Binary Logistic Regression in Python

Complete the iris dataset

Importing all the necessary libraries and adding the category species with species names.

In [3]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd
import numpy as np

# Importing Sklearn module and classes
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Convert 'iris.data' numpy array to 'iris.dataframe' pandas dataframe
# complete the iris dataset by adding species
iris = datasets.load_iris()
iris = pd.DataFrame(
    data= np.c_[iris['data'], iris['target']],
    columns= iris['feature_names'] + ['target']
    )

species = []

for i in range(len(iris['target'])):
    if iris['target'][i] == 0:
        species.append("setosa")
    elif iris['target'][i] == 1:
        species.append('versicolor')
    else:
        species.append('virginica')


iris['species'] = species
iris
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0	virginica
146	6.3	2.5	5.0	1.9	2.0	virginica
147	6.5	3.0	5.2	2.0	2.0	virginica
148	6.2	3.4	5.4	2.3	2.0	virginica
149	5.9	3.0	5.1	1.8	2.0	virginica

150 rows × 6 columns

The result is the complete dataframe. Next we need to split our data into a training and a testing set. We can subset the dataset first using .iloc() method then use the function train_test_split().

In [13]:

```
x = iris.iloc[:, 0:4]
y = iris.iloc[:, 4]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=
1, stratify=y)
x_train.head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
33	5.5	4.2	1.4	0.2
20	5.4	3.4	1.7	0.2
115	6.4	3.2	5.3	2.3
124	6.7	3.3	5.7	2.1
35	5.0	3.2	1.2	0.2

Feature scaling

In [14]:

```
sc = StandardScaler()
sc.fit(x_train)
x_train_std = sc.transform(x_train)
x_test_std = sc.transform(x_test)
```

Train a Logistic Regression Model

Next step is to train a logistic regression model. The following needs to be noted while using LogisticRegression algorithm sklearn.linear_model implementation:

1. Usage of C parameters. Smaller values of C specify stronger regularization.
2. The multi_class parameter is assigned to ‘ovr‘. It represents one-vs-rest algorithm to be used. Other option is multinomial.
3. The solver parameter is assigned to ‘lbfsg‘. Other solvers which can be used are newton-cg, sag, saga, lib linear

In [15]:

```
# Create an instance of LogisticRegression classifier
lm = LogisticRegression(C=100.0, random_state=1, solver='lbfgs', multi_class='ovr')

# Fit the model
#
lm.fit(x_train_std, y_train)
```

Out[15]:

```
LogisticRegression(C=100.0, multi_class='ovr', random_state=1)
```

Measure the model's performance

In [16]:

```
# Create the predictions
#
y_predict = lm.predict(x_test_std)

# Use metrics.accuracy_score to measure the score
print("LogisticRegression Accuracy %.3f" %metrics.accuracy_score(y_test, y_predict))
```

```
LogisticRegression Accuracy 0.978
```

In [ ]:

In [ ]:

In [ ]: