- Ambiente
- `Scikit-learn`
- Procedimiento
- Transformaciones
- Feature Engineering
- Evaluación *Off-line*
- Selección de modelos
- *Hyperparameter tuning*
- *Pipeline* de Modelos
- Producción
- Evaluación *On-line*
- ¿Por qué de todo esto?
- ¿Qué nos faltó de cubrir?
- Conclusiones

Instalacion

NOTA: Todo lo que sigue, presupone que estás trabajando en la carpeta de la clase, i.e. `models`

```
python --version
```

```
Python 3.5.2
```

Debido a un *bug* de `pip`

Primero instala `Cython` en Anaconda

```
pip install Cython
```

Luego instala `feather`

```
pip install feather-format
```

Y ahora el resto de los paquetes

```
pip install -r requirements.txt
```

Si todo salió bien deberías de poder hacer lo que sigue:

Abre tu `jupyter notebook`

Teclea lo siguiente

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
## Paquete para mejorar la estética de matplolib
import seaborn as sns
sns.set()
```

Si todo funcionó bien, prosigue.

En **Rstudio** teclea lo siguiente

```
library(feather)
write_feather(x = iris, path='iris.feather')
```

Y en jupyter:

```
import feather
iris_df = feather.read_dataframe('iris.feather')
iris_df.head()
```

```
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Si llegaste hasta aquí, **felicidades** `:D`

Scikit-learn

Scikit-learn

- Implementación de muchos algoritmos de aprendizaje de máquina
- `API` limpia y uniforme

¿Cómo represento los datos?

- Para muchos algoritmos de **Aprendizaje de Máquina** los datos deben de venir en una tabla 2D y de preferencia ser *tidy*
    - Aunque como vimos antes no necesariamente es lo óptimo
- En `scikit-learn` esta tabla será numérica y se le llamará *features matrix* regularmente representada por X, con dimensiones `[ num_samples, num_features ]`
- El *sample* representa un individuo: Una persona, una imagen, un documento, vídeo, etc,
- También (en el caso supervisado) se tiene un arreglo (*array*) llamado *target* o *label*, que se representará por y
- y regularmente es unidimensional, con una longitud de `[num_samples,]`
- y puede ser numérico o discreto.

Ejemplo

```
iris = sns.load_dataset('iris')
X_iris = iris.drop('species', axis=1)
print(X_iris.shape)
```

```
(150, 4)
```

```
y_iris = iris['species']
print(y_iris.shape)
```

```
(150,)
```

Estimator API

Filosofía

Lo que sigue está basado en el artículo API design for machine learning software: experiences from the scikit-learn project (https://arxiv.org/abs/1309.0238) de *Buitinck et al*.

La `API` de `scikit-learn` se diseñó guiándose en los siguientes principios:

- **Consistencia**: Todos los objetos comparten una interfaz, con buena documentación
- **Inspección**: Todos los parámetros son públicos
- **Jerarquía de objetos limitada**: Los algoritmos son clases de `Python` , los *datasets* son arreglos de `numpy` , o *data frames* de `pandas` o matrices *sparse* de `scipy` , los parámetros son cadenas.
- **Composición**. Esto tendrá sentido cuando lleguemos a `Pipelines`
- **Valores por omisión**: Cuando haya valores por especificar en los modelos, la API definirá valores por omisión apropiados.

Parámetros e *hiper-parámetros*

En los modelos de aprendizaje automatico, los parametros son los valores que se estiman durante el proceso de entrenamiento con el conjunto de datos. Sus valores no los indica manualmente el cientifico de datos, sino que son obtenidos al final del proceso de entrenamiento.Algunos ejemplos de parametros de modelos de aprendizaje automatico son:

1. Los coeficientes en una regresion lineal.
2. Los pesos en una red neuronal artificial.
3. Los vectores de soporte en una maquina de vectores de soporte

Por otra parte, los hiperparametros de un modelo son los valores de las configuraciones utilizadas durante el proceso de entrenamiento. Son valores que generalmente se no se obtienen de los datos, por lo que suelen ser indicados por el cientifico. El valor optimo de un hiperparametro no se puede conocer a priori para un problema dado. Por lo que se tiene que utilizar valores genericos, los valores que han funcionado anteriormente en problemas similares.

Al entrenar un modelo de aprendizaje de maquina se fijan los valores de los hiperparametros para que con estos se obtengan los parametros. Algunos ejemplos de hiperparametros utilizados para entrenar los modelos son:

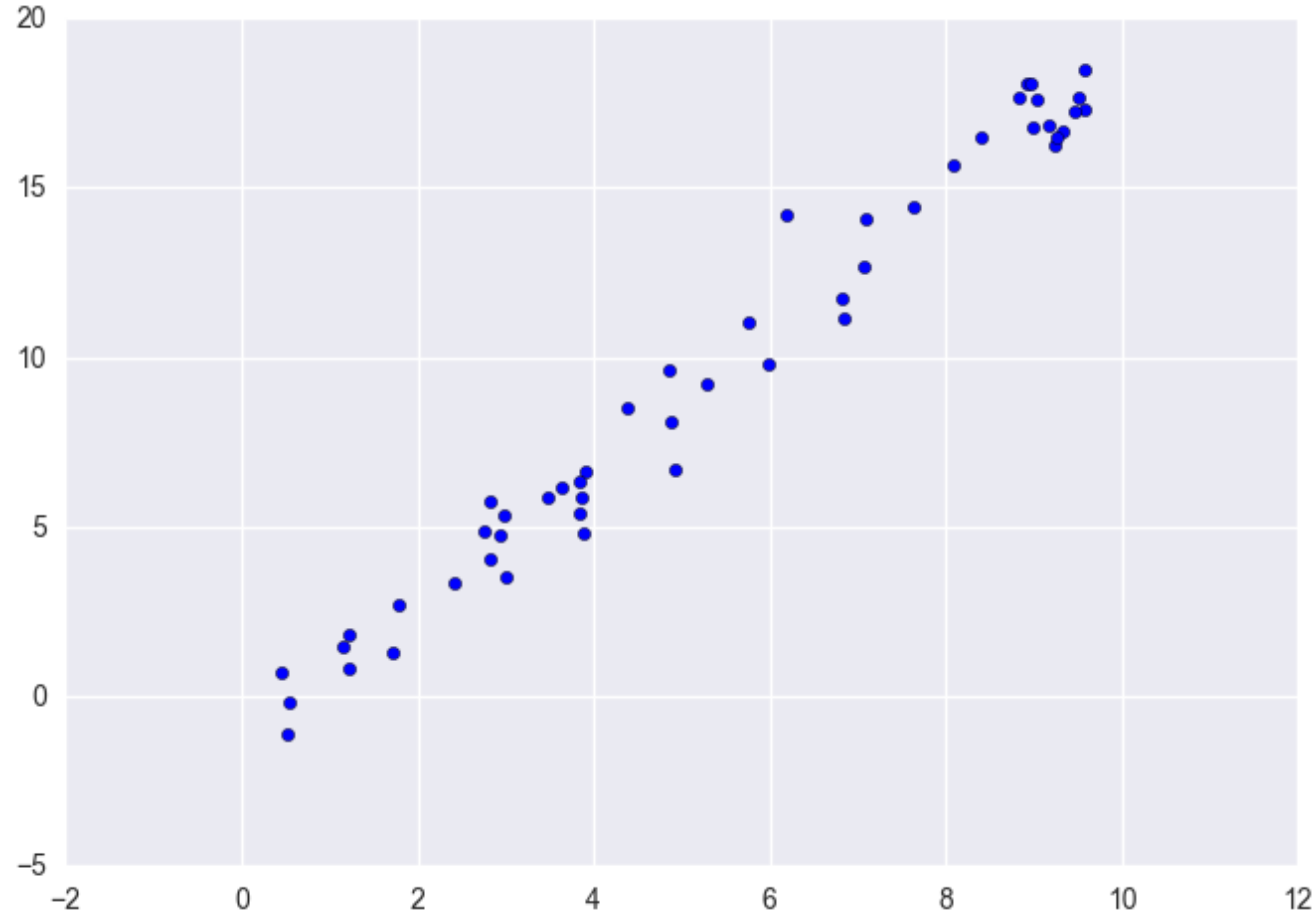
1. El numero de vecinos en k-vecinos mas cercanos (k-nn).
2. La profundidad maxima en un arbol de decision

Manual de supervivencia

1. Escoje un modelo
2. Escoje los *hiper-parámetros* del modelo
3. Prepara el *dataset* en *features matrix* y el vector *target*
4. Divide el *dataset* en *train* y *test*
5. Aplica el algoritmo a tus datos de entrenamiento
6. Aplica el algoritmo a tus datos de prueba
7. Evalúa el modelo
8. Aplica el modelo en nuevos datos

Generamos un *dataset* de juguete

```
rng = np.random.RandomState(5432)
x = 10 * rng.rand(50)
y = 2 * x  - 1 + rng.randn(50)
plt.scatter(x,y)
```



img

Ejemplo: *Supervised Learning*

```
## 1.  Escoje un modelo
from sklearn.linear_model import LinearRegression

## 2. Escoje los hiper-parametros
## Ve la documentacion:
## ?LinearRegression
modelo = LinearRegression(fit_intercept = True)
```

```
## 3.Prepara el /dataset/ en /features matrix/ y el vector /target/
## x no es una matriz, es un vector, i.e. su shape es (50,)
X = x[:, np.newaxis]
print(X.shape)
```

```
(50, 1)
```

```
## 4. Divide el dataset en train y test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```
## 5. Aplica el algoritmo a tu datos de entrenamiento
modelo.fit(X_train, y_train)

print("Coeficientes: {}".format(modelo.coef_))

print("Interceptor: {}".format(modelo.intercept_))
```

```
## 6. Aplica el algoritmo a tus datos de prueba
y_from_model = modelo.predict(X_test)

## 7. Evalúa el modelo
modelo.score(X_test, y_test) ## Devuelve el R^2
```

```
## 8. Aplica el modelo en nuevos datos
x_new = np.linspace(-1, 11)

X_new = x_new[:, np.newaxis]
y_new = modelo.predict(X_new)
## Graficamos el modelo y los datos
plt.scatter(x, y)
plt.plot(X_new, y_new, c='green')
```

Ejemplo: *Transformers*

En Python es posible limpiar, reducir, expandir o generar caracteristicas. Los transformers ayudan en dicha tarea.Ejemplos de transformadores predefinidos incluyen StandardScaler, LabelEncoder, Normalize.

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split

boston = load_boston()

X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)

print("Media: %s " % X_train.mean(axis=0))
print("Desviacion estandar: %s " % X_train.std(axis=0))
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(X_train)

X_scaled = scaler.transform(X_train)

print("Media: %s " % X_scaled.mean(axis=0))
print("Desviacion estandar: %s " % X_scaled.std(axis=0))
```

Ejemplo: Reducción de dimensiones

```
from sklearn.datasets import make_s_curve

X, color= make_s_curve(n_samples=1000)

from mpl_toolkits.mplot3d import Axes3D

plt.close()

fig = plt.figure()

ax = plt.axes(projection = '3d')

ax.scatter3D(X[:,0], X[:,1], X[:,2], c=color, cmap=plt.cm.Spectral)
ax.view_init(10, -60)
```

```
## Linear transformers en el paquete decomposition
from sklearn.decomposition import PCA
pca = PCA(n_components = 2).fit(X)
X_pca = pca.transform(X)
plt.close()
plt.scatter(X_pca[:,0], X_pca[:,1], c=color, cmap=plt.cm.Spectral)
```

```
## No linear transformers en el paquete  manifold
from sklearn.manifold import Isomap
iso = Isomap(n_neighbors = 20)
iso.fit(X)
X_iso = iso.transform(X)

plt.close()
plt.scatter(X_iso[:,0], X_iso[:,1], c=color, cmap=plt.cm.Spectral)

plt.savefig('../images/s-figure-isomap.png')
```

```
from sklearn.manifold import TSNE

tsne = TSNE()

## t-sne no tiene una API estándar :/
## Esto falla:
## tsne.fit(X)
## X_tsne = tsne.transform(X)
## BOOOM!!!
## Se puede arreglar con:
## X_tsne = tsne.embedding_

X_tsne = tsne.fit_transform(X)

plt.close()
plt.scatter(X_tsne[:,0], X_tsne[:,1], c=color, cmap=plt.cm.Spectral)

plt.savefig('../images/s-figure-tsne.png')
```

Ejemplo final: Dígitos

```
from sklearn.datasets import load_digits
digits = load_digits()

digits.images.shape

import matplotlib.pyplot as plt

fig, axes = plt.subplots(10,10, figsize=(8,8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```

```
X = digits.data
print("Dimensiones de los features: {}".format(X.shape))

y = digits.target
print("Dimensiones del target: {}".format(y.shape))
```

```
from sklearn.manifold import TSNE

tsne = TSNE()

X_projected = tsne.fit_transform(X)

X_projected.shape

plt.close()

plt.scatter(X_projected[:,0], X_projected[:,1], c=y,
            edgecolor = 'none',
            alpha = 0.5,
            cmap = plt.cm.get_cmap('Spectral', 10))

plt.colorbar(label = 'etiqueta', ticks = range(10))

plt.clim(-0.5, 9.5)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=1000)
model.fit(X_train, y_train)
y_model = model.predict(X_test)


## Accuracy
from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(y_test, y_model)))
```

```
## Métricas clasicas de Clasificadores
from sklearn import metrics
print(metrics.classification_report(y_model, y_test))
```

```
## Matriz de confusión
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(y_test, y_model)

plt.close()

sns.heatmap(mat, square = True, annot = True, cbar = False)
plt.xlabel('Valor predicho')
plt.ylabel('Valor real')
```

```
## O visualmente

plt.close()

fig, axes = plt.subplots(10,10, figsize=(8,8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(X_test.reshape(450,8,8)[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(y_model[i]),
            transform=ax.transAxes,
            color='green' if (y_test[i] == y_model[i]) else 'red')
```

Ejercicio: Inténtalo con `PCA` e `Isomap` ¿Cuél divide mejor este *dataset*?

Ejercicio: Cambia los hiper-parámetros del `RandomForestClassifier` , primero disminuye los arboles a 10. ¿Cual resulta mejor?

Ejercicio: Cambia de clasificador a `GaussianNB` ¿Cual resulta mejor para este *dataset*?

Resumen de la API

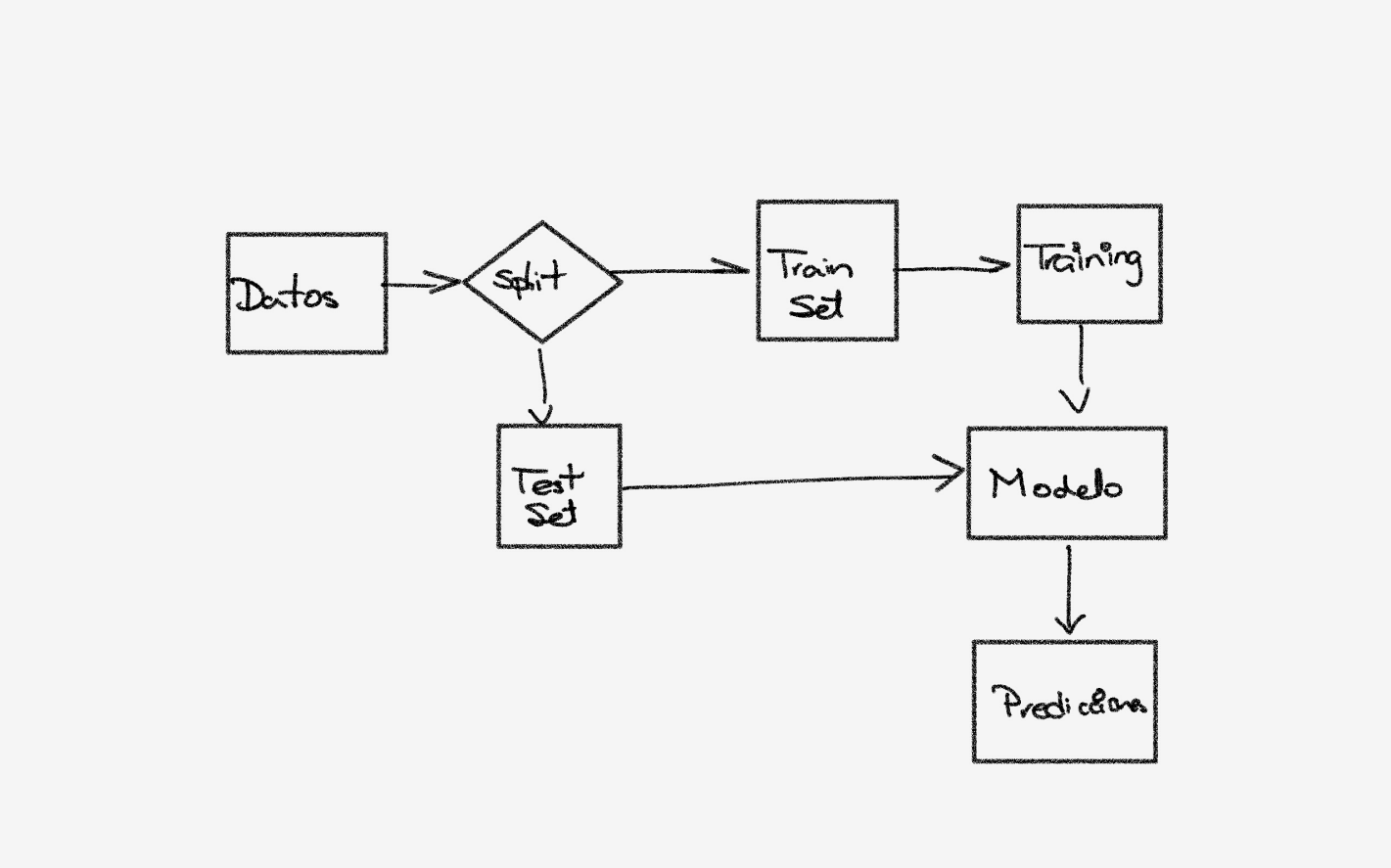
Para todos los estimadores: **`model.fit(X_train, [y_train])`**

Por tipo:

`model.predict(X_test)`	**`model.transform(X_test)`**
Clasificacion	Preprocesamiento
Regresión	Reducción de dimensiones
Clustering	*Feature Extraction*
	Feature Selection

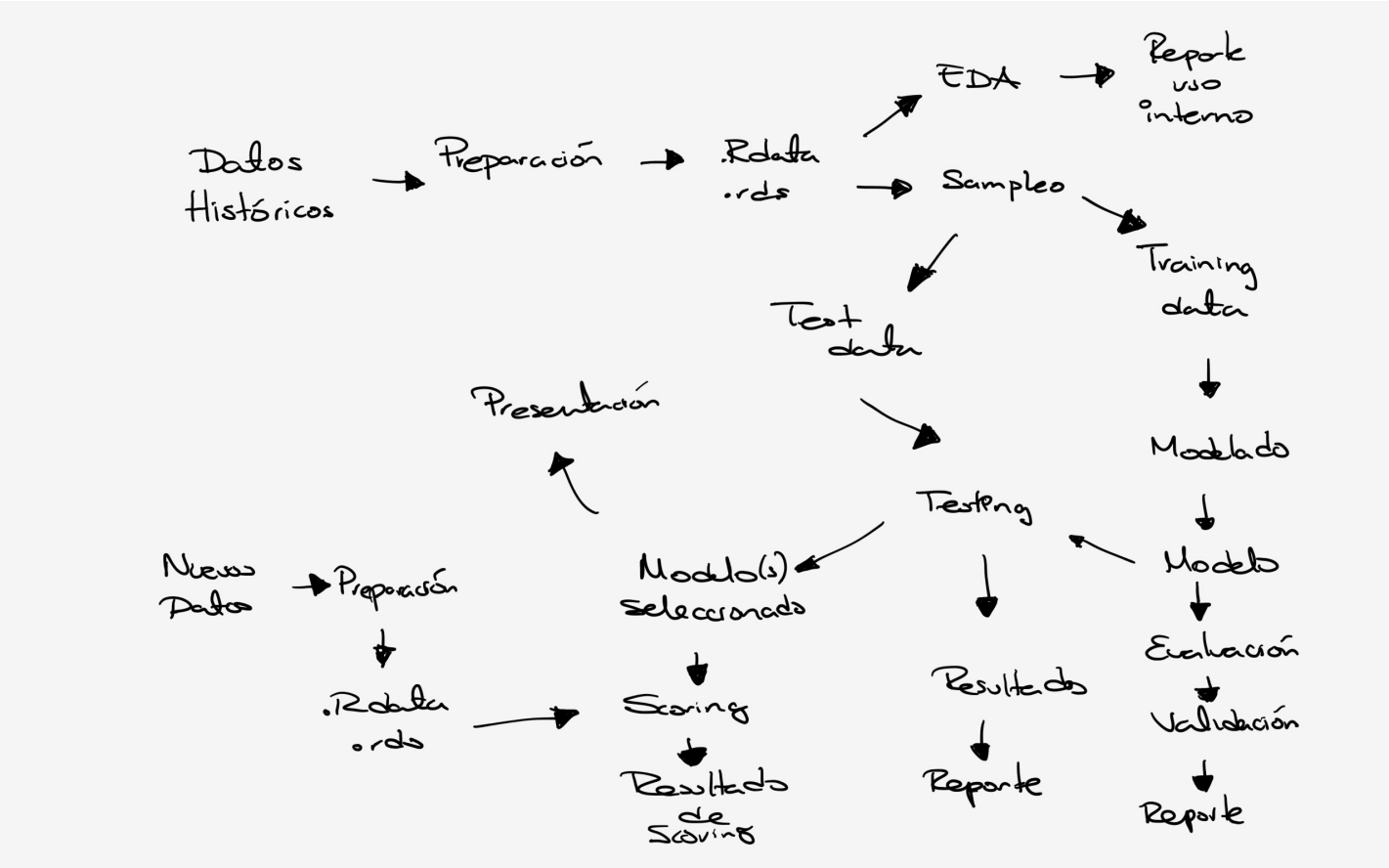
Procedimiento

Versión simplificada



img

Versión completa



img

Procedimiento

- Mapeamos el problema de negocio a una técnica de modelado.
- Dividimos (por lo menos) a los datos en dos: *training* y *testing*
    - Aunque ve más adelante
- Esto lo hacemos para hacer pruebas en el modelo + datos:
    - Evaluación del modelo
    - Validación del modelo
- Evaluación del modelo
    - Cuantificación del desempeño del modelo.
    - Encontrar medidas que sean apropiadas para la meta de negocio y para la técnica que estamos usando.
- Validación del modelo
    - Procedimiento para verificar (con una medida de certidumbre) de que el modelo se comportará en producción tan bien como lo hizo en entrenamiento.
    - Un problema muy grande para este procedimiento pueden ser:
        - No hay suficientes datos para tener un conjunto de *training data*.
        - Los datos de entrenamiento no tienen la suficiente variedad comparada con producción.

Clasificación

- ¿Qué tipo es?
- Pertenece a las técnicas de *supervised learning*
- Crear un conjunto de datos para entrenamiento es muy caro (en `$`), a este proceso se le conoce como etiquetado

Algunos ejemplos

Técnica	**Notas**
Naïve Bayes	Muchas variables de entrada, categóricas con muchos niveles, clasificación de texto
Árboles de decisión	Variables de entrada interactúan con la salida de manera `if-then`. Las variables de entrada son redundantes o están correlacionadas.
Regresión Logística	Estimar las probabilidades de pertenencia a la clase de salida. Quieres conocer el impacto relativo de las variables de entrada a las de salida.
SVM	Muchas variables de entrada que interactuan de maneras complicadas o no-lineales. Hace pocas suposiciones sobre la distribución de las variables, lo cual lo hace bueno cuando los datos de entrenamiento no sean tan representativos de lo que pasa en producción.

Regresión/Scoring

- ¿Cuánto?
  - Ejemplo: ¿Cuánto van a aumentar las ventas por esta campaña?
  - Detección de fraude, puede ser considerado *scoring* si tratas de estimar la probabilidad de que una transacción en particular sea fraudulenta.
- Ejemplos son la regresión logística y la regresión lineal.

Cuando no hay variable de salida *target*

- Pertenece a las técnicas de *unsupervised learning*
- En lugar de predecir las variables de salida usando las variables de entrada, el objetivo es descubir similitudes y relaciones en los datos.
- Ejemplos:
  - Agrupamiento por `k-means`
    - Segmentar `clientes` por patrones similares de compra.
  - Algoritmo `apriori`.
    - Segmentar `productos` que se compran juntos.
  - Vecinos cercanos.
    - Decir algo sobre el punto `p` respecto a puntos que más se parecen a `p`.

Transformaciones

Variables Categóricas

- Las dos representaciones más usadas para variables categóricas son *one-hot-encoding* (**Machine Learning**) o *dummy variables* (**Estadística**)
- Es importante verificar que la columna contenga realmente datos categóricos (y no por ejemplo texto capturado por humanos)

One-hot-encoding

- *One-hot-encoding* codifica la variable categórica con k niveles en k *features*

var	var_A	var_B	var_C
A	1	0	0
B	0	1	0
C	0	0	1
A	1	0	0

Dummy variables

- *Dummy variables* codifica la variable categórica con k niveles en $k-1$ *features* (el último representado por ceros)

var	var_A	var_B

var	var_A	var_B
A	1	0
B	0	1
C	0	0
A	1	0

Ejemplo

```
import feather

iris_df = feather.read_dataframe('iris.feather')

iris_df.head()
```

- Usando Pandas

```
## Usando pandas
print(pd.get_dummies(iris_df['Species']).head())
```

- Usando `Scikitlearn`

```
## Usando OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

label_encoder = LabelEncoder()  ## Para convertir a enteros
one_hot_encoder = OneHotEncoder()

## Convertirmos a enteros, i.e. setosa -> 0, etc
species = label_encoder.fit_transform(iris_df['Species'])

## Ya no se le pueden pasar vectores (arreglos 1D) a los preprocesadores
species = species[:, np.newaxis]

## Debemos invocar todense() ya que OneHotEncoder devuelve una matriz rala
## La otra opción es OneHotEncoder(sparce=False)
species_one_hot = one_hot_encoder.fit_transform(species).todense()
print(species_one_hot[:5,:])
```

Binning

- Transformar variables numéricas a categóricas.
- Pueden ayudar a construir modelos más complejos con regresiones lineales (i.e. aumentan su expresividad)
- Los árboles de decisión lo hacen automáticamente, por lo que no se ven beneficiados de este tipo de transformaciones
- En `python` se realiza con la función `np.digitize`

Feature Engineering

¿Qué es?

- Es el proceso de determinar que variables productivas contribuyen mejor al poder predictivo del algoritmo.
- ****FE**** es, quizá, la parte más importante del proceso de minería de datos.
    - Con buenas variables, un modelo simple puede ser mejor que un modelo complicado con malas variables.
- Es el elemento humano en el modelado: El entendimiento de los datos, más la intuición y la creatividad, hacen toda la diferencia.
- Es más un arte que una ciencia.
- Regularmente es un proceso iterativo con el EDA.
- Un **domain expert** puede ser de mucha utilidad en esta etapa.
- *Feature Learning* es lo que hace *Deep Learning* y no se verá en este curso

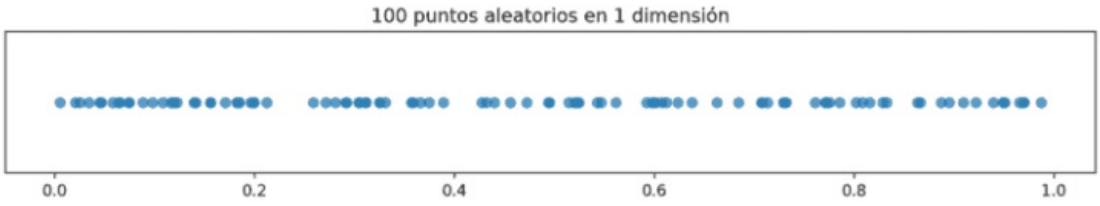
La maldición de la dimensionalidad

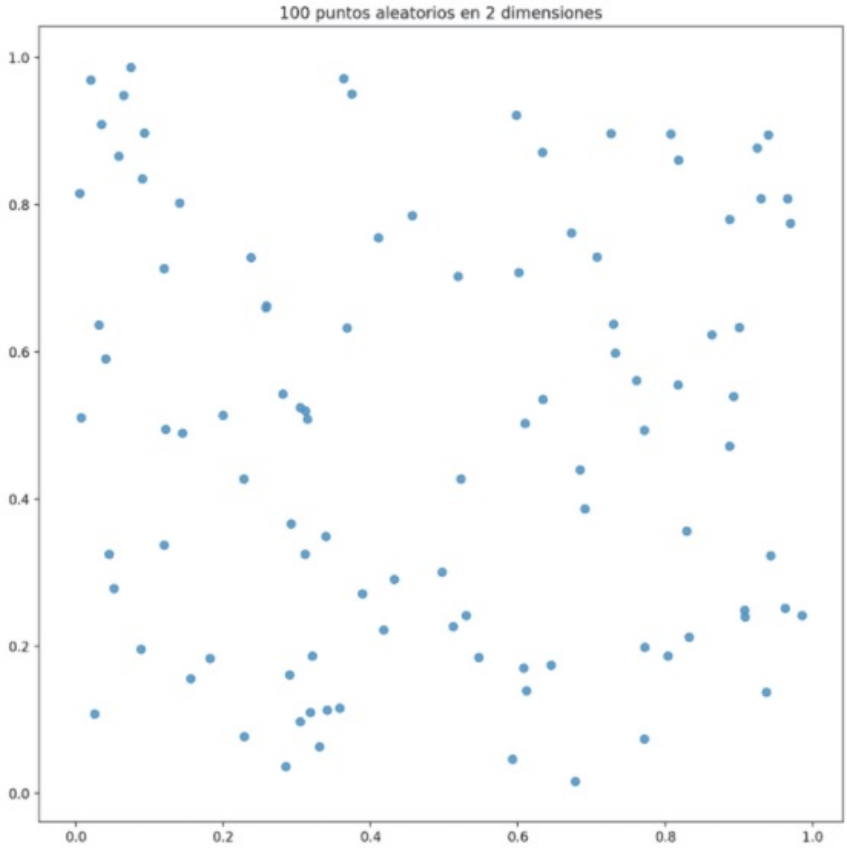
- El número de combinaciones valores-variables puede ser muy grande en un problema típico de ML.
- Sea n el número de variables y sea a_i el número de posibles valores de la variable i, $1 \leq i \leq n$. El número de combinaciones está dado por

$$m = \Pi_i^n a_i$$

- Para 100 variables con 10 valores cada uno, m es mayor que el número de partículas en el Universo ($\sim 10^{80}$).
- Posibles soluciones:
    - Reducir el espacio de búsqueda.
    - Realizar búsqueda inteligente (heurística)

La causa comun de estos problemas es que cuando aumenta la dimensionalidad, el volumen del espacio aumenta exponencialmente haciendo que los datos disponibles se vuelven dispersos. Con el fin de obtener un resultado estadisticamente solido y fiable, la cantidad de datos necesarios para mantener el resultado a menudo debe crecer tambien exponencialmente con la dimensionalidad.





img

Feature Generation

Proceso Manual

- *Brainstorming*
  - No juzguen en esta etapa
  - Permitan y promuevan ideas muy locas.
  - Construyan en las ideas de otros
  - No divaguen
  - Sean visuales
  - Vayan por cantidad, la calidad se verá luego
  - Otros consejos se puden consultar aquí (https://challenges.openideo.com/blog/seven-tips-on-better-brainstorming)

Proceso Manual (continuación)

- Decidir que *features* crear
    - No hay tiempo infinito
- Crear esos *features*
- Estudiar el impacto de los *features* en el modelo
- Iterar

Proceso Automatizado

- Interacción multiplicativa
    - $C = A \cdot B$
    - Hacer para todas las posibles combinaciones.
    - Es importante mencionar que estas nuevas variables benefician mucho a los regresores lineales, pero no afectan mucho a árboles de decisión (e.g. el Random Forest)
- Interacción de razón
    - $C = A/B$
    - Tener cuidado con dividir por cero $\rightarrow$ hay que tomar una decisión
    - Hacer para todas las posibles combinaciones.

Ejemplo: Interacciones y Polinomios

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn import datasets

boston = datasets.load_boston()

X = boston.data
y = boston.target

X_train, X_test, y_train, y_test = train_test_split(X,y)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

interactions = PolynomialFeatures(degree=2).fit(X_train_scaled)
X_train_inter = interactions.transform(X_train_scaled)
X_test_int = interactions.transform(X_test_scaled)

## ¡Ahora hay 105 variables!
print(X_train_inter.shape)
```

```
"Nombres de las variables de interacción: {}\n".format(interactions.get_feature_names())
```

Proceso Automatizado (continuación)

- Transformar una variable numérica en una binaria.
    - Se trata de encontrar el `cut-off` que maximize tu variable dependiente.
    - Muy parecido a lo que hacen algoritmos como el `J48` (en su versión comercial se conoce como `C5`).
    - Hay un paquete de `R` que lo implementa: `C50`.
- Numérica $\rightarrow$ bin.
- Otras
    - X^2
    - $\log X$
    - etc.

Feature Selection

¿Qué es?

- El proceso de seleccionar variables antes que ejecutar los algoritmos.
- Realiza `cross-validation`
    - Realizar `cross-validation` sólo en una parte del proceso (i.e. el modelo) es hacer trampa.
- ¡Cuidado! No hagas `feature selection` en todos tus datos antes de construir el modelo.
    - Aumenta el riesgo de `over-fitting`.
    - Aún realizando `cross-validation`.
- Hay todo un paquete en `sklearn` : `sklearn.feature_selection`

Filtrado basado en las propiedades de la distribución

- Si hay poca variabilidad, no pueden ser usados para distinguir entre clases.
- Podemos utilizar como medidas de variabilidad a la mediana y al inter-quartile range IQR.
- En `sklearn` puedes utilizar `VarianceThreshold`

Filtrado basado en las propiedades de la distribución (Algoritmo)

- Obtenga para cada variable su mediana.
- Obtenga para cada variable sus `quartiles`, en particular, reste el tercer `quartil` del primero, para obtener el `IQR`.
- Realice un *scatter-plot* entre ambas variables, esta gráfica nos da una visión de la distribución de las variables.
- Eliminemos las variables que tengan “baja variabilidad” i.e. que sean menores que un porcentaje del `IQR` global.
    - e.g. $< 1/5$ ó $< 1/6$.
- **¡Cuidado!** Que las variables individuales tengan baja variabilidad, no significa que unidas con otras variables la tengan. Para una posible solución ver “A practical approach to Feature Selection” (http://sci2s.ugr.es/keel/pdf/algorithm/congreso/kira1992.pdf) de Kira and Rendell, 1992.

Correlation Filtering

- Eliminar las variables que estén muy correlacionadas, conservando solo una.
- Problema: ¿Cuál tiras?

    - No hay criterio establecido
    - A veces se puede tirar la mejor …

Fast correlation-based filtering

- Descrito en "Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution (http://pdf.aminer.org/000/335/746/feature_selection_for_high_dimensional_data_a_fast_correlation_based.pdf) de Yu & Liu ICML 2003
- Obtienes un conjunto de variables no muy relacionado entre sí, pero altamente relacionado a la variable de salida.

Fast correlation-based filtering (Algoritmo)

- Encuentra una medida de relación entre cada par de variables.
    - Aquí usaremos la correlación, el artículo usa otra cosa.
- Encuentra la correlación de cada variable con la variable de salida.
- Ordena las variables según su correlación con la variable de salida.
- Elige la mejor variable (la de hasta arriba).
- Tira las variables muy correlacionadas con esta.
- Repite el proceso.

Métodos comúnes usados

- Existen dos métodos utilizados comúnmente para este menester.
- *Forward Selection*
    - El cual inicia sin variables y va agregando una a una las variables, hasta que no mejora la metrica de evaluación.
- *Backward Selection*
    - Empieza con todas las variables en el modelo, y se van removiendo.

Forward selection (Algoritmo)

- Ejecuta el algoritmo con cada variable (i.e. de manera individual)
    - Si tienes x número de variables, ejecutas el algoritmo x veces.
    - Como siempre, usando `cross-validation` .
- Elige el mejor modelo y quédate con esa variable.
- Ahora, ejecuta el modelo de nuevo, pero ahora con la variable recién seleccionada y con cada variable restante.
- Elige el mejor modelo y quédate con esas dos variables.
- Repite hasta que no mejore el modelo agregando más variables.

Backward selection

- *Backward selection* es el mismo algoritmo, pero invertido
    - kind-of …
- En `sklearn` , este algoritmo está implementado con el nombre `RFE` (*Recursive Feature Elimination*)

Filtros **ANOVA**

- Si la variable tiene una distribución similar para los posibles valores de la variable a predecir, seguramente no sirve para discriminar.
- Compararemos la media condicionada a los valores de la variable de salida.
- Para las variables que tengamos una confianza estadística elevada de que son iguales a lo largo de los valores de la variable dependiente, serán descartados.
- Para eso usaremos métodos `ANOVA` .
- En `sklearn` este método está implementado en la sección de “Univariate feature selection” el cuál contiene los métodos `SelectKBest` , `SelectPercentile` , `SelectFpr` (*False positive rate test*), `SelectFdr` (*False discovery rate test*), entre otros. Estos objetos reciben como parámetro la prueba estadística a utilizar, en particular **ANOVA** está implementado mediante `f_classif` .

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

## Generamos 50 columnas aleatorias
boston = datasets.load_boston()

X = boston.data
y = boston.target


random_generator = np.random.RandomState(1234)
noise_cols = random_generator.normal(size=(len(boston.data), 50))
X_noisy = np.hstack([boston.data, noise_cols])
y = boston.target

X_train, X_test, y_train, y_test = train_test_split(X_noisy, y, random_state=0, test_size=.3)

from sklearn.feature_selection import SelectPercentile

select_percentile = SelectPercentile(percentile=10)
select_percentile.fit(X_train, y_train)
X_train_selected = select_percentile.transform(X_train)

## Para generar la imágen
# mask = select_percentile.get_support()
# mask = mask[np.newaxis, :]
# plt.close()
# plt.matshow(mask.reshape(1, -1), cmap='gray_r')
# plt.xlabel("Índice de columnas")
# plt.savefig('../images/select_percentile.png')
```

img

Random Forest

- El algoritmo del `RF` puede ser usado para obtener un rankeo de las variables en términos de su utilidad para la tarea de clasificación.

  library(randomForest) rf <- randomForest(formula, df, importance=TRUE) imp <- importance(rf) rf.vars <- names(imp)[order(imp, decreasing=TRUE)[1:30]] # Gráfica varImpPlot(fmodel, type=1)

- Esto puede ser usado para no tener árboles tan pesados y lentos o pueden ser usados para selección de variables de otros algoritmos (como la regresión logística)
- En `sklearn` se utiliza el objeto `SelectFromModel`

Aglomeración

- Si tenemos muchas variables y muchas muy correlacionadas, podemos formar clústers con ellas.
- Elegir sólo una de cada grupo.
- Se puede combinar con métodos de ensamble.

Evaluación *Off-line*

Referencias

Esta sección está basada en los libros **Practical Data Science with R** y **Evaluating Machine Learning Models** y en otros varios artículos que son citados en el texto.

Generalidades

- Después de construir un modelo, hay que verificar que por lo menos funcione con los datos con los que fué creado (entrenado).
- Para cuantificar el “que funcione” debemos de escoger algunas métricas.
- Los factores que influyen en la elección de la métrica son:
  - La meta del negocio
  - El algoritmo elegido
  - El *dataset*

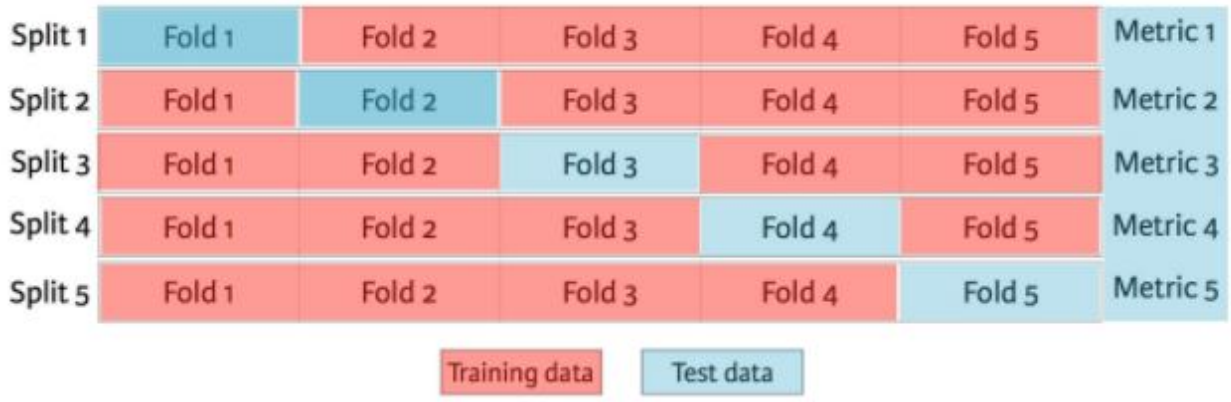
¿Dónde evalúo?

- En la parte de prototipado, en la cual entrenamos, con datos históricos, diferentes modelos para encontrar el mejor (*model selection*)
  - Evaluación *off-line*
  - Su principal objetivo es encontrar el modelo correcto que mejor se adapte a los datos.
- Una vez encontrado el mejor modelo, lo ponemos en **producción** y ahí lo probamos con datos en vivo.
  - Ver la sección de Evaluación *on-line*
    - Una de las técnicas más usadas es **A/B testing**

        - Otra es *multiarmed bandits*

Cosas a tomar en cuenta

- Es probable que las métricas para *off-line* y *on-line* sean diferentes
    - *off-line*: e.g. *precision-recall*
    - *on-line*: e.g. *customer lifetime value*
    - Esto es difícil, ya que se le pide al modelo ser bueno en algo en lo que no fué entrenado (!)
- Es probable que el entrenamiento y la validación tengan diferentes métricas
- Otro posible problema son fuentes de datos *skewed*: desbalanceadas, existencia de *outliers* o con rarezas
- Hay dos fuentes de datos: datos históricos y datos “vivos”
    - Hay que tomar en cuenta el *temporal drift* (estacionalidad)
    - Una manera de crear “nuevos datos” es utilizar varias técnicas como:
        - *hold-out validation*. Es cuando divides los datos en “train” y “test”.
        - *k-fold cross-validation*. El dataset se divide aleatoriamente en k grupos. Un grupo es usado como “test” y el resto como “training”. El modelo se entran y se prueba en cada grupo. El proceso se repite hasta que todos los grupos fueron utilizados como “test”.
        - *bootstrapping*. Tambien conocido como baggging, ayuda a evitar overfitting por medio de la creacion de subconjuntos que son extraidos con remplazamiento.
        - *jacknife resampling*. Funciona mediante el borrado secuencial de una observacion en el dataset y recalculando la estadistica deseada. El objetivo es reducir el sesgo y evaluar la varianza para un estimador.



img

Evaluación

Cada conjunto de tareas como clasificación, regresión, *ranking*, *clustering* etc tienen diferentes métricas.

Evaluación de modelos de clasificación

Accuracy:

- ¿Qué tan frecuente el clasificador hace la predicción correcta?

- ****Pregunta de negocio****: **Necesitamos que la mayoría de las decisiones sean las correctas**.

$$\begin{aligned}\text{accuracy} &= \frac{\text{Número de predicciones correctas}}{\text{Número total de observaciones}} \\ &= \frac{TP + TN}{TP + FP + TN + FN}\end{aligned}$$

- El *accuracy* es un ejemplo de *micro-average*
- No sirve para dataset no balanceados (por ejemplo en detección de fraude).
    - En este caso el modelo nulo es muy preciso (very accurate), pero obviamente esto no lo hace el mejor modelo.
    - Hay que considerar una función de costo.

Matriz de confusión

		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN

img

Una de las desventajas de la métrica de *accuracy* es que no hace distinción entre las clases, y en muchas aplicaciones los **falsos positivos** cuestan diferente que los **falsos negativos**.

Una **matriz de confusión** (Tabla 5) muestra un resumen más detallado de las clasificaciones correctas o incorrectas para cada clase.

| Predicha como positivo | Predicha como negativo |
Etiquetada como positivo | *true positive* (`TP`) | *false negative* (`FN`) |
Etiquetada como negativa | *false positive* (`FP`) | *true negative* (`TN`) |
Esto permite calcular el *accuracy* de cada clase.

Per-class Accuracy

- Esta métrica es el promedio del *accuracy* de cada clase.
- Esto es un ejemplo de un *macro-average*

Evaluación de modelos de probabilidad

- Sirven para clasificación o para regresión.
- Indican la probabilidad estimada (confianza) de que la observación pertenezca a una clase.
- Hay que elegir un `cut-off` .

Log-loss

- En términos sencillos mide que tan equivocado está la clasificación
    - e.g. /Si la etiqueta verdadera es 0 y el clasificador dice que es 1 con una probabilidad de 0.51 y la frontera de decisión es 0.50, es un “near-miss”.
- *Log-loss*, entonces, se puede considerar como una medida “suave” de *accuracy*.

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log p_i + (1-y_i)\log(1-p_i)$$

Para un problema de clasificacion, un valor de Log-loss bajo significa mejores predicciones.

Ejemplo:

Un modelo predice la probabilidad de venta de 3 casas: [0.8, 0.4, 0.1] Las primeras dos casas se vendieron y la tercera no. Los resultados se representan como [1, 1, 0]

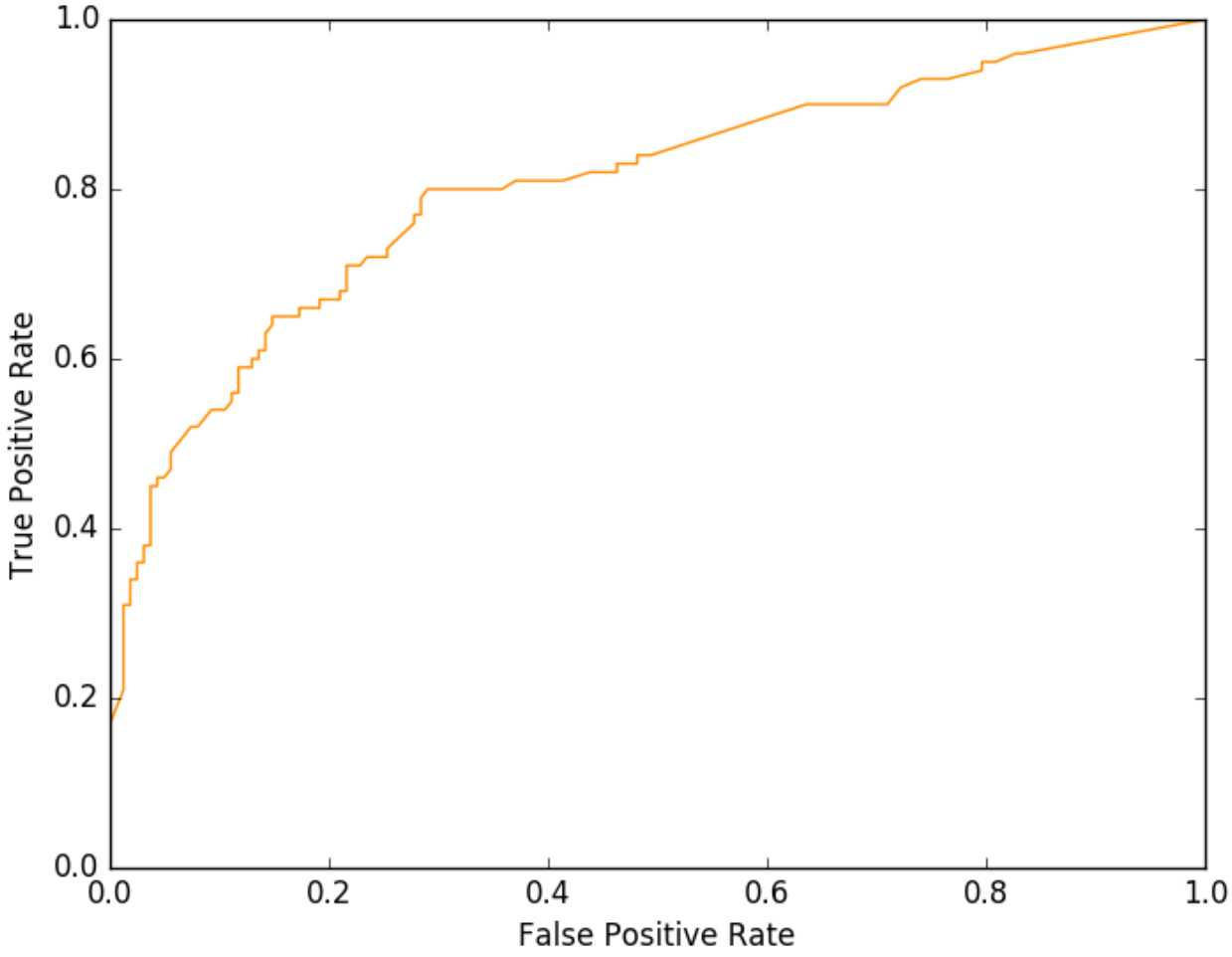
Calulamos la funcion de verosimilitud: La primera casa se vendio y el modelo dijo que era 80% probable. Entonces, la probabilidad despues de mirar una prediccion es 0.8.

La segunda casa se vendio y el modelo dijo que asi seria con 40% de probabilidad. Existe una regla de probabilidad de que la probabilidad de multiples eventos independientes es el producto de sus probabilidades individuales. Entonces, obtenemos la probabilidad combinada de las dos primeras predicciones multiplicando sus probabilidades asociadas. Eso es 0.8 * 0.4, es decir 0.32.

Ahora llegamos a nuestra tercera prediccion. Esa casa no se vendio. El modelo dijo que tenia un 10% de probabilidades de venderse. Eso significa que tenia un 90% de probabilidades de no venderse. Entonces, el resultado observado de no vender fue 90% probable segun el modelo. Entonces, multiplicamos el resultado anterior de 0.32 por 0.9.

AUC y **ROC**

- **ROC** es un tipo de curva y sus iniciales significan *receiver operating characteristic curve* (Figura 274).
- La curva **ROC** muestra la sensibilidad del clasificador, al dibujar la tasa de verdaderos positivos (`TRP`) contra la tasa de falsos positivos (`FPR`).
- Se volvió popular por el artículo *Basic Principles of ROC Analysis* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.692.1962) de **Charles Metz** en 1978
    - Desafortunadamente el artículo no se puede consultar en línea, pero una buena explicación está aquí (http://mlwiki.org/index.php/ROC_Analysis) (texto) o aquí (http://bit.ly/roc-auc) (vídeo).
- Aunque debería de ser obvio, la curva **ROC no** es un *número*, es toda una *curva*
- **AUC** significa *área bajo la curva* donde la curva es la curva **ROC**
- **AUC** es una manera de resumir la curva **ROC**



img

Precision

- ****Pregunta de negocio****: **Lo que marquemos como x, más vale que sea x.**
- ¿Qué fracción clasificada por el modelo están en la clase? Ejemplo: Si el modelo tiene una precision de 0.5, entonces, cuando predice que un tumor es maligno, acierta el 50% de las veces.
- Cuando el modelo dice que el *data point pertenece a la clase*, que tan frecuentemente le atina.
- La proporción de observaciones clasificadas como C correctamente de todas las que se clasificaron como C
- Mide la capacidad del sistema de rechazar aquellas observaciones no relevantes en el conjunto clasificado
- La precisión se ve afectada por la cantidad de *falsos positivos* (el sistema clasificó una observación erróneamente)

$$\text{prec} = \frac{TP}{TP + FP}$$

Recall

- ****Pregunta de negocio****: **Queremos reducir x por en un tanto por ciento**.
- ¿Qué fracción que están en la clase fueron detectadas por el modelo? Ejemplo: Si el modelo tiene una recuperacion de 0.11, entonces identifica correctamente el 11% de los tumores malignos
- Qué tan frecuentemente el clasificador encuentra lo que debe de encontrar.
- La proporción de observaciones clasificadas como C de todas las posibles que podían ser C.

- Mide la capacidad del sistema de encontrar todas las observaciones relevantes
- El *recall* se ve afectado por la cantidad de *falsos negativos* (el sistema falló al clasificar una observación relevante)

$$\text{rec} = \frac{TP}{TP + FN}$$

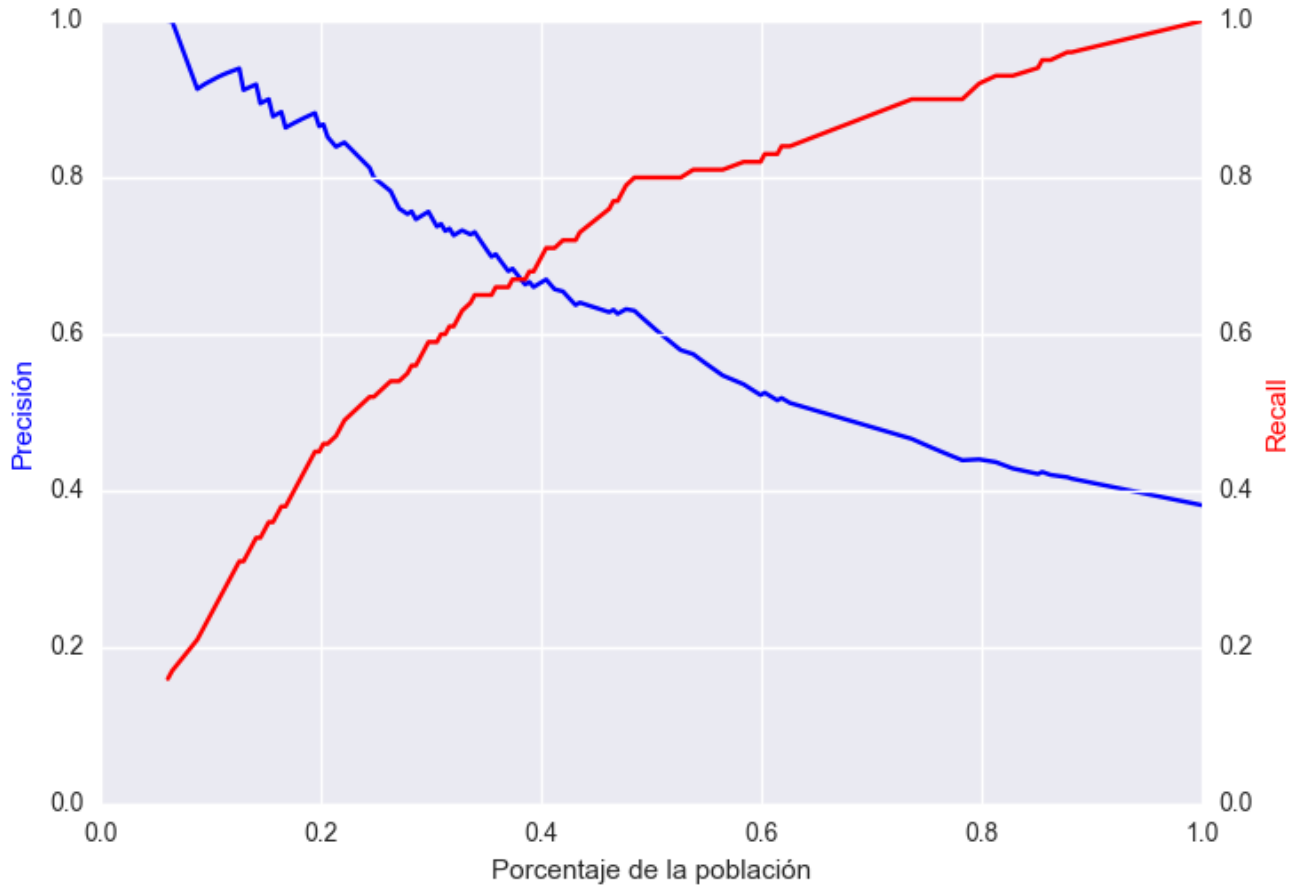
Precision/Recall

- Regularmente son utilizados juntos, y son una métrica utilizada para *ranking* y clasificadores
- En el caso de un *ranker*, puede ser de interés sólo fijarse en los primeros k -items devueltos, Lo cual nos define las medidas *precision@k (mailto:precision@k)* y *recall@k (mailto:recall@k)*.
- De manera análoga a la curva **ROC** se puede definir la curva *precision-recall*.

Ejercicio Crea una función para generar la gráfica de **precision/recall**, llámale `plot_prec_rec`

Precision-Recall@k (mailto:Precision-Recall@k) curve (a.k.a as Rayid's plot)

Créditos: Rayid Ghani@U Chicago (http://github.com/rayidghani/magicloops=)



img

F1 score

- Se usa en conjunto con `precision` y `recall` .

- Mide el sacrificio de `recall` y/o `precision` uno respecto al otro.
- Es una manera de resumir la curva de *precision-recall*
    - Análogo al **AUC** para la curva **ROC**

$$\text{F1} = \frac{2 * \text{prec} * \text{rec}}{\text{prec} + \text{rec}}$$

Sensitivity

- ****Pregunta de negocio****: **Necesitamos reducir la clase x o no hay negocio**.
- Conocida como `true positive rate` (TPR) es igual al `recall`.

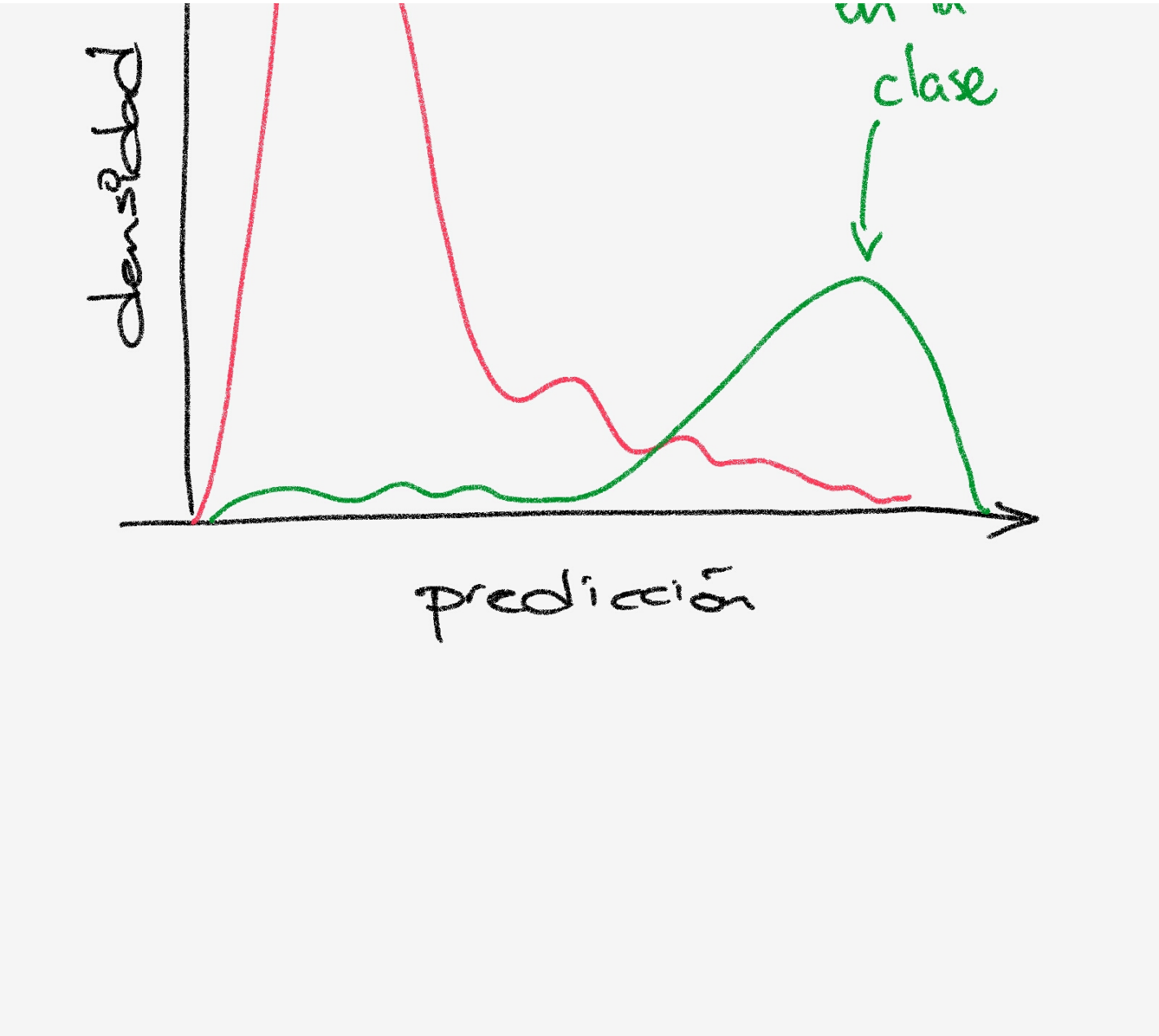
Specificity

- ****Pregunta de negocio****: **No podemos equivocarnos en $\sim x$, el sistema (o el usuario) deben de tener este servicio altísimo**.
- Conocida también como `true negative rate`

$$\text{spec} = \frac{TN}{TN + FP}$$

- El modelo nulo regularmente clasifica con 0 en una de los dos, por lo que los modelos que no sirven, tienen muy bajo uno de estas métricas.

Gráfica de doble densidad



img

Ejercicio Crea una función para generar esta gráfica, llámale `plot_double_density`

Evaluación de modelos de regresión

Residuos es la palabra clave.

- Diferencia entre nuestras predicciones $\hat{y}$ y los valores reales de salida y.

RMSE: *Root mean square error*

- ****Pregunta de negocio****: **Queremos un error (en promedio) menor de tantos miles por unidad estudiada**.
- Se puede pensar como una desviación estándar.
- Está en las mismas unidades que y.

$$\mathbf{RMSE} = \sqrt{\frac{\sum_i (y_i - \hat{y}_i)^2}{n}}$$

- **RMSE** tiene problemas, en particular es sensible a *outliers* (ya que es un promedio)
  - Es decir no es una medida robusta (en el sentido estadístico)
- Para tratar de resolver esto se pueden utilizar **cuantiles** o **percentiles** del error, por ejemplo **MAPE** la mediana (el percentil 50%) del porcentaje del valor absoluto de los errores:

$$\mathbf{MAPE} = \text{median}\left(\left|\frac{y_i - \hat{y}_i}{y_i}\right|\right)$$

R^2

- ****Pregunta de negocio****: **Queremos un modelo que explique tanto porcentaje del valor de tal**.
- 1.0 menos cuanta varianza no estamos explicando por el modelo.

$$1 - \frac{\sum(\hat{y} - y)}{[\sum(\bar{y} - y)]^2}$$

- No tiene unidades.
- Cerca de cero o negativa significa que el modelo es lo peor que nos pudo pasar.

Evaluación de modelos de *clustering*

- Son difíciles de evaluar $\rightarrow$ verificar resumenes de la clusterización.
- Número de *clusters*
- Número de observaciones por cluster.
  - *hair clusters* : Muy pocas observaciones
  - *waste clusters*: Muchos puntos
- Compactos
  - Comparar la distancia entre dos puntos en el cluster con la distancia típica entre dos clusters.

Selección de modelos

Selección de modelos

Durante la etapa de prototipado, estamos ajustando todo en el modelo: *features*, tipos de modelo, métodos de entrenamiento, hiper-parámetros etc. Por cada cambio generamos un nuevo modelo.

Selección de modelos (*model selection*) es el proceso de seleccionar el modelo (o el tipo de modelo correcto) que mejor se adapta a los datos

Después de entrenar el modelo, se debe de ejecutar pruebas en un *dataset* que sea *estadísticamente independiente* del usado para entrenar. Esto nos dará una estimación del **error de generalización**.

Hold-out validation

- Suponiendo que todas las observaciones son i.i.d., seleccionamos una parte de los datos y la separamos para hacer la validación (Figura 343)
- Computacionalmente es la más fácil de hacer y la más rápida en ejecutar.

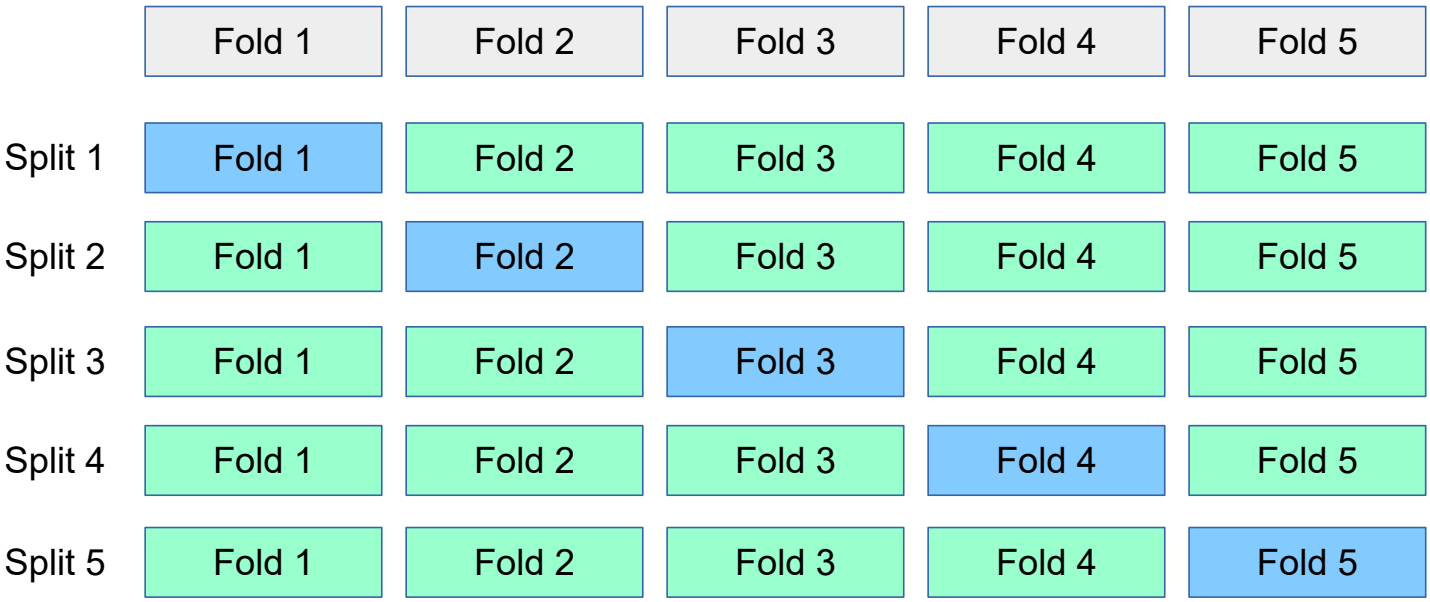
img

```
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=1234)

print(X_train.shape, X_test.shape)
```

Cross-validation

- Es un método estadístico para evaluar la generalización del desempeño del modelo.
- Hay muchas variantes, la más común es **k-fold cross-validation** (Figura 352)
- En **k-fold cross-validation** se divide el conjunto de entrenamiento en k pedazos. Para cada conjunto de hiperparámetros, un k -pedazo será usado como el conjunto de validación y los $k-1$ -pedazos restantes serán usados para entrenamiento (Figura 353)
- El desempeño del modelo se toma como el promedio del desempeño en los k -pedazos
- Una de las ventajas es la siguiente: Si hacemos *hold-out* tendremos, por ejemplo, 80% de los datos para entrenar y 20% para probar, si hacemos *10-fold cross-validation* tendremos 90% de los datos para entrenar y 10% para probar.
- La desventaja más grande es el costo computacional
- Otra variante es *leave-one-out cross-validation* (LOOCV), es casi lo mismo que **k-fold cv** pero en este caso $k=n$.
- **¡Cuidado!** Cross-validation no es una manera de construir un modelo que se puede aplicar a nuevos datos, es una manera de evaluar modelos.

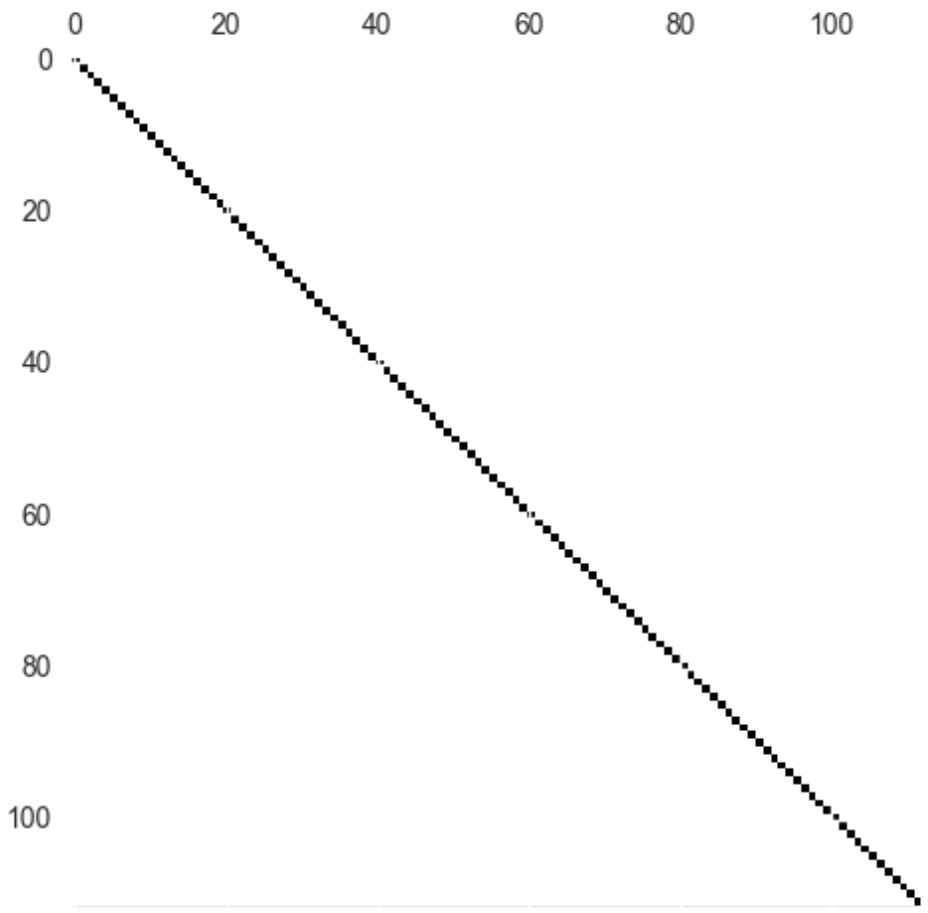
img



img

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

from sklearn.model_selection import cross_val_score
scores = cross_val_score(classifier, X, y, cv=5)

print(scores)  ## Todos los scores

print(np.mean(scores)) ## El promedio de los scores

# Si queremos controlar los folds podemos usar KFold, StratifiedKFold, ShuffleSplit, LeaveOneOut
# Algunos de ellos se muestran en las imágenes siguientes
```

img



img

img

Bootstrap y *Jacknife*

- *Bootstrap* es una técnica de remuestreo: Genera varios *datasets* muestreando repetidamente del conjunto de datos original. Este muestreo es con **reemplazo**.
- Queremos hacer el muestreo con reemplazo, ya que no queremos cambiar la distribución empírica de los datos.
- *Jacknife* inspiró a *bootstrap* y es muy parecido a LOOCV

Temporal cross-validation

La manera **canónica** de hacer *temporal cross-validation* está descrita en este *blog post* (http://robjhyndman.com/hyndsight/crossvalidation/) de Rob J. Hyndman. Básicamente es como sigue:

Supón que tienes n observaciones temporales (con el ordenamiento correcto) yquieres hacer k - fold cross-validation con $k = 4$. Divide tu data set en 5 pedazos, llámalos n_i con $i \in \{1, 2, 3, 4, 5\}$. Entonces, tus *folds* serían:

fold	entrenamiento	prueba
1	n_1	n_2
2	n_1, n_2	n_3
3	n_1, n_2, n_3	n_4
4	n_1, n_2, n_3, n_4	n_5

Temporal cross-validation: hv - cross-validation

Existe otra técnica descrita en el artículo: *Consistent cross-validatory model-selection for dependent data: hv-block cross-validation* J. Racine, Journal of Econometrics **2000**

En esta aproximación se dejan v observaciones para prueba y se borran h observaciones de cualquier lado de los datos de prueba.

fold	entrenamiento	prueba
1	n_1, n_2, n_3, n_4^h	n_5
2	$n_1, n_2, n_3^h, ^h\, n_5$	n_4
3	$n_1, n_2^h, ^h\, n_4, n_5$	n_3
4	$n_1^h, ^h\, n_3, n_4, n_5$	n_2
5	$^h n_2, n_3, n_4, n_5$	n_1

La justificación de esta aproximación se basa en que tiene que haber una independencia entre el conjunto de prueba y el de entrenamiento para que cross-validation pueda funcionar. Dado que en datos temporales, datos adyacentes son (pueden) dependientes entre sí, cross-validation normal no se puede aplicar, pero si se deja un **hueco** entre los datos de prueba y los de entrenamiento a lo largo de **ambos** lados del conjunto de prueba.

Más lecturas recomendadas

- **A survey of cross-validation procedures for model selection** (https://projecteuclid.org/euclid.ssu/1268143839) de Arlot y Celisse.

Hyperparameter tuning

Hyperparameter tuning

- El ajuste ó selección de hiper-parámetros es una meta-tarea de aprendizaje

- Un *parámetro del modelo* es algo que es aprendido durante la fase de entrenamiento
  - e.g. el vector de pesos $\vec{w}^T$ en una regresión lineal $\vec{w}^T \cdot \vec{x} = \vec{y}$
- Los hiper-parámetros deben de ser especificados fuera del procedimiento de entrenamiento.
  - e.g. La profundidad de los árboles en los árboles de decisión, o el número de árboles en un **RF**.
- Debido a que el proceso de entrenamiento no establece (o fija) los hiperparámetros, tiene que haber un meta-proceso que se encargue de hacerlo, este es el proceso de *hyperparameter tuning*.
- La salida de este proceso es el mejor conjunto de hiper-parámetros
- Para evitar *over-fitting* necesitamos no utilizar el *test dataset* para ajustar los parámetros, ahora tendremos tres conjuntos: *training*, *validation* y *test*.

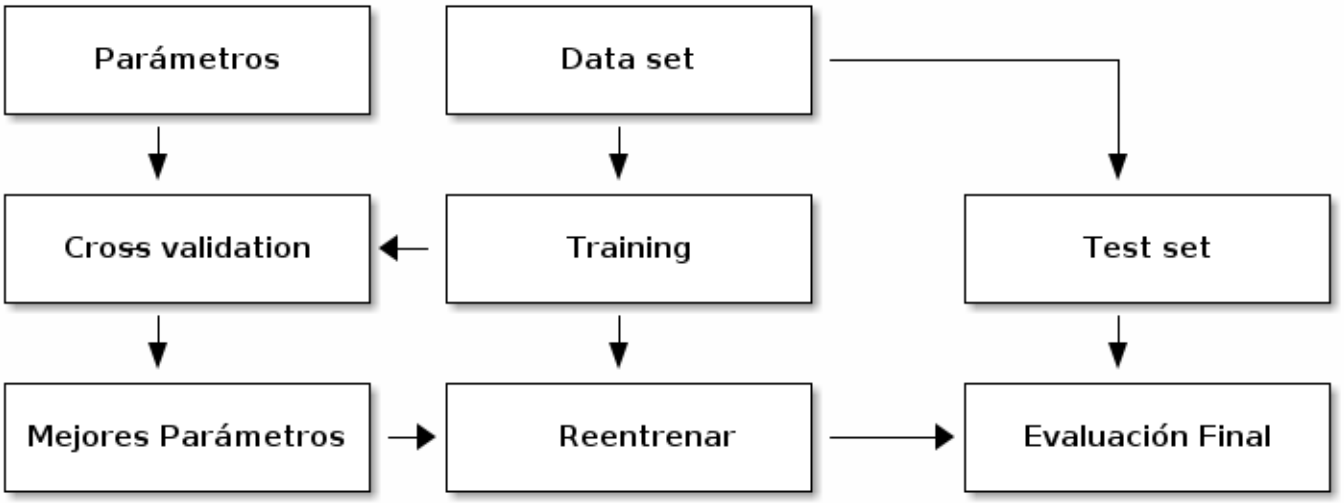
img

Recuerda

Los modelos son como ganado, no mascotas

Procedimiento

1. Divide el *dataset* en *training* y *testing*
2. Para cada conjunto de hiperparámetros,
   a. Genera un modelo con esos hiperparámetros b. Evalúa el modelo (usando *hold-out* o *cross-validation*) Esto genera *training* y *validation*
3. Selecciona el mejor conjunto de hiperparámetros.
4. Entrena el modelo con esos hiperparámetros usando **todo** el *dataset* (*training* y *validation*)



img

Algoritmos: *Grid Search*

Grid search Dado una rejilla de hiperparámetros, evalúa cada uno de ellos y regresa un ganador.

- Simple y fácil de paralelizar (*embarrassingly parallel*)
- En tiempo de ejecución es el más caro

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier() hyper_param_grid = {'n_estimators': [1,10,100,1000,10000], 'max_depth': [1,5,10,20,50,100], 'max_features': ['sqrt','log2'],'min_samples_split': [2,5,10]},

from sklearn.grid_search import GridSearchCV

grid_search = GridSearchCV(classifier, hyper_param_grid, cv = 5, verbose = 3)

grid_search.fit(X, y)

grid_search.best_params_

grid_search.best_score_

Para eviatr overfit

from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y)

grid_search = GridSearch(classifier, hyper_param_grid, cv=5) grid_search.fit(X_train, y_train)

grid_search.predict(X_test)

grid_search.score(X_test, y_test)

O puedes usar de nuevo cv

cross_val_score(grid_search, X, y, cv = 5)

Algoritmos: *Random Search*

Random search, es una pequeña variación del *Grid search*: sólo evalúa al azar un muestreo de puntos de la rejilla.

- Ve el artículo de **Random Search for Hyper Parameter Optimization** (http://www.jmlr.org/papers/v13/bergstra12a.html) de Bergstra y Bengio.
- Funciona casi también como el *Grid search*, probando, aproximadamente 60 puntos al azar de la rejilla.
    - El *caveat* al menos el 5% de los puntos de la rejilla deben de estar cerca de la solución óptima.
- Es fácil de paralelizar y fácil de codificar
- En ~ scikit-learn~ esta clase se llama `RandomizedSearchCV`

Ejercicio Repite el código anterior pero ahora usa `RandomizedSearchCV`

Algoritmos: *Smart hyperparameter search*

Smart hyperparameter search. Son procesos secuenciales: escogen unos hiperparámetros, evaluan su calidad y deciden en que región muestrear.

- Estos no son paralelizables (por lo menos no de manera trivial)
- Muchos de estas técnicas usan parámetros que deben de ser ajustados
- Las tres variantes más importantes son: *Derivative free optimization* (http://epubs.siam.org/doi/book/10.1137/1.9780898718768), *Bayesian optimization* (https://arxiv.org/abs/1206.2944) y *Random Forest smart tuning* (https://www.cs.ubc.ca/~hutter/papers/10-TR-SMAC.pdf).
- Estas técnicas no las veremos en este curso, pero son muy importantes.

Nested cross-validation

¿Qué tal si queremos seleccionar entre diferentes familias de modelos?

- Para cada modelo en la lista de modelos
    a. Divide el *dataset* en *training* (A) y *meta-evaluation* (B)
    b. Subdivide el set de training de nuevo en *training* (C) y *validation* (D)
    c. Genera una lista de hiperparámetros para este modelo
    d. Ejecuta el ajuste de hiperparámetros para seleccionar los mejores hiperparámetros
    e. Evalúa el mejor modelo en B
    f. Guarda la evaluación en una lista (o diccionario) y haz lo mismo con el mejor conjunto de hiperparámetros
- Selecciona el mejor modelo y el mejor conjunto de hiperparámetros
- Entrena el mejor modelo, con sus hiperparámetros en todo el *dataset*
- Regresa la mejor evaluación, los mejores hiperparámetros y el modelo entrenado en todo el *dataset*.

Un ejemplo de implementación: Rayid’s magicloop

Créditos: Rayid Ghani@U Chicago (http://github.com/rayidghani/magicloops=)

```
def define_hyper_params():
    """
        Esta función devuelve un diccionario con
        los clasificadores que vamos a utilizar y
        una rejilla de hiperparámetros
    """
    ## Por ejemplo
    ## classifiers = {
    ##     'RF': RandomForestClassifier(n_estimators=50, n_jobs=-1),
    ##     'NB': GaussianNB(), ...
    ## }

    ## grid = {
    ##     'RF': {'n_estimators': [1,10,100,1000,10000],
    ##            'max_depth': [1,5,10,20,50,100],
    ##            'max_features': ['sqrt','log2'],
    ##            'min_samples_split': [2,5,10]
    ##     },
    ##     'NB': { ... },
    ##     ...
    ## }

    return classifiers, grid
```

```
def magic_loop(models_to_run, clfs, grid, X, y):
    for n in range(1, 2):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
        for index, clf in enumerate([clfs[x] for x in models_to_run]):
            logger.debug(models_to_run[index])
            parameter_values = grid[models_to_run[index]]
            for p in ParameterGrid(parameter_values):
                try:
                    clf.set_params(**p)
                    logger.debug(clf)
                    y_pred_probs = clf.fit(X_train, y_train).predict_proba(X_test)[:,1]
                    logger.debug(precision_at_k(y_test,y_pred_probs,.05))
                    #plot_precision_recall_n(y_test,y_pred_probs,clf)
                except IndexError as e:
                    print('Error:', e)
                    continue
```

```
def precision_at_k(y_true, y_scores, k):
    threshold = np.sort(y_scores)[::-1][int(k*len(y_scores))]
    y_pred = np.asarray([1 if i >= threshold else 0 for i in y_scores])
    return metrics.precision_score(y_true, y_pred)
```

Pipeline de Modelos

Algunos pasos antes de modelar

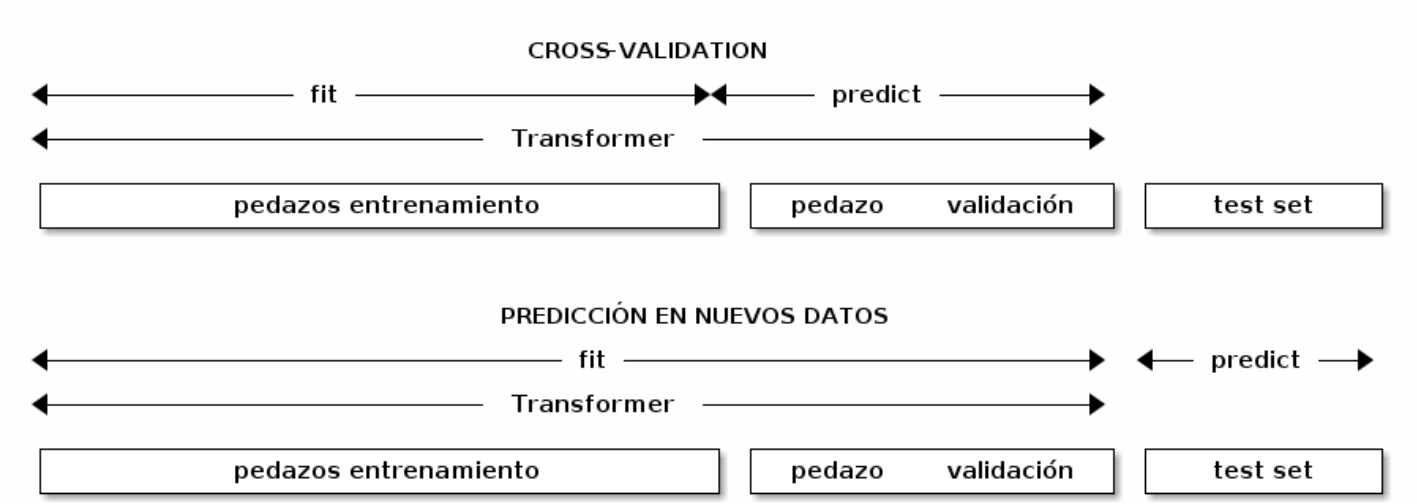
- *Data sampling*
- Crear nuevas variables.
- Discretizar variables cuantitativas.
- Convertir a numéricas las variables cuantitativas.
- Manejo de variables de fecha.
- Unir (`merge`), ordenar, reshape los conjuntos de datos
- Cambiar las variables categóricas a múltiples variables binarias.
- Resolver que se hará con los datos faltantes.
- Escalamiento y normalización, otras transformaciones.
- Reducción de dimensionalidad.
    - PCA, *Factor Analysis*, *Clustering*, MCA, CA, t-SNE, etc.
- etc.

Pipelines básico

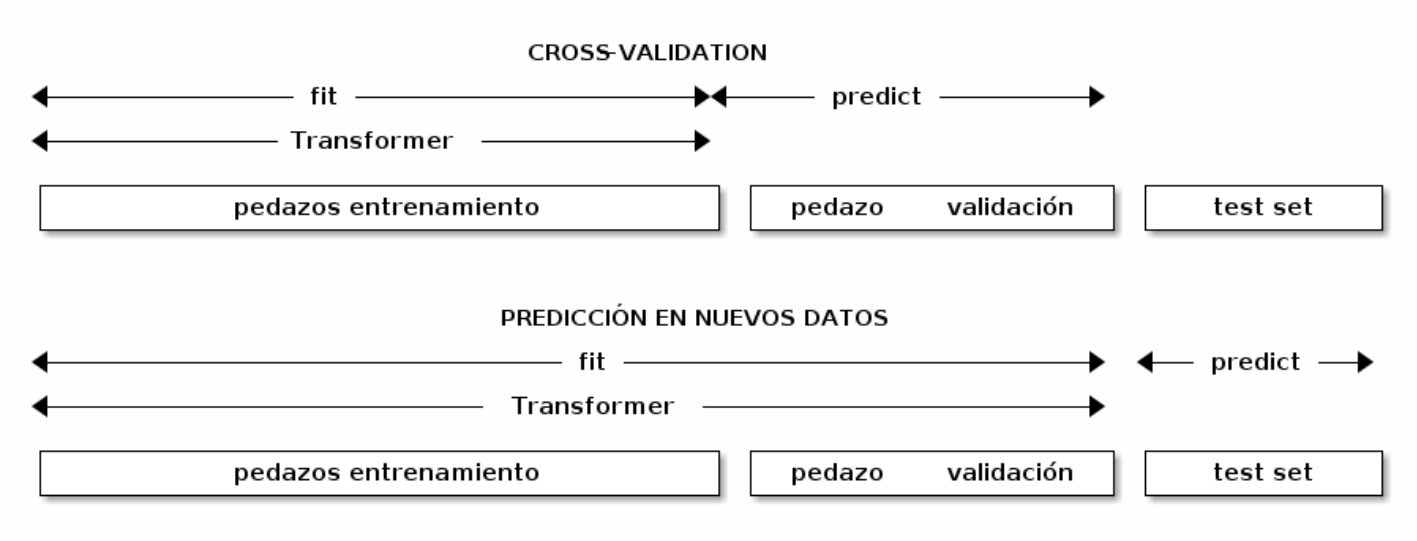
El objeto `Pipeline` en `scikit learn` nos permite encadenar varias transformaciones y para luego ser reutilizadas

```
## Versión larga
from sklearn.pipeline import Pipeline
pipe = Pipeline([("my_scaler", StandardScaler()), ("my_svm", SVC())])
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)

## Versión corta
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(StandardScaler(), SVC())
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Además, si queremos usar `cross-validation` nos permite evitar problemas como el mostrado en la figura 429 (compara con la figura 430) cuando se estén creando los diferentes pasos para el modelo



img



img

Ejemplo de *information leakage*

Tomado de **Hastie, Tibshirani, and Friedman**, *The Elements of Statistical Learning*

```
# coding: utf-8
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge

## Generemos un data set de 100 observaciones y 1000 variables
## X y y serán independientes
rnd = np.random.RandomState(seed = 1234)
X = rnd.normal(size=(100, 10000))
y = rnd.normal(size=(100,))

## Recuerda que no deberíamos de aprender nada de este dataset

## Seleccionamos el percentil 5  de las variables
select = SelectPercentile(score_func=f_regression, percentile=5).fit(X,y)
X_selected = select.transform(X)

"R^2 Cross-Validation: {:.2f}".format(np.mean(cross_val_score(Ridge(), X_selected, y, cv=5)))
```

R^2 Cross-Validation: 0.91

Esto obviamente no debería de estar pasando. Lo que ocurre es que el *feature selection* tomó algunas de las 10,000 variables que estaban (por puro azar) muy correlacionadas en *training* y *test*. Hagámoslo bien ahora:

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([("select", SelectPercentile(score_func=f_regression, percentile=5)),
                 ("ridge", Ridge())])

"R^2 Cross-Validation: {:.2f}".format(np.mean(cross_val_score(pipe, X, y, cv=5)))
```

R^2 Cross-Validation: -0.10

Este resultado tiene mucho más sentido. Ahora, el *feature selection* se realizó **dentro** del *cross-validation*

Referencias sobre *information leakage*

Para más información sobre *information leakage* ver aquí (https://www.kaggle.com/c/the-icml-2013-whale-challenge-right-whale-redux/forums/t/4865/the-leakage-and-how-it-was-fixed), aquí (http://machinelearningmastery.com/data-leakage-machine-learning/) ó en los siguientes artículos: n

- *Leakage in Data Mining: Formulation, Detection, and Avoidance* S. Kaufman et al. **2011**
- *No unbiased estimator of the variance of k-fold cross-validation* Y. Bengio

and Y. Grandvalet **2004**

- *A study of cross-validation and bootstrap for accuracy estimation and model selection* R. Kohavi. **1995**
- *On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation* G.C. Cawley and N. L. C. Talbot **2010**
- *Medical data mining: insights from winning two competitions* S. Rosset, C. Perlich, G. Świrszcz, P. Melville and Y. Liu **2009**

Pipelines con *Hyperparameter search*

```
grid = GridSearchCV(pipe, hyper_param_grid, cv)
grid.fit(X_train, y_train)
grid.best_score_
grid.score(X_test, y_test)
grid.best_params_
```

Ejercicio

- Modifica tu `magic_loop` para reescribirlo en términos de `Pipeline`
- Agrega pasos para crear más variables usando polinomios, realiza *binning* en la edad

Producción

Tipos de producción

Batch

Se le pasan los datos al modelo, se realiza el *scoring* y este *score* se escribe a base de datos, archivo, etc.

El punto de contacto es la base de datos.

Ligar con otros lenguajes

- Otros lenguajes (`C/C++` , `Java` , `Python` , `clojure` , `bash` , etc.) se conectan al lenguaje usado en el modelo, usando las ligas o API implementadas en el lenguaje (e.g. `Rpy` , `Rcpp` (http://www.rcpp.org)) y continúan con su ejecución.

Exportar

- A veces, la evaluación del modelo es simple comparada con la construcción del modelo. En este caso, es posible (o deseable) transformar la evaluación a otro lenguaje (`SQL` , `Java` , `etc`)
    - e.g. para árboles: Mi respuesta en Stackoverflow (http://stackoverflow.com/questions/11831794/testing-rules-generated-by-rpart-package)
    - `PMML`
        - **Predictive Model Markup Language**.
        - Formato en `XML` .
        - Promovido por el Data Mining Group.
        - Depende del paquete `pmml`
        - Se exporta a otra herramienta que soporte el estándar.
        - Funciona en ambas direcciones.
        - Quizá nunca funcionó

Servicios Web

- El modelo es expuesto a través de un servicio `HTTP`
    - En `R` se puede utilizar el paquete `plumber` (https://github.com/trestletech/plumber)
    - En `python` usando `Flask` (http://flask.pocoo.org/) (ejemplos (http://blog.luisrei.com/articles/flaskrest.html))

from flask import jsonify, request, Flask from sklearn.externals import joblib

Leemos el modelo desde el binario (ver más adelante)

model = joblib.load('model.pkl')

Creamos el servicio web

app = Flask(**name**)

Sólo un método "predict" que está ligado a la dirección http://0.0.0.0:5000/ (http://0.0.0.0:5000/)

En este caso recibe el vector que recibe es un texto en json

y regresa un json con los resultados

@app.route('/', methods=['POST']) def predict(): text = request.form.get('text') results = {} for name, clf in models.iteritems(): results[name] = clf.predict([text])[0] return jsonify(results)

if **name** == '**main**': app.run()

Se puede invocar con

curl -H “Content-type: application/json”

-X POST http://0.0.0.0:5000/ (http://0.0.0.0:5000/) -d ‘{“text”:“Hello Data”}’

Los pasos a seguir luego del entrenamiento y selección del modelo son:

- Guarda el modelo entrenado (como `rds` para `R` / como `pickle` o como `sklearn.externals.joblib`)

  import pickle

model es el modelo entrenado

Guardarlo a archivo

with open(“model.pkl”, “wb”) as f: pickle.dump(model, f)

Cargarlo de nuevo

with open(“model.pkl”, “rb”) as f: model_loaded = pickle.load(f)

- Crea un servicio web (`plumber` en `R` , `Flask` en `python`) que use el modelo en binario y que tenga un método que reciba el vector de datos (o *features*).
- Este método responde con un `json` que incluye el vector de datos de entrada y la respuesta (predicción) más otra meta-data.

Ejercicio

- Crea un servicio web que despliegue el mejor modelo de `Titanic` . Este servicio debe de recibir un vector de datos que y regrese la probabilidad de supervivencia (si aplica).

Problemas

- Los modelos necesitan ser entrenados, actualizados y desplegados (*deployed*) sin mucho problema
- La mayoría de las fuentes de datos que alimentan a los modelos no son controlados por ustedes
- Varios tipos de modelos
- Varios lenguajes de programación son necesarios
  - `R` , `python` , `SQL` , `spark` , etc.

Más problemas

Más que problemas son suposiciones con las que muchos modelan:

- El mundo no sabe que están tratando de modelarlo y por lo tanto no toma contra-medidas (No hay *adversarios*)
- El modelo no tiene ningún efecto en el mundo de ningún tipo (No hay adaptación, *temporal drifting* y obvio no hay consecuencias éticas)

Adversarial Learning

- Ve por ejemplo:
  - Huang, Ling, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. **2011**. *Adversarial Machine Learning.* IEEE Internet Computing 15 (5): 4–6. <10.1109/MIC.2011.112>.
  - Laskov, Pavel, and Richard Lippmann. **2010**. *Machine Learning in Adversarial Environments.* Machine Learning. <10.1007/s10994-010-5207-6>.
  - También es importante este libro: Cesa-Bianchi, Nicolò, and Gábor Lugosi. **2006**. *Prediction, Learning, and Games.* Cambridge University Press.
- En cierta medida, uan vez que está en producción, tu eres un adversario.
  - Ver por ejemplo, el escenario de *Alyssa Frazee* aquí (http://www.win-vector.com/blog/2016/09/adversarial-machine-learning/)

Temporal drift

- Conforme pasa el tiempo, los datos cambian
  - El mundo cambia con el tiempo, y así nuestro modelos sobre ese mundo
  - **Hackeo** de los modelos (**adversarial domains**) intencionales
- ¿Cómo identificamos que hubo **drift**?
  - De tal manera que podamos actualizar los modelos, esto es necesario ya que el patrón cambió
  - Esto es un área activa de investigación
- ¿Ideas?
  - Modelos de ensamble con diferentes ventanas de tiempo de entrenamiento
  - Reentrenamiento frecuente
  - Comparación de modelos: entrenamos con datos actuales contra datos históricos quizá permitan detectar **drift**

Reproducibilidad

- Todos los experimentos y los modelos productivos deben de ser reproducibles.
  - Más difícil de lo que parece…
  - Requiere código histórico, datos históricos, formatos originales, **feature extraction**, etc.
- Si hay **adversarial domains** hay que saber que no evolucionan todos iguales.

Gobernanza de Modelos

- Ventanas de entrenamiento
- Publicación de modelos
  - ¿ `PMML` ? ¿Serialización? ¿Imágenes de `Docker` ?
- Selección de modelos

Almacenando los modelos

Dado lo anterior es importante almacenar **todo**:

- Datos de entrenamiento, validación y prueba
- Hiper-parámetros
- El modelo
- Los resultados: métricas

- Desempeño en producción

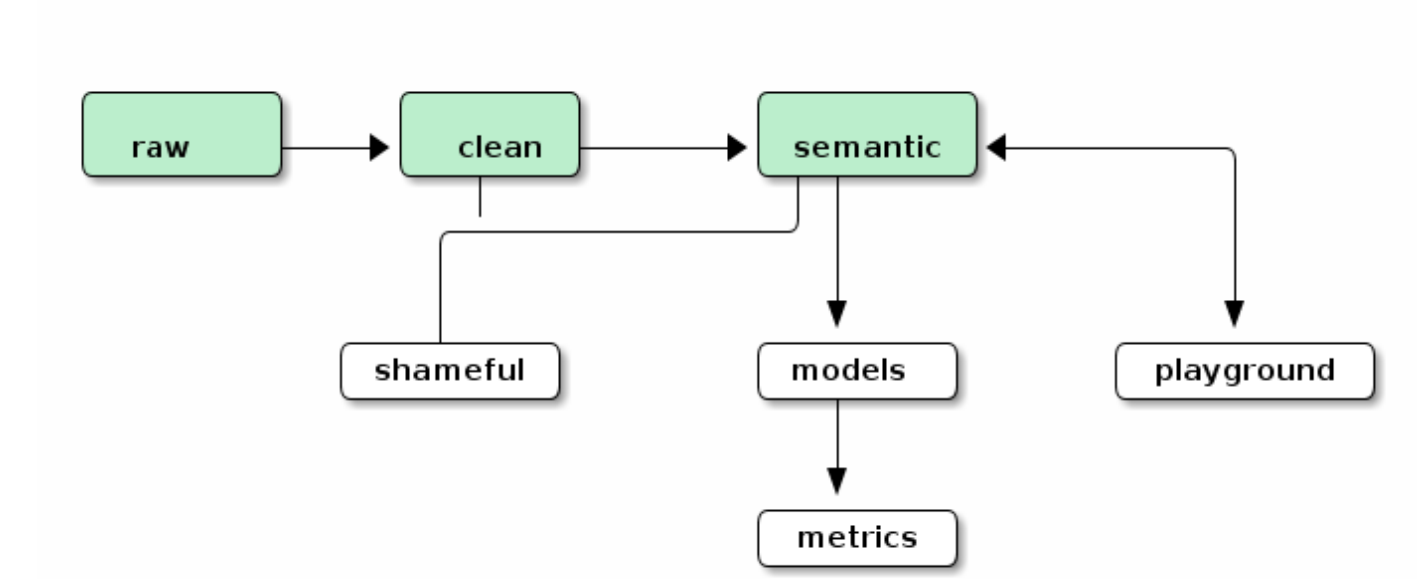
Recordando a nuestra base de datos relacional, teníamos los siguientes *esquemas*:

- `raw` : Datos *as-is*
- `clean` o `gold` : Datos limpios
- `semantic` : Escoge el objeto(s) de tu **modelo**, unifica en una tabla *tidy* (si aplica).

Vamos a agregar:

- `features` : Nuevas variables, variables *transformadas* y variables *derivadas*
- `shameful` o `burning_bus` (https://www.explainxkcd.com/wiki/images/d/d0/code_quality_2.png): Todos aquellos *hacks* que algún día nos alcanzarán
- `experiments` o `playground` : Experimentos y transformaciones de datos
- `models` : Modelos, *metadatos* de los modelos, conjuntos de entrenamiento, etc.
- `metrics` : Resultados de los modelos *on-line* y *off-line*
- `ontology` [Opcional]

Es importante recordar que esto es un diseño conceptual basado en una base de datos relacional, el siguiente semestre veremos *data lakes* y será un poco diferente, pero la esencia es la misma.



img

Almacenando los modelos: Metadatos

Ejemplo de metadata de experimento

```
{
    "tiempo": "2016-10-20 02:36:45",
    "proceso": "",
    "modelo": "",
    "version": "",
    "parametros": {  },
    "test_id": "UUID",
    "output": ["0.98227654 de average accuracy sobre 15 resultados"],
    "resultados": "",
    "duracion": {
        "entrenamiento": 78990,
        "prueba": 1340
    },
    "query": "select * from ...",
    "records": 456987123
}
```

¿Otras ideas?

Evaluación *On-line*

Baseline

- Utilizaremos algunos modelos idealizados:
  - Modelo nulo
    - Nos enseña cuál es el mínimo.
  - Modelo `Bayes rate`
    - Indica cuál puede ser el máximo posible.
  - El mejor modelo de una sóla variable.
    - Indica que es lo mejor que puede hacer un modelo simple.

Modelo nulo

- Es aquel modelo que quieres vencer (es el mínimo, *lower bound*).
- Dos modelos típicos: **constante** e **independiente**
- El modelo constante, siempre devuelve la misma respuesta para todas las ocasiones.
  - e.g. Si es categórica la salida, el modelo siempre devuelve el valor más popular (se equivoca menos). Si es numérica regresará la media (su desviación cuadrática es menor).
- El modelo independiente, no guarda ninguna relación o interacción entre las variables de entrada y salida, puede ser un modelo al azar (e.g. tirando una moneda para decidir en una clasificación binaria).
- En `sklearn` la clase que implementa esto es `DummyClassifier`

Mejor modelo de una variable

- Un modelo complicado no puede justificarse si no puede mejorar un modelo simple de una sola variable.
- Muchos clientes o usuarios que pueden manejar MS Excel y sus `pivot tables` pueden generar modelos de una sola variable, ellos querrán comparar a este nivel, por lo que siempre es bueno tenerlos en mente.

A/B testing

- **A/B testing** es el método utilizado para responder preguntas como *¿Qué color es mejor para este botón: azul o rojo?* ó preguntas como *¿Es el nuevo modelo mejor que el anterior?*

Setup

- Hay dos modelos (o dos diseños): el **nuevo** y el **actual**
- El tráfico (o los datos) se dividirán en dos grupos A (**control**) y B (**experimento**)
- El flujo A se dirige al modelo **actual** y el flujo B al modelo **nuevo**
- Se compara su desempeño y se toma una decisión, si el modelo **nuevo** tiene un desempeño mejor que el **actual**, el modelo **nuevo** sustituye al modelo **actual**
- Esto es simplemente una *prueba de hipótesis* comparando la hipótesis *nula* (H_0) con la hipótesis *alternativa* (H_1).
    - La pregunta es *¿Este nuevo modelo produce un cambio estadísticamente significativo en esta métrica?*
    - H_0 : *el nuevo modelo no cambia el valor promedio de la métrica*
    - H_1: *el nuevo modelo cambia el valor promedio de la métrica*

A/B testing: pasos

1. Divide de manera aleatoria en grupos de **control** y **experimentación**
2. Observa el comportamiento de ambos grupos en los métodos propuestos
3. Calcula las estadísticas de prueba
4. Calcula el *p-value*
5. Decide

A/B testing: Cuestiones a tener en cuenta

- Revisa el artículo *Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained* (http://www.exp-platform.com/Documents/puzzlingOutcomesInControlledExperiments.pdf) de Kohavi et al.
- Realiza antes (o contínuamente) un **A/A testing**, esto te ayuda a probar tu método de separación.
- Recuerda que hay varias métricas: de negocio, métricas vivas, métricas *off-line* y métricas de entrenamiento lleva un histórico de todas y revísalas continuamente te pueden ayudar a detectar un **temporal drift** (e.g. evalúa la métrica *off-line* con datos “vivos”)
- ¿ *One-side test* (efecto en una dirección: ¿es mejor?) o *Two-side test* (efecto en dos direcciones: ¿puede ser mejor o peor?)?
- **FP** Rechazar H_0 cuando H_0 es verdadera: Aquí significa cambiar a un modelo que no mejora el modelo **actual** ¿Qué *p-value* es aceptable para el negocio?
- Calcula **antes** del experimento el número de observaciones.
- ¿La métrica tiene una distribución gaussiana? (e.g. **AUC** no es un promedio, por lo que no aplica el teorema del límite central)

Multiarmed bandit

Referencia

Consultar *Bandit Algorithms for Website Optimization* de O'Reilly. Aquí lo estamos adaptando a modelos, no a diseño web.

Explotación y Exploración

- *Exploración*: Aprender nuevas cosas
- *Explotación*: Tomar ventaja de lo que ya sabes
- Podemos usar estos dos conceptos para caracterizar el proceso de **A/B testing**:
- Un corto periodo de *exploración*, en la cual se asignan igual número de datos a los grupos A y B.
- Un periodo largo de *explotación* en la cual mandas todos los datos a la versión del modelo más exitosa.

Problemas con A/B testing

- Brinca directamente de *exploración* a *explotación*.
- Durante la etapa de *exploración* gasta recursos en una opción inferior para poder obtener toda los datos requeridos.

¿ Multiarmed bandit ?: Vocabulario

- Cada opción que tengamos (e.g modelos $\{m_1, m_2, \ldots, m_n\}$) será llamado *arm* (brazo).
- *Recompensa*, r, es una médida de éxito, pedimos que se pueda medir y que se pueda ordenar (e.g. $r \in \mathbf{R}$)

Bandit problem

- Tenemos una máquina de moneda con N brazos que podemos jalar.
- Cuando son actuados, cada uno de estos brazos da una recompensa, pero la recompensa no está garantizada (e.g. el brazo 1 puede darnos una recompensa $r = 1$ únicamente el 1% de las veces.).
- Además no sabemos qué brazo da qué recompensa. Esto lo debemos de hacer experimentando, i.e. jalando los brazos.
- Sólo recibimos información (recompensa) del brazo que jalamos.
- Cada vez que experimentamos con un brazo que no es el mejor brazo, perdemos recompensa, ya que , al menos en principio, pudimos haber jalado oun mejor brazo.
- Un algoritmo solución debe de dar una regla (o función) que seleccione brazos en una secuencia, además debe de balancear la explotación/exploración mientras que maximiza la recompensa recibida.

Ejemplo: *Epsilon-greedy algorithm*

- Supón que hay dos modelos a analizar.
    - La idea es muy sencilla: Tira una moneda, si sale águila, exploras por un momento (con probabilidad ϵ, si sale cara, debes de explotar (con probabilidad $1 - \epsilon$).

    - Si debes de explotar, el algoritmo revisa la métrica, determina cuál es el modelo con la mejor métrica en el pasado, y aplica ese modelo a los datos.
    - Si debes de explorar, el algoritmo tira otra moneda (ahora justa) para elegir cualquiera de los dos modelos.
    - Luego de seleccionar, actualiza el valor estimado del brazo.

Ejemplo: *Softmax algorithm*

- Hay un problema con el algoritmo *epsilon-greedy*: explora las opciones completamente al azar, sin tomar en cuenta sus méritos.
    - Si la diferencia en las tasas de recompensa es pequeña, debes de explorar mucho más para determinar correctamente cual de las dos opciones es mejor.
    - Si la diferencia es muy grande, debes de explorar menos para estimar correctamente cuál es la mejor opción.
- El nuevo algoritmo necesita tomar en cuenta acerca de las diferencias en los valores estimados de cada brazo. A esto se le conoce como *exploración estructurada*.
- El algoritmo *softmax* escoge cada brazo en proporción a su valor estimado, según el siguiente algoritmo:
    1. Establece τ, la temperatura (En los sistemas físicos a mayor temperatura, existen más oscilaciones moleculares, i.e. más azar)
    2. En cada tiempo T seleccionamos un brazo usando

$$\frac{e^{\frac{r_i}{\tau}}}{\sum_j e^{\frac{r_j}{\tau}}}$$

    3. Actualiza el valor de tu estimado.
- Existen otros algoritmos más elaborados.

Ejercicio

¿Cómo modificarías el servicio web que creaste para implementar evaluación en línea y que además permita ejecutar un algoritmo *multiarmed bandit*?

¿Por qué de todo esto?

- **Malas noticias**: Aprender a partir de casos particulares (es decir, lo que hace *machine learning*) no es un problema bien definido
- Esto en general se llama *inducción*
- En palabras de **David Hume**:

> Instancias de las cuales no tenemos ninguna experiencia se parecen a aquellas de las cuales tenemos experiencia

Problema de la inducción

- Y hay un problema con su nombre: *el problema de la inducción* (http://plato.standford.edu/entries/induction-problem), planteado por **Sextus Empiricus** en el siglo I E.C.: *una regla universal no puede ser establecida a partir de un conjunto incompleto de instancias*.
    - Si son creyentes **Ockham** (el de la navaja) también tiene cosas que decir al respecto.
- **David Hume** lo expresó indicando la circularidad: La única justificación para la inducción es el método inductivo: *funciona para unos casos, esperamos que funcione para todos los problemas inductivos*.
- **J. S. Mill** *¿Por qué una instancia es suficiente para una inducción completa, cuando en otras ocasiones, saber hacer de miles de instancias, no nos lleva a establecer una proposición universal?*
- En tiempos más recientes, **Nelson Goodman** y su *acertijo de la inducción*:

> Algo es “grue” si y sólo si ha sido observado como *verde* antes de cierto tiempo t y azul a partir de ahí. Si todas las esmeraldas son verdes y “grue” ¿Por qué suponemos que luego de t serán todas verdes y ninguna “grue”?

No free lunch theorem

Ningún algoritmo puede ser m ejor que cualquier otro cuando sea evaluado sobre todos los posibles problemas de clasificación, así el desempeño de cualquier algoritmo sobre todo el conjunto de todos los posibles problemas de aprendizaje no es mejor que adivinar.

¿Qué nos faltó de cubrir?

- **On-line learning**: Exponer a los *learners* (algoritmos) a los datos uno a la vez, sólo vimos *batch*.
- **Predicción conforme**: ¿Qué hago para ver como generaliza mi modelo si el entrenamiento es *online*? ¿Qué hago con diferentes tipos de ruido?
- Series de tiempo (que no cumplen con las condiciones de análisis de series de tiempo clásicas) i.e. *filtros de Kalman extendidos*, *RNN*, *Feedforward ANN*, etc.

Conclusiones

- Debemos tener conocimiento de la distribución
- Los *features* son tan importantes o más que los datos
- Y los hiperparámetros…
- Al final el modelo
- Hay que tener cuidado con la *fuga de información* para generalizar: Usa cross-validation, etc.
- Los problemas no acaban con el primer modelo, hay que ponerlo en producción y el problema de la reproducibilidad continúa ahí