In [1]:

```
%matplotlib inline
```

Tema: Redes Neuronales Feed-forward

Al igual que para la notebook de la clase 5 de SVM, utilizaremos el dataset "empty.all.csv" que contiene como clase positiva, 900 artículos de Wikipedia en inglés que presentan la falla "Empty Section" y como clase negativa, contiene 900 artículos destacados. El mismo se encuentra en el subdirectorio "miscelaneos" del repositorio Github. Los datos se cargan como un DataFrame mediante un método de la biblioteca seaborn. A tal fin es necesario copiar el dataset en el home local de seaborn. Por defecto usa ~/seaborn-data/, en Windows: "C:\Users\Nbre_Usuario\seaborn-data".

Ejemplos

In [2]:

```
import seaborn as sns
empty = sns.load_dataset('empty.all',cache=True)
empty.head()
```

Out[2]:

	wordSyllables	wordLength	wordCount	weaselWordRate	triviaSectionsCount	to_be_verbRate
0	1.000000	3.021739	46	0.000000	0	0.000000
1	2.000000	7.500000	4	0.000000	0	0.000000
2	1.642857	5.714286	14	0.000000	0	0.000000
3	1.642857	5.714286	14	0.000000	0	0.000000
4	1.902256	5.393484	399	0.250627	0	3.759398

5 rows × 96 columns

In [3]:

```
list(empty.columns.values)
```

Out[3]:

```
['wordSyllables',
 'wordLength',
 'wordCount',
 'weaselWordRate',
 'triviaSectionsCount',
 'to_be_verbRate',
 'templateCount',
 'tableCount',
 'syllableCount',
 'subsubsectionLength',
 'subsubsectionCount',
 'subsectionNesting',
 'subsectionLength',
 'subsectionCount',
 'stopwordRate',
 'smogIndex',
 'shortestSubsubsectionLength',
 'shortestSubsectionLength',
 'shortestSentenceLength',
 'shortestSectionLength',
 'shortSentenceRate',
 'sentenceLength',
 'sentenceCount',
 'sentenceBeginSubordinatingConjunctionRate',
 'sentenceBeginPronounRate',
 'sentenceBeginPrepositionRate',
 'sentenceBeginInterrogativePronounRate',
 'sentenceBeginCoordinatingConjunctionRate',
 'sentenceBeginArticleRate',
 'sectionNesting',
 'sectionLength',
 'sectionCount',
 'registeredEditorRate',
 'referenceWordRate',
 'referenceSectionsCount',
 'referenceSectionRate',
 'referenceCount',
 'reciprocity',
 'questionRate',
 'questionCount',
 'pronounRate',
 'prepositionRate',
 'peacockWordRate',
 'passiveSentenceRate',
 'paragraphLength',
 'paragraphCount',
 'pageRank',
 'oneSyllableWordRate',
 'oneSyllableWordCount',
 'nominalizationRate',
 'miyazaki',
 'longestSubsubsectionLength',
 'longestSubsectionLength',
 'longestSentenceLength',
 'longestSectionLength',
```

```
 'longWordRate',
 'longSentenceRate',
 'lix',
 'listRate',
 'linkRate',
 'languageLinkCount',
 'internalLinkCount',
 'informationToNoiseRatio',
 'inLinkCount',
 'imagesPerSection',
 'imageCount',
 'headingCount',
 'gunningFogIndex',
 'forcastGradeLevel',
 'fleschReadingEase',
 'fleschKincaidGradeLevel',
 'fileCount',
 'externalLinksPerSection',
 'externalLinkCount',
 'editsPerEditor',
 'editorRate',
 'editorCount',
 'editCount',
 'easyWordRate',
 'discussionEditCount',
 'difficultWordRate',
 'daleChall',
 'currency',
 'conjunctionRate',
 'complexWordRate',
 'colemanLiauIndex',
 'characterCount',
 'categoryCount',
 'brokenLinkCount',
 'bormuth',
 'auxiliaryVerbRate',
 'ari',
 'anonymousEditorRate',
 'agePerEdit',
 'age',
 'has_flaw']
```

In [4]:

```
empty.shape
```

Out[4]:

```
(1800, 96)
```

In [5]:

```
X_empty = empty.drop('has_flaw', axis=1)
X_empty.shape
```

Out[5]:

```
(1800, 95)
```

In [6]:

```
y_empty = empty['has_flaw']
y_empty.shape
```

Out[6]:

```
(1800,)
```

In [7]:

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

In [8]:

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

```
X_train shape: (1350, 95)
y_train shape: (1350,)
```

In [9]:

```
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_test shape: (450, 95)
y_test shape: (450,)
```

La celda a continuación tiene como objetivo estandarizar los valores de las características para que tengan media 0 y varianza 1. Hacemos esto pues de forma similar a cómo sucedía con SVM, las NNs son sensitivas al escalado de características. Para corroborar esto se sugiere primeramente no ejecutar la celda de estandarización y ver la performance que tiene el clasificador. Luego, ejecutar la misma y las que siguen a continuación para poder comparar la diferencia existente en la calidad predictiva del clasificador.

In [10]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# fit only on training data
scaler.fit(X_train)
X_train = scaler.transform(X_train)
# apply same transformation to test data
X_test = scaler.transform(X_test)
```

Descripción de los parámetros de la NN

activation es la función de activación: {'identity', 'logistic', 'tanh', 'relu'}, por defecto 'relu'.

alpha es el parámetro de penalización del término de regularización introducido en la función a optimizar para evitar el sobre-ajuste. En la transparencia 44 de la clase 7, es referido como *lambda*.

hidden_layer_sizes es el número de capas ocultas y de neuronas en cada capa. Por defecto, 1 capa oculta de 100 neuronas *(100,)*. Si quisiéramos poner tres capas ocultas de 10 neuronas cada una, deberíamos especificar *(10,10,10)*.

solver es el método utilizado para realizar la optimización de pesos: {‘lbfgs’, ‘sgd’, ‘adam’}, por defecto ‘adam’.

- *lbfgs*: optimizador de la familia de los métodos "quasi-Newton".
- *sgd*: descenso del gradiente estocástico.
- *adam*: descenso del gradiente estocástico propuesto por Kingma, Diederik, and Jimmy Ba.

learning_rate constante positiva que nos permite moderar cuán pronunciada es la actualización de los pesos en cada paso: {‘constant’, ‘invscaling’, ‘adaptive’}, por defecto ‘constant’.

- *constant* es un valor constante dado por el parámetro ‘learning_rate_init’.
- *invscaling* decrementa gradualmente el learning rate en cada paso de tiempo ‘t’: effective_learning_rate = learning_rate_init / pow(t, power_t).
- *adaptive* mantiene el learning rate constante en ‘learning_rate_init’ mientras la función de pérdida en el entrenamiento siga disminuyendo.

momentum valor constante para actualizar el descenso del gradiente, por defecto ‘0.9’. Sólo usado cuando solver=‘sgd’.

In [11]:

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(activation='logistic', solver='lbfgs', alpha=0.0001, hidden_layer_sizes=

nn.fit(X_train, y_train)
```

Out[11]:

```
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=(5,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

In [12]:

```
y_model = nn.predict(X_test)
```

In [13]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_model)
```

Out[13]:

```
0.98444444444444446
```

In [14]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_model))
```

```
             precision    recall  f1-score   support

         no       0.99      0.98      0.98       220
        yes       0.98      0.99      0.98       230

avg / total       0.98      0.98      0.98       450
```

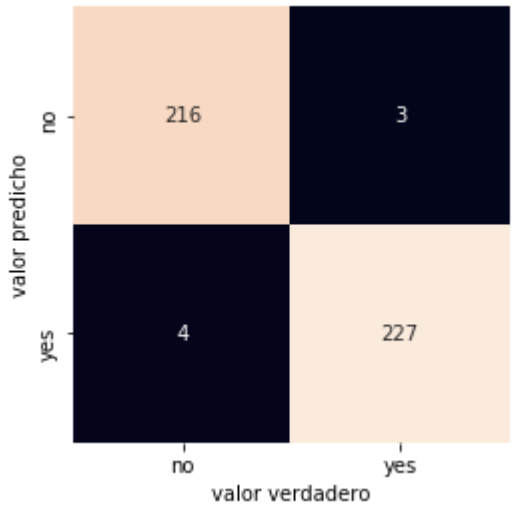
In [15]:

```
target_names = ['no', 'yes']

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(y_test, y_model)

sns.heatmap(mat.T, square=True, annot=True, cbar=False, fmt="d", xticklabels=target_names,
plt.xlabel('valor verdadero')
plt.ylabel('valor predicho');
```



In [16]:

```
nngs = MLPClassifier(activation='logistic', solver='sgd', alpha=0.0001, hidden_layer_sizes=

from sklearn.model_selection import GridSearchCV
parameters = {'momentum': [0.7,0.8,0.9], 'learning_rate_init': [0.001,0.01,0.1]}

scores = ['precision', 'recall']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(nngs, parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    #print()
    #print("Grid scores on development set:")
    #print()
    #means = clf.cv_results_['mean_test_score']
    #stds = clf.cv_results_['std_test_score']
    #for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    #    print("%0.3f (+/-%0.03f) for %r"
    #          % (mean, std * 2, params))
    #print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
```

```
# Tuning hyper-parameters for precision
```