ANN-Intro-Parte-2

May 19, 2019

Por: V. Robles B.

1 Redes Neuronales con Scikit-Learn: una introducción - Parte 2

En este cuaderno se presenta una breve introducción de los principales aspectos para crear, entrenar y validar redes neuronales artificiales en Python con la librería scikit-learn. A lo largo del cuaderno se hará especial énfasis en el **Perceptrón multicapa** como herramienta para realizar tareas de clasificación.

1.1 Ejemplo básico 2: Corpus Iris

El corpus Iris es uno de los ejemplos más utilizados para aprender a diseñar redes neuronales. Este corpus tiene las siguientes características:

- Posee 150 muestras de 3 tipos de flores (50 c/u): Iris-Setosa, Iris-Virgínica e Iris-Versicolor.
- Cada muestra contiene 4 variables (decimales) que representan estas características de cada flor:
  - Longitud del pétalo
  - Longitud del sépalo
  - Grosor del pétalo
  - Grosor del sépalo

El objetivo consiste en diseñar y entrenar una red neuronal que permita clasificar las muestras en base a las características de dichas flores.

1.1.1 Lectura de los datos

Como primer paso procederemos a cargar los datos empleando para ello la librería Pandas. Para ello, emplearemos el método **read_csv** y dado que el fichero no tiene nombres en cada columna (los datos inician en la primera fila), especificamos los nombres que deseamos que sean cargados al leer el fichero.

Es importante observar, que el método **read_csv** devuelve un objeto dataframe.

```
In [29]: import pandas as pd
         import numpy as np

         %matplotlib inline
```

```
datos = pd.read_csv('corpus/iris/iris.data',names=['longitud_sepalo','grosor_sepalo',\
                                                   'longitud_petalo','grosor_petalo','f

datos.head()
```

Out[29]:

	longitud_sepalo	grosor_sepalo	longitud_petalo	grosor_petalo	flor
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

También podemos analizar el corpus empleando la función **describe** que provee **Pandas**. Los valores que podemos obtener de cada variable, son los siguientes: * Conteo (*count*) * Media (*mean*) * Desviación estándar (*std*) * Mínimo (*min*) * Percentiles, que son los valores que están entre *25%, 50%, 75%*. * Máximo (*max*)

```
In [30]: datos.describe().transpose()
```

Out[30]:

	count	mean	std	min	25%	50%	75%	max
longitud_sepalo	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
grosor_sepalo	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
longitud_petalo	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9
grosor_petalo	150.0	1.198667	0.763161	0.1	0.3	1.30	1.8	2.5

Como podemos apreciar, la última columna contiene cadenas de texto que describen el tipo de flor. Para ello, podemos emplear la función map que provee **Pandas** y reemplazar las cadenas por valores numéricos que sí puedan ser entendidos por la red neuronal.

```
In [31]: datos['flor']=datos['flor'].map({'Iris-setosa':0, 'Iris-versicolor':1,'Iris-virginica':
         datos.head()
```

Out[31]:

	longitud_sepalo	grosor_sepalo	longitud_petalo	grosor_petalo	flor
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

1.1.2 Diseño de la Red Neuronal

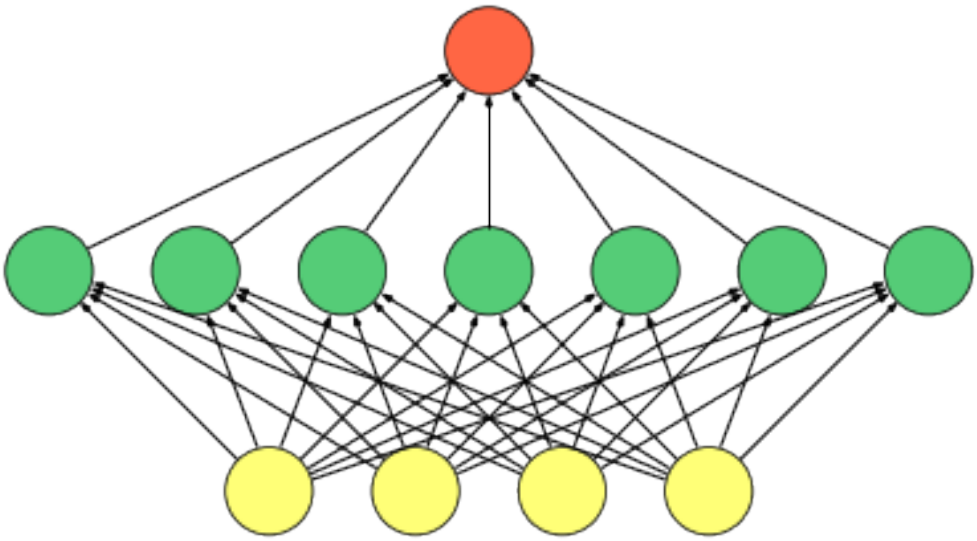
Como siguiente punto, diseñaremos una red neuronal (clasificador perceptrón multicapa) para poder aprender a distinguir entre los diferentes tipos de flores.

La red neuronal artificial tendrá las sigiuentes características:

- Entradas: 4
- Número de capas: 2
- Neuronas en la capa oculta: 7

- Neuronas en la capa de salida: 1

```
In [32]: from viznet import connecta2a, node_sequence, NodeBrush, EdgeBrush, DynamicShow

         # Creamos variables con los parametros que tendra la red
         entradas = 4
         neuronas_capa_oculta = 7
         neuronas_capa_salida = 1


         def dibujar_red_neuronal(ax, num_node_list):

             num_hidden_layer = len(num_node_list) - 2
             token_list = ['\sigma^z'] + \
                 ['y^{(%s)}' % (i + 1) for i in range(num_hidden_layer)] + ['\psi']
             kind_list = ['nn.input'] + ['nn.hidden'] * num_hidden_layer + ['nn.output']
             radius_list = [0.3] + [0.2] * num_hidden_layer + [0.3]
             y_list = 1.5 * np.arange(len(num_node_list))

             seq_list = []
             for n, kind, radius, y in zip(num_node_list, kind_list, radius_list, y_list):
                 b = NodeBrush(kind, ax)
                 seq_list.append(node_sequence(b, n, center=(0, y)))

             eb = EdgeBrush('-->', ax)
             for st, et in zip(seq_list[:-1], seq_list[1:]):
                 connecta2a(st, et, eb)


         def real_bp():
             with DynamicShow((6, 6), '_feed_forward.png') as d:
                 dibujar_red_neuronal(d.ax, num_node_list=[entradas, neuronas_capa_oculta, neuro


         real_bp()

Press `c` to save figure to "_feed_forward.png", `Ctrl+d` to break >>
> /home/frank/.anaconda3/lib/python3.7/site-packages/viznet/context.py(61)__exit__()
-> plt.savefig(self.filename, dpi=300, transparent=True)
(Pdb) c
```



1.1.3 Preprocesamiento de los datos y generación de los corpus de entrenamiento y pruebas

Como paso previo a entrenar la red neuronal, es fundamental preprocesar los datos (escalar, cambiar formatos, etc.), ya que en caso contrario no se obtendrán resultados óptimos en el proceso de clasificación.

```
In [33]: # Importamos la función para separar test y train
         from sklearn.model_selection import train_test_split

         # Importamos la función para escalar los valores
         from sklearn.preprocessing import StandardScaler
```

```
        # Separamos en una variable los datos de entrada, para ello generamos una copia del dat
        # eliminando la última columna del corpus (la que tiene los tipos de flores)
        X=datos.drop('flor',axis=1)

        # Procedemos de la misma forma, pero en este caso para generar un arreglo que tenga las
        # deseadas
        d=datos['flor']

        # Mostramos en pantalla los primeros datos con la función 'head'
        X.head()
        d.head()

        # Generamos los corpus para entrenamiento y pruebas de modo que se tome el mismo número
        X_train, X_test, d_train, d_test = train_test_split(X,d,train_size=0.70,random_state=0,


        # Generamos un objeto para escalar los valores
        scaler=StandardScaler()

        print(scaler)

        # Ajuste solo en los datos de entrenamiento
        scaler.fit(X_train)

        # Escalamos el corpus de entrenamiento
        X_train=scaler.transform(X_train)
        X_test=scaler.transform(X_test)

        # Visualizamos las 7 primeras filas de datos
        X_train[1:7,:]
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
/home/frank/.anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2179: Futur
  FutureWarning)
```

```
Out[33]: array([[ 2.14194987,  1.67393943,  1.63709442,  1.31005616],
                [ 0.62547602,  0.34908787,  0.85828251,  1.44106177],
                [-1.47425699,  1.23232224, -1.53378264, -1.31005616],
                [-0.77434598,  0.79070506, -1.31126495, -1.31005616],
                [-1.70756066, -0.09252931, -1.36689437, -1.31005616],
                [-1.12430149, -1.41738087, -0.25430593, -0.26201123]])
```

1.1.4 Creación, entrenamiento y validación de la Red Neuronal

A continuación emplearemos **sckikit learn** para crear, entrenar y probar la red neuronal MLPClassifier que se especificó con anterioridad. Los parámetros que se usarán son los siguientes:

- Algoritmo para la reducción del error en el entrenamiento: **lbfgs** optimizador basado en métodos cuasi-Newtonianos. Mayor información en este link.
- Función de activación de las neuronas: **logística** (*logistic*)
- Máximo número de iteraciones (*max_iter*): 10000

```
In [37]: # Importamos el Perceptron Multicapa para Clasificacion
         from sklearn.neural_network import MLPClassifier

         # Creamos la red neuronal
         mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True, alpha=1e-4, to
                           hidden_layer_sizes=(neuronas_capa_oculta, neuronas_capa_salida))

         print(mlp)
         # Realizamos el proceso de entrenamiento
         mlp.fit(X_train, d_train)
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(7, 1), learning_rate='constant',
       learning_rate_init=0.001, max_iter=10000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
       validation_fraction=0.1, verbose=True, warm_start=False)


Out[37]: MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
                beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(7, 1), learning_rate='constant',
                learning_rate_init=0.001, max_iter=10000, momentum=0.9,
                n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
                validation_fraction=0.1, verbose=True, warm_start=False)
```

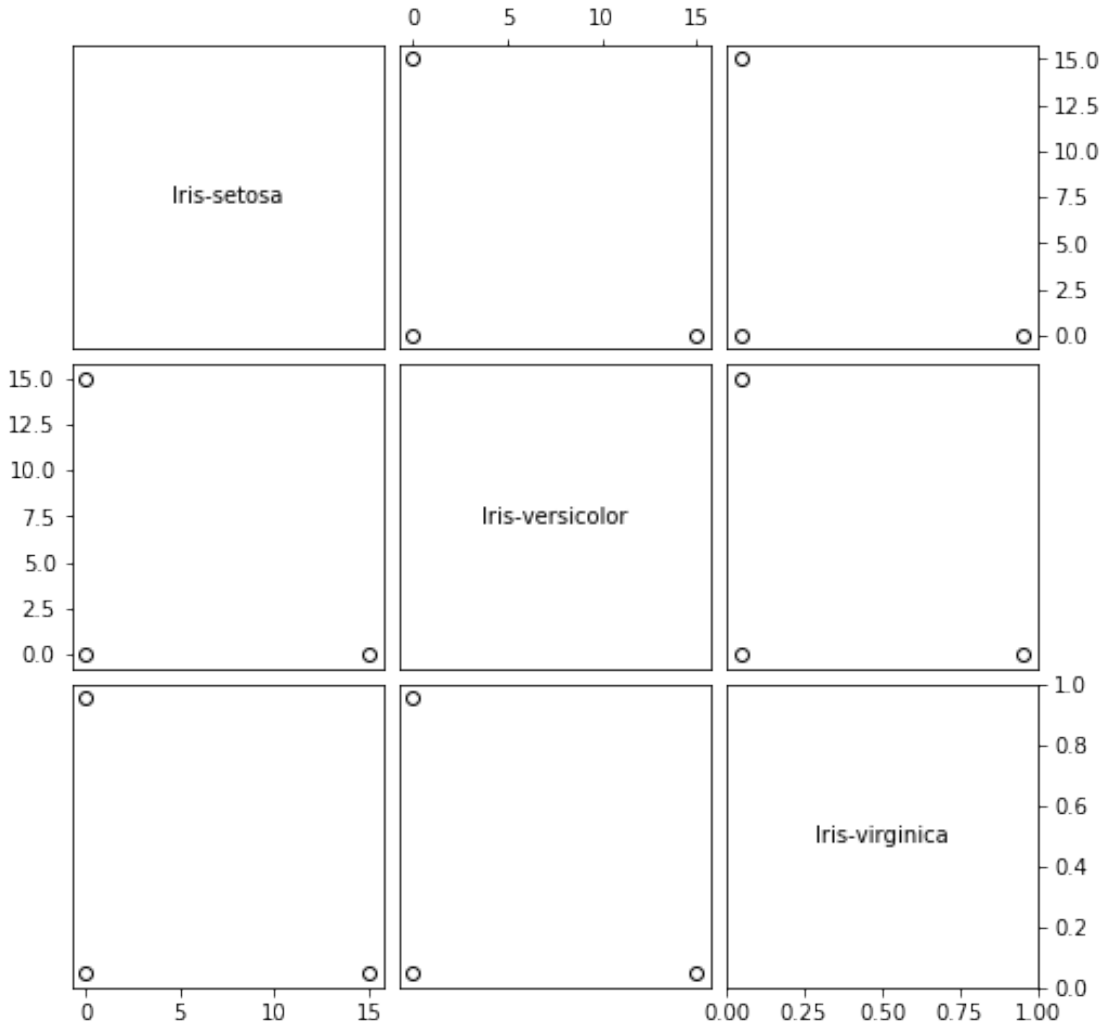
1.1.5 Predicción y evaluación de la red

El último paso es evaluar el funcionamiento de la red. Para ello, determinaremos cómo se comporta en tareas de predicción con la parte de pruebas (**X_test, d_test**):

```
In [39]: from sklearn.metrics import classification_report, confusion_matrix
         import matplotlib.pyplot as pp
         #d_test = d_test.map({0:'Iris-setosa', 1: 'Iris-versicolor',2:'Iris-virginica'})
         print(d_test.value_counts())

         prediccion = mlp.predict(X_test)
         print('Matriz de Confusion\n')
         matriz = confusion_matrix(d_test, prediccion)
         print(confusion_matrix(d_test, prediccion))
         print('\n')
```

```
        print(classification_report(d_test, prediccion))

        fig = scatterplot_matrix(matriz, ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
                    linestyle='none', marker='o', color='black', mfc='none')
        pp.show()
```

```
2    15
1    15
0    15
Name: flor, dtype: int64
Matriz de Confusion

[[15  0  0]
 [ 1 13  1]
 [ 0  0 15]]


              precision    recall  f1-score   support

           0       0.94      1.00      0.97        15
           1       1.00      0.87      0.93        15
           2       0.94      1.00      0.97        15

   micro avg       0.96      0.96      0.96        45
   macro avg       0.96      0.96      0.95        45
weighted avg       0.96      0.96      0.95        45
```



In [12]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt

def scatterplot_matrix(data, names, **kwargs):
    """Plots a scatterplot matrix of subplots.  Each row of "data" is plotted
    against other rows, resulting in a nrows by nrows grid of subplots with the
    diagonal subplots labeled with "names".  Additional keyword arguments are
    passed on to matplotlib's "plot" command. Returns the matplotlib figure
    object containg the subplot grid."""
    numvars, numdata = data.shape
    fig, axes = plt.subplots(nrows=numvars, ncols=numvars, figsize=(8,8))
    fig.subplots_adjust(hspace=0.05, wspace=0.05)

    for ax in axes.flat:
```

```
            # Hide all ticks and labels
            ax.xaxis.set_visible(False)
            ax.yaxis.set_visible(False)

            # Set up ticks only on one side for the "edge" subplots...
            if ax.is_first_col():
                ax.yaxis.set_ticks_position('left')
            if ax.is_last_col():
                ax.yaxis.set_ticks_position('right')
            if ax.is_first_row():
                ax.xaxis.set_ticks_position('top')
            if ax.is_last_row():
                ax.xaxis.set_ticks_position('bottom')

        # Plot the data.
        for i, j in zip(*np.triu_indices_from(axes, k=1)):
            for x, y in [(i,j), (j,i)]:
                axes[x,y].plot(data[x], data[y], **kwargs)

        # Label the diagonal subplots...
        for i, label in enumerate(names):
            axes[i,i].annotate(label, (0.5, 0.5), xycoords='axes fraction',
                    ha='center', va='center')

        # Turn on the proper x or y axes ticks.
        for i, j in zip(range(numvars), itertools.cycle((-1, 0))):
            axes[j,i].xaxis.set_visible(True)
            axes[i,j].yaxis.set_visible(True)

        return fig
```

1.1.6 Práctica ANN-3:

Modifique el código anterior, a fin de incorporar las siguientes modificaciones:

- Incorporar Hot Encoding y contar con 3 salidas en lugar de 1. Compare los resultados.
- Generar una gráfica donde se puedan analizar las 4 variables y cómo se distribuyen la mismas (analizadas de dos en dos). Para ello, se recomienda emplear el siguente ejemplo:

- Pruebe la misma red neuronal pero sin escalar los datos y compare los resultados.

```
In [24]: # TO-DO:
         # Emplear el código anterior a fin resolver los aspectos planteados en la práctica ANN-
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.preprocessing import LabelEncoder
         import pandas as pd
         import numpy as np

         %matplotlib inline
```

```
datos = pd.read_csv('corpus/iris/iris.data',names=['longitud_sepalo','grosor_sepalo',\
                                                   'longitud_petalo','grosor_petalo','f
datos.head()

datos.describe().transpose()

le_iris = LabelEncoder()
iris_ohe = OneHotEncoder()

datos['flor']=le_iris.fit_transform(datos.flor)
datos.head()

X = iris_ohe.fit_transform(datos.flor.values.reshape(-1,1)).toarray()

dfOneHot = pd.DataFrame(X, columns = [""+str(int(i)) for i in range(X.shape[1])])
datos = pd.concat([datos, dfOneHot], axis=1)

datos.head()
```

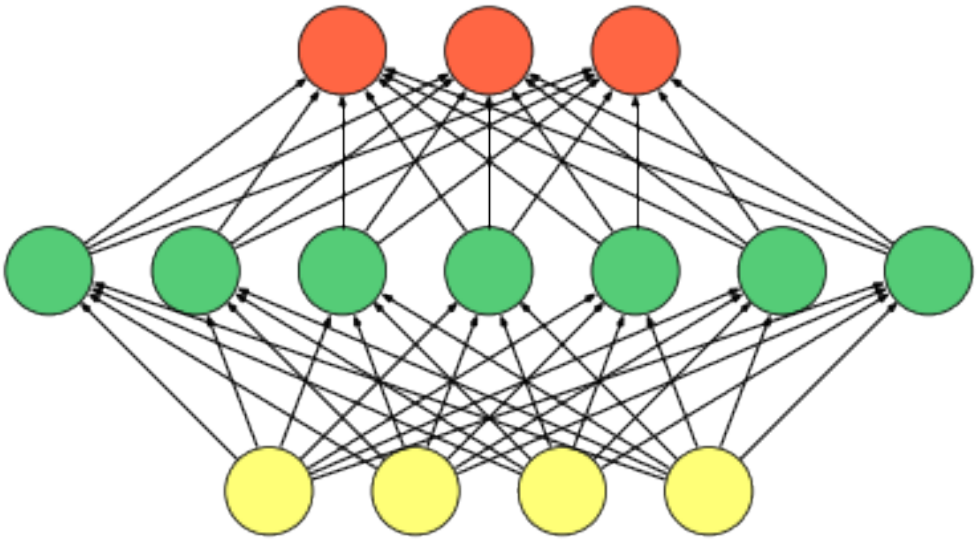
```
/home/frank/.anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_encoders.py:368: Futur
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers,
  warnings.warn(msg, FutureWarning)
```

Out[24]:

	longitud_sepalo	grosor_sepalo	longitud_petalo	grosor_petalo	flor	0	1	2
0	5.1	3.5	1.4	0.2	0	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	0	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	0	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	0	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	0	1.0	0.0	0.0

In [8]:
```
from viznet import connecta2a, node_sequence, NodeBrush, EdgeBrush, DynamicShow

# Creamos variables con los parametros que tendra la red
entradas = 4
neuronas_capa_oculta = 7
neuronas_capa_salida = 3
```

```
        def dibujar_red_neuronal(ax, num_node_list):

            num_hidden_layer = len(num_node_list) - 2
            token_list = ['\sigma^z'] + \
                ['y^{(%s)}' % (i + 1) for i in range(num_hidden_layer)] + ['\psi']
            kind_list = ['nn.input'] + ['nn.hidden'] * num_hidden_layer + ['nn.output']
            radius_list = [0.3] + [0.2] * num_hidden_layer + [0.3]
            y_list = 1.5 * np.arange(len(num_node_list))

            seq_list = []
            for n, kind, radius, y in zip(num_node_list, kind_list, radius_list, y_list):
                b = NodeBrush(kind, ax)
                seq_list.append(node_sequence(b, n, center=(0, y)))

            eb = EdgeBrush('-->', ax)
            for st, et in zip(seq_list[:-1], seq_list[1:]):
                connecta2a(st, et, eb)


        def real_bp():
            with DynamicShow((6, 6), '_feed_forward.png') as d:
                dibujar_red_neuronal(d.ax, num_node_list=[entradas, neuronas_capa_oculta, neuron


        real_bp()

Press `c` to save figure to "_feed_forward.png", `Ctrl+d` to break >>
> /home/frank/.anaconda3/lib/python3.7/site-packages/viznet/context.py(61)__exit__()
-> plt.savefig(self.filename, dpi=300, transparent=True)
(Pdb) c
```



In [45]:
```
# Importamos la función para separar test y train
from sklearn.model_selection import train_test_split

# Importamos la función para escalar los valores
from sklearn.preprocessing import StandardScaler


# Separamos en una variable los datos de entrada, para ello generamos una copia del dat
# eliminando la última columna del corpus (la que tiene los tipos de flores)
X=datos.drop('flor',axis=1)

# Procedemos de la misma forma, pero en este caso para generar un arreglo que tenga las
# deseadas
```

```
        d=datos['flor']

        # Mostramos en pantalla los primeros datos con la función 'head'
        X.head()
        d.head()

        # Generamos los corpus para entrenamiento y pruebas de modo que se tome el mismo número
        X_train, X_test, d_train, d_test = train_test_split(X,d,train_size=0.70,random_state=0,


        # Generamos un objeto para escalar los valores
        scaler=StandardScaler()

        print(scaler)

        # Ajuste solo en los datos de entrenamiento
        scaler.fit(X_train)

        # Escalamos el corpus de entrenamiento
        X_train=scaler.transform(X_train)
        X_test=scaler.transform(X_test)

        # Visualizamos las 7 primeras filas de datos
        X_train[1:7,:]
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
Out[45]: array([[ 2.14194987,  1.67393943,  1.63709442,  1.31005616],
                [ 0.62547602,  0.34908787,  0.85828251,  1.44106177],
                [-1.47425699,  1.23232224, -1.53378264, -1.31005616],
                [-0.77434598,  0.79070506, -1.31126495, -1.31005616],
                [-1.70756066, -0.09252931, -1.36689437, -1.31005616],
                [-1.12430149, -1.41738087, -0.25430593, -0.26201123]])
```

```
In [27]: # Importamos el Perceptron Multicapa para Clasificacion
         from sklearn.neural_network import MLPClassifier

         # Creamos la red neuronal
         mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True, alpha=1e-4, to
                           hidden_layer_sizes=(neuronas_capa_oculta, neuronas_capa_salida))

         print(mlp)
         # Realizamos el proceso de entrenamiento
         mlp.fit(X_train, d_train)
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(7, 3), learning_rate='constant',
```

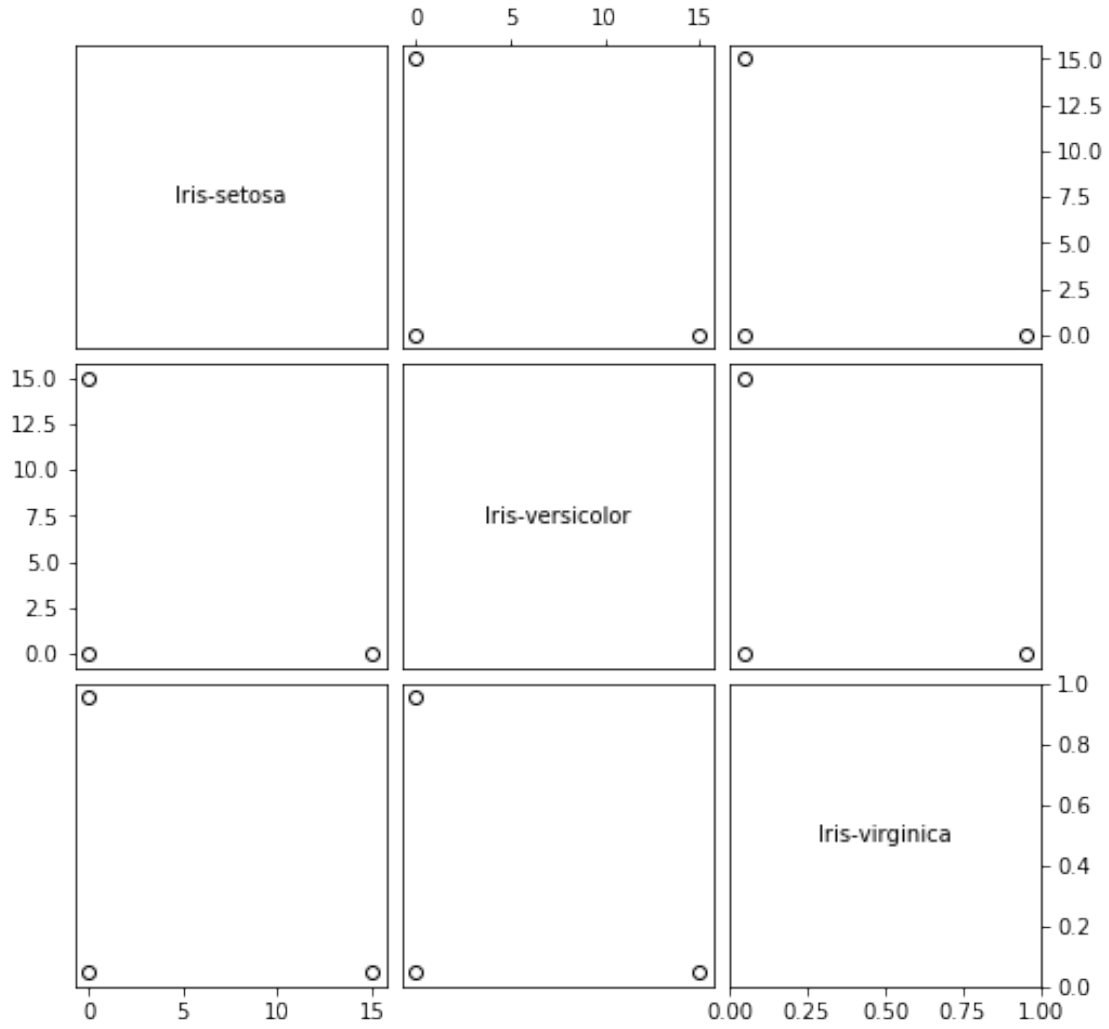
```
       learning_rate_init=0.001, max_iter=10000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
       validation_fraction=0.1, verbose=True, warm_start=False)
```

```
Out[27]: MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
                       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
                       hidden_layer_sizes=(7, 3), learning_rate='constant',
                       learning_rate_init=0.001, max_iter=10000, momentum=0.9,
                       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                       random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
                       validation_fraction=0.1, verbose=True, warm_start=False)
```

```
In [28]: from sklearn.metrics import classification_report, confusion_matrix
         import matplotlib.pyplot as pp
         #d_test = d_test.map({0:'Iris-setosa', 1: 'Iris-versicolor',2:'Iris-virginica'})
         print(d_test.map({0:'Iris-setosa', 1: 'Iris-versicolor',2:'Iris-virginica'}).value_coun

         prediccion = mlp.predict(X_test)
         print('Matriz de Confusion\n')
         matriz = confusion_matrix(d_test, prediccion)
         print(confusion_matrix(d_test, prediccion))
         print('\n')

         print(classification_report(d_test, prediccion))

         fig = scatterplot_matrix(matriz, ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
                     linestyle='none', marker='o', color='black', mfc='none')
         pp.show()
```

```
Iris-setosa        15
Iris-virginica     15
Iris-versicolor    15
Name: flor, dtype: int64
Matriz de Confusion

[[15  0  0]
 [ 0 15  0]
 [ 0  0 15]]


              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00        15
```

```
   micro avg       1.00      1.00      1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



1.1.7 Práctica ANN-4:

Reproducir el mismo experimento para el corpus del censo realizado en el año 1996 (**censo**), considerando las siguientes premisas:

1. Realizar todas las fases de diseño, entrenamiento y validación de la red neuronal
2. Probar la red con el método de validación visto, y realizar el contraste de forma manual
3. Realizar la gráfica de los datos

In [2]:
```
import pandas as pd
import numpy as np

%matplotlib inline

datos = pd.read_csv('corpus/censo/censo1.data',names=['age','workclass', 'fnlwgt', 'educ
                                                      'education_num', 'marital_status',
                                                      'relationship', 'race', 'sex', 'cap
                                                      'capital_loss', 'hours_per_week', '

datos.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education_num	\
0	39	State-gov	77516	Bachelors	13	
1	50	Self-emp-not-inc	83311	Bachelors	13	
2	38	Private	215646	HS-grad	9	
3	53	Private	234721	11th	7	
4	28	Private	338409	Bachelors	13	

	marital_status	occupation	relationship	race	sex	\
0	Never-married	Adm-clerical	Not-in-family	White	Male	
1	Married-civ-spouse	Exec-managerial	Husband	White	Male	
2	Divorced	Handlers-cleaners	Not-in-family	White	Male	
3	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	
4	Married-civ-spouse	Prof-specialty	Wife	Black	Female	

	capital_gain	capital_loss	hours_per_week	native_country	salida
0	2174	0	40	United-States	<=50K
1	0	0	13	United-States	<=50K
2	0	0	40	United-States	<=50K
3	0	0	40	United-States	<=50K
4	0	0	40	Cuba	<=50K

In [3]:
```
datos.describe().transpose()
```

Out[3]:

	count	mean	std	min	25%	50%	\
age	500.0	37.984	13.449965	17.0	28.0	36.5	
fnlwgt	500.0	195607.564	115217.251793	21174.0	116563.5	183923.0	
education_num	500.0	10.076	2.599951	1.0	9.0	10.0	
capital_gain	500.0	564.648	2646.056895	0.0	0.0	0.0	
capital_loss	500.0	107.898	440.092589	0.0	0.0	0.0	
hours_per_week	500.0	39.588	11.795445	1.0	40.0	40.0	

	75%	max
age	46.00	90.0
fnlwgt	250379.00	1033222.0
education_num	12.00	16.0
capital_gain	0.00	34095.0

```
capital_loss         0.00     2415.0
hours_per_week      40.25       98.0
```

In [4]:
```
from sklearn.preprocessing import LabelEncoder

le_workclass = LabelEncoder()
le_education = LabelEncoder()
le_marital_status = LabelEncoder()
le_occupation = LabelEncoder()
le_relationship = LabelEncoder()
le_race = LabelEncoder()
le_sex = LabelEncoder()
le_native_country = LabelEncoder()
le_salida = LabelEncoder()

datos['workclass'] = le_workclass.fit_transform(datos.workclass)
datos['education'] = le_education.fit_transform(datos.education)
datos['marital_status'] = le_marital_status.fit_transform(datos.marital_status)
datos['occupation'] = le_occupation.fit_transform(datos.occupation)
datos['relationship'] = le_relationship.fit_transform(datos.relationship)
datos['race'] = le_race.fit_transform(datos.race)
datos['sex'] = le_sex.fit_transform(datos.sex)
datos['native_country'] = le_native_country.fit_transform(datos.native_country)
datos['salida'] = le_salida.fit_transform(datos.salida)

datos.head()
```

Out[4]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	salida
0	39	6	77516	9	13	4	1	1	4	1	2174	0	40	25	0
1	50	5	83311	9	13	2	4	0	4	1	0	0	13	25	0
2	38	3	215646	11	9	0	6	1	4	1	0	0	40	25	0
3	53	3	234721	1	7	2	6	0	2	1	0	0	40	25	0
4	28	3	338409	9	13	2	9	5	2	0	0	0	40	4	0

In [9]:
```
# Importamos la función para separar test y train
```

```
from sklearn.model_selection import train_test_split

# Importamos la función para escalar los valores
from sklearn.preprocessing import StandardScaler


# Separamos en una variable los datos de entrada, para ello generamos una copia del data
# eliminando la última columna del corpus (la que tiene los tipos de flores)
X=datos.drop('salida',axis=1)

# Procedemos de la misma forma, pero en este caso para generar un arreglo que tenga las
# deseadas
d=datos['salida']

# Mostramos en pantalla los primeros datos con la función 'head'
X.head()
d.head()

# Generamos los corpus para entrenamiento y pruebas de modo que se tome el mismo número
X_train, X_test, d_train, d_test = train_test_split(X,d,train_size=0.70,random_state=0,s


# Generamos un objeto para escalar los valores
scaler=StandardScaler()

print(scaler)

# Ajuste solo en los datos de entrenamiento
scaler.fit(X_train)

# Escalamos el corpus de entrenamiento
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)

# Visualizamos las 7 primeras filas de datos
X_train[1:7,:]
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)


/home/frank/.anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2179: Futur
  FutureWarning)
/home/frank/.anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConver
  return self.partial_fit(X, y)
/home/frank/.anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:33: DataConversionWarni
/home/frank/.anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:34: DataConversionWarni


Out[9]: array([[-0.10166844,  0.02096035,  0.07189987,  1.22791755, -0.02868032,
```

```
       -1.74481322,  0.38864357,  1.47038956,  0.42311396, -1.33333333,
       -0.20473975, -0.22319133,  0.13593855, -1.30739528],
      [-1.3805843 ,  0.02096035, -0.78495373,  0.19481384, -0.4147615 ,
        1.03734909,  1.1615143 ,  1.47038956, -1.85519198,  0.75      ,
       -0.20473975, -0.22319133, -0.70318828, -2.70858445],
      [ 0.72586535,  1.6512099 , -0.41073044,  0.45308977,  1.51564441,
        1.03734909,  0.64626714, -0.3311379 ,  0.42311396, -1.33333333,
       -0.20473975, -0.22319133,  0.97506537,  0.29396377],
      [-1.53104499,  0.02096035, -1.10712304, -2.38794543, -1.18692386,
        1.03734909,  1.1615143 ,  0.8698804 ,  0.42311396, -1.33333333,
       -0.20473975, -0.22319133, -2.21361657,  0.29396377],
      [-0.85397188,  0.02096035, -1.15753059, -0.32173801,  1.12956323,
        1.03734909,  1.1615143 , -0.3311379 ,  0.42311396,  0.75      ,
       -0.20473975, -0.22319133,  0.13593855,  0.29396377],
      [-0.10166844,  0.02096035, -0.58002233, -0.32173801,  1.12956323,
       -0.35373207,  0.38864357, -0.93164706, -1.85519198,  0.75      ,
        2.43591961, -0.22319133, -0.19971218,  0.29396377]])
```

In [14]:
```
# Importamos el Perceptron Multicapa para Clasificacion
from sklearn.neural_network import MLPClassifier

# Creamos la red neuronal
mlp=MLPClassifier(solver = 'lbfgs', activation='logistic', verbose=True, alpha=1e-4, to
                  hidden_layer_sizes=(100, 2))

print(mlp)
# Realizamos el proceso de entrenamiento
mlp.fit(X_train, d_train)
```

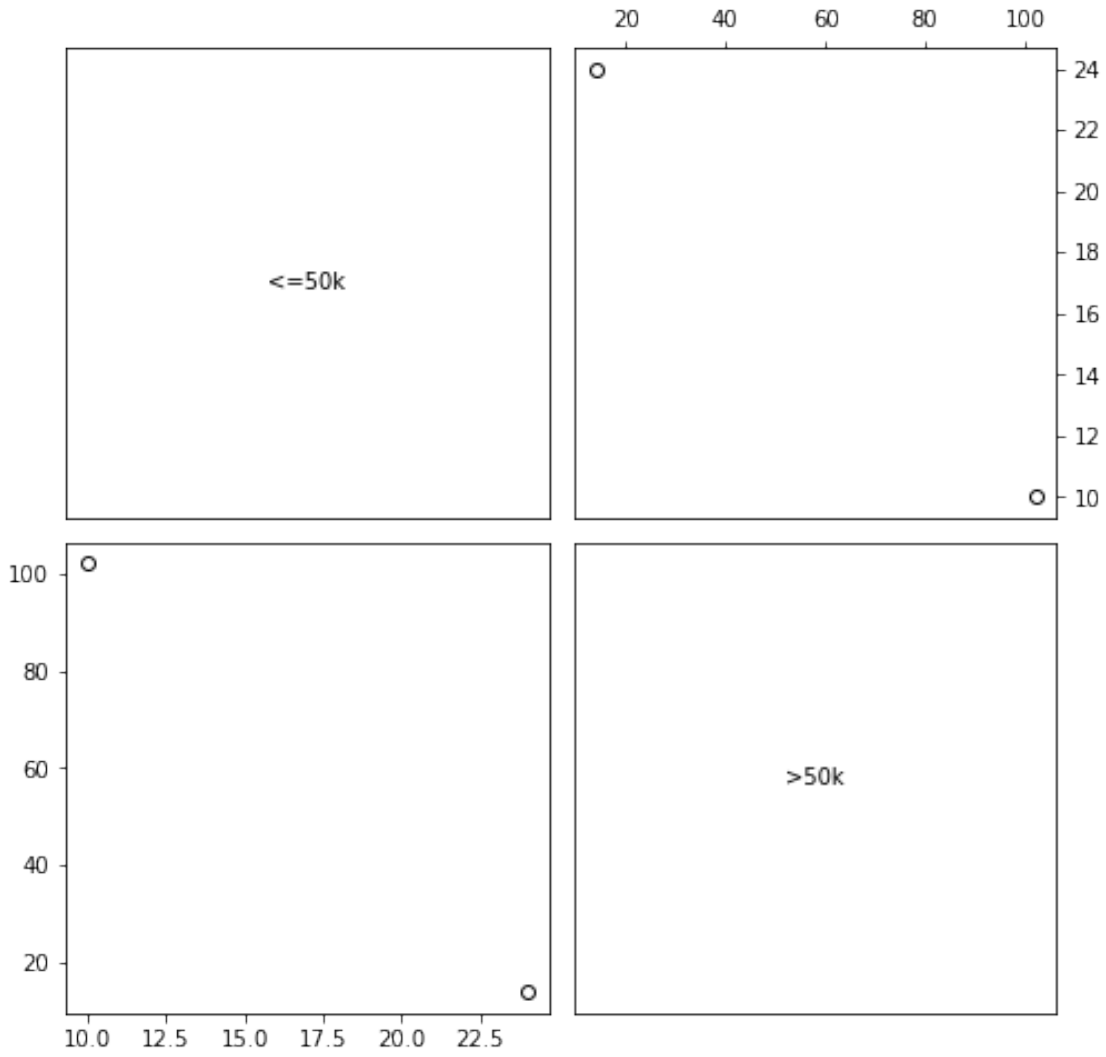
```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 2), learning_rate='constant',
       learning_rate_init=0.001, max_iter=10000, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
       validation_fraction=0.1, verbose=True, warm_start=False)
```

Out[14]:
```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100, 2), learning_rate='constant',
              learning_rate_init=0.001, max_iter=10000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='lbfgs', tol=1e-15,
              validation_fraction=0.1, verbose=True, warm_start=False)
```

In [15]:
```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as pp
#d_test = d_test.map({0:'Iris-setosa', 1: 'Iris-versicolor',2:'Iris-virginica'})
```

```
        print(d_test.map({0:'<=50k', 1: '>50k'}).value_counts())

        prediccion = mlp.predict(X_test)
        print('Matriz de Confusion\n')
        matriz = confusion_matrix(d_test, prediccion)
        print(confusion_matrix(d_test, prediccion))
        print('\n')

        print(classification_report(d_test, prediccion))

        fig = scatterplot_matrix(matriz, ['<=50k', '>50k'],
                    linestyle='none', marker='o', color='black', mfc='none')
        pp.show()
```

```
<=50k    116
>50k      34
Name: salida, dtype: int64
Matriz de Confusion

[[102  14]
 [ 10  24]]


              precision    recall  f1-score   support

           0       0.91      0.88      0.89       116
           1       0.63      0.71      0.67        34

   micro avg       0.84      0.84      0.84       150
   macro avg       0.77      0.79      0.78       150
weighted avg       0.85      0.84      0.84       150
```



1.2 Referencias

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[2] Portilla, J. (2017). A Beginner's Guide to Neural Networks in Python and SciKit Learn 0.18. Retrieved from https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/.

[3] The Asimov Institute. (2018). THE NEURAL NETWORK ZOO. Retrived from: http://www.asimovinstitute.org/neural-network-zoo/

In [ ]: