Final Project Report

Wavelix

Abstract

In this project we apply K-Means++, soft K-means, PCA, Nonlinear Autoencoders, MLP SVM, SVM with Gaussian kernel, and AdaBoost on the dataset. The results demonstrate that MLP and AdaBoost achieved the highest accuracy for binary classification tasks, with accuracies exceeding 0.97. Soft K-Means showed greater flexibility and stability compared to K-Means++, particularly with careful selection of the parameter. Dimensionality reduction techniques, such as PCA and nonlinear autoencoders, effectively simplified the dataset while preserving essential structures, improving clustering performance on reduced dimensions. SVM with Gaussian kernels excelled in handling complex decision boundaries but showed sensitivity to hyperparameter tuning.

The study highlights the importance of method selection based on dataset characteristics and task requirements, emphasizing that combining complementary approaches can enhance performance and provide deeper insights. Balancing accuracy, computational efficiency, and model complexity is essential for real-world machine learning applications.

Contents

1 Introduction **1**

2 Methodology and Solution **1**
2.1 K-Means++ Algorithm 1
2.2 Soft K-Means Algorithm 2
2.3 PCA Implementation 2
2.4 Nonlinear Autoencoder Implementation 3
2.5 Clustering with Reduced dimensions 3
2.6 MLP for Multi-Class Classification 3
2.7 SVM and SVM with Gaussian Kernel 3
2.8 AdaBoost Algorithm 4

3 Results and Analysis **4**
3.1 Results 4
3.2 Analysis 6

4 Conclusion **7**

1 Introduction

The project aims to analyze and classify the wheat seed dataset using K-Means++, soft K-means, PCA, Nonlinear Autoencoders, MLP, SVM, SVM with Gaussian kernel, and AdaBoost, and will compare the performance of these methods on both multi-class and binary classification tasks.

2 Methodology and Solution

2.1 K-Means++ Algorithm

This section aims to apply the K-Means++ algorithm to the dataset with K=3 to perform clustering. The first step is to initialize the center:

1. Randomly select the first data point as the initial center using random seeds.
2. For each remaining data point $x^{(n)}$, compute the squared distance $D(x^{(n)})$ to its nearest centroid.

3. Construct a probability distribution for selecting the next centroid:

$$P(x^{(n)}) = \frac{D(x^{(n)})^2}{\sum_j D(x^{(j)})^2}$$

   Points farther from existing centroids have a higher probability of being selected. Use the probability to select the next centroid based on the distribution.

4. Repeat the process until k centroids are chosen.

Once the initial centroids are determined, assign each data point to the cluster of its nearest centroid. For each point $x^{(n)}$, compute the Euclidean distance to all centroids, and assign the point to the cluster. After assigning samples to clusters, update the centroids based on the current cluster memberships. The new centroid is calculated as the mean of all points in the cluster.

To evaluate clustering performance, compare the predicted clusters with the true labels using accuracy metrics:

$$\text{Accuracy} = \frac{\text{number of correctly classified samples}}{\text{total number of samples}}$$

Since different random seeds can influence the results, we choose to repeat the algorithm with various random seeds and calculate the mean accuracy.

2.2 Soft K-Means Algorithm

This section aims to apply the Soft K-Means algorithm to the dataset with K=3 to perform clustering.

Firstly, randomly initialize k centroids. Then, calculate responsibility for each point respect to each cluster:

$$r_k^{(n)} = \frac{\exp[-\beta d(m_k, x^{(n)})]}{\sum_j \exp[-\beta d(m_j, x^{(n)})]}$$

The centroid are updated by computing a weighted mean of the data points:

$$m_k = \frac{\sum_n r_k^{(n)} x^{(n)}}{\sum_n r_k^{(n)}}$$

Also, calculate the mean accuracy to evaluate clustering performance.

2.3 PCA Implementation

This section aims to use PCA to reduce the dimensionality of the dataset and visualize the principal components with dimension being 2 and 3, respectively.

Firstly, subtract the mean of each features from the data, which ensure the data is centered around the origin. Then, compute the covariance matrix:

$$C = \frac{1}{n} X^T X$$

Compute the eigenvalues and eigenvectors of the covariance matrix. Then, Select the top k eigenvectors corresponding to the largest eigenvalues to construct the transformation matrix:$W = [v_1 \quad v_2 \quad \cdots \quad v_k]$. After that, project the data by $X_{\text{reduced}} = X \cdot W$. The reduced data can be transformed back to the original space to approximate the original data:

$$X_{\text{reconstructed}} = X_{\text{reduced}} \cdot W^T + \text{mean}$$

The reconstruction error is calculated by:

$$\text{reconstruction error} = \frac{1}{n} \sum_{i=1}^{n} ||X_{\text{original}} - X_{\text{reconstructed}}||^2$$

2.4 Nonlinear Autoencoder Implementation

This section aims to use nonlinear autoencoder to reduce the dimensionality of the dataset and visualize the principal components with dimension being 2 and 3, respectively.

We choose the leaky **ReLU** (with α=0.01) as the activation function. The reduced dimension is determined by the size of the last hidden layer.

Since the structure and updating process of autoencoder is very similar to that of MLP, here we have no more explanation.

The reconstruction error is also calculated after the training.

2.5 Clustering with Reduced dimensions

This section aims to apply K-Means++ and Soft K-Means on the reduced dimensions obtained from PCA and Nonlinear autoencoder, and compare the result with the ones obtained by original data. The algorithms are in **KMeansPPReducedDim.py** and **SoftKMeansReducedDim.py**, which just need to import the classes and functions in previous python files.

2.6 MLP for Multi-Class Classification

This Section aims to Apply a MLP to solve the multi-class classification problem, and compare the its performance with the clustering results in previous section.

Split the data to training set and test set, with the ratio of 7:3. BGD update is applied during the training process. We choose **ReLU** as the activation function for the hidden layers while **Softmax** for the output layer.

In forward pass, for each hidden layer:

$$z = a_{i-1}W + b \qquad a_i = \mathbf{ReLU}(z)$$

for output layer:

$$z = a_{i-1}W + b \qquad a_i = \mathbf{Softmax}(z)$$

The cross-entropy loss is applied to measure the error:

$$\mathcal{L} = -\frac{1}{m}\sum_{m}\sum_{K} y_{i,k}\log(p_{i,k})$$

In backward pass, output layer:

$$\delta^L = a^L - y$$

$$\frac{\partial \mathcal{L}}{\partial W^L} = \frac{1}{m}a^{L-1}\delta^L \quad \frac{\partial \mathcal{L}}{\partial b^L} = \frac{1}{m}\sum \delta^L$$

hidden layer:

$$\delta^l = (\delta^{l+1}W^{l+1}) \cdot f'(z^l)$$

$$\frac{\partial \mathcal{L}}{\partial W^l} = \frac{1}{m}a^{l-1}\delta^l \quad \frac{\partial \mathcal{L}}{\partial b^l} = \frac{1}{m}\sum \delta^l$$

In prediction stage, choose the class with the highest probability. Accuracy is calculated to estimate the performance.

The algorithm also allows binary classification mode.

2.7 SVM and SVM with Gaussian Kernel

This section aims to develop algorithm for SVM and SVM with Gaussian kernel using python and numpy.

Here I use the code provided by the professor, which includes the algorithm for SVM and SMO. In **SVM/classify.py**, both linear SVM and SVM with Gaussian kernel are implemented. Since the original dimension of the dataset is 7, which can not be visualized, I developed the SVM with reduced dimensioned obtained from PCA in **SVM/visualclassify.py**.

2.8 AdaBoost Algorithm

This section aims to develop the AdaBoost algorithm to solve the binary classification problem.

Firstly, change the label of the target to +1 and −1, then initialize the weights $w = \frac{1}{N}$. Define cost function:

$$J_m = \sum_{n=1}^{N} w_n^m [y_m(x^n) \neq t^{(n)}]$$

Weighted error on a class:

$$\epsilon_m = \frac{J_m}{\sum_n w_n^m}$$

Quality of the classifier:

$$\alpha = \ln(\frac{1-\epsilon_m}{\epsilon_m})$$

Update the weights by:

$$w_n^{m+1} = w_n^m \exp(-\frac{1}{2} t^{(n)} \alpha_m y_m(x^{(n)}))$$

The prediction is calculated by:

$$y(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$$

Accuracy is used to evaluate the performance of the model.

3 Results and Analysis

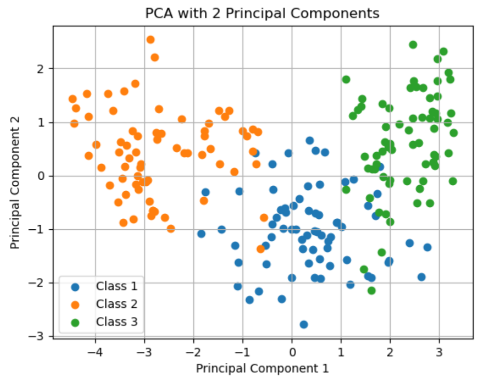
3.1 Results

Table 1 shows the accuracy of K-Means++ and Soft K-Means on both original dataset and reduced data, with 100 different random seeds tested on each case.

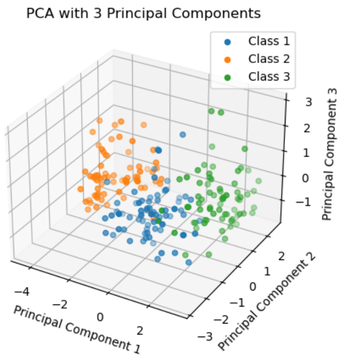
	Original Data		PCA 2D		PCA 3D		AE 2D		AE 3D	
	mean	var	mean	var	mean	var	mean	var	mean	var
K-Means++	0.8532	0.06	0.9126	0.28	0.9106	1.49	0.9116	0.06	0.8992	0.05
Soft K-Means $\beta = 0.8$	0.9048	0	0.8858	0	0.8810	0	0.6667	0	0.6652	0.31
Soft K-Means $\beta = 1$	0.8952	0	0.9000	0	0.9048	0	0.8000	0	0.6622	0
Soft K-Means $\beta = 2$	0.8952	0	0.9238	0	0.9238	0	0.8619	0	0.9095	0

Table 1: Results of K-Means++ and Soft K-Means

Figure 1 shows the visualization of PCA. The reconstruction error for 2D PCA is 0.110175, while for 3D PCA is 0.013318.



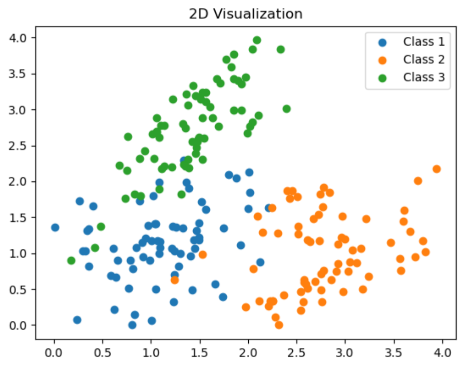
(a) PCA 2D



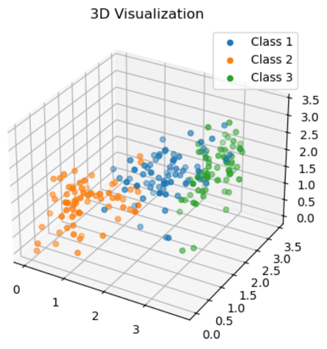
(b) PCA 3D

Figure 1: Visualization of PCA

Figure 2 shows the visualization of nonlinear autoencoder. The reconstruction error for 2D AE is 0.110175, while for 3D AE is 0.013318, the same as PCA.



(a) Autoencoder 2D



(b) Autoencoder 3D

Figure 2: Visualization of Nonlinear Autoencoder

Figure 3 shows the loss curve of MLP for multi-class classification. The final accuracy is 0.9524, with learning rate of 0.1.

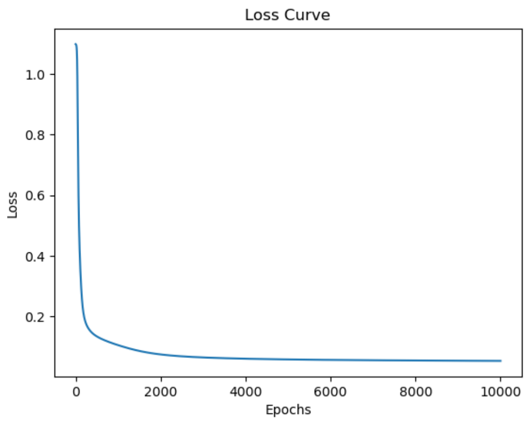


Figure 3: MLP Loss Curve for Multi-Class Classification

Table 2 and Table 3 shows the accuracy of linear SVM and SVM with Gaussian kernel, respectively.

C	0.1	1	10
Accuracy	0.9286	0.9500	0.9571

Table 2: Linear SVM

C	0.1	1	10	10
gamma	scale	scale	scale	auto
Accuracy	0.9143	0.9286	0.9357	0.9571

Table 3: SVM with Gaussian Kernel

Figure 3 shows the loss curve of MLP for multi-class classification. The final accuracy is 0.9762, with learning rate of 0.05.

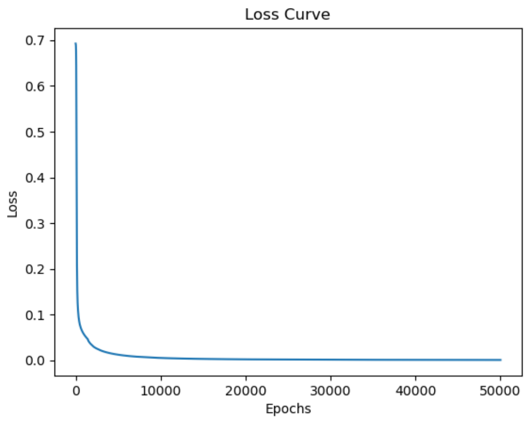


Figure 4: MLP Loss Curve for Binary Classification

For AdaBoost algorithm, we averaged the results generated from different data splitting, and the accuracy can get 0.9967 with the variance less than 0.0002.

3.2 Analysis

From Table 1, we can compare the performance between K-Means++ and Soft K-Means on both original dataset and reduced data. We can draw the following conclusions:

1. From the variance of accuracy, we know the Soft K-Means is less sensitive to the initial condition, compared to K-Means++.
2. The magnitude of β can strongly affect the accuracy of Soft K-Means. Smaller β is suitable for dataset with ambiguous boundary, while the larger one is suitable for that with clear boundary.
3. When the data is reduced, K-Means++ can have better performance. Different β need to be chosen to ensure good performance of Soft K-Means when dimension changes.

For multi-class classification problem, MLP can easily get the highest accuracy over K-Means++ and Soft K-Means. Soft K-Means is more flexible than K-Means++, while β can make the clustering process complex. A proper β can make Soft K-Means performs better than K-Means++.

C and *Gamma* can influence the performance of SVM. Larger C tends to make the SVM model strictly fit the data, while may cause overfitting. For SVM with Gaussian kernel, when *scale* is applied, $\gamma = \frac{1}{\text{n features}}$, *Gamma* is only determined by the number of features, while *auto* will consider the variance of the input data.

For binary classification problem, MLP and AdaBoost perform the best, with accuracy larger than 0.97. It's hard to compare the performance of Linear SVM and SVM with Gaussian kernel, since different hyperparameters can lead to different result. Since they have similar results, linear SVM may become a better choice for less hyperparameters.

Advantages and Disadvantages:

1. K-Means++ is easy to implement, fast to converge compared to Soft K-Means. However, the initial location of centroid can still affect the overall performance.
2. Soft K-Means is suitable for fuzzy boundary, and can reduce the likelihood of being stuck in local minima. However, the model is more complex and β is difficult to select.
3. PCA is easy and fast to implement. However, it fails to construct non-linear structure.
4. Non-linear autoencoder captures non-linear relationships, while the training process can be long and it is sensitive to hyperparameters.
5. MLP is highly flexible and is suitable for various problems. However, it requires time for training, and has many hyperparameters to choose.

6. Linear SVM has few hyperparameters, strong generalization ability, and avoids overfitting, while it is limited to linearly separable data.

7. SVM with Gaussian kernel handles non-linear data well, ideal for complex decision boundaries. However, it has many hyperparameters to choose.

8. AdaBoost can convert weak classifiers to strong one, and it is highly flexible. However, it is sensitive to noise and outliers, prone to overfitting.

4 Conclusion

This project conducted an analysis of the wheat seed dataset using a variety of machine learning techniques, including clustering, dimensionality reduction, and classification methods. Each approach provided unique insights and showed strengths and limitations in addressing the challenges posed by the dataset.

MLP and AdaBoost emerged as the most effective methods for binary classification tasks, achieving high accuracy with proper hyperparameter tuning. Among clustering techniques, Soft K-Means shows greater flexibility and stability compared to K-Means++, particularly with the careful selection of the β parameter. Dimensionality reduction techniques such as PCA and nonlinear autoencoders effectively simplified the dataset, preserving essential structures and improving clustering performance on reduced dimensions.

SVM, especially with the Gaussian kernel, excelled in handling complex decision boundaries, although its performance was sensitive to hyperparameter adjustments. Linear SVM and PCA offered simplicity but were limited in capturing non-linear relationships. In contrast, methods like nonlinear autoencoders and MLP provided the flexibility to model complex patterns.

The study highlights the importance of method selection based on the dataset characteristics and task requirements. Combining complementary approaches, as demonstrated, can enhance performance and provide deeper insights. This work underscores the value of balancing accuracy, computational efficiency, and model complexity when choosing machine learning techniques for real-world applications.