**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**

**THUẬT TOÁN**

**Giáo viên hướng dẫn: Vũ Quang Huy**

**Sinh viên thực hiện: Lâm Thành Tài**

**Mã số sinh viên: 15110121**

**Tp. Hồ Chí Minh, tháng 12 năm 2018**

**Mục Lục**

[1. Linear Regression 3](#_Toc531085076)

[2. K-means Clustering 4](#_Toc531085077)

[3. K-means Clustering: Simple Applications 5](#_Toc531085078)

[4. K-nearest neighbors 6](#_Toc531085079)

[5. Gradient Descent 8](#_Toc531085080)

[6. Perceptron Learning Algorithm 9](#_Toc531085081)

[7. Logistic Regression 9](#_Toc531085082)

[8. Binary Classifiers 9](#_Toc531085083)

[9. Softmax Regression 11](#_Toc531085084)

[10. Multi-layer Perceptron 11](#_Toc531085085)

[12. Convex Optimization Problems 12](#_Toc531085086)

[13. Support Vector Machine 13](#_Toc531085087)

[14. Content-based Recommendation Systems 15](#_Toc531085088)

[15. Neighborhood-Based Collaborative Filtering 16](#_Toc531085089)

[16. Matrix Factorization Collaborative Filtering 17](#_Toc531085090)

[17. Principal Component Analysis 17](#_Toc531085091)

[18. Linear Discriminant Analysis 18](#_Toc531085092)

[19. Naive Bayes Classifier 18](#_Toc531085093)

[20. Accuracy, Confusion matrix, Reciever Opearating Characteristic curve, Precision and Recall, Micro – average, Macro – average 19](#_Toc531085094)

[21. Decision Trees 21](#_Toc531085095)

# **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, vậy mô hình Linear Regression nhiều khả năng sẽ cho kết quả tốt: (cân nặng) = w\_1\*(chiều cao) + w\_0
* 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.

# **K-means Clustering: Simple Applications**

* Trước tiên các bạn vào trang chủ của MNIST để download bộ cơ sở dữ liệu này. Mặc dù trong bài này chúng ta chỉ dùng bộ dữ liệu test với 10k ảnh và không cần label, các bạn vẫn cần download cả hai file t10k-images-idx3-ubyte.gz và t10k-labels-idx1-ubyte.gz vì thư viện python-mnist cần cả hai file này để load dữ liệu từ tập test.
* numpy cho các phép toán liên quan đến ma trận. [mnist](https://pypi.python.org/pypi/python-mnist/) để đọc dữ liệu từ MNIST. matplotlib để hiển thị hình vẽ. sklearn chính là scikit-learn mà chúng ta đã làm quen trong các bài trước.
* **Để hiện thị nhiều bức ảnh các chữ số cùng một lúc, tôi có dùng thêm hàm số**[display\_network.py](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/display_network.py).
* **Thực hiện thuật toán K-means clustering trên toàn bộ 10k chữ số.**
* Kết quả khi chọn ngẫu nhiên 20 bức ảnh từ mỗi cluster.
* Mỗi hàng tương ứng với một cluster, cột đầu tiên có nền xanh bên trái là centers tìm được của các clusters (màu đỏ hơn là các pixel có giá trị cao hơn). Chúng ta thấy rằng các center đều hoặc là giống với một chữ số nào đó, hoặc là kết hợp của hai/ba chữ số nào đó. Ví dụ: center của nhóm thứ 4 là sự kết hợp của các số 4, 7, 9; của hàng thứ 7 là kết hợp của chữ số 7, 8 và 9.
* Tuy nhiên, các bức ảnh lấy ra ngẫu nhiên từ mỗi nhóm trông không thực sự giống nhau. Lý do có thể là những bức ảnh này ở xa các center của mỗi nhóm (mặc dù center đó đã là gần nhất).
* Đổi phương pháp thực hiện bài toán bằng cách thay vì chọn ngẫu nhiên các bức ảnh của mỗi cluster, thì sẽ chọn 20 bức ảnh gần center của mỗi cluster nhất vì càng gần center thì độ tin cậy càng cao.
* Kết quả: có thể thấy dữ liệu trong mỗi hàng khá giống nhau và giống với center ở cột đầu tiên bên trái.
* Ví dụ thứ 2 nói về việc load ảnh, biến đổi thành 1 ma trận mà mỗi hàng là 1 pixel với 3 giá trị màu.
* Sau khi tìm được các cluster, tôi thay giá trị của mỗi pixel bằng center của cluster chứa nó, và được kết quả: ba màu - Hồng, đen, và màu da đã được phân nhóm. Và khuôn mặt có thể được tách ra từ phần có màu da (và vùng bên trong nó). Vậy là K-means clustering tạo ra một kết quả chấp nhận được.
* Nén ảnh và nén dữ liệu: có thể quan sát là khi số lượng clusters tăng lên, chất lượng bức ảnh đã được cải thiện. Đồng thời, chúng ta chỉ cần lưu các centers và label của mỗi điểm ảnh là đã có được một bức ảnh nén (có mất dữ liệu).

# **K-nearest neighbors**

* Trong ví dụ này thì chúng ta sẽ tách 150 dữ liệu trong Iris flower dataset ra thành 2 phần, gọi là training set và test set.
* Thuật toán KNN sẽ dựa vào trông tin ở training set để dự đoán xem mỗi dữ liệu trong test settương ứng với loại hoa nào.
* Dữ liệu được dự đoán này sẽ được đối chiếu với loại hoa thật của mỗi dữ liệu trong test set để đánh giá hiệu quả của KNN.
* Do gói dữ liệu Iris flower dataset có sẵn thư viện scikit – learn.
* Tiếp theo thì chùng ta sẽ load dữ liệu ảnh và hiển thị vài dữ liệu mẫu tượng trưng thông qua các class được gán nhãn : 0,1 và 2.
* Nếu nhận xét một vài dữ liệu mẫu thì hai cột cuối sẽ cho chúng ta thông tin giúp phân biệt giữa các dữ liệu với nhau.
* Đối với việc tách training và test sets, scikit – learn có hàm cho phép chúng ta ngẫu nhiên lựa chọn các điểm này.
* Trước hết xét trường hợp đơn giản K = 1, tức là với mỗi điểm test data, ta chỉ xét 1 điểm training data gần nhất và lấy label của điểm đó để dự đoán cho điểm test này.
* Kết quả cho thấy label dự đoán gần giống với label thật của test data, chỉ có 2 điểm trong số 20 điểm được hiển thị có kết quả sai lệch.
* Để đánh giá độ chính xác của thuật toán KNN classifier này, chúng ta xem xem có bao nhiêu điểm trong test data được dự đoán đúng. Lấy số lượng này chia cho tổng số lượng trong tập test data sẽ ra độ chính xác. Scikit-learn cung cấp hàm số [accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) để thực hiện công việc này.
* Trong ví dụ này, sử dụng p = 2 nghĩa là khoảng cách ở đây được tính là khoảng cách theo norm 2. Các bạn cũng có thể thử bằng cách thay p = 1 cho norm 1, hoặc các gía trị p khác cho norm khác.
* Nếu xét 1 điểm gần nhất thì có thể dẫn đến kết quả sai nếu điểm đó làn nhiễu. Để có thể thu được kết quả tốt hơn thì chúng ta nên tăng số lượng điểm lân cận, ví dụ ta lấy 10 điểm và trong 10 điểm gần nhất đó, class nào chiếm đa số thì dự đoán kết quả là class đó – major voting.
* Đánh trọng số cho các điểm lân cận:
  + Cách đánh trọng số phải thoải mãn điều kiện là một điểm càng gần điểm test data thì phải được đánh trọng số càng cao (tin tưởng hơn).
  + Cách đơn giản nhất là lấy nghịch đảo của khoảng cách này. (Trong trường hợp test data trùng với 1 điểm dữ liệu trong training data, tức khoảng cách bằng 0, ta lấy luôn label của điểm training data).
* Scikit-learn giúp chúng ta đơn giản hóa việc này bằng cách gán gía trị weights = 'distance'.
* Ngoài ra chúng ta có thể sử dụng công thức để có thể đánh trọng số, trong đó x là test data, xi là một điểm trong K-lân cận của x, wi là trọng số của điểm đó (ứng với điểm dữ liệu đang xét x), σ là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần x thì trọng số càng cao (cao nhất bằng 1).
* Trong trường hợp đánh trọng số bằng công thức trên cũng sẽ cho ra kết quả tương tự với kỹ thuật major voting.

# **Gradient Descent**

* grad để tính đạo hàm
* cost để tính giá trị của hàm số. Hàm này không sử dụng trong thuật toán nhưng thường được dùng để kiểm tra việc tính đạo hàm của đúng không hoặc để xem giá trị của hàm số có giảm theo mỗi vòng lặp hay không.
* myGD1 là phần chính thực hiện thuật toán Gradient Desent nêu phía trên. Đầu vào của hàm số này là learning rate và điểm bắt đầu. Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ.
* Thử nghiệm các điểm khởi tạo khác nhau.
* Tốc độ hội tụ của GD không những phụ thuộc vào điểm khởi tạo ban đầu mà còn phụ thuộc vào learning rate:
  + Với *learning rate* nhỏ η=0.01η=0.01, tốc độ hội tụ rất chậm. Trong ví dụ này tôi chọn tối đa 100 vòng lặp nên thuật toán dừng lại trước khi tới *đích*, mặc dù đã rất gần. Trong thực tế, khi việc tính toán trở nên phức tạp, *learning rate* quá thấp sẽ ảnh hưởng tới tốc độ của thuật toán rất nhiều, thậm chí không bao giờ tới được đích.
  + Với *learning rate* lớn η=0.5η=0.5, thuật toán tiến rất nhanh tới *gần đích* sau vài vòng lặp. Tuy nhiên, thuật toán không hội tụ được vì *bước nhảy* quá lớn, khiến nó cứ *quẩn quanh* ở đích.
* Tiếp theo chung ta sẽ khởi tạo thuật toán với hàm nhiều biến.
* Chúng ta tạo 1000 điểm dữ liệu được chọn gần với đường thẳng y=4+3x, hiển thị chúng và tìm nghiệm theo công thức.
* Đường thẳng tìm được là đường có màu vàng có phương trình y≈4+2.998x.
* Tiếp theo ta viết đạo hàm và hàm mất mát.
* Kiểm tra đạo hàm và có thể áp dụng với một hàm số.
* Sau 49 vòng lặp, thuật toán đã hội tụ với một nghiệm khá gần với nghiệm tìm được theo công thức.
* Nếu biết trước điểm đặt bi ban đầu theta, đạo hàm của hàm mất mát tại một điểm bất kỳ grad (theta), lượng thông tin lưu trữ từ vận tốc trước đó gamma và learning rate eta, chúng ta có thể viết hàm số GD\_momentum
* Ví dụ với bài toán Linear Regression theo SGD(Stochastic Gradient Descent): Chúng ta thấy rằng đường đi khá là zigzag chứ không mượt như khi sử dụng GD. Điều này là dễ hiểu vì một điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu được. Tuy nhiên, chúng ta cũng thấy rằng thuật toán hội tụ khá nhanh đến vùng lân cận của nghiệm. Với 1000 điểm dữ liệu, SGD chỉ cần gần 3 epoches (2911 tương ứng với 2911 lần cập nhật, mỗi lần lấy 1 điểm). Nếu so với con số 49 vòng lặp (epoches) như kết quả tốt nhất có được bằng GD, thì kết quả này lợi hơn rất nhiều.

# **Perceptron Learning Algorithm**

* Chúng ta sẽ tạo hai nhóm dữ liệu, mỗi nhóm có 10 điểm, mỗi điểm dữ liệu có hai chiều để thuận tiện cho việc minh họa. Sau đó, tạo dữ liệu mở rộng bằng cách thêm 1 vào đầu mỗi điểm dữ liệu.
* Sau khi thực hiện đoạn code này, biến X sẽ chứa dữ liệu input (mở rộng), biến y sẽ chứa nhãn của mỗi điểm dữ liệu trong X.
* h(w, x): tính đầu ra khi biết đầu vào x và weights w.
* has\_converged(X, y, w): kiểm tra xem thuật toán đã hội tụ chưa. Ta chỉ cần so sánh h(w, X) với *ground truth* y. Nếu giống nhau thì dừng thuật toán.
* perceptron(X, y, w\_init): hàm chính thực hiện PLA.
* Sau khi cập nhật 18 lần, PLA đã hội tụ. Điểm được khoanh tròn màu đen là điểm misclassified tương ứng được chọn để cập nhật đường boundary.

# **Logistic Regression**

* Ví dụ với dữ liệu 1 chiều, ta cần khai báo vài thư viện và dữ liệu. (numpy, matplotlib, division, …).
* Với kết quả tìm được, đầu ra yy có thể được dự đoán theo công thức: y = sigmoid(-4.1 + 1.55\*x). Với dữ liệu trong tập training, kết quả là:
* Ví dụ với dữ liệu 2 chiều: Chúng ta xét thêm một ví dụ nhỏ nữa trong không gian hai chiều. Giả sử chúng ta có hai class xanh-đỏ với dữ liệu được phân bố tùy ý.
* Với dữ liệu đầu vào nằm trong không gian hai chiều, hàm sigmoid có dạng như thác nước.
* Kết quả tìm được khi áp dụng mô hình logistic regression được minh họa với màu nền khác nhau thể hiện xác suất điểm đó thuộc class đỏ. Đỏ hơn tức gần 1 hơn, xanh hơn tức gần 0 hơn.
* Nếu phải lựa chọn một *ngưỡng cứng* (chứ không chấp nhận xác suất) để phân chia hai class, chúng ta quan sát thấy đường thẳng nằm nằm trong khu vực xanh lục là một lựa chọn hợp lý.
* Đường phân chia giữa hai class tìm được bởi logistic regression có dạng một đường phẳng, tức vẫn là linear.

# **Binary Classifiers**

* Sử dụng hàm sklearn.linear\_model.LogisticRegression trong thư viện sklearn – nhận dữ liệu ở dạng vector hàng.
* Khai báo thư viện: numpy, sklearn, scipy, … .
* Phân chia training set và test set, lựa chọn các views.
* Tạo random projection matrix.
* Xây dựng danh sách các tên files.
* **Feature Extraction:** Xây dựng dữ liệu cho training set và test set.
* sử dụng phương pháp chuẩn hóa dữ liệu Standardization. Trong đó x\_mean và x\_var lần lượt là vector kỳ vọng và phương sai của toàn bộ dữ liệu training. X\_train\_full, X\_test\_full là các ma trận dữ liệu đã được giảm số chiều nhưng chưa được chuẩn hóa. Hàm feature\_extraction giúp chuẩn hóa dữ liệu dựa vào x\_mean và x\_var của X\_train\_full.
* Thực hiện thuật toán Logistic Regression, dự đoán output của test data và đánh giá kết quả.
* Đặt giá trị cho C là một số lớn – C = 1e5.
* Để xác định nhãn của một ảnh, đầu ra của hàm [sigmoid](https://machinelearningcoban.com/2017/01/27/logisticregression/#sigmoid-function) được so sánh với 0.5. Nếu giá trị đó lớn hơn 0.5, ta kết luận đó là ảnh của nam, ngược lại, đó là ảnh của nữ.
* Để xem giá trị sau hàm sigmoid chúng ta sử dụng hàm predict\_proba.
* Kết quả thu được là xác suất để bức ảnh đó là ảnh của nam (cột thứ nhất) và của nữ (cột thứ hai).
* Ví dụ thứ 2 nói về bài toán phân biệt chữ số viết tay: Dùng bộ cơ sở dữ liệu MNIST.
* Chúng ta cũng có những bước như khai báo thư viện, load toàn bộ dữ liệu, sau bưóc này, toàn bộ dữ liệu training data và test data được lưu ở hai ma trận X\_train\_all và X\_test\_all, mỗi hàng của các ma trận này chứa một điểm dữ liệu, tức một bức ảnh đã được vector hóa.
* Để lấy các hàng tương ứng với chữ số 0 và chữ số 1, ta khai báo biến:

cls = [[0],[1]].

* Nếu bạn muốn thử với cặp 3 và 4, chỉ cần thay dòng này bằng cls = [[3], [4]]. Nếu bạn muốn phân loại (4, 7) và (5, 6), chỉ cần thay dòng này bằng cls = [[4, 7], [5, 6]]. Các cặp bất kỳ khác đều có thể thực hiện bằng cách thay chỉ một dòng này.
* Extract toàn bộ dữ liệu cho các chữ số 0 và 1 trong tập training data và test data.
* Chuẩn hóa để đưa dữ liệu về đoạn [0, 1] bằng cách chia toàn bộ hai ma trận dữ liệu cho 255.0.
* Train mô hình Logistic Regression và đánh giá mô hình.
* Chỉ có 1 ảnh bị sai lệch so với kết quả mong đợi có thể vì nét đậm nhất của nó rất giống với chữ số 1 nhưng thật ra nó là số 0.
* Hàm Logistic Regression trong thư viện sklearn có thể được dùng trực tiếp để áp dụng vào các bài toán multi-class classification với phương pháp **one-vs-rest**.
* Ba dòng lệnh cuối dùng để chạy trên toàn bộ 10 classeses.

# **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 trong hàm 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.

# **Multi-layer Perceptron**

* Tạo dữ liệu cho 3 classes mà không có hai class nào là linearly separable.
* Sử dụng một số hàm phụ trợ như: softmax, convert\_labels, cost.
* Hàm mất mát giảm dần khi số vòng lặp tăng lên.
* Áp dụng ngược network này vào phân loại dữ liệu training.
* Biểu diễn của MLP tốt hơn rất nhiều so với 1-layer Neural Network.
* Xây dựng được boundary phi tuyến.
* Thay đổi giá trị này bởi d1 = 5, 10, 15, 20.
  + Với d1 = 5, đường phân định giữa ba classes gần như là đường thẳng.
  + Với d1 = 15, mặc dù kết quả đã đạt 99.33%, vẫn có một vùng đỏ nhỏ nằm giữa nhánh màu lục và màu lam, và một vùng màu lam khá lớn giữa màu đỏ và lục. Khi một điểm dữ liệu test rơi vào những vùng này, nó sẽ bị phân loại sai.
  + Với d1 = 20, kết quả nhận được đã tương đối giống với d1 = 100. Mặc dù các đường boundary không được trơn tru cho lắm.

#### **Weight Decay**

* Dữ liệu được tạo là ba cụm tuân theo phân phối chuẩn có tâm ở [[-1, -1], [1, -1], [0, 1]].
* Trong ví dụ này, chúng ta sử dụng số hạng regularization
* Với ∥.∥F là Frobenius norm, là căn bậc hai của tổng bình phường các phẩn tử của ma trận.
* Thay đổi tham số regularization λλ và nhận được kết quả như sau:
  + Khi λ=0, tức không có regularization, ta nhận thấy gần như toàn bộ dữ liệu trong tập training được phân lớp đúng. Việc này khiến cho các class bị phân làm nhiều mảnh không được tự nhiên. Khi λ=0.001, vẫn là một số nhỏ, các đường phân chia trông tự nhiên hơn, nhưng lớp màu xanh lam vẫn bị chia làm hai bởi lớp màu xanh lục. Đây chính là biểu hiện của overfitting.
  + Khi λ tăng lên, tức sự ảnh hưởng của regularization tăng lên (xem hàng dưới), đường ranh giới giữa các lớp trở lên tự nhiên hơn. Nói cách khác, với λ đủ lớn, weight decay có tác dụng hạn chế overfitting trong MLP.
* Thay λ = 1 bằng cách thay dòng cuối cùng – mynet(1), rồi chạy lại toàn bộ code, xem các đường phân lớp thay đổi.

# **Convex Optimization Problems**

* Dùng thư viện CVPOPT để giải Bài toán canh tác.
* Sử dụng hàm solvers.lp của cvxopt giải bài toán.
* Do cần tìm giá trị lớn nhất nên ta phải đổi hàm mục tiêu về dạng −5x−3y−5x−3y. Chính vì vậy mà c = matrix([-5., -3.]).
* Hàm matrix nhận đầu vào là một list (trong Python), list này thể hiện một vector cột. Nếu muốn biểu diễn một ma trận, đầu vào của matrix là một list của list, trong đó mỗi list bên trong thể hiện một vector cột của ma trận đó.
* Thêm dấu “.” vào sau các số đó thể thể hiện đó là số thực.
* Với đẳng thức ràng buộc Ax=bAx=b, solvers.lp lấy giá trị mặc định của A và b là None, tức nếu không khái báo thì nghĩa là không có đẳng thức ràng buộc nào.
* Đối với bài toán đóng thùng:  tìm optimal point của bài toán này bằng CVXOPT.
* Không có ràng buộc và hàm mục tiêu:
* Hàm số trên là một posynomial 🡪 một Geometric Programming.
* Nghiệm thu được chính là  .

# **Support Vector Machine**

* Trước tiên chúng ta gọi các modules cần dùng và tạo dữ liệu giả .
* Giải bài toán bằng thư viện CVXOPT.
* Kết quả: hầu hết các giá trị của lambda đều rất nhỏ, tới 10-9 hoặc 10-10.
* Đây chính là các giá trị bằng 0 nhưng vì sai số tính toán nên nó khác 0 một chút.
* Vì chỉ có 3 giá trị khác 0, ta dự đoán là sẽ có 3 điểm là support vectors.
* tìm support set S rồi tìm nghiệm của bài toán.
* Kết quả: Đường màu đen đậm ở giữa chính là mặt phân cách tìm được bằng SVM. Từ đây có thể thấy nhiều khả năng là các tính toán của ta là chính xác. Để kiểm tra xem các tính toán phía trên có chính xác không, ta cần tìm nghiệm bằng các công cụ có sẵn, ví dụ như sklearn.
* Sử dụng hàm [sklearn.svm.SVC](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) cũng sẽ cho kết quả tương tự.
* Đối với soft margin SVM có 3 cách giải khác nhau:
  + Khai báo thư viện và tạo dữ liệu giả.
  + Giải bài toán bằng thư viện sklearn, chọn C=100C=100 trong thí nghiệm.
  + Tìm nghiệm bằng giải bài toán đối ngẫu: thêm ràng buộc về chặn trên của các nhân thử Lagrange. Tiếp theo, ta cần tính w và b,tìm tập hợp các điểm support và những điểm nằm trên margins – kết quả tìm được cũng tương tự như kết quả khi dùng sklearn.
  + Tìm nghiêm bằng giải bài toán không ràng buộc: tính gradient của hàm mất mát, so sánh với numerical gradient, dùng tham số lam = 1/C. Vì sự khác nhau giữa hai cách tính gradient là bằng 0 nên gradient tính được là chính xác, sau đó ta làm Gradient Descent.
  + Kết quả cho ra của cả 3 cách như nhau.
* Đối với Kernel SVM:
  + Bài toán XOR: sử dụng một bộ phân lớp
    - sigmoid: nghiệm tìm được không thật tốt vì có 3 trong 4 điểm nằm chính xác trên đường phân chia. Nói cách khác, nghiệm này rất nhạy cảm với nhiễu.
    - poly: Nghiệm này có tốt hơn nghiệm của sigmoid nhưng kết quả có phần giống với overfitting.
    - rbf: Dữ liệu được tạo ra một cách đối xứng, đường phân lớp tìm được cũng tạo ra các vùng đối xứng với mỗi class. Nghiệm này được cho là hợp lý hơn. Trên thực tế, các rbf kernel được sử dụng nhiều nhất và cũng là lựa chọn mặc định trong hàm sklearn.svm.SVC.
  + Bài toán phân biệt giới tính: sử dụng thư viện sklearn.svm.SVC để giải quyết bài toán, vì dữ liệu giữa hai classes là gần phân biệt tuyến tính nên không có sự khác nhau nhiều giữa các kernel, với dữ liệu gần phân biệt tuyến tính, linear và poly kernels cho kết quả tốt hơn.
* Đối với Multi – class SVM:
  + Tính đơn giản hàm mất mát và đạo hàm của ví dụ.
  + Hàm mất mát thì dương và không có regularization sẽ có loss tổng cộng nhỏ hơn.
  + So sánh đạo hàm với  [numerical gradient](https://machinelearningcoban.com/2017/01/12/gradientdescent/#kiem-tra-dao-ham). Nếu sự sai khác là nhỏ, nhỏ hơn 1e-7 thì ta có thể coi là gradient tính được là chính xác.
  + Sử dụng vectorized để tính đạo hàm mà không cần dùng vòng lặp for.
  + Tính hàm mất mát và đạo hàm thông qua các bước sau:
    - **Bước 1:** Tính score matrix Z=WTX.
    - **Bước 2:** Với mỗi ô, tính max(0,1−xn+xn). Sau khi tính được giá trị của từng ô, lấy các ô có giá trị lớn hơn 0 - là các ô được tô nền màu xanh lục. Lấy tổng của tất cả các phần tử ở các ô xanh lục, tính data loss.
    - **Bước 3:** với ô màu xanh lục ở hàng 2, cột 1, thì đạo hàm theo vector hệ số w2­ sẽ được cộng thêm một lượng x1 và đạo hàm theo vector hệ số w1 sẽ được trừ đi một lượng x1. Tương tự với các ô màu xanh lục còn lại. Với các ô màu đỏ ở hàng 1 cột 1, chúng ta chú ý rằng trong cùng cột 1, có bao nhiêu ô màu xanh lục thì có bấy nhiêu lần đạo hàm của w1 bị trừ đi một lượng x1. Từ đó suy ra trong khối ô vuông thứ 3, giá trị của ô màu đỏ sẽ bằng đối số của tổng số lượng các ô màu xanh lục. Vậy nên ô màu đỏ ở hàng 1 cột 1 phải bằng -2.
    - **Bước 4:** tính đạo hàm theo hệ số của class tương ứng.
  + correct\_class\_score chính là tập hợp các giá trị trong các ô màu đỏ ở khối thứ nhất.
  + Kết quả nhận được cho chúng ta thấy rằng cách tính bằng vectorized nhanh hơn rất nhiều (khoảng 120 lần) so với cách tính naive. Sự chênh lệch giữa kết quả của hai cách tính là rất nhỏ, đều nhỏ hơn 1e-10 (tức 10-10).
  + Gradient Descent cho Multi – class SVM: thử visisualize giá trị của loss sau mỗi vòng lặp, giá trị của loss sau mỗi vòng lặp có xu hướng giảm và hội tụ.

# **Content-based Recommendation Systems**

* Đầu tiên chúng ta phải download bộ cơ sở dữ liệu của MovieLens 100k.
* Cơ sở dữ liệu của chúng ta sẽ bao gồm các file: u.data, ua.base, ua.test, ub.base, ub.test, u.user, u.genre, u.item.
* Sử dụng thư viện [pandas](https://pandas.pydata.org/) để trích xuất dữ liệu, có thể được cài đặt bằng pip install pandas.
* Tiếp theo, chúng ta sẽ xây dựng item profiles, tức feature vector của mỗi item.
* Load toàn bộ thông tin về các items vào biến items.
* Xây dựng feature vector cho mỗi item dựa trên ma trận thể loại phim và feature TF-IDF 🡪 sử dụng thư viện sklearn.
* Sau bước này, mỗi hàng của tfidf tương ứng với feature vector của một bộ phim.
* Tiếp theo, với mỗi user, chúng ta cần đi tìm những bộ phim nào mà user đó đã rated, và giá trị của các rating đó.
* Tìm mô hình cho mỗi user 🡪 tìm các hệ số của Ridge Regression cho mỗi user.
* Sau khi tính được các hệ số W và b, ratings cho mỗi items được dự đoán bằng cách tính: Yhat = tfidf.dot(W) + b.
* Thử nghiệm với user có id là 100.
* Đánh giá mô hình: sử dụng Root Mean Squared Error (RMSE), tức căn bậc hai của trung bình cộng bình phương của lỗi. Lỗi được tính là hiệu của true ratingvàpredicted rating.
* Kết quả:  với tập training, sai số vào khoảng 0.9 sao; với tập test, sai số lớn hơn một chút, rơi vào khoảng 1.3.

# **Neighborhood-Based Collaborative Filtering**

* Lập trình theo hướng Hướng Đối Tượng cho class CF. Class này được sử dụng chung cho cả User-user và Item-item CF.
* Khởi tạo class CF, dữ liệu đầu vào của hàm khởi tạo class CF là ma trận Utility Y\_data được lưu dưới dạng một ma trận với 3 cột, k là số lượng các điểm lân cận được sử dụng để dự đoán kết quả. dist\_func là hàm đó similaritygiữa hai vectors, mặc định là cosine\_similarity được lấy từ sklearn.metrics.pairwise. Bạn đọc cũng có thể thử với các giá trị k và hàm dist\_func khác nhau. Biến uuCF thể hiện việc đang sử dụng User-user CF (1) hay Item-item CF(0).
* **Khi có dữ liệu mới, cập nhận Utility matrix** bằng cách thêm các hàng này vào cuối Utility Matrix. Để cho đơn giản, giả sử rằng không có users hay items mới, cũng không có ratings nào bị thay đổi.
* **Tính toán normalized utility matrix và Similarity matrix.**
* **Thực hiện lại hàm refresh, fit để cập nhật thêm dữ liệu mới.**
* Hàm \_\_pred là hàm dự đoán rating mà user u cho item i cho trường hợp User-user CF.
* \_\_pred là một phương thức private, chỉ được gọi trong class CF; predlà một phương thức public, thứ tự của biến đầu vào luôn là (user, item), bất kể phương pháp sử dụng là User-user CF hay Item-item CF.
* **Tìm tất cả các**items nên được gợi ý cho user u trong trường hợp User-user CF, hoặc tìm tất cả các users có khả năng thích item u trong trường hợp Item-item CF.
* **In toàn bộ kết quả thông qua hàm print\_recommendation.**
* **Khi áp dụng vào ví dụ MovieLens 100k,** chúng ta cùng xem kết quả với User-user CF và Item-item CF.
* Ta cần phải load dữ liệu trước, kết quả được hiển thị thông qua dòng print(‘User – user CF, RMSE = ’, RMSE) và print(‘Item – item CF, RMSE = ’, RMSE).
* Item-item CF cho lỗi nhỏ hơn (0.987) so với User-user CF (0.995) và tốt hơn so với Content-based Recommendation Systems.

# **Matrix Factorization Collaborative Filtering**

* Khởi tạo và chuẩn hóa class MF.
* Tính giá trị hàm mất mát.
* **Xác định các**items**được đánh giá bởi 1**user**, và**users**đã đánh giá 1**item**và các**ratings**tương ứng: get\_items\_rated\_by\_user(self, user\_id) và get\_users\_who\_rate\_item(self, item\_id).**
* **Cập nhật**X,W: update(X), update(W).
* **Phần thuật toán chính: fit(self).**
* **Dự đoán: pred(self, u ,i) và pred\_for\_user(sefl, user\_id).**
* **Đánh giá kết quả bằng cách đo Root Mean Square Error: evaluate\_RMSE(self, rate\_test).**
* **Đối với ví dụ về MovieLens 100k: khi sử dụng chuẩn hóa dựa trên user,** giá trị loss giảm dần và RMSE train cũng giảm dần khi số vòng lặp tăng lên. Khi chuẩn hóa dựa trên item, kết quả có tốt hơn một chút.
* **Đánh giá:** RMSE có cao hơn so với Neighborhood-based Collaborative Filtering (~0.99) một chút nhưng vẫn tốt hơn kết quả của Content-based Recommendation Systems rất nhiều (~1.2).
* Khi không sử dụng regularization, tức lam = 0 🡪 chất lượng của mô hình giảm đi rõ rệt (RMSE cao).
* Đối với ví dụ MovieLens 1M:
  + Load dữ liệu.
  + **Tách tập training và test, sử dụng 1/3 dữ liệu cho test.**
  + **Áp dụng Matrix Factorization.**
  + **Kết quả:** sau 10 vòng lặp, kết quả của Neighborhood-based Collaborative Filtering là khoảng 0.92 nhưng thời gian inference khá lớn.

# **Principal Component Analysis**

* Ví dụ này sử dụng cơ sở dữ liệu “Yale face database”.
* Sử dụng thư viện sklearn để xử lý bài toán PCA (Principal Component Analysis).
* Các hàm của sklearn đều chọn dữ liệu ở dạng hàng, nhưng trong ví dụ này chúng ta sẽ chọn dữ liệu ở dạng cột để dễ biễu diễn.
* Trong dòng pca = PCA(n\_components=K), nếu n\_components là một số thực trong khoảng (0,1)(0,1), PCA sẽ thực hiện việc tìm K.
* Các vector tìm được ở dạng vector cột, ta cần phải reshape chúng để có thể minh hoạ như các bức ảnh trong cơ sở dữ liệu “Yale face database”.
* Khi sử dụng PCA thì có thể xem được mức độ kết quả minh họa của các bức ảnh gốc so với gói dữ liệu ban đầu.

# **Linear Discriminant Analysis**

* Tạo dữ liệu giả: khai báo thư viện và các biến cần thiết.
* Các điểm cho 2 classes này được minh hoạ bởi các điểm màu lam và đỏ.
* Tính các within-class và between-class covariance matrices.
* Nghiệm của bài toán là vector riêng ứng với trị riêng lớn nhất của np.linalg.inv(S\_W).dot(S\_B).
* Đường thẳng có phương w được minh hoạ bởi đường màu lục.
* Thử nghiệm với thư viện sklearn 🡪 nghiệm tìm theo công thức và nghiệm tìm theo thư viện là như nhau.

# **Naive Bayes Classifier**

* Trước khi muốn xử lý bài toán này thì chúng ta cần phải download bộ cơ sở dữ liệu của nó có tên:  ex6DataPrepared.
* Sử dụng thử viện sklearn để giải quyết bài toán.
* Khai báo thư viện và tạo các dữ liệu cần thiết.
* Đối với mô hình Bernoulli Naive Bayes, cần thay đổi một chút về feature vectors, các giá trị khác 0 sẽ đều được đưa về 1.
* Cả hai mô hình đều cho ra cùng một kết quả đối với bài toán này.
* **Khai báo thư viện và đường dẫn tới files.**
* Hàm số đọc dữ liệu từ file data\_fn với labels tương ứng label\_fn. Chú ý rằng số lượng từ trong từ điển là 2500.
* Dữ liệu sẽ được lưu trong một ma trận mà mỗi hàng thể hiện một email. Ma trận này là một ma trận sparse nên chúng ta sẽ sử dụng hàm [scipy.sparse.coo\_matrix](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html).
* Đọc training data và test data, sử dụng class MultinomialNB trong sklearn để xây dựng mô hình và dự đoán đầu ra cho test data.
* Tiếp tục thử với các bộ dữ liệu training nhỏ hơn: khi tập training là rất nhỏ, 50 emails tổng cộng, kết quả đạt được tốt hơn.
* Trong bài toán này thì MultinomialNB hoạt động hiệu quả hơn.

# **Accuracy, Confusion matrix, Reciever Opearating Characteristic curve, Precision and Recall, Micro – average, Macro – average**

* Đối với Accuracy:
  + Tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.
  + Trong ví dụ này có thể đếm được 6 điểm dữ liệu được dự đoán đúng trên tổng số 10 điểm 🡪 độ chính xác của mô hình là 0.6(hay 60%).
  + Bài toán chỉ có 3 class nên suy ra độ chính xác nhỏ nhất là 1/3, khi tất cả các điểm được dự đoán là thuộc vào một class nào đó.
  + Tính bằng thư viện sklearn cũng cho kết quả tương tự.
* Đối với Confusion matrix:
  + Sử dụng cách biểu diễn unnormalized confusion matrix, tức ma confusion matrix chưa chuẩn hoá.
  + Có thể dùng normalized confuion matrix, tức confusion matrix được chuẩn hoá để thấy rõ hơn về cách thức hoạt động của phương pháp.
  + Lấy mỗi hàng của unnormalized confusion matrix sẽ được chia cho tổng các phần tử trên hàng đó 🡪 tổng các phần tử trên một hàng của normalized confusion matrix luôn bằng 1.
  + Khi tính bằng thư viện sklearn cũng cho kết quả tương tự.
  + Từ hàm plot\_confusion\_matrix(…) trở xuống là hiển thị confusion matrix ở cả hai dạng.
  + Với các bài toán với nhiều lớp dữ liệu, cách biểu diễn bằng màu này rất hữu ích. Các ô màu đậm thể hiện các giá trị cao. Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt. Từ hai hình trên ta thấy rằng confusion matrix đã chuẩn hoá mang nhiều thông tin hơn. Sự khác nhau được thấy ở ô trên cùng bên trái. Lớp dữ liệu 0 được phân loại không thực sự tốt nhưng trong unnormalized confusion matrix, nó vẫn có màu đậm như hai ô còn lại trên đường chéo chính.
* Đối với Reciever Operating Characteristic curve:
  + Ví dụ với hai lớp dữ liệu. Lớp thứ nhất là lớp Negative có 20 điểm dữ liệu, 30 điểm còn lại thuộc lớp Positive. Giả sử mô hình đang xét cho các đầu ra của dữ liệu (xác suất) được lưu ở biến scores.
  + Các điểm thuộc lớp 1 có score cao hơn. Thư viện sklearn sẽ giúp chúng ta tính các thresholds cũng như FPR và TPR tương ứng 🡪 ứng với threshold = 0.69637251, fpr = 0 và tpr = 0.03
* ROC cho bài toán này được minh hoạ từ hàm “plt.figure()” đến “plt.show()”.
* Đối với Precision và Recall:
  + Dựa vào Confusion Matrix cho bài toán phân loại nhị phân bắt đầu từ hàm “def cm2pr\_binary(cm)” đến “print(“precition …”).
  + Khi Precision = 1, mọi điểm tìm được đều thực sự là positive, tức không có điểm negative nào lẫn vào kết quả. Tuy nhiên, Precision = 1 không đảm bảo mô hình là tốt, vì câu hỏi đặt ra là liệu mô hình đã tìm được tất cả các điểm positive hay chưa. Nếu một mô hình chỉ tìm được đúng một điểm positive mà nó chắc chắn nhất thì ta không thể gọi nó là một mô hình tốt.
  + Khi Recall = 1, mọi điểm positive đều được tìm thấy. Tuy nhiên, đại lượng này lại không đo liệu có bao nhiêu điểm negative bị lẫn trong đó. Nếu mô hình phân loại mọi điểm là positive thì chắc chắn Recall = 1, tuy nhiên dễ nhận ra đây là một mô hình cực tồi.
  + Một mô hình phân lớp tốt là mô hình có cả Precision và Recall đều cao, tức càng gần một càng tốt.
* Đối với Micro – average:
  + Ví dụ bài toán với 3 lớp dữ liệu, bộ phân lớp cho các tham số FP, TP, FN của mỗi lớp.
  + Với ví dụ trên, micro-average precision và recall tính được là:

total\_tp = tp1 + tp2 + tp3

total\_fp = fp1 + fp2 + fp3

total\_fn = fn1 + fn2 + fn3

micro\_ap = float(total\_tp)/(total\_tp + total\_fp)

micro\_ar = float(total\_tp)/(total\_tp + total\_fn)

* + Micro-average F-Score cũng được tính tương tự như F-score nhưng dựa trên micro-average precision và micro-average recall.
  + Macro-average precision là trung bình cộng của các precision theo class, tương tự với Macro-average recall. Với ví dụ trên, ta có:

macro\_ap = (P1 + P2 + P3)/3

macro\_ar = (R1 + R2 + R3)/3

* + Macro-average F-Score cũng được tính tương tự như F-score nhưng dựa trên macro-average precision và macro-average recall.

# **Decision Trees**

* Ví dụ này làm việc với cả dữ liệu ở dạng categorical và thực hiện thuật toán ID3.
* Đầu tiên chúng ta xây dựng class TreeNode.
* **Hàm tính entropy dựa trên tần suất**: Trong hàm này, chúng ta phải chú ý bỏ các tần suất bằng 0 đi vì logarit tại đây không xác định.
* Dữ liệu trong trong ví dụ được được lưu trong file [weather.csv](https://github.com/tiepvupsu/DecisionTreeID3/blob/master/weather.csv).
* Đối với các hàm như DecisionTree hay TreeNode đều được dùng để chuyển đổi các thuộc tính đó sang dạng numerical (1, 2, 3 cho mỗi giá trị), việc chuyển đổi có thể ảnh hưởng lớn tới kết quả.
* Kết quả: dự đoán đúng 100% dữ liệu các điểm trainnig set.