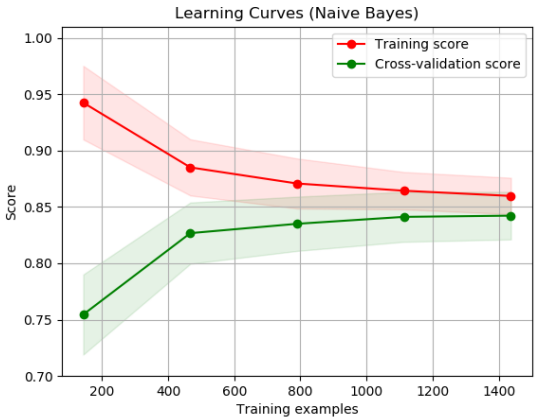
Otros usos de este método es el de ayudarnos a diagnosticar el rendimiento de nuestro algoritmo en diferentes situaciones:

- Añadir más observaciones para el training
- Intentar menos features
- Añadir nuevas features
- Hacer transformaciones sobre las features que tenemos
- Cambiar el parámetro de regularización $(C, \lambda, \cdots)$

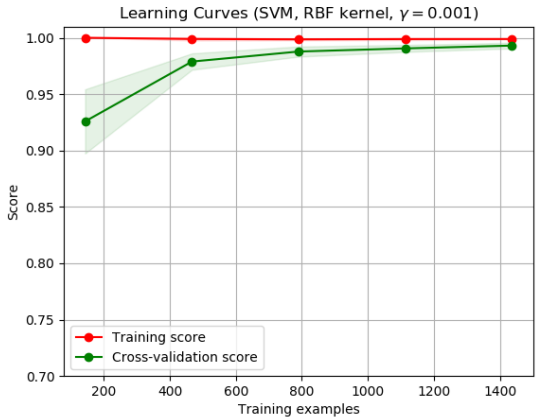
Diagnóstico 1: tamaño de la partición

Podemos ver cómo cambia el rendimiento del clasificador variando el número de observaciones en el training set. Nos permite saber si nuestro clasificador sufre exceso de bias u overfitting.

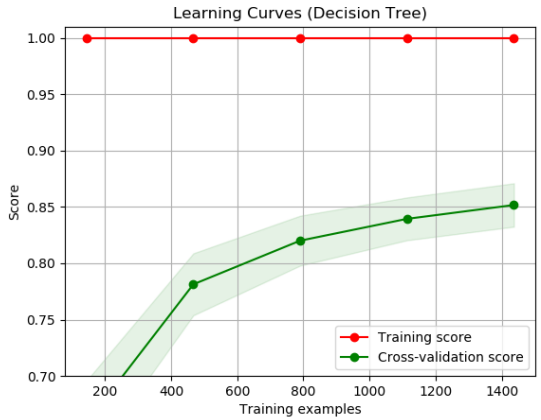
HIGH BIAS



OPTIMAL CASE



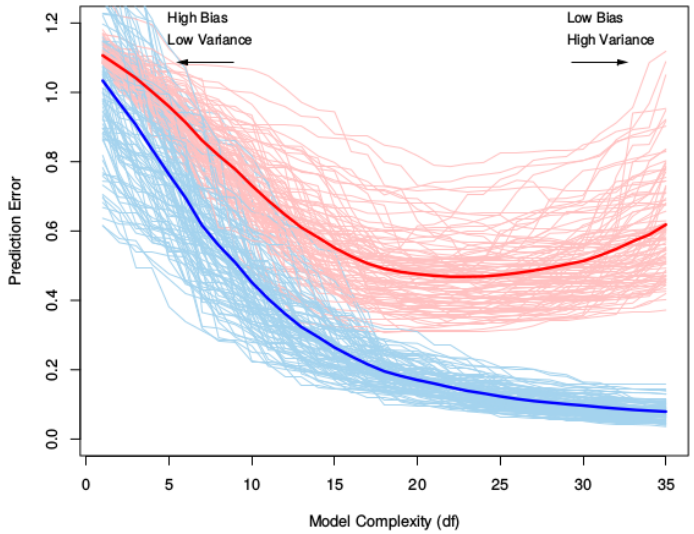
HIGH OVERFITTING



En scikit, esto se puede mirar mediante el uso de la función **model_ selection.validation_ curves**

Diagnóstico 2: Complejidad del modelo

- El error en el training set disminuye a medida que hacemos más complejo el modelo.
- Complejidad pequeña puede dar un modelo muy rígido. Sufre de mucho bias.
- En estos casos, el aumento de la complejidad, mejora la generalización.
- Demasiada complejidad hace que el error en el test set se dispare. Se empieza a sufrir de de overfitting.



En scikit, esto se puede mirar mediante el uso de la función **model_selection.learning_curves**

Métodos de evaluación

- Uno de los problemas del método anterior de coger una partición de los datos en training y test set es que los resultados dependen de la partición en particular.
- Esto es especialmente sensible cuando el tamaño de nuestra dataset es pequeña.
- Debemos por tanto comprobar la estabilidad de nuestros resultados para diferentes particiones.
- Para estos casos, las técnicas más usads son Bootstrap y cross validation.

Método Bootstrap

La sensibilidad de los resultados a los diferentes datos que pueda haber en el training set se reduce mediante el promedio sobre muchas muestras extraídas de la data set original

- Dada una dataset de tamaño N y un numero B de boostraps a crear.
- Para cada b boostrap
  - Extraer una observación con reemplazamiento de la dataset completa y asignarla al bootstrap sample
  - Repetir esto hasta que el bootstrap sample tenga el mismo tamaño que nuestra dataset original
- Ajustar el modelo para cada bootrsap sample b y calcular su accuracy sobre la data original, que aquí actua como test set
- Compuar el accuracy del modelo promediando sobre las accuracy de los bootstrap samples

$$Error_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i^{teor}, y_i^{pred}) \tag{7}$$

Método Bootstrap

- El problema de hacer esto es que los bootstrap sets y la data original no son independientes. Esto tiende a generar resultados muy optimistas.
- Para evitar esto, es mejor realizar las predicciones sobre aquellos ejemplos que no están en el bootstrap set

$$Err_{boot}(1) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i^{teor}, y_i^{pred}) \tag{8}$$

- La probabilidad de una observación i de estar dentro del bootstrap sample es

$$P(i \in b) = 1 - (1 - 1/N)^N \approx 0.632 \tag{9}$$

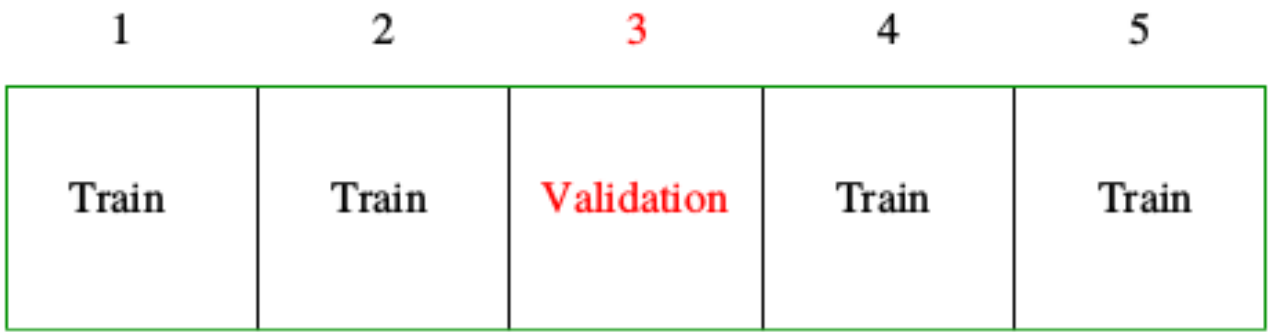
- De esta forma, podemos estimar el error como

$$Err_{.632} = 0.368 err_{train} + 0.632 Err_{boot} \tag{10}$$

En scikit esto se puede calcular esto mediante de la clase **ensemble.BaggingClassifier**

Cross-validation

- Para evitar resultados demasiado optimistas, cross-validation divide los datos en varios trozos o *fold*, de tal forma que cada trozo sirva una vez como test set y el resto como training.
- De esta forma, aseguramos que tanto el test set como training set sea independenientes.

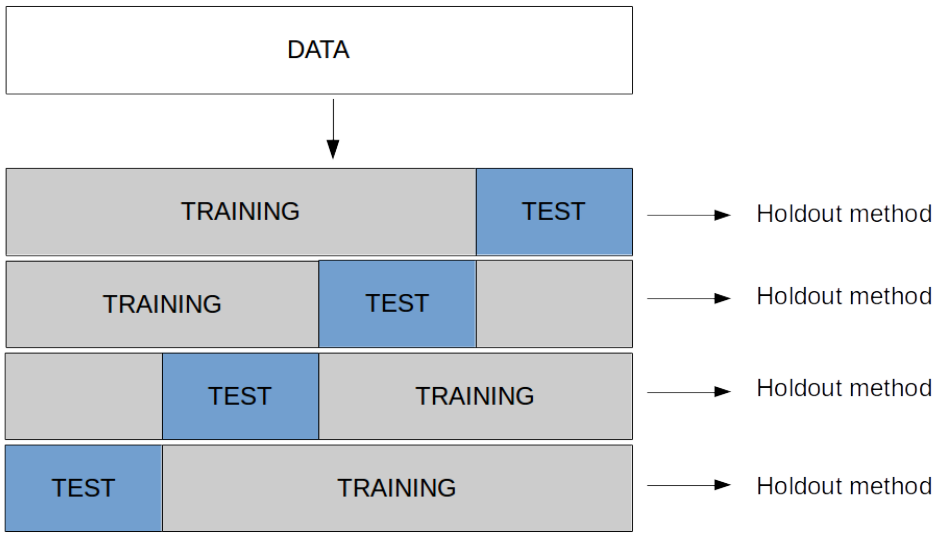


- Los resultados son promediados a lo largo de los folds

En scikit, dada una configuración del número de folds, las predicciones se realizan mediante: **model_ selection.cross_ val_ score, model_ selection.cross_ val_ predict**

K-Fold cross-validation

- Se particionan los datos enteros en K-folds del mismo tamaño más o menos.
- K-1 se usan como training y el restante como validation
- Puede ser usado para evaluación del modelo y/o selección del modelo



K-Fold cross-validation

Qué valor de K debemos escoger?

- El bias decrece con K, ya que el training set es más grande.
- La varianza aumenta con K, ya que el test set es cada vez más pequeño.
- El coste computacional aumenta, ya que se hacen más fittings y los training sets son más grandes.
- Valores pequeños de K en datasets pequeños también aumenta la variabilidad debido a efectos aleatorios del sampling

Normalmente una elección de K= 5 o 10 es lo recomendado.

K-Fold cross-validation

En scikit, esto se puede realizar mediante las siguientes clases:

- **model_ selection.KFold** genera K-folds.
- **model_ selection.StratifiedKFold** K-folds manteniendo la proporción de clases.
- **model_ selection.LeaveOneOut** genera tantos folds como observaciones haya.
- **model_ LeavePOut** genera todos las posibles particiones eliminando p obersvaciones de los datos.
- **model_ selection.LeaveOneGroupOut**
- **model_ selection.LeavePGroupsOut**

Extensión: Repeated K-Fold cross-validation

- K-fold cross-validition sigue sufriendo de variabilidad debido a la partición aleatoria de los datos en K Folds
- Para reducir esto, se puede repetir varias veces el K-Fold cross-validation.
- Las predicciones son por tanto promediadas a lo largo de las repeticiones y folds.

En scikit **sklearn.model_ selection.RepeatedKFold,sklearn.model_ selection.RepeatedStratifiedKFold**

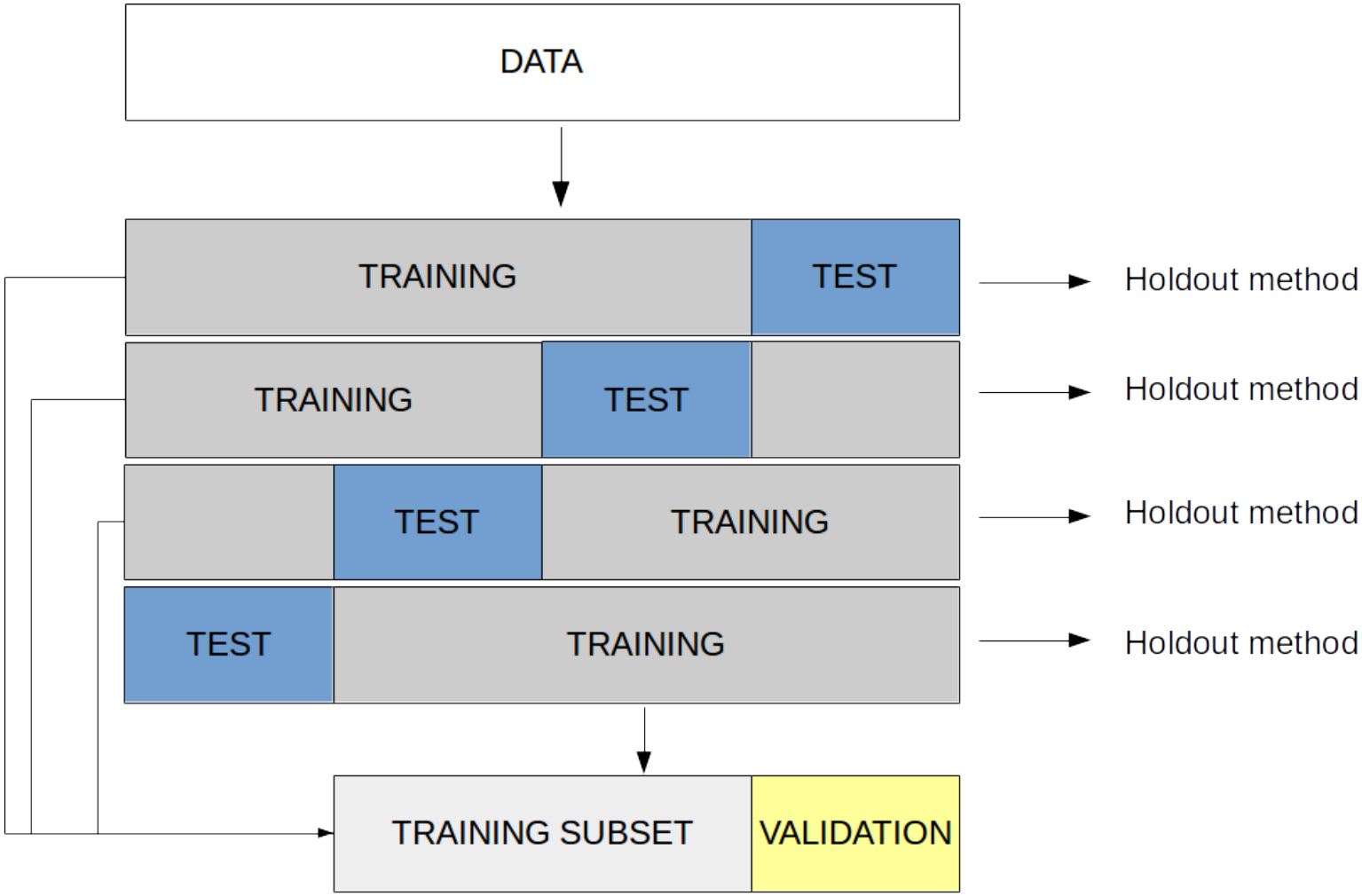
Selección de modelo

Cross-validation y selección de modelo

- Todo lo dicho anteriormente tenía que ver con cómo producíamos resultados fiables y generalizables.
- Como hemos visto, estos resultados dependen también de los parámetros de nuestro algoritmo.
- Necesitamos introducir un nuevo paso que nos permita optimizarlos.
- Cross-validation dividiendo los datos en tres: Training set para ajustar un modelo determinado, validation set para seleccionar el modelo adecuado y test set para medir la generalización del modelo.

Cross-validation y selección de modelo

En este caso, dentro del ajuste del modelo añadimos otro paso de optimización del modelo.



Cross-validation y selección de modelo

En scikit, la busqueda consiste en

- Un clasificador o regresor
- Un espacio de hiperparámetros, definido como un diccionario o una lista de diccionarios.
- Un método para buscar candidatos
- Un esquema de cross-validation, como definidas anteriormenente
- Una función *score* que queremos optimizar

Cross-validation y selección de modelo

Según el metodo de búsqueda de hiperparámetros, podemos encontar en scikit las dos siguientes opciones

- **model_ selection.GridSearchCV**, donde se define un grid sobre el que hacer la búsqueda de hiperparámetros. Por ejemplo [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},]
- **model_ selection.RandomizedSearchCV**, donde la busqueda se hace de manera aleatoria seleccionando el valor de los hiperparámetros definiedo una distribución de cada uno. Por ejemplo [{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),'kernel': ['rbf'], 'class_ weight':['auto', None]}]

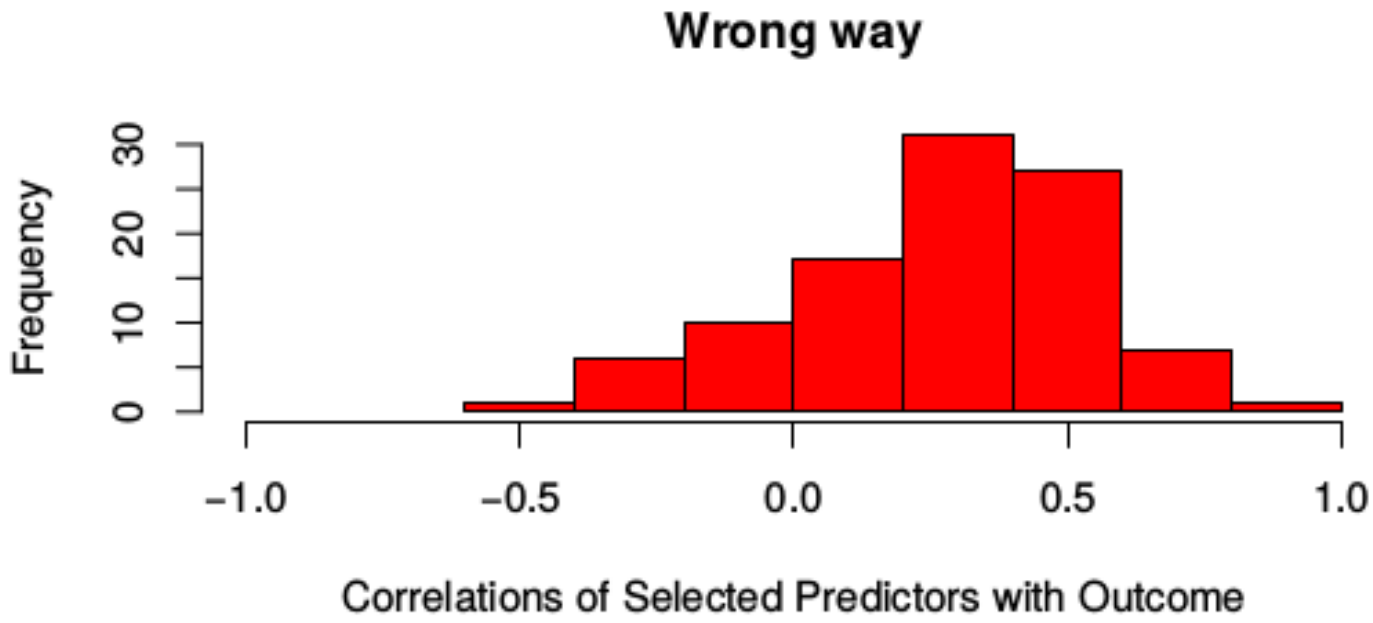
Cross-validation y selección de modelo

A su vez, scikit implementa algunos algoritmos que incorpora un proceso de cross-validation para buscar los hiperparámetros óptimos

- Ridge regression **linear_ model.RidgeCV**
- Lasso regression **linear_ model.LassoCV**
- ElasticNet regresssion**linear_ model.ElasticNetCV**
- Logistic regressio clasifiación **linear_ model.LogisticRegressionCV**

Precaución: Manera INCORRECTA CV

1. Filtrar aquellas features que muestran una correlación fuerte con el target.
2. Usando este subset, seleccionar un clasificador particular.
3. Usar cross-validation para estimar los hiperparámetros y el error en la predicción del modelo final



Precaución: Manera CORRECTA CV

1. Usar cross-validation para dividir los datos en K folds.
2. Para cada fold, encontrar aquellos features con mayor correlación con el target usando todas las observaciones en el **training** set.
3. Usando el subset encontrado, ajustar un clasificador particular en el **training** set.
4. Usar el clasificador para predecir el error en la predicción en el test set