



Representación de características de datos

Héctor Sotelo

Queremos analizar los siguientes datos

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K
5	37	Private	Masters	Female	40	Exec-managerial	<=50K
6	49	Private	9th	Female	16	Other-service	<=50K
7	52	Self-emp-not-inc	HS-grad	Male	45	Exec-managerial	>50K
8	31	Private	Masters	Female	50	Prof-specialty	>50K
9	42	Private	Bachelors	Male	40	Exec-managerial	>50K
10	37	Private	Some-college	Male	80	Exec-managerial	>50K



Regresión logística para predecir el ingreso

La regresión logística hace predicciones \hat{y} , usando la siguiente formula:

➡ $\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b > 0$

Donde:

- ➡ $W[i]$ y b son los coeficientes obtenidos del conjunto de entrenamiento.
- ➡ $X[i]$ son las entradas de nuestros datos

Sustituyendo valores de nuestros conjunto de datos

$$\hat{Y}(0) = w[0] * 39 + w[1] * \text{State-gov} + w[2] * \text{Bachelors} + w[3] * \text{male} + w[4] * 40 + w[5] * \text{Adm-clerical} + b$$

- ➡ *Esta formula tiene sentido para las variables $x[0]$ y $x[4]$*
- ➡ *Pierde el sentido para las variables $x[1]$, $x[2]$, $x[3]$ y $x[5]$*

Necesitamos otra forma de representar los datos.



Variables continuas y variables categóricas.

- Variable continua: una variable numérica que puede tomar cualquier valor dentro de un intervalo. (edad, horas de trabajo, salario)
- Variable categórica: es aquella que permite clasificar una serie de datos por medio de valores fijos asociados a una cualidad o categoría concreta. (puesto de trabajo, sexo)

Diferencias:

- Una sirve para contar y la otra para agrupar.
- La variable continua permite cálculos numéricos
- Una describe información cualitativa y la otra cuantitativa.



Transformando las variables (one-hot-encoding)

La forma mas común de representar variables categóricas es la codificación one-hot también conocida como codificación uno de N.

- Consiste en reemplazar la variable categórica con dos o mas variables que pueden tomar valores entre 0 y 1.
- Los valores 0 y 1 tienen sentido en las formulas para clasificación binaria y el resto de modelos de scikit-learn
- Podemos reemplazar cualquier numero de categorías, introduciendo una nueva característica por categoría



Codificando las variables categóricas

Tomemos la variable workclass de nuestro conjunto de datos, así tenemos los posibles valores

- Government Employee
- Private Employee
- Self Employed
- Self Employed Incorporated

Cada uno de estos valores será una nueva característica, donde el valor será 1 si la persona tiene el valor correspondiente y 0 si no lo tiene.



Codificamos una sola variable en 4 nuevas variables

workclass	Government Employee	Private Employee	Self Employed	Self Employed Incorporated
Government Employee	1	0	0	0
Private Employee	0	1	0	0
Self Employed	0	0	1	0
Self Employed Incorporated	0	0	0	1

Implementando el código en Python

- Podemos codificar nuestras variables categóricas usando pandas, en primer lugar importamos los datos

```
import pandas as pd
```

```
data = pd.read_csv( "/home/andy/datasets/adult.data", header=None, index_col=False,  
names=['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation',  
'relationship', 'race', 'gender', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',  
'income'])
```

```
data = data[['age', 'workclass', 'education', 'gender', 'hours-per-week', 'occupation', 'income']]  
display(data.head())
```

Con esta celda obtenemos los datos previamente usados en el ejemplo.

Comprobamos que las columnas tengan información significativa.

- Al crear bases de datos podemos encontrarnos con problemas como que no exista un numero de categorías fijo, o puede haber diferencias en cuanto a la escritura (Es diferente empleado de Empleado). Para ello usamos el siguiente código:

In[3]:

```
print(data.gender.value_counts())
```

Out[3]:

Male 21790

Female 10771

Name: gender, dtype: int64

Podemos comprobar que el genero esta bien definido y puede codificarse correctamente

Para codificar pandas con one-hot usaremos la función `get_dummies`

```
➤ In:
➤ print("Original features:\n",
list(data.columns), "\n")
➤ data_dummies =
pd.get_dummies(data)
➤ print("Features after
get_dummies:\n",
list(data_dummies.columns))
```

	age	hours- per- week	workclass_?	workclass_ Federal- gov	workclass_ Local-gov	...	occupation_ Tech- support	occupation_ Transport- moving	income_ <=50K	income_ >50K
0	39	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
1	50	13	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
2	38	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
3	53	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
4	28	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0



Después de codificar los datos podemos crear un modelo

- Convertimos nuestro marco de datos “data_dummies” en un arreglo NumPy para entrenar el modelo.
- Tenemos que separar la variable objetivo, en este caso el ingreso del resto de los datos.
- Así, extraemos todas las columnas que contienen características, que son todas desde la edad a ocupación:



Extraemos las columnas desde
age hasta occupation_
Transport-moving

```
features = data_dummies.ix[:,  
    'age':'occupation_ Transport-moving']  
  
# Extract NumPy arrays  
  
X = features.values  
  
y = data_dummies['income_ >50K'].values  
  
print("X.shape: {} y.shape:  
{}".format(X.shape, y.shape))
```

Asi obtenemos

X.shape: (32561, 44) y.shape: (32561,)



Ahora scikit-learn puede trabajar con nuestros datos

In:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print("Test score: {:.2f}".format(logreg.score(X_test, y_test)))
```

Out:

Test score: 0.81

Otra forma de codificar categorías.

En el ejemplo anterior al codificar one-hot con pandas se respetan las variables numéricas como la edad

	age	hours-per-week	workclass_?	workclass_Federal-gov	workclass_Local-gov	...	occupation_Tech-support	occupation_Transport-moving	income_<=50K	income_>50K
0	39	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
1	50	13	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
2	38	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
3	53	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
4	28	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0

Podemos usar scikit_learn para evitar esto

Consideremos las siguientes variables, una es categórica y la otra es continua

Categorical Feature		Integer Feature
0	socks	0
1	fox	1
2	socks	2
3	box	1

Aplicando pandas obtenemos

```
pd.get_dummies(demo_df)
```

Integer Feature		Categorical Feature_box	Categorical Feature_fox	Categorical Feature_socks
0	0	0.0	0.0	1.0
1	1	0.0	1.0	0.0
2	2	0.0	0.0	1.0
3	1	1.0	0.0	0.0

Indicamos las variables de las que queremos obtener características.

```
demo_df['Integer Feature'] = demo_df['Integer Feature'].astype(str)  
pd.get_dummies(demo_df, columns=['Integer Feature', 'Categorical Feature'])
```

	Integer Feature_0	Integer Feature_1	Integer Feature_2	Categorical Feature_box	Categorical Feature_fox	Categorical Feature_socks
0	1.0	0.0	0.0	0.0	0.0	1.0
1	0.0	1.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0	1.0
3	0.0	1.0	0.0	1.0	0.0	0.0

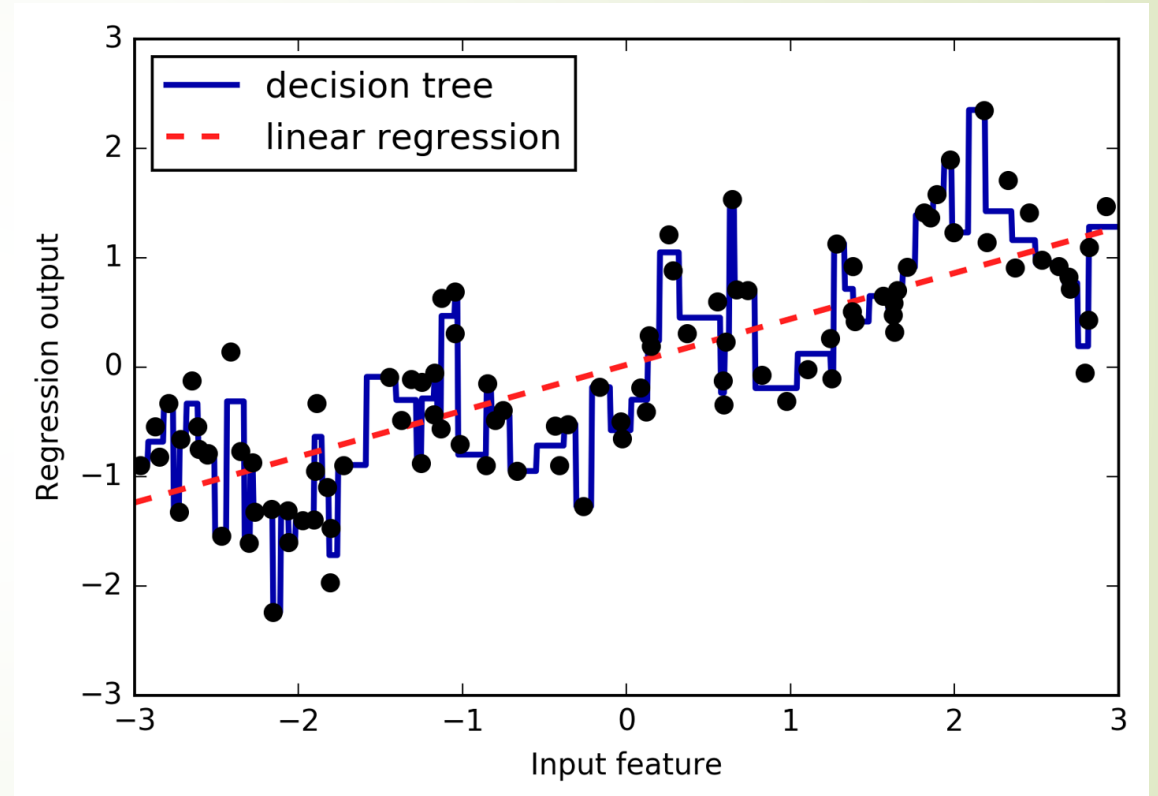



Continuación:

Discretización de datos (binning)

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

X, y = mglearn.datasets.make_wave(n_samples=100)
line = np.linspace(-3, 3, 1000, endpoint=False).reshape(-1, 1)
reg = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
plt.plot(line, reg.predict(line), label="decision tree")
reg = LinearRegression().fit(X, y)
plt.plot(line, reg.predict(line), label="linear regression")
plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")
```





Transformando datos de una sola variable

- Podemos transformar una única variable continua en una categórica para aplica One-Hot.
- Para esto establecemos un intervalo y el numero de particiones que queremos.
- Después asignamos cada valor a su partición correspondiente y obtenemos una variable categórica.
- Finalmente podemos procesar los datos como lo habíamos hecho previamente

Construimos las categorías de la siguiente manera.

- Dividiremos los datos en compartimientos de forma que cada dato se encuentre representado por el compartimento en el que se encuentra.
- En este ejemplo obtenemos 10 compartimentos representados por intervalos.

In:

```
bins = np.linspace(-3, 3, 11)  
print("bins: {}".format(bins))
```

Out:

```
bins: [-3. -2.4 -1.8 -1.2 -0.6  0.  0.6  1.2  1.8  2.4  
 3.]
```



Asignamos cada dato a su
compartimento correspondiente.

In:

```
which_bin = np.digitize(X,  
bins=bins)  
  
print("\nData points:\n", X[:5])  
  
print("\nBin membership for  
data points:\n", which_bin[:5])
```

Out:

Data points:

```
[[ -0.753] [ 2.704] [ 1.392] [ 0.592]  
[ -2.064]]
```

Bin membership for data points:

```
[[ 4] [10] [ 8] [ 6] [ 2]]
```

Aplicamos one-hot en los datos que acabamos de procesar

In:

```
from sklearn.preprocessing import  
OneHotEncoder  
  
encoder = OneHotEncoder(sparse=False)  
encoder.fit(which_bin)  
X_binned = encoder.transform(which_bin)  
print(X_binned[:5])
```

Out:

```
[[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]  
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]  
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

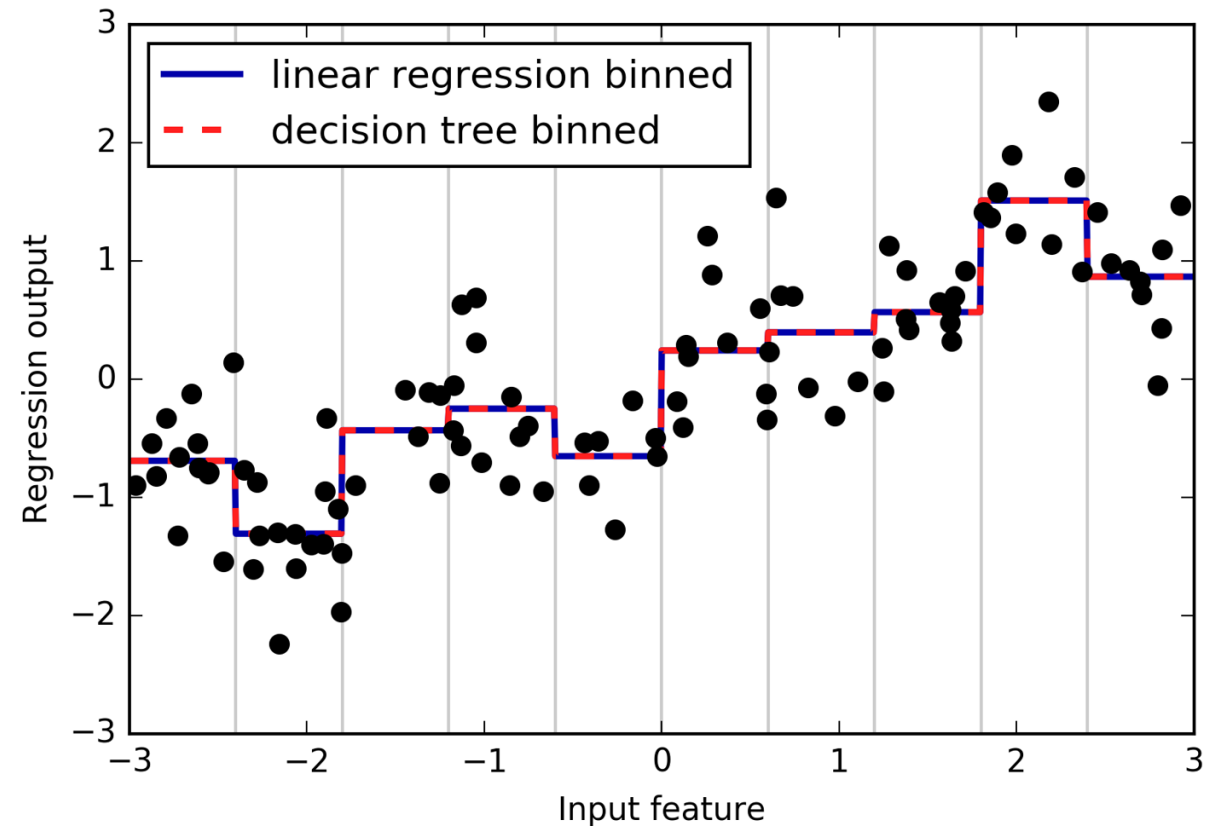
Construimos los modelos nuevamente después de tratar los datos

In[16]:

```
line_binned = encoder.transform(np.digitize(line, bins=bins))
reg = LinearRegression().fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='linear regression
binned')

reg =
DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='decision tree
binned')

plt.plot(X[:, 0], y, 'o', c='k')
plt.vlines(bins, -3, 3, linewidth=1, alpha=.2)
plt.legend(loc="best")
plt.ylabel("Regression output")
plt.xlabel("Input feature")
```





Selección de características (feature selection)

Tenemos varios métodos de seleccionar características, sin embargo:

- Hacemos crecer la dimensión de los datos (muchas veces de manera exponencial como lo hemos visto antes)
- Obtenemos modelos mas complejos
- Aumentan las probabilidades de overfitting

Por estas razones puede ser una buena idea reducir el numero de características , descartando las que sean menos relevantes.



Estadísticas univariantes

- Con el método de estadística univariante evaluamos la relación de cada característica con el objetivo y vemos si hay relación estadística significativa.
- Después seleccionamos las características que se relacionan con mayor nivel de confianza y descartamos aquellas con menor relación.
- Este método se conoce como análisis de varianza (Prueba ANOVA).
- El punto clave de este tipo de prueba es que se analiza cada característica por separado y no se considera su relación con otras características.
- La ventaja de las pruebas univariantes es que suelen ser rápidas y no necesitan un modelo.
- Una desventaja es que es independiente del modelo.



Selección de características univariantes en scikit-learn

Primero necesitamos una prueba de clasificación, usualmente

- `f_classif` (método predeterminado)
- `f_regression`

Después necesitamos un método para descartar características basados en un p-valor.

- Todos los métodos usan un umbral (threshold) para descartar características usando un p-valor grande
- Uno de los métodos mas simples es `SelectKBest`

Selección de características univariantes en un conjunto de datos.

- ▶ Tomaremos como ejemplo la base de datos "breast_cancer"
- ▶ Añadiremos ruido a esta base de datos para probar la selección de características

In:

```
from sklearn.datasets import  
load_breast_cancer  
  
from sklearn.feature_selection import  
SelectPercentile  
  
from sklearn.model_selection import  
train_test_split  
  
cancer = load_breast_cancer()  
rng = np.random.RandomState(42)  
noise = rng.normal(size=(len(cancer.data), 50))  
X_w_noise = np.hstack([cancer.data, noise])
```

Establecemos el conjunto de prueba y aplicamos selección de características

In:

```
X_train, X_test, y_train, y_test = train_test_split(
X_w_noise, cancer.target, random_state=0,
test_size=.5)

select = SelectPercentile(percentile=50)

select.fit(X_train, y_train)

X_train_selected = select.transform(X_train)

print("X_train.shape: {}".format(X_train.shape))

print("X_train_selected.shape:
{}".format(X_train_selected.shape))
```

Out:

```
X_train.shape: (284, 80)
X_train_selected.shape: (284, 40)
```

Comprobemos cuales características fueron eliminadas

In[40]:

```
mask = select.get_support()
```

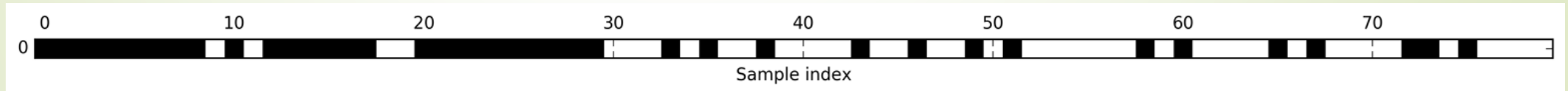
```
print(mask)
```

```
plt.matshow(mask.reshape(1,  
-1), cmap='gray_r')
```

```
plt.xlabel("Sample index")
```

Out:

```
[ True True True True True True True True True True False True False  
 True True True True True True False False True True True True  
 True True True True True True False False False True False True  
 False False True False False False False True False False True  
 False False True False True False False False False False False  
 True False True False False False False True False True False  
 False False False True True False True False False False False]
```



Aplicamos regresión logística en los datos antes y después de la selección de características

In[41]:

```
from sklearn.linear_model import  
LogisticRegression  
X_test_selected = select.transform(X_test)  
lr = LogisticRegression()  
lr.fit(X_train, y_train)  
print("Score with all features:  
{:.3f}".format(lr.score(X_test, y_test)))  
lr.fit(X_train_selected, y_train)  
print("Score with only selected features:  
{:.3f}".format(  
lr.score(X_test_selected, y_test)))
```

Out:

Score with all features: 0.930

Score with only selected features: 0.940



Selección de características basada en modelos (Model-based)

- Este método machine learning supervisado para realizar la selección de características.
- El modelo seleccionado seleccionara las características mas importantes y descartar a las menos relevantes.
- Este modelo de selección de características necesita proveer alguna medida a cada característica para poder ordenarlas en base a su importancia.
- Los arboles de decisión y los modelos basados en esto proveen una esta medida llamada `feature_importances_attributes`
- Los modelos lineales también pueden capturar la importancia considerando el valor absoluto de los coeficientes.

Escogiendo el modelo para extraer las características.

In[42]:

```
from sklearn.feature_selection import  
SelectFromModel  
  
from sklearn.ensemble import  
RandomForestClassifier  
  
select = SelectFromModel(  
    RandomForestClassifier(n_estimators=100,  
                           random_state=42),  
    threshold="median")
```

- El modulo SelectFromModel selecciona todas las características que tienen una medida de importancia mayor al umbral seleccionado.
- En este caso consideramos la mediana como umbral
- Así seleccionamos la mitad de las característica y lo podemos comparar con el método univariante

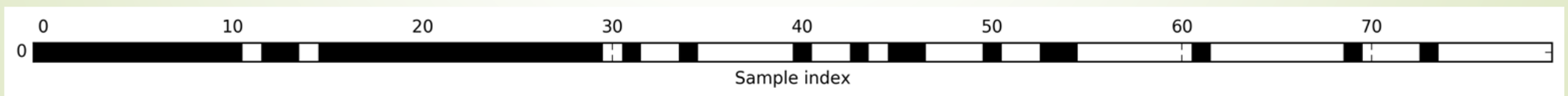
Aplicando el modelo a los datos del cáncer

In:

```
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Sample index")
```

Out:

```
X_train.shape: (284, 80)
X_train_l1.shape: (284, 40)
```





Aplicamos regresión logista a nuestros datos y comprobamos su desempeño

In:

```
X_test_l1 = select.transform(X_test)

score = LogisticRegression().fit(X_train_l1,
y_train).score(X_test_l1, y_test)

print("Test score: {:.3f}".format(score))
```

Out:

Test score: 0.951

Hubo una mejora considerando el método univariante



Selección de características iterativo (Iterative Feature Selection)

- Con el método univariante no usamos ningún modelo, mientras que con el método basado en modelos usamos un solo modelo para seleccionar las características.
- Para la selección de características iterativa construiremos una serie de modelos para seleccionar las características.
- Este método es mas costoso computacionalmente.



Tipos de selección iterativa

El primer método consiste en crear un modelo cuyas entradas sean solamente los datos.

- ▶ Así, nuestro modelo empieza a añadir características hasta que un se alcance un criterio deseado.

El segundo método consiste en proporcionar los datos y características al modelo.

- ▶ De esta forma nuestro modelo eliminara características has alcanzar el criterio que deseamos

Eliminación de características recursiva (RFE)

- Este método inicia con los datos y todas las características .
- Primero crea un modelo y descarta la característica menos importante.
- Después vuelve a crear el modelo usando todas las características menos la descartada y elimina la característica menos importante de las restantes.
- Se repite el proceso hasta que se elimino el numero de características deseadas.

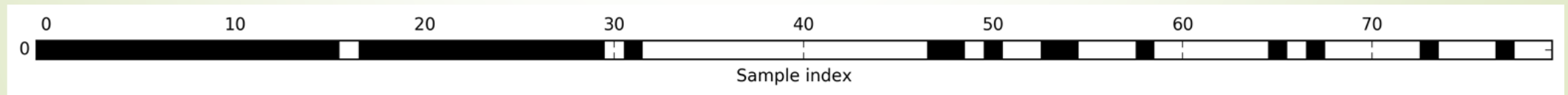
In:

```
from sklearn.feature_selection import RFE  
  
select =  
RFE(RandomForestClassifier(n_estimators=100,  
random_state=42), n_features_to_select=40)
```

Aplicamos este método a los datos del cáncer

In:

```
select.fit(X_train, y_train)  
mask = select.get_support()  
plt.matshow(mask.reshape(1, -1), cmap='gray_r')  
plt.xlabel("Sample index")
```



Desempeño del modelo después de la regresión de características

In[47]:

```
X_train_rfe= select.transform(X_train)
X_test_rfe= select.transform(X_test)
score = LogisticRegression().fit(X_train_rfe,
y_train).score(X_test_rfe, y_test)
print("Test score: {:.3f}".format(score))
#comparamos con random forest
print("Test score random forest:
{:.3f}".format(select.score(X_test, y_test)))
```

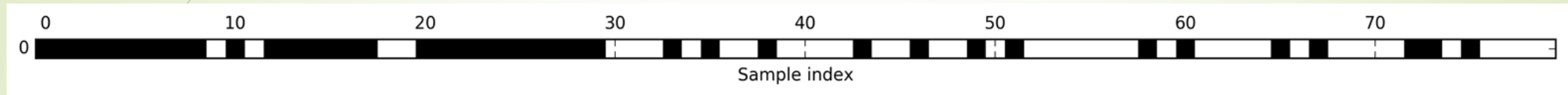
Out:

Test score: 0.951

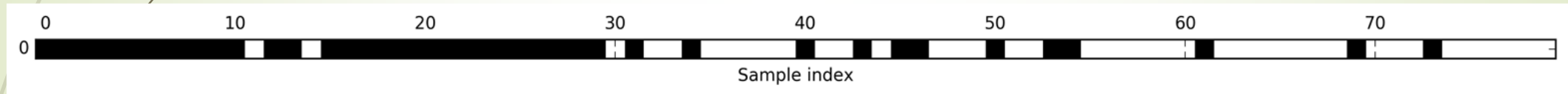
Test score random forest: 0.951

Podemos ver que el modelo lineal tiene el mismo desempeño que random forest

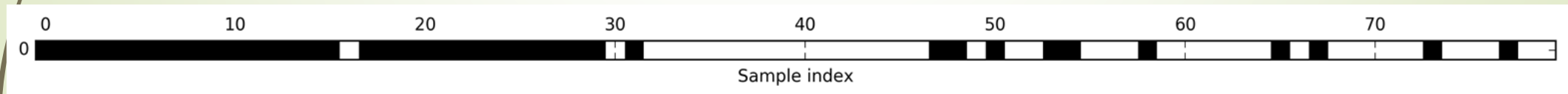
Comparando los métodos de selección de características.



Selección de características univariante



Selección de características basada en modelos



Selección de características iterativa



Gracias