Introduction to Machine Learning in Engineering Science

National Cheng Kung University Department of Engineering Science

Instructor: Chi-Hua Yu

Lab 3

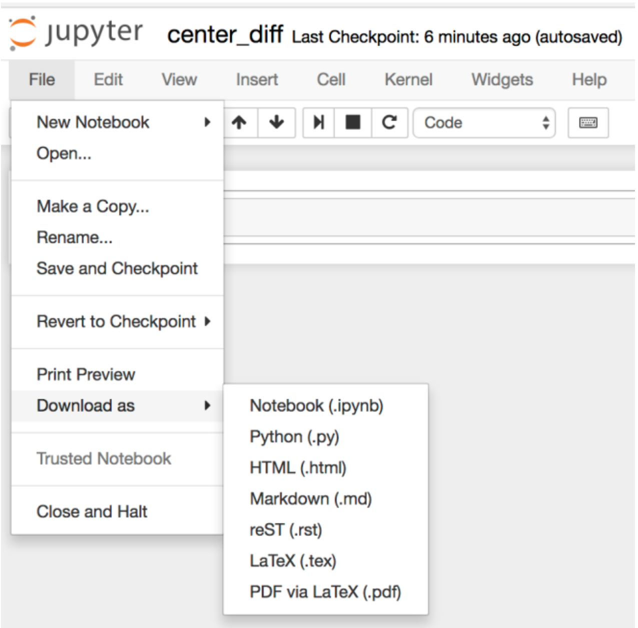
Programming, Due 12:00, Saturday, October 23rd, 2021

Late submission before post of solution: score*0.8 (the solution will usually be posted within a week); no late submission after the post of solution

---

Lab Submission Procedure (請仔細閱讀)

1. You should submit your Jupyter notebook and Python script (`*.py`, in Jupyter, click File, Download as, Python (`*.py`)).



2. Name a folder using your student id and lab number (e.g., n96081494_lab1), put all the python scripts into the folder and zip the folder (e.g., n96081494_ lab 1.zip).

3. Submit your lab directly through the course website.

Total 120%

1. **(120%)** Name your Jupyter notebook `Perceptron.ipynb` and Python script `preceptron.py`. As we mentioned in lecture last Friday, the learning rule can be expressed as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot \left(y - d^{(k)}\right) \cdot \boldsymbol{x}^{(k)}$$

where

$\boldsymbol{w} = [\theta \quad w_1 \quad w_2 \quad \cdots \quad w_n]$ is the vector containing the threshold and weights;

$\boldsymbol{x}^{(k)} = \left[-1 \quad x_1^{(k)} \quad x_2^{(k)} \quad \cdots \quad x_n^{(k)}\right]$ is the k^{th} training sample;

$d^{(k)}$ is the desired value for the k^{th} training sample;

η is a constant that defines the learning rate of the Perceptron.

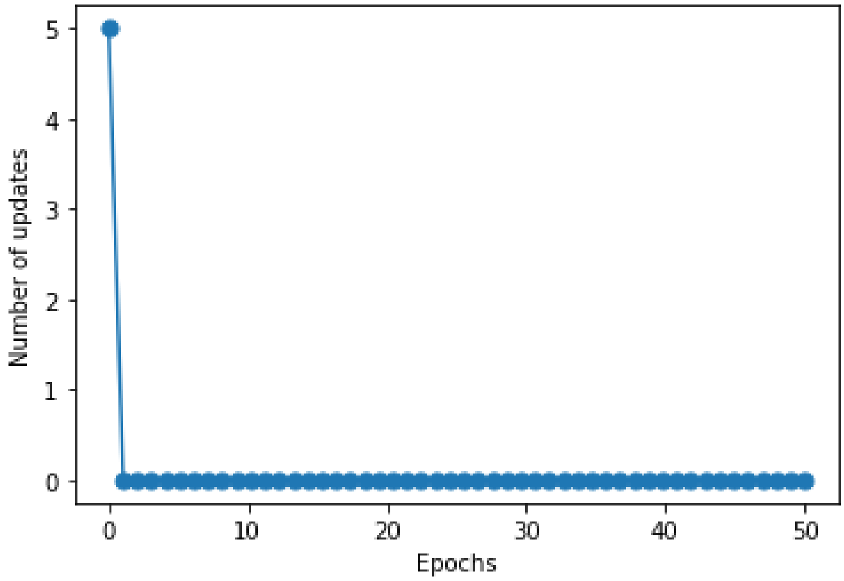
(a) (80%)Please finish the fit function we provide in Perception.py file and test the code using the following code fragment.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
#from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from perceptron import Perceptron

iris = load_iris()
X = iris.data[0:100, (0, 2)] # sepal length, petal length
y = iris.target[0:100] # Setosa or Versicolor
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')

per_clf = Perceptron(random_state=42)
#default learning rate = 1.0
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)
per_clf.fit(X_train,y_train)
y_pred = per_clf.predict(X_test)
print('Misclassified samples: %d' % (y_test != y_pred).sum())
```

```
Misclassified samples: 0
```

(b)(40%) Plot the training history and write a function `plot_decision_regions` to visualize the decision region. You can use the below code fragment

Below is the running example:




```
plot_decision_regions(X, y, classifier=per_clf)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')


# plt.savefig('images/02_08.png', dpi=300)
plt.show()
```

The filled contour of red and blue colors in the example figure is drawn by `plt.contourf`. The grids coordinates needed to draw filled contour can use `np.meshgrid`, and use the trained perceptron to predict the height values of the contour.