Sets de Entrenamiento, Test y Validación

Vamos a comentar las diferencias entre los conjuntos de Entrenamiento, Validación y Test utilizados en Machine Learning ya que suele haber bastante confusión en para qué es cada uno y **cómo utilizarlos adecuadamente**.

un ejemplo práctico por eso de ser didácticos

Además veremos que tenemos distintas técnicas de hacer la validación del modelo y aplicarlas con Scikit Learn en Python.

Un nuevo Mundo

Al principio de los tiempos, sólo tenemos un conjunto Pangea que contiene todo nuestro dato disponible. Digamos que tenemos un archivo csv con 10.000 registros.

Para entrenar nuestro modelo de Machine Learning y poder saber si está funcionando bien, alguien dijo:

Separemos el conjunto de datos inicial en 2: conjunto de entrenamiento (train) y conjunto de Pruebas (test). Por lo general se divide haciendo “80-20”.

Y se toman muestras aleatorias -no en secuencia, si no, mezclado.

Para hacer el ejemplo sencillo, supongamos que queremos hacer clasificación usando un algoritmo supervisado, con lo cual tendremos:

- **X_train** con 8.000 registros para entrenar
- **y_train** con las “etiquetas” de los resultados esperados de X_train
- **X_test** con 2.000 registros para test
- **y_test** con las “etiquetas” de los resultados de X_test

Hágase el conjunto de Test

Lo interesante y a destacar de esto es que una vez los separamos en 8.000 registros para entrenar y 2.000 para probar, usaremos **sólo esos 8.000 registros** para alimentar al modelo al entrenarlo haciendo:

```
modelo.fit(X_train, y_train)
```

Luego de entrenar nuestro modelo y habiendo decidido como **métrica de negocio el Accuracy** (el % de aciertos) obtenemos un 75% sobre el set de entrenamiento (y asumimos que ese porcentaje nos sirve para nuestro objetivo de negocio).

Los 2.000 registros que separamos en **X_test** aún nunca han pasado por el modelo de ML. ¿Se entiende esto? porque eso es muy importante!!! Cuando usemos el set de test, haremos:

```
modelo.predict(X_test)
```

Como verás, **no estamos usando *fit()*!**!!! s**ólo pasaremos los datos sin la columna de "y_test" que contiene las etiquetas**. Además remarco que estamos haciendo predicción; me refiero a que el modelo **NO se está entrenando** ni <<incorporando conocimiento>>. El modelo se limita a "ver la entrada y escupir una salida".

Cuando hacemos el *predict()* sobre el conjunto de test y obtenemos las predicciones, las podemos comprobar y **contrastar con los valores reales** almacenados en *y_test* y hallar así la métrica que usamos. Los resultados que nos puede dar serán:

1. Si el **accuracy** en Test es <<cercano>> al de Entrenamiento (dijimos 75%) por ejemplo en este caso si estuviera **entre 65 ú 85%** quiere decir que n**uestro modelo entrenado está generalizando bien** y lo podemos dar por bueno (siempre y cuando estemos conformes con las métricas obtenidas).

2. Si el **Accuracy** en Test es muy distinto al de Entrenamiento tanto por encima como por debajo, **nos da un 99% ó un 25% (lejano al 75%) entonces es un indicador de que nuestro modelo no ha entrenado bien y no nos sirve**. De hecho este podría ser un indicador de Overfitting.

Para evaluar mejor el segundo caso, es donde aparece el “conjunto de Validación”.

Al Séptimo día Dios creo el Cross-Validation

Si el conjunto de **Train y Test nos está dando métricas muy distintas esto es que el modelo no nos sirve.**

Para mejorar el modelo, p**odemos pensar en Tunear sus parámetros y volver a entrenar** y probar, podemos **intentar obtener más registros, cambiar el preprocesado de datos, limpieza, balanceo de clases, selección de features, generación de features… De hecho, podemos pensar que seleccionamos un mal modelo, y podemos intentar con distintos modelos: de árbol de decisión, redes neuronales, ensambles…**

> La técnica de *Validación Cruzada* nos ayudará a medir el comportamiento el/los modelos que creamos y nos ayudará a encontrar un mejor modelo rápidamente.

Aclaremos antes de empezar: hasta ahora contamos con 2 conjuntos: el de Train y Test.

El “set de validación” ***no es realmente un tercer set*** *si no que “vive” dentro del conjunto de Train*. Reitero: el set de validación no es un conjunto que apartemos de nuestro archivo csv original. El set de validación se utilizará durante iteraciones que haremos con el conjunto de entrenamiento.

Sets de Entrenamiento, Test y Validación

Técnicas de Validación Cruzada

Entonces volvamos a tener las cosas claras: **SOLO tenemos conjunto de Train y Test,** ok?. El de Test seguirá tratándose como antes: lo apartamos y lo usaremos al final, una vez entrenemos el modelo.

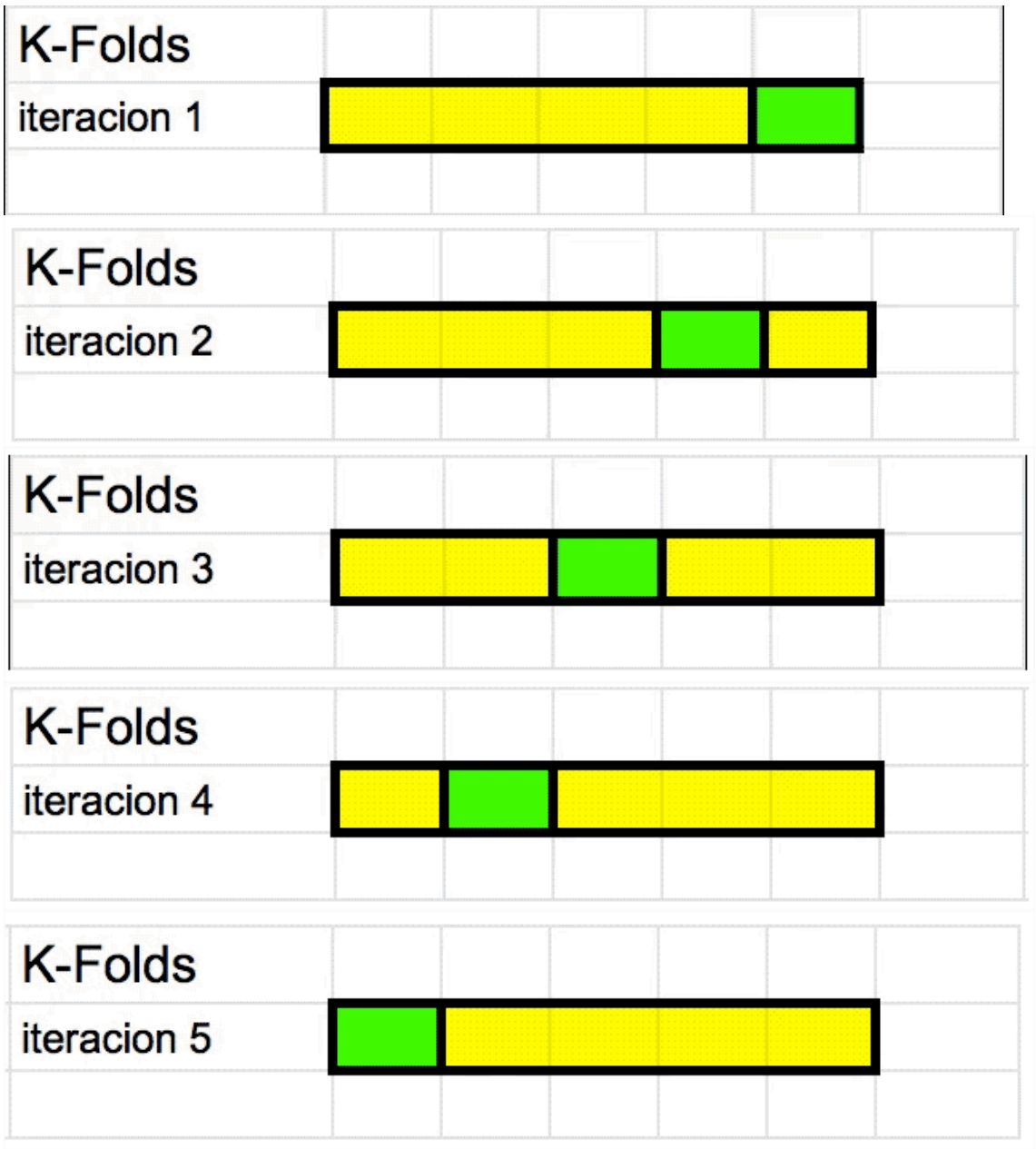
Dentro del conjunto de Train, y siguiendo nuestro ejemplo inicial, tenemos 8.000 registros. **La validación más común utilizada** y que nos sirve para entender el concepto es "**K-folds**", vamos a comentarla:

Cross-Validation: K-fold con 5 splits

Lo que hacemos normalmente al entrenar el modelo es pasarle los 8.000 registros y que haga el fit(). Con K-Folds -en este ejemplo de 5 splits- para entrenar, en vez de pasarle todos los registros directamente al modelo, haremos así:

- Iterar 5 veces:
  1. Apartaremos 1/5 de muestras, es decir 1600.
  2. Entrenamos al modelo con el restante 4/5 de muestras = 6400.
  3. Mediremos el accuracy obtenido sobre las 1600 que habíamos apartado.
- Esto quiere decir que hacemos 5 entrenamientos independientes.
- El Accuracy final será el promedio de las 5 accuracies anteriores.

Sets de Entrenamiento, Test y Validación



En amarillo las muestras para entrenar y en verde el conjunto de Validación.

Entonces fijémonos que **estamos “ocultando” una quinta parte del conjunto de train durante cada iteración**.

Esto es similar a lo que explique antes, pero esta vez aplicado al momento de entrenamiento.

Al cabo de esas 5 iteraciones, obtenemos **5 accuracies** que deberían **ser “similares**” entre sí, esto sería un indicador de que el modelo **está funcionando bien**.

Sets de Entrenamiento, Test y Validación

Ejemplo K-Folds en Python

Veamos en código python usando la librería de data science scikit-learn como podemos hacer el cross-validation con K-Folds:

```
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.2, random_state=0)

kf = KFold(n_splits=5)

clf = LogisticRegression()

clf.fit(X_train, y_train)

score = clf.score(X_train,y_train)

print("Metrica del modelo", score)

scores = cross_val_score(clf, X_train, y_train, cv=kf,
scoring="accuracy")

print("Metricas cross_validation", scores)

print("Media de cross_validation", scores.mean())

preds = clf.predict(X_test)

score_pred = metrics.accuracy_score(y_test, preds)

print("Metrica en Test", score_pred)
```

En el ejemplo vemos los pasos descritos anteriormente:

- Cargar el dataset
- Dividir en Train y Test (en 80/20)
- Creamos un modelo de Regresión Logística (podría ser otro) y lo entrenamos con los datos de Train
- Hacemos Cross-Validation usando K-folds con 5 splits
- Comparamos los resultados obtenidos en el modelo inicial, en el cross validation y vemos que son similares.
- Finalmente hacemos predict sobre el Conjunto de Test y veremos que también obtenemos buen Accuracy

Más técnicas para Validación del modelo

Otras técnicas usadas y que nos provee sklearn para python son:

Stratified K-Fold

Statified K-fold es una **variante mejorada de K-fold**, que cuando hace los splits (las divisiones) **del conjunto de train tiene en cuenta mantener equilibradas las clases**. Esto es muy útil, porque imaginen que tenemos que clasificar en “SI/NO” y si una de las iteraciones del K-fold normal **tuviera muestras con etiquetas sólo “SI” el modelo no podría aprender a generalizar y aprenderá para cualquier input a responder “SI”.** Esto lo soluciona el Stratified K-fold.

Leave P Out

Leave P Out selecciona una cantidad P por ejemplo 100. Entonces se separarán de a 100 muestras contra las cuales validar y se iterará como se explico anteriormente. Si el valor P es pequeño, esto resultará en muchísimas iteraciones de entrenamiento con un alto coste computacional (y seguramente en tiempo). Si el valor P es muy grande, podría contener más muestras que las usadas para entrenamiento, lo cual sería absurdo. Usar esta técnica con algo de sentido común y manteniendo un equilibrio entre los scores y el tiempo de entreno.

ShuffleSplit

ShuffleSplit primero mezcla los datos y nos deja indicar la cantidad de splits (divisiones) es decir las iteraciones independientes que haremos y también indicar el tamaño del set de validación.

Series Temporales: Atención al validar

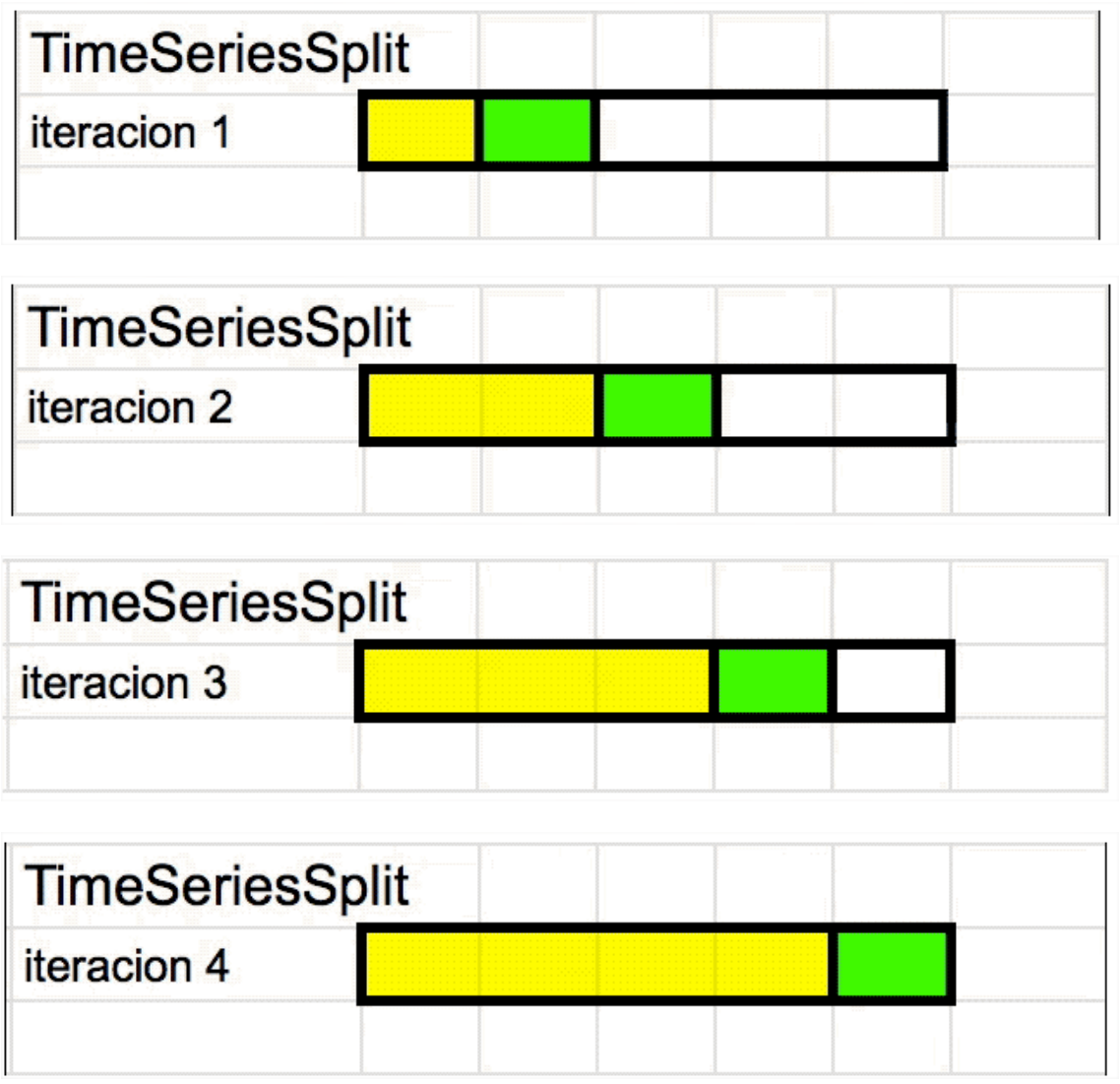
Para problemas de Series temporales tenemos que prestar especial cuidado con los datos. Pues si pasamos al modelo “dato futuro” antes de tiempo estaríamos haciendo Data Leakage, esto es como si le hiciéramos spoiler al modelo y le contaremos el final de la película antes de que la vea. Esto causaría overfitting.

Para empezar al hacer el split inicial de datos estos deberán estar ordenados por fecha y no podemos mezclarlos.

Para ayudarnos con el cross-validation sklearn nos provee de TimeSeriesSplit.

TimeSeriesSplit

TimeSeriesSplit es una variante adaptada de K-folds que evita “la fuga” de datos. Para hacerlo va iterando los “folds” de a uno (usando una ventana de tiempo que se desplaza) y usando el “fold más reciente” cómo el set de validación. Se puede entender mejor viendo una animación:



En Amarillo las muestras para entrenar y en verde el conjunto de Validación.

Pero entonces? Cuando uso Cross-Validation?

Es una buena práctica **usar cross-validation en nuestros proyectos**. De hecho **usarlo nos ayudará a elegir el modelo correcto** y nos da mayor seguridad y respaldo ante nuestra decisión.

PERO… (siempre hay un pero)

En casos en los que hacer 1 sólo entrenamiento “normal” tome muchísimo tiempo y recursos, podría ser nuestra perdición. Imaginen que hacer un k-folds de 10 implica hacer 10 entrenos -aunque un poco más pequeños-, pero que consumirían mucho tiempo y dinero.

Entonces en la medida de lo posible siempre usar validación cruzada. Y -vuelvo a reforzar el concepto- luego se probará el modelo contra el conjunto de Pruebas (test).

Para hacer tuneo de Hiper-parámetros como RandomSearch, GridSearch ó Tuneo Bayesiano es muy útil hacer Cross-Validation.

¿Si ya estoy “conforme” y quiero llevar el modelo a un entorno de Producción?

Supongamos que el entrenamiento haciendo Cross Validation y el predict() en Test nos están dando buenos accuracy (y similares) y estamos conformes con nuestro modelo. PUES si lo queremos usar en un entorno REAL y productivo, ANTES de publicarlo es recomendado que agreguemos el conjunto de test al modelo!!!, pues así estaremos aprovechando el 100% de nuestros datos. Espero que esto último también se entienda porque es super importante: lo que estoy diciendo es que si al final de todas nuestras iteraciones, pre procesado de dato, mejoras de modelo, ajuste de hiper-parámetros y comparando con el conjunto de test, estamos seguros que el modelo funciona correctamente, es entonces ahora, que usaremos las 10.000 muestras para entrenar al modelo, y ese modelo final, será el que publicamos en producción.

Es una última iteración que debería mejorar el modelo final aunque este no lo podemos contrastar contra nada… excepto con su comportamiento en el entorno real.

Si esta última iteración te causara dudas, no la hagas, excepto que tu problema sea de tipo Serie Temporal. En ese caso sí que es muy importante hacerlo o quedaremos con un modelo que no “es el más actual”.

Sets de Entrenamiento, Test y Validación

Resumen, Conclusiones y por favor Que quede claro!

Lo más importante que quisiera que quede claro es que entonces tenemos 2 conjuntos: u**no de Train y otro de Test**.

El "conjunto de validación" no existe como tal, si no, que "vive temporalmente" al momento de entrenar y nos ayuda a obtener al mejor modelo de entre los distintos que probaremos para conseguir nuestro objetivo.

Esa técnica es lo que se llama Validación Cruzada ó en inglés cross-validation.

NOTA: en los ejemplos de la documentación de s**klearn podremos ver que usan las palabras train y test. Pero conceptualmente se está refiriendo al conjunto de validación y no al de Test que usaremos al final. Esto es en parte el causante de tanta confusión con este tema.**

Tener en cuenta el tamaño de split 80/20 es el usual pero puede ser distinto, y esta proporción puede cambiar sustancialmente las métricas obtenidas del modelo entrenado! Ojo con eso. El tamaño ideal dependerá del dominio de nuestro problema, deberemos pensar en una cantidad de muestras para test que nos aseguren que estamos el modelo creado está funcionando correctamente. Teniendo 10.000 registros puede que con testear 1000 filas ya estemos conformes ó que necesitemos 4000 para estar mega-seguros. Por supuesto debemos recordar que las filas que estemos "quitando" para testear, no las estamos usando al entrenar.

Otro factor: al hacer el experimento y tomar las muestras mezcladas, mantener la "semilla" ó no podremos reproducir el mismo experimento para comparar y ver si mejora o no. Este suele ser un parámetro llamado "random_state" y está bien que lo usemos para fijarlo.

Recomendaciones finales:

- En principio separar Train y Test en una proporción de 80/20
- Hacer Cross Validation siempre que podamos:
    - No usar K-folds. Usar Stratified-K-folds en su lugar.
    - La cantidad de "folds" dependerá del tamaño del dataset que tengamos, pero la cantidad usual es 5 (pues es similar al 80-20 que hacemos con train/test).
    - Para problemas de tipo time-series usar TimeSeriesSplit
- Si el Accuracy (ó métrica que usamos) es similar en los conjuntos de Train (donde hicimos Cross Validation) y Test, podemos dar por bueno al modelo

Sets de Entrenamiento, Test y Validación