NATURAL LANGUAGE PROCESSING COURSE
FINAL PROJECT

Analyzing Last Words of Death Row Inmates using NLP techniques

Atiqurahman Mayar

CONTENTS

Introduction

- **Context**: The analysis of inmate last statements can provide valuable insights into the emotions and mental states of individuals on death row. This project aims to classify these statements based on the race of the inmates, providing a novel application of text classification techniques.

- **Primary Goal**: To classify the race of inmates using their last statements with Naive Bayes and CNN models. This helps understand how different racial groups express themselves in their final moments.

- **Methods Used**: Naive Bayes for its simplicity and CNN for its ability to capture complex patterns in text data.

Major Problem

- **Understanding Expressions**:
  - The primary problem is understanding and classifying the emotional and linguistic patterns in the last statements of inmates based on their race.

- **Challenges**:
  - Dealing with complex, unstructured text data and making sense of potentially ambiguous or emotionally charged statements.
  - Ensuring the model accurately captures any subtle differences in language use across different racial groups

- **Importance**:
  - Provides insights into how different racial groups express themselves under extreme stress.
  - Highlights potential disparities and biases in the criminal justice system, as seen through the lens of the inmates' final words.
  - This can inform future studies in linguistics, psychology, and criminal justice.

Purpose

- **Analytical Insight**: To gain a deeper understanding of how inmates of different races articulate their final words.
- **Technical Demonstration**: To compare the effectiveness of traditional machine learning (Naive Bayes) and deep learning (CNN) techniques in text classification tasks.
- **Social Impact**: To highlight potential disparities in how different racial groups are represented and understood in the criminal justice system.

Dataset Overview

Basic Information: The dataset consists of 545 observations with 21 variables. They are:

- Execution: The order of execution, numeric.
- LastName: Last name of the offender, character.
- FirstName: First name of the offender, character.
- TDCJNumber: TDCJ Number of the offender, numeric.
- Age: Age of the offender, numeric.
- Race: Race of the offender, categorical : Black, Hispanic, White, Other.
- CountyOfConviction: County of conviction, character.
- AgeWhenReceived: Age of offender when received, numeric.
- EducationLevel: Education level of offender, numeric.
- Native County: Native county of offender, categorical : 0 = Within Texas, 1= Outside Texas.
- PreviousCrime : Whether the offender committed any crime before, categorical: 0= No, 1= Yes.
- Codefendants: Number of co-defendants, numeric.
- NumberVictim: Number of victims, numeric.
- WhiteVictim, HispanicVictim, BlackVictim, VictimOtherRace. FemaleVictim, MaleVictim: Number of victims with specified demographic features, numeric.
- LastStatement: Last statement of offender, character.

Link to the Data Source: https://www.kaggle.com/datasets/mykhe1097/last-words-of-death-row-inmates

Dataset Overview

Why Choose this Data ?

1. **Diverse Attributes:** The dataset offers a mix of numeric, categorical, and textual data, allowing for a variety of analyses.
2. **Text Analysis Potential:** The "Last Statement" column is particularly suitable for NLP tasks such as sentiment analysis, text classification, or extracting patterns in the statements.
3. **Demographic and Physical Data:** These can be used to explore correlations and patterns, providing a multi-faceted approach to analysis.
4. **Sufficient Data:** With 545 records, there is a reasonable amount of data to train and validate models.

The analysis primarily focuses on the Last Statement and Race columns for classification purposes.

Data Attributes

- **Key Columns Used:**
  - **Date of Birth**: Helps in calculating the age at execution.
  - **Date of Execution**: Indicates when the execution took place.
  - **Race**: The target variable for classification.
  - **Last Statement**: The text data used as input for the models.
- **Additional Columns:**
  - **Date of Offence**: Provides context about the timing of the crime.
  - **Highest Education Level**: Offers insight into the educational background of the inmates.

Data Preprocessing

- **Handling Missing Values**: Missing values in Date of Birth are filled with Unknown and missing Last Statements are filled with an empty string.
- **Text Preprocessing**: Includes removing punctuation, converting text to lowercase, and tokenization.
- **Tokenization and Padding**: Necessary for CNN models to handle text input.

In [37]:
```
import pandas as pd
import numpy as np

# Load the Dataset
file_path = r'C:\Users\Atiq\Downloads\NLP final project\tx_deathrow_full.csv'
data = pd.read_csv(file_path, nrows=10000)
# data.head()
```

2- Data Preprocessing

In [16]:
```
# Fill missing values
data['Date of Birth'].fillna('Unknown', inplace=True)
data['Last Statement'].fillna('', inplace=True)

# Convert 'Date of Offence' to datetime and extract year
data['Date of Offence'] = pd.to_datetime(data['Date of Offence'], errors='coerce')
data['Year of Offence'] = data['Date of Offence'].dt.year

# Drop rows with invalid dates
data.dropna(subset=['Year of Offence'], inplace=True)
```

Research Overview

- **Definition of Text Classification**
  - **Text Classification**: The process of categorizing text into organized groups. It is a fundamental task in natural language processing (NLP) with applications in spam detection, sentiment analysis, and topic labeling.
- **Techniques Used**
  - **Naive Bayes**: A probabilistic classifier that applies Bayes theorem with strong independence assumptions between features.
  - **Convolutional Neural Networks (CNNs)**: A class of deep neural networks that can capture spatial hierarchies in data, making them well-suited for text and image classification.

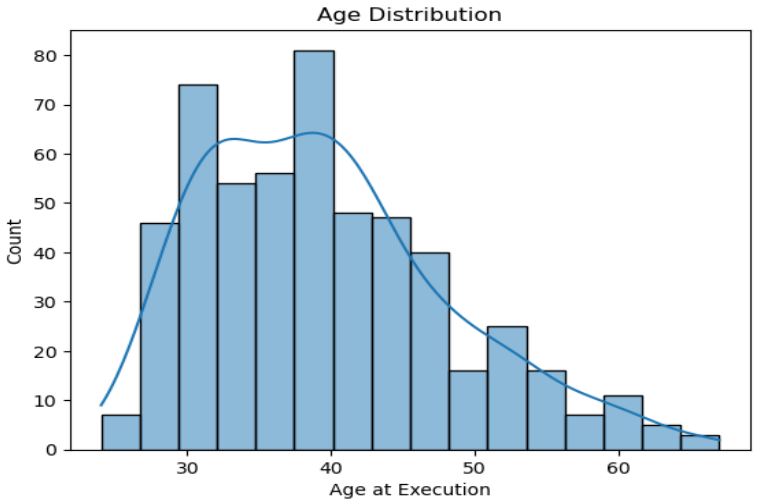
Questions Asked

- **Age Distribution**: What is the distribution of age at execution?
- **Education Distribution**: What is the distribution of the highest education level among inmates?
- **Race Classification**: Can we classify the race of inmates based on their last statements?

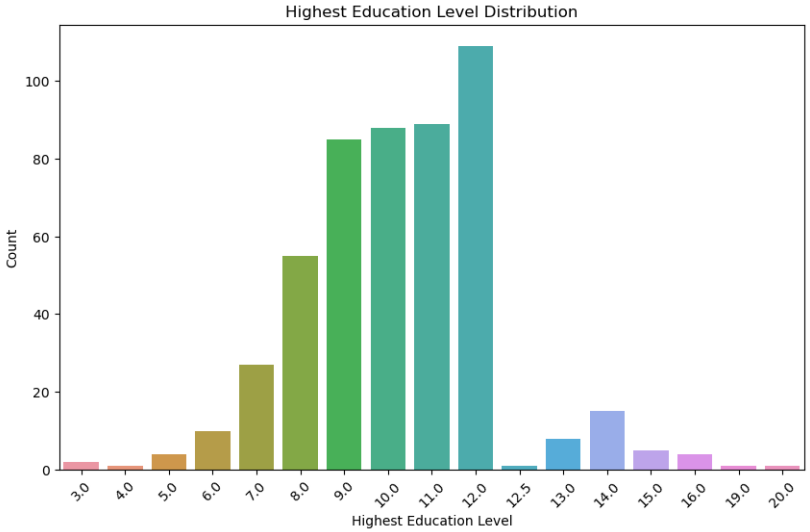
```
import seaborn as sns
import matplotlib.pyplot as plt

# Age Distribution
sns.histplot(data['Age at Execution'], kde=True)
plt.title('Age Distribution')
plt.show()

# Highest Education Level Distribution
plt.figure(figsize=(10, 6))
sns.countplot(x='Highest Education Level', data=data)
plt.title('Highest Education Level Distribution')
plt.xlabel('Highest Education Level')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Atiq\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Questions Asked



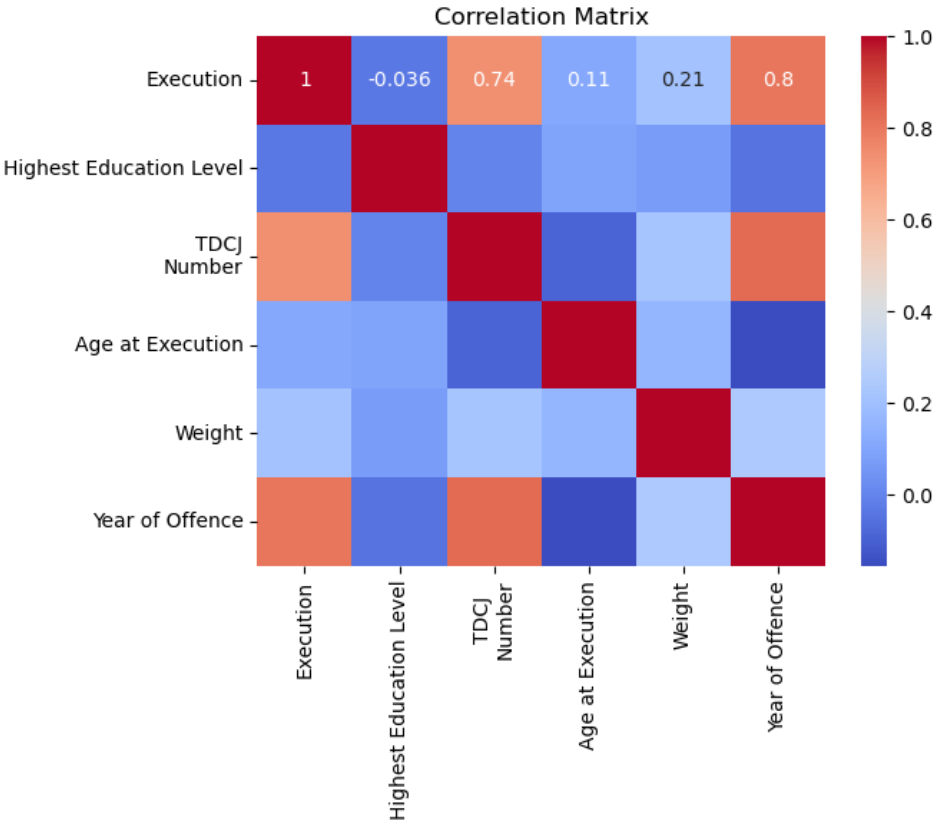
The age distribution graph for executions reveals that most occur between 35 and 45 years old, peaking around 40. The distribution is slightly skewed to the right, with fewer executions at older ages. Ages range from about 20 to 70 years, but executions are rare for those in their 20s or over 60. This pattern suggests that middle-aged individuals are most commonly executed, possibly due to various societal, legal, or health-related factors influencing the timing of executions.



The graph showing the highest education level distribution of executed individuals reveals that most had 12 years of education, equivalent to a high school diploma in the US. There's a sharp decline in frequency for education levels beyond high school, with very few individuals having higher education. The range spans from about 3 to 20 years of education, but frequencies decrease at higher levels. This pattern suggests that those executed typically had high school or lower education, possibly indicating a correlation between lower education levels and crimes leading to capital punishment, which may reflect broader socio-economic factors.

Questions Asked

Used correlation matrices to identify relationships between features.



The correlation matrix reveals strong positive relationships between execution and both TDCJ number (0.74) and year of offense (0.8). This suggests that more recent cases (indicated by higher TDCJ numbers) and offenses committed in later years are more likely to result in executions. There's a moderate correlation (0.53) between TDCJ number and year of offense, implying that more recent offenses tend to have higher TDCJ numbers. Other variables like education level, age at execution, and weight show weak or no correlations with the other factors, indicating they don't have significant linear relationships in this context.

Methodology and Techniques

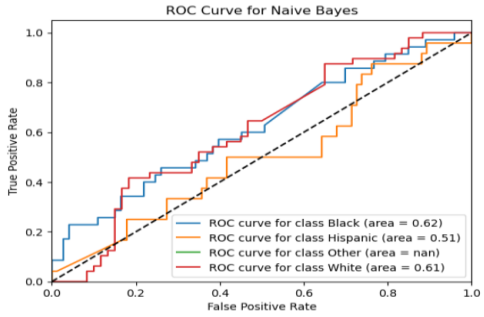
- **Naive Bayes**
  - **Text Vectorization**: Using TF-IDF to convert text data into numerical features.
  - **Training and Testing**: Split the dataset into training and testing sets.
  - **Model Performance**: Evaluated using accuracy, precision, recall, and F1-score.
  - **Confusion Matrix**: Visual representation of classification results.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score, roc
import re

# Text Preprocessing
def preprocess_text(text):
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = text.lower()  # Convert to lowercase
    return text

data['Last Statement'] = data['Last Statement'].apply(preprocess_text)

# Vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X = vectorizer.fit_transform(data['Last Statement'])
y = data['Race']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Naive Bayes Model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

# Evaluation for Naive Bayes
nb_accuracy = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Accuracy:", nb_accuracy)
print(classification_report(y_test, y_pred_nb))

# Confusion Matrix for Naive Bayes
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)
sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap='Blues')
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

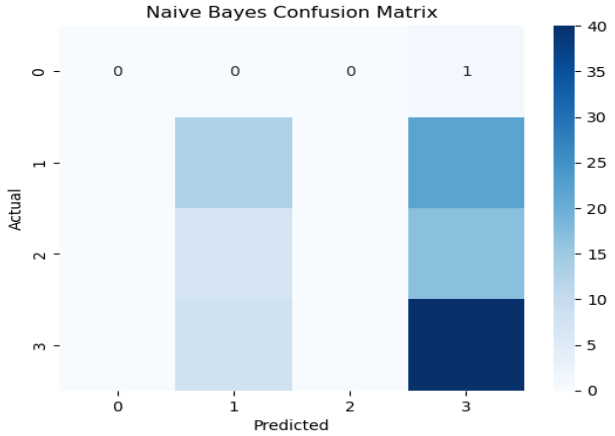
```
Naive Bayes Accuracy: 0.49074074074074076
              precision    recall  f1-score   support

       Asian       0.00      0.00      0.00         1
       Black       0.46      0.37      0.41        35
    Hispanic       0.00      0.00      0.00        24
       White       0.50      0.83      0.62        48

    accuracy                           0.49       108
   macro avg       0.24      0.30      0.26       108
weighted avg       0.37      0.49      0.41       108
```

```
# Calculate precision, recall, and F1-score for Naive Bayes
nb_precision = precision_score(y_test, y_pred_nb, average='weighted')
nb_recall = recall_score(y_test, y_pred_nb, average='weighted')
nb_f1 = f1_score(y_test, y_pred_nb, average='weighted')

print(f"Naive Bayes Precision: {nb_precision}")
print(f"Naive Bayes Recall: {nb_recall}")
print(f"Naive Bayes F1-Score: {nb_f1}")

# ROC Curve and AUC for Naive Bayes
y_pred_nb_proba = nb_model.predict_proba(X_test)

fpr = {}
tpr = {}
roc_auc = {}
for i in range(len(nb_model.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_pred_nb_proba[:, i], pos_label=nb_model.classes_[i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure()
for i in range(len(nb_model.classes_)):
    plt.plot(fpr[i], tpr[i], label=f'ROC curve for class {nb_model.classes_[i]} (area = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Naive Bayes')
plt.legend(loc='lower right')
plt.show()
```

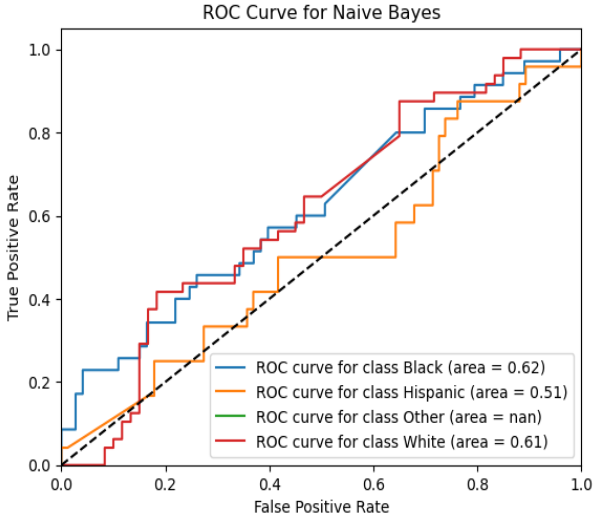
```
C:\Users\Atiq\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision is ill-def
ined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Atiq\anaconda3\Lib\site-packages\sklearn\metrics\_ranking.py:1029: UndefinedMetricWarning: No positive samples in y_tr
ue, true positive value should be meaningless
  warnings.warn(

Naive Bayes Precision: 0.3726851851851852
Naive Bayes Recall: 0.49074074074074076
Naive Bayes F1-Score: 0.4115226337448559
```



Methodology and Techniques



The Naive Bayes confusion matrix shows varying levels of prediction accuracy across different classes. Class 3 appears to have the highest prediction accuracy, with many instances correctly classified. Classes 1 and 2 show moderate accuracy, but with notable misclassifications - some class 1 instances are incorrectly labeled as class 2 or 3, while some class 2 instances are mislabeled as class 3. The model completely fails to correctly classify the single instance of class 0, misidentifying it as class 3. This suggests the model struggles most with distinguishing classes 0 and 2 from others, possibly due to feature overlap or insufficient distinguishing characteristics in the data used for these classes.



The ROC curve analysis for the Naive Bayes model shows poor to moderate performance across different classes. For the Black and White classes, the Area Under the Curve (AUC) values are 0.62 and 0.61 respectively, indicating slightly better than random performance. The Hispanic class has an AUC of 0.51, suggesting the model's predictions for this class are barely better than random guessing. No AUC value is provided for the Other class, likely due to insufficient data. Overall, the ROC curves are close to the diagonal line (which represents random guessing), indicating that the Naive Bayes classifier struggles to effectively distinguish between these classes based on the given features.

Methodology and Techniques

- **CNN**
  - **Tokenization and Padding**: Converting text data into sequences and ensuring uniform length.
  - **CNN Architecture**: Embedding layer, Conv1D, GlobalMaxPooling, Dense layers.
  - **Training Process**: Model training with early stopping to prevent overfitting.
  - **Model Performance**: Evaluated using accuracy, precision, recall, and F1-score.

```
# CNN predictions
y_pred_cnn_proba = cnn_model.predict(X_test_seq)
y_pred_cnn = (y_pred_cnn_proba > 0.5).astype(int)

from sklearn.metrics import precision_recall_fscore_support

# Calculate precision, recall, and F1-score for CNN
cnn_precision, cnn_recall, cnn_f1, _ = precision_recall_fscore_support(y_test_seq, y_pred_cnn, average='weighted', zero_division=

print(f"CNN Precision: {cnn_precision}")
print(f"CNN Recall: {cnn_recall}")
print(f"CNN F1-Score: {cnn_f1}")

from sklearn.metrics import roc_curve, auc

# ROC Curve and AUC for CNN (handling multiclass)
n_classes = len(np.unique(y_test_seq))

# Binarize the output
from sklearn.preprocessing import label_binarize
y_test_binarized = label_binarize(y_test_seq, classes=np.arange(n_classes))
y_pred_cnn_proba_binarized = np.hstack([(y_pred_cnn_proba == i).astype(int) for i in range(n_classes)])

# Compute ROC curve and ROC area for each class
fpr_cnn = dict()
tpr_cnn = dict()
roc_auc_cnn = dict()
for i in range(n_classes):
    fpr_cnn[i], tpr_cnn[i], _ = roc_curve(y_test_binarized[:, i], y_pred_cnn_proba_binarized[:, i])
    roc_auc_cnn[i] = auc(fpr_cnn[i], tpr_cnn[i])

# Plot ROC curves for each class
plt.figure()
for i in range(n_classes):
    plt.plot(fpr_cnn[i], tpr_cnn[i], label=f'ROC curve for class {i} (area = {roc_auc_cnn[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for CNN')
plt.legend(loc='lower right')
plt.show()

# Print ROC AUC for each class
for i in range(n_classes):
    print(f"Class {i} ROC AUC: {roc_auc_cnn[i]:.2f}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
CNN Precision: 0.10502400548696844
CNN Recall: 0.32407407407407407
CNN F1-Score: 0.15863765863765864
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import LabelEncoder

# Label Encoding
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Tokenization and Padding
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(data['Last Statement'])
sequences = tokenizer.texts_to_sequences(data['Last Statement'])
X_seq = pad_sequences(sequences, maxlen=100)

# Train-test split
X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(X_seq, y_encoded, test_size=0.2, random_state=42)

# Build CNN Model
cnn_model = Sequential([
    Embedding(input_dim=5000, output_dim=128, input_length=100),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(10, activation='relu'),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = cnn_model.fit(X_train_seq, y_train_seq, epochs=10, batch_size=10, validation_split=0.2, callbacks=[early_stopping])

# Evaluate CNN Model
cnn_accuracy = cnn_model.evaluate(X_test_seq, y_test_seq)[1]
print("CNN Accuracy:", cnn_accuracy)
```

```
Epoch 1/10
C:\Users\Atiq\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprec
ated. Just remove it.
  warnings.warn(
35/35 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.2911 - loss: -1.6546 - val_accuracy: 0.3023 - val_loss: -44.1973
Epoch 2/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.4157 - loss: -100.8101 - val_accuracy: 0.3023 - val_loss: -627.9985
Epoch 3/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.3912 - loss: -949.1670 - val_accuracy: 0.3023 - val_loss: -3516.3767
Epoch 4/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.3287 - loss: -4674.1709 - val_accuracy: 0.3023 - val_loss: -11766.3164
Epoch 5/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.3827 - loss: -13123.1963 - val_accuracy: 0.3023 - val_loss: -28994.2031
Epoch 6/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.3494 - loss: -34266.4883 - val_accuracy: 0.3023 - val_loss: -62174.6211
Epoch 7/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.3343 - loss: -70894.0156 - val_accuracy: 0.3023 - val_loss: -115899.8828
Epoch 8/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.3908 - loss: -119034.3672 - val_accuracy: 0.3023 - val_loss: -201834.6562
Epoch 9/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.4116 - loss: -188725.2812 - val_accuracy: 0.3023 - val_loss: -331134.0938
Epoch 10/10
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.3711 - loss: -349615.0312 - val_accuracy: 0.3023 - val_loss: -513665.9688
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3307 - loss: -425662.4688
CNN Accuracy: 0.32407405972480774
```

Results and Comparison

- **Comparison of Naive Bayes and CNN Models**
  - **Naive Bayes Accuracy:** 0.49
  - **CNN Accuracy**: 0.32
  - **Naive Bayes Precision**: 0.37
  - **Naive Bayes Recall**: 0.49
  - **Naive Bayes F1-Score**: 0.41
  - **CNN Precision**: 0.10
  - **CNN Recall:** 0.32
  - **CNN F1-Score:** 0.15

- **Analysis for Niave Bayes Accuracy:**
  - **Accuracy**: The Naive Bayes model has an accuracy of 49.07%, which is significantly higher than the CNN model.
  - **Precision**: Precision of 37.27% indicates that out of all predicted positive labels, 37.27% are true positives.
  - **Recall**: A recall of 49.07% means that the model correctly identifies 49.07% of all actual positive labels.
  - **F1-Score**: The F1-Score of 41.15% balances precision and recall, providing a single metric that accounts for both false positives and false negatives.

Analysis for CNN Accuracy:

- **Accuracy**: The CNN model has an accuracy of 32.41%, which is lower than the Naive Bayes model, indicating it correctly classifies 32.41% of instances.
- **Precision**: With a precision of 10.50%, the CNN model struggles with false positives, indicating it correctly identifies 10.50% of all predicted positive labels.
- **Recall**: A recall of 32.41% shows that the CNN model correctly identifies 32.41% of actual positive labels.
- **F1-Score**: The F1-Score of 15.86% is lower than that of the Naive Bayes model, indicating a poorer balance between precision and recall.

Conclusion

- **Impact on NLP Projects**:
  - This project demonstrates the importance of choosing the right model and preprocessing techniques for text classification tasks in NLP projects.
  - While Naive Bayes offers a quick and efficient baseline, CNNs have the potential for deeper insights, making them suitable for more complex analyses.

- **Societal Implications**:
  - Analyzing last statements of death row inmates can provide insights into their state of mind, social backgrounds, and potential systemic biases.
  - Understanding these factors can inform policy changes, improve psychological support for inmates, and address racial and educational disparities in the criminal justice system.

- **Future Work**:
  - **Data Enhancement**: Incorporating more data and refining preprocessing steps (e.g., advanced tokenization, better handling of stop words) can improve model performance.
  - **Model Optimization**: Further tuning of CNN hyperparameters and experimenting with different architectures could yield better results.
  - **Feature Analysis**: Investigating the impact of specific features, such as education level and age, on model predictions can provide deeper insights into the factors influencing the last statements.


Danke!