5

May 17, 2023

1 Ejercicios

1.1 Librerías

Importaremos todas las librerías que se usaran durante toda la tarea.

[ ]:
```
import numpy as np
import pandas as pd
import math

import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier #for classification problems
from sklearn.neural_network import MLPRegressor  #for regression problems
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

import warnings

warnings.filterwarnings("ignore")
```

1.2 LDA vs LR

1.2.1 Problema

Considera las siguientes muestras de dos distribuciones normales con misma matriz de covarianza pero promedio diferente.

[ ]:
```
cov = [[1, 0], [0, 1]]
mus = [[-2, -2], [2, 2]]
N = 30
atypical_data = np.array([[10,10]])
```

```
samples = [np.random.multivariate_normal(mu, cov, N) for mu in mus]

fig = go.Figure()
fig.add_scatter(
    x = np.concatenate((samples[0][:,0],atypical_data[:,0])),
    y = np.concatenate((samples[0][:,1],atypical_data[:,1])),
    name = 0,
    mode="markers"
)
fig.add_scatter(
    x = np.concatenate((samples[1][:,0],atypical_data[:,0])),
    y = np.concatenate((samples[1][:,1],atypical_data[:,1])),
    name = 1,
    mode="markers"
)
fig.update_layout(
    template="simple_white",
    title="LR",
    width=1000,
    height=1000
)
fig.show()
```

Añadimos una observación atípica (punto rojo) de la clase verde. ¿Cúal método es más robusto a este dato atípico: LDA o regresión logística? ¿Por qué?

1.2.2 Solución

[ ]:
```
lda = LinearDiscriminantAnalysis()
proj = lda.fit_transform(np.concatenate((samples[0],samples[1],atypical_data)),␣
 ↪[0] * N + [1] * N + [0])

fig = ff.create_distplot(
    [
        np.concatenate((proj[:N], [proj[-1]]))[:, 0],
        proj[N:2*N][:, 0]
    ],
    ["0", "1"],
    bin_size = 0.1,
    curve_type='normal',
)
fig.update_layout(
    template="simple_white",
    title="LDA",
    width=1000,
    height=500
)
fig.update_yaxes(zeroline=True)
```

```
fig.show()

lr = LogisticRegression().fit(np.
  ↪concatenate((samples[0],samples[1],atypical_data)), [0] * N + [1] * N + [0])
x = np.array([np.linspace(np.min(samples),10)])
y = np.array([np.linspace(np.min(samples),10)])
xx, yy = np.meshgrid(x,y)
Xfull = np.c_[xx.ravel(), yy.ravel()]
fig = go.Figure()
fig.add_surface(
    x=x.squeeze(),
    y=y.squeeze(),
    z=lr.predict_proba(Xfull)[:,1].reshape(50,50)
)
fig.add_scatter3d(
    x=np.concatenate((samples[0][:,0],atypical_data[:,0])),
    y=np.concatenate((samples[0][:,1],atypical_data[:,1])),
    z=np.zeros(N+1),
    mode="markers"
)
fig.add_scatter3d(
    x=samples[1][:,0],
    y=samples[1][:,1],
    z=np.ones(N),
    mode="markers"
)
fig.update_layout(
    template="simple_white",
    title="LDA",
    width=1000,
    height=1000
)
fig.show()
```

Recordemos que el LDA busca encontra la dirección con mayor "separación" de las proyecciones de los datos de diferentes clases. Esto la hace asumiendo distribuciones normales con misma covarianza y diferente media. Si un dato se encuentra muy "lejos" del resto hará que estás distribuciones al ser estimadas se trasladen y se aplanen por lo que habrá una gran interferencia una con la otra, ocasionando que los datos muy cercanos entre ellos de diferentes clases sean indistingibles.

Por otra parte el LR busca encontrar una función sigmoide que mejor reparametrice el parametro de una bernoulli a traves de una función lineal, podemos notar como está al estar en terminos de una función logarítmica esta será menos propensa a caer en temas de distancia ya que su crecimiento es cada vez menor mientras mas es la distancia. Aunque este dato atípico siempre será mal clasificado, en caso de que no todo la clase contraria será mal clasificada.

1.3 LR

1.3.1 Problema

En este ejercicio trabajamos con los Horseshoe crab data. Una descripción de los datos tomada del libro de Agresti es:

Each female horseshoe crab had a male crab resident in her nest. The study investigated factors affecting whether the female crab had any other males, called satellites, residing nearby. Explanatory variables are the female crabs color, spine condition, weight, and carapace width.

1.3.2 Solución

[ ]:
```
data = pd.read_csv('crabs.txt', delimiter='\s+')
data
```

[ ]:

	color	spine	width	satell	weight	y
0	3	3	28.3	8	3050	1
1	4	3	22.5	0	1550	0
2	2	1	26.0	9	2300	1
3	4	3	24.8	0	2100	0
4	4	3	26.0	4	2600	1
..	…	…	…	…	…	..
168	4	3	26.1	3	2750	1
169	4	3	29.0	4	3275	1
170	2	1	28.0	0	2625	0
171	5	3	27.0	0	2625	0
172	3	2	24.5	0	2000	0

[173 rows x 6 columns]

[ ]:
```
data_widths = data[["width","y"]].groupby(by=(lambda x : max(0, min(math.
 ↪ceil(data["width"].loc[x] - 23.5), 7)))).mean()
X = data['width'].values
X = np.reshape(X, (-1, 1))
lr = LogisticRegression().fit(X, data['y'].values)

fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x = data_widths["width"],
        y = data_widths["y"],
        mode="markers",
        name="Mean crabs with satellites"
    )
)

x = np.reshape(np.linspace(min(X[:, 0]), max(X[:, 0])), (-1, 1))
fig.add_trace(
```

```
    go.Scatter(
        x=x[:, 0],
        y=lr.predict_proba(x)[:, 1],
        mode="lines",
        name="LR"
    )
)
fig.update_layout(
    template="simple_white",
    title="LR by width",
    width=1000,
    height=1000,
    xaxis_title="Width",
    yaxis_title="Probability"
)
fig.show()
```

[ ]:
```
lr.score(X, data['y'].values)
```

[ ]:
```
0.7052023121387283
```

En la gráfica anterior realizamos el modelo descrito en el libro tomando en cuenta intervalos del ancho de los cangrejos así como el promedio de su etiqueta. Donde podemos observar que el estimar una sigmoide parece ser una buena idea.

[ ]:
```
X = data[['color', 'spine', 'width', 'weight']].values
predictors_score = pd.DataFrame(columns=['predictores','score'])
for i in range(1,16):
    index = np.array(np.where(np.array(list(bin(i))[2:])=='1')).flatten() + 4 -␣
 ↪len(list(bin(i))[2:])
    XX = X[:,index]
    XX = np.reshape(XX, (-1, bin(i).count("1")))
    lr = LogisticRegression().fit(XX, data['y'].values)
    predictors = ''
    for j in index:
        predictors += '+' + ['color', 'spine', 'width', 'weight'][j]
    predictors_score.loc[len(predictors_score)] = [predictors[1:], lr.score(XX,␣
 ↪data['y'].values)]
predictors_score
```

[ ]:

	predictores	score
0	weight	0.682081
1	width	0.705202
2	width+weight	0.647399
3	spine	0.641618
4	spine+weight	0.687861
5	spine+width	0.699422
6	spine+width+weight	0.653179

```
7                      color  0.687861
8                color+weight  0.722543
9                 color+width  0.722543
10         color+width+weight  0.710983
11                color+spine  0.670520
12         color+spine+weight  0.716763
13          color+spine+width  0.734104
14  color+spine+width+weight  0.687861
```

Ahora después de hacer regresión logística con todas las posibles combinaciones de predictores. Podemos observar primero que en general todos están en una probabilidad buena (arriba de 0.5), el color y el width parecen ser caracteristicas relacionadas con los satelites, y podemos ver como el poder predictivo de todos juntos empeora esto puede ser por considerar carácteristicas que son poco relacionadas con lo que se intenta predecir.

1.4 E-mail SPAM

1.4.1 Problema

Considera los datos spam de: http://search.r-project.org/library/kernlab/html/spam.html. Busca y discute modelos RL y compáralos con el clasificador LDA.

1.4.2 Solución

[ ]:
```
data = pd.read_csv("spambase.data")
data
```

[ ]:
```
      word_freq_make  word_freq_address  word_freq_all  word_freq_3d
0               0.00               0.64           0.64           0.0  \
1               0.21               0.28           0.50           0.0
2               0.06               0.00           0.71           0.0
3               0.00               0.00           0.00           0.0
4               0.00               0.00           0.00           0.0
…                …                  …              …              …
4596            0.31               0.00           0.62           0.0
4597            0.00               0.00           0.00           0.0
4598            0.30               0.00           0.30           0.0
4599            0.96               0.00           0.00           0.0
4600            0.00               0.00           0.65           0.0

      word_freq_our  word_freq_over  word_freq_remove  word_freq_internet
0              0.32            0.00              0.00                0.00  \
1              0.14            0.28              0.21                0.07
2              1.23            0.19              0.19                0.12
3              0.63            0.00              0.31                0.63
4              0.63            0.00              0.31                0.63
…               …               …                 …                   …
4596           0.00            0.31              0.00                0.00
4597           0.00            0.00              0.00                0.00
```

```
4598           0.00            0.00              0.00                0.00
4599           0.32            0.00              0.00                0.00
4600           0.00            0.00              0.00                0.00

      word_freq_order  word_freq_mail  …  char_freq_;  char_freq_(
0                0.00            0.00  …        0.000        0.000  \
1                0.00            0.94  …        0.000        0.132
2                0.64            0.25  …        0.010        0.143
3                0.31            0.63  …        0.000        0.137
4                0.31            0.63  …        0.000        0.135
…                   …               …  …            …            …
4596             0.00            0.00  …        0.000        0.232
4597             0.00            0.00  …        0.000        0.000
4598             0.00            0.00  …        0.102        0.718
4599             0.00            0.00  …        0.000        0.057
4600             0.00            0.00  …        0.000        0.000

      char_freq_[  char_freq_!  char_freq_$  char_freq_#
0             0.0        0.778        0.000        0.000  \
1             0.0        0.372        0.180        0.048
2             0.0        0.276        0.184        0.010
3             0.0        0.137        0.000        0.000
4             0.0        0.135        0.000        0.000
…               …            …            …            …
4596          0.0        0.000        0.000        0.000
4597          0.0        0.353        0.000        0.000
4598          0.0        0.000        0.000        0.000
4599          0.0        0.000        0.000        0.000
4600          0.0        0.125        0.000        0.000

      capital_run_length_average  capital_run_length_longest
0                          3.756                          61  \
1                          5.114                         101
2                          9.821                         485
3                          3.537                          40
4                          3.537                          40
…                              …                           …
4596                       1.142                           3
4597                       1.555                           4
4598                       1.404                           6
4599                       1.147                           5
4600                       1.250                           5

      capital_run_length_total  spam
0                          278     1
1                         1028     1
2                         2259     1
```

```
3                          191     1
4                          191     1
…                          …   …
4596                        88     0
4597                        14     0
4598                       118     0
4599                        78     0
4600                        40     0

[4601 rows x 58 columns]
```

[ ]:
```
clasf_score = pd.DataFrame(columns=["Method","Score"])
fig = make_subplots(rows=1,cols=5,subplot_titles=("LR-All", "LR-Freq_Name",␣
 ↪"LR-Freq_Symbol", "Freq_Cap_letters", "LDA-All"))

data_norm = preprocessing.StandardScaler().fit_transform(data.iloc[:,:-1].
 ↪values)
lr = LogisticRegression().fit(data_norm, data["spam"].values)
clasf_score.loc[len(clasf_score)] = ["LR-All", lr.score(data_norm, data["spam"].
 ↪values)]
conf_mat = confusion_matrix(data["spam"].values, lr.predict(data_norm))
fig.add_trace(
    go.Heatmap(
        z = conf_mat,
        x = ['No Spam', 'Spam'],
        y = ['No Spam', 'Spam'],
        coloraxis="coloraxis"
    ),
    row=1,
    col=1
)

data_norm = preprocessing.StandardScaler().fit_transform(data.iloc[:,:47].
 ↪values)
lr = LogisticRegression().fit(data_norm, data["spam"].values)
clasf_score.loc[len(clasf_score)] = ["LR-Freq_Name", lr.score(data_norm,␣
 ↪data["spam"].values)]
conf_mat = confusion_matrix(data["spam"].values, lr.predict(data_norm))
fig.add_trace(
    go.Heatmap(
        z = conf_mat,
        x = ['No Spam', 'Spam'],
        y = ['No Spam', 'Spam'],
        coloraxis="coloraxis"
    ),
    row=1,
    col=2
```

```
)

data_norm = preprocessing.StandardScaler().fit_transform(data.iloc[:,48:53].
 ↪values)
lr = LogisticRegression().fit(data_norm, data["spam"].values)
clasf_score.loc[len(clasf_score)] = ["LR-Freq_Symbol", lr.score(data_norm,␣
 ↪data["spam"].values)]
conf_mat = confusion_matrix(data["spam"].values, lr.predict(data_norm))
fig.add_trace(
    go.Heatmap(
        z = conf_mat,
        x = ['No Spam', 'Spam'],
        y = ['No Spam', 'Spam'],
        coloraxis="coloraxis"
    ),
    row=1,
    col=3
)

data_norm = preprocessing.StandardScaler().fit_transform(data.iloc[:,53:56].
 ↪values)
lr = LogisticRegression().fit(data_norm, data["spam"].values)
clasf_score.loc[len(clasf_score)] = ["Freq_Cap_letters", lr.score(data_norm,␣
 ↪data["spam"].values)]
conf_mat = confusion_matrix(data["spam"].values, lr.predict(data_norm))
fig.add_trace(
    go.Heatmap(
        z = conf_mat,
        x = ['No Spam', 'Spam'],
        y = ['No Spam', 'Spam'],
        coloraxis="coloraxis"
    ),
    row=1,
    col=4
)

data_norm = preprocessing.StandardScaler().fit_transform(data.iloc[:,:-1].
 ↪values)
lr = LinearDiscriminantAnalysis().fit(data_norm, data["spam"].values)
clasf_score.loc[len(clasf_score)] = ["LDA-All", lr.score(data_norm,␣
 ↪data["spam"].values)]
conf_mat = confusion_matrix(data["spam"].values, lr.predict(data_norm))
fig.add_trace(
    go.Heatmap(
        z = conf_mat,
        x = ['No Spam', 'Spam'],
```

```
        y = ['No Spam', 'Spam'],
        coloraxis="coloraxis"
    ),
    row=1,
    col=5
)

fig.update_layout(
    template="simple_white",
    title="Confusion matrix",
    width=1800,
    height=450,
    xaxis_title="Predict",
    yaxis_title="True"
)
fig.show()

clasf_score
```

[ ]:

	Method	Score
0	LR-All	0.930450
1	LR-Freq_Name	0.915671
2	LR-Freq_Symbol	0.816344
3	Freq_Cap_letters	0.736579
4	LDA-All	0.888720

Podemos observar como el poder predictivo de la regresión logística con todas las características es el mejor. El clasificar correctamente un correo como no spam parece ser una tarea dificil, puesto que todos hacen un trabajo bastante bueno; en cambio el clasificar correctamente un correo como spam es más dificl pero a pesar de ello la regresión logística es el que mejor despeño tuvo. Son pocas las veces en que la regresión logística clasifica incorrectamente un correo. Por último podemos notar como para la regresión logística tienen desmpeños muy similares el tomar en cuenta todas las características y solo las frecuencias en el nombre del correo, con lo que indica que hay una mayor relación con esta caráteristica y la clasificación de correos spam que el resto.

En los scores podemos notar en efecto como el la regresión logística tanto considerando todos las caraterísticas como solo la frecuencia del nombre, son las mejores. Aunque el LDA no hace un mal desempeño.

1.5 Feedataorward Neural networks with sklearn

We generate data from the model Y=f(X) + ϵ with f(X) a polynomial of degree 7

[ ]:
```
x = np.random.uniform(0, 2, size=300)
x.sort()
y_true=0.2*x**5 - x**3 + 0.03*x**7
y = y_true + 0.5 * np.random.normal(size=300)
```

We adjust a NN with one hidden layer; alpha refers to the constant of the L2 regularization term

(in class we called it lambda).

Different solvers are available; the solver lbfgs is recommended for small data sets (see help)

[ ]:
```
mlp = MLPRegressor(hidden_layer_sizes=(10),solver="lbfgs",max_iter=200 ,alpha=0)
mlp.fit(x.reshape(-1, 1),y)
pn=mlp.predict(x.reshape(-1, 1))
fig = go.Figure()
fig.add_scatter(
    x = x,
    y = y,
    name = "train_data",
    mode="markers"
)
fig.add_scatter(
    x = x,
    y = y_true,
    name = "Verdadero valor",
)
fig.add_scatter(
    x = x,
    y = pn,
    name = "LR",
)
fig.update_layout(
    template="simple_white",
    title="LR",
    width=1000,
    height=1000,
    sliders=[{"transition":dict(duration=0.01),'currentvalue':{"prefix":␣
 ↪r"\sigma = "},}]
)
fig.show()
```

2 Excercices

Experiment 1: show how different runs of the algoritm lead to different solutions

[ ]:
```
fig = go.Figure()
fig.add_scatter(
    x = x,
    y = y,
    name = "train_data",
    mode="markers"
)
fig.add_scatter(
    x = x,
    y = y_true,
```

```
    name = "Verdadero valor",
)
for i in range(5):
    mlp.fit(x.reshape(-1, 1),y)
    pn=mlp.predict(x.reshape(-1, 1))
    fig.add_scatter(
        x = x,
        y = pn,
        name = "LR" + str(i+1),
    )
fig.update_layout(
    template="simple_white",
    title="LR",
    width=1000,
    height=1000,
    sliders=[{"transition":dict(duration=0.01),'currentvalue':{"prefix":␣
 ↪r"\sigma = "},}]
)
fig.show()
```

Se ejecuto 5 veces el mismo código, y en cada una podemos notar como el resultado fue distinto.

Experiment 2: show how the number of neurons in the hidden layer affects the training error

[ ]:
```
sizes = 2**np.arange(11, dtype=np.int64)
errors = []

fig = go.Figure()
for size in sizes:
    tests = []
    for t in range(10):
        mlp =␣
 ↪MLPRegressor(hidden_layer_sizes=(size),solver="lbfgs",max_iter=500 ,alpha=0)
        mlp.fit(x.reshape(-1, 1),y)
        tests.append(1-mlp.score(x.reshape(-1, 1), y_true))
    fig.add_box(
        name = str(size),
        y = tests,
    )
fig.update_layout(
    template="simple_white",
    title="LR con diferente cantidad de neuronas",
    width=1500,
    height=500,
    sliders=[{"transition":dict(duration=0.01),'currentvalue':{"prefix":␣
 ↪r"\sigma = "},}]
)
fig.show()
```

Podemos notar como al agregar más neuronas la medio va decreciendo así como la varianza de los scores, con ello un mejor resultado en el conjunto de entrenamiento.

Experiment 3: show how the number of neurons in the hidden layer affects the test error (using a new set of data)

[ ]:
```
x_1 = np.random.uniform(0, 2, size=300)
x_1.sort()
y_true=0.2*x**5 - x**3 + 0.03*x**7
errors = []

fig = go.Figure()
for size in sizes:
    tests = []
    for t in range(10):
        mlp =␣
 ↪MLPRegressor(hidden_layer_sizes=(size),solver="lbfgs",max_iter=500 ,alpha=0)
        mlp.fit(x.reshape(-1, 1),y)
        tests.append(1-mlp.score(x_1.reshape(-1, 1), y_true))
    fig.add_box(
        name = str(size),
        y = tests,
    )
fig.update_layout(
    template="simple_white",
    title="LR con diferente cantidad de neuronas",
    width=1500,
    height=500,
    sliders=[{"transition":dict(duration=0.01),'currentvalue':{"prefix":␣
 ↪r"\sigma = "},}]
)
fig.show()
```

Podemos notar como al agregar más neuronas la medio va decreciendo así como la varianza de los scores, con ello un mejor resultado en el conjunto de prueba.