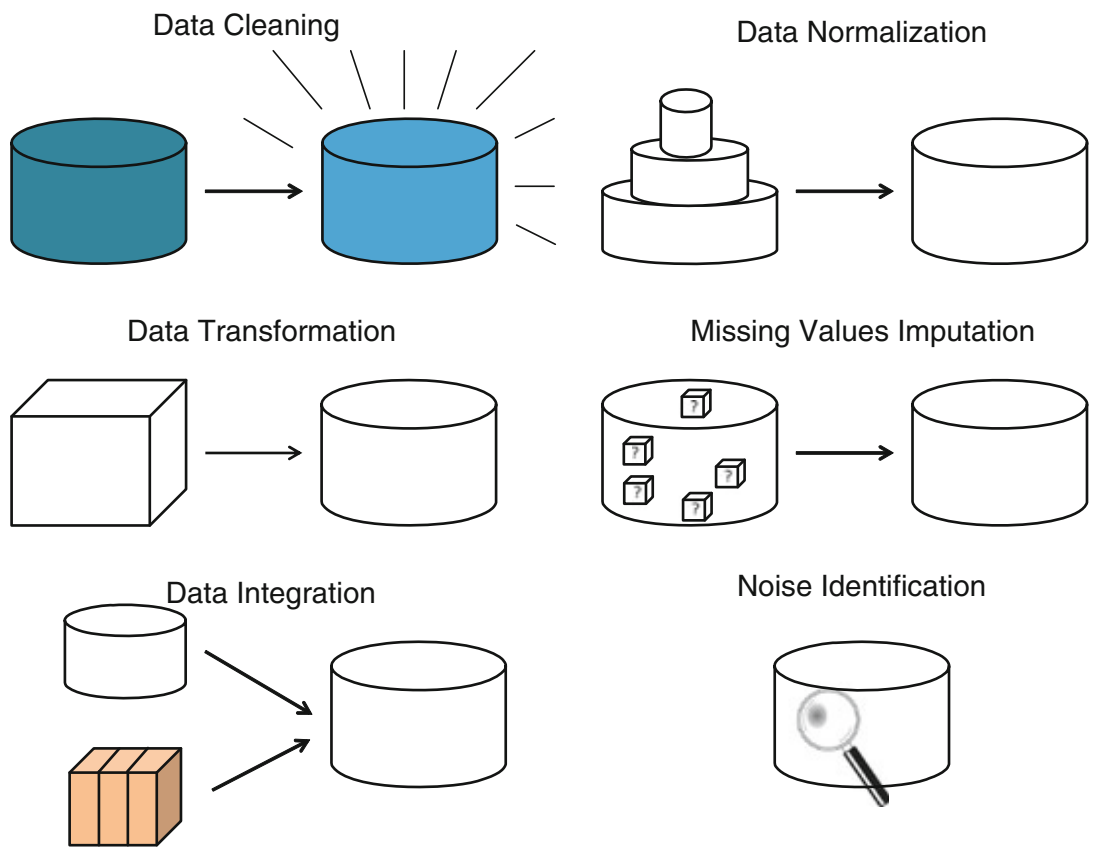
Introducción al preprocesamiento de datos

- Los datos de entrada deben ser proporcionados en la cantidad, estructura y formato que requiera cada tarea de minería de datos
- Las bases de datos reales están influenciadas por factores negativos como
  - ruido
  - valores perdidos
  - datos inconsistentes
  - datos superfluos
  - gran número de características
  - gran número de instancias

Preparación de datos



Preparación de los datos

- **Ajuste de datos de múltiples fuentes**: Integración de datos
- **Inconsistencias**: Limpieza de datos
- **Unificación y escalado de datos**: Normalización de datos
- **Datos precisos**: Transformación de datos
- **Datos perdidos**: Imputación de datos
- **Ruido**: Identificación de ruido
- **Alta dimensionalidad**: selección de características y selección de instancias

Técnicas para la preparación de los datos

1. Integración de datos
2. Limpieza de datos
3. Normalización de datos
4. Transformación de datos
5. Manejo de valores perdidos
6. Selección de instancias
7. Selección de características

Integración de datos

- Se desea evitar redundancias e inconsistencias en la mezcla de datos de múltiples almacenes
- Operaciones típicas:
  - búsqueda de atributos redundantes
  - detección de tuplas duplicadas e inconsistentes

Búsqueda de atributos redundantes

- **Test de correlación de la chi cuadrado:** dos atributos A y B contienen c y r valores diferentes cada uno. oij es la frecuencia observada del evento (Ai, Bj), eij es la frecuencia esperada de ese evento

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}},$$

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{m}$$

Búsqueda de atributos redundantes

- Coeficiente de correlación para datos numéricos: dependencia lineal

$$r_{A,B} = \frac{\sum_{i=1}^{m}(a_i - \overline{A})(b_i - \overline{B})}{m\sigma_A\sigma_B} = \frac{\sum_{i=1}^{m}(a_i b_i) - m\overline{A}\ \overline{B}}{m\sigma_A\sigma_B}$$

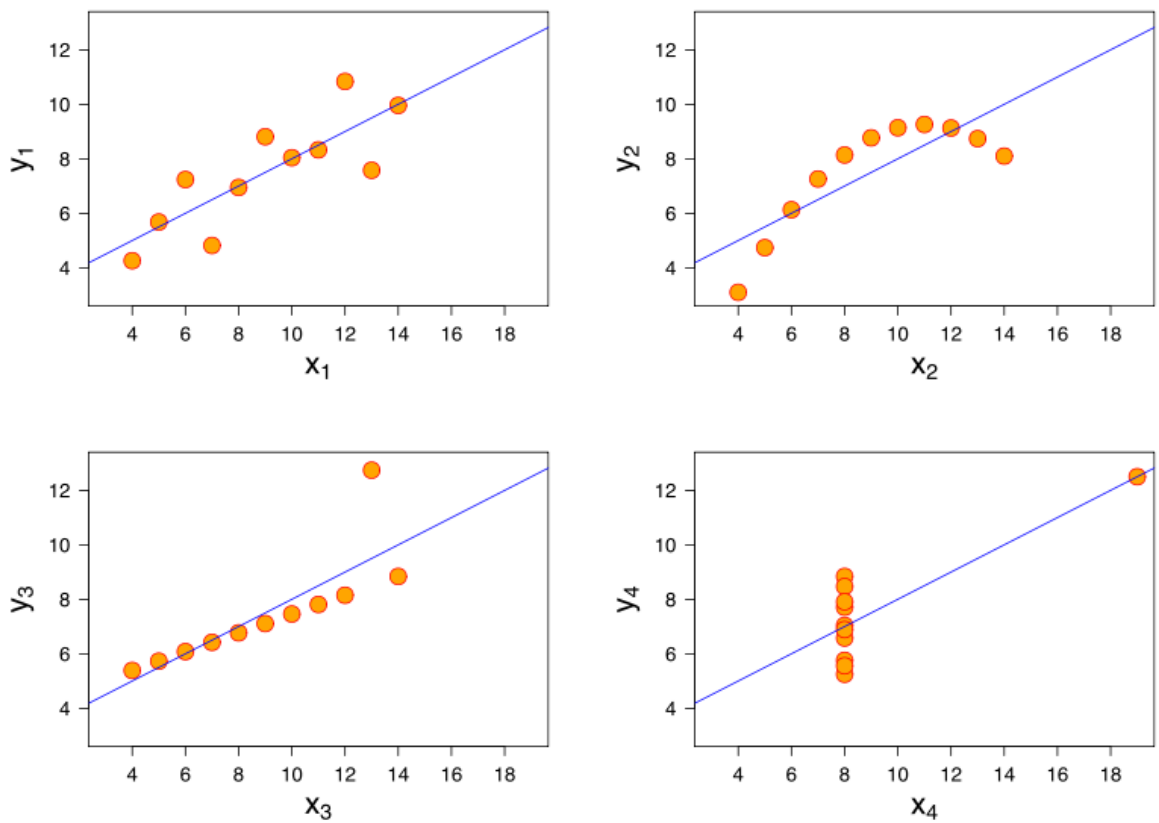
- Covarianza para datos numéricos: datos independientes tienen covarizanza cero

$$Cov(A, B) = E((A - \overline{A})(B - \overline{B})) = \frac{\sum_{i=1}^{m}(a_i - \overline{A})(b_i - \overline{B})}{m}$$

$$r_{A,B} = \frac{Cov(A, B)}{\sigma_A\sigma_B}.$$

Propiedades de la correlación

- Las cuatro funciones de la derecha tienen R=0.82



Detección de duplicidades e inconsistencias

- En determinados algoritmos es necesario comprobar que no existen instancias duplicadas (o instancias con las mismas características y diferente valor de la variable dependiente)
- Puede ocurrir que la duplicidad no sea exacta sino que haya redondeos o pequeños errores
- Cuando los atributos son nominales la comprobación de las duplicidades no es trivial si hay errores

Duplicidades e inconsistencias

- La distancia de edición entre dos cadenas es el número mínimo de operaciones de edición necesarias para convertir una cadena en la otra. Las operaciones son: insertar un carácter, reemplazar un carácter o eliminar un carácter. Pueden incluirse costes en las operaciones. Se usa programación dinámica para hacer una implementación eficiente.

- Las similaridades entre datos numéricos son más complejas de detectar, ya que las distancias entre vectores no contemplan determinados tipos de error, como las permutaciones de valores. Normalmente se detectan las inconsistencias mediante análisis de outliers (se verá más adelante)

Detección de redundancias e inconsistencias en python

- No hay módulos específicos para realizar este tratamiento de los datos en scikit o pandas por instancias (sí los hay para detectarlo por características)
- Existen módulos para calcular la distancia de edición entre dos cadenas o para computar la covarianza o el test chi cuadrado
- Ejemplo de comprobación de que dos atributos son redundantes

```
# Discretización
sepal = pd.cut(iris['sepal_length'],4)
petal = pd.cut(iris['petal_length'],4)
# Tabla de contingencia
tcon = pd.crosstab(petal,sepal)
import scipy as sc
result = sc.stats.chi2_contingency(tcon)
```

Limpieza de datos

- Los datos sucios aparecen en dos formas: datos perdidos y datos ruidosos

- El tratamiento de ambos es diferente:

  - Los **datos perdidos** se procesan antes de analizar los datos: se ignoran, se reemplazan por un valor constante o dependiente de la variable, o bien se estiman en función de los datos cercanos. La imputación de datos se estudia más adelante.

  - Los **datos con ruido** se procesan mediante técnicas estadísticas, generalmente con filtrado o bien reemplazando cada valor por la predicción de un modelo de regresión en los datos cercanos.

Limpieza de datos en python

- Sklearn tiene módulos para imputaciones sencillas (valor constante, media o mediana de la columna). Hay módulos experimentales para realizar imputación basada en KNN o SVM. Se estudiarán por separado en esta sesión.

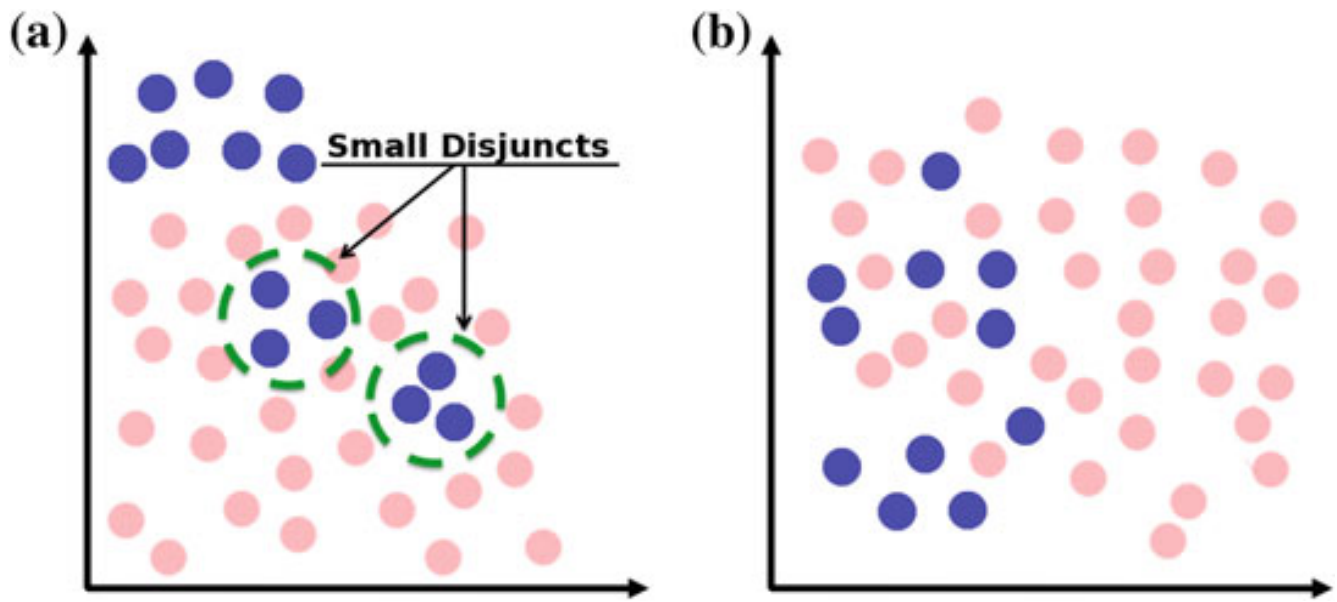
- Los algoritmos de filtrado son específicos del problema. Se pueden solucionar con suavizados o con técnicas espectrales. Se estudiarán en la sesión de series temporales y sistemas dinámicos.

- Existen algoritmos de aprendizaje robusto, que se ven menos influidos por el ruido.

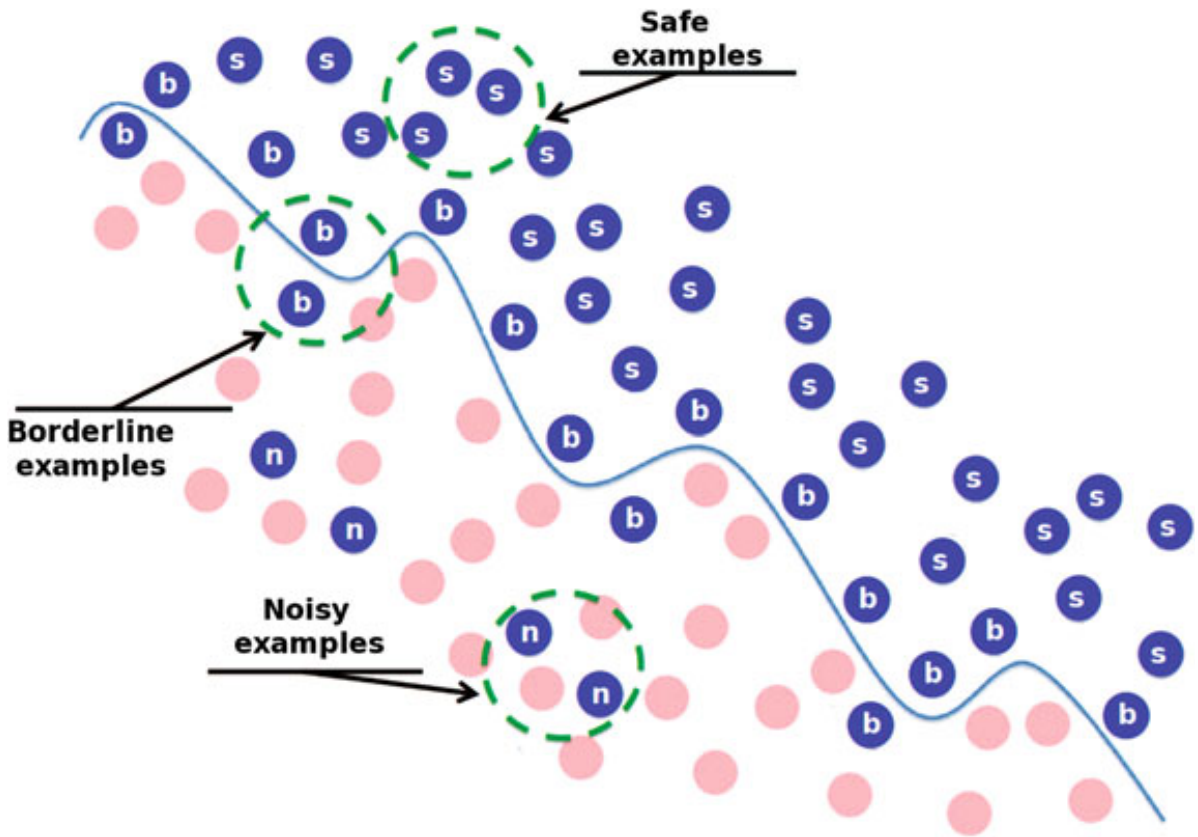
- Las distorsiones del modelo causadas por datasets con ruido también pueden controlarse limitando los grados de libertad del modelo (criterio de Akaike, MDL, etc.)

Tipos de ruido

- Puntos aislados o clases solapadas (clasificación)



Tipos de ruido



Normalización de datos

- Normalización min-max

$$v' = \frac{v - min_A}{max_A - min_A}(new - max_A - new - min_A) + new - min_A$$

- Normalización z-score

$$v' = \frac{v - \overline{A}}{\sigma_A}$$

- Normalización de escala digital

$$v' = \frac{v}{10^j}$$

Escalado de datos con scikit-learn

- Estandarización: datos con media cero y varianza unidad

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X = np.array([[ 1., -1.,  2.],
...               [ 2.,  0.,  0.],
...               [ 0.,  1., -1.]])
>>> X
array([[ 1., -1.,  2.],
       [ 2.,  0.,  0.],
       [ 0.,  1., -1.]])
>>> X_scaled = preprocessing.scale(X)
>>> X_scaled
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

Escalado en scikit-learn

- Para escalar con las mismas media y varianza el conjunto de test se usa la clase StandardScaler

```
 X_scaled.mean(axis=0)
 X_scaled.std(axis=0)

>>> scaler = preprocessing.StandardScaler().fit(X)
>>> scaler
StandardScaler(copy=True, with_mean=True, with_std=True)
>>> scaler.mean_
array([ 1.        ,  0.        ,  0.33333333])
>>> scaler.scale_
array([ 0.81649658,  0.81649658,  1.24721913])
>>> scaler.transform(X)
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
>>> scaler.transform([[1,2,3]])
array([[ 0.        ,  2.44948974,  2.13808994]])
```

Escalado a un rango

- El escalado entre un valor mínimo y un valor máximo se resuelve con el mismo paquete

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>>
>>> min_max_scaler =
preprocessing.MinMaxScaler()
>>> min_max_scaler
MinMaxScaler(copy=True, feature_range=(0, 1))
>>> X_train_minmax =
min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[ 0.5       ,  0.        ,  1.        ],
       [ 1.        ,  0.5       ,  0.33333333],
       [ 0.        ,  1.        ,  0.        ]])
```

Escalado por el máximo valor absoluto

- Se divide por el máximo valor absoluto de cada característica, en datos que estén ya centrados en cero

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>>
>>> max_abs_scaler = preprocessing.MaxAbsScaler()
>>> X_train_maxabs = max_abs_scaler.fit_transform(X_train)
>>> X_train_maxabs
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
>>> X_test = np.array([[ -3., -1.,  4.]])
>>> X_test_maxabs = max_abs_scaler.transform(X_test)
>>> X_test_maxabs
array([[-1.5, -1. ,  2. ]])
```

Transformación de datos

- Transformaciones lineales, cuadráticas, no polinomiales, etc.
- Aproximaciones polinomiales
- Transformaciones de rango
- Transformaciones Box-Cox
- Ecualización de histograma
- Transformación nominal a binario
- Transformaciones mediante reducciones de datos

Transformaciones

- Transformación lineal

$$Z = r_1 B_1 + r_2 B2 + \cdots + r_m B_M$$

- Transformación cuadrática

$$Z = r_{1,1} B_1^2 + r_{1,2} B_1 B_2 + \cdots + r_{m-1,m} B_{m-1} B_m + r_{m,m} B_m^2$$

Transformaciones

- Transformaciones de rango: reemplazar el valor por su rango (orden dentro de los datos, entre 1 y m)

- Transformación del rango en z-scores (phi es la función de distribución normal) – pero no puede aplicarse de forma separada a train y a test

$$y = \Phi^{-1}\left(\frac{r_i - \frac{3}{8}}{m + \frac{1}{4}}\right)$$

Transformaciones

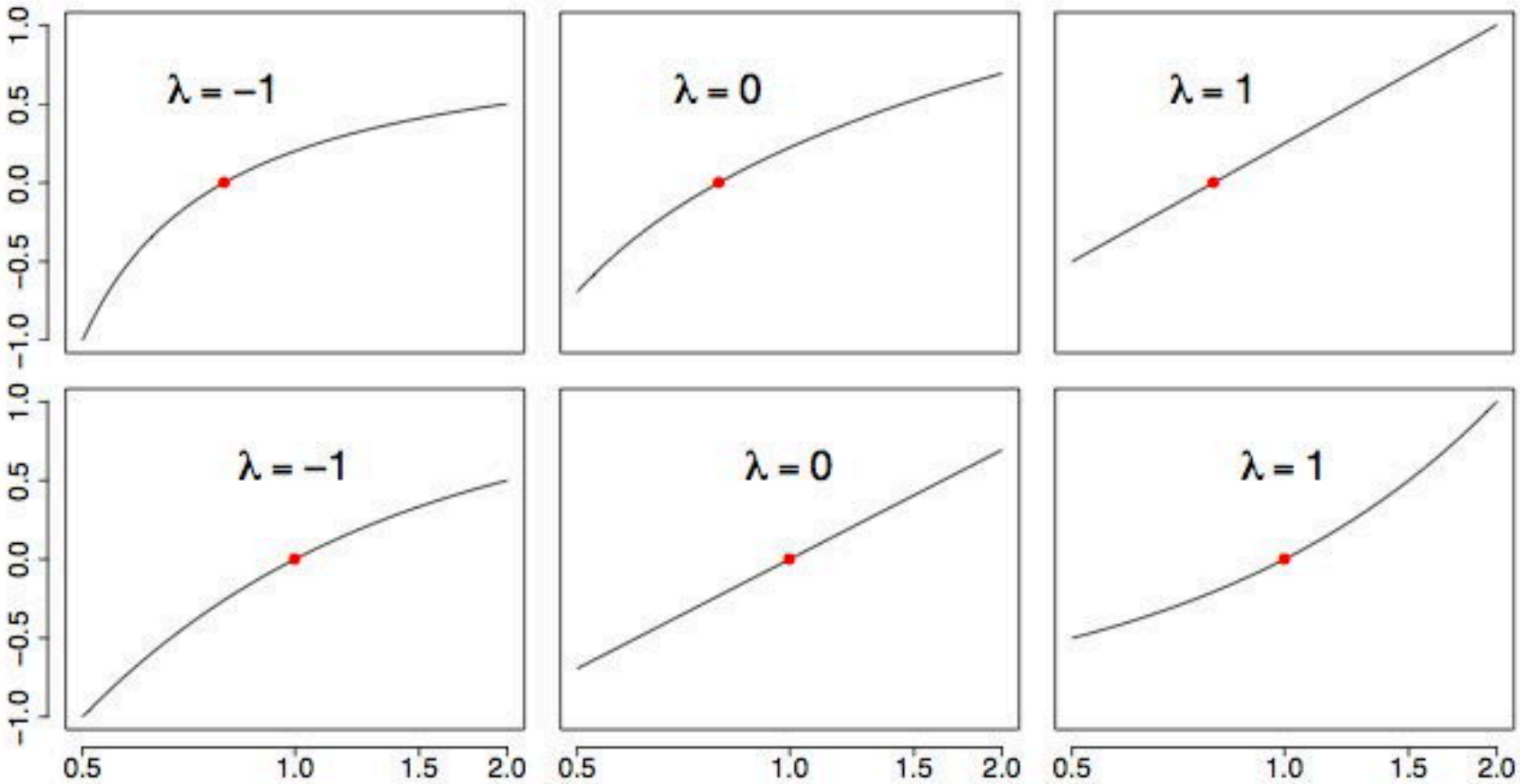
- Transformación Box-Cox para valores positivos

$$y = \begin{cases} x^{\lambda-1}/\lambda, & \lambda \neq 0 \\ log(x), & \lambda = 0 \end{cases}$$

- Transformación Box-Cox para valores positivos y negativos (c corrige el offset y g es la media geométrica de los datos)

$$y = \begin{cases} (x+c)^{\lambda-1}/g\lambda, & \lambda \neq 0 \\ log(x+c)/g, & \lambda = 0 \end{cases}$$

Transformaciones Box-Cox



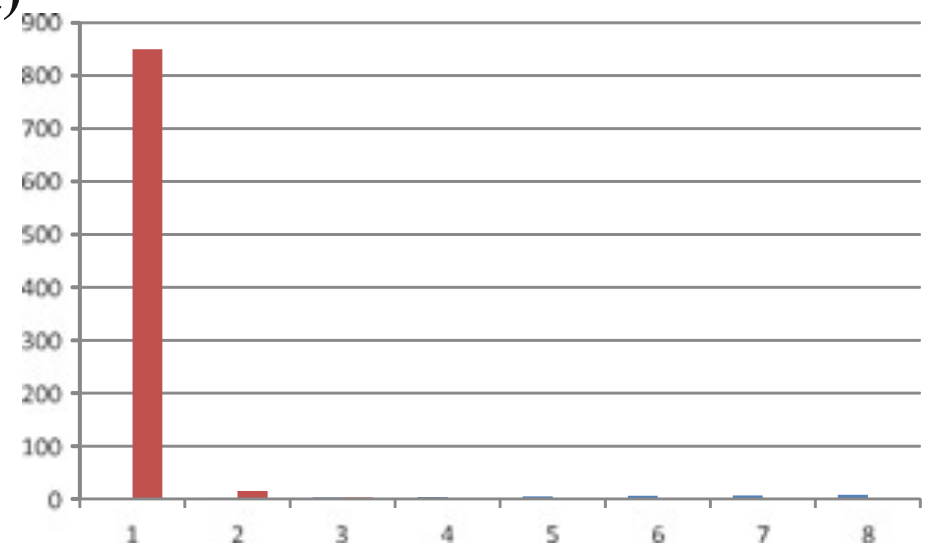
y vs. x

y vs. log(x)

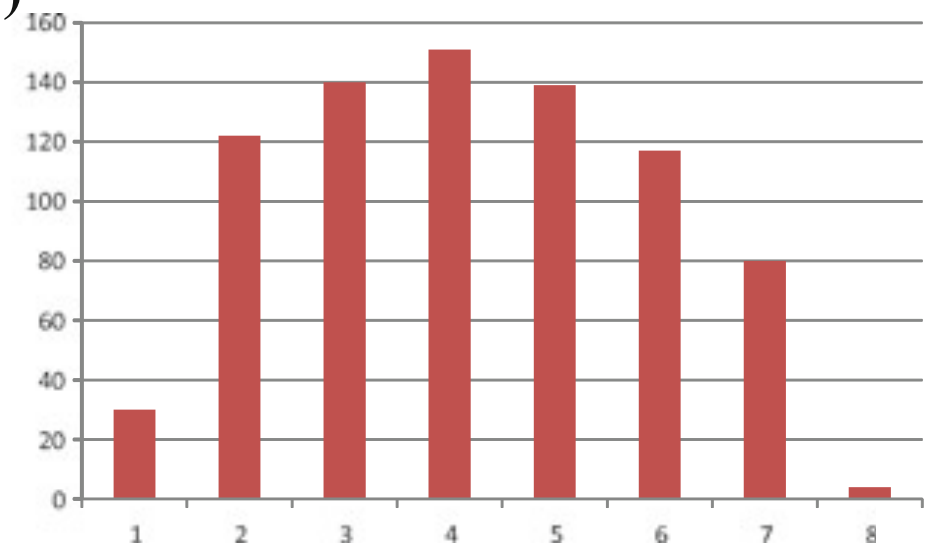
Transformaciones

- Las transformaciones Box-Cox y de rangos transforman el histograma de los datos

(a)



(b)



Transformaciones polinómicas en sklearn

- Se transforma un dataset en todas las transformaciones polinómicas de sus variables, con interacción
- polinomios(X,2) = 1, X, X**2
- polinomios(X,Y,2) = 1, X, Y, X*Y, X**2, Y**2
- polinomios sin interaccion(X,Y,Z,3) = 1,X,Y,XY,XZ,YZ,XYZ

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
# Todos los terminos
poly = PolynomialFeatures(2)
poly.fit_transform(X)

# Solo productos
poly = PolynomialFeatures(3,interaction_only=True)
poly.fit_transform(X)
```

Transformaciones a la medida sklearn

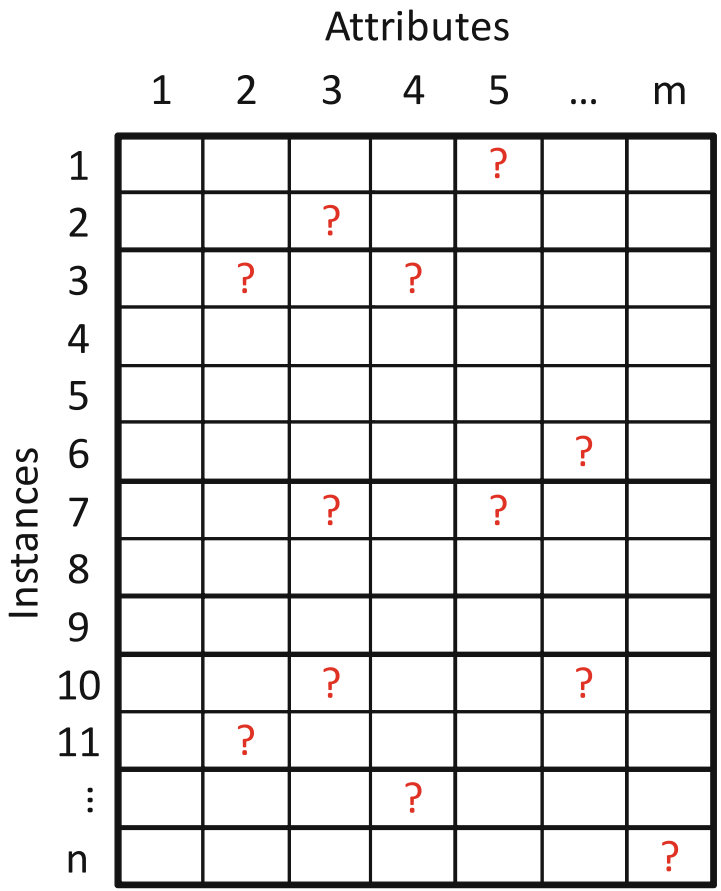
- Para cualquier otra función, se crea un FunctionTransformer

```
import numpy as np
from sklearn.preprocessing import
FunctionTransformer
transformer = FunctionTransformer(np.log1p)
transformer.transform(X)
```

Transformaciones Box-Cox y rango: boxcox.py

```
X = uniform.rvs(loc=0,scale=1,size=1000).reshape(-1,1)

def bc(x,l=0):
    return np.vectorize(lambda t:(t**l-1)/l if l!=0 else np.log(t))(x)

def rango(x):
    return np.argsort(x,axis=0)

def rng(x):
    m = x.shape[0]
    return np.vectorize(lambda x : norm.ppf((x-3.0/8.0)/(m+0.25)))(rango(x))

transformerBC = FunctionTransformer(bc)
transformerRN = FunctionTransformer(rng)
XBC = transformerBC.transform(X)
XRN = transformerRN.transform(X)

plt.subplot(131); plt.hist(X,range=(0,1))
plt.subplot(132); plt.hist(XBC,range=(-10,10))
plt.subplot(133); plt.hist(XRN,range=(-3,3))
plt.show()
```

Manejo de valores perdidos



- Hipótesis MAR (Missing At Random: el que una característica esté perdida no depende de su valor)
- Bajo MAR, la distribución de los datos en torno al valor perdido es la misma que entre los valores observados
- Aproximaciones simples: DNI (do not impute), case deletion/ignore missing, global average value o concept average value, cold deck, hot deck (reemplazar valores perdidos por una instancia elegida al azar)

Imputación basada en ML

- **Imputación KNN:** Se buscan las instancias más cercanas y se reemplaza el valor perdido por el más frecuente o por el promedio
- **Imputación WKNNI:** El promedio es ponderado por la distancia al individuo
- **Imputación KMI:** Se hace un clustering de los datos y se reemplaza el valor perdido por el valor correspondiente al centroide del cluster
- **Imputación FKMI:** Fuzzy k-means y promedio de los centroides ponderado por las pertenencias a los clusters
- **Imputación SVMI:** Se utiliza regresión SVM para hacer un modelo basado en los datos completos, se reemplazan los valores perdidos por los resultados del modelo de regresión
- **Singular Value Decomposition Imputation (SVDI):** Se hace una regresión regularizada y se predicen las características perdidas
- **Local Least Squares Imputation (LLSI):** Regresión por mínimos cuadrados ponderada por la similaridad entre las instancias

Imputación en python: sklearn

- Fichero **imputacion.py**
  - Asignación de una constante a cada valor perdido
  - Asignación de la media de la característica a cada valor perdido
  - Asignación de la mediana
  - Asignación del valor más frecuente
- Comparativa numérica entre eliminar datos perdidos, diferentes tipos de imputación

fancyimpute

- Imputación de valores perdidos por métodos no implementados en scikit-learn
- KNN, SoftImpute, NuclearNormMinimization
- **Website:** https://github.com/hammerlab/fancyimpute
- **from fancyimpute import KNN, NuclearNormMinimization, SoftImpute**

Imputación en python: fancyimpute

- Imputación basada en machine learning
    - **KNN:** promedio de los valores más cercanos
    - **NuclearNormMinimization, SoftImpute:** se supone que el dataset puede comprimirse en una matriz de rango menor, y se buscan los coeficientes de la matriz comprimida de forma que su expansión coincida con los valores observados de las características. Los coeficientes de la matriz descomprimida sirven para realizar la imputación.
- Más detalles en el archivo **imputacion_avanzado.py**

Selección de instancias

- La selección de instancias se aplica en los problemas en que el volumen de los datos es alto y los algoritmos tardan demasiado (o no pueden aplicarse) al dataset completo.
- Suele hacerse a la vez que la selección de características (se explica más adelante en esta sección)
- Las propuestas iniciales se basan en el KNN
- Se distingue entre Prototype Selection (PS) y Training Set Selection (TSS)

TSS y PS

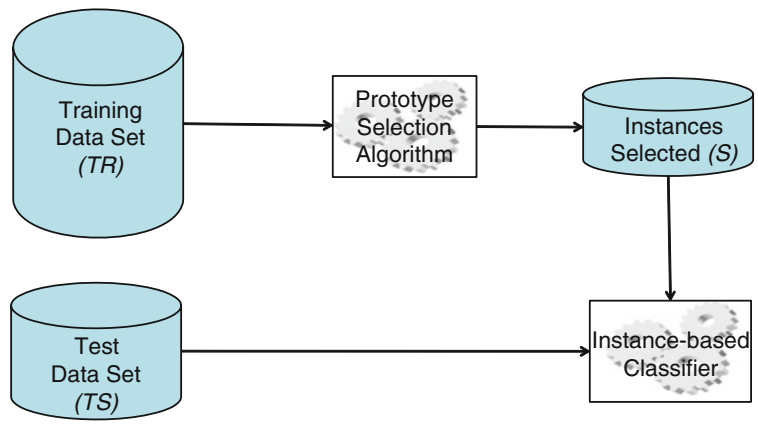


Fig. 8.1 PS process

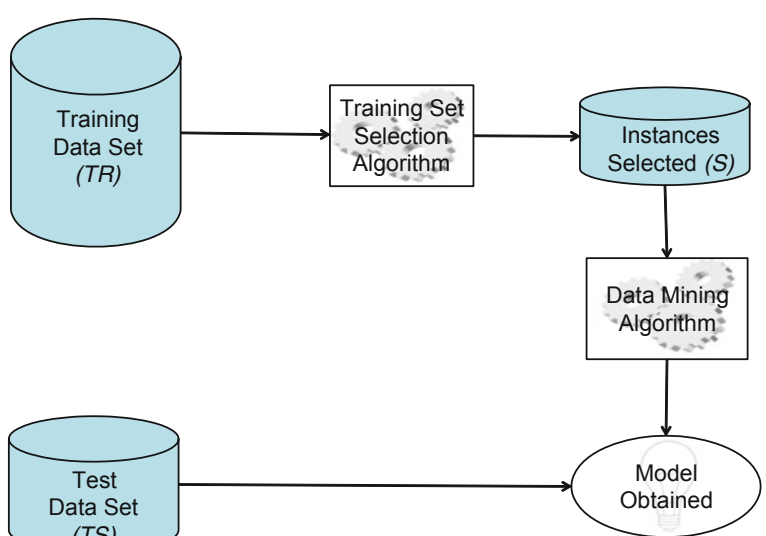


Fig. 8.2 TSS process

Taxonomía en la selección de prototipos

- **Dirección en la búsqueda**: *incremental* (se comienza con conjunto vacío), decremental (se comienza con conjunto completo), *batch* (se comienza con un conjunto pre-seleccionado) y *fijo* (conjunto de tamaño constante)

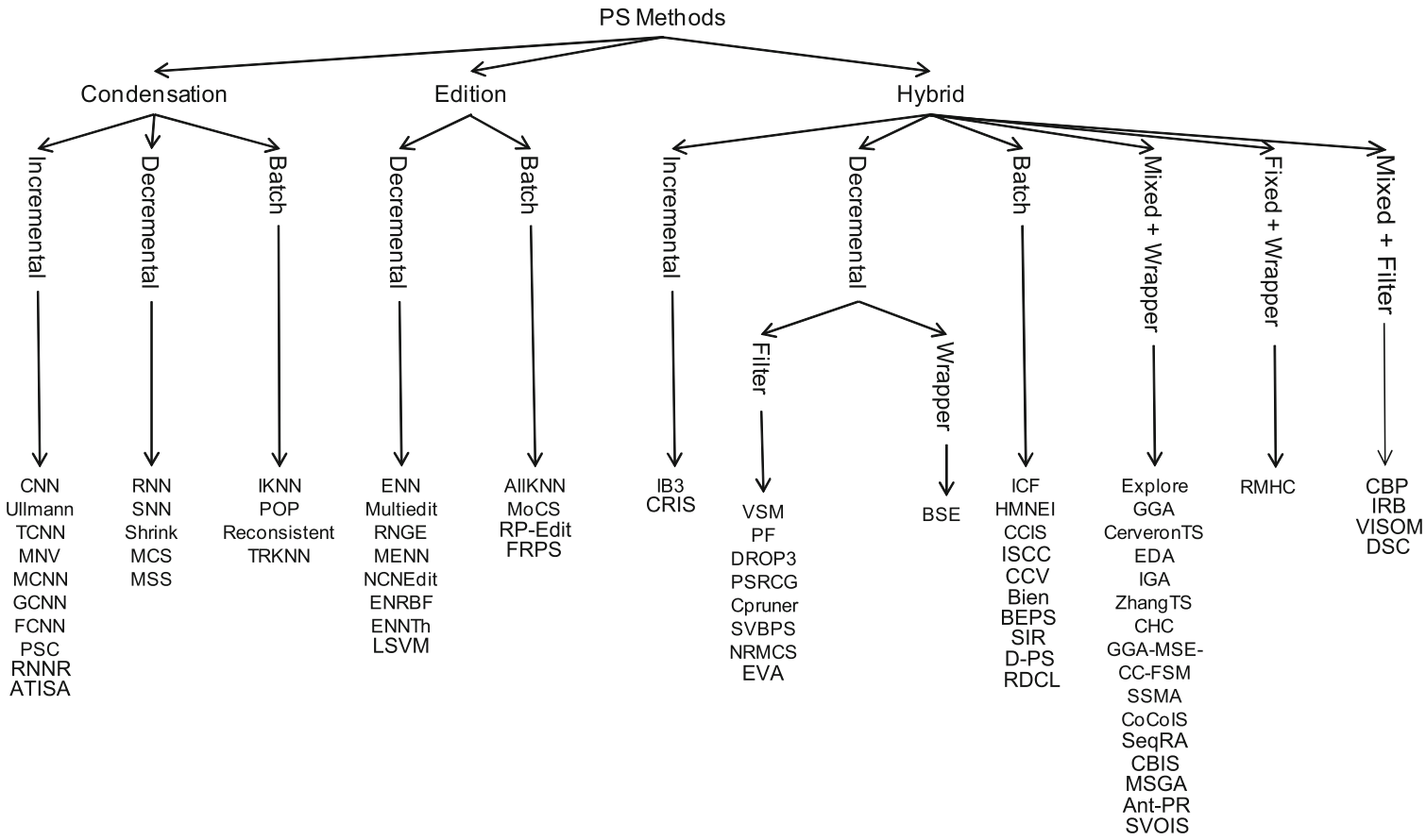
- **Tipo de selección:** *condensación* (se retienen los puntos próximos a las fronteras o puntos borde, que son los que más afectan a la clasificación), *edición* (se eliminan los puntos borde y solamente se mantienen los puntos cuya clase es segura) e *híbrido* (conjunto más pequeño que mantiene el error de validación)

Evaluación de la búsqueda

- **Filtro:** Se usa KNN sobre los datos parciales para estimar el porcentaje de aciertos del clasificador

- **Wrapper:** Se usa KNN sobre el conjunto completo con leave-one-out (muy costoso)

Métodos



- Hay más de 100 métodos de selección de instancias en la literatura
- ~~Python aún no cuenta con un paquete estable de selección de instancias~~
- CNN está implementado en el paquete imblearn, entre otros

Selección de instancias en python

- Estudiar ~~la implementación de ejemplo~~ el uso de CNN en el archivo **instanceselection.py**

Selección de características

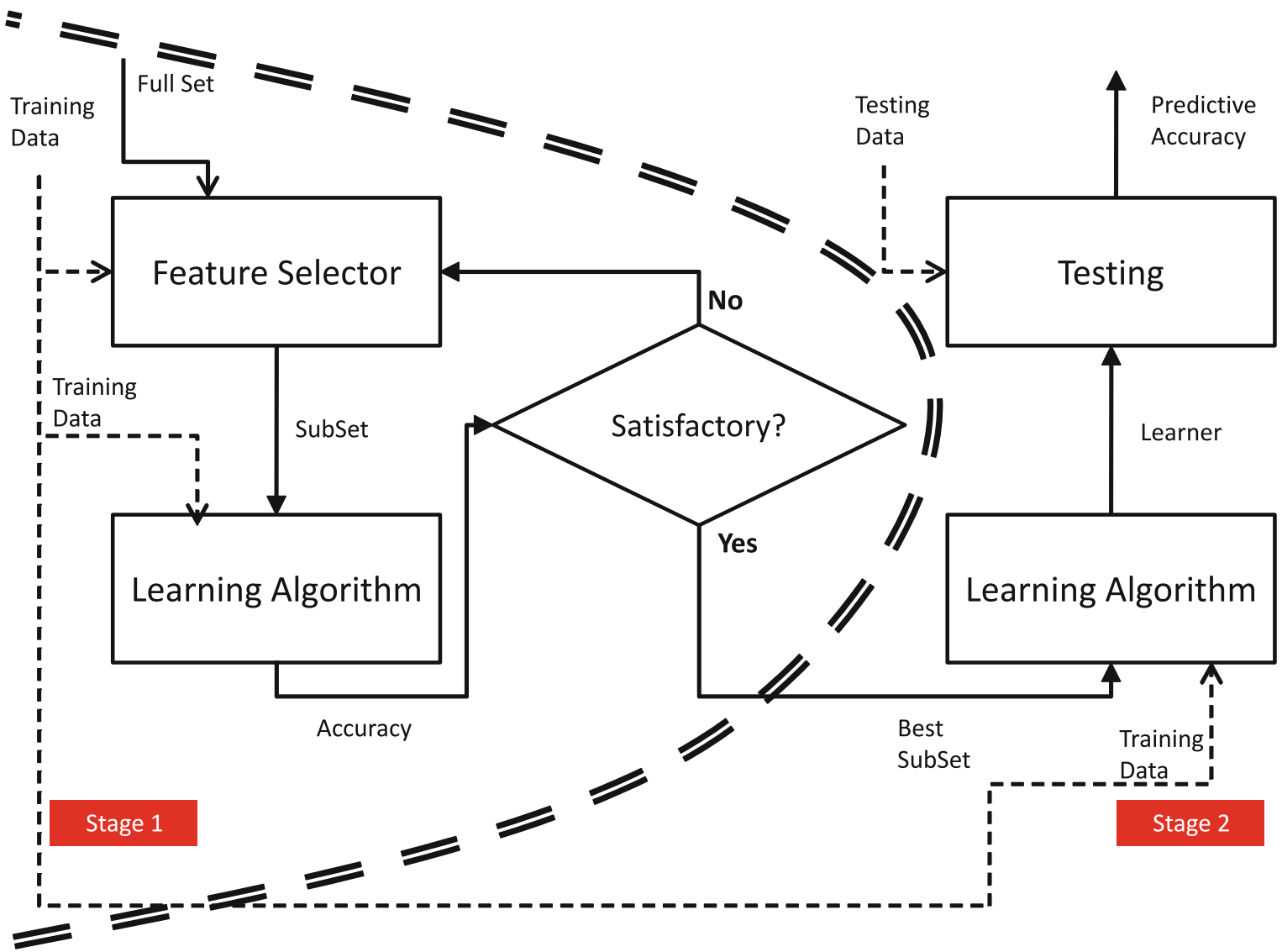
- Los problemas de selección de características pueden enfocarse de tres formas:

1. **Envolvente:** Como un problema de búsqueda del subconjunto de variables que maximizan el error de validación del algoritmo de clasificación o de un algoritmo alternativo

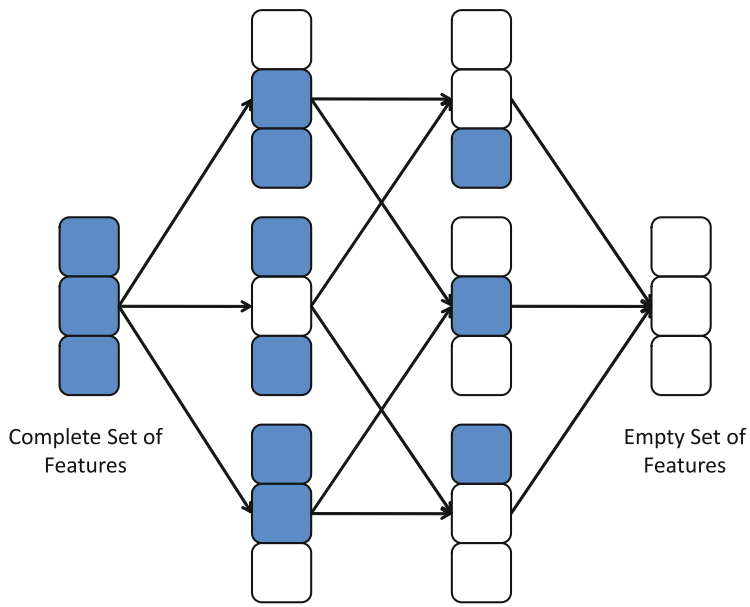
2. **Filtro:** Como la búsqueda del conjunto de variables que maximizan la información acerca de la variable de salida y que aportan la mínima información una acerca de la otra

3. **Selección por modelo o "embebida":** Como el conjunto de variables más frecuentemente usadas por un modelo, generalmente un árbol de decisión o un ensemble de clasificadores como el Random Forest

Selección envolvente



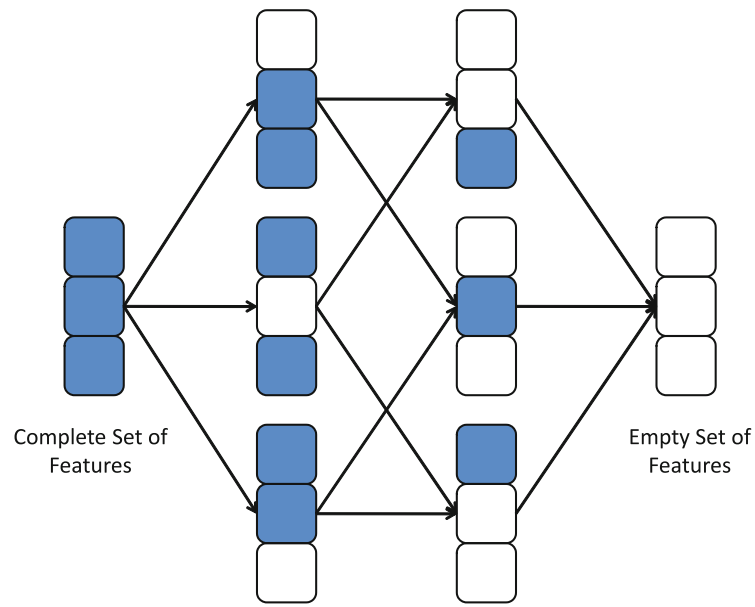
Selección de características envolvente



- Si no hay información acerca del problema, la selección de características es un problema de búsqueda donde se prueban todas las combinaciones de variables.

- SFG: Se comienza con un conjunto vacío, se añade cada característica, se decide cuál es la mejor, se consolida y se repite con la segunda, tercera, etc. Se para cuando aumenta el error de validación al añadir una característica

Selección de características envolvente

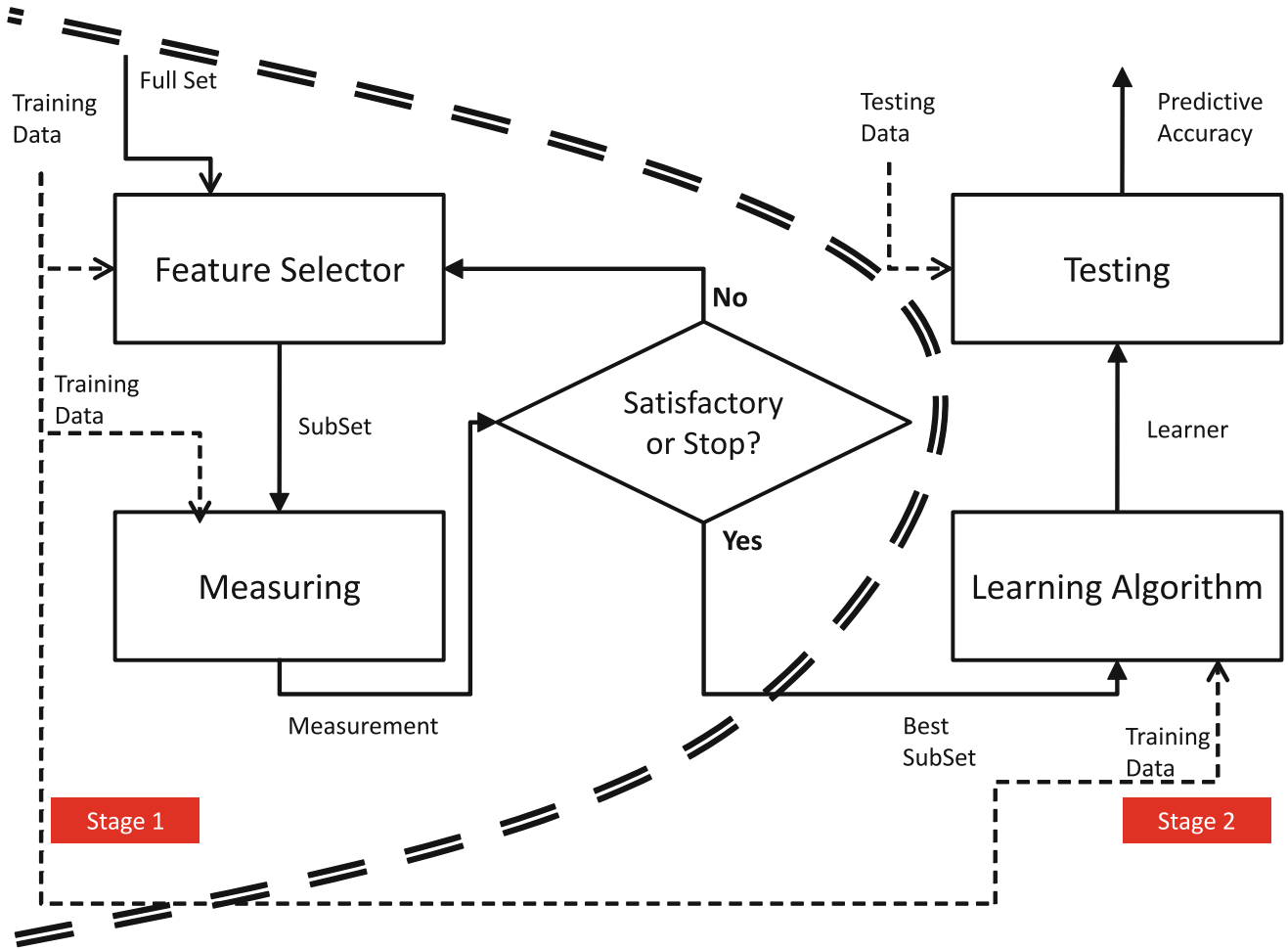


- SBG: Se comienza con un el conjunto completo, se elimina cada característica, se decide cuál es la que menos información aporta,se consolida un conjunto en que esa característica no esté y se repite hasta llegar al conjunto vacío. Se para cuando aumenta el error de validación al eliminar una característica

Selección de características envolvente

- Hay esquemas mixtos, y búsquedas guiadas por metaheurísticas (no deterministas)
- La búsqueda puede apoyarse en un clasificador más sencillo, como KNN

Selección tipo filtro



Selección de características filtro

- Se basan en un contraste de independencia entre la variable ‘clase’ y cada una de las características
- La independencia se mide mediante la información mutua, o en términos generales como la divergencia entre la distribución conjunta de los datos y los productos de las marginales (p.e. criterio chi2)
- En problemas de regresión, también pueden usarse medidas de correlación (como el coeficiente de correlación de Pearson para la dependencia lineal)
- Se combinan con una medida de consistencia para detectar variables redundantes

Selección de características con sklearn

- Eliminación de variables constantes (con poca varianza)
- Eliminación basada en estadísticos univariantes (cantidad de información)
- Eliminación recursiva de variables
- Eliminación mediante SelectFromModel

Selección de características con sklearn

Algoritmos tipo filtro:

- SelectKBest: elimina todas las variables menos las k mejor valoradas
- SelectPercentile: elimina todas las variables menos el porcentaje indicado

Hacen uso de una de las siguientes funciones para determinar si una variable es relevante:

- Para regresión: `f_regression`
- Para clasificación: `chi2` o `f_classif`

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
sel = SelectKBest(chi2,k=2)
iris_reducido1 =
sel.fit_transform(iris.iloc[:,0:4],iris.iloc[:,4])
```

Selección de características con sklearn

- **Eliminación recursiva de variables:** La eliminación recursiva de variables consiste en entrenar un modelo con todas las variables y determinar la importancia de cada una en el resultado. Se eliminan las variables con importancia menor y se repite el proceso. Mediante validación cruzada, se valora cada uno de los modelos que se vayan obteniendo y se determina el punto óptimo como aquél en el que el error del modelo sea menor.
- Es necesario que el clasificador/modelo de regresión utilizado proporcione una medida de la importancia de cada variable para poder emplear este método

```
from sklearn.feature_selection import RFECV
from sklearn.svm import SVC
estimator = SVC(kernel="linear")
sel = RFECV(estimator, step=1, cv=5)
iris_reducido2 =
sel.fit(iris.iloc[:,0:4],iris.iloc[:,4])
print sel.ranking_
print sel.support_
```

Selección de características con sklearn

- **Selección por modelo:** Los árboles de decisión no necesariamente emplean todas las variables. La fracción de veces que una variable es elegida tras lanzar repetidamente un árbol de decisión con semilla aleatoria es una medida de la importancia de la variable.

```
forest =
ExtraTreesClassifier(n_estimators=250,random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in
forest.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
```

- Más información en fichero modelselector.py