CS235 Fall'23 Implementation Correctness Report: Use various Data Mining Techniques to Analyse and Predict the Smoking History of a User

VASANTHAKUMAR SHANMUGAVADIVEL #1, NetID: vshan008
RAMSUNDAR KONETY GOVINDARAJAN #2, NetID: rkone003
DEEPAKINDRESH NARAYANA GANDHI #3, NetID: dnara017
VENKATA RANGA SAI VISHAL MATCHA #4, NetID: vmatc002
SUDHANSHU NITIN GULHANE #5, NetID: sgulh001

Additional Key Words and Phrases: Neural Network, Gradient Boosting, Random Forest, Support Vector Machine, Smoking, Precision, Recall, f1-score, Binary Classification

ACM Reference Format:
Vasanthakumar Shanmugavadivel #1, Ramsundar Konety Govindarajan #2, Deepakindresh Narayana Gandhi #3, Venkata Ranga Sai Vishal Matcha #4, and Sudhanshu Nitin Gulhane #5. 2024. CS235 Fall'23 Implementation Correctness Report: Use various Data Mining Techniques to Analyse and Predict the Smoking History of a User. In . ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 GRADIENT BOOSTING CLASSIFIER IMPLEMENTATION BY VISHAL

ALGORITHM

The consolidated code for Gradient boosting classifier is available on **implementation** for comparison between Sci-Kit Learn and the from-scratch implementations.

One well-known machine learning algorithm that is included in the group of ensemble learning approaches is the Gradient Boosting Classifier. To construct a stronger and more reliable predictive model, ensemble methods integrate the predictions of many weak learners, which are models that perform better than random guessing but are not always extremely accurate. Gradient Boosting is a particular kind of ensemble learning technique that constructs trees one after the other, fixing each other's mistakes. Gradient Boosting Classifiers use decision trees as foundation models or weak learners; these are usually shallow trees known as "stumps." These are basic, shallow decision trees that are introduced to the ensemble one after the other.

The following are the actions taken in order to construct the model:

(1) The dataset is pre-processed by dropping irrelevant columns and cleaned.
(2) Initialization settings for the GradientBoosting class include the number of estimators (`n_estimators`), learning rate (`learning_rate`), and maximum decision tree depth (`max_depth`).
(3) Utilizing the mean of the target values, compute the initial prediction (`initial_prediction = np.mean(y)`).

Manuscript submitted to ACM

(4) By deducting the initial forecast from the actual target values, compute the residuals (`residuals = y - initial_prediction`).
(5) For each boosting round (iteration):
    - `tree = DecisionTree(max_depth=self.max_depth)`: Create a new decision tree.
    - Use the residuals (`tree.fit(X, residuals)`) to fit the decision tree to the training set.
    - Use `self.trees.append(tree)` to add the decision tree to the ensemble.
    - Update the residuals (`residuals -= self.learning_rate * tree.predict(X)`) by deducting the product of the learning rate and the predictions from the just inserted decision tree.
(6) Establish a decision tree's initial state with the chosen maximum depth and further properties.
(7) Then assigned the target variable's mean value to the current node (`self.value = np.mean(y)`) if all samples belong to the same class or if the maximum depth has been achieved.
(8) For every feature, set a threshold point:
    - Following the threshold split, compute a score based on the variance of the target variable.
    - Select the characteristic and threshold combination that yields the lowest score.
(9) Update the tree's features (`feature, threshold`) and fit the left and right subtrees iteratively if the optimal split lowers the score (impurity).
(10) Assign the target variable's mean value to the current node if the optimal split does not lower the score.

HYPER-PARAMETERS OF GRADIENT BOOSTING CLASSIFIER

- `n_estimators`
- `learning_rate`
- `max_depth`

BASELINE

The implementation of the gradient boosting classifier using the Scikit Learn library—short for sklearn—represents the baseline work. `GradientBoostingClassifier` is an inherent module available in sklearn.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

POTENTIAL DISCREPANCIES

(1) **S**cikit-learn is implemented in low-level languages like C and Cython, making computations faster than Python-based implementations. The library is performance-optimized, working well with large datasets.
(2) **T**o expedite the training process, Scikit-learn's implementation frequently utilizes parallelization and multiple CPUs, enhancing efficiency.
(3) **S**cikit-learn's default hyperparameter settings are an excellent starting point, often well-chosen. However, users can customize hyperparameters to explore model efficiency by experimenting with different values compared them in table Fig. 1 and plotted them in graphs as in Fig. 2.
(4) **S**cikit-learn is a robust and dependable library, subjected to rigorous testing and validation. The codebase is regularly updated, and new versions are released to fix bugs and enhance functionality.
(5) **N**umPy, pandas, and Matplotlib are prominent data science and machine learning libraries in the Python ecosystem that easily interface with Scikit-learn. This makes it easier to manipulate, analyze, and visualize data in a coordinated workflow.

n_estimators	GBclassifier_Scratch	GBClassifier_Sklearn
125	0.814587	0.814587
100	0.817663	0.81283
75	0.818541	0.814587
50	0.81942	0.817223

Fig. 1. Comparision for diffeent estimators

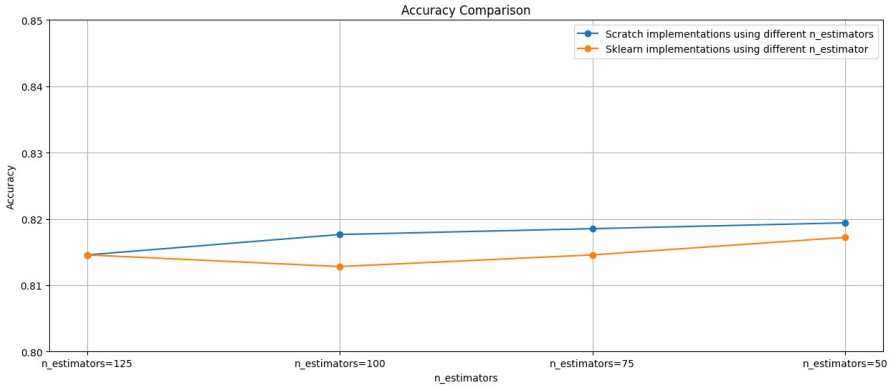

Fig. 2. tplotting of comparision

SANITIC CHECK EVALUATION

(1) T-statistic: The measures used for the sake of comparison are T-statistic.

$$t\text{-statistic} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

(2) Precision: Precision, sometimes referred to as positive predictive value, is a metric used to assess how well a classifier makes positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

(3) Recall: Recall, sometimes referred to as sensitivity or true positive rate, assesses a classifier's capacity to identify each and every pertinent instance in the dataset.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

(4) Accuracy: The ratio of accurately predicted occurrences—both true positives and true negatives—to the total number of instances is known as accuracy, which serves as a gauge of overall correctness.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

(5) F1-Score: The harmonic mean of recall and precision is known as the F1-score. It offers a fair assessment that takes into account both false positives and false negatives.

$$\text{F1-Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

EXPERIMENTAL ANALYSIS

Findings: The gradient boosting classifier's experimental results are presented in Fig. 3 using the T-statistic and P-value. In this case, the implementation from scratch and the default scikit-learn implementation are being compared. Additionally, Fig. 4 compares the k-fold accuracy of sklearn and scratch implementation.

Method	P-value	T-statistic
Xgb_Scratch	0.32	-0.99
Xgb_Sklearn	1.48E-08	-5.67
Ada_Scratch	0.147	-1.44
Ada_Sklearn	2.70E-19	-2.206
Random Forest Scratch	4.15E-14	-7.5681
Random Forest Sklearn	1.26E-17	-8.564
SVM Scratch	1.22E-17	-10.929
SVM Sklearn	3.05E-27	-10.844

Fig. 3. Experimental Results

2 MULTI LAYER PERCEPTRON (ARTIFICIAL NEURAL NETWORK) IMPLEMENTED BY SUDHANSHU GULHANE

ALGORITHM

The consolidated code for Artificial Neural Network is available on **Google Colab** for comparison between Sci-Kit Learn and the from-scratch implementations. The consolidated code for scratch implementation of Artificial Neural Network is available on **Google Colab** which I used for hyperparameter tuning.

An artificial feed forward neural network with 5 layers is implemented. The first layer of the architecture is the input layer (which contains the input features from the data set in the form of neurons) with size 23 neurons. The second, third and the fourth layers are the hidden layers with sizes 16, 8 and 4 neurons. The activation function used in the hidden layers is Relu. The fifth layer is the output layer which contains one neuron, and it is responsible to predict the binary output of either 0 or 1. The activation function used for the output layer is sigmoid function. A

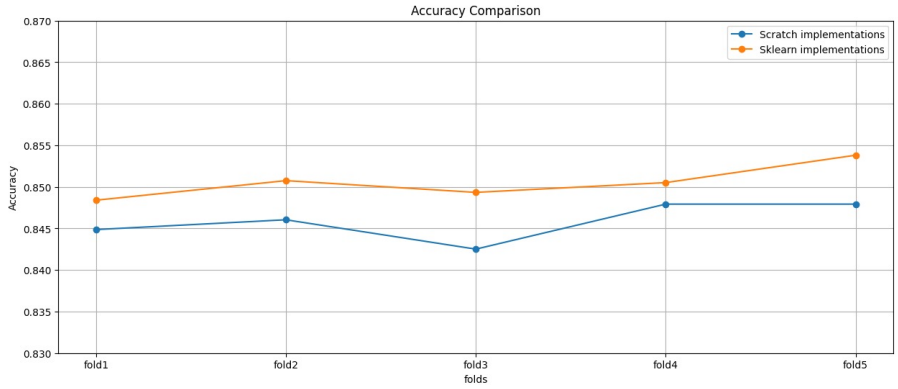

Fig. 4. Comparison of k-fold Accuracy

Training Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.86	0.85	0.84	0.84	0.86	0.85	0.85	0.85
2	0.86	0.85	0.84	0.85	0.86	0.85	0.85	0.85
3	0.85	0.84	0.84	0.84	0.86	0.85	0.85	0.85
4	0.86	0.85	0.85	0.85	0.86	0.85	0.85	0.85
5	0.86	0.85	0.85	0.85	0.86	0.85	0.85	0.85

Validation Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.85	0.84	0.84	0.84	0.84	0.83	0.83	0.84
2	0.82	0.81	0.81	0.81	0.82	0.81	0.81	0.81
3	0.86	0.85	0.85	0.85	0.85	0.84	0.84	0.84
4	0.85	0.83	0.83	0.84	0.84	0.83	0.83	0.83
5	0.82	0.81	0.81	0.81	0.82	0.81	0.81	0.81

Testing Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.83	0.82	0.81	0.82	0.83	0.82	0.81	0.82
2	0.83	0.81	0.81	0.82	0.83	0.82	0.82	0.82
3	0.83	0.82	0.82	0.82	0.83	0.82	0.82	0.82
4	0.83	0.82	0.82	0.82	0.82	0.82	0.81	0.82
5	0.83	0.82	0.82	0.82	0.83	0.82	0.82	0.82

Fig. 5. tabular representation of cross validation (Gradient Boosting Classifier)

kernel_ regularizer parameter that uses L2 regularization penalty, is used to apply regularization to the weights of the dense layer, with the regularization strength equal to 0.001. The algorithm uses batch size of 200, and Adam optimizer

with a learning rate of 0.0002. The loss function used for training purpose is binary cross entropy suitable for binary classification. The architecture of the model is shown in Fig.5. The training loss v/s validation loss plot for the proposed Neural Network architecture is shown in Fig.8.

BASELINE

The sklearn class sklearn.neural_network.MLPClassifier is used to train the Multilayer Perceptron Binary classifier. It has the same architecture as the one implemented from scratch with 3 hidden layers with sizes 16, 8, 4 respectively, learning rate (learning_rate_init) as 0.0002, and batch size set to "auto".

HYPERPARAMETER TUNING/EXPERIMENTAL ANALYSIS

Fig.6. and Fig.7. shows the results of the Hyperparameter tuning for training and testing data sets in a tabular format. Although the model is robust to the changes in the Hyperparameter values, the learning rate value when chosen 0.0002 not only gives comparatively lower training and validation loss but also gives a smooth plot (without distortion) for training vs validation loss and training vs validation accuracy indicating training stability and convergence to a stable solution.

SANITY CHECK EVALUATION USING K-FOLD CROSS VALIDATION

Cross-validation is a statistical technique employed to assess and compare learning algorithms. It involves partitioning the dataset into two segments: one for model training and learning, and the other for model validation. In this section k-fold cross-validation is discussed, specifically with k=5, comparing the performance of a proposed neural network model and a neural network model implemented using the scikit-learn library. Through the evaluation as seen in the table Fig.10. of **precision**, **f1-score**, **recall**, and **accuracy** across the **five** folds, there are minimal discrepancies in their respective values. Furthermore, we graphically depicted the testing accuracy outcomes for both the custom-implemented and scikit-learn implemented neural network models. The visual representation substantiates the similarity in performance between the two models. Based on these findings, it can be asserted that the proposed neural network model closely aligns with the neural network model utilizing the scikit-learn library, particularly concerning the parameters detailed in the accompanying table.

3 SUPPORT VECTOR MACHINE IMPLEMENTED BY RAMSUNDAR KONETY GOVINDARAJAN

ALGORITHM

The consolidated code for Linear SVM is available on **implementation** for comparison between Sci-Kit Learn and the from-scratch implementations.

A support vector machine (SVM) is a machine learning technique that is used for performing regression and classification operations. Linear SVM is a type of supervised machine learning for classification operations. It is used to find the best possible hyperplane that separates different classes in the input feature space. It is mainly used for binary classification for large datasets because it can handle the computations without a significant increase in computational cost compared to other algorithms. In machine learning, Stochastic Gradient Descent (SGD) is an iterative optimization process that finds the minimum of a function, specifically for reducing a model's loss function. Because our dataset is huge, I use stochastic gradient descent to optimize machine learning models because the computing cost per iteration is substantially lower when compared to traditional Gradient Descent methods that involve processing the full dataset.

Sr No.	Learning Rate	L2 Regularization	Precision	Recall	f1-score	Accuracy
1	0.0001	0.0001	0.84	0.83	0.83	0.83
2	0.0002	0.001	0.84	0.83	0.83	0.83
3	0.0004	0.001	0.84	0.83	0.83	0.83
4	0.0008	0.001	0.84	0.83	0.83	0.83
5	0.001	0.001	0.84	0.83	0.83	0.829
6	0.002	0.001	0.84	0.83	0.83	0.829
7	0.004	0.001	0.84	0.83	0.83	0.829

Sr No.	Training_loss	Validation_loss
1	0.413	0.409
2	0.4099	0.4071
3	0.4099	0.4074
4	0.4105	0.4076
5	0.4096	0.4065
6	0.4119	0.4098
7	0.4148	0.4109

Fig. 6. Hyperparameter tuning (Neural network) results on training set

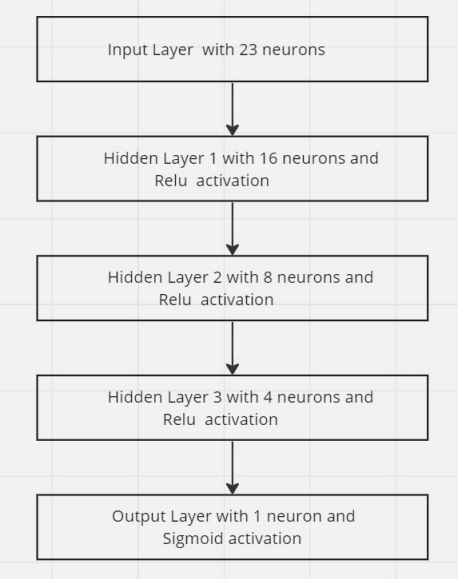

Fig. 7. Architecture of the Neural network model

Sklearn's Linear SVC provides numerous parameters for customizing the classifier's performance. The parameters include the following: multiclass, max iter (maximum iterations), loss function, C (regularization parameter), and tol (tolerance). When using the default value, the accuracy of the sklearn's linearSVC is approximately 0.83. I tested with different parameters for the implementation of LinearSVM from scratch, however the accuracy was quite close when

Sr No.	Learning Rate	L2 Regularization strength	Precision	Recall	f1-score	Accuracy
1	0.0001	0.0001	0.84	0.83	0.83	0.829
2	0.0002	0.001	0.84	0.83	0.83	0.829
3	0.0004	0.001	0.84	0.83	0.83	0.829
4	0.0008	0.001	0.84	0.83	0.83	0.829
5	0.001	0.001	0.84	0.83	0.83	0.829
6	0.002	0.001	0.84	0.83	0.83	0.829
7	0.004	0.001	0.84	0.83	0.83	0.829

Fig. 8. Hyperparameter tuning (Neural network) results on testing set

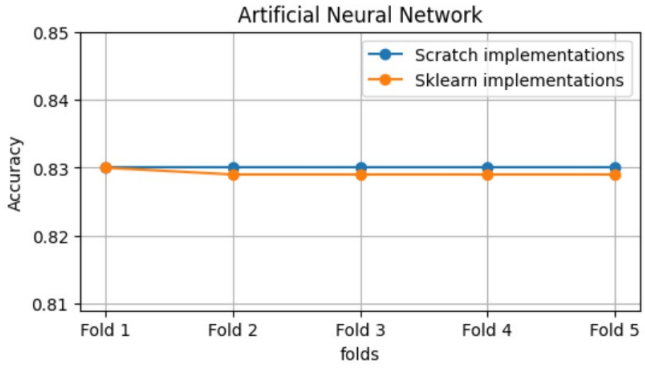

Fig. 9. Testing accuracy comparison for the sklearn and proposed Artificial Neural Network

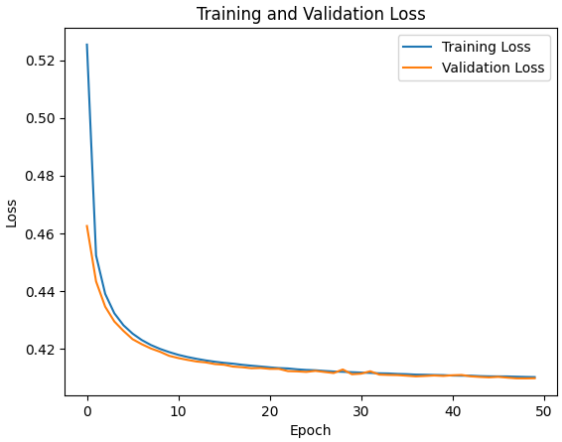

Fig. 10. Loss plot of the proposed Neural Network

compared to the sklearns implementation when using three parameters (Learning rate, Regularization parameter, Number of iterations).

The steps taken to create the linear SVM from scratch are as follows:

(1) The dataset is cleaned and processed using data preprocessing techniques.

Training Set Comparision

Folds	Sklearn Implementation				Scratch Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.83	0.83	0.84	0.83	0.83	0.83
2	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83
3	0.84	0.83	0.83	0.83	0.84	0.83	0.83	0.83
4	0.84	0.83	0.83	0.831	0.84	0.83	0.83	0.831
5	0.84	0.83	0.83	0.83	0.84	0.83	0.83	0.83

Validation Set Comparision

Folds	Sklearn Implementation				Scratch Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.83	0.831	0.84	0.83	0.83	0.831
2	0.84	0.83	0.83	0.831	0.84	0.83	0.83	0.83
3	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83
4	0.84	0.83	0.83	0.826	0.84	0.83	0.83	0.827
5	0.84	0.83	0.83	0.831	0.84	0.83	0.83	0.832

Testing Set Comparision

Folds	Sklearn Implementation				Scratch Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.83	0.83	0.84	0.83	0.83	0.83
2	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83
3	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83
4	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83
5	0.84	0.83	0.83	0.829	0.84	0.83	0.83	0.83

Fig. 11. Cross validation (Neural Network) results

(2) At first, A function SVM is defined to take training data (`X_train, y_train`), test data and optional parameters for learning rate(lr), regularization strength (`lamba_parameter`) and number of iterations.

(3) W(weight) and B(bias) are initialized. Weight corresponds to the coefficient of the features and Bias helps in shifting the decision boundary from the origin. At the initial stage, weight and bias is set to zero.

(4) Y is a modified version of (`y_train`) where all the elements on the positive labels are set to 1 and non-positive labels are set to -1.

(5) An iterative optimization method is used to update the weights and bias in each iteration by evaluating one or a few training samples at a time. The function iterates for a specified number of iterations (iterations), and within each iteration, it loops through each training example (`x_i`) along with its corresponding label (y[id]).

(6) A condition = y[id] * (np.dot(`x_i`, w) - b) >= 1 is the hinge loss function to maximize the margin between classes while minimizing classification errors and it is checking whether a particular data point (`x_i`) is correctly classified or if it violates the margin constraint.
- If the condition is met, w is updated to decrease its magnitude, which is a part of the regularization to avoid overfitting.
- If the condition is not met, both w and b are updated to correctly classify that example. This involves moving the decision boundary.

(7) After training SVM, the function determines the values of the decision function for the test and training sets of data. For training data, (`approx_train`) is calculated by removing the bias term from the dot product of the training features and the weight vector (np.dot(X, w) – b). The sign of this decision function is used as the predicted class label ((`y_pred_train`) for training data and (`y_pred`) for test data). Finally, The function returns the predictions on the training data and the test data.

HYPER-PARAMETERS OF LINEAR SVM

- `Learning rate`
- `Regularization Strength`
- `Number of Iterations`

BASELINE

The baseline library used is the Sci-Kit Learn library for Support Vector Machine.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

POTENTIAL DISCREPANCIES

(1) For optimization, the implementation from scratch of linearSVM uses a basic approach as Stochastic Gradient Descent (SGD). On the other hand, depending on the kernel and settings, sklearn's Linear SVM employs more sophisticated optimization methods like Sequential Minimal Optimization (SMO), Coordinate Descent(CD) Method, Linear kernel optimization etc. In general, more sophisticated algorithms are more successful with real time datasets and efficient at identifying the optimal solution.

(2) Sklearn's Linear SVC provides numerous parameters for customizing the classifier's performance. The parameters include the following: multiclass, (`max_iter`) (maximum iterations), loss function, C (regularization parameter), and tol (tolerance). When using the default value, the accuracy of the sklearn's linearSVC is approximately 0.83. I tested with different parameters for the implementation of LinearSVM from scratch, however the accuracy was quite close when compared to the sklearns implementation when using three parameters (Learning rate, Regularization parameter, Number of iterations).

(3) The loss function in the implementation from scratch is limited to a hinge loss with L2 regularization. LinearSVC from Scikit-learn allows users to choose between L1 and L2 regularization and utilizes hinge loss by default, which smoothes decision boundaries and provides robustness against outliers.

SANITIC CHECK EVALUATION

(1) T-statistic: The measures used for the sake of comparison are T-statistic.

$$t\text{-statistic} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

(2) Precision: Precision, sometimes referred to as positive predictive value, is a metric used to assess how well a classifier makes positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

(3) Recall: Recall, sometimes referred to as sensitivity or true positive rate, assesses a classifier's capacity to identify each and every pertinent instance in the dataset.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

(4) Accuracy: The ratio of accurately predicted occurrences—both true positives and true negatives—to the total number of instances is known as accuracy, which serves as a gauge of overall correctness.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

(5) F1-Score: The harmonic mean of recall and precision is known as the F1-score. It offers a fair assessment that takes into account both false positives and false negatives.

$$\text{F1-Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

Training Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.82	0.82	0.822	0.84	0.82	0.82	0.822
2	0.84	0.83	0.83	0.822	0.84	0.83	0.82	0.821
3	0.84	0.83	0.83	0.824	0.84	0.82	0.82	0.823
4	0.84	0.83	0.82	0.826	0.84	0.82	0.82	0.825
5	0.83	0.82	0.82	0.819	0.84	0.82	0.82	0.819

Validation Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.82	0.822	0.84	0.83	0.82	0.823
2	0.84	0.83	0.82	0.826	0.84	0.83	0.82	0.825
3	0.83	0.82	0.82	0.819	0.83	0.82	0.82	0.819
4	0.82	0.82	0.81	0.809	0.83	0.82	0.81	0.809
5	0.84	0.84	0.84	0.837	0.85	0.84	0.83	0.835

Testing Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.82	0.822	0.84	0.82	0.82	0.822
2	0.84	0.82	0.82	0.822	0.84	0.82	0.82	0.822
3	0.84	0.82	0.82	0.822	0.84	0.82	0.82	0.822
4	0.84	0.82	0.82	0.822	0.84	0.83	0.82	0.822
5	0.84	0.82	0.82	0.822	0.84	0.82	0.82	0.822

Fig. 12. Cross Validation(Linear SVM) results

EXPERIMENTAL ANALYSIS

Findings: Cross-validation is a statistical technique employed to assess and compare learning algorithms. It involves partitioning the dataset into two segments: one for model training and learning, and the other for model validation. In this section K-fold cross-validation is discussed, specifically with k=5, comparing the performance of a proposed Linear SVM and Linear SVM implemented using the scikit-learn library. Through the evaluation as seen in the table Fig.12. of precision, f1-score, recall, and accuracy across the five folds, there are minimal discrepancies in their respective values.The graphic representation substantiates the two models' performance similarities.Based on these data, it can be said that the proposed Linear SVM model matches very closely with the Sklearns Linear SVM based on the scikit-learn package, particularly in terms of the parameters shown in the table. Both implementations have a comparable level of model complexity, which is influenced by the regularization strength. Model complexity is similar if your regularization term finds a reasonable balance between fitting the data and avoiding overfitting. The three hyperparameters that were selected are the main reasons for very close values in all cross-validation tests between the proposed approach and sklearns implementation.The model's convergence and performance depend significantly on the chosen set of hyperparameters, which include the learning rate (lr), number of iterations (iterations), and regularization strength (`lamba_parameter`) of my proposed model. Thus, the implementation of Linear SVM from scratch achieves comparable accuracy to scikit-learn's LinearSVC with k-fold cross-validation, showcasing the effectiveness of your algorithm. The chosen hyperparameters, including learning rate and regularization strength is well tuned and processed with our dataset, contributing to the model's robust performance.

4 RANDOM FOREST BY DEEPAKINDRESH NARAYANA GANDHI

ALGORITHM

The consolidated code for Random forest classifier is available on **implementation** for comparison between Sci-Kit Learn and the from-scratch implementations.

The Random Forest algorithm is a bagging-type ensemble learning method that operates by constructing multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Below are the steps of execution for the Random Forest algorithm from scratch:

The following are the actions taken to construct the model:

(1) Define the number of trees (num_trees) in the forest.
(2) Specify hyperparameters such as maximum depth (max_depth), minimum samples required to split a node (min_split), and the number of features considered for each split (num_features).
(3) For each tree in the forest sample with replacement from the training dataset using bootstrapping to create a random subset (bagging).
(4) Create a decision tree using the bootstrapped dataset.
(5) At each node of the tree, randomly select a subset of features from the total features to consider for the split. This introduces diversity among the trees.
(6) Recursively split nodes based on the best split determined by information gain or another criterion.
(7) Repeat until a stopping criterion is met (e.g., maximum depth reached, minimum samples for a split not met).
(8) For each data point to be predicted aggregate predictions from each tree in the forest and use majority voting for classification

HYPER-PARAMETERS OF FROM SCRATCH IMPLEMENTATION OF RANDOM FOREST

- `num_trees`
- `max_depth`
- `min_split`

BASELINE

The implementation of the Random Forest classifier using the Scikit Learn library—short for sklearn—represents the baseline work. `RandomForestClassifier` is an inherent module available in sklearn.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

POTENTIAL DISCREPANCIES

(1) The scikit-learn implementation is highly optimized and written in a combination of Python and low-level languages (such as Cython and C). This can lead to better performance and faster execution times compared to a pure Python implementation.
(2) Scikit-learn's Random Forest can take advantage of parallel processing to speed up the training process. It parallelizes the training of individual trees, making it more efficient on multi-core machines.
(3) The scikit-learn implementation is designed to be memory-efficient, which is crucial when dealing with large datasets. It uses techniques such as data structures and algorithms that minimize memory usage.
(4) Scikit-learn provides built-in methods to assess feature importance after training a Random Forest model. This can be useful for understanding which features contribute most to the model's predictions.
(5) Scikit-learn allows for easy configuration of hyperparameters and provides additional options for controlling the Random Forest's behavior. It also includes features like out-of-bag (OOB) score estimation.

SANITIC CHECK EVALUATION

(1) T-statistic: The measures used for the sake of comparison are T-statistic.

$$t\text{-statistic} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

(2) Precision: Precision, sometimes referred to as positive predictive value, is a metric used to assess how well a classifier makes positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

(3) Recall: Recall, sometimes referred to as sensitivity or true positive rate, assesses a classifier's capacity to identify each and every pertinent instance in the dataset.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

(4) Accuracy: The ratio of accurately predicted occurrences—both true positives and true negatives—to the total number of instances is known as accuracy, which serves as a gauge of overall correctness.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

(5) F1-Score: The harmonic mean of recall and precision is known as the F1-score. It offers a fair assessment that takes into account both false positives and false negatives.

$$\text{F1-Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

Training Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.88	0.87	0.828	0.867	1	1	1	1
2	0.87	0.86	0.86	0.862	1	1	1	1
3	0.87	0.86	0.86	0.857	1	1	1	1
4	0.87	0.86	0.86	0.861	1	1	1	1
5	0.88	0.87	0.87	0.867	1	1	1	1

Validation Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.82	0.81	0.81	0.807	0.83	0.82	0.83	0.816
2	0.82	0.81	0.81	0.815	0.82	0.81	0.81	0.811
3	0.82	0.81	0.81	0.812	0.83	0.82	0.82	0.818
4	0.84	0.83	0.83	0.832	0.84	0.83	0.83	0.833
5	0.82	0.81	0.81	0.81	0.82	0.81	0.81	0.81

Testing Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.84	0.83	0.83	0.828	0.84	0.83	0.83	0.834
2	0.84	0.83	0.83	0.826	0.84	0.83	0.83	0.831
3	0.84	0.83	0.83	0.829	0.83	0.82	0.82	0.832
4	0.83	0.82	0.82	0.832	0.82	0.82	0.81	0.832
5	0.83	0.82	0.82	0.829	0.83	0.82	0.82	0.83

Fig. 13. Comparision for different estimators (Random Forest)

EXPERIMENTAL ANALYSIS

Findings: Upon running both Sklearn models and the from-scratch implementation of Random Forest we can see subtle differences in accuracy and run time of the model. Sklearn's Random Forest has been seen to perform better in terms of accuracy, f1 score, precision, and recall but it is also found to overfit if we you look at the Fig 13, we can find that in Training accuracy the Sklearn's Random Forest has got an accuracy of 100% and this gives a clear sign that it is overfitting even though Random Forest's are supposed to solve overfitting.

Thus the from-scratch implementation could have been generalized better since it does not fall far behind the Sklearn's implementation and uses a lot less parameters for example Sklearn uses a lot more parameters like min_samples_leaf,

min_impurity_decrease and a lot more parameters which were never created for the from scratch model and yet it was able to give comparable results even though the model is a nondeterministic model using Random values.

5 ADABOOST CLASSIFIER IMPLEMENTATION BY VASANTHAKUMAR

ALGORITHM

The consolidated code for adaboost classifier is available on **implementation** for comparison between Sci-Kit Learn and the from-scratch implementations.

Adaptive Boosting, or AdaBoost, is a statistical classification meta-algorithm that can be used to enhance performance when combined with a variety of different learning algorithm types. The final output of the boosted classifier is represented by a weighted sum that is created by combining the output of the other learning algorithms, or "weak learners." AdaBoost is typically described for binary classification, while it can be extended to bounded intervals on the real line or multiple classes. AdaBoost is adaptive in that it adjusts weaker learners in the future to take into account cases when prior classifiers misclassified the data.

Decision stumps and other weak base learners are commonly combined using AdaBoost. A one-level decision tree makes up a decision stump. Put otherwise, this decision tree has a single internal node (the root) that is directly connected to all of the terminal nodes (its leaves). A forecast made by a decision stump is dependent only on the value of one input characteristic. AdaBoost's individual learners may be weak, but the entire model will converge to a strong learner as long as each learner's performance is marginally better than chance.

(1) Data Loading and Preprocessing, Under-sampling for Imbalanced Data, Dimensionality Reduction and Feature Engineering
(2) Define stump: A decision stump is a tree with depth 1. This stump serves as weak learners for our AdaBoost classifier.
(3) Weights: Each learner is assigned a weight that indicates trust in the learner. The higher the weight, the more it contributes to the final output. f = α1H1 + α2H2 + α3H3 +.. where H stands for the weak learners and denotes the weights.
(4) Dependency: AdaBoost depends on the previous stumps to make it better at each stage by transmitting the errors of the previous stump. This enables the classifier to progress towards lower bias.
(5) Initialize: Initialize weights. wi = C , i = 1,2,..N. I have chosen 1/N as my constant.
(6) Fit: In the fit method, we first create a weight vector that initializes all the weights and to fill in equal weights.
    - Fit classifier Hm with weights w
    - Compute err = SUM(wi*I(yi!=Hm(xi)) / SUM(wi)
    - Compute α = log((1-err)/err)
    - Update the weights wi = wi * exp(*I(yi!=Hm(xi))
(7) Predict: SUM (α*Hm(x)). The polarity parameter controls whether we want our 'Yes'/'No' classifications to be 1 or -1. The threshold is used to split and to make the stump on that feature. We find the best decision stump based on minimum error. We multiply alpha with the weak learners for all stumps, and the final sign output that we get is the final prediction.
(8) We loop over the unique thresholds, and calculate the error which is just the sum of all weights of misclassified samples. If the error is greater than 0.5, we flip both the polarity and the error. We then compare it with min error and update accordingly. We repeat this for all features.

HYPER-PARAMETERS OF ADABOOSTING CLASSIFIER

- `n_estimators`
- `learning_rate`

BASELINE

The implementation of the AdaBoosting classifier using the Scikit Learn represents the baseline work. `AdaBoostingClassifier` is an inherent module available in sklearn.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

SANITIC CHECK EVALUATION

(1) T-statistic: The measures used for the sake of comparison are T-statistic.

$$t\text{-statistic} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

(2) Precision: Precision, sometimes referred to as positive predictive value, is a metric used to assess how well a classifier makes positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

(3) Recall: Recall, sometimes referred to as sensitivity or true positive rate, assesses a classifier's capacity to identify each and every pertinent instance in the dataset.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

(4) Accuracy: The ratio of accurately predicted occurrences—both true positives and true negatives—to the total number of instances is known as accuracy, which serves as a gauge of overall correctness.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

(5) F1-Score: The harmonic mean of recall and precision is known as the F1-score. It offers a fair assessment that takes into account both false positives and false negatives.

$$\text{F1-Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

EXPERIMENTAL ANALYSIS

Findings: The gradient boosting classifier's experimental results are presented in Fig. 14 and 15 using the T-statistic and P-value. In this case, the implementation from scratch and the default scikit-learn implementation are being compared. Additionally, Fig. 14 compares the k-fold accuracy of sklearn and scratch implementation.

Training Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.75	0.75	0.74	0.746	0.78	0.78	0.78	0.777
2	0.75	0.75	0.74	0.744	0.77	0.77	0.77	0.772
3	0.75	0.75	0.75	0.746	0.77	0.77	0.77	0.772
4	0.75	0.75	0.75	0.752	0.78	0.78	0.78	0.78
5	0.74	0.74	0.74	0.743	0.75	0.75	0.75	0.754

Validation Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.75	0.74	0.74	0.745	0.75	0.75	0.75	0.746
2	0.76	0.75	0.75	0.752	0.76	0.75	0.75	0.755
3	0.75	0.74	0.74	0.742	0.75	0.75	0.75	0.752
4	0.72	0.72	0.72	0.721	0.72	0.72	0.72	0.722
5	0.75	0.75	0.75	0.754	0.77	0.77	0.77	0.771

Testing Set Comparision

Folds	Scratch Implementation				Sklearn Implementation			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.75	0.74	0.74	0.746	0.76	0.76	0.76	0.761
2	0.75	0.74	0.74	0.746	0.77	0.76	0.76	0.764
3	0.75	0.74	0.74	0.743	0.77	0.77	0.77	0.766
4	0.74	0.74	0.74	0.738	0.76	0.76	0.76	0.76
5	0.74	0.74	0.74	0.738	0.77	0.77	0.76	0.765

Fig. 14. Cross validation (adaboost Classifier) results

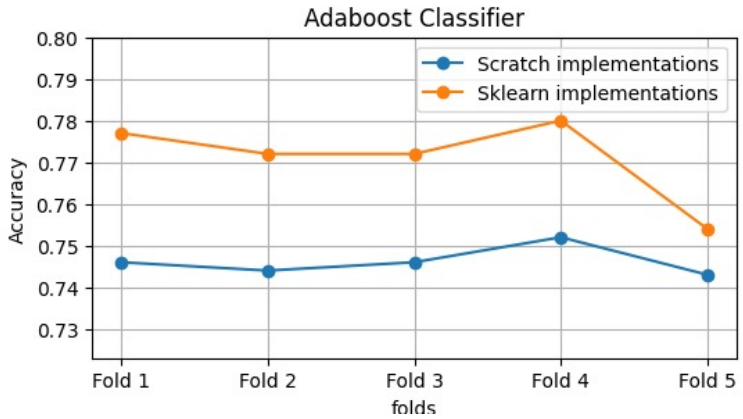

Fig. 15. line graph comparision (adaboost Classifier) results