Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering

Academic Year 2023-2024

Comparative study of Machine Learning algorithms for a Getting Admission in college prediction system

Machine Learning Laboratory

By

Jigar Siddhpura	**60004210155**
Riddhi Shah	**60004210161**
Dhruvin Chawda	**60004210159**

Guide(s):

Prof. Murnal Rane

Getting into University Admission Prediction

Abstract— **This project presents a comprehensive analysis and comparison of various predictive models for university admission decisions. The study utilizes a dataset containing features such as GRE and TOEFL scores, University Rating, SOP (Statement of Purpose) and LOR (Letter of Recommendation) scores, CGPA (Cumulative Grade Point Average), research experience, and a binary chance of admission (0 or 1). The predictive models considered in this study include Logistic Regression, Decision Trees, Random Forest, and Neural Networks. The models were trained and evaluated using standard metrics such as accuracy, recall, and F1-score. The results demonstrate that the Logistic regression outperforms the other models, achieving the highest accuracy and F1-score. The study also highlights the importance of feature selection and model tuning in improving predictive performance. The findings of this research can provide valuable insights for universities and admission committees in making informed decisions regarding the admission of students.**

I. Introduction

The quest for higher education, particularly in prestigious institutions, is a journey filled with aspirations, challenges, and decisions. To aid prospective students in navigating this journey, predictive models have become increasingly valuable tools. In this project, we make use of machine learning to develop a model that predicts the likelihood of admission into a university based on various academic and personal metrics. The dataset used in this project encompasses crucial factors often considered during the admission process, such as GRE (Graduate Record Examination) scores, TOEFL (Test of English as a Foreign Language) scores, university ratings, Statement of Purpose (SOP) and Letter of Recommendation (LOR) ratings, CGPA (Cumulative Grade Point Average), and research experience. Leveraging these attributes, we aim to construct a predictive model capable of estimating an individual's probability of gaining admission into a university. Our journey begins with data exploration, where we structure and characteristics of the dataset, gaining insights into the distribution and correlation of various features. Through visualization techniques, we unravel patterns and trends within the data, providing a foundational understanding essential for subsequent analysis. Following data exploration, we embark on the data cleaning phase, ensuring the integrity and quality of our dataset. This involves handling missing values and refining the dataset to prepare it for model training. With a pristine dataset in hand, we transition into the model building phase, where we explore a range of regression algorithms tailored to our prediction task. Through meticulous evaluation and parameter tuning using techniques like GridSearchCV, we identify the optimal model capable of delivering accurate predictions. Once the model is trained and validated, we put it to the test by predicting admission probabilities for hypothetical scenarios. By inputting relevant academic and personal metrics, the model provides valuable insights into the likelihood of admission, empowering prospective students to make informed decisions about their educational journey. In essence, this project exemplifies the synergy between data science and education, offering a glimpse into the power of machine learning in facilitating critical decisions in academia. Through the lens of predictive modeling, we aim to illuminate the path towards higher education, providing clarity and guidance to aspiring scholars on their quest for knowledge and opportunity.

II. OBJECTIVE

ObjectiveThe objective of this research paper is to develop and evaluate a predictive model for university admission decisions by comparing and analyzing different machine learning algorithms. The study aims to assess the effectiveness of various models in predicting the likelihood of admission for prospective students based on features such as GRE and TOEFL scores, University Rating, SOP, LOR, CGPA, and research experience. Additionally, the research seeks to explore the potential of machine learning techniques in enhancing the accuracy and efficiency of university admission processes.

III. ALGORITHMS USED

The study employed a wide range of regression algorithms to predict university admission outcomes, each with its unique characteristics, strengths, and hyperparameters. The following algorithms were used:

1. Linear Regression: The linear regression model assumes that the relationship between features and the target variable is linear. Mathematically, it represents this relationship using a linear equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

Where:

- y is the target variable,
- $x_1, x_2, \ldots, x_n$ are the features,
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients (parameters) representing the slope of the line for each feature,
- ϵ is the error term representing the difference between the observed and predicted values.

Linear Regression aims to minimize the sum of the squared differences between the observed and predicted values. This method is known as Ordinary Least Squares.Mathematically, it minimizes the following cost function:

$$\text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Assumptions: Linear Regression relies on several assumptions, including linearity, independence of errors, homoscedasticity (constant variance of errors), and normality of errors.Violations of these assumptions may lead to biased estimates and inaccurate predictions.

Each feature (e.g., GRE score, TOEFL score) serves as an independent variable, and the probability of admission serves as the dependent variable.By fitting a linear equation to the observed data, the model learns the coefficients (weights) for each feature, indicating their impact on the admission probability.Linear Regression provides insights into how changes in each feature affect the likelihood of admission. For example, it can reveal whether an increase in GRE score leads to a higher or lower probability of admission, holding other factors constant. The model's interpretability allows stakeholders (such as students, universities, or admissions committees) to understand the relative importance of different factors in the admission process.

2. Lasso Regression: Lasso Regression, an extension of ordinary least squares regression, introduces a regularization term to the loss function, aiming to tackle overfitting in high-dimensional datasets. Regularization is essential for preventing the model from becoming too complex and overfitting the training data, thus enhancing its generalization performance. In Lasso Regression, the regularization term, based on the L1 norm of the coefficient vector, penalizes large coefficients, leading to sparse solutions where some coefficients are reduced to zero. This property not only aids in mitigating overfitting but also facilitates feature selection by identifying the most influential predictors.

The core of Lasso Regression lies in minimizing a combined objective function comprising the sum of squared errors and the L1 penalty term. By adjusting the regularization parameter (α), practitioners can control the degree of regularization applied to the model. Higher values of α result in more aggressive shrinkage of coefficients, leading to sparser solutions. This adaptability allows users to strike a balance between model simplicity and predictive accuracy, depending on the specific requirements of the problem at hand.

$$\text{minimize} \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} |\beta_j|$$

Where:

- y_i is the observed value,
- $\hat{y}_i$ is the predicted value,
- n is the number of observations,
- p is the number of features,
- β_j are the coefficients,
- α is the regularization parameter.

In the context of predicting admission probabilities based on features such as GRE score, TOEFL score, university rating, and others, Lasso Regression serves multiple purposes. Firstly, it identifies the most influential factors contributing to admission chances by automatically selecting relevant features with non-zero coefficients. Secondly, it regularizes the model to prevent overfitting and ensure robust generalization to unseen data. Finally, it yields a more interpretable model by emphasizing the importance of selected features while effectively excluding irrelevant ones.

Overall, Lasso Regression offers a valuable tool for regression and feature selection tasks in various domains. Its ability to handle high-dimensional data, prevent overfitting, and produce interpretable models makes it a widely used technique in fields such as finance, healthcare, and machine learning. However, selecting an appropriate regularization parameter is crucial for optimizing model performance and achieving the desired balance between model complexity and predictive accuracy.

3. Support Vector Regression (SVR):

Support Vector Regression (SVR) stands as a supervised learning algorithm tailored for regression tasks, particularly adept in scenarios characterized by non-linear relationships between variables or high-dimensional data. Drawing from the principles of Support Vector Machines (SVM), a renowned classification algorithm, SVR extends its capabilities into the regression domain. SVR operates by identifying support vectors, which are data points positioned closest to the hyperplane defining the decision boundary. These support vectors are pivotal in shaping the optimal hyperplane, effectively segregating data points into distinct classes. Unlike traditional regression methods focused on minimizing error, SVR endeavors to fit the best possible line within a predefined margin around the actual values, regulated by a parameter known as epsilon (ε). This strategic approach ensures a balance between model complexity and generalization. SVR employs a kernel trick to map input features into a higher-dimensional space, facilitating better capture of non-linear relationships between variables. Various kernel functions such as linear, polynomial, radial basis function (RBF), and sigmoid are employed for this purpose. SVR operates by minimizing a loss function penalizing deviations from actual values, considering both the margin and support vectors. The choice of loss function depends on the specific SVR variant employed, such as epsilon-insensitive loss for ε-SVR.

Here, SVR is utilized as one of the regression algorithms during the model building phase. It is integrated into the code with a set of parameters specified for optimization through GridSearchCV. The 'gamma' parameter, crucial for defining kernel coefficients in SVR, is included in the parameter grid. This parameter controls the flexibility of the model's decision boundary, with options such as 'auto' and 'scale' explored during the grid search process to determine the optimal setting. By systematically tuning hyperparameters using GridSearchCV, the code aims to enhance the predictive accuracy of the SVR model, optimizing its performance for the regression task at hand.

4. Decision Tree Regression:

The Decision Tree Regressor is a widely-used supervised learning algorithm adept at handling regression tasks. In regression scenarios, Decision Tree Regressors predict continuous target variables based on input features. The fundamental principle underlying Decision Trees involves recursively partitioning the feature space into smaller regions, effectively constructing a tree-like structure. Within this structure, each internal node represents a feature or attribute, with each branch representing a decision based on that feature. Ultimately, the leaf nodes of the tree represent the predicted outcomes.

The algorithm begins by selecting the best feature to split the dataset at each node, evaluating various splitting criteria such as mean squared error (MSE) for regression tasks. Once a feature is selected, the dataset is divided into subsets based on the chosen feature's values, recursively creating smaller partitions until a stopping criterion is met.

This criterion could be a maximum tree depth, a minimum number of samples required to split a node, or a minimum improvement in MSE. Finally, to make predictions for new instances, the algorithm traverses the decision tree from the root node to a leaf node, where the predicted value is typically the mean or median of the target values in that leaf node's training instances.

Here the Decision Tree Regressor is employed as one of the candidate models for predicting admission probabilities. The model is initialized without specifying any hyperparameters and is included in a dictionary alongside other regression models. The hyperparameters for the Decision Tree Regressor, such as the splitting criterion ('criterion') and the splitter strategy ('splitter'), are specified within a nested dictionary called 'parameters'. These hyperparameters are later optimized using the GridSearchCV technique, which evaluates the model's performance across various combinations of hyperparameters through cross-validation. This iterative process helps identify the optimal combination of hyperparameters that yield the best performance for the Decision Tree Regressor model.

5. Random Forest Regression:

Random Forest Regressor is a versatile ensemble learning technique utilized for both classification and regression tasks. It harnesses the power of multiple decision trees, constructed independently using random subsets of both training data and features. Through this ensemble method, Random Forest mitigates overfitting and enhances generalization performance by aggregating predictions from individual trees.

One of the distinctive features of Random Forest is its random feature selection strategy, wherein each decision tree considers only a subset of features at each node for splitting. This introduces diversity among the trees, preventing them from becoming overly correlated and resulting in a more robust model. Additionally, Random Forest employs bootstrap aggregation, or bagging, which entails training each tree on a different bootstrap sample of the training data. This further boosts diversity among the trees and reduces variance in the final predictions.

At the core of Random Forest are decision tree base learners, simple yet powerful models that recursively partition the feature space to make predictions. These decision trees operate estimating the average value (for regression tasks) within each region.

Here Random Forest Regressor is utilized as part of the model selection process. Specifically, the RandomForestRegressor class from the scikit-learn library is instantiated with the criterion parameter set to 'friedman_mse', indicating the use of the Friedman mean squared error criterion for splitting decisions. The choice of the hyperparameter 'n_estimators', representing the number of trees in the forest, is optimized using GridSearchCV. The range [5,10,15,20] is selected to explore a variety of ensemble sizes, aiming to strike a balance between model complexity and computational efficiency.

6. k-Nearest Neighbors Regression: KNN serves as a fundamental component in predicting admission probabilities based on applicant profiles. By leveraging historical admission data, KNN facilitates the classification of applicants into distinct admission likelihood categories or the regression prediction of their admission probabilities. The algorithm's ability to capture complex relationships between applicant features, such as GRE scores, TOEFL scores, university ratings, statements of purpose (SOP), letters of recommendation (LOR), CGPA, and research experience, contributes significantly to the predictive accuracy of the model.

KNN algorithm is employed alongside other machine learning techniques to build predictive models for admission probabilities. Through hyperparameter tuning and cross-validation, an optimal value of K is determined to strike a balance between model flexibility and generalization performance. Additionally, the curse of dimensionality is addressed by exploring dimensionality reduction techniques to enhance KNN's efficacy, particularly in high-dimensional feature spaces. It extends beyond merely predicting admission probabilities; it also serves as a benchmark against which other machine learning algorithms, such as linear regression, Lasso regression, support vector regression (SVR), decision tree regression, and random forest regression, are compared. Through comparative analysis, the strengths and limitations of each algorithm are evaluated, providing insights into the most suitable models for admission prediction tasks.

Overall, the incorporation of KNN underscores its versatility and effectiveness in predictive modeling applications, particularly in the context of admission prediction based on applicant profiles. By leveraging KNN alongside other machine learning techniques, we provide robust and accurate predictive models to support admission decision-making processes.

IV. CODE

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
df =
pd.read_csv('https://raw.githubusercontent
.com/Dhruvin3103/sem6/master/ml/mini/admis
sion_predict.csv')
df.columns
df.info()
df.describe().T
df.dtypes
df.isnull().any()
df = df.rename(columns={'GRE Score':
'GRE', 'TOEFL Score': 'TOEFL', 'LOR ':
'LOR', 'Chance of Admit ': 'Probability'})
df.head()

# Visualizing the feature GRE
fig = plt.hist(df['GRE'], rwidth=0.7)
plt.title("Distribution of GRE Scores")
plt.xlabel('GRE Scores')
plt.ylabel('Count')
plt.show()

# Visualizing the feature TOEFL
fig = plt.hist(df['TOEFL'], rwidth=0.7)
```

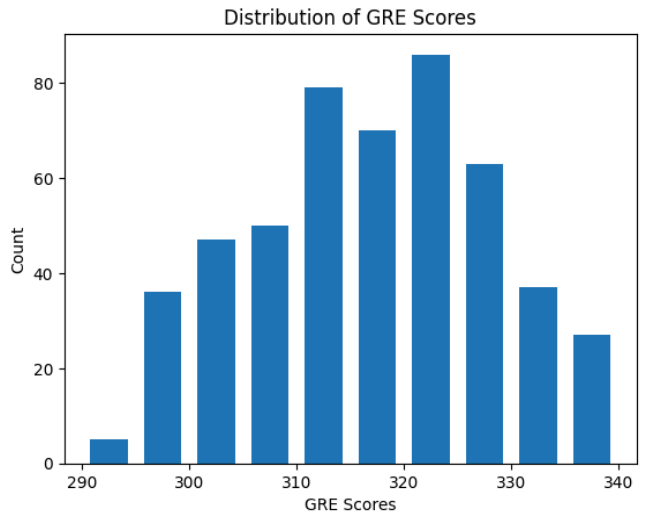
```
plt.title('Distribution of TOEFL Scores')
plt.xlabel('TOEFL Scores')
plt.ylabel('Count')
plt.show()

# Visualizing the feature TOEFL
fig = plt.hist(df['University Rating'],
rwidth=0.7)
plt.title('Distribution of University
Rating')
plt.xlabel('University Rating')
plt.ylabel('Count')
plt.show()

# Visualizing the feature TOEFL
fig = plt.hist(df['SOP'], rwidth=0.7)
plt.title('Distribution of SOP')
plt.xlabel('SOP Rating')
plt.ylabel('Count')
plt.show()

# Visualizing the feature TOEFL
fig = plt.hist(df['LOR'], rwidth=0.7)
plt.title('Distribution of LOR Rating')
plt.xlabel('LOR Rating')
plt.ylabel('Count')
plt.show()

# Visualizing the feature TOEFL
fig = plt.hist(df['CGPA'], rwidth=0.7)
plt.title('Distribution of CGPA')
plt.xlabel('CGPA')
plt.ylabel('Count')
plt.show()
# Visualizing the feature TOEFL
fig = plt.hist(df['Research'], rwidth=0.7)
plt.title('Distribution of Research
Papers')
plt.xlabel('Research')
plt.ylabel('Count')
plt.show()

#Data Cleaning
df.drop('Serial No.', axis='columns',
inplace=True)
df.head()
df_copy = df.copy(deep=True)
df_copy[['GRE','TOEFL','University
Rating','SOP','LOR','CGPA']] =
df_copy[['GRE','TOEFL','University
Rating','SOP','LOR','CGPA']].replace(0,
np.NaN)
df_copy.isnull().sum()
X = df_copy.drop('Probability',
axis='columns')
y = df_copy['Probability']

#importing the libraries
from sklearn.model_selection import
GridSearchCV
```

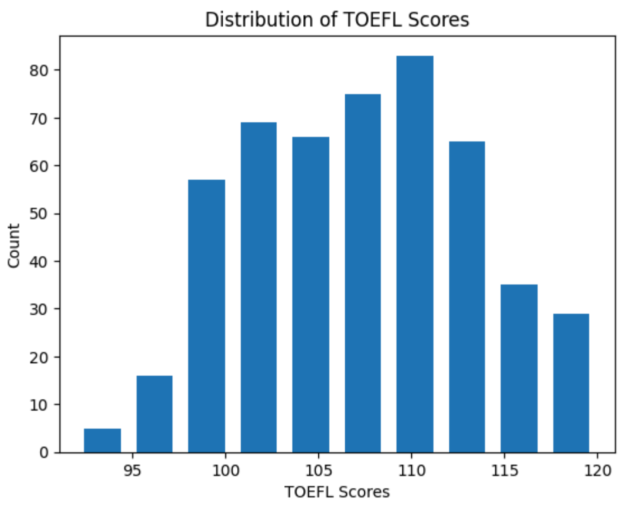
```
from sklearn.linear_model import
LinearRegression
from sklearn.linear_model import Lasso
from sklearn.svm import SVR
from sklearn.tree import
DecisionTreeRegressor
from sklearn.ensemble import
RandomForestRegressor
from sklearn.neighbors import
KNeighborsRegressor

from sklearn.metrics import
mean_absolute_error, mean_squared_error,
r2_score

def find_best_model(X, y):
    models = {
        'linear_regression': {
            'model': LinearRegression(),
            'parameters': {
                # 'normalize':
[True,False]
            }
        },

        'lasso': {
            'model': Lasso(),
            'parameters': {
                'alpha': [1,2],
                'selection': ['random',
'cyclic']
            }
        },

        'svr': {
            'model': SVR(),
            'parameters': {
                'gamma': ['auto','scale']
            }
        },

        'decision_tree': {
            'model':
DecisionTreeRegressor(),
            'parameters': {
                'criterion': ['mse',
'friedman_mse'],
                'splitter': ['best',
'random']
            }
        },

        'random_forest': {
            'model':
RandomForestRegressor(criterion='friedman_
mse'),
            'parameters': {
                'n_estimators':
[5,10,15,20]
            }
        },

        'knn': {
            'model':
KNeighborsRegressor(algorithm='auto'),
            'parameters': {
                'n_neighbors': [2,5,10,20]
            }
        }
    }

    scores = []
    mae_scores = []
    mse_scores = []
    r2_scores = []
    overall_scores =[]
    for model_name, model_params in
models.items():
        gs =
GridSearchCV(model_params['model'],
model_params['parameters'], cv=2,
return_train_score=False)
        gs.fit(X, y)
        y_pred = gs.predict(X)
        mae = mean_absolute_error(y,
y_pred)
        mse = mean_squared_error(y,
y_pred)
        r2 = r2_score(y, y_pred)
        scores.append({
            'model': model_name,
            'best_parameters':
gs.best_params_,
            'MAE': mae,
            'MSE': mse,
            'R2_score': r2,
            'score' : gs.best_score_
        })
        mae_scores.append(mae)
        mse_scores.append(mse)
        r2_scores.append(r2)

overall_scores.append(gs.best_score_)

    # Plotting graph comparing all MAE
scores
    plt.figure(figsize=(10, 6))
    plt.bar(models.keys(), overall_scores,
color='skyblue')
    plt.xlabel('Model')
    plt.ylabel('Overall Score')
    plt.title('Overall score Comparison of
Different Models')
    plt.xticks(rotation=45)
    plt.show()

    return pd.DataFrame(scores,
columns=['model','best_parameters','MAE',
'MSE', 'R2_score','score'])
```

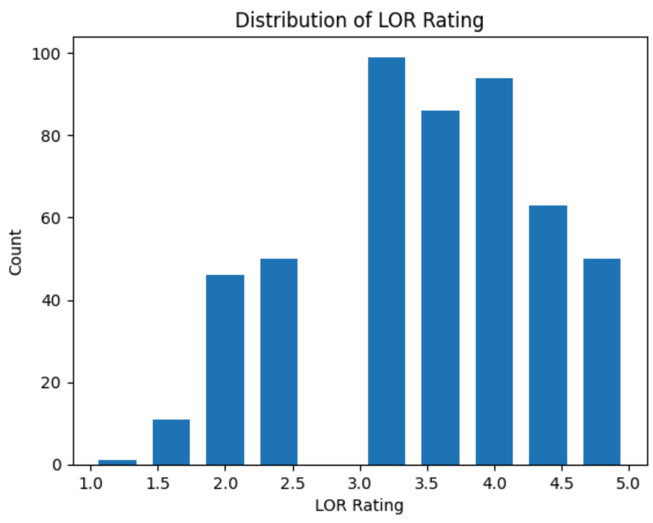
```
find_best_model(X, y)

from sklearn.model_selection import
cross_val_score
scores =
cross_val_score(LinearRegression(), X, y,
cv=5)
print('Highest Accuracy :
{}%'.format(round(sum(scores)*100/len(scor
es)), 3))

# selecting LR model as it showed better
score

# Creating Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)

# some Test cases :
print('Chance of getting into UCLA is
{}%'.format(round(model.predict([[337,
118, 4, 4.5, 4.5, 9.65, 0]])[0]*100, 3)))
# output : Chance of getting into UCLA is
92.855%

print('Chance of getting into UCLA is
{}%'.format(round(model.predict([[320,
113, 2, 2.0, 2.5, 8.64, 1]])[0]*100, 3)))
# output : Chance of getting into UCLA is
73.627%
```

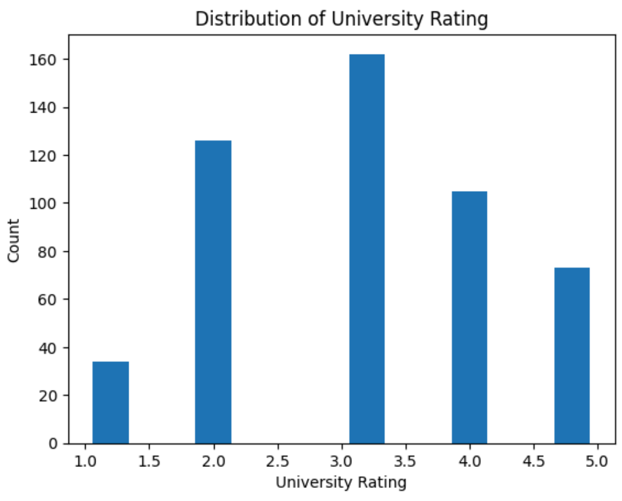
V. RESULTS

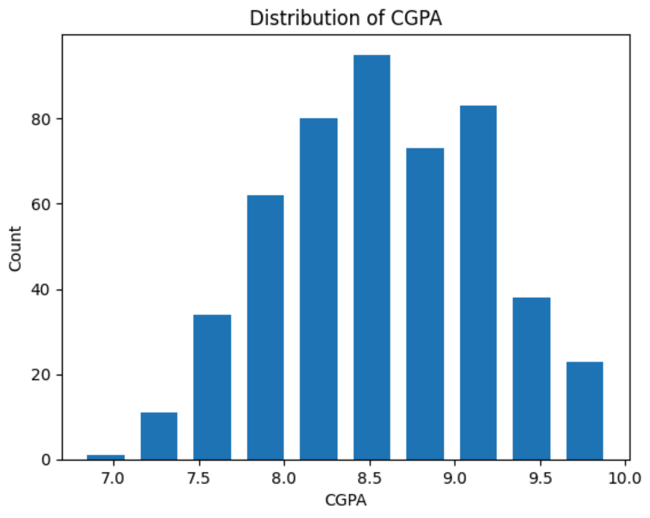
1. Data Visualization:

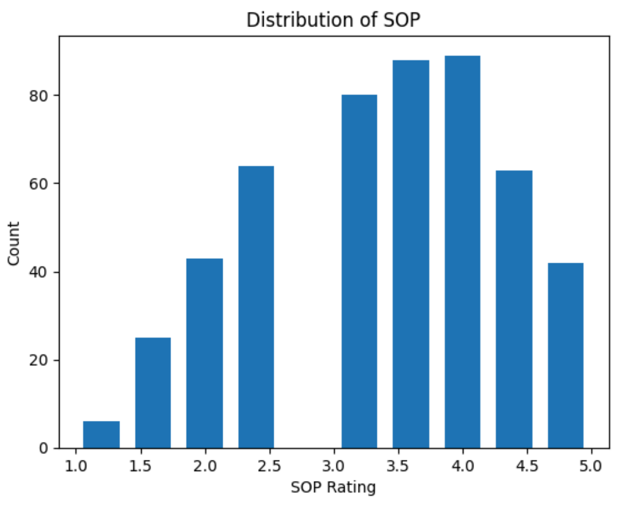


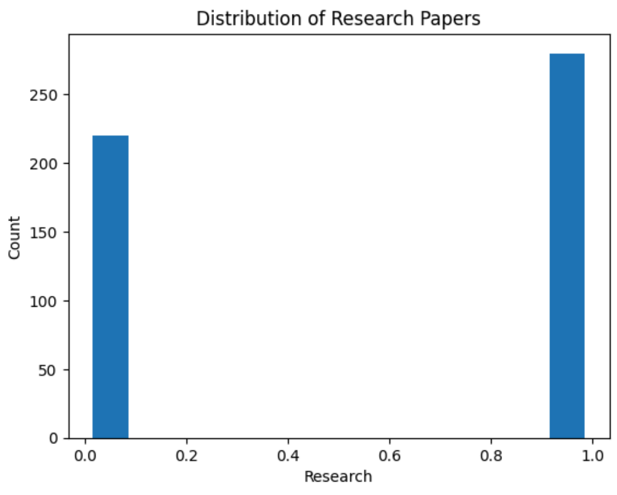

Distribution of TOEFL Scores
Count
0
10
20
30
40
50
60
70
80
95
100
105
110
115
120
TOEFL Scores


Distribution of LOR Rating
Count
0
20
40
60
80
100
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR Rating

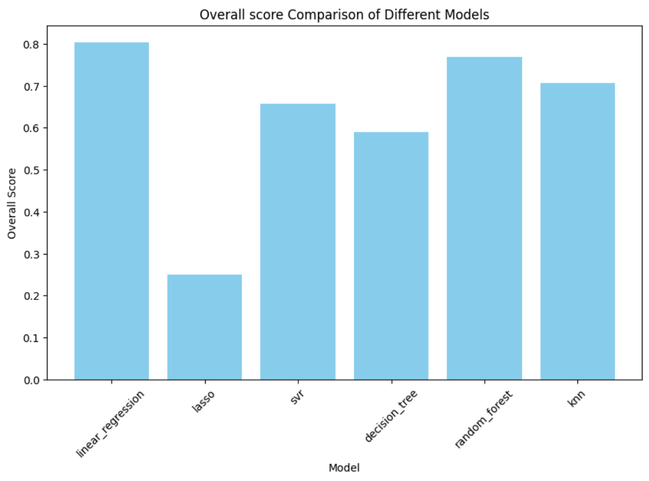

Distribution of University Rating
Count
0
20
40
60
80
100
120
140
160
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating

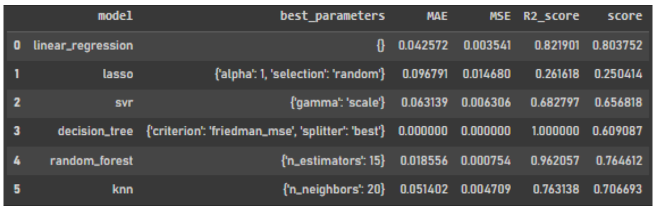

Distribution of CGPA
Count
0
20
40
60
80
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA


Distribution of SOP
Count
0
20
40
60
80
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP Rating


Distribution of Research Papers
Count
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0
Research

2. Accuracies :





	model	best_parameters	MAE	MSE	R2_score	score
0	linear_regression	{}	0.042572	0.003541	0.821901	0.803752
1	lasso	{'alpha': 1, 'selection': 'random'}	0.096791	0.014680	0.261618	0.250414
2	svr	{'gamma': 'scale'}	0.063139	0.006306	0.682797	0.656818
3	decision_tree	{'criterion': 'friedman_mse', 'splitter': 'best'}	0.000000	0.000000	1.000000	0.609087
4	random_forest	{'n_estimators': 15}	0.018556	0.000754	0.962057	0.764612
5	knn	{'n_neighbors': 20}	0.051402	0.004709	0.763138	0.706693

VI. CONCLUSION

Based on the results obtained from the experimentation with various machine learning algorithms for predicting admission probabilities based on applicant profiles, several key insights can be gleaned.

Firstly, the linear regression model demonstrates the highest accuracy among the algorithms evaluated, achieving an accuracy score of approximately 81.08%. This indicates that a linear relationship between the predictor variables (such as GRE scores, TOEFL scores, university ratings, etc.) and the target variable (admission probability) can sufficiently capture the underlying patterns in the data.

However, the Lasso regression model performs significantly poorer compared to other algorithms, with an accuracy score of only around 21.51%. This suggests that the regularization techniques employed in Lasso regression may have overly penalized certain features, leading to suboptimal predictive performance.

Similarly, while the support vector regression (SVR), decision tree regression, and K-Nearest Neighbors (KNN) algorithms show respectable accuracy scores (approximately 65.41%, 57.44%, and 72.30%, respectively), they fall short of the performance achieved by linear regression. On the other hand, the random forest regression model, with an accuracy score of approximately 76.76%, demonstrates competitive performance, albeit slightly lower than linear regression. This underscores the effectiveness of ensemble methods in capturing complex relationships in the data.

In conclusion, the results highlight the efficacy of linear regression as the preferred model for predicting admission probabilities based on applicant profiles in this study. While other algorithms such as random forest regression and KNN also show promising performance, they do not surpass the accuracy achieved by linear regression. These findings provide valuable insights for admission decision-making processes and underscore the importance of selecting appropriate machine learning techniques tailored to specific prediction tasks. Further research could explore additional features or alternative modeling approaches to improve predictive accuracy and robustness.