**Building an AI-Generated Text Classifier using Naive Bayes**

In this comprehensive guide, we will walk through the process of building a text classifier to distinguish between human-generated and AI-generated essays using the Naive Bayes algorithm. The goal is to create a model that can accurately predict the origin of an essay based on its content.

# Step 1: Data Preparation

The first step involves loading and preparing the data. We begin by loading the training dataset, which typically contains a mix of human-generated and AI-generated essays. In our case, we remove unnecessary columns such as 'ID' and 'prompt\_id' and then divide the data into training and development sets.

1. import pandas as pd

2. from sklearn.model\_selection import train\_test\_split

3.

4. # Load data

5. train\_essays = pd.read\_csv('train\_essays.csv')

6.

7. # Drop unnecessary columns

8. train\_essays = train\_essays.drop(['prompt\_id', 'id'], axis=1)

9.

10. # Divide into train and dev sets

11. train\_data, dev\_data = train\_test\_split(train\_essays, test\_size=0.2, random\_state=42)

# Step 2: Balancing the Dataset

1. print(train\_data['generated'].value\_counts())

Output:

0 1100

1 2

Name: generated, dtype: int64

Dataset balancing is crucial for training a model that can generalize well. If there is an imbalance between the number of human and AI-generated essays, the model might be biased towards the majority class. In this step, we ensure a balanced dataset by adding more AI-generated essays. This can be achieved by using an external source or generating additional data.

1. ai\_generated\_data = pd.read\_csv('ai\_generated\_data.csv')

2. df = pd.concat([train\_data, ai\_generated\_data], ignore\_index=True)

# Step 3: Vocabulary and Word Probabilities

Building a vocabulary is fundamental for text classification. We create a list of all words in the dataset and count the occurrences of each word. The vocabulary is then constructed by selecting words with a sufficient frequency. We also calculate the probabilities of each word occurring in the dataset.

1. from collections import Counter

2.

3. # Create a list of all words in the dataset

4. all\_words = ' '.join(df['text']).lower().split()

5.

6. # Count word occurrences

7. word\_counts = Counter(all\_words)

8.

9. # Build vocabulary

10. vocab = [word for word, count in word\_counts.items() if count >= 5]

11.

12. # Calculate P[word]

13. num\_documents = len(df)

14. word\_probabilities = {word: count / num\_documents for word, count in word\_counts.items()}

15.

16. # Separate human and LLM data

17. human\_data = df[df['generated'] == 0]

18. llm\_data = df[df['generated'] == 1]

19.

20. # Calculate P[word | LLM]

21. llm\_word\_probabilities = {word: df['text'].apply(lambda essay: word in essay.lower()).mean() for word in vocab}

print(llm\_word\_probabilities)

Output:

'cars,': 0.2044653349001175,

'they': 0.7914218566392479,

'make': 0.554641598119859,

'life': 0.35663924794359575,

'so': 0.9876615746180963,

'much': 0.43772032902467684,

'easier,': 0.010575793184488837,

'or,': 0.2009400705052879,

'do': 0.8584018801410106,

'them': 0.5282021151586369,

…

}

# Step 4: Naive Bayes Classifier

Now, we implement the Naive Bayes classifier. The Naive Bayes algorithm is a probabilistic algorithm based on Bayes' theorem. We calculate the probability of each class (human or AI-generated) given the words in an essay.

We classify essays based on word probabilities and Laplace smoothing.

1. import numpy as np

2.

3. # Laplace Smoothing function

4. def laplace\_smoothing(count, total\_count, vocab\_size, alpha=1):

5.     return (count + alpha) / (total\_count + alpha \* vocab\_size)

6.

7. # Implement Naive Bayes Classifier with Laplace Smoothing

8. def classify\_essays\_with\_smoothing\_probabilities(essays, word\_probabilities, llm\_word\_probabilities, vocab, alpha=1):

9.     probabilities = []

10.

11.     for essay in essays:

12.         # Initialize probabilities

13.         human\_prob = 0.0

14.         llm\_prob = 0.0

15.

16.         for word in essay.lower().split():

17.             if word in vocab:

18.                 human\_prob += np.log(laplace\_smoothing(word\_probabilities[word], len(train\_data), len(vocab), alpha))

19.                 llm\_prob += np.log(laplace\_smoothing(llm\_word\_probabilities[word], len(llm\_data), len(vocab), alpha))

20.

21.         # Apply sigmoid function to get probabilities in decimal

22.         human\_prob = 1 / (1 + np.exp(-human\_prob))

23.         llm\_prob = 1 / (1 + np.exp(-llm\_prob))

24.

25.         probabilities.append({'human\_prob': human\_prob, 'llm\_prob': llm\_prob})

26.

27.     return probabilities

28.

29. # Apply the classifier with Laplace smoothing on the dev dataset

30. dev\_probabilities\_with\_smoothing = classify\_essays\_with\_smoothing\_probabilities(dev\_data['text'], word\_probabilities, llm\_word\_probabilities, vocab, alpha=1)

# Step 5: Evaluation and Top Words

Evaluate the model accuracy on the development set and find the top words predicting each class.

1. # Convert probabilities to binary predictions

2. dev\_predictions\_with\_smoothing = [1 if prob['llm\_prob'] > prob['human\_prob'] else 0 for prob in dev\_probabilities\_with\_smoothing]

3.

4. # Calculate accuracy

5. accuracy\_with\_smoothing = (dev\_predictions\_with\_smoothing == dev\_data['generated']).mean()

6. print(f"Accuracy on dev dataset with Laplace smoothing: {accuracy\_with\_smoothing}")

7.

8. # Top 10 words predicting each class

9. top\_10\_human\_words = sorted(vocab, key=lambda word: -np.log(laplace\_smoothing(word\_probabilities[word], len(train\_data), len(vocab))))

10. top\_10\_llm\_words = sorted(vocab, key=lambda word: -np.log(laplace\_smoothing(llm\_word\_probabilities[word], len(llm\_data), len(vocab))))

11.

12. print("\nTop 10 words predicting Human essays:")

13. print(top\_10\_human\_words[:10])

14.

15. print("\nTop 10 words predicting LLM essays:")

16. print(top\_10\_llm\_words[:10])

17.

Output:

Accuracy on dev dataset with Laplace smoothing: 0.6598240469208211

Top 10 words predicting Human essays:

['the', 'to', 'of', 'and', 'a', 'in', 'is', 'that', 'for', 'it']

Top 10 words predicting LLM essays:

['the', 'a', 'he', 'i', 'u', 't', 'c', 'to', 'in', 'and']

# Step 6: Kaggle Submission

Finally, apply the classifier on the test dataset using optimal hyperparameters and submit the results to Kaggle.

1. test\_data = pd.read\_csv('test\_essays.csv')

2. test\_predictions = classify\_essays\_with\_smoothing\_probabilities(test\_data['text'], word\_probabilities, llm\_word\_probabilities, vocab, alpha=1)

3.

4. prob\_ = []

5. for dict1 in test\_predictions:

6. prob\_.append(dict1['llm\_prob'])

7.

8. # Prepare the Kaggle submission

9. kaggle\_submission = pd.DataFrame({'id': test\_data['id'], 'generated': prob\_})

10.

11. # Save the submission to a CSV file

12. kaggle\_submission.to\_csv('kaggle\_submission.csv', index=False)

13.

14. # Print the submission

15. print(kaggle\_submission)

# Conclusion

This project demonstrates how a simple Naive Bayes classifier can serve as an effective approach for identifying AI-generated text from human writing. With more training data and refinements to the features and model, the accuracy would likely improve further. This provides a strong baseline for text classification tasks. This step-by-step guide provides an overview of building a text classifier to distinguish between human and AI-generated essays. Adjustments and improvements can be made based on specific requirements and feedback from model evaluations.

# References:

[1] Pandas read\_csv documentation (https://pandas.pydata.org/docs/reference/api/pandas.read\_csv.html)

[2] Combining DataFrames with pandas concat (https://pandas.pydata.org/pandas-docs/stable/user\_guide/merging.html)

[3] Scikit-learn model validation documentation (https://scikit-learn.org/stable/modules/cross\_validation.html)

[4] Feature extraction from text (https://monkeylearn.com/blog/entity-extraction/)

[5] Laplace smoothing for text classification (https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-classifier-using-laplace-smoothing/)

[6] Naive Bayes algorithm (https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c)

[7] Applying Naive Bayes to NLP tasks (https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a)