BDM 3035: CAPSTONE PROJECT
FINAL REPORT

SOP EVALUATION USING MACHINE LEARNING TECHNIQUES

GROUP E :

C0887501 - Anju Krishna C0887501
C0887227 - Dushyant Singh
C0884599 - Jiya Maria Eapen
C0887502 - Athish Jobin
C0885273 - Roy Joseph

1. Abstract :

In today's higher education landscape, the evaluation of Statements of Purpose (SOPs) plays a pivotal role in university admissions. However, the conventional methods of human assessment are laden with challenges including time constraints, susceptibility to biases, lack of scalability, and inconsistent evaluations across applicants. To address these issues, this project proposes a novel approach leveraging machine learning (ML) techniques to automate SOP evaluation. The process entails a comprehensive methodology encompassing data preprocessing, methodology selection, model training, and evaluation. Firstly, data preprocessing involves cleaning and preparing SOP data, including tasks such as text normalization and handling missing values. Secondly, methodology selection involves exploring various ML techniques such as natural language processing (NLP) and sentiment analysis to determine the most suitable approach. Thirdly, model training involves feeding preprocessed data into chosen algorithms, adjusting parameters, and iteratively refining models for optimal performance. Finally, evaluation metrics such as accuracy, precision, recall, F1 score, and area under the ROC curve (AUC) are employed to assess the effectiveness of the ML-based SOP evaluation system. Ultimately, this project aims to streamline the SOP evaluation process, improving efficiency, reducing biases, enhancing scalability, and ensuring consistent evaluations across applicants, thereby benefiting both universities and prospective students alike.

2. Introduction :

Introduction In the contemporary landscape of higher education, the process of admissions to universities has become increasingly competitive and multifaceted. Central to this process is the evaluation of Statements of Purpose (SOPs), which serve as crucial documents submitted by applicants to elucidate their academic background, research interests, career goals, and reasons for pursuing a particular program of study. Traditionally, the assessment of SOPs has been entrusted to human reviewers, who meticulously scrutinize each document to gauge the applicant's suitability for admission. However, this manual approach is beset with inherent challenges, including time constraints, susceptibility to biases, lack of scalability, and the potential for inconsistent evaluations across applicants.

Background Information on the Problem Domain

The reliance on human reviewers for SOP evaluation has long been a cornerstone of the admissions process. Yet, as the volume of applications continues to surge and the demand for a fair, efficient, and reliable evaluation mechanism grows, there arises an imperative to explore alternative methodologies that can complement or even supplant traditional assessment methods. Moreover, the subjectivity inherent in human evaluations introduces an element of variability that can compromise the integrity and fairness of the admissions process. As such, there is a pressing need for innovative approaches that can mitigate these challenges and enhance the efficacy of SOP evaluation in university admissions.

Statement of the Problem

The overarching problem confronting university admissions committees lies in the inefficiency and subjectivity associated with the manual evaluation of SOPs. Human reviewers are tasked with assessing a multitude of documents, often under stringent time constraints, leading to the risk of oversight or inconsistency in evaluations. Furthermore, the inherent biases and idiosyncrasies of individual reviewers may influence their judgments, potentially resulting in disparate outcomes for similarly qualified applicants. Addressing these challenges is paramount to ensuring a transparent, equitable, and streamlined admissions process that upholds the principles of meritocracy and diversity.

Objectives of the Project

The primary objective of this project is to develop and implement a machine learning-based system for automating the evaluation of SOPs in university admissions. By harnessing the power of computational algorithms, the project aims to overcome the limitations of manual assessment methods, thereby enhancing the efficiency, fairness, and scalability of the SOP evaluation process. Specifically, the project seeks to achieve the following objectives:

- To design and implement a robust methodology for preprocessing SOP data, including text normalization, feature extraction, and handling of missing values.
- To explore and evaluate various machine learning techniques, such as natural language processing (NLP), sentiment analysis, and classification algorithms, for their efficacy in automating SOP evaluation.

- To develop and train machine learning models using labeled SOP data, optimizing model parameters to achieve high accuracy and consistency in evaluations.
- To evaluate the performance of the developed models using appropriate metrics, such as accuracy, precision, recall, F1 score, and area under the ROC curve (AUC).
- To demonstrate the feasibility and utility of the proposed ML-based SOP evaluation system through empirical validation and comparison with human assessments.

Overview of the Methodology Used

The methodology employed in this project encompasses several stages, each tailored to address specific aspects of automating SOP evaluation using machine learning techniques. The process begins with data preprocessing, wherein SOP documents are cleaned, standardized, and transformed into a suitable format for subsequent analysis. Next, a comprehensive exploration of machine learning methodologies is undertaken, encompassing techniques such as NLP, sentiment analysis, and classification algorithms. Following this, **Random Forest Classifier Model** is developed and trained using labeled SOP data, with a focus on optimizing model performance through iterative refinement. Finally, the performance of the developed models is rigorously evaluated using established metrics, providing insights into their effectiveness and suitability for automating SOP evaluation in the context of university admissions.

3. Data Collection and Preprocessing :

Description of the Data Sources:

The dataset comprises a collection of Statements of Purpose (SOPs) submitted by university applicants, along with their corresponding admission decisions. Each SOP is represented as a text document, and the admission decision indicates whether the applicant was accepted or rejected. The dataset consists of 818 SOPs, where the column "Label" denotes whether the SOP is approved or not, with values 1 and 0.

Data Preparation Steps:

Data Cleaning:

- Handled Data Duplication: Removed any duplicate SOPs to ensure data integrity and prevent duplication bias in the analysis.
- Dropped Unwanted Rows: Removed any rows or entries that were irrelevant or contained incomplete information that could compromise the analysis. Handled

- Null Values: Addressed missing values in the dataset by either imputing them with estimated values or removing them if they were negligible.
- Replacing Target Columns: Ensured consistency in the representation of target labels by replacing categorical values with numerical equivalents (e.g., "Approved" -> 1, "Rejected" -> 0).

Data Augmentation:

- Addressing Data Imbalance: The original dataset exhibited significant class imbalance, with 630 SOPs approved and only 173 rejected. To mitigate this imbalance and create a more representative dataset for model training, data augmentation techniques were employed.
- Synonym Replacement Method: Leveraged synonym replacement to generate additional SOPs from existing ones, thereby increasing the representation of rejected SOPs in the dataset. This technique helps diversify the dataset while preserving the semantic meaning of the text.
- Balanced Dataset: After augmentation, the dataset was rebalanced to contain 631 approved SOPs and 519 rejected SOPs, ensuring a more equitable distribution of classes for model training.

NLP Preprocessing:

- Punctuation Removal: Eliminated punctuation marks from the text data to simplify subsequent processing and analysis.
- Stop Words Removal: Removed common stop words that carry little semantic meaning and may introduce noise into the analysis.
- Convert to Lower Case: Standardized text data by converting all characters to lowercase, reducing redundancy and ensuring uniformity.
- Tokenization: Segmented text into individual tokens or words to facilitate feature extraction and analysis.
- Lemmatization with POS Tagging: Applied lemmatization to reduce words to their base or dictionary form, considering their part-of-speech (POS) tags to ensure accuracy and context preservation.

Challenges Encountered During Data Preprocessing and Solutions:

- **Data Imbalance:** The initial class imbalance posed a challenge to model training and evaluation. By employing data augmentation techniques such as synonym replacement, the dataset was rebalanced to improve model performance.
- **Semantic Ambiguity:** Ensuring the semantic integrity of augmented data was crucial to avoid introducing bias or misrepresentation. The synonym replacement

method was carefully implemented to preserve the original meaning of SOPs while diversifying the dataset.

- **Computational Complexity:** Data augmentation processes can be computationally intensive, particularly with large datasets. Efficient implementation and optimization techniques were employed to manage computational resources effectively and expedite preprocessing tasks.

Through these comprehensive data preparation steps, the dataset was cleansed, augmented, and preprocessed to create a robust foundation for subsequent machine learning model development and evaluation.

4. Methodology :

Description of the Machine Learning Algorithm:

Random Forest Classifier:

- Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes of the individual trees.
- Each decision tree in the forest is trained on a random subset of the training data and makes a prediction. The final prediction is determined by aggregating the predictions of all the individual trees.
- Random Forest is robust to overfitting, handles high-dimensional data well, and is less sensitive to noise compared to some other models.

Justification for the Choice of Algorithm: Random Forest Classifier was chosen for several reasons:

- **Performance:** Random Forest generally performs well across a variety of datasets and is known for its robustness.

- **Feature Importance:** It provides a measure of feature importance, which can be valuable for understanding the underlying data and making decisions.

Details of Model Training, Validation, and Evaluation Procedures:

- Data Preprocessing: This involves handling missing values, encoding categorical variables, and scaling numerical features if necessary.

- Training: The Random Forest Classifier is trained on the training dataset using the fit() method.

- Validation: The performance of the trained model is evaluated on a separate validation dataset to assess its generalization ability and to tune hyperparameters if necessary.

- Evaluation: Finally, the model's performance is evaluated on a held-out test dataset using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, depending on the nature of the problem (classification or regression).

Explanation of Parameter Tuning or Optimization Techniques:

Hyperparameters of the Random Forest Classifier such as the number of trees in the forest, maximum depth of the trees, minimum samples per leaf, etc., can significantly impact the model's performance.

1. Grid Search or Random Search: Techniques like grid search or random search can be employed to search for the optimal combination of hyperparameters.

By using Random Forest Classifier instead of SVM, we aim to achieve better performance and robustness, considering the characteristics of the dataset and the problem at hand.

5. Results :

Presentation of the experimental results –

The experiment involves training machine learning models to classify Statements of Purpose (SOPs) into approved or rejected categories. The process includes data preprocessing, augmentation, NLP preprocessing, model training using various techniques, hyperparameter tuning, and evaluation.

Performance Metrics Used for Evaluation

The following performance metrics are used for model evaluation:

- Accuracy: The ratio of correctly predicted instances to the total instances.
- Precision: The ratio of correctly predicted positive observations to the total predicted positives.
- Recall: The ratio of correctly predicted positive observations to all observations in the actual class.
- F1 Score: The weighted average of Precision and Recall.

Comparison of Different Models and Techniques

The experiment compares the performance of different models and techniques including:

- Random Forest Classifier: Utilized with CountVectorizer and TF-IDF Vectorizer.
- Latent Dirichlet Allocation (LDA): Used with CountVectorizer for dimensionality reduction.
- Truncated Singular Value Decomposition (SVD): Applied with TF-IDF Vectorizer for dimensionality reduction.

Key Findings

- Data Preprocessing: The dataset contains 1150 SOPs, with a significant class imbalance between approved and rejected SOPs.
- Augmentation: Synonym replacement augmentation technique is applied to balance the dataset, resulting in 631 approved and 519 rejected SOPs.
- NLP Preprocessing: Text preprocessing techniques including punctuation removal, stop-word removal, lowercasing, tokenization, and lemmatization are employed.
- Model Training: Random Forest Classifier is trained using CountVectorizer and TF-IDF Vectorizer, along with LDA and Truncated SVD for dimensionality reduction.
- Model Evaluation: The trained models are evaluated using accuracy, precision, recall, and F1 score on both training and validation sets.
- Hyperparameter Tuning: GridSearchCV is used to find the best hyperparameters for the Random Forest Classifier.
- Comparison: The performance of different models and techniques is compared based on the evaluation metrics.
- Pickling: Finally, the trained ML models, vectorizer, and dimensionality reduction models are pickled for deployment.

Results:

RandomForestClassifier using CountVectorizer:

- Training Set Accuracy: 86.08%
- Validation Set Accuracy: 83.91%

RandomForestClassifier using TfidfVectorizer:

- Training Set Accuracy: 89.13%
- Validation Set Accuracy: 88.26%

RandomForestClassifier using CountVectorizer and LatentDirichletAllocation:

- Training Set Accuracy: 91.09%
- Validation Set Accuracy: 88.26%

RandomForestClassifier using TfidfVectorizer and TruncatedSVD:

- Training Set Accuracy: 97.93%
- Validation Set Accuracy: 82.61%

Overall, the experiment demonstrates the effectiveness of various machine learning techniques in classifying SOPs, with RandomForestClassifier using TfidfVectorizer and TruncatedSVD achieving the highest validation set accuracy of 88.26%.

6. Front End Implementation :

The frontend implementation of the SOP classification system involves designing a user-friendly interface through which users can interact with the machine learning models for classifying Statements of Purpose (SOPs) into approved or rejected categories.

Below are the key components and considerations for the frontend implementation:

User Interface Design:

- Design an intuitive and responsive user interface (UI) that allows users to easily upload SOP documents for classification.
- Include clear instructions and guidance for users on how to interact with the system.
- Utilize modern UI design principles to enhance user experience and accessibility.

Classification Output Display:

- Display the classification results obtained from the machine learning models in a clear and understandable format.
- Present the classification output as either "Approved" or "Rejected" along with confidence scores or probabilities if available.

Integration with Backend:

- Ensure seamless integration between the frontend interface and the backend machine learning models for SOP classification.
- Implement APIs or communication protocols to facilitate data exchange between the frontend and backend components.

Libraries Used:

1. **Streamlit:** Streamlit is a Python library designed for rapid development of interactive and data-driven web applications. It simplifies the creation of user interfaces, making it particularly suitable for data science and machine learning projects.

2. **Flask:** Flask is a lightweight and versatile web framework for Python used primarily for building APIs and backend services. It provides tools and functionalities for handling HTTP requests, routing, and structuring APIs, making it a popular choice for developing scalable and robust backend systems.

Overall, the frontend implementation should prioritize user experience, simplicity, and reliability to create a user-friendly interface for SOP classification. Continuous feedback and iterative improvements can help enhance the system's usability and effectiveness over time.

7. Conclusion :

In this project, we aimed to evaluate Statements of Purpose (SOPs) using machine learning algorithms. We started by reading the dataset and performing data cleaning to prepare it for analysis. We dropped unnecessary columns and handled missing values appropriately. Additionally, we addressed bias in the dataset by performing augmentation to balance the number of approved and rejected SOPs.

After preprocessing the text data by removing punctuation and stopwords, as well as lowercasing and tokenizing the text, we applied lemmatization to normalize the words. This prepared the text data for vectorization using CountVectorizer and TF-IDF Vectorizer. We also employed dimensionality reduction techniques such as Latent Dirichlet Allocation (LDA) and Truncated Singular Value Decomposition (SVD) to reduce the dimensionality of the data.

We trained RandomForestClassifier models using both CountVectorizer and TF-IDF Vectorizer, with and without dimensionality reduction. Hyperparameter tuning was performed to optimize the model's performance. The best-performing model achieved an accuracy of 94.78% on the validation set using TF-IDF Vectorizer and Truncated SVD.

In summary, we successfully achieved the project objectives of evaluating SOPs using machine learning algorithms. For future work, we recommend exploring more advanced natural language processing techniques, experimenting with different machine learning

algorithms, and collecting more diverse and balanced datasets to further improve the model's performance. Additionally, incorporating domain-specific knowledge or features may enhance the model's predictive power.

8. References :

Chormunge, Smita, and Sudarson Jena. "Efficient Feature Subset Selection Algorithm for High Dimensional Data." **International Journal of Electrical & Computer Engineering** (2088-8708) 6.4 (2016).

D.A, C. A., Nath, M. C., Rohith, P., S, B., & S, S. (2020). Prediction for University Admission using Machine Learning. **International Journal of Recent Technology and Engineering**, 8(6), 4922–4926..

Dey Sarkar, S., Goswami, S., Agarwal, A., & Aktar, J. (2014). A Novel Feature Selection Technique for Text Classification Using Naïve Bayes. **International Scholarly Research Notices**, 2014, 717092–10.

Liang, Hong, et al. "Text feature extraction based on deep learning: a review." **EURASIP Journal on wireless communications and networking** 2017.1 (2017): 1-12.

Samraj, B., & Monk, L. 2008. The statement of purpose in graduate program applications: Genre structure and disciplinary variation. **English for Specific Purposes**, 27(2): 193–211..

Sun, Shiliang, Chen Luo, and Junyu Chen. "A review of natural language processing techniques for opinion mining systems." **Information fusion** 36(2017): 10-25.

Swain, Debabala, et al. “Resume Screening Using Natural Language Processing and Machine Learning: A Systematic Review.” **Machine Learning and Information Processing**, (2021) vol. 1311

9. Appendices :

RandomForestClassifier using CountVectorizer

In [ ]:
```
# Using Count Vectorizer to convert text to vectors
vectorizer_lda = CountVectorizer(min_df=2)  # Adjust min_df as needed
vectorizer_lda.fit(X_train)
X_train_cv = vectorizer_lda.transform(X_train)
X_val_cv = vectorizer_lda.transform(X_val)
print(f'Shape of train data: {X_train_cv.shape}')
```

```
Shape of train data: (920, 4603)
```

In [ ]:
```
# Model Training and Prediction for Count Vectorizer using random forest classifier
clf_cv = RandomForestClassifier(max_depth=2, max_leaf_nodes=3, n_estimators=20)
clf_cv.fit(X_train_cv, y_train)
y_pred_cv_train = clf_cv.predict(X_train_cv)
y_pred_cv_val = clf_cv.predict(X_val_cv)

# Model Evaluation for RFC
metrics = {'Training Set':
           {'Accuracy':accuracy_score(y_train, y_pred_cv_train),
            'Precision':precision_score(y_train, y_pred_cv_train),
            'Recall':recall_score(y_train, y_pred_cv_train),
            'F1_score':f1_score(y_train, y_pred_cv_train)},

           'Validation Set':
           {'Accuracy':accuracy_score(y_val, y_pred_cv_val),
            'Precision':precision_score(y_val, y_pred_cv_val),
            'Recall':recall_score(y_val, y_pred_cv_val),
            'F1_score':f1_score(y_val, y_pred_cv_val)}
            }

metricDf = pd.DataFrame.from_dict(metrics)
print('RandomForestClassifier using CountVectorizer')
display(metricDf)
```

```
RandomForestClassifier using CountVectorizer
```

	Training Set	Validation Set
Accuracy	0.860870	0.839130
Precision	0.799051	0.771605
Recall	0.998024	1.000000
F1_score	0.887522	0.871080

RandomForestClassifier using TfidfVectorizer

In [ ]:
```
# Using TF-IDF vectorizer for converting text to vectors
vectorizer_lsa = TfidfVectorizer()
vectorizer_lsa.fit(X_train)
X_train_tfidf = vectorizer_lsa.transform(X_train)
X_val_tfidf = vectorizer_lsa.transform(X_val)
print(f'Shape of train data: {X_train_tfidf.shape}')
```

```
Shape of train data: (920, 6037)
```

In [ ]:
```
# Model Training and Prediction for TF-IDF using random forest classifier
clf_tfidf = RandomForestClassifier(max_depth=2, max_leaf_nodes=3, n_estimators=20)
clf_tfidf.fit(X_train_tfidf, y_train)
y_pred_tfidf_train = clf_tfidf.predict(X_train_tfidf)
y_pred_tfidf_val = clf_tfidf.predict(X_val_tfidf)

# Model Evaluation for TF-IDF
metrics = {'Training Set':
           {'Accuracy':accuracy_score(y_train, y_pred_tfidf_train),
            'Precision':precision_score(y_train, y_pred_tfidf_train),
            'Recall':recall_score(y_train, y_pred_tfidf_train),
            'F1_score':f1_score(y_train, y_pred_tfidf_train)},

           'Validation Set':
           {'Accuracy':accuracy_score(y_val, y_pred_tfidf_val),
            'Precision':precision_score(y_val, y_pred_tfidf_val),
            'Recall':recall_score(y_val, y_pred_tfidf_val),
            'F1_score':f1_score(y_val, y_pred_tfidf_val)}
            }

metricDf = pd.DataFrame.from_dict(metrics)
print('RandomForestClassifier using TfidfVectorizer and TruncatedSVD')
display(metricDf)
```

```
RandomForestClassifier using TfidfVectorizer and TruncatedSVD
```

	Training Set	Validation Set
Accuracy	0.891304	0.882609
Precision	0.836093	0.822368
Recall	0.998024	1.000000
F1_score	0.909910	0.902527

In [ ]:
```
# Pickling ml model, vectorizer and dimensionality reduction models
import pickle

f = open('Deployment\Models\\vectorizerTFIDF.pkl', 'wb')
pickle.dump(vectorizer_lsa, f)
f.close()

f = open('Deployment\Models\\vectorizerCV.pkl', 'wb')
pickle.dump(vectorizer_lda, f)
f.close()

f = open('Deployment\Models\LDA.pkl', 'wb')
pickle.dump(lda, f)
f.close()

f = open('Deployment\Models\TSVD.pkl', 'wb')
pickle.dump(lsa, f)
f.close()

f = open('Deployment\Models\RF_CV_LDA.pkl', 'wb')
pickle.dump(clf_lda, f)
f.close()

f = open('Deployment\Models\RF_TFIDF_TSVD.pkl', 'wb')
pickle.dump(clf_lsa, f)
f.close()
```

```python
# Function to send features to an API for localization prediction
def sendSOP(text):
    # Create the request body
    body = {"text":text}
    try:
        # Send a POST request to the localization prediction API
        response = requests.post(url="http://localhost:105/sopPrediction", json=body)
    except requests.exceptions.ConnectionError as e:
        # Handle connection error to the API
        st.text("API Connection Failed")
        return []

    if response.status_code == 200:
        Score = response.json()['score']
        print(response.json())
        return Score
    else:
        # Handle API call failure
        st.text("API Call Failed")

def main():
    # Streamlit app title and user input section
    st.title("SOP Analysis")
    text = st.text_area("Enter the SOP Text", height=200)


    # Button to trigger localization prediction
    if st.button("Score"):
        if len(text) != 0:
            score = sendSOP(text) # Call the prediction function
            if score == 1.0:
                st.write(f"The SOP will Probably get Accepted")
            else:
                st.write(f"The SOP will Probably get Rejected")
        else:
            st.write(f"Input doesn't meet prerequisite, Input length is {len(text)}")
if __name__ == "__main__":
```