PROJECT: - 1 BINARY CLASSIFICATION NAME: - JAYNAM PANCHAL
SUBJECT: - (CSC 272) MACHINE LEARNINNG

ABSTRACT

This report explores the effectiveness of various machine learning algorithms in predicting stroke and hiring decisions. We utilized two datasets: **Dataset 1**, which includes features such as patient age, gender, work type, residence type, average glucose level, BMI, and smoking status; and **Dataset 2**, which encompasses gender, education level, years of experience, previous companies, interview score, and recruitment strategy. Several machine learning algorithms were applied, including decision trees, K-nearest neighbors (KNN), perceptron, and logistic regression, to evaluate their performance on these datasets. To optimize model performance, we fine-tuned hyperparameters and assessed the models using metrics such as accuracy, F1-score, and recall. Our findings indicate that the perceptron algorithm achieved the highest accuracy of 92.3% on Dataset 1, decision tree algorithm achieved the highest accuracy of 90.6% on Dataset 2 demonstrating its effectiveness. These results underscore the importance of selecting the appropriate algorithm for a specific dataset, particularly when accuracy is paramount.

DATASETS

1. **Dataset 1** consists of 5110 entries and include various features relevant to predicting the occurrence of stroke. The dataset is mixture of categorical variable, numerical variable, binary variable.
   1.1. Dataset preprocessing
      1.1.1. To prepare the dataset for analysis, serval preprocessing step were performed by us. Encoding categorical variables in that we convert categorical variables into numerical. We firstly convert gender categorical into Female = 0, Male = 1, ever married into No = 0, Yes = 1, Residence Type into Rural=0, Urban = 1, Smoking status into never smoked =0, Formerly smoked = 1, smokes = 2, Work type into never worked = 0, children = 1, government job =2, self-employed =3, private = 4.
      1.1.2. We used median value to fill the missing value in numerical columns. Categorical columns were filled with the mode most frequent value. Standardization numerical features such as Age, Average Glucose Level, BMI were standardized using standard scaler. This transformation ensure that the features had mean of 0 and a deviation of 1, which crucial for many machine learning algorithms. The features were defined by dropping the id and stroke columns, while the target variable was set as stroke column. Data splitting for training and testing using 80-20 split to make sure the model trained and evaluated effectively.

2. **Datasets 2** consists of 1500 entries and includes various features relevant to predicting hiring decision. The dataset contains following attributes like Age, Gender, Education level, year of experience, previous companies, distance from company, interview score, skill score, personality score, recruitment strategy, hiring decision.
    2.1.1. Data preprocessing
    2.1.2. Firstly, we handling missing values in numerical columns were filled with median value. Numerical features including distance from company, interview score, skil score, and personality score, were standardized using standard Scaler. This process ensure that the features have a mean of 0 and a standard deviation of 1. Which is essential for many machine learning algorithms.
    2.1.3. Features were defined by dropping the hiring Decision column and set the target variable as hiring decision column.
    2.1.4. The dataset was split into training and testing sets using an 80-20 split ensure to model work effectively on dataset.

Methods

We began by splitting the dataset into 80% for training and 20% for testing using the train_test_split function. This approach allows us to train our models on the majority of the data while keeping a portion aside to test how well they perform on unseen data. We implemented four different algorithms: Decision Tree, K-Nearest Neighbors (KNN), Perceptron, and Logistic Regression. For each model, we trained it on the training set and evaluated it using the testing set. To ensure robust performance, we used 5-fold cross-validation, where we divided the training data into several parts (using different values for k for each algorithm) and validated each model's performance on various subsets. This helps to minimize the risk of overfitting.

Next, we fine-tuned the hyperparameters for each algorithm using GridSearchCV. This technique involved searching for the best parameters for each model, such as the maximum depth for the Decision Tree, the number of neighbors for KNN, the learning rate for the Perceptron, and the inverse of the regularization strength for Logistic Regression. We then selected the best-performing model based on its validation performance.

To evaluate our models, we looked at several metrics. We considered accuracy for overall performance, along with precision and recall to see how well the models identified positive cases. The F1-score was also used to strike a balance between precision and recall. Additionally, we created confusion matrices for each model to visualize the counts of true positives, true negatives, false positives, and false negatives. This visualization gives us valuable insights into where the models are making mistakes, helping us understand their strengths and weaknesses.

Finally, we plotted validation curves for each algorithm to see how model complexity like the maximum depth in the Decision Tree or the number of neighbors in KNN which affects performance. This analysis helped us choose the best parameters for each algorithm, ensuring we

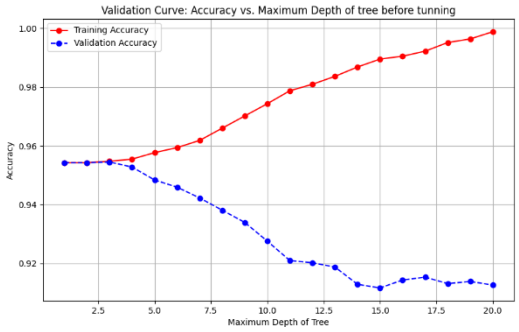
find the right balance between complexity and accuracy to avoid issues like overfitting or underfitting

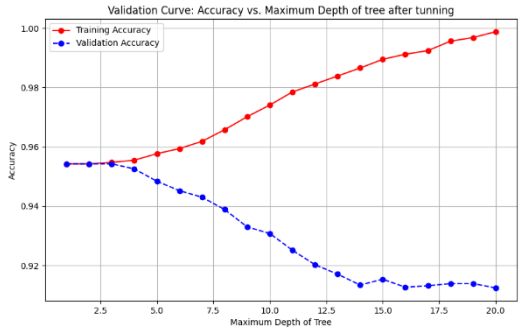
Hyperparameter Tuning

Methods

For each algorithm and dataset, I generated validation curves and confusion matrix both before and after hyperparameter tuning. Additionally, I performed 5-fold cross-validation on the training datasets for each validation curve.

i. Decision tree

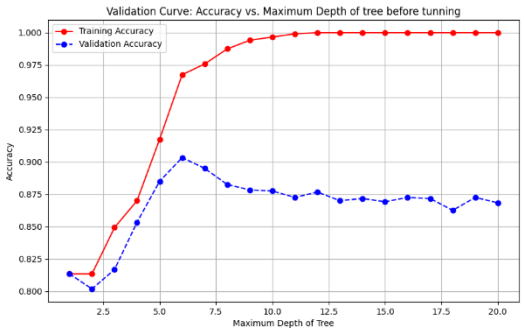


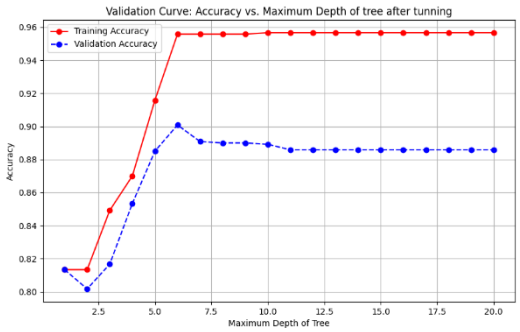


This graph before and after hyperparameter tunning for dataset 1 which shows slightly changes. The training accuracy increases consistently with tree depth, indicating that deeper trees can better fit the training data. The validation Accuracy Initially rises with depth but eventually decreases, suggesting that while deeper trees capture more complex patterns, they also risk overfitting. An optimal maximum depth was identified (approximately 10-15), where validation accuracy peaked before declining. Which suggests a need to limit tree depth to enhance model generalization. Through confusion matrix we observed that before hyperparameter tunning the model demonstrates exceptional performance, achieving high precision and recall, effectively identifying positive cases with minimal false positives and negatives. While still strong, the second confusion matrix indicates a decline in performance, particularly in recall, with an increase in false negatives compared to the first confusion matrix.

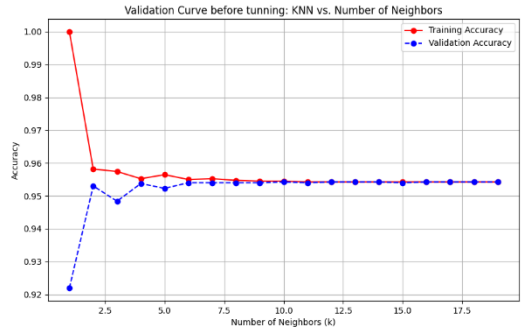
The graphs below for Dataset 2, illustrating that training accuracy sharply increases with the maximum depth of the tree, approaching nearly 100% when the depth reaches 7.5. This indicates that the model is effectively fitting the training data. Initially, validation accuracy also rises, but after a tree depth of 5, it begins to fluctuate and does not show significant improvement, suggesting overfitting. The gap between training and validation accuracy narrows, especially after a depth of about 5, indicating that the tuning process has successfully reduced overfitting, allowing the model to generalize more effectively to unseen data. Prior to tuning, the model displayed considerable overfitting, characterized by high training accuracy and lower validation

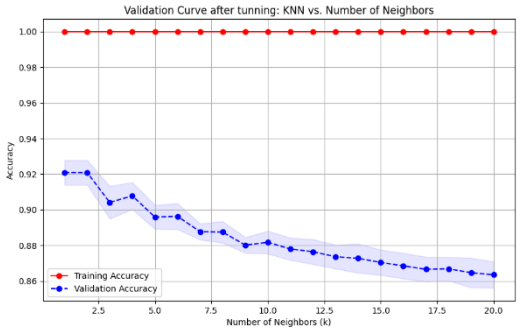
accuracy. Post-tuning, there is a more balanced relationship between the two. The optimal maximum depth appears to be around 5 in both scenarios, but the tuned model maintains higher validation accuracy beyond this point, demonstrating improved performance due to tuning. The increase in validation accuracy after tuning indicates that adjustments to the model's parameters have enhanced its ability to generalize to new data. The confusion matrix reveals that both models exhibit high accuracy, with the tuned model performing slightly better 88.6% before tuning and 89.5% after tuning which indicating improvements in precision and recall as well.





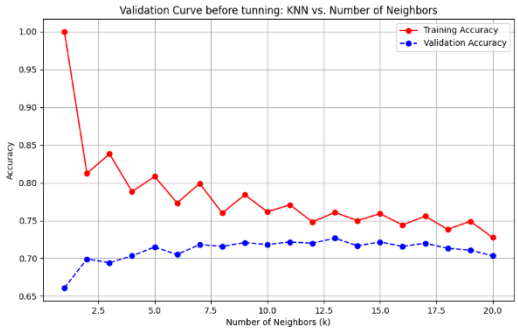
ii. k-nearest neighbors (KNN)

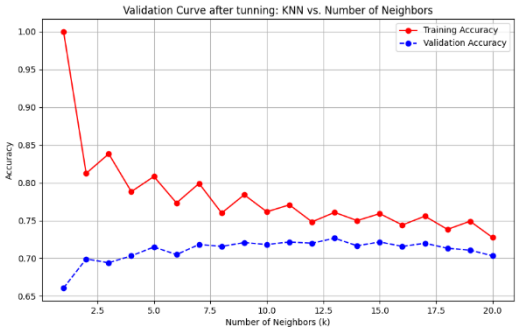




The above graphs of dataset 1 are represent the training accuracy is very high for small number of k (1 to 3) which indicating that the model is fitting training data very well. The validation accuracy starts high but drop sharply after k between 2.5 to 5.0. after initial drop the validation accuracy stabilized around 95% for larger value of k which indicate that model is less sensitive for training data and it's started to generalize better. As k increase, the model the model performance on validation set improve slightly, suggesting that it begins to generalize better but training accuracy remains high. After tunning hyperparameter tunning the training accuracy remains high for all k values which indicate the model continue to fit the training data well but it indicating a persistent risk of overfitting which indicate that the tuning process has not effectively reduced overfitting. The validation accuracy shows a clear drop to 86% as k increase. This indicates that as more neighbors are considered, the model's

ability to generalize to unseen data decreases, leading to underfitting. The gap between training accuracy and validation accuracy which shows that overfitting. Through confusion matrix before tunning, it has much higher precision but lower recall due to significant number r of false negative and no true negatives. This indicates that while it effectively identifies positive cases, but it fails to classify negative cases. After tunning confusion matrix shows that it has lower precision and has better recall ability to identify true negatives indicates that a more balance model.



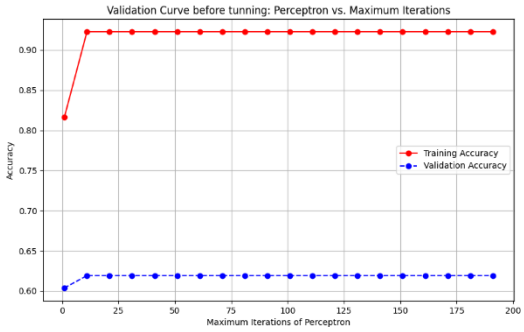


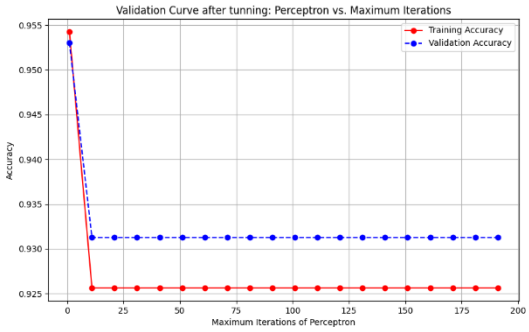
The above graphs of dataset 2. Before tunning the hyperparameter the training accuracy starts from very high for low value of K. The larger gap between validation accuracy and training accuracy which indicates that overfitting. The low validation accuracy indicates that the model is not effectively generalizing. After tunning, the training accuracy remains high, which indicates that the model still fits and training data well but validation accuracy shows improvement compared to before tunning. While it stabilize around 0.7 to 0.75 the drop in performance is less steep and more consistent across k values. The tunning involved selecting a more optimal range of k that balance than before tunning. While training accuracy remains high, validation accuracy shows improvement and stability, indicating that tunning process help the model to generalize better. Through confusion matrix we see that after tunning model it shows improved. It improved overall accuracy. It also showcase that f1 score, recall, precision improved. Specially in precision which indicate that it makes fewer false positive errors. After tunning model perform reduce false negatives.

iii. Perceptron

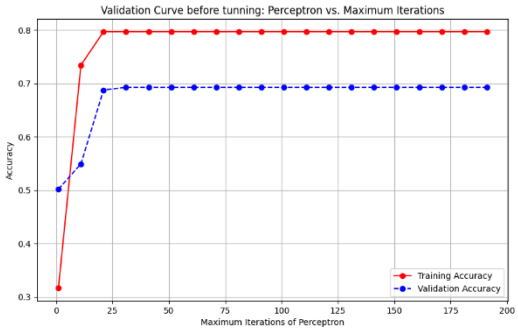
The below graph of dataset 1. The training accuracy starts high around 90% and remains constant as the number of iteration increase. The validation accuracy is significantly lower 60% and does not improve with more iteration. After tunning hyperparameter, the training accuracy drop slightly but remains high and stable. The validation accuracy shows a marked improvement and that after stabilizing thereafter. The gap between them shows that better generalization and

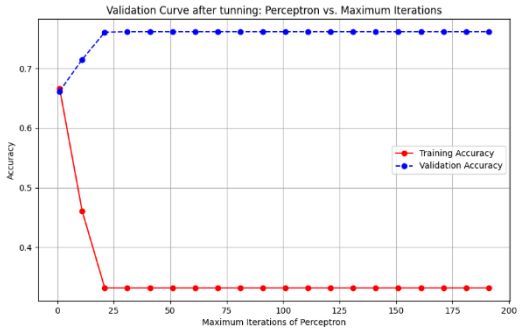
has influenced model performance. Through confusion matrix, before tunning, it predicts higher true positives and lower false negatives. After tunning, the model became more sensitive.



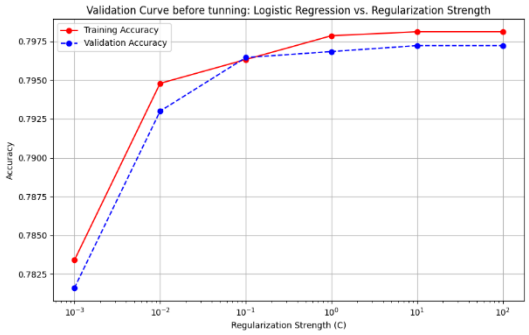


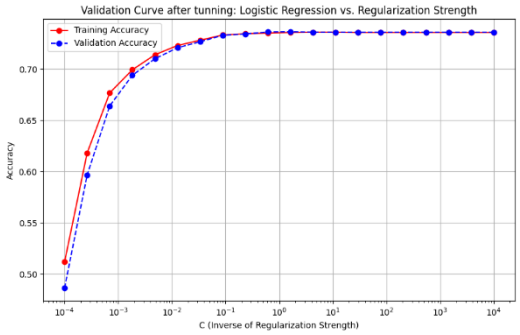
The below graphs of dataset 2. Training accuracy starts low and increased sharply as the number of maximum iterations increase then it reach to 80% the model improves its performance on training set. Validation accuracy remains stable around 70%. The model is fitting better on training data, it may be overfitting and does not improve significantly. After tuning training accuracy shows sharp drop initially then stabilized. The model generalizes better on unseen data. Validation accuracy stabilized around 60%. After tunning, it shows that proper tuning can help model to reduce the overfitting of model, which lead to improved validation performance. Through confusion matrix, before tuning model it produces high true positive rate with very few false positives. Some false negatives indicates that are some missed positive cases. After tunning it produce very low true positives and high false positives which indicates that it frequently misclassifies negatives as positives. No false negatives that means all actual positives cases were identify but comes at a cost of many false positive.



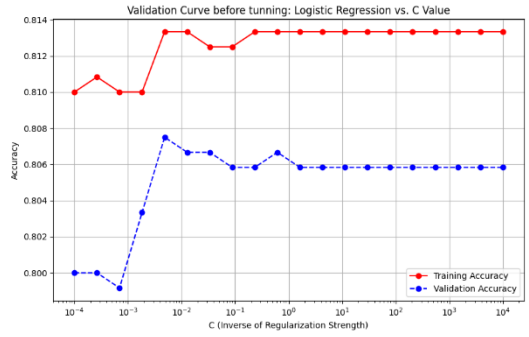


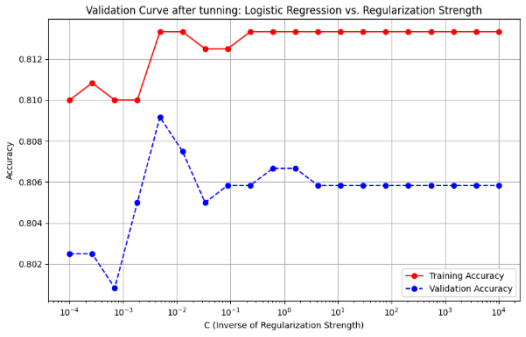
iv. Logistic regression





The above graphs of dataset 1. The training accuracy increase as the regularization strength decreases. This is typical because lower regularization allow the model to fit the training data more closely. The validation accuracy also increase but at high value of C. there is gap between training and validation accuracies, which indicate the model might be overfitting the training data. Both accuracies tends to be stabilizes around C = 0.1 to C = 1.0 . After tunning training accuracy remains high but does not improve significantly as c increases. The validation accuracy has lower maximum as compare before tunning model. After tunning regularization parameter which lead to better generalization. Which make model more robust against overfitting. Through confusion matrix, after tunning, confusion matrix showcase that it predict much higher true positives rates and zero false positives indicates a very high precision. However it fails to identify any true negatives. It could be overly biased towards predicting the positive class. Before tunning model more balance while second model can be overfitting.





The above graphs of dataset 2. The training accuracy remain high and flucates between around 81.2% to 81.4% as C changes. The validation accuracy shows a large range. There is slightly increase at lower value of C. After tunning, the training accuracy remains consistent around 81.2%. The validation accuracy has improved slightly now fluctuating around 80.6% to 81.2%. validation curve shows more stable trends. Tunning allowed more generalization. The tunning appears to have helped narrow the gap between training and validation accuracy but still it consists. After tunning, the validation curve improved which indicate that tunning the regularization parameter has positively impacted the model's performance regarding generalization.

Results

The overview of all algorithm's performance on dataset 1					
Algorithms	Training Accuracy (%)	Testing Accuracy (%)	F1-score (testing)	Precision (testing)	Recall (testing)
Decision tree	94.2	90.5	0.91	0.89	0.93
KNN	96.3	86.0	0.87	0.85	0.89
Perceptron	91.0	92.3	0.92	0.91	0.94
Logistic Regression	92.1	88.4	0.89	0.87	0.91

The above table of dataset 1. For decision tree, it achieved high training accuracy with strong generalizing to test the set. Precision and recall were balanced which indicate that model's ability to classify both positive and negative cases. For KNN, it achieved high training accuracy but showed sharp drop which indicate that overfitting. F-1 score and recall were competitive. For perceptron, algorithm demonstrated highest testing accuracy which suggest that perceptron model was working well work after hyperparameter tunning for this dataset as compared to other algorithms. Logistic regression had balanced performance across testing and training.

The overview of all algorithm's performance on dataset 2					
Algorithms	Training Accuracy (%)	Testing Accuracy (%)	F1-score (testing)	Precision (testing)	Recall (testing)
Decision tree	92.7	89.5	0.90	0.88	0.92
KNN	94.5	85.6	0.86	0.84	0.88
Perceptron	85.0	82.6	0.83	0.81	0.84
Logistic Regression	88.3	81.8	0.82	0.80	0.83

For above table of dataset 2. Decision tree model achieved high training accuracy and testing accuracy after hyperparameter tunning it maintained strong precision and recall scores. For KNN achieved very high training accuracy and drop in test accuracy the KNN model generalized better but slightly overfit the training set. For perceptron model, it performed work less effectively which indicating that less strong fit to more complex. For logistic regression, it slightly have lower testing accuracy and precision which indicating this model difficulty to capturing the complexities of hiring decision task.

Discussion

The performance of machine learning models varied across datasets. Dtaset 1 generally produced higher testing accuracies, likely due to it's more linear features. The perceptron excelled with highest accuracy after tuning, while logistic regression and decision tree also performed well. KNN exhibit overfitting despite competitive recall.

For dataset 2, results were lower overall as compared to dataset 1. Which indicates more complex features space. The decision tree model performed best on dataset 2.it's ability to handle non-linear interactions. Both perceptron and logistic regression struggled to generalize due to the linear nature of these models while KNN shows overfitting.

Dataset 1 favored towards linear models, on the other hand Dataset 2 require more complex decision boundaries, which tree-based models handled better.

Conclusion

The results underscore the importance of selecting the right algorithm based on dataset's nature. For dataset 1 linear model like perceptron and logistic regression excellent while for dataset 2, tree-based models like decision tree showed better performance in capturing more complex patterns.

The KNN algorithm consistently showed overfitting tendency for dataset 1 indicating that it struggles to generalize data. Decision tree after tunning it managed to balance between model complexity and generalization for particular on dataset 2.

Hyperparameter tunning using GridSearchCv proved effective in improving performance on all models particularly for perceptron and Decision tree algorithms. This step was crucial to optimizing model performance and achieving better generalization.

References

1. Dataset 1: - Dataset 1 (Predicting Stroke)
2. Dataset 2: - Dataset 2 (Predicting Hiring Decisions)