**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Môn học: Machine Learning**

**BÁO CÁO BÀI TẬP**

**Giáo viên hướng dẫn: Vũ Quang Huy**

**Sinh viên thực hiện: Nguyễn Thái Học**

**Mã số sinh viên: 15110052**

**Tp. Hồ Chí Minh, tháng 11 năm 2018**

**Mục Lục**

[**1.** **Linear Regression** 3](#_Toc532830656)

[**2.** **K-means Clustering** 7](#_Toc532830657)

[**3.** **Softmax Regression** 13](#_Toc532830658)

# **Linear Regression**

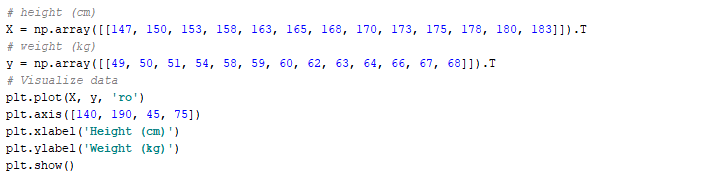
* Chúng ta có 1 bảng dữ liệu về chiều cao và cân nặng của 15 người như dưới đây:

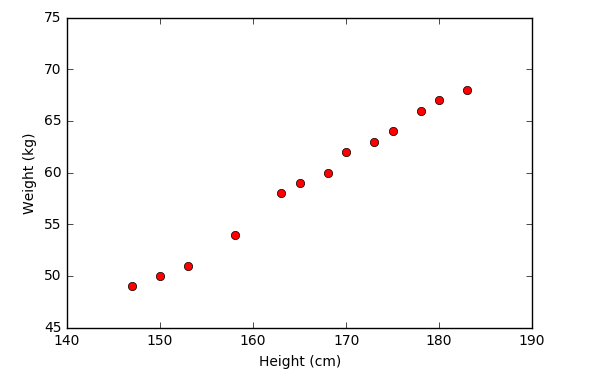
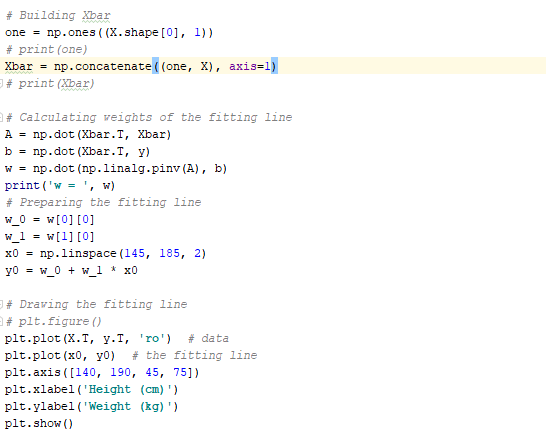
| **Chiều cao (cm)** | **Cân nặng (kg)** | **Chiều cao (cm)** | **Cân nặng (kg)** |
| --- | --- | --- | --- |
| 147 | 49 | 168 | 60 |
| 150 | 50 | 170 | 72 |
| 153 | 51 | 173 | 63 |
| 155 | 52 | 175 | 64 |
| 158 | 54 | 178 | 66 |
| 160 | 56 | 180 | 67 |
| 163 | 58 | 183 | 68 |
| 165 | 59 |  |  |

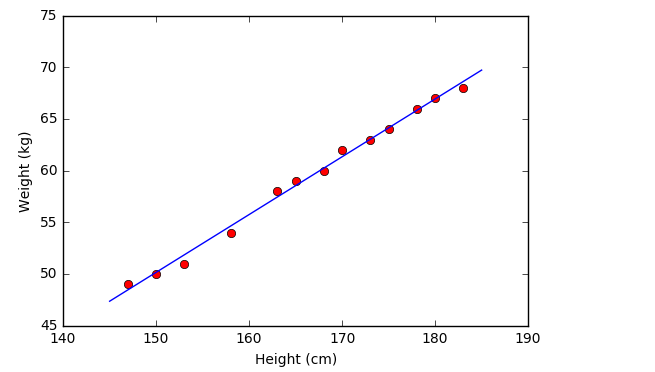
* Chúng ta có thể thấy là cân nặng sẽ tỉ lệ thuận với chiều cao (càng cao càng nặng), nên có thể sử dụng Linear Regression model cho việc dự đoán này. Để kiểm tra độ chính xác của model tìm được, chúng ta sẽ giữ lại cột 155 và 160 cm để kiểm thử, các cột còn lại được sử dụng để huấn luyện model.
* Trước tiên, chúng ta cần có hai thư viện numpy cho đại số tuyến tính và matplotlib cho việc vẽ hình.



* Tiếp theo, chúng ta khai báo và biểu diễn dữ liệu trên một đồ thị.



* Từ đồ thị này ta thấy rằng dữ liệu được sắp xếp gần như theo 1 đường thẳng.
* Tiếp theo, chúng ta sẽ tính toán các hệ số w\_1 và w\_0 dựa vào công thức w=A†b=(¯XT¯X)†¯XTy. Chú ý: giả nghịch đảo của một ma trận A trong Python sẽ được tính bằng numpy.linalg.pinv(A), pinv là từ viết tắt của pseudo inverse.
* Từ đồ thị bên trên ta thấy rằng các điểm dữ liệu màu đỏ nằm khá gần đường thẳng dự đoán màu xanh. Vậy mô hình Linear Regression hoạt động tốt với tập dữ liệu training. Bây giờ, chúng ta sử dụng mô hình này để dự đoán cân nặng của hai người có chiều cao 155 và 160 cm mà chúng ta đã không dùng khi tính toán nghiệm.

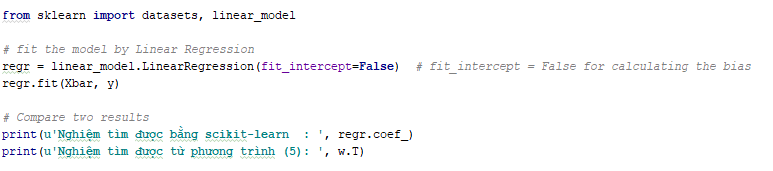


* Chúng ta thấy rằng kết quả dự đoán khá gần với số liệu thực tế.



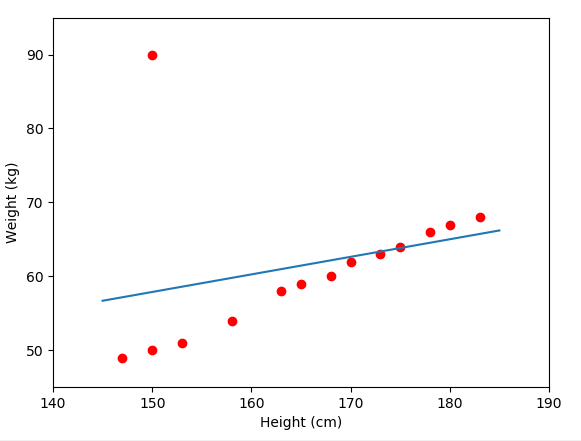


* Chúng ta sẽ sử dụng thư viện scikit – learn của Python để tìm nghiệm và có kết quả tương tự.





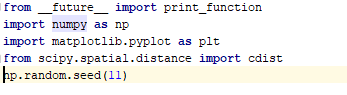
* Hạn chế đầu tiên của Linear Regression là nó rất **nhạy cảm với nhiễu**(sensitive to noise).
* Trong ví dụ về mối quan hệ giữa chiều cao và cân nặng bên trên, nếu có chỉ một cặp dữ liệu nhiễu (150 cm, 90kg) thì kết quả sẽ sai khác đi rất nhiều.



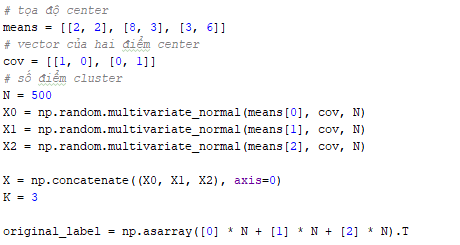
* Vì vậy, trước khi thực hiện Linear Regression, các nhiễu (outlier) cần phải được loại bỏ. Bước này được gọi là tiền xử lý (pre-processing).
* Hạn chế thứ hai của Linear Regression là nó **không biễu diễn được các mô hình phức tạp**. Mặc dù trong phần trên, chúng ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa outcomevà input không nhất thiết phải là tuyến tính, nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế.

# **K-means Clustering**

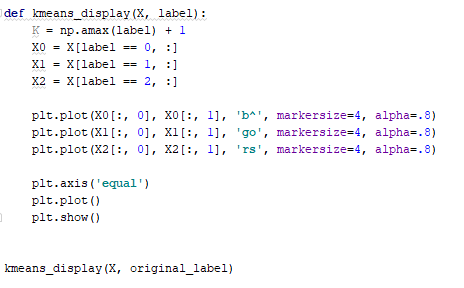
* Chúng ta chọn center cho từng cluster và tạo dữ liệu cho từng cluster bằng cách lấy mẫu theo phân phối chuẩn có kỳ vọng là center của cluster đó và ma trận hiệp phương sai (covariance matrix) là ma trận đơn vị.
* Trước tiên, chúng ta cần khai báo các thư viện cần dùng. Chúng ta cần numpy và matplotlib như trong bài Linear Regression cho việc tính toán ma trận và hiển thị dữ liệu. Chúng ta cũng cần thêm thư viện scipy.spatial.distance để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả.



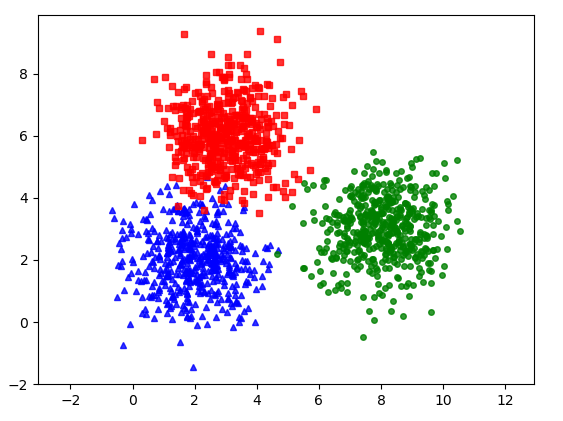
* Tiếp theo, ta tạo dữ liệu bằng cách lấy các điểm theo phân phối chuẩn có kỳ vọng tại các điểm có tọa độ (2, 2), (8, 3) và (3, 6), ma trận hiệp phương sai giống nhau và là ma trận đơn vị. Mỗi cluster có 500 điểm. (Chú ý rằng mỗi điểm dữ liệu là một hàng của ma trận dữ liệu.



* Hiển thị dữ liệu trên đồ thị: chúng ta cần một hàm kmeans\_display để hiển thị dữ liệu. Sau đó hiển thị dữ liệu theo nhãn ban đầu.



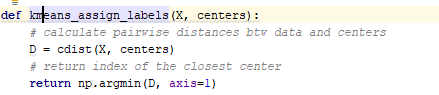
* Trong đồ thị trên, mỗi cluster tương ứng với một màu. Có thể nhận thấy rằng có một vài điểm màu đỏ bị lẫn sang phần cluster màu xanh.



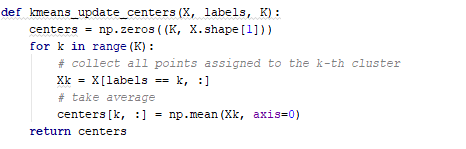
* Các hàm số cần thiết cho K – means clustering:
  + kmeans\_init\_centers để khởi tạo các centers ban đầu.



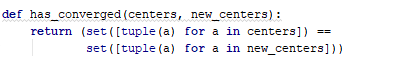
* + kmeans\_asign\_labels để gán nhán mới cho các điểm khi biết các centers.



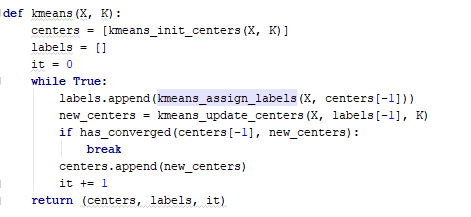
* + kmeans\_update\_centers để cập nhật các centers mới dữa trên dữ liệu vừa được gán nhãn.

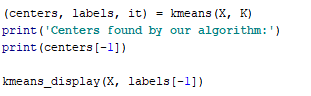


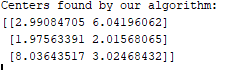
* + has\_converged để kiểm tra điều kiện dừng của thuật toán.

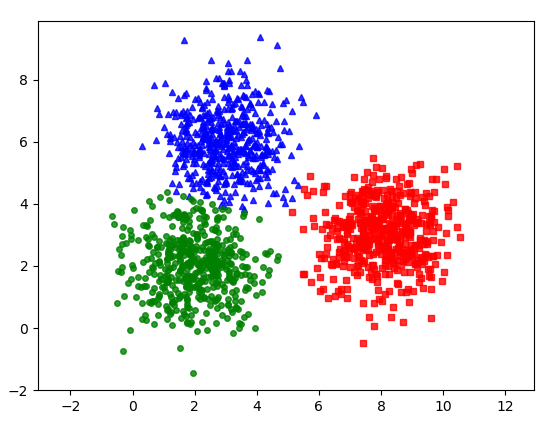


* Phần chính của K-means clustering: Áp dụng thuật toán vừa viết vào dữ liệu ban đầu, hiển thị kết quả cuối cùng.

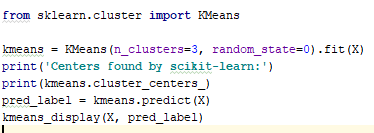


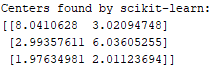


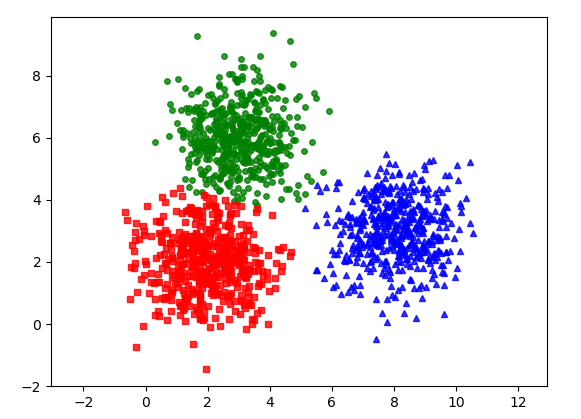




* Từ kết quả này chúng ta thấy rằng thuật toán K-means clustering làm việc khá thành công, các centers tìm được khá gần với kỳ vọng ban đầu. Các điểm thuộc cùng một cluster hầu như được phân vào cùng một cluster (trừ một số diểm màu đỏ ban đầu đã bị phân nhầm vào cluster màu xanh da trời, nhưng tỉ lệ là nhỏ và có thể chấp nhận được).
* Chúng ta thấy rằng thuật toán trên hội tụ rất nhanh, chỉ cần 6 vòng lặp để có được kết quả cuối cùng.
* Chúng ta cũng có thể tìm được kết quả tương tự đối với thư viện scikit – learn.



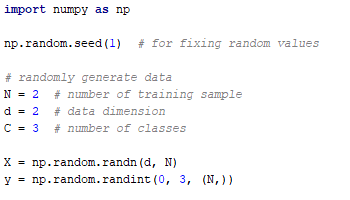




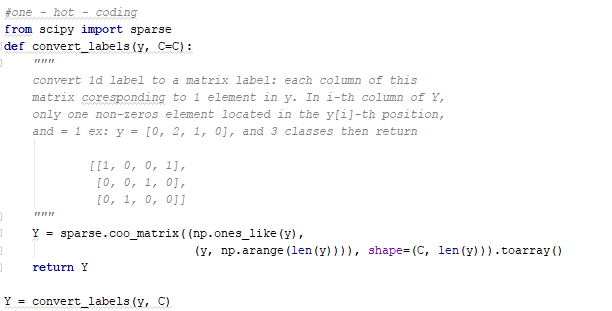
* Các hạn chế của bài toán K – mean là:
  + Chúng ta cần phải biết số lượng cluster cần clustering.
  + Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu.
  + Các cluster cần phải có số lượng điểm gần bằng nhau.
  + Các cluster cần có dạng hình tròn.
  + Khi 1 cluster nằm phía trong 1 cluster khác.

# **Softmax Regression**

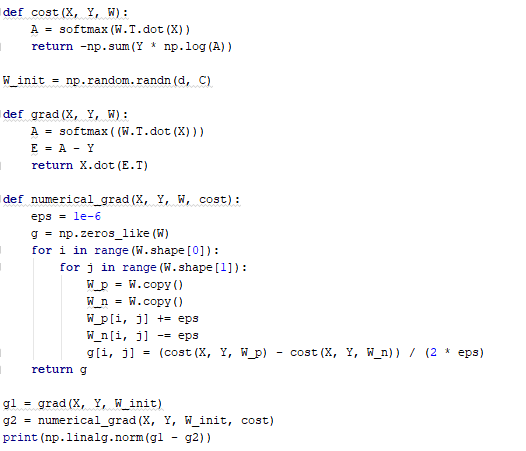
* Tạo simulated data.



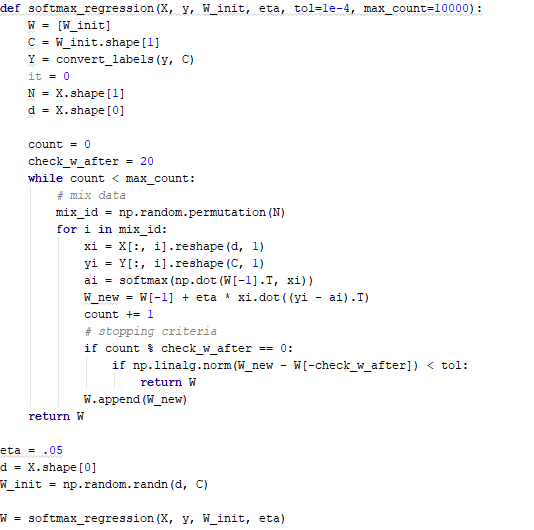
* Trong ví dụ đơn giản này, số điểm dữ liệu chỉ là N = 2, số chiều dữ liệu d = 2, và số classes C = 3. Những giá trị đủ nhỏ này giúp cho việc kiểm tra có thể được thực hiện một cách tức thì. Sau khi thuật toán chạy đúng với những giá trị nhỏ này, ta có thể thay N, d, C bằng vài giá trị khác trước khi sử dụng dữ liệu thật.
* Đối với ma trận one – hot – coding: phải chuyển đổi mỗi label yi thành một vector yi dưới dạng one-hot coding – chỉ có đúng một phần tử của yi bằng 1, các phần tử còn lại bằng 0.
* Hàm scipy.sparse.coo\_matrix giúp lưu ma trận output Y dưới dạng sparse matrix, lưu các **giá trị** khác 0 của ma trận.

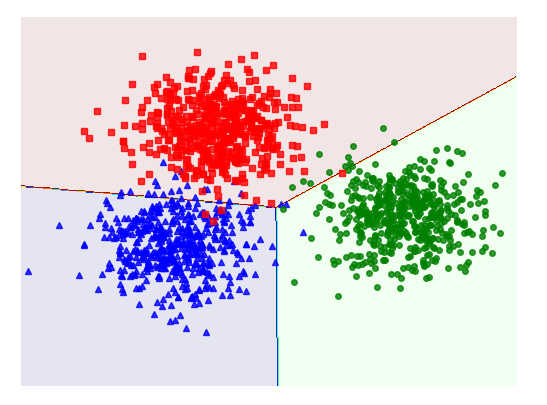


* Kiểm tra đạo hàm sau đó viết hàm chính có training Softmax Regression (theo SGD).



* Hàm dự đoán class cho dữ liệu mới - Sau khi train Softmax Regression và tính được ma trận hệ số W, class của một dữ liệu mới có thể tìm được bằng cách xác định vị trí của giá trị lớn nhất ở đầu ra dự đoán.



* Tạo ba cụm dữ liệu, thực hiện softmax regression, kết quả thu được là tạo ra các vùng cho mỗi class, xác định được đường ranh giới giữa các classes là đường thẳng.
* SR được tích hợp tronghàm sklearn.linear\_model.LogisticRegression của thư viện sklearn.
* Thêm một vài thuộc tính như: C, solver, multi\_class.
  + multi\_class = 'ovr' là giá trị mặc định, tương ứng với **one-vs-rest**.
  + solver = 'lbfgs' là một phương pháp tối ưu cũng dựa trên gradient nhưng hiệu quả hơn và phức tạp hơn Gradient Descent.
* Kết quả tạo ra – các đường biên là các đường tuyến tính.