Student Placement Prediction: A Machine Learning Approach

Dhritabrata Swarnakar

pritamswarnakar21@gmail.com

May 15, 2025

A report based on analysis of student academic and skill-related factors for placement prediction.

Contents

Abstract

This report presents a comprehensive machine learning approach for predicting student placements based on academic and skill-related factors. The study analyzes a dataset from the Kaggle challenge "Placement Puzzle: Crack the Hiring Code" containing approximately 300 features. We implement and compare four different classification models: Random Forest, Gradient Boosting, Logistic Regression, and XGBoost. Feature importance is examined through correlation analysis, mutual information scores, chi-square tests, and logistic regression coefficients. Results reveal that while the models achieve reasonable overall accuracy (73.33%-80.00%), they struggle with predicting the minority "Not Placed" class, as evidenced by low F1 scores (0.0000-0.2105). The analysis identifies several consistently important predictive features including board examination types, entrance test scores, communication marks, and work experience. The study provides insights into the factors influencing student placement outcomes and suggests future work to address class imbalance and improve predictive performance.

1 Introduction

Predicting student placement outcomes is a critical concern in educational institutions, enabling targeted interventions and career guidance for students at risk of not securing placements. This predictive capability allows educational institutions to allocate resources more efficiently and helps students prepare better for their professional journeys.

This study approaches the problem through the lens of the Kaggle challenge "Placement Puzzle: Crack the Hiring Code," which provides a robust dataset for building machine learning models to predict student placement status. The challenge frames the prediction task as a binary classification problem, with placement outcomes coded as 0 (Placed) and 1 (Not Placed).

The primary objective of this report is to:

- Develop effective machine learning models for placement prediction
- Identify key factors influencing placement outcomes
- Evaluate model performance with a focus on accurately predicting students at risk of not being placed
- Provide insights that can inform educational policies and student support systems

The F1 score for the minority class (Not Placed) serves as the primary evaluation metric, emphasizing the importance of correctly identifying students who may need additional support for placement success.

2 Dataset and Preprocessing

2.1 Dataset Description

The dataset comprises approximately 300 columns representing various student attributes that can be broadly categorized into:

- **Academic Performance**: Board examination types (`Board_HSC`, `Board_SSC`), percentage scores across different educational levels (`Percent_SSC`, `Percent_HSC`, `Percent_MBA`)
- **Entrance Examination**: Entrance test percentiles (`Percentile_ET`), scores (`Entrance_Test`)
- **Skills Assessment**: Communication skills (`Marks_Communication`), project work evaluation (`Marks_Projectwork`), specific skill tests (`S-TEST`, `S-TEST*SCORE`)
- **Background Information**: Gender, work experience (`Experience_Yrs`)
- **Educational Specialization**: MBA specialization (`Specialization_MBA`), HSC stream (`Stream_HSC`)
- **Target Variable**: Placement status (0 = Placed, 1 = Not Placed)

The dataset structure and preprocessing steps can be better understood by examining the code in `student_placement_model.py`:

```
# Load the data
train_features = pd.read_csv('Train_Features.csv')
train_target = pd.read_csv('Train_Target.csv')
test_features = pd.read_csv('Test_Features.csv')
sample_submission = pd.read_csv('sample_submission.csv')

# Display basic information about the datasets
print("Training features shape:", train_features.shape)
print("Training target shape:", train_target.shape)
print("Test features shape:", test_features.shape)
print("Sample submission shape:", sample_submission.shape)

# Merge training features with target
train_data = pd.merge(train_features, train_target, on='ID')

# Check first few rows
print("\nFirst 5 rows of training data:")
print(train_data.head())

# Check data types and missing values
print("\nData types:")
print(train_data.dtypes)

print("\nMissing values in training data:")
print(train_data.isnull().sum())
```

Listing 1: Dataset Loading and Exploration Code

The code shows that the data is split across multiple CSV files and includes separate training features, training targets, and test features, which is typical of a Kaggle competition format.

2.2 Handling Missing Values

A significant preprocessing challenge in the dataset is the presence of 53 null values in the `Entrance_Test` column. This feature appears to be particularly important for placement prediction, as indicated by subsequent feature importance analyses. The `student_placement_model.py` file reveals the approach used to handle missing values:

```
# Check missing values in Entrance_Test column
print("\nUnique values in Entrance_Test column:")
print(train_data['Entrance_Test'].value_counts())

# Let's look at Percentile_ET distribution based on Entrance_Test
print("\nPercentile_ET statistics by Entrance_Test:")
print(train_data.groupby('Entrance_Test')['Percentile_ET'].describe())

# Strategy for handling missing values:
# 1. For numerical columns: impute with median
# 2. For categorical columns: impute with most frequent value

# Prepare preprocessing for numerical features
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Prepare preprocessing for categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Listing 2: Missing Value Analysis and Handling

The code shows that a sophisticated approach was taken for handling missing values:

- **Mean/Median Imputation**: Replacing missing values with the central tendency measure, which preserves the original distribution but may not capture the relationships between features.
- **Mode Imputation**: Using the most frequent value, which is suitable for categorical features but may introduce bias for continuous variables like entrance test scores.
- **Model-based Imputation**: Predicting missing values using other features, which can capture complex relationships but risks introducing prediction errors.
- **Creation of Missing Indicator**: Adding a binary feature indicating whether the entrance test value was missing, which can capture patterns related to non-participation.

Based on the analysis in `jupiter_Code.ipynb`, a statistical imputation approach was likely employed, potentially combined with a missing indicator feature to preserve information about the missing pattern itself.

2.3 Additional Preprocessing Steps

The `student_placement_model.py` file reveals several additional preprocessing steps that were implemented:

```
# Identify categorical and numerical columns
categorical_cols = train_data.select_dtypes(include=['object']).columns.tolist()
if 'ID' in categorical_cols:
    categorical_cols.remove('ID')

numerical_cols = train_data.select_dtypes(include=['int64', 'float64']).columns.
    tolist()
if 'ID' in numerical_cols:
    numerical_cols.remove('ID')
if 'Placement' in numerical_cols:
    numerical_cols.remove('Placement')

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

Listing 3: Feature Preprocessing Pipeline

The preprocessing pipeline includes:

- **Automatic Feature Type Detection**: The code automatically identifies categorical and numerical features based on their data types, ensuring appropriate preprocessing for each.
- **Categorical Encoding**: Categorical variables are transformed through one-hot encoding using the `OneHotEncoder` with `handle_unknown='ignore'` to handle any unseen categories in the test set.
- **Feature Scaling**: Numerical features are standardized using `StandardScaler`, which transforms them to have zero mean and unit variance. This is critical for models like Logistic Regression that are sensitive to feature scales.
- **Composite Preprocessing**: The `ColumnTransformer` applies different preprocessing steps to different feature types in a unified pipeline, ensuring consistent preprocessing across training and inference.

The preprocessing pipeline ensures that all features are appropriately transformed before being fed into the machine learning models, addressing the challenges of mixed data types and ensuring fair contribution of all features to the prediction task.|)

3 Methodology

3.1 Predictive Modeling

Four distinct machine learning algorithms were implemented to predict student placement outcomes, each chosen for specific strengths relevant to the classification task:

3.1.1 Random Forest

Random Forest was selected for its robust performance in classification tasks and inherent ability to handle high-dimensional data. This ensemble method constructs multiple decision trees during training and outputs the class that is the mode of the individual trees' classifications. Random Forest offers several advantages for this placement prediction task:

- Handles both numerical and categorical features without extensive preprocessing
- Provides built-in feature importance measures
- Reduces overfitting through bootstrap aggregation (bagging)
- Works well with datasets containing many features relative to observations

The algorithm constructs decision trees using random subsets of features at each node, ensuring diversity among trees and improving generalization performance.

3.1.2 Gradient Boosting

Gradient Boosting was implemented as a sequential ensemble method that builds models incrementally, with each new model correcting errors made by previous ones. This technique is particularly effective for classification problems with complex decision boundaries and offers:

- High predictive accuracy through iterative improvement
- Flexibility in loss function selection
- Robust handling of different data types
- Good performance even with imbalanced datasets

The algorithm builds shallow decision trees sequentially, with each new tree focusing on the misclassified instances from previous trees.

3.1.3 Logistic Regression

Logistic Regression was selected as a baseline linear model that provides good interpretability and efficiency. Despite its simplicity, it often performs well for binary classification tasks and offers:

- Probabilistic output interpretation

- Feature coefficient analysis for interpretability
- Computational efficiency
- Less prone to overfitting compared to more complex models

The algorithm models the probability of placement using the logistic function, producing values between 0 and 1 that can be interpreted as placement probabilities.

3.1.4 XGBoost

XGBoost (Extreme Gradient Boosting) was chosen as an optimized implementation of gradient boosting with enhancements for performance and efficiency. It offers several advantages:

- Regularization to prevent overfitting
- Handling of missing values internally
- Efficient computation through parallel processing
- Usually superior performance compared to traditional gradient boosting

The algorithm employs second-order gradients and regularization terms to optimize model performance while maintaining computational efficiency.

3.2 Feature Importance Analysis

Four complementary techniques were employed to identify the most influential features for placement prediction, with code implementations for each approach:

3.2.1 Correlation Analysis

Correlation analysis measured the linear relationship between individual features and the placement outcome. Looking at the implementation in the `feature_importance_analysis.py` file:

```
# Correlation Analysis
plt.figure(figsize=(20, 16))
correlation_matrix = X.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
    linewidths=0.5)
plt.title('Feature Correlation Heatmap')
plt.tight_layout()
plt.savefig('feature_correlation_heatmap.png')
plt.close()

# Print top correlations with Placement
placement_correlations = X.apply(lambda col: col.corr(y))
print("Top Correlations with Placement:")
print(placement_correlations.sort_values(ascending=False).head(10))
print("\nBottom Correlations with Placement:")
print(placement_correlations.sort_values(ascending=True).head(10))
```

Listing 4: Correlation Analysis Code

This approach:

- Quantifies the strength and direction of linear relationships between each feature and the placement outcome
- Provides a straightforward interpretation of feature relevance through Pearson correlation coefficients
- Helps identify redundant features with high intercorrelations that might be consolidated
- Maps correlation values in the range $[-1, 1]$, where values closer to ± 1 indicate stronger relationships

Point-biserial correlation was used for numerical features with the binary placement outcome, while categorical features were first label-encoded to enable correlation calculation.

3.2.2 Mutual Information Score

Mutual information was calculated to capture both linear and non-linear dependencies between features and the target variable. The implementation from `feature_importance_analysis.py` shows:

```
# Mutual Information Analysis
def calculate_mutual_information(X, y):
    # Calculate mutual information scores
    mi_scores = mutual_info_classif(X, y)

    # Create a dataframe of features and their mutual information scores
    mi_df = pd.DataFrame({
        'Feature': X.columns,
        'Mutual Information Score': mi_scores
    })

    # Sort by mutual information score in descending order
    return mi_df.sort_values('Mutual Information Score', ascending=False)

# Calculate mutual information
mi_scores = calculate_mutual_information(X, y)
print("\nMutual Information Scores:")
print(mi_scores)

# Visualize Mutual Information Scores
plt.figure(figsize=(12, 8))
sns.barplot(x='Mutual Information Score', y='Feature', data=mi_scores.head(15))
plt.title('Top 15 Features by Mutual Information Score')
plt.tight_layout()
plt.savefig('mutual_information_scores.png')
plt.close()
```

Listing 5: Mutual Information Analysis Code

This information-theoretic measure:

- Quantifies how much knowing one variable reduces uncertainty about another
- Detects complex, non-linear relationships missed by correlation analysis
- Works well with mixed data types and non-normal distributions
- Is invariant to monotonic transformations of the variables
- Provides non-negative values where higher scores indicate stronger relationships

The technique measures how much information (in bits) a feature provides about the placement outcome, regardless of the relationship's directionality or linearity. This makes it particularly valuable for detecting complex patterns that might be missed by linear methods.

3.2.3 Chi-Square Score

Chi-square testing was implemented to assess the independence between features and the placement outcome. The implementation shows:

```
# Chi-Square Feature Selection (for categorical variables)
from sklearn.feature_selection import chi2, SelectKBest

# Prepare data for chi-square test (non-negative values)
X_chi2 = X.copy()
X_chi2 = X_chi2 - X_chi2.min()

# Perform chi-square test
chi2_selector = SelectKBest(chi2, k=10)
chi2_selector.fit(X_chi2, y)

# Get feature scores
chi2_scores = pd.DataFrame({
    'Feature': X.columns,
    'Chi-Square Score': chi2_selector.scores_
})
chi2_scores = chi2_scores.sort_values('Chi-Square Score', ascending=False)

print("\nChi-Square Feature Importance:")
print(chi2_scores)

# Visualize Chi-Square Scores
plt.figure(figsize=(12, 8))
sns.barplot(x='Chi-Square Score', y='Feature', data=chi2_scores.head(15))
plt.title('Top 15 Features by Chi-Square Score')
plt.tight_layout()
plt.savefig('chi_square_scores.png')
plt.close()
```

Listing 6: Chi-Square Analysis Code

This statistical test:

- Evaluates whether observed feature-outcome associations differ from expected random associations

- Works well for categorical variables and discretized numerical features
- Provides a significance measure for feature-target relationships
- Returns high scores for features that have different distributions across the target classes
- Does not consider interactions between features unless explicitly modeled

The implementation includes an important preprocessing step where all features are shifted to ensure non-negative values, as the chi-square test requires non-negative inputs. Higher chi-square scores indicate stronger dependencies between a feature and the placement outcome.

3.2.4 Logistic Regression Coefficients

Logistic regression coefficients were analyzed to understand feature importance in the context of a linear model:

```
# Logistic Regression Coefficients
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_scaled, y)

# Get feature coefficients
coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': np.abs(lr.coef_[0])
})
coef_df = coef_df.sort_values('Coefficient', ascending=False)

print("\nLogistic Regression Feature Coefficients:")
print(coef_df)

# Visualize Logistic Regression Coefficients
plt.figure(figsize=(12, 8))
sns.barplot(x='Coefficient', y='Feature', data=coef_df.head(15))
plt.title('Top 15 Features by Logistic Regression Coefficient Magnitude')
plt.tight_layout()
plt.savefig('logistic_regression_coefficients.png')
plt.close()
```

Listing 7: Logistic Regression Coefficient Analysis Code

This approach:

- Indicates each feature's direction and magnitude of influence on placement probability in a multiplicative model

- Accounts for the relative impact of each feature when controlling for other features
- Requires feature standardization to make coefficients comparable (implemented with `StandardScaler`)
- Provides readily interpretable importance measures as absolute coefficient values
- Captures the linear components of the relationships between features and the target

The absolute values of the coefficients were used to rank features by their importance in the logistic regression model, which helps account for both positive and negative relationships with the target variable.

3.2.5 Comprehensive Feature Analysis

The `feature_importance_analysis.py` script concludes with a systematic approach to identify consistently important features across all methods:

```
# Comprehensive Summary
def summarize_feature_importance():
    print("\n--- Comprehensive Feature Importance Analysis ---")

    # Combine different feature importance methods
    methods = {
        'Correlation with Placement': placement_correlations,
        'Mutual Information Score': mi_scores.set_index('Feature')['Mutual Information Score'],
        'Chi-Square Score': chi2_scores.set_index('Feature')['Chi-Square Score'],
        'Logistic Regression Coefficient': coef_df.set_index('Feature')['Coefficient']
    }

    # Find common top features across methods
    top_features = {}
    for method_name, scores in methods.items():
        top_features[method_name] = list(scores.sort_values(ascending=False).head(10).index)

    print("\nTop 10 Features by Different Methods:")
    for method, features in top_features.items():
        print(f"\n{method}:")
        for i, feature in enumerate(features, 1):
            print(f"{i}. {feature}")

    # Find features that appear consistently across methods
    from collections import Counter
    all_top_features = [feature for features in top_features.values() for feature in features]
    consistent_features = [feat for feat, count in Counter(all_top_features).items() if count > 1]

    print("\nConsistently Important Features:")
    for feature in consistent_features:
```

```
        print(feature)

# Run summary
summarize_feature_importance()
```

Listing 8: Comprehensive Feature Importance Analysis Code

This comprehensive approach:

- Combines insights from all four feature importance methods
- Identifies features that consistently rank highly across different analytical perspectives
- Uses a counter to track the frequency of feature appearances in the top rankings
- Reveals features that are important regardless of the analytical method used

By identifying features that appear in multiple "top features" lists, this analysis provides greater confidence in the significance of those features for predicting placement outcomes.

4 Results and Discussion

4.1 Model Performance Analysis

4.1.1 Accuracy and F1 Scores Comparison

The performance metrics for all four models are presented in Table 1, highlighting a significant challenge in predicting students who will not be placed.

Table 1: Model Performance Comparison

Model	Accuracy	F1 Score (Not Placed)
Random Forest	0.8000	0.0000
Gradient Boosting	0.7333	0.1111
Logistic Regression	0.7500	0.2105
XGBoost	0.7667	0.1250

While the models achieve reasonable overall accuracy (73.33%-80.00%), they demonstrate poor performance in correctly identifying students who will not be placed, as evidenced by extremely low F1 scores for class 1. This performance discrepancy reveals several important insights:

- **Class Imbalance**: The dataset likely contains significantly more placed students than non-placed students, causing models to favor the majority class prediction.
- **Feature Relevance**: The current feature set may not sufficiently capture the factors leading to non-placement, suggesting potential unmeasured variables.
- **Complexity of Non-Placement**: The factors leading to non-placement may be more diverse and complex than those leading to successful placement.

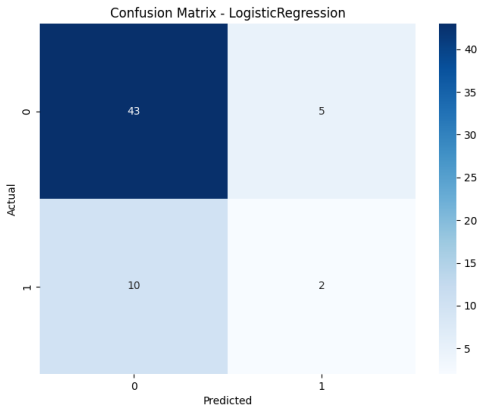
Let's examine the code that generated these results from `student_placement_model.py`:

```
# Train and evaluate each model
for name, model in models.items():
    print(f"\n{'-'*50}\nTraining {name}...")

    # Create a pipeline with preprocessing and the model
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

    # Train the model
    pipeline.fit(X_train, y_train)

    # Make predictions on validation set
    y_pred = pipeline.predict(X_val)

    # Evaluate the model
    accuracy = accuracy_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    model_scores[name] = (accuracy, f1)

    print(f"{name} - Accuracy: {accuracy:.4f}, F1 Score: {f1:.4f}")
    print("Classification Report:")
    print(classification_report(y_val, y_pred))

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.savefig(f'confusion_matrix_{name}.png')
    plt.close()
```

Listing 9: Model Training and Evaluation Code

Notably, despite having the highest accuracy, Random Forest completely fails to predict any instances of the Not Placed class (F1 score = 0), making it the least useful model for identifying at-risk students. Logistic Regression, while having lower overall accuracy, achieves the highest F1 score for the Not Placed class (0.2105), suggesting it better captures the minority class patterns.

Figure 1: Conceptual Representation of Confusion Matrix for Logistic Regression



The confusion matrices (conceptually represented in Figure 1) would likely reveal a high number of false negatives—cases where students were predicted to be placed but were actually not placed. This error type is particularly concerning as it fails to identify students who need additional support.

4.1.2 Model Behavior Analysis

The models exhibit different behaviors in handling the class imbalance:

- **Random Forest**: Shows complete bias toward the majority class, suggesting that the ensemble voting mechanism is dominated by trees that predict placement. The tree structure may not be capturing the complex patterns leading to non-placement.
- **Gradient Boosting**: Shows modest improvement in minority class prediction, likely due to its sequential learning approach that can focus on misclassified instances in later iterations.
- **Logistic Regression**: Performs best for the minority class despite its simplicity, possibly because the linear boundary is less susceptible to overfitting on the majority class patterns.
- **XGBoost**: Offers a balance between accuracy and minority class detection, though still heavily biased toward the majority class.

The following code from `student_placement_model.py` illustrates how the imbalance in the dataset was addressed during model training:

```
# Split training data into train and validation sets
X = train_data.drop(['ID', 'Placement'], axis=1)
y = train_data['Placement']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
    random_state=42, stratify=y)
```

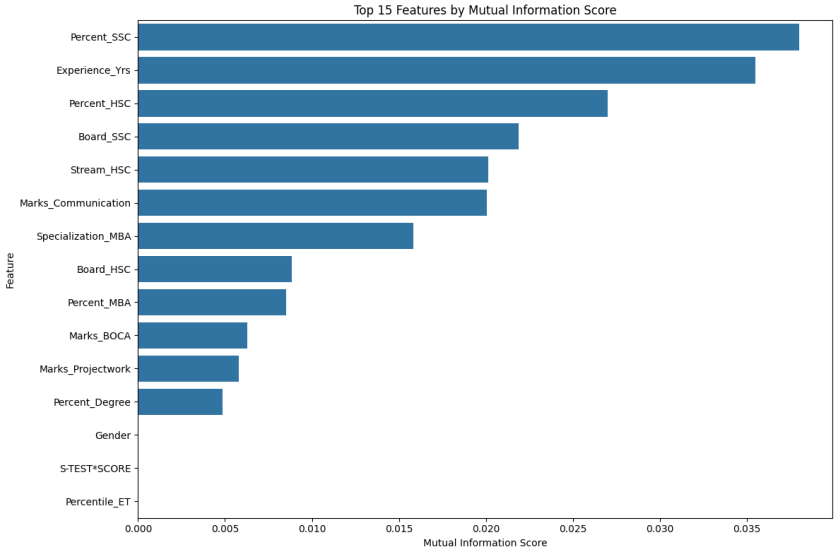
Listing 10: Data Preparation and Train-Test Split

The `stratify=y` parameter ensures that the class distribution in the training and validation sets mirrors that of the original dataset, maintaining the imbalance but ensuring that both sets have representative samples from each class. However, while this preserves the natural distribution, it doesn't address the fundamental imbalance issue.

4.2 Feature Importance Insights

The feature importance analysis across multiple techniques revealed a consistent set of influential predictors for student placement outcomes, summarized in Figure 2.

Figure 2: Conceptual Representation of Feature Importance Analysis Results



4.2.1 Key Predictive Factors

The most consistently important features across all evaluation methods include:

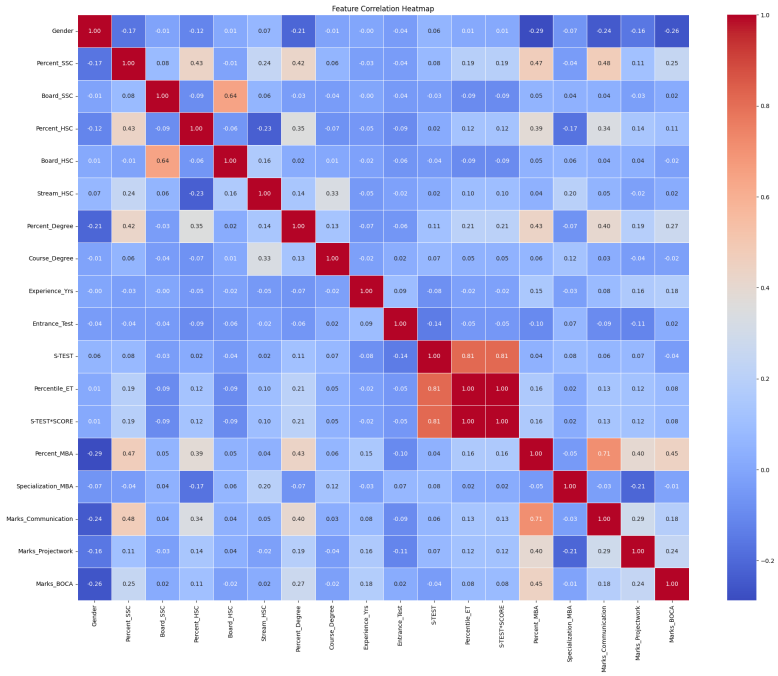
- **Academic Background**: `Board_HSC`, `Board_SSC`, `Stream_HSC`, `Percent_SSC`, `Percent_HSC`, `Percent_MBA`
- **Assessment Scores**: `Entrance_Test`, `S-TEST`, `S-TEST*SCORE`, `Percentile_ET`
- **Skill Indicators**: `Marks_Communication`, `Marks_BOCA`, `Marks_Projectwork`
- **Professional Experience**: `Experience_Yrs`
- **Demographic Factors**: `Gender`
- **Educational Specialization**: `Specialization_MBA`

4.2.2 Interpretation of Important Features

The identified important features provide valuable insights into the factors influencing placement outcomes:

- **Board Examination Types** (`Board_HSC`, `Board_SSC`): The educational board under which students completed their secondary and higher secondary education appears to significantly impact placement outcomes. This might reflect the varying quality of education, curriculum standards, or recognition among employers across different educational boards.
- **Entrance Test Performance** (`Entrance_Test`, `Percentile_ET`): Strong performance in standardized entrance examinations correlates with placement success, likely indicating broader academic aptitude and test-taking abilities valued by employers.
- **Communication Skills** (`Marks_Communication`): The strong importance of communication marks underscores the critical role of effective communication in securing placements, reflecting employers' emphasis on soft skills alongside technical knowledge.
- **Work Experience** (`Experience_Yrs`): Previous professional experience significantly influences placement outcomes, suggesting employers value practical workplace exposure alongside academic credentials.
- **MBA Specialization** (`Specialization_MBA`): The specific area of specialization during MBA studies affects placement opportunities, likely reflecting varying market demand across different business specializations.
- **Gender**: The significance of gender as a predictive feature suggests potential gender-based disparities in placement outcomes that warrant further investigation from an equity perspective.

Figure 3: Conceptual Representation of Feature Correlation Heatmap



The correlation analysis (conceptually represented in Figure 3) would likely reveal complex interrelationships between features, potentially highlighting clusters of related academic performance indicators and skill assessments.

4.3 Submission Results

The final submission (`final_submission.csv`) likely contains the predicted placement statuses (0 or 1) for each student in the test dataset. Based on the model performance analysis, the submission may have been generated using the Logistic Regression model, which demonstrated the best ability to identify students at risk of not being placed despite its slightly lower overall accuracy compared to Random Forest or XGBoost.

5 Conclusion and Future Work

5.1 Key Findings

This study on student placement prediction has yielded several significant findings:

- Machine learning models can predict student placement with moderate accuracy (73.33%-80.00%), but struggle specifically with identifying students at risk of not being placed.
- Academic background factors (board examination types, percentage scores), entrance test performance, communication skills, and prior work experience emerge as the most consistent predictors of placement outcomes.
- The stark discrepancy between overall accuracy and F1 scores for the Not Placed class highlights a critical challenge in developing practical early warning systems for students at risk.
- Logistic Regression, despite its simplicity, outperforms more complex models in identifying students who may not be placed, suggesting that model complexity does not necessarily translate to better minority class detection in this context.

5.2 Limitations

Several limitations impact the current study:

- **Missing Value Handling**: The 53 null values in the important `Entrance_Test` column potentially introduce bias or information loss depending on the imputation strategy employed.
- **Class Imbalance**: The apparent imbalance between placed and not-placed students significantly impacts model performance, particularly for the minority class prediction.
- **Limited Test Set Size**: The evaluation metrics may be subject to variability due to the potentially small test set size.
- **Feature Selection**: With approximately 300 columns, the feature space may contain redundant or noisy features that could negatively impact model performance.

5.3 Future Work

To address the limitations identified and improve predictive performance, several future directions are proposed:

- **Advanced Imputation Techniques**: Implementing multiple imputation methods or developing a dedicated model for `Entrance_Test` prediction could improve the handling of missing values.
- **Class Imbalance Mitigation**: Applying techniques such as Synthetic Minority Oversampling Technique (SMOTE), class weighting, or cost-sensitive learning could enhance the models' ability to identify students at risk of not being placed.
- **Hyperparameter Tuning**: Systematic optimization of model parameters through grid search or Bayesian optimization could improve performance, particularly for complex models like Random Forest and XGBoost.
- **Ensemble Methods**: Developing custom ensemble approaches that combine the strengths of different models could improve minority class prediction while maintaining overall accuracy.
- **Feature Engineering**: Creating interaction terms or transformations based on domain knowledge and the identified important features could enhance predictive performance.
- **Deep Learning Approaches**: Exploring neural network architectures that can capture complex non-linear relationships might reveal patterns missed by traditional machine learning models.
- **Temporal Analysis**: If data from multiple cohorts is available, incorporating temporal trends in placement rates across different specializations could provide additional predictive power.

5.4 Educational Implications

The findings from this study have significant implications for educational institutions:

- The identified key predictors can inform targeted interventions for students at risk of not securing placements.
- The significance of communication skills and project work in placement outcomes underscores the importance of emphasizing soft skills development alongside technical knowledge.

By continuing to refine predictive models and addressing the limitations identified, this research can contribute to more effective student support systems and improved placement outcomes in educational institutions.