eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Programa de doctorado en investigación biomédica

MDe

Master eta Doktorego Eskola

Escuela de Máster y Doctorado

Master and Doctoral School

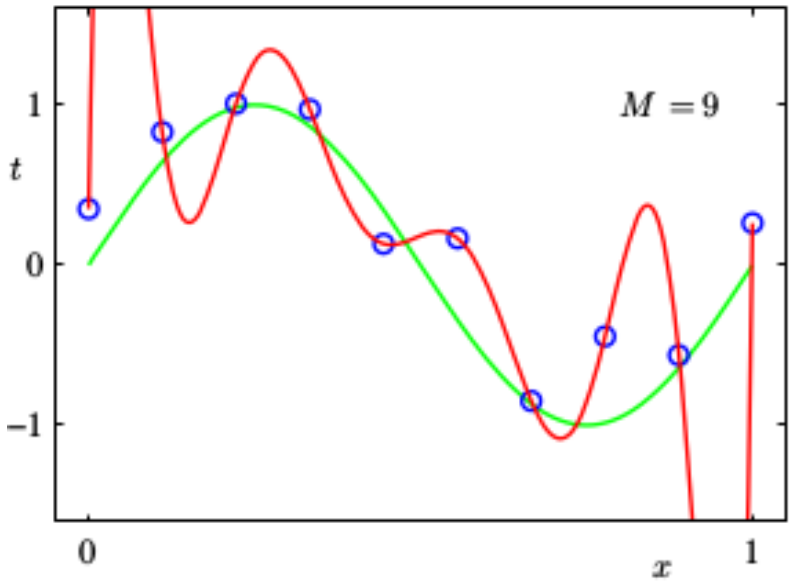
Selección del modelo e hiperparámetros y su evaluación

3rd April 2019

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

Nuestro modelo tiene que ser generalizables a nuevo datos

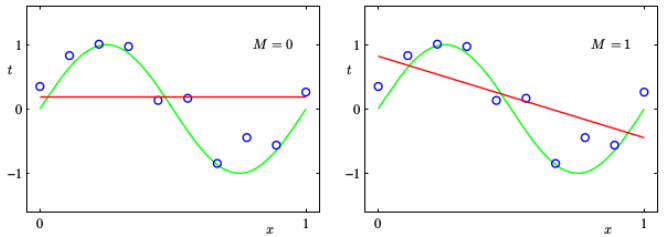
- La generalización es uno de los conceptos claves en machine learning.
- La busqueda de dicha generalización guia la elección del algoritmo.
- Proporciona una medida de la calidad del modelo escogido finalmente.
- Tenemos que buscar una medida del error, de modo que lo mantegamos bajo cuando intentemos generalizar los resultados. El error se suele medir de la siguiente manera:

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

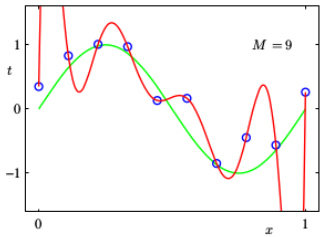
$$Error = RSS \text{ (regresión)} \tag{2}$$

Todo gira en torno al Bias y Variance

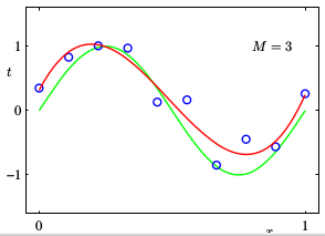
- Pocas potencias (variables) hace que la curva no se ajusten bien a las observaciones. Se dice que en este caso, el modelo sufre de mucho **BIAS**



- Muchas potencias (variables) hacen que la curva se ajuste demasiado bien a los puntos, de manera muy compleja pero nada generalizable. 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 el error en la generalización, esto es, el rendimiento predictivo de nuestro modelo en una futura data no vista aún.
- Aumentar el poder predictivo de nuestro modelo ajustando sus parámetros (aka **hiperparámetros**) y seleccionando aquéllos que den mejor rendimiento.
- Elegir el mejor modelo para el problema en estudio. Para 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**.

Método Holdout

- 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

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

$$Err^{train} = 1 - ACC^{train} \qquad (clasificación) \tag{4}$$

Método Holdout

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

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

$$Err^{holdout} = 1 - ACC^{holdout} \qquad (clasificación) \tag{6}$$

- Normalmente $Err^{holdout} > Err^{training}$

En scikit, ya hemos visto que esto se puede hacer mediante **model_selection.train_test_split**

Holdout Estratificado

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

- Aunque nuestra data esté 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** (también en **model_selection.train_test_split** automáticamente)

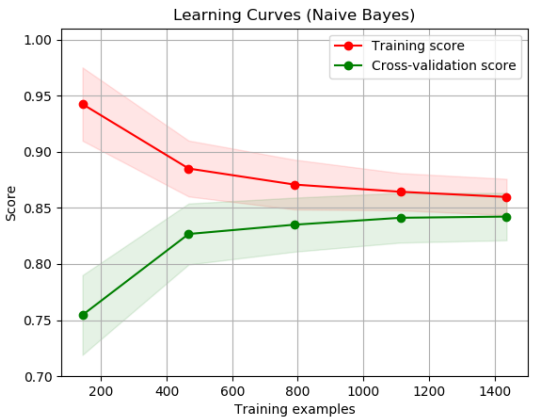
Otros usos de este método es el de ayudarnos a diagnosticar problemas en el rendimiento de nuestro algoritmo, que pueden requerir las siguientes soluciones:

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

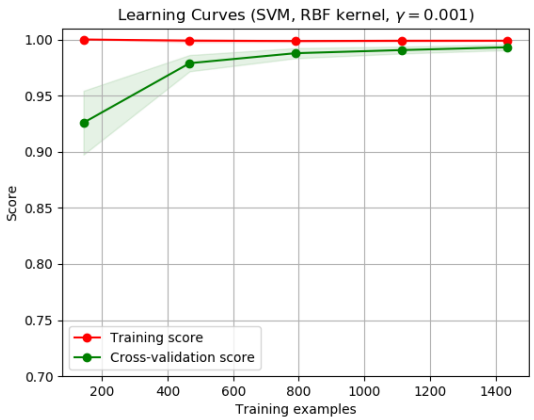
Diagnóstico 1: tamaño de la partición

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

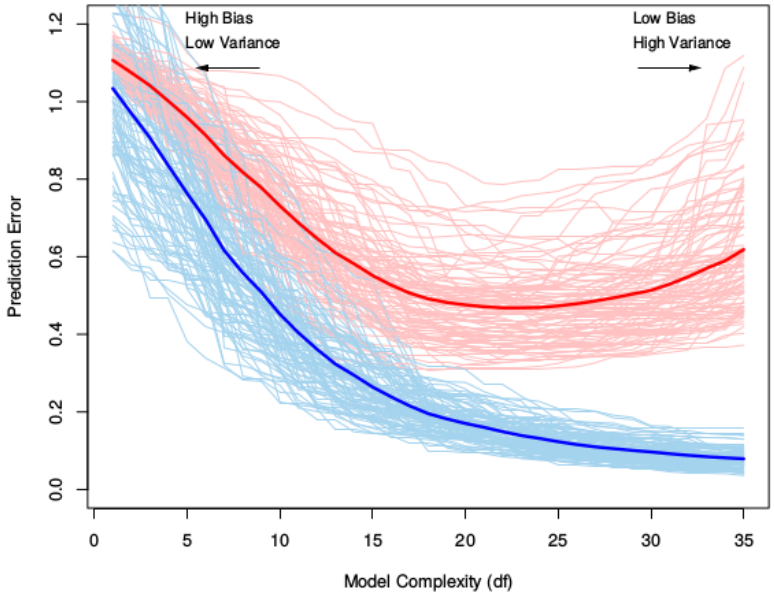


HIGH OVERFITTING

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 holdout 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**

Evaluación de un modelo: Resampling methods

- Uno de los problemas del método anterior de coger una partición de los datos en training y holdout 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.
- Esto ya se aplica hoy en día de manera estándar en problemas de machine learning, mediante las técnicas de Bootstrap (resampling con reemplazamiento) y cross-validation (resampling sin reemplazamiento).

Estimación del error mediante 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 dataset 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

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

"Leave-one-out" Bootstrap

- El problema del anterior método es que los bootstrap sets y la data original no son independientes. Esto tiende a generar resultados muy optimistas.
- Para evitar esto, "Leave-one-out" Bootstrap: realizar las predicciones sobre aquellos ejemplos que no están en el bootstrap set

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

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

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

- Es como si en promedio hiciéramos el training sobre la mita de los datos (luego 2-Fold CV). Este tamaño de los datos suele **sufrir** de **bias** y por tanto sobreestimar el error.

.632 y .632+ Bootstrap

- Ambos se basan en corregir este exceso de bias mediante la inclusión de un peso en los errores estimados.
- .632 Bootstrap corrige mediante un peso constante (Efron and Tibshirani, 1993)

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

- .632+ Bootstrap usa un peso que depende del nivel de overfitting (Efron and Tibshirani, 1997)

$$Err_{.632+} = (1 - w) \cdot err_{train} + w \cdot Err_{boot} \tag{11}$$

donde

$$w = \frac{0.632}{1 - 0.368R} \qquad R = \frac{Err_{boot} - err_{train}}{\gamma - err_{train}} \tag{12}$$

y γ es lo que se conoce como no-information error rate.

En scikit este método se puede implementar usando **ensemble.BaggingClassifier**

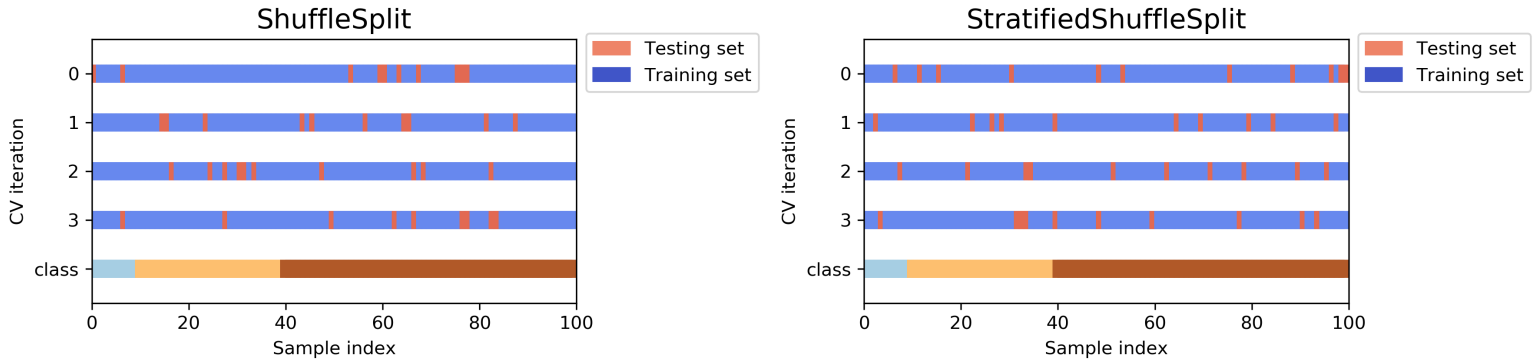
Cross-validation

- Para evitar resultados que dependan de la partición realizada en el método holdout, cross-validation genera diferentes particiones.
- En cada partición, a diferencia de bootstrap, se samplea sin reemplazamiento.
- En cada partición, una parte se usa para ajustar el modelo (training set) y en la otra, para hacer predicciones (test o validation set).
- Los resultados son promediados a lo largo de cada partición.

En scikit, dada una estrategia de cross-validation, las predicciones se realizan mediante: **model_selection.cross_val_score, model_selection.cross_val_predict**

Monte Carlo Cross-validation

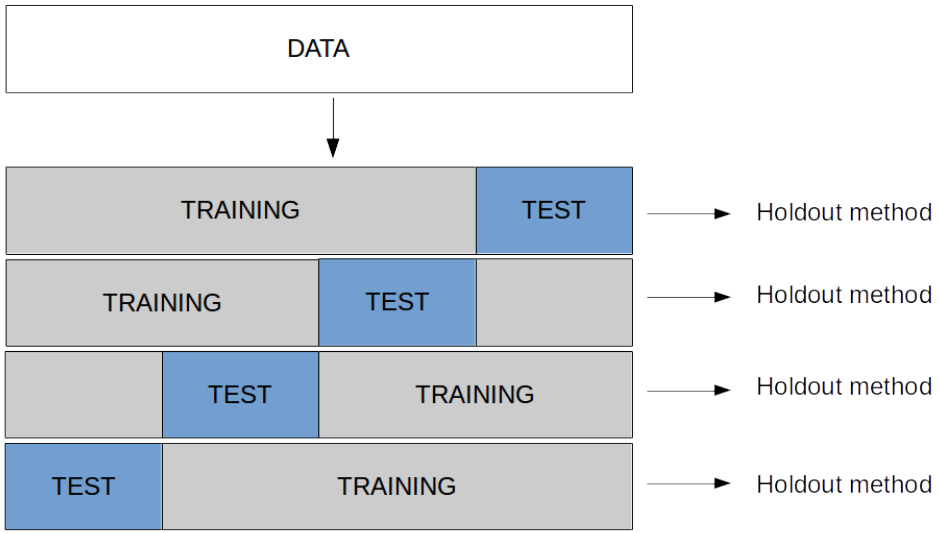
- Se generan diferentes particiones y se aplica el método holdout en cada una de ellas.
- De esta manera, se pierde la variabilidad por la partición específica usada mediante el método holdout.
- Suele sufrir de bias, ya que puede ocurrir que no todos los puntos se usen para el ajuste. Ocurre sobre todo en datasets grandes.



En scikit, se puede realizar mediante **model_selection.ShuffleSplit** y **model_selection.StratifiedShuffleSplit**

K-Fold cross-validation

- Se usa una sola partición de lo datos.
- Para dicha partición, los datos se dividen en K-folds del mismo tamaño más o menos.
- K-1 se usan como training y el restante como holdout set.
- Puede ser usado para evaluación del modelo (holdout set $\equiv$ test set) y/o selección del modelo (holdout set $\equiv$ validation set).



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 holdout 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 aumentan 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_selection.LeavePOut** genera todos las posibles particiones eliminando p observaciones de los datos.
- **model_selection.LeaveOneGroupOut**
- **model_selection.LeavePGroupsOut**

Extensión: Repeated K-Fold cross-validation

- K-fold cross-validation 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**

¿Qué método escoger para estimar el error?

BIOINFORMATICS ORIGINAL PAPER Vol. 21 no. 15 2005, pages 3301–3307
doi:10.1093/bioinformatics/bti499

Data and text mining

Prediction error estimation: a comparison of resampling methods

Annette M. Molinaro[1,3,*], Richard Simon[2] and Ruth M. Pfeiffer[1]

[1]Biostatistics Branch, Division of Cancer Epidemiology and Genetics and [2]Biometric Research Branch, Division of Cancer Treatment and Diagnostics, NCI, NIH, Rockville, MD 20852 USA and [3]Department of Epidemiology and Public Health, Yale University School of Medicine, New Haven, CT 06520, USA

Received on April 6, 2005; revised on April 28, 2005; accepted on May 12, 2005
Advance Access publication May 19, 2005

ABSTRACT

Motivation: In genomic studies, thousands of features are collected on relatively few samples. One of the goals of these studies is to build classifiers to predict the outcome of future observations. There are three inherent steps to this process: feature selection, model selection and prediction assessment. With a focus on prediction assessment, we compare several methods for estimating the 'true' prediction error of a prediction model in the presence of feature selection.

Results: For small studies where features are selected from thousands of candidates, the resubstitution and simple split-sample estimates are seriously biased. In these small samples, leave-one-out cross-validation (LOOCV), 10-fold cross-validation (CV) and the .632+ bootstrap have the smallest bias for diagonal discriminant analysis, nearest neighbor and classification trees. LOOCV and 10-fold CV have the smallest bias for linear discriminant analysis. Additionally, LOOCV, 5- and 10-fold CV, and the .632+ bootstrap have the lowest mean square error. The .632+ bootstrap is quite biased in small sample sizes with strong signal-to-noise ratios. Differences in performance among resampling methods are reduced as the number of specimens available increase.

Including too many noisy variables reduces accuracy of the prediction and may lead to over-fitting of data, resulting in promising but often non-reproducible results (Ransohoff, 2004).

Another difficulty is model selection with numerous classification models available. An important step in reporting results is assessing the chosen model's error rate, or generalizability. In the absence of independent validation data, a common approach to estimating predictive accuracy is based on some form of resampling the original data, e.g. cross-validation. These techniques divide the data into a learning set and a test set, and range in complexity from the popular learning-test split to v-fold cross-validation, Monte-Carlo v-fold cross-validation and bootstrap resampling. Few comparisons of standard resampling methods have been performed to date, and all of them exhibit limitations that make their conclusions inapplicable to most genomic settings. Early comparisons of resampling techniques in the literature are focussed on model selection as opposed to prediction error estimation (Breiman and Spector, 1992; Burman, 1989). In two recent assessments of resampling techniques for error estimation (Braga-Neto and Dougherty, 2004; Efron, 2004), feature selection was not included as part of the resampling procedures, causing the

¿Qué método escoger?

- Al final, todos los métodos tienden a solaparse entre ellos y a medida que aumenta el tamaño de los datos, las diferencias entre ellos disminuyen.
- 0.632 y 0.632+ Bootstrap permiten realizar muchas iteraciones, útil cuando el tamaño de los datos son pequeños, pero tienden a generar resultados que están biased.
- K-Fold Cross-validation predice los puntos sólo una vez, por lo que parece más justo, aunque no explora diferentes particiones, por lo que los resultados pueden ser variables.
- Monte Carlo cross-validation permite explorar gran infinidad de particiones, pero tiende a estar bias porque puede ocurrir que una observación no haya caído nunca en el training y holdout set.
- Yo normalmente uso K-Fold Cross-validation con diferentes particiones, ya que supone un compromiso entre todos los métodos.

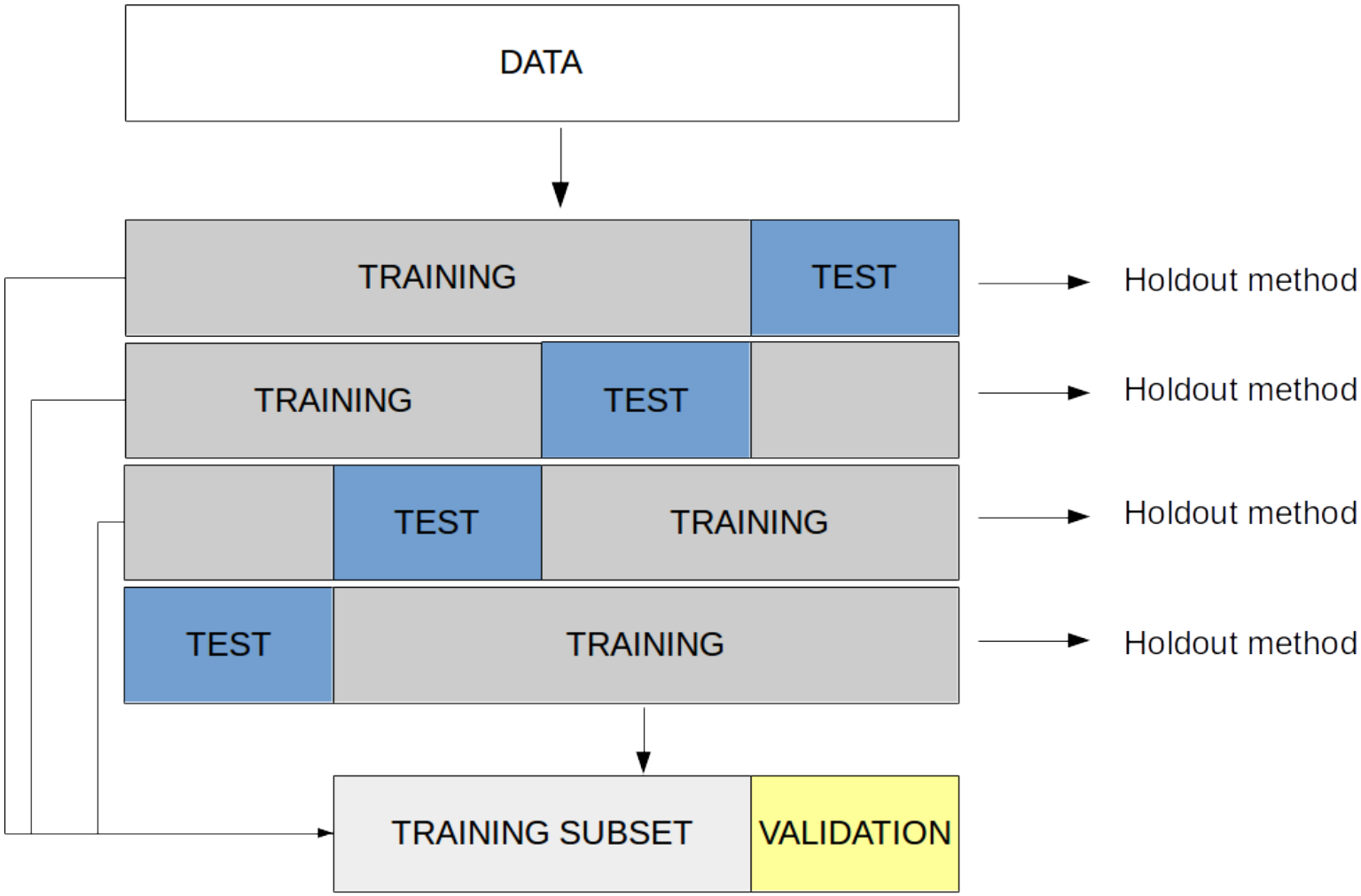
Selección de modelo

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

Cross-validation y selección de modelo

Según el metodo de búsqueda de hiperparámetros, podemos encontrar 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 incorporan un proceso de cross-validation interno para buscar los hiperparámetros óptimos cuando se llama al método *fit*:

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

Precaución 1: Feature selection

La selección de las features más predictivas se tiene que ver como un paso más dentro de la optimización del modelo y por tanto, se tiene que realizar exclusivamente sobre el training set.

Precaución 1: Feature selection

Error cometido incluso en revistas de alto impacto.

Selection bias in gene extraction on the basis of microarray gene-expression data

Christophe Ambroise[†] and Geoffrey J. McLachlan[‡§]

[†]Laboratoire Heudiasyc, Unité Mixte de Recherche/Centre National de la Recherche Scientifique 6599, 60200 Compiègne, France; and [‡]Department of Mathematics, University of Queensland, Brisbane 4072, Australia

Edited by Stephen E. Fienberg, Carnegie Mellon University, Pittsburgh, PA, and approved March 21, 2002 (received for review February 20, 2002)

In the context of cancer diagnosis and treatment, we consider the problem of constructing an accurate prediction rule on the basis of a relatively small number of tumor tissue samples of known type containing the expression data on very many (possibly thousands) genes. Recently, results have been presented in the literature suggesting that it is possible to construct a prediction rule from only a few genes such that it has a negligible prediction error rate. However, in these results the test error or the leave-one-out cross-validated error is calculated without allowance for the selection bias. There is no allowance because the rule is either tested on tissue samples that were used in the first instance to select the genes being used in the rule or because the cross-validation of the rule is not external to the selection process; that is, gene selection is not performed in training the rule at each stage of the cross-validation process. We describe how in practice the selection bias can be assessed and corrected for by either performing a cross-validation or applying the bootstrap external to the selection process. We recommend using 10-fold rather than leave-one-out cross-validation, and concerning the bootstrap, we suggest using the so-called .632+ bootstrap error estimate designed to handle overfitted prediction rules. Using two published data sets, we demonstrate that when correction is made for the selection bias, the cross-validated error is no longer zero for a subset of only a few genes.

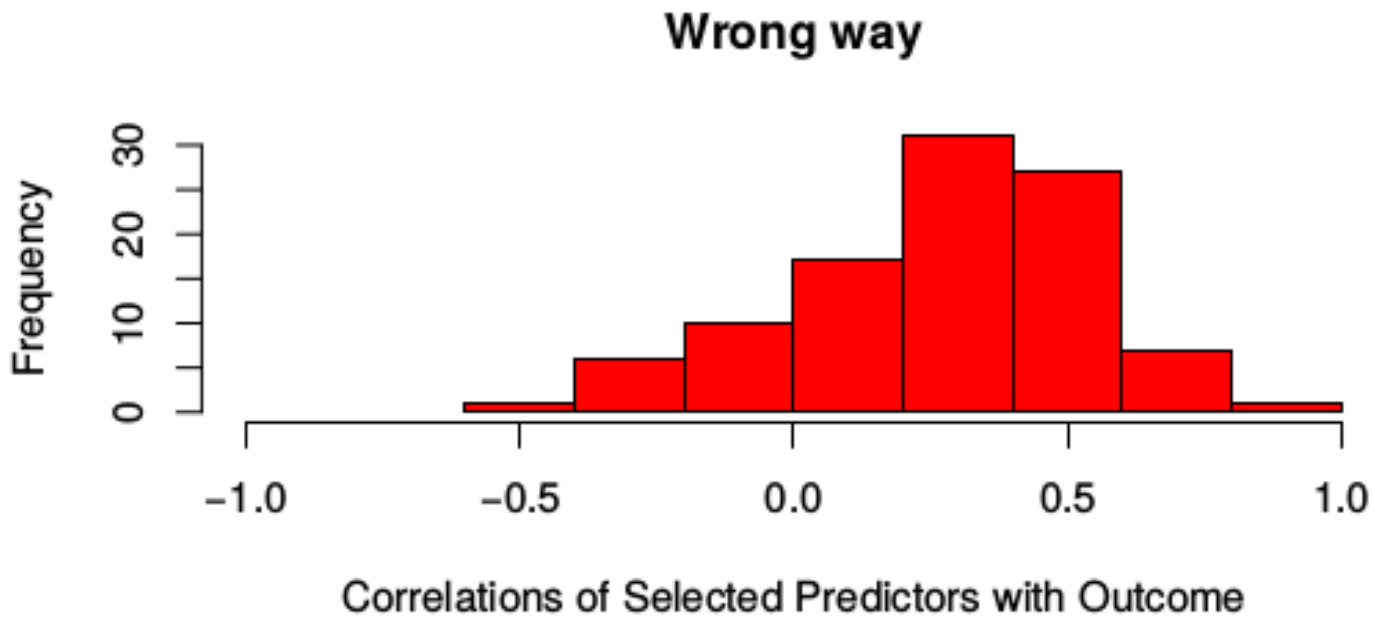
Fisher's linear discriminant function is singular if $n < g + p$. Second, even if all the genes can be used as, say, with a Euclidean-based rule or a support vector machine (SVM; refs. 9, 16, and 17), the use of all the genes allows the noise associated with genes of little or no discriminatory power, which inhibits and degrades the performance of the rule R in its application to unclassified tumors. That is, although the apparent error rate (AE) of the rule R (the proportion of the training tissues misallocated by R) will decrease as it is formed from more and more genes, its error rate in classifying tissues outside of the training set eventually will increase. That is, the generalization error of R will be increased if it is formed from a sufficiently large number of genes. Hence, in practice consideration has to be given to implementing some procedure of feature selection for reducing the number of genes to be used in constructing the rule R.

A number of approaches to feature-subset selection have been proposed in the literature (18). All these approaches involve searching for an optimal or near optimal subset of features that optimize a given criterion. Feature-subset selection can be classified into two categories based or whether the criterion depends on the learning algorithm used to construct the prediction rule (19). If the criterion is independent of the prediction rule, the method is said to follow a filter approach, and if the

Proceedings of the National Academy of Sciences May 2002, 99 (10) 6562-6566; DOI: 10.1073/pnas.102102699

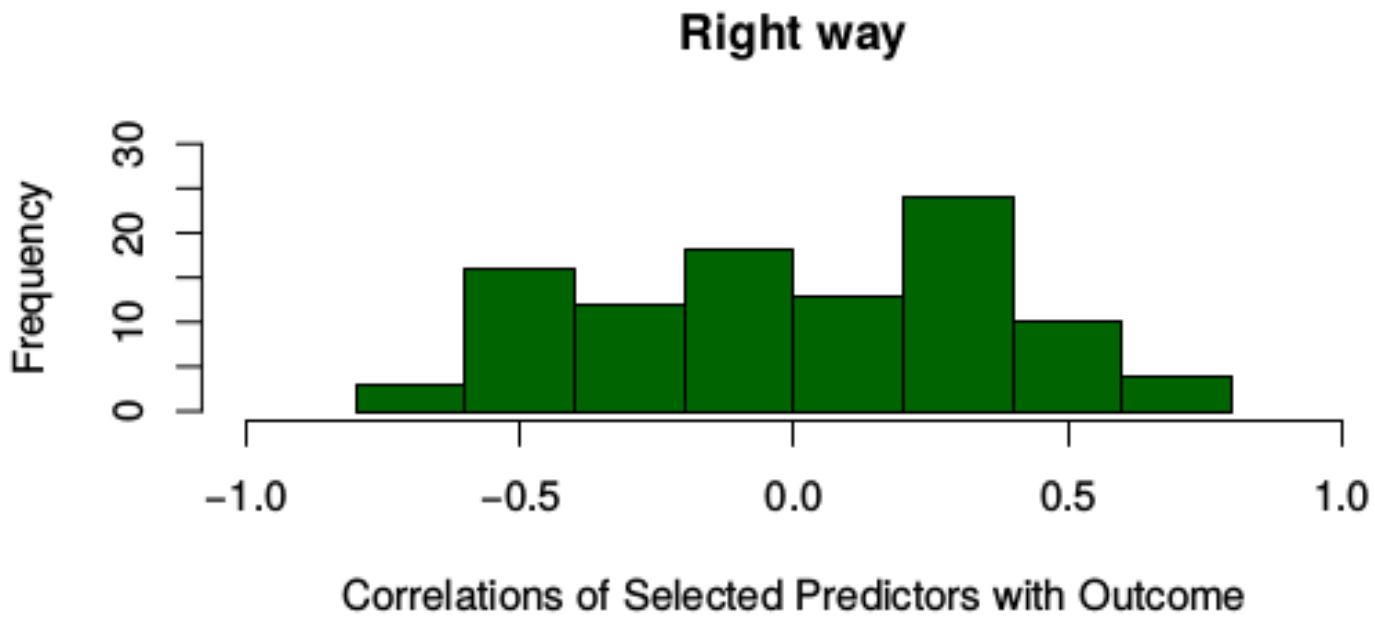
Precaución 1: Manera INCORRECTA CV

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 1: 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



Precaución 2: Preprocessing

Todos los ajustes para el preprocesado de los datos se tienen que realizar sobre el training set, y usar dicha información para transformar el test set.

Precaución 2: Preprocessing

Data transformation with held out data

Just as it is important to test a predictor on data held-out from training, preprocessing (such as standardization, feature selection, etc.) and similar data transformations similarly should be learnt from a training set and applied to held-out data for prediction:

```
>>> from sklearn import preprocessing
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train_transformed = scaler.transform(X_train)
>>> clf = svm.SVC(C=1).fit(X_train_transformed, y_train)
>>> X_test_transformed = scaler.transform(X_test)
>>> clf.score(X_test_transformed, y_test)
0.9333...
```

A `Pipeline` makes it easier to compose estimators, providing this behavior under cross-validation:

```
>>> from sklearn.pipeline import make_pipeline
>>> clf = make_pipeline(preprocessing.StandardScaler(), svm.SVC(C=1))
>>> cross_val_score(clf, iris.data, iris.target, cv=cv)
...
array([0.977..., 0.933..., 0.955..., 0.933..., 0.977...])
```

See Pipelines and composite estimators.

Precaución 2: Preprocessing

Data transformation with held out data

Just as it is important to test a predictor on data held-out from training, preprocessing (such as standardization, feature selection, etc.) and similar data transformations similarly should be learnt from a training set and applied to held-out data for prediction:

```
>>> from sklearn import preprocessing
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train_transformed = scaler.transform(X_train)
>>> clf = svm.SVC(C=1).fit(X_train_transformed, y_train)
>>> X_test_transformed = scaler.transform(X_test)
>>> clf.score(X_test_transformed, y_test)
0.9333...
```

A `Pipeline` makes it easier to compose estimators, providing this behavior under cross-validation:

```
>>> from sklearn.pipeline import make_pipeline
>>> clf = make_pipeline(preprocessing.StandardScaler(), svm.SVC(C=1))
>>> cross_val_score(clf, iris.data, iris.target, cv=cv)
...
array([0.977..., 0.933..., 0.955..., 0.933..., 0.977...])
```

See Pipelines and composite estimators.

Pipeline en scikit

El módulo "Pipeline" es una de las funcionalidades más potentes de scikit y muy posiblemente lo que le hace superior frente a otras librerías de machine learning. Permite realizar una infinidad de operaciones secuenciales sobre los datos en un par de líneas de código.

Ventajas

- **Convenciencia y encapsulacion**: Sólo hay que llamar una vez a "fit" y "predict" para que se realice sobre toda la secuencia de pasos.
- **Selección de parámetros unificada**: Cada paso seguramente dependa de algún hiperparámetro que ajustar. Se pueden usar los metodos "grid search" para explorar dichos parámetros a la vez
- **Seguridad**: Evita resultados demasiado optimistas al hacer algún paso antes de cross-validation. Una vez definido el pipeline, si lo metemos dentro de un esquema de cross-validation, **todos** los pasos se realizan sobre el training set.

Resumen

Todas los conceptos de esta semana se encuentran dentro módulo **model_selection**:

- Método holdout: **train_test_split**.
- Diagnóstico de problemas: **validation_curves**,**learning_curves**.
- Evaluación con reemplazamiento (Bootstrap): **BaggingClassifier**.
- Evaluación sin reemplazamiento (cross-validation): **cross_val_score, cross_val_predict**.
- Esquema de cross-validation: **ShuffleSplit**, **StratifiedShuffleSplit**, **KFold**, **StratifiedKFold**, **LeaveOneOut**, **LeavePOut**, **LeaveOneGroupOut**, **LeavePGroupsOut**, **RepeatedKFold**, **RepeatedStratifiedKFold**.
- Selección del modelo: **GridSearchCV**, **RandomizedSearchCV**, **RidgeCV** , **LassoCV**,**ElasticNetCV**, **LogisticRegressionCV** (En módulo **linear_model** estos cuatros últimos).