Notes on Logistic Regression and Preprocessing

I. Logistic Regression

What is Logistic Regression?

It is a **supervised** machine learning algorithm, that falls under the category of **classifiers**, and it's used to predict the probability of a given point belonging to a certain category in the dependent variable (target).

Logistic regression is commonly used in binary classification but it can be used in multiclass problems as well.

How does it work?

Logistic regression almost works as a simple linear regression algorithm in the sense that it also computes the coefficients of a linear equation. However, the output of the linear equation will be passed again to a non-linear function called **sigmoid** whose job is to convert that output into a range of values going from 0 to 1 (i.e., a probability).

Based on the threshold chosen (let's say 0.5), the output will be then either 0 if $y < 0.5$ or 1 if $y \geq 0.5$.

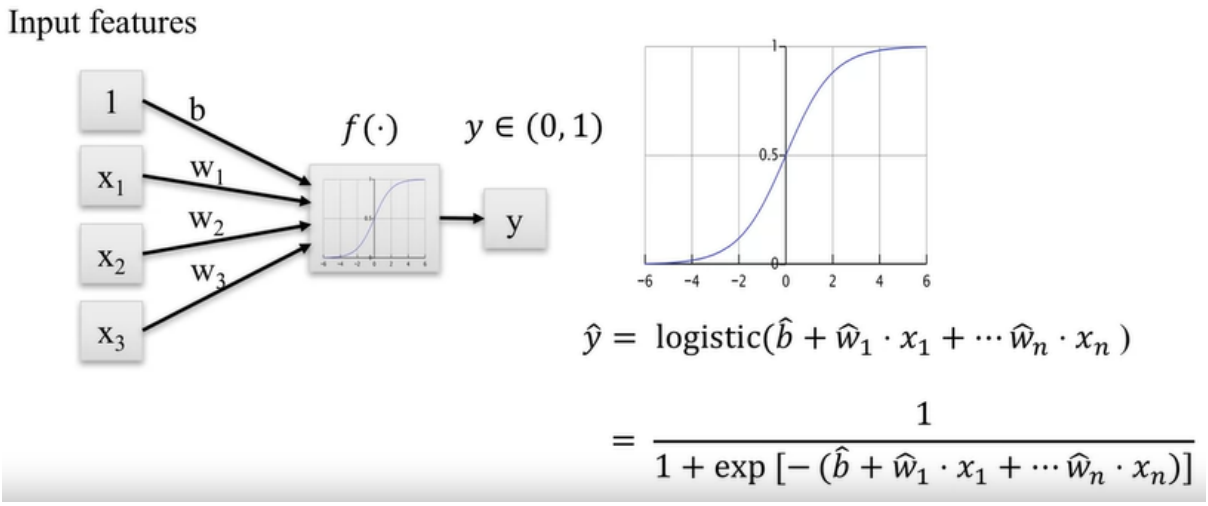

Image from http://luisevalencia.com/algorithms-logistic-regression

Pros and cons

Some advantages of logistic regression are:

- Ease of implementation and interpretation
- It can be extended to multiclass problems
- Performs quite good when the dataset is linearly separable
- Coefficients of the model can serve as indicators of feature importance

Some disadvantages are:

- Tends to overfit when the number of observations is lesser than the number of features
- It assumes a linearity relationship between the target and the features
- In the case of non-linearly separable datasets, logistic regression cannot be used.

Code in python

```
# Import libraries
import pandas as pd
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression


# Load data
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name='iris_type')


# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)


# Preprocessing and fitting
scaler = StandardScaler()
lgr = LogisticRegression()

pipe = make_pipeline(scaler, lgr)

pipe.fit(X_train, y_train)


# If you want to check the weights of the model
print(pipe['logisticregression'].coef_)

print(pipe['logisticregression'].intercept_)


# Accuracy
print(pipe.score(X_test, y_test))
```

II. Preprocessing

Why do we need preprocessing?

Preprocessing helps to prepare and transform the raw data into a form that is suitable for Machine learning to train on.

For example, standardization is a required step for ML algos that rely on distance measurements (e.g., KNN, Kmeans, PCA, etc...) as these are very sensitive to wide ranges of values.

In fact, let me quote this sentence from the scikit-learn webpage:

> **Standardization** of datasets is a **common requirement for many machine learning estimators** implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with **zero mean and unit variance**.

How?

It involves among others:

- Any form of **cleaning**: (removing NaNs, duplicates, outliers, leading/trailing spaces in strings, etc...)
- **Imputation**: Handling of missing values by replacing them for example with mean, median (for numerical features), or mode (for categorical features).
- **Standardization**: It enforces transformed values to have a mean of 0 and a standard deviation of 1.
- **Min-max scaling**: This one helps to rescale the data so that it ranges between 0 and 1.
- **Log scaling**: Transform a skewed distribution into a more or less normal distribution.
- **Encoding**: Maps categorical features to numbers (e.g., One hot encoding, ordinal encoding)

In scikit-learn

Here are shown only some `Classes` (mainly from the `sklearn.preprocessing` module) to preprocess your data.

But there are many more!!! Feel free to check the scikit-learn documentation 😉.

Class	Goal	Module
`StandarScaler`	Standardization	`sklearn.preprocessing`
`PowerTransform`	Non-linear transformation	`sklearn.preprocessing`
`Normalizer`	Normalization	`sklearn.preprocessing`

`OrdinalEncoder`	Encoding	`sklearn.preprocessing`
`KBinsDiscretizer`	Discretization	`sklearn.preprocessing`
`SimpleImputer`	Imputation	`sklearn.impute`
`PolynomialFeatures`	Higher degree polynomials	`sklearn.preprocessing`