COMPARISON OF PATTERN RECOGNITION TECHNIQUES FOR CLASSIFICATION OF IRIS FLOWER SPECIES

Professor: Alireza Mohammadi

Course: ECE 5831

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Abhishek Kaniganti | Daniel Winger | Mohammed-Baker Habhab | Ruopeng Luo | Saniya Bhalla |

# INTRODUCTION

Pattern recognition systems form the backbone of numerous applications in science and technology, enabling machines to identify patterns and to make informed decisions. This project delves into the development and comparative analysis of three prevalent models in the realm of machine learning: k-means clustering, logistic regression, and neural networks. By applying these models to the well-known Iris dataset, which consists of 150 samples from three species: Iris Setosa, Iris Virginica, and Iris versicolor, we aim to classify the flowers based on their features. The k-means algorithm offers a simplistic, unsupervised approach, clustering data into pre-defined groups based on feature similarity. Logistic regression, a supervised method, employs a statistical model to predict categorical outcomes, providing a probabilistic framework for classification. Neural networks, encompassing multiple layers of interconnected nodes, harness the power of computational intelligence to capture complex, non-linear relationships within the data.

In the industry, developers use various tools and methodologies to implement machine learning models. One of the most popular programming languages used is Python. Python is a strong dynamic typed programming language. Developers and companies worldwide have created thousands of powerful outsourced libraries and modules for different applications and domains. Some libraries were created for the application of machine learning such as sklearn. The development of sklearn started in 2007 as a Google Summer of Code project by David Cournapeau. Later that year, Matthieu Brucher started work on this project as part of his thesis which later grew into a standard library used by developers and leading companies worldwide [5].

In this project, we aim, first, to implement these models from scratch and second, to implement these models using the outsourced sklearn Python library. We also perform a rigorous comparison of their performance metrics, such as accuracy, precision, and computational efficiency. Insights gained from this analysis will reveal the strengths and weaknesses of each model, guiding their optimal application in various pattern recognition tasks. In the report, we will introduce pattern recognition, then we will discuss the iris dataset and the preprocessing, then we will discuss the background, the “from scratch” implementation, the “from library” implementation, and the evaluation of each model, including the mathematical background and mechanics. Finally, we will compare the models together and derive a conclusion.

# BACKGROUND

Pattern recognition systems have evolved significantly, providing robust solutions to intricate problems across various domains. This project, structured into four parts—data exploration and preprocessing, K-means clustering, logistic regression, and neural network-based classification—aims to demonstrate the efficacy of these systems. Data visualization and preprocessing prepare the dataset for analysis. K-means clustering involves both custom and scikit-learn implementations, with performance evaluation using the elbow method and confusion matrices. Logistic regression will be implemented from scratch and with scikit-learn, focusing on model training, evaluation, and optimization through gradient descent. Lastly, the neural network section will explore various architectures and activation functions using scikit-learn's MLP Classifier. A comparative analysis will assess the accuracy and efficiency of each method, providing insights into their suitability for different types of datasets and problems. This report will walk through each part methodically, emphasizing the practical and theoretical insights gained from each approach. The thesis of this project is to demonstrate that through a detailed exploration and implementation of pattern recognition techniques, we can achieve accurate classification and a deeper understanding of machine learning principles.

# DATASET

The Iris dataset is a classic dataset in machine learning, consisting of 150 samples from three species of Iris: Iris setosa, Iris virginica, and Iris versicolor. Each sample is described by four features: sepal length, sepal width, petal length, and petal width. These features will be used to classify the species of iris flowers, making the dataset ideal for this project. The simplicity and size of the dataset allows for the practical application of various algorithms without the need for extensive computational resources.

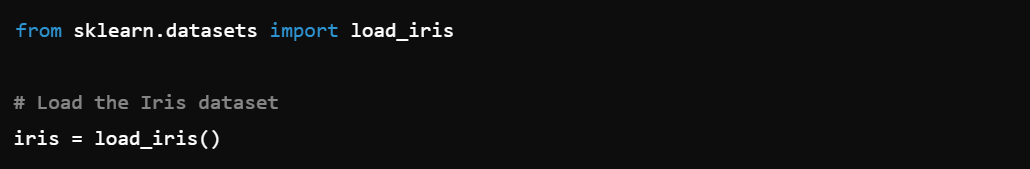
## Data Preparation and Data Cleaning

### *Introduction*

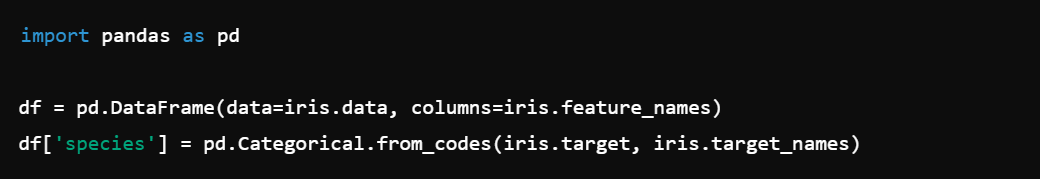
This report outlines the process of data preparation and data cleaning for the Iris dataset, a classic dataset commonly used in machine learning and statistics. The Iris dataset contains 150 samples of iris flowers from three different species, with four features measured: sepal length, sepal width, petal length, and petal width. The aim of this data preparation process is to standardize and normalize the features, ensuring that the data is in a suitable format for further analysis, such as clustering or classification.

### *Data Loading*

The first step in the process is to load the Iris dataset:



This loads the dataset into a structured format with the features stored in iris.data and the target labels (species) stored in iris.target. The dataset is then converted into a pandas DataFrame for ease of manipulation:

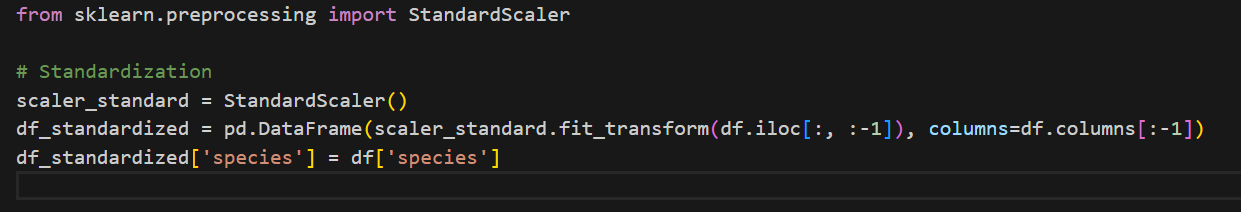


The resulting DataFrame df contains five columns: four representing the features and one for the species labels.

### *Data Standardization*

Standardization is a process that transforms the data to have a mean of zero and a standard deviation of one. This is crucial when the features have different units or scales, as it ensures that each feature contributes equally to the analysis.

The code performs standardization using StandardScaler from sklearn.preprocessing:

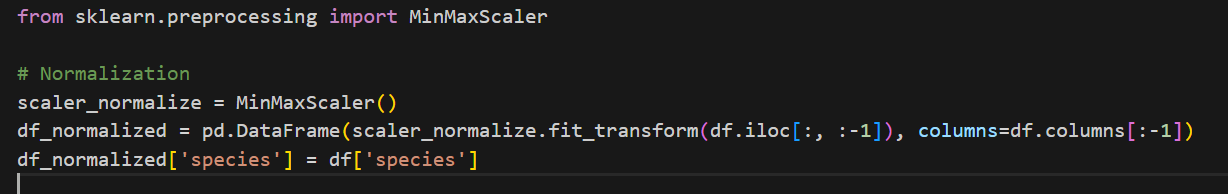


**Output**: The standardized DataFrame df\_standardized now has transformed features with a mean of 0 and standard deviation of 1. The species column remains unchanged.

### *Data Normalization*

Normalization scales the data to a fixed range, typically [0, 1]. This is useful when the goal is to constrain the features to a specific range, which can improve the performance of certain machine learning algorithms.

Normalization is performed using MinMaxScaler:

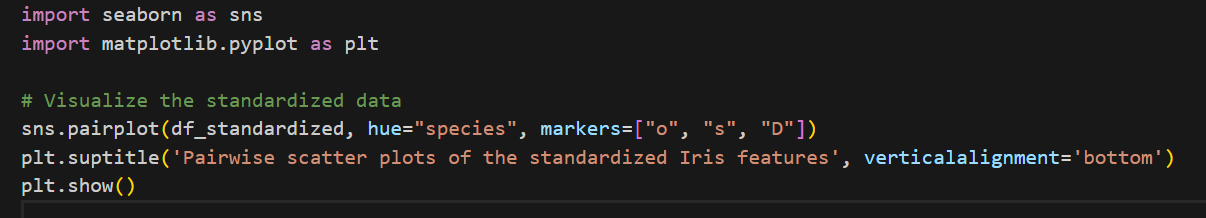


Output: The normalized DataFrame df\_normalized now has features scaled to the [0, 1] range. The species column remains unchanged.

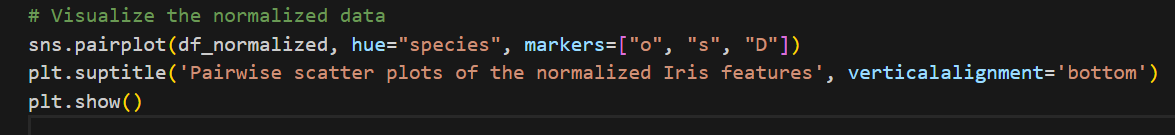
### *Data Visualization*

To visually assess the effect of standardization and normalization, pairwise scatter plots are created using Seaborn's pairplot function. These plots help to understand the relationships between different features and how the data is distributed across the species after transformation.

Visualizing Standardized Data:



Visualizing Normalized Data:



Analysis:

The pairwise scatter plots for the standardized data show how the features are centered around zero, with equal variance.

The plots for the normalized data show the features scaled within the [0, 1] range, providing a different view of the data distribution.

### *Outcome*

The data preparation and cleaning process successfully standardized and normalized the Iris dataset's features, enabling consistent and fair comparisons across features. These transformations are essential steps before applying machine learning algorithms, as they can significantly impact the model's performance. The visualizations provide a clear understanding of the data's distribution after these transformations, ensuring that the data is ready for further analysis.

# MACHINE LEARNING MODELS & METHODOLOGIES

## K-Means Classification

*About K-Means*

K-Means clustering is an unsupervised machine learning technique used for clustering data points without assigning labels. The algorithm is relatively simple. Given a dataset consisting of any number of points in an N-dimensional Euclidean space (representing, for example, data points with N features each), the points will be assigned to K clusters.

First, K centroids are initialized with random positions in the space. The algorithm then alternates between two main steps:

1. Each data point is assigned to the nearest centroid, dividing the data into clusters.
2. The centroids are updated by moving them to the average position of each point in the cluster [2].

These steps repeat until a given set of convergence criteria is met. Generally, convergence means that the clusters and centroids cease to change significantly for each iteration.

If it is unclear what number of clusters K should be used, the process can be repeated for different values of K, each time recording the objective function J, the sum of the squared distances between each point and its respective centroid. Larger values of K will always result in lower values of J, until J=0 when K equals the number of data points. However, if we plot J versus K, we will note an elbow shape. The elbow is the value of K after which J decreases much more slowly. This is generally the desired value for K.

*Mathematical Foundations of K-Means Clustering*

The core steps involve assigning data points to the nearest cluster centroids and then updating the centroids based on these assignments.

1. Centroid Update Rule:

where Cj is the set of points that belong to the j-th cluster, is the number of points in ., stands for the data points in the j-th cluster, and is the new centroid of the j-th cluster.

This formula calculates the arithmetic mean of all the data points in the cluster, ensuring that the new centroid is at the geometric center of the cluster. The centroid update rule ensures that the sum of squared distances from the data points to their respective centroids is minimized.

2. Algorithm Convergence Criteria

The K-means algorithm iterates through the following steps until convergence is achieved:

* For each data point x\_i , calculate the Euclidean distance to each centroid μ\_j .
* Assign each data point x\_i to the cluster Cj whose centroid is the nearest.
* Recalculate the centroids using the centroid update rule described above.
  + Check for Convergence:
* The algorithm stops when one of the following convergence criteria is met:
* Centroid Stability: The centroids no longer change significantly between iterations. This is measured by checking if the movement of the centroids is smaller than a predefined tolerance level: where denotes the Euclidean norm (distance), is the centroid in the current iteration, and is the centroid from the earlier iteration.

Maximum Iterations: The algorithm stops after a predefined maximum number of iterations max\_iter is reached. This prevents the algorithm from running indefinitely in cases where it might oscillate or converge very slowly.

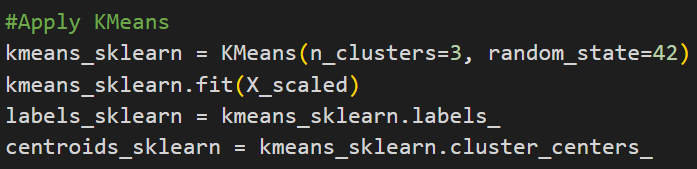
3. Objective Function and its Minimization

The K-means algorithm looks to minimize the following objective function, also known as the within-cluster sum of squares (WCSS) or inertia, where is the objective function that K-means minimizes. is the squared Euclidean distance between a data point and its assigned centroid . The sum is taken over all clusters k and all points within each cluster.

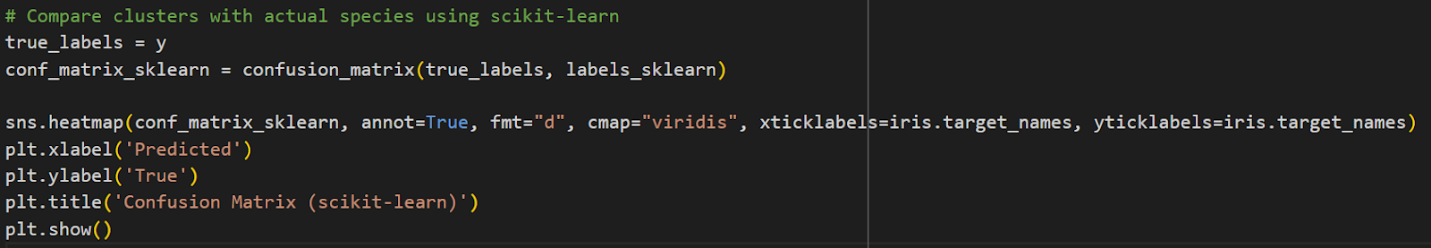
The centroid update step ensures that in each iteration, the objective function J decreases or stays the same, as the centroids are moved to the geometric center of their respective clusters. Since the function is non-increasing and bounded below zero, the algorithm is guaranteed to converge to a local minimum.

*Library Implementation*

When implementing K-Means using the SciKit-Learn library, we first import the necessary libraries and load and standardize the data. We then create an instance of KMeans and cluster the data with the following code:



Next, we use the results to create a confusion matrix to compare the generated clusters with the actual species labels:



The output is as follows:

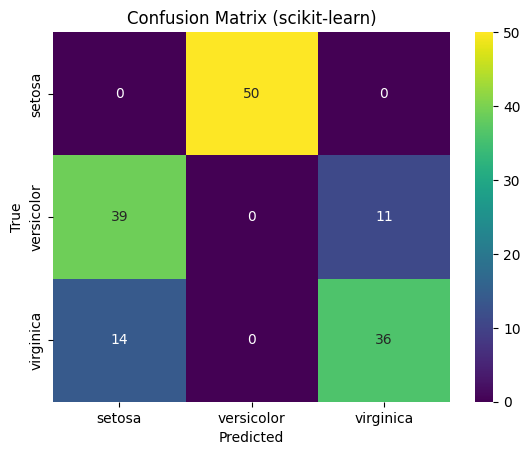
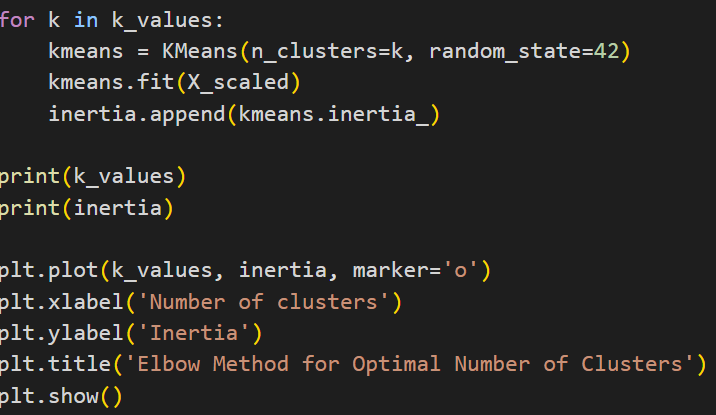


Fig 1 Confusion Matrix of Library Model (K-Means)

It is evident that our implementation of K-Means is highly inaccurate. The data is divided into three clusters with moderate accuracy, but as shown, two of the three clusters have been labeled inaccurately. This is due to the unsupervised nature of the learning algorithm.

To confirm our selection of K=3, we loop the relevant code for K=1 through K=10, record the inertia, and plot the data. The code is as follows:



The output is the following elbow graph:

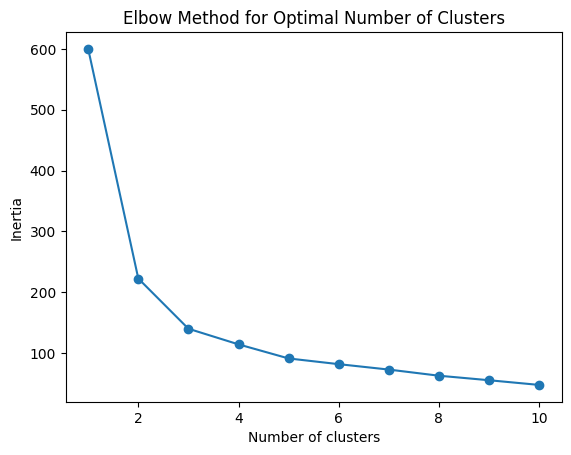
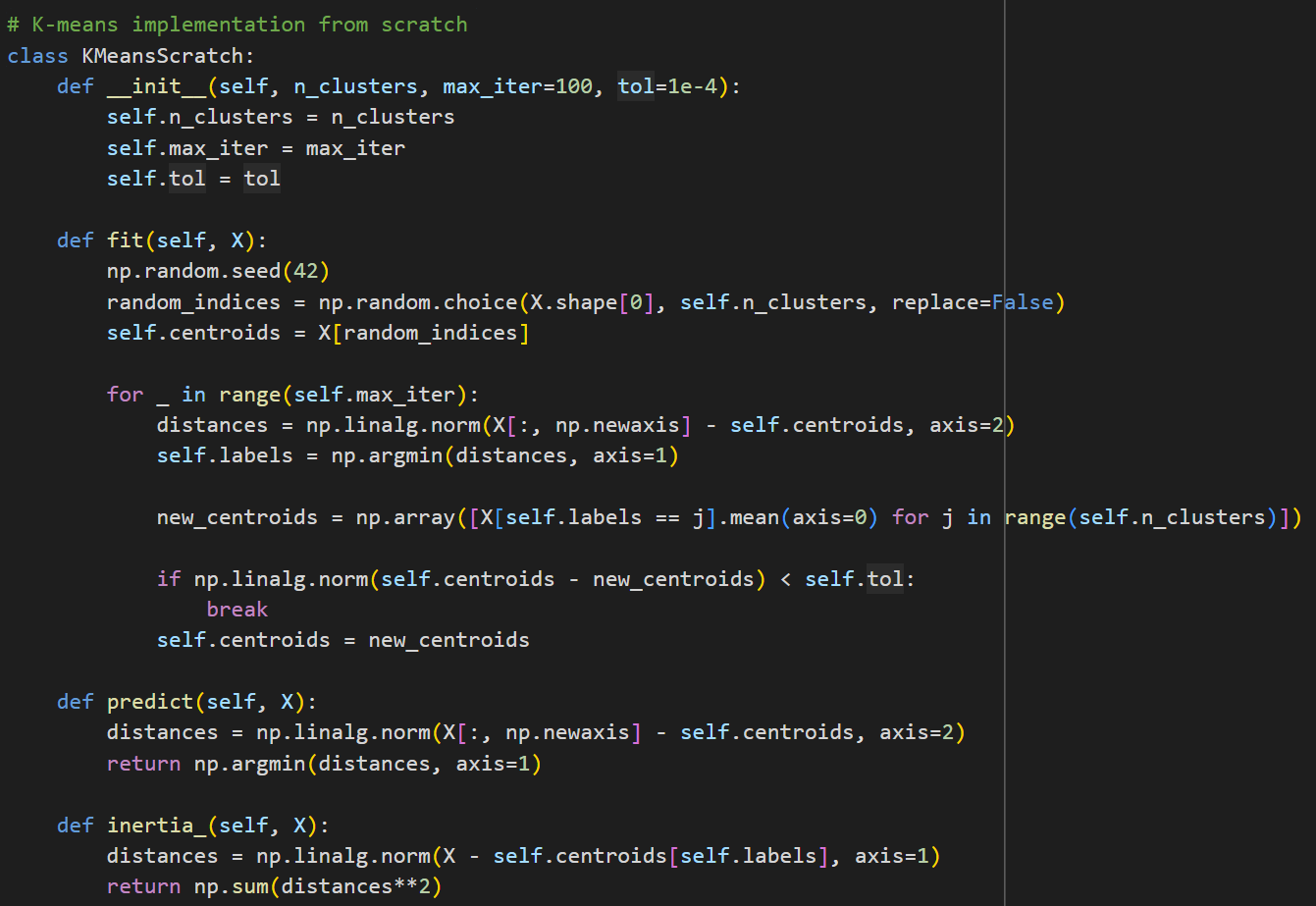


Fig 2 Elbow Method for Optimal Number of Cluster

The elbow is indeed at K=3; greater values of K reduce the inertia (i.e. the J function) only slightly.

*From-Scratch Implementation*

To implement the K-Means clustering algorithm from scratch, we first import the necessary libraries and load and standardize the data. Then we create a K-Means class as follows:



We implement the new KMeansScratch class, similarly to how this was done when using Sci-Kit Learn. Again, we create a confusion matrix to compare the clustered data to the actual species labels:

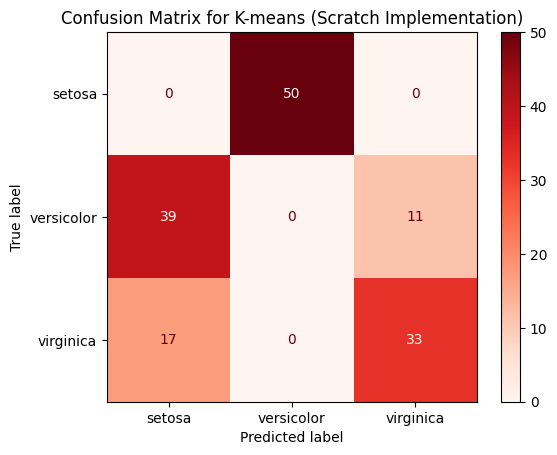


Fig 3 Confusion Matrix of Scratch Model (K-Means)

Plotting the inertia (J function) as before, we obtain the following elbow graph:

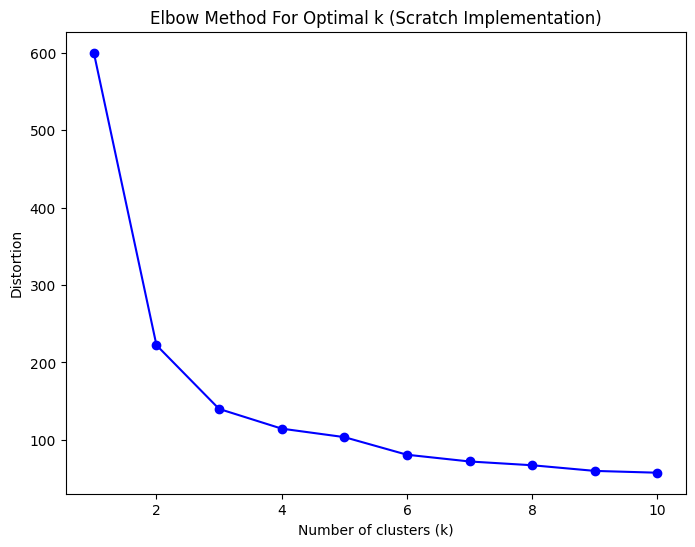


Fig 4 Elbow Method For Optimal k (Scratch Implementation)

Again, we find the elbow at K=3.

*Comparison Between Library and from-Scratch Implementations*

Both implementations of K-Means clustering produced very similar results. The SciKit Learn library implementation yielded a test accuracy of .09, and the from-scratch version, .22. Neither approach was able to label the data correctly, as K-Means clustering is unsupervised.

## Logistic Regression Classification

*Basic Concept*

Logistic Regression is a predictive analysis algorithm categorized under supervised learning. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables. In its simplest form, logistic regression models the probability of a binary outcome based on one or more predictor variables.

While basic logistic regression is typically used for binary classification problems (with two possible outcomes), it can be extended to handle multiple classes without resorting to multiple binary classifiers. This extension is known as Multinomial Logistic Regression or SoftMax Regression. It is particularly useful when the dependent variable is categorical and includes more than two levels/classes [1].

*Model Mechanisms*

In multinomial logistic regression, the probability of each category is modeled using the softmax function, which generalizes the logistic (sigmoid) function used in binary logistic regression. For each category k, the model calculates a linear combination of the predictors [2]:

where are the parameters associated with category k.

*One-Hot Encoding*

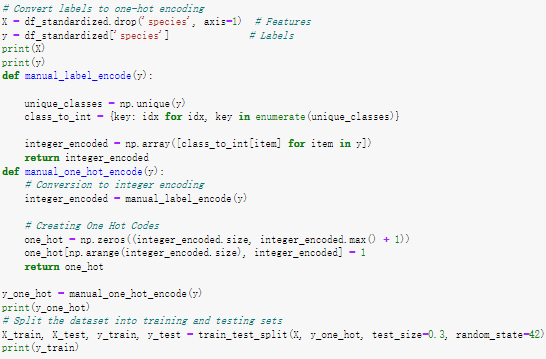
In logistic regression, particularly when implemented from scratch for multiclass classification problems, one-hot encoding plays a crucial role by transforming categorical target labels into a binary matrix representation.

Given a set of categorical labels, one-hot encoding transforms each label into a binary vector. For a label variable with distinct classes:

* Each unique label is assigned a unique position in a vector of length .
* The position corresponding to the label is set to 1, while all other positions are set to 0.

If is a label for a sample, and it belongs to class (out of classes), the one-hot encoded vector is:

where the position is 1, and all other elements are 0.



#### Softmax Function

The softmax function transforms the linear scores calculated for each category into probabilities ​, which denote the probability that the observation belongs to category k:

Here, is the exponential of the linear score for category k, and the denominator is the sum of the exponentials of the linear scores for all categories, ensuring that the probabilities sum up to 1.

图片包含 图形用户界面

描述已自动生成

##### Cross-Entropy Loss Function

Cross-entropy is a measure from the field of information theory, built upon the concept of entropy and generally used to quantify the difference between two probability distributions. For classification, it's used to measure the difference between the true label distribution and the predictions:

Here, N is the number of samples in the dataset, ​ is a binary indicator (0 or 1) if class label is the correct classification for observation , and is the predicted probability of observation being of class . This loss function penalizes the probabilities assigned to the incorrect class labels, promoting higher probabilities for the true class label.

文本

描述已自动生成

#### Gradient Computation

To optimize the parameters of the model, we need to compute the gradients of the loss function with respect to each parameter:

where is the input feature matrix (with samples as rows), represents the predicted probabilities obtained from the Softmax function, and is the matrix of true labels in one-hot encoded form. This gradient represents the average direction in the feature space across all data points where the function's output will increase most quickly.

文本, 信件

描述已自动生成

Weight Updates

Using the gradients computed, we update the weights in the opposite direction of the gradient (since we want to minimize the loss) by a step size known as the learning rate:

This iterative process, known as gradient descent, helps in finding the set of weights that minimizes the loss function, ideally leading to a better generalization on unseen data.

图片包含 图示

描述已自动生成

### Model comparison (using library & from scratch)

We have utilized two different approaches to build the multiclass logistic regression: one using a library (scikit-learn) and one from scratch. Here is a comparison of the differences between the two methods.

#### From Scratch

##### Training Process

The training process consists of several epochs where each epoch runs through the entire dataset, calculates the loss, computes the gradients, and updates the weights, as described in the previous section:

* Calculate scores: Compute the linear combinations of inputs and weights.
* Apply Softmax: Transform these scores into probabilities.
* Compute loss: Evaluate how well the model's predictions match the actual labels.
* Compute gradients: Determine how to adjust the weights to decrease the loss.
* Update weights: Adjust the weights by a small step towards reducing the loss.

文本, 信件

描述已自动生成

##### Prediction Process

For prediction, the trained model uses the weights to compute the scores for new, unseen inputs. These scores are then passed through the Softmax function to predict probabilities for each class. The final prediction is often the class with the highest probability:

* Calculate scores:
* Apply Softmax:
* Select class:

文本

描述已自动生成

#### Using Library (scikit-Learn)

##### Training Process

We initialize the logistic regression model using ‘LogisticRegression()’ class. This class can be customized with several parameters such as ‘solver’, ‘penalty’, and ‘C’ (inverse of regularization strength).

The model is trained on the dataset using the ‘.fit()’ method, which takes feature matrices and target arrays as inputs. This method adjusts the model weights based on the logistic regression algorithm.

图片包含 文本

描述已自动生成

##### Prediction Process

Use the ‘.predict()’ method for making predictions on new data, which is useful for threshold tuning.



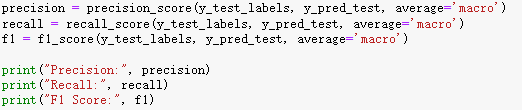
*Results* Performance and Analysis

#### Precision/Recall/F1

In order to have a comprehensive analysis of the logistic regression model implemented from scratch and using standard libraries, we have computed the effect of iteration on precision, recall and F1 score for different training cycles and maximum number of iterations, and here are the results we have obtained

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Precision Score | Recall Score | F1 Score |
| From Scratch  (Epoch = 1000) | 0.847 | 0.821 | 0.814 |
| From Scratch  (Epoch = 10000) | 1 | 1 | 1 |
| Using Library  (Epoch = 1, Max\_Iter = 1) | 0.873 | 0.795 | 0.773 |
| Using Library  (Epoch = 8, Max\_Iter = 1) | 1 | 1 | 1 |
| Using Library  (Epoch = 1, Max\_Iter = 8) | 1 | 1 | 1 |

Table 1 Precision, Recall, and F1 score for different methods and different epochs or maximum iteration



In order to show better the effect of different training Epochs on Precision, recall and f1 scores, we have conducted several training sessions, and the following are the Precision, Recall, and F1 scores of the model under different training Epochs.

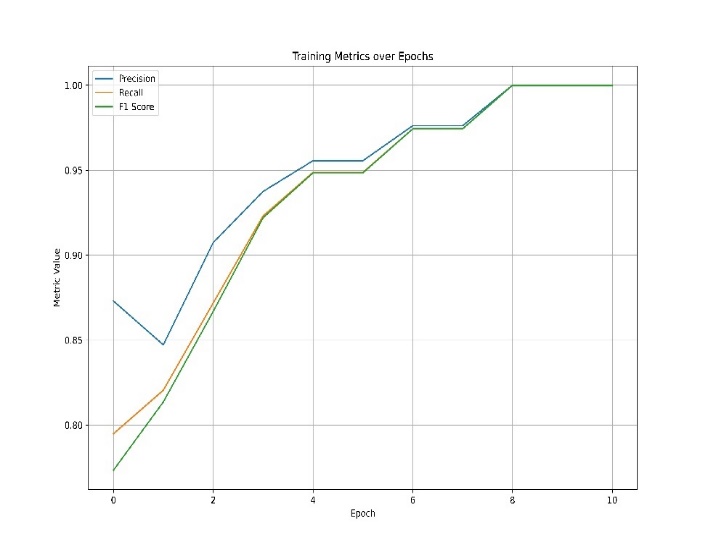
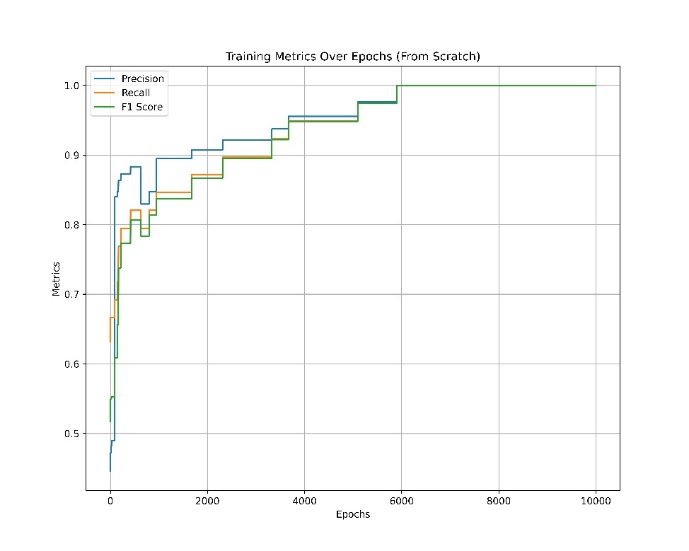


Fig 5 Training Metrics over Epochs (Left: From Scratch; Right: Library)

The scratch model shows a much slower convergence, requiring thousands of epochs to reach a comparable level of metric stability while exhibiting a rapid rate of convergence, with metrics stabilizing within relatively few epochs. As demonstrated, library models are characterized by high efficiency and fast convergence, whereas from-scratch models, although they have some efficiency limitations, provide valuable educational insights and allow detailed control over all aspects of the model training process.

#### Cross-validation Techniques: K-fold cross-validation

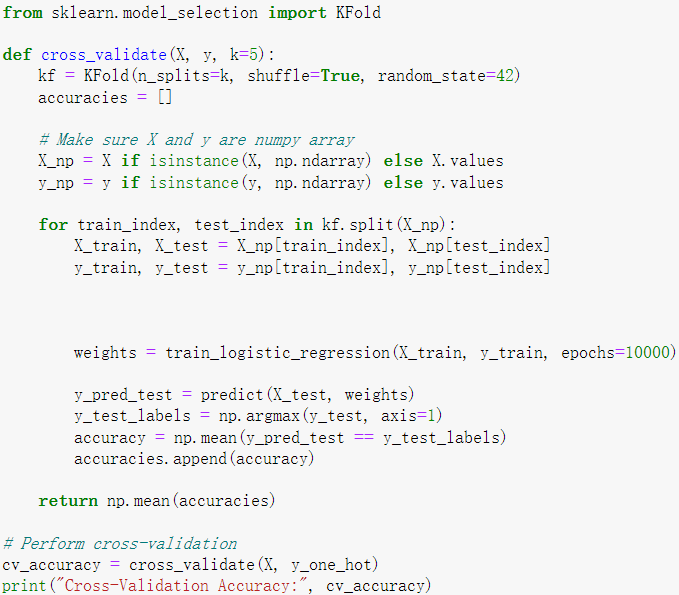
K-fold cross-validation is a statistical method used to estimate the skill of machine learning models. It is primarily used to mitigate overfitting and evaluate how a model generalizes to an independent data set. The process involves partitioning the original sample into a set of equally (or nearly equally) sized segments or "folds". The model training and validation are performed times, with each of the folds used exactly once as the test set (also known as the validation set) and the remaining folds collectively used as the training set.

Mathematical Representation

If is the accuracy of the model on the test set, then the overall cross-validation accuracy is computed as:

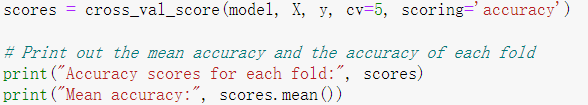
##### From Scratch

We cross-validated the accuracy of the constructed logistic regression model of “from scratch” with the following code and obtained an accuracy of 0.953.



##### Using Library

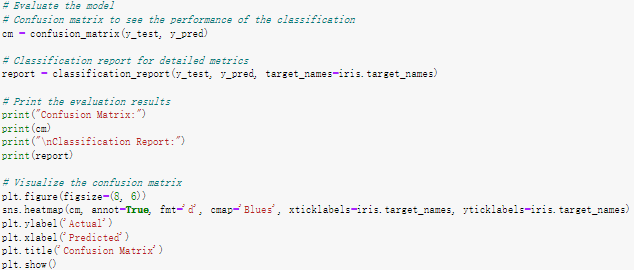
In the logistic regression model using the library, we used the cross\_val\_score from the sklearn library to calculate the accuracy of the cross-validation, and the result was calculated with the same accuracy of 0.953 as the result obtained from “from scratch”.



#### Confusion Matrices Analysis

A confusion matrix is a powerful tool in machine learning for evaluating the performance of classification models. It is essentially a table used to describe the performance of a classification model on a set of data for which the true values are known.

Confusion matrices are often visualized using heat maps, which can help quickly identify which classes are being confused by the model. We will show the confusion matrix obtained from our training results.



##### From Scratch

Below are the confusion matrices obtained from scratch's logistic regression training. The upper figure shows the results obtained at epochs of 1000, and the lower figure shows the results obtained at epochs. Because the Iris dataset is not complex, we did not take the approach of updating the weights in mini-batches during the training. Rather, we updated the weights at each epoch, so the actual number of epochs is equal to the number of iterations.

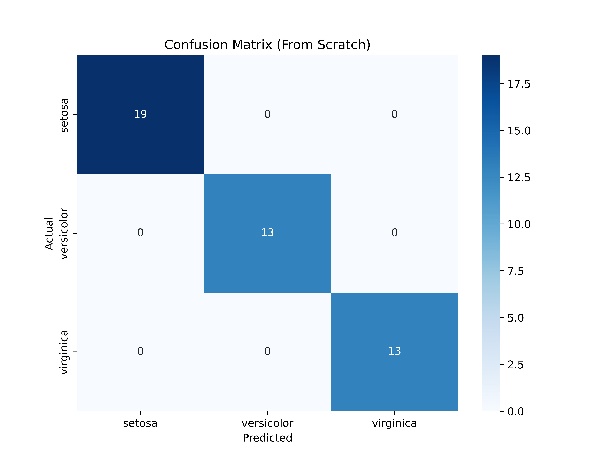
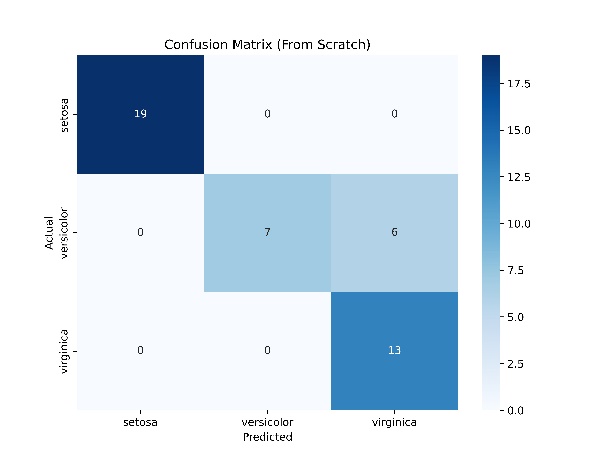


Fig 6 Confusion Matrix of Scratch Models (Left: Epochs = 1000; Right: Epochs = 1000)

We can observe an improvement as the training epochs increase, with the number of versicolor misclassified as virginica dropping from 7 in epochs = 1000 to perfect classification in epochs = 10000. Increasing the number of epochs significantly enhances the model’s ability to classify accurately the more confusable classes (versicolor and virginica). This suggests that the model requires extensive training to learn effectively and to distinguish between these classes.

##### Using Library

The following figure shows the confusion matrix obtained from logistic regression training using Library, the top figure shows the results obtained by setting ‘max\_iter = 1’ and the bottom figure shows the results obtained by setting ‘max\_iter = 8’. We can see that by changing the number of iterations we can greatly improve the accuracy of the model and get the desired confusion matrix.

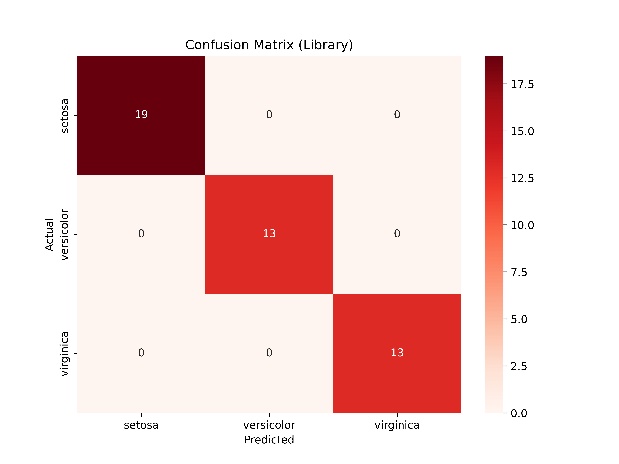
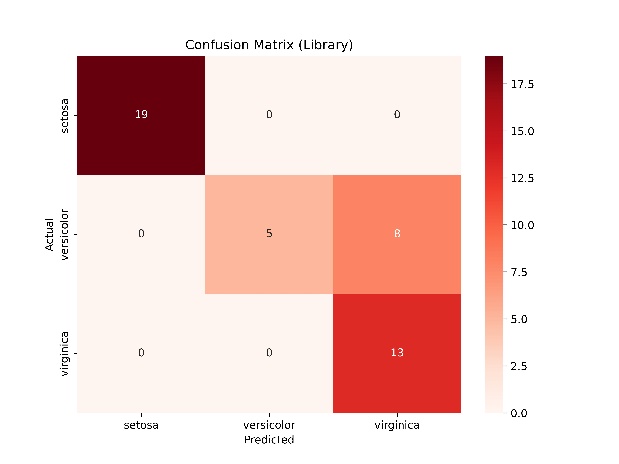


Fig 7 Confusion Matrix of Library Models (Left: Iteration = 1; Right: Iteration = 8)

Increasing max\_iter from 1 to 8 allows the model more opportunities to adjust its weights, leading to a significant reduction in errors, particularly for the classes versicolor and virginica, which are more challenging to differentiate than setosa.

#### Decision Boundary

The decision boundary visualizations are crucial for understanding how logistic regression models, implemented both from scratch and using a library, perform classification tasks on the Iris dataset [3]. Here, we show the decision boundaries of the iris features in groups of two, from which we can observe how the model differentiates between iris species from them.

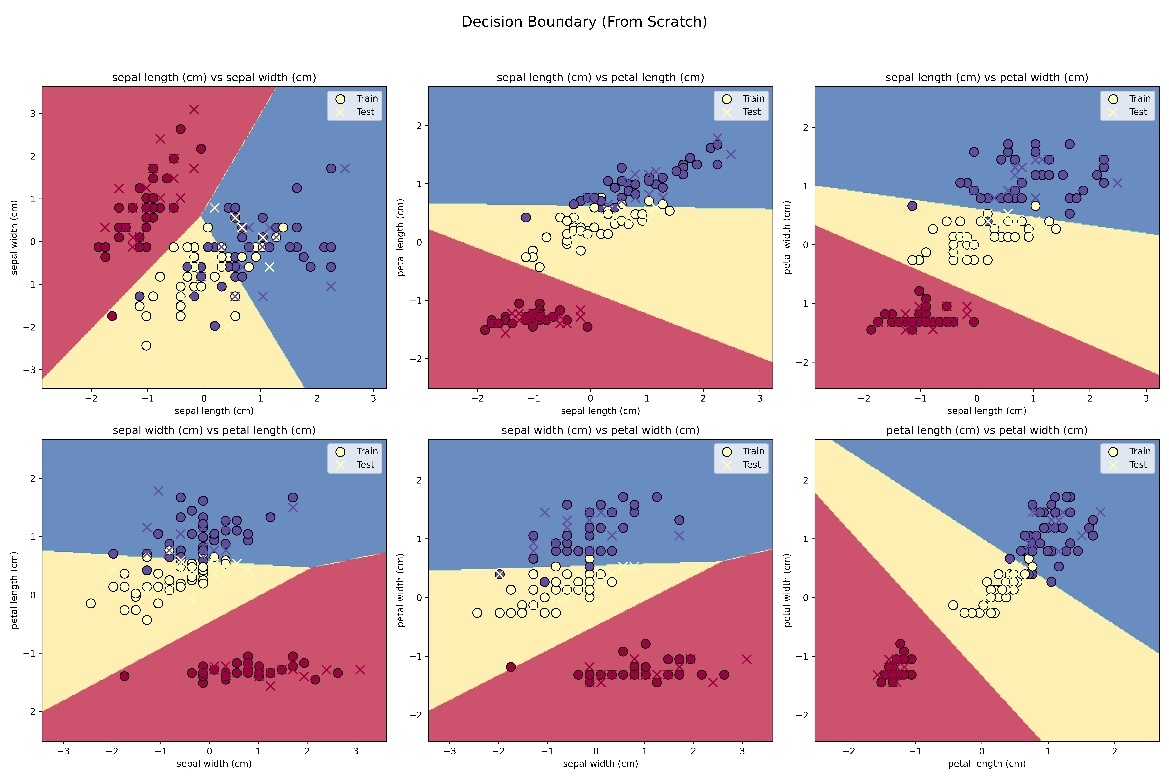


Fig 8 Decision Boundary (From Scratch)

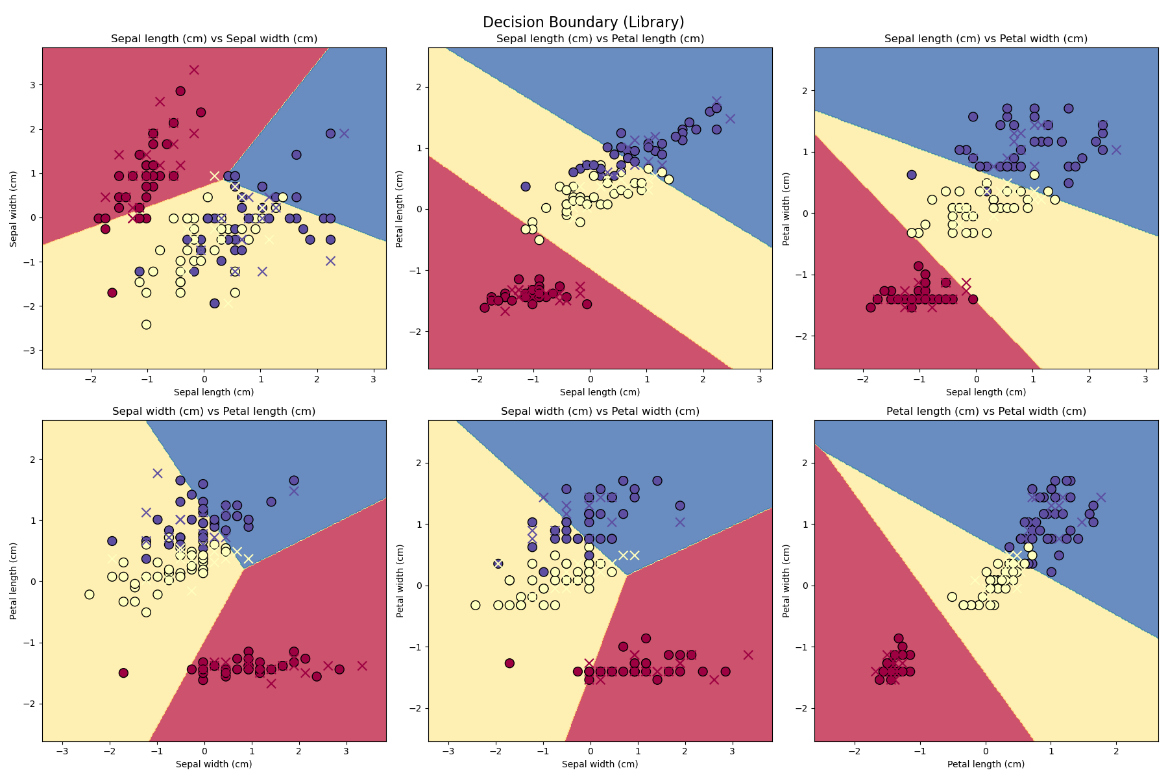


Fig 9 Decision Boundary (Library)

We can see that there are some minor differences in the decision boundaries of the models from scratch and using Library, but generally they are basically the same, and the decision boundaries are also able to distinguish between the three categories of data, which indicates that in this dataset both From scratch and using Library's logistic regression in this dataset, indicating that both From scratch and using Library's logistic regression have very favorable results.

#### Comparative Analysis of Library vs. Scratch Models

##### General Performance

Both types of implementations, when adequately trained (max\_iter = 8 for library models and epochs = 10000 for scratch models), achieve perfect classification across all classes. However, the paths to achieving this are different due to inherent differences in implementation specifics and optimization algorithms used.

##### Learning Speed and Efficiency:

Library Models: Tend to converge faster with fewer iterations or epochs needed, likely due to more sophisticated optimization algorithms (e.g., advanced variants of gradient descent like Adam or RMSprop) that are not typically part of a basic from-scratch implementation.

From Scratch Models: Require a significantly larger number of epochs to converge to similar performance levels, indicating a less efficient learning process which could be due to simpler update rules like basic gradient descent.

##### Utility and Application:

Library Model: Ideal for applications requiring rapid deployment and high reliability. The quick tuning of metrics to optimal levels makes it suitable for industry use where time and resource efficiency are critical.

From Scratch Model: Provides deeper insights into the learning mechanics of logistic regression, making it suitable for academic environments or research settings where understanding the underlying model behavior is as important as the outcome.

## Neural Network Classification

*About Neural Networks*

Neural networks are computational models inspired by biological neurons, used in machine learning and AI to recognize patterns and to solve complex problems. They consist of interconnected layers of nodes that, by adjusting parameters, process input data to make predictions.

At their core, neural networks are function approximators, creating decision boundaries for data classification. Each neuron processes multiple inputs (n) to produce an output (y) using a linear function, , where weights (W) and biases (b) are parameters. However, Linear functions alone can't handle non-linearity. Herein lies the importance of activation functions.

Activation functions () introduce non-linearity and adjust neuron outputs, allowing . The combination and layering neurons in a network with different activation functions forms a "Universal Function Approximator", capable of approximating almost any function.

*Neural Network Mechanics and Scratch Implementation*

A neural network learns by finding the boundaries of a given dataset. In a general sense, this is achieved by (1) feeding the input data into the network and retrieving the output, (2) comparing the output to the desired output and calculating the error (3) adjusting the weights and biases parameters using gradient descent (a partial derivative of error with respect to the parameter) to decrease the error, and (4) repeat (Code, 2021). The “learning” part of a neural network happens at step 3 using backpropagation.

Given the large number of nodes in a given layer of a neural network, it is easier to think of the inputs ( ), biases ( ), and outputs () as vectors, and the weights () as a matrix when analyzing a given layer. Each layer takes in inputs and returns outputs , where . This is known as forward propagation.

#### Dense Layer and Forward Propagation:

A diagram of a layer

Description automatically generatedThe dense layer is a hidden layer composed of nodes where each node is connected to the nodes from the previous layer, where each node outputs a single value. Therefore, each node in a dense layer is computed as follows , where is the activation function, is a vector of weights, is the input vector from all of the nodes in the previous layer, and is the bias. Taking this a step further, if we want to generalize this function to compute the output of all of the nodes of a given layer we will have a weight matrix , an input vector , a bias vector , and an output vector . Therefore, the function becomes. This is the forward propagation described above and is implemented as follows:

A screen shot of a computer code

Description automatically generated

Note, when a neural network is first created, its nodes are initialized with random weights and biases which are later tuned through backpropagation. This is implemented in the MLP class constructor.

A computer code with text

Description automatically generated

#### Backpropagation:

The tuning of the neuron parameters is handled by a process called backpropagation. Back-propagation is based on gradient descent. In other words, it is the process of finding which weights and biases minimize the cost function and allow for a more accurate approximation.

Therefore, for every trainable layer, a set of parameters (and ) need to be updated based on the gradient descent. This is achieved by computing the derivative of the error with respect to the parameter: , , and using the chain rule. It is important to note that of one layer is of the previous layer in backpropagation .

Similar to forward propagation, in backpropagation, the derivatives of the error with respect to the parameters are seen as vectors (, *,* and) and a matrix , where each element of these vectors and matrices is a partial derivative and are as follows: , , and . Understanding the computations and relations from a linear algebra perspective allows us to use numpy’s powerful linear algebra methods to compute these relations.

A computer screen with text and images

Description automatically generated

#### Activation Functions:

Finally, the activation functions. Activation functions are what allow the function to learn by approximation. There is a wide range of activations, each more suitable for certain applications compared to others. Some of the most common are Rectified Linear Unit (ReLU), Sigmoid, Tanh, and Softmax, which is typically used on the output layer. The forward method applies the activation function () to the input as shown above. The backpropagation, however, applies the derivative of the error with respect to the input. Looking at the error with respect to x for a given node we get . Generalizing this to all of the nodes of a given layer we get an elementwise dot product .

In our code, we implemented each with their respective derivative to allow for ease of customization and model comparison.

|  |  |
| --- | --- |
| A screen shot of a computer code  Description automatically generated | A screenshot of a computer screen  Description automatically generated |

#### Loss Function:

Loss functions allow us to measure how far the output is from the true value. This is also the error () to which we were referring previously. There are many loss functions that can be used to compute the error. In this project, we chose the cross-entropy (CE) loss function since it is known to be particularly effective in classification problems. In addition, it works well with probabilistic models that output probabilities, such as models with Softmax output activation, which is the case in our “from-scratch” implementation. The cross-entropy function is where is the true value/label (binary indicator) and is the predicted value. The CE value increases as the predicted probability diverges from the actual label. In the code, we implemented within the fit function.

A computer code with text

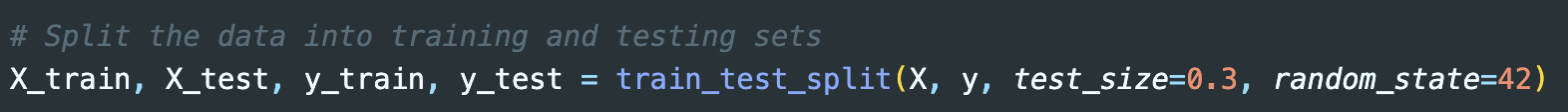
Description automatically generated

#### Implementation:

Putting these mathematical models and components together forms the neural network mechanics. In short, the data is passed to the first layer as input, the data cascades through the hidden dense layers through the forward propagation process with predefined activation functions and random weights and biases in the initial pass. The network returns a single output and is compared to the target/label to find the error. Based on the error, the backpropagation process, driven by the gradient descent, updates the weights and biases of each layer. This process repeats until the error is minimized. Then the model is tested on the “test” data and is compared to the expected output for accuracy and performance measurement.

*Library Implementation*

In this project, we prepared the data and created a Multi-Layer Perceptron neural network using the sklearn library. First, we prepared the data and split it into training and testing sets with a 70/30 split.



Then, we created three instances of the MLPClassifier class and defined the parameters of interest with different activation functions for comparison. The goal is to identify which activation function is best suited for the iris classification assuming the remaining parameters are constant such as the number of hidden layers and the number of nodes per hidden layer.

A screen shot of a computer code

Description automatically generated

The models are then trained on the “train” dataset with the target (label) and tested on the “test” dataset. Finally, during the test or prediction step, the model predictions are compared to the expected output for accuracy and performance measurements.

*Training and Implementation*

While training the models, it is important to assess the learning curves which showcase the change in error over accumulated training samples. These learning curves help us see the model’s accuracy and/or deviation (loss) throughout the learning process. With the help of chat GPT-4o (Information and Technology Services, n.d.), we created custom functions for plotting the learning curves for both the “from scratch” and the “from libraries” neural network implementations. While training the “from scratch” implementation, we see that the model with the ReLU activation performed best followed by that with the Tanh function activation. While not far behind, the model with Sigmoid activation had the slowest learning rate (Fig.10). The learning curves for the “from library” implantation are almost identical showing a steady linear increase in the model’s validation accuracy. However, the model with Logistic regression activation is slightly slower than the models with ReLU and Tanh activation functions (Fig.11). This is also reflected in the classification reports of the two sets of MLP models (Fig.12) showing large ratios - rounded to the nearest hundredth - for precision, accuracy, and recall.

A graph with a blue line

Description automatically generated

Fig 10 loss function of epochs for 3 “from scratch” MLP models with ReLU, Tanh, and Sigmoid activation functions

A graph with a green line

Description automatically generated

Fig 11 loss function of epochs for 3 “from library” MLP models with ReLU, Tanh, and Logistic activation functions

A graph of a graph of a graph

Description automatically generated with medium confidence

Fig 12 Comparison of classification reports for the three “from scratch” and “from library” MLP models

*Neural Network Prediction Analysis and Discussion*

The goal of training MLP models with various activation functions is to evaluate their impact on the specific task of iris classification. This analysis helps identify which activation function makes the model most suitable for the application. In addition to the classification report, confusion matrices are crucial tools for evaluating and comparing several MLP models with different activation functions because they provide a detailed breakdown of prediction results. By showing the counts of true positives, true negatives, false positives, and false negatives, confusion matrices allow us to assess the performance. This detailed analysis helps in identifying which activation function enables the MLP model to perform better in terms of classification accuracy and error rates, leading to more informed decisions about model selection and optimization. Given this, figure 12 lists the confusion matrixes of the “from scratch” and “from library” implementations for each activation function. By assessing the confusion matrixes (Fig.13) and the classification reports (Fig.12) we see that the activation functions ReLU, Tanh, and Logistic had identical outcomes in terms of performance and prediction. On the other hand, The ReLU and Tanh activation functions of the “from scratch” implementation showcased a 100% prediction accuracy from the confusion matrixes, and almost perfect scores in the classification report. These two models also outperformed the “from library” models. Furthermore, the two implementations had variations in the number of layers and neurons. 1 layer and 128 neurons in the case of the “from library” implementation, and 3 layers with 4, 10, and 3 neurons respectively in the case of the “from scratch” implementation. Hence, the variation in activation functions had a slight impact on the models. This indicates that the activation functions chosen were all suitable for this application given their non-linear nature.

A group of different colored squares

Description automatically generated

Fig 13 Comparison of confusion matrices of the three “from scratch” and “from library” MLP models. First row lists the “from scratch” implementation, and the second row lists the “from library” implementation.

# RESULTS & ANALYSIS

*Data Outcome and Comparison*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | K-Means | | Logistic Regression | | MLP | |
|  | sklearn | scratch | sklearn | scratch | sklearn | scratch |
| Test Accuracy | .09 | .22 | 1 | 1 | .98 | 1 |
| Training Accuracy | - | - | .96 | .96 | .96 | .98 |
| Accuracy F1 score | .09 | .23 | 1 | 1 | .98 | 1 |

Table 2 Comparison of test and training accuracy between the models: K-means, logistic regression, and the MLP with ReLU activation

The summary table compares the test and training accuracy of three models: K-means, Logistic Regression, and MLP (Multi-Layer Perceptron) with ReLU activation. The K-means algorithm performed poorly, with test accuracy of only 0.09 and 0.22 for the Scikit-learn and scratch implementations respectively. In contrast, Logistic Regression showed perfect performance, achieving 1.0 in both test and training accuracy for both implementations. The MLP model also performed very well, with nearly perfect accuracy (0.98) in the Scikit-learn implementation and 1.0 in the scratch implementation. Overall, Logistic Regression and MLP significantly outperformed K-means, especially in the Scikit-learn implementations, highlighting the superiority of these algorithms for the given dataset.

*Comparison of Confusion Matrices*

When comparing the confusion matrices of K-means (Fig 1, Fig 3), Logistic Regression (Fig 6, Fig 7), and MLP (Fig 13), similar outcomes were deduced where the K-means algorithm performed the worst, particularly in distinguishing between Setosa and Versicolor, leading to almost complete misclassification. Logistic Regression showed improvement in classifying Setosa, but there was still some confusion between Versicolor and Virginica. However, increasing the training epochs or maximum iterations significantly improved accuracy. MLP delivered the best performance, like Logistic Regression with increased training epochs or iterations, with nearly perfect classification of Setosa, though there was still some minor misclassification between Versicolor and Virginica. Overall, models implemented using libraries outperformed those built from scratch, highlighting the importance of optimized algorithms and fine-tuned parameters.

*Observations and Assessment*

Considering the unsupervised nature of the K-Means model and the accuracy score, we can conclude that the K-Means model is not suitable for classifying the Iris features. The Logistic regression model and the multi-layer perceptron (MLP) model yielded similar results in terms of performance and accuracy. It is important to note that model complexity is an important factor especially when a model is compared to a less complex model that yields similar or greater performance results. However, in the case of the iris dataset, given that the dataset is small, coupled with the fact that it does not have a large number of features, the model complexity becomes less of a deciding factor. Therefore, for this application, both the Logistic Regression and the MLP models are suitable for the Iris classification task.

# CONCLUSION

Our exploration into pattern recognition systems via k-means clustering, logistic regression, and neural networks on the Iris dataset has yielded insightful findings. The k-means algorithm, due to its unsupervised nature, showcased poor performance and accuracy, underscoring its limitations in tasks requiring precise classification without labeled data. On the other hand, both logistic regression and the multi-layer perceptron (MLP) neural network exhibited excellent performance in classification with remarkably similar accuracy scores.

Despite the simplicity and efficiency of logistic regression, the added complexity of the MLP coupled with the dataset characteristics, did not diminish its applicability. Given the modest size and feature count of the Iris dataset, deploying an MLP remains justified for this application.

WORKS CITED

|  |  |
| --- | --- |
| [1] | "scikit-learn," Google, [Online]. Available: https://scikit-learn.org/stable/. |
| [2] | Chris, "Stanford 2013," Stanford, 2012. [Online]. Available: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html. [Accessed 15 08 2024]. |
| [3] | codebasics, "Machine Learning Tutorial Python - 8: Logistic Regression (Binary Classification)," 7 9 2018. [Online]. Available: https://www.youtube.com/watch?v=zM4VZR0px8E. |
| [4] | LaValley, P Michael., "“Logistic regression”," *Circulation 117.18,* pp. 2395-2399, 2008. |
| [5] | kvssetty@gmail.com, "How To Plot A Decision Boundary For Machine Learning Algorithms in Python," 10th September 2020. [Online]. Available: https://hackernoon.com/how-to-plot-a-decision-boundary-for-machine-learning-algorithms-in-python-3o1n3w07. |
| [6] | T. I. Code, "youtube," 13 1 2021. [Online]. Available: https://www.youtube.com/watch?v=pauPCy\_s0Ok. [Accessed 2 8 2024]. |
| [7] | "Information and Technology Services," University of Michigan-Dearborn, [Online]. Available: https://umgpt.umich.edu/. |
| [8] | E. Garden, "youtube," 12 05 2022. [Online]. Available: https://www.youtube.com/watch?v=0QczhVg5HaI. [Accessed 16 08 2024]. |
| [9] | 3Blue1Brown, "youtube," 3 11 2017. [Online]. Available: https://www.youtube.com/watch?v=Ilg3gGewQ5U. [Accessed 15 8 2024]. |

# TABLE OF CONTENT

[INTRODUCTION 1](#_Toc175301514)

[BACKGROUND 2](#_Toc175301515)

[DATASET 2](#_Toc175301516)

[Data Preparation and Data Cleaning 2](#_Toc175301517)

[1. Introduction 2](#_Toc175301518)

[2. Data Loading 2](#_Toc175301519)

[3. Data Standardization 3](#_Toc175301520)

[4. Data Normalization 3](#_Toc175301521)

[5. Data Visualization 4](#_Toc175301522)

[6. Conclusion 4](#_Toc175301523)

[MACHINE LEARNING MODELS & METHODOLOGIES 5](#_Toc175301524)

[K-Means Classification 5](#_Toc175301525)

[A 5](#_Toc175301526)

[Mathematical Foundations of K-Means Clustering 5](#_Toc175301527)

[Library Implementation 6](#_Toc175301528)

[From-Scratch Implementation 8](#_Toc175301529)

[Comparison Between Library and from-Scratch Implementations 10](#_Toc175301530)

[Logistic Regression Classification 10](#_Toc175301531)

[Basic Concept 10](#_Toc175301532)

[Model Mechanisms 10](#_Toc175301533)

[One-Hot Encoding 11](#_Toc175301534)

[Softmax Function 11](#_Toc175301535)

[Cross-Entropy Loss Function 12](#_Toc175301536)

[Gradient Computation 12](#_Toc175301537)

[Model comparison (using library & from scratch) 13](#_Toc175301538)

[From Scratch 13](#_Toc175301539)

[Training Process 13](#_Toc175301540)

[Prediction Process 13](#_Toc175301541)

[Using Library (scikit-Learn) 14](#_Toc175301542)

[Training Process 14](#_Toc175301543)

[Prediction Process 14](#_Toc175301544)

[Results 14](#_Toc175301545)

[Precision/Recall/F1 14](#_Toc175301546)

[Cross-validation Techniques: K-fold cross-validation 15](#_Toc175301547)

[From Scratch 16](#_Toc175301548)

[Using Library 16](#_Toc175301549)

[Confusion Matrices Analysis 16](#_Toc175301550)

[From Scratch 17](#_Toc175301551)

[Using Library 18](#_Toc175301552)

[Decision Boundary 18](#_Toc175301553)

[Comparative Analysis of Library vs. Scratch Models 20](#_Toc175301554)

[General Performance 20](#_Toc175301555)

[Learning Speed and Efficiency: 20](#_Toc175301556)

[Utility and Application: 20](#_Toc175301557)

[Neural Network Classification 20](#_Toc175301558)

[About Neural Networks 20](#_Toc175301559)

[Neural Network Mechanics and Scratch Implementation 21](#_Toc175301560)

[Dense Layer and Forward Propagation: 21](#_Toc175301561)

[Backpropagation: 22](#_Toc175301562)

[Activation Functions: 23](#_Toc175301563)

[Loss Function: 24](#_Toc175301564)

[Implementation: 24](#_Toc175301565)

[Library Implementation 24](#_Toc175301566)

[Training and Implementation 25](#_Toc175301567)

[Neural Network Prediction Analysis and Discussion 26](#_Toc175301568)

[RESULTS & ANALYSIS 27](#_Toc175301569)

[Data Outcome and Comparison 27](#_Toc175301570)

[Observations and Assessment 28](#_Toc175301571)

[CONCLUSION 28](#_Toc175301572)

[TABLE OF CONTENT 30](#_Toc175301573)