DSC 190 - Introduction to Data Mining: Final Report

JERRY CHAN, University of California San Diego

.

1 INTRODUCTION

In this project, I implement eight machine learning models: Linear Regression, Logistic Regression, Naive Bayes, K-Means, Gaussian Mixture, Decision Tree, Random Forest, and Matrix Factorization. For each model, I fit them with real-world dataset and explore how different parameter sets effects the result of categorization, regression, and clustering tasks. In this report, I'll briefly explain the algorithms and implementation details and the models performance.

2 LINEAR REGRESSION

2.1 Introduction

Linear Regression is a simple regression model that aims to find the linear relationship between variables in the given data. In this section, I implemented a plain linear regression model without any regularization. There are two ways to fit the model: inverting the input matrix and gradient descent. I'm going to use gradient descent as the starter code suggested. Mathematical Interpretation:

$$Y_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... \tag{1}$$

Y_i is the target variable, x's are the features, and θ's is the weight for each features. The goal of the model is finding those θ's.

2.2 Algorithm

Begin with the input data, if a intercept is required (the input is not centered at origin) we append 1 to each column. The weight assigned to this column will be the intercept. The model initial the weight with a uniform distribution from -1 to 1. Then it runs gradient descent for multiple iterations or until early stopping is triggered (when the error rate decreasing rate is small). Following is the equation for gradient of Loss (MSE) w.r.t. weights:

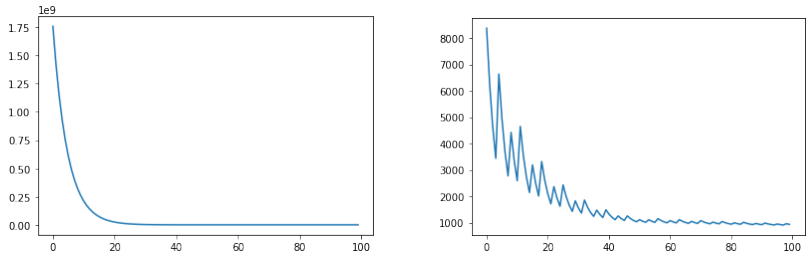
$$\frac{\partial L}{\partial W} = \frac{-2}{n} X^T (\vec{y} - XW) \tag{2}$$

Where L is the loss, n is the number of samples, X is the input matrix, y is the target output, and W is the weight matrix. The weight is updated by subtracting the product of learning rate and the gradient. The update is made only if the error rate will decreased. For learning rate, I'm using an adaptive learning rate, which decrease when the update doesn't decrease the learning rate and increase if it does.

2.3 Performance

In this section, I test the model's performance on the wine-quality dataset. The task is predicting the wine quality (integer 3 8) by density and alcohol concentration. I run the model with and without min max normalization for 100000 iteration each and compare the result. The following graph is the Loss vs Number of Iteration plot for the first 100 iterations. I choose to only show the first 100 iteration is because (1) the initial loss is too large to see any trend for the

later loss (2) the change of loss in later iteration is very small. The loss is adjusted to the not normalized loss so they can be compared.



(a) Without Normalization (b) With Normalization

Fig. 1. Training Error of Model with Normalization vs without Normalization

Table 1. Performance Comparison

Model	Loss
Sklearn Benchmark	800.6677
Without normalization	805.6663
With normalization	800.6678

The model trained with normalized data converge faster and start has a better accuracy. From the gradient descend formula, we can see that features with smaller scale gets a smaller gradient. Since all weights share the same learning rate, weights are tuned unevenly, making the model hard to train.

3 LOGISTIC REGRESSION

3.1 Introduction

Logistic regression is a model for binary classification. It is composed of a linear regression model and a logistic function. The logistic function takes the output of the linear function and outputs a prediction between 0 and 1. Mathematical interpretation:

$$Y = \frac{1}{1 + e^{-(XW+b)}} \tag{3}$$

Y is the prediction, X is the input, W is the weights, and b is the bias.

3.2 Algorithm

Similar as linear regression, I use gradient descend to train the model. The gradient is calculated as follow:

$$\frac{\partial L}{\partial y} = \frac{\hat{y} - y}{n} \tag{4}$$

$\hat{y}$: output, y: target

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial y}\sigma(\hat{y})(1-\sigma(\hat{y})) \tag{5}$$

σ: logistic function, Z: input to the logistic function

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Z} \tag{6}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial Z} \tag{7}$$

3.3 Performance

I continued to use the wine quality dataset. I extract the wine that has a quality rating of 5 or 6. The task is to predict whether the quality rating is 5 or 6 on this subset of data using all the features (acidity, dioxide, density, etc.). The result is measured by cross entropy and train the model for 100000 iterations.

The final training loss is and the final training accuracy is .

4 NAIVE BAYES

4.1 Introduction

Naive Bayes is a classification model using probabilistic approach. It uses the assumption that all features are conditional independent.

4.2 Algorithm

In training set, calculate all $P(y)$ and $P(x_i|y)$ for all category ys and features x_i from training data. In prediction step, it approximate

$$P(y|x_1, x_2, ...) \tag{8}$$

with

$$\begin{aligned} P(y|x_1, x_2, ...) &\propto P(y, x_1, x_2, ...) \\ &\propto P(y)P(x_1|y)P(x_2|y)... \\ &\propto P(y)\Pi_{i=1}^{n} P(x_i|y) \end{aligned} \tag{9}$$

4.3 Performance

The dataset I used for this section is the balance scale dataset. The data is classified as tip to the right, tip to the left, or be balanced. The features are the left weight, the left distance, the right weight, and the right distance. Below is the comparison between the Naive Bayes model from Sklearn (CategoricalNB) and our model.

My model achieve the same accuracy as the one from Sklearn, meaning my implementation is correct.

Table 2. Performance Comparison

Model	Test Accuracy
Sklearn Benchmark	0.884
Our model	0.884

5 K-MEANS

5.1 Introduction

K mean is a clustering algorithm that takes in a k and splits the given data points into k clusters. The model aims to minimize the in cluster sum of squares.

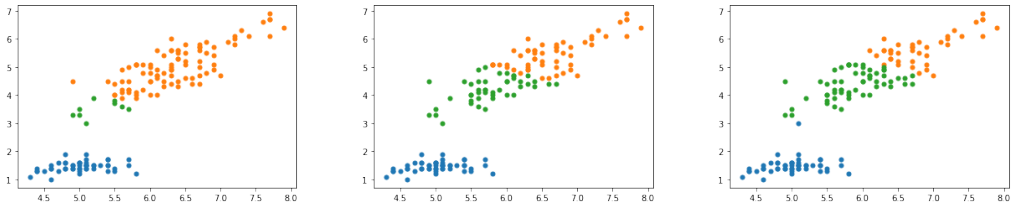
5.2 Algorithm

The model is initialized by randomly select k data points as "centroids". We then fit the model to the dataset by repeating the following steps:

1. Assign each data points to the nearest centroid. The points assigned to the same centroid is now a cluster.
2. Calculate the mean of the data points in each clusters.
3. If the mean equals to the centroid, the model is fitted, else make the means the new centroids.

5.3 Performance

I perform k-means on the Iris dataset. The dataset contains three kinds of Iris and I select 'petal length (cm)' and 'petal width (cm)' to be the only features used by the model. The figure below is the visualization of the training process. It took the model 11 iteration to converge. The centroid of the green cluster shifted upward as the training proceeds, leading to more reasonable clusters.



(a) Iteration 1 (b) Iteration 5 (c) Iteration 11

Fig. 2. Training Process of K-mean Model

K-means model also works on higher dimension data space. I repeat the same task with one more feature: 'petal length (cm)'. The following plot is the result:

6 GAUSSIAN MIXTURE

6.1 Introduction

Gaussian Mixture is a "soft" version of K-means algorithm. Each cluster is a Gaussian distribution. For each data point, we calculate the probability of it being generated by each clusters. Then, it's assigned to the cluster which its most

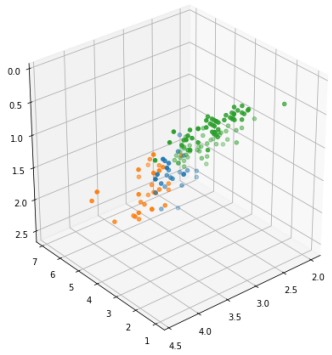


Fig. 3. Result of 3 Dimension K-means

likely to be in. Then we update the parameter for the distribution. It's a soft clustering algorithm as it gives each data point a probability distribution instead of a hard label.

6.2 Algorithm

1. Decide the initial centroids with K-means algorithm. 2. Calculate the posterior probability. 3. Find the parameters for each cluster's distribution. 4. Repeat 2 3 for a set amount of time.

6.3 Performance

In this section, we test our Gaussian Mixture model with simulated dataset generated by "Sklearn.dataset.make_blobs". I set the cluster standard deviation to 2, number of cluster to 3, and the number of feature (the dimension of the feature space) to 3. I generate 500 samples with the function and the following plot shows the 3 dimension scatter plot of the data.

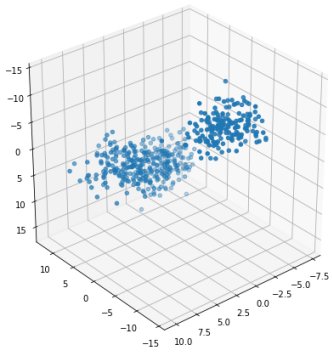


Fig. 4. Simulated Data for Testing Gaussian Mixture Model

I fit the data to my Gaussian Mixture model and trained it for 100 epochs. I label the data to the cluster with max likelihood. The result is shown as the figure below:

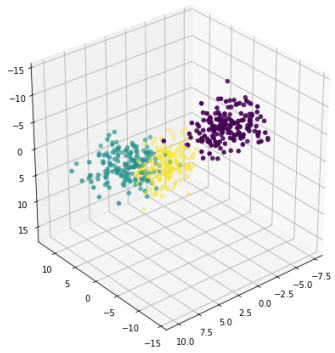


Fig. 5. Clustering Result of Gaussian Mixture Model

We can see that our model make a pretty reasonable clustering result. To further confirm my implementation is correct, I ran the same test on sklearn model "sklearn.mixture.GaussianMixture". The following table is the clusters produced by each model:

Cluster	Feature 1	Feature 2	Feature 3
1	-1.734	8.038	8.216
2	0.143	-9.169	-8.314
3	4.705	5.265	2.924

Cluster	Feature 1	Feature 2	Feature 3
1	-1.741	8.041	8.223
2	0.143	-9.169	-8.314
3	4.696	5.269	2.930

Table 3. Sklearn (left) vs My Model (right): Cluster Means

The mean absolute difference of weighted log probabilities for each sample between the result of the two models is 0.03. Based on the comparisons, we can conclude that the results are identical.

7 DECISION TREE

7.1 Introduction

Decision tree is a tree like model. The tree is built by splitting the training data to maximize the information gain of each split. The model predict the output by observing some features of the input to decide which leaf the input belongs and make the prediction based on the training data in that leaf. In this section, we are going to implement a classification decision tree.

7.2 Algorithm

7.2.1 Train.

1. List all the splitting rule and select random subset of features and rules (the number of using features and rules is controlled by the user)
2. Find the splitting rule with the highest information gain
3. Record the splitting rule and split the data correspondingly
4. Determine if the split is valid (check if the tree exceed the depth limit limit, if the size of the spitted data is greater

than size limit)
5. If the split is valid, built the two child nodes with those data, else calculate the mode of the data label as the prediction of the node and mark it as a leaf node.

7.2.2 Predict.

1. start from the root, follow the rule of each node until reaches the leaf node
2. return the prediction of the leaf node

7.2.3 Information Gain.

We use the information gain to measure the quality of the split. Information gain equals to the cross entropy of the parent node (before the split) and the weighted (by the size of child node) mean of the cross entropy of the child node (after the split). The higher the information gain is, the better the split is.

7.3 Performance

I test the model using wine quality dataset. The dataset contains 1599 samples of wine properties, such as acidity and density, and their quality ratings, integer from 3 to 8. The task is to predict the quality rating by the properties. I trained the model with 80 percent of the data and tested it on the other 20 percent. I calculated the training and testing accuracy and compare the model's performance with the decision tree in sklearn package (sklearn.tree.DecisionTreeClassifier). The sklearn model is set The result is set to use entropy as splitting criterion, and rest of the settings are default. The result is shown in the table below:

Model	Train Accuracy	Test Accuracy
Sklearn Benchmark	1.000	0.6281
My Model	1.000	0.6281

Table 4. Performance Comparison - Decision Tree

The train accuracy and test accuracy of both model are the same, proving my implementation is correct. We can see that both model achieve perfect accuracy on training set. This is because we didn't give really loose restriction on node size (2) and tree depth (1000). The tree can split the data until they are fully separated and overfitting to the training data. This causes both model to have a very low test accuracy of 0.6.

8 RANDOM FOREST

8.1 Introduction

Random forest is a ensemble model using the bagging method to combine multiple independent decision trees. The model is widely use in prediction task as it requires very little configuration.

8.2 Algorithm

The model build multiple decision trees. Each tree are limited to random subset of splitting rules on every split, resulting different tree in each built. The model generated its prediction by calculating the mean or mode of all the decision trees' predictions.

8.3 Performance

I perform the same test on the wine dataset as I did in the decision tree section. I compared the outcome with the random forest model from sklearn (sklearn.ensemble.RandomForestClassifier). For my model, I set the number of splitting rules and number of features accessible in each split to the square root of the total number of rules / features. This prevent the model from overfitting to the training data and generate unique decision trees. For the sklearn model, I set the splitting criterion to "entropy" to match the criterion used by my model and keep other settings as default. I trained 100 decision trees for each model. The result is shown as the table below:

Model	Train Accuracy	Test Accuracy
Sklearn Benchmark	1.000	0.7031
My Model	1.000	0.7125

Table 5. Performance Comparison - Random Forest

Both model achieved similar accuracy, proving my implementation is correct.

9 MATRIX FACTORIZATION

9.1 Introduction

Matrix factorization is a recommend system algorithm that generate latent features of the ranking matrix. The model is trained by gradient descend.

9.2 Algorithm

Let the number of feature in the latent space be l. The goal of this algorithm is to generate a latent feature matrix P and a matrix Q such that rating matrix $R = PQ^T$. The shape of R is $(numberofusers, numberofitems)$. The shape of P is $(numberofusers, l)$. The shape of Q is $(numberofitems, l)$. P and Q are derived by gradient descend. Given learning rate α and normalization parameter λ for each observation (user u gives item i rating $R_{u,i}$) we update P and Q follows the equation below:

$$\hat{R_{u,i}} = \Sigma_{k=1}^{l} P_{u,k} * Q_{i,k} \tag{10}$$

$$e = R_{u,i} - \hat{R_{u,i}} \tag{11}$$

$$P_u += 2 * \alpha * e * Q_i - \lambda * P_u \tag{12}$$

$$Q_i += 2 * \alpha * e * P_u - \lambda * Q_i \tag{13}$$

The model can have a bias b_u for each users, a bias b_i for each items and a global bias b, which is set to the overall mean rating. We prediction the rating with:

$$\hat{R_{u,i}} = \Sigma_{k=1}^{l} P_{u,k} * Q_{i,k} + b_u + b_i + b \tag{14}$$

We update the bias with the following equations:

$$b_u += 2 * \alpha * e - \lambda * b_u \tag{15}$$

$$b_i += 2 * \alpha * e - \lambda * b_i \tag{16}$$

The model also uses adaptive learning rate as in the linear regression section.

9.3 Performance

The model is tested with the MovieLens Latest Datasets. The dataset contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The dense rating matrix is transformed to a list of tuples each contains a user id, a movie id, and a rating. The data is splitted into train set and test set with a 8:2 ratio. I tuned the model with and without bias terms. The best set of hyperparameters and the result is shown below:

Learning Rate	Normalization	Number of Iterations	Number of Latent Features
5e-4	6e-3	50	5

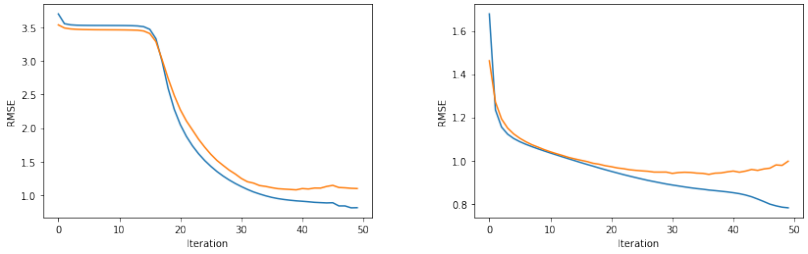
Table 6. Hyperparameters for Matrix Factorization

Result:

Model	Test RMSE
without bias	1.080
with bias	0.936

Table 7. Result for Matrix Factorization

Error vs iteration plot:



(a) Without bias (b) With bias

Fig. 6. Training Error of Model Without Bias vs Without Bias

The model with bias has a better performance and converges faster. The errors are under the provided benchmark (1.1 for model without bias and 1.0 for model with bias).

A SOURCE CODE

- Linear Regression - Linear_Regression.ipynb
- Logistic Regression - Logistic_Regression.ipynb
- Naive Bayes - Naive_Bayes.ipynb
- K Means - KMeans.ipynb
- Gaussian Mixture - KMeans.ipynb
- Decision Tree - DecisionTree&RandomForest.ipynb
- Random Forest - DecisionTree&RandomForest.ipynb
- Matrix Factorization - Matrix_Factorization.ipynb

B DATASETS

- Wine-Quality Dataset: https://archive.ics.uci.edu/ml/datasets/wine+quality
- Balance Scale Dataset: https://archive.ics.uci.edu/ml/datasets/balance+scale
- MovieLens Latest Dataset: https://grouplens.org/datasets/movielens/
- Iris Dataset: https://archive.ics.uci.edu/ml/datasets/iris