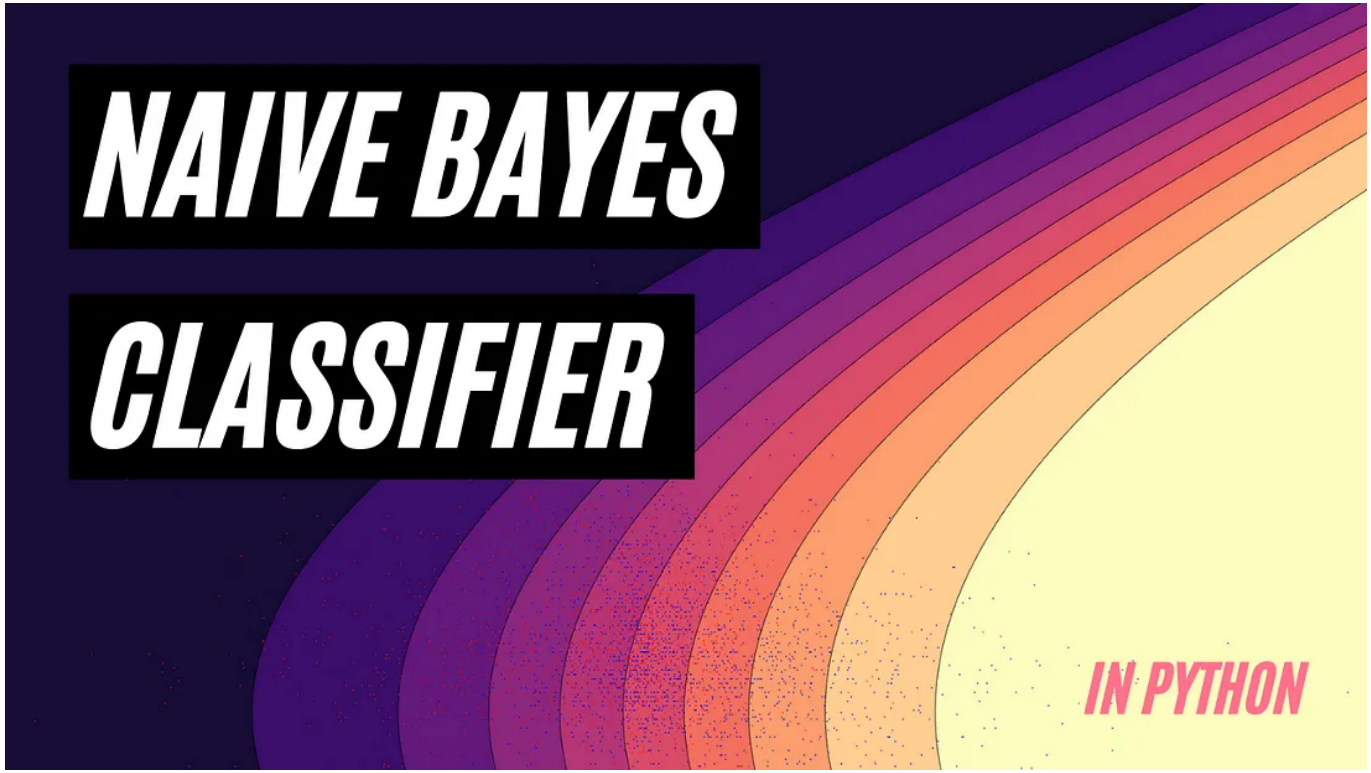
Revealing the Authors: An Exploration of Language Model vs. Human Text Recognition Using Naive Bayes Classifier

Keerthi Raj

7 min read · 7 hours ago

5

The “Naive-Bayes Classification” is a frequently used classification concept that is introduced and implemented through this blog. The written code is executed without utilizing the built-in naive bayes function.

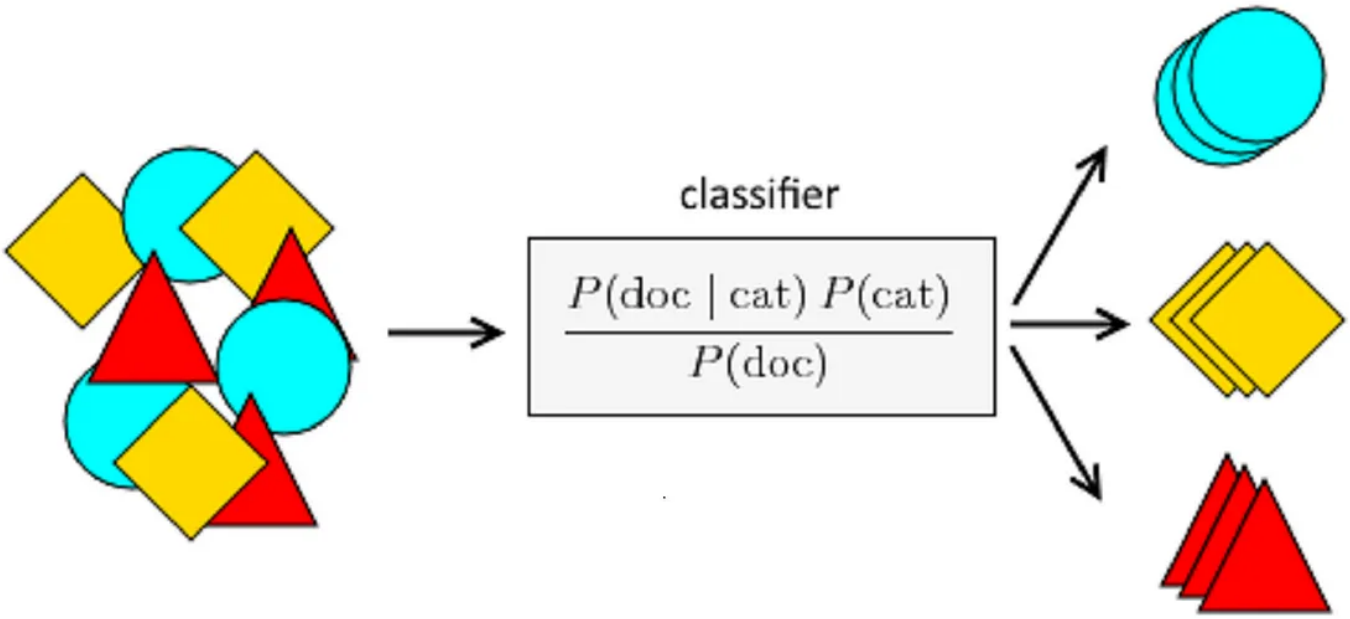


INTRODUCTION

Language models are becoming more complex in the rapidly developing field of artificial intelligence, making it harder to distinguish between text produced by machines and material authored by humans. It becomes crucial to distinguish between the two as we go through the digital world for a number of reasons, such as information integrity and content moderation. We'll examine how the Naive Bayes Classifier is used to determine the source of a text in this blog article, illuminating the minute details that set language models (LLMs) apart from human writers.

WHAT IS NAIVE BAYES CLASSIFIER 🤔

Based on Bayes' theorem, the Naive Bayes Classifier is a probabilistic algorithm. Because it assumes feature independence, text classification jobs are a particularly good fit for it. Through the use of statistical probabilities and patterns, this classifier may be trained to identify unique linguistic characteristics that differentiate LLM-generated content from that created by humans.



In a "cat vs. dog" classifier, the goal is to determine whether an input image contains a cat or a dog. For simplicity, let's consider a binary classification

scenario where the only two possible classes are “cat” and “dog.”

Bayes’ Theorem

Bayes’ theorem is a fundamental concept in probability theory, and it is expressed as:

$$\mathbf{P(A|B)= (P(B|A)\cdot P(A))/}P(A|B)=P(B)P(B|A)\cdot P(A)$$

In the context of a classifier:

- $P(A|B)$ is the probability that the image contains a cat (or dog) given the observed features (pixels in the image).
- $P(B|A)$ is the probability of observing the features given that the image contains a cat (or dog).
- $P(A)$ is the prior probability of an image containing a cat (or dog).
- $P(B)$ is the probability of observing the features across all possible classes.

Assumption

Given the class label, features are assumed to be conditionally independent, which gives rise to the word “naive” in Naive Bayes. This indicates that, given

the class label (cat or dog), an image's existence or absence in the context of an image classifier is independent of the presence or absence of any other image. The algorithm frequently performs very well, despite the fact that this assumption is rarely true in real life.

DATASET

I combined many datasets to make sure that our Naive Bayes Classifier had a complete set of examples that included both text produced by humans and artificial intelligence. The combined dataset is intended to be varied, encompassing a range of themes, genres, and writing styles. For supervised learning to be possible, each text sample must have annotated labels indicating the source (human or AI).

To increase the number and diversity of instances, I have carefully combined other datasets in light of the competition's scant data on essays created by AI. A link to the dataset is provided in the reference section, offering transparency into the data sources that were used for this project. By preparing the dataset with such care, the classifier should be better able to identify and generalize patterns.

Rather than separating the dataset as I usually would, I have also created a development dataset.

```
import pandas as pd
data = pd.read_csv('train_essays.csv')
development_data = pd.read_csv('development.csv')
```

FEATURE EXTRACTION USING VOCABULARY AND REVERSE INDEX

In this section, we'll dive into the essential steps of building a vocabulary and creating a reverse index, crucial components for extracting features that empower the Naive Bayes Classifier to distinguish between human and AI-generated text.

BUILDING VOCABULARY

The first stage in our feature extraction journey is to build a comprehensive vocabulary that is suited to the intricacies of our text data. Let's look at the code more closely:

```
import re
from collections import Counter

def buildVocabulary(texts, occurence=5):
    all_text = ' '.join(texts)
    words = re.findall(r'\b\w+\b', all_text.lower())
    word_counts = Counter(words)
```

```
    vocabulary = [word for word, count in word_counts.items() if count >= occure
    return vocabulary
```

The buildVocabulary function in this example combines the input sentences into a single string. The text is tokenized into words using regular expressions, guaranteeing consistency by turning everything to lowercase. The Counter class then counts the number of times a word appears, and a minimum threshold (min_occurrence) filters out less frequent terms, resulting in a more refined lexicon.

IMPORTANCE

This created vocabulary is more than just a collection of words; it forms the foundation of our feature extraction process. It incorporates the distinct set of terms found in our training data, allowing for a more sophisticated view of language diversity. Whether or not a word is included in this vocabulary determines its relevance as a feature during classification.

REVERSE INDEX

The next critical step is to create a reverse index. This index provides a dynamic mapping of words to their vocabulary indexes. Let's have a look at how it's done:

```
def reverseIndex(vocabulary):
    reverse_index = {word: index for index, word in enumerate(vocabulary)}
    return reverse_index
```

The create_reverse_index method cleverly assigns a unique index to each word, setting the framework for efficient computations during both the training and prediction stages.

IMPORTANCE

The reverse index speeds up information retrieval during the Naive Bayes Classifier's decision-making procedures by tying each word to a specific index.

MODEL TRAINING

Now that we've carefully built our vocabulary and established a reverse index, we're ready to dive into the heart of our Naive Bayes Classifier: model training. In this phase, we will compute occurrence and conditional probabilities, which will allow our model to make intelligent predictions about the authorship of a particular text.

CALCULATE OCCURRENCE PROBABILITY

```
def occurrenceProbability(word, all_documents):
    total_words = sum(1 for doc in all_documents if word in doc)
    total_documents = len(all_documents)
    return total_words / total_documents
```

Statistically measure the prevalence of a given word (`num_documents_with_word`) over our entire dataset. This result is then normalized by dividing it by the total number of documents (‘total_documents’), producing the chance of that word appearing.

CALCULATE CONDITIONAL PROBABILITY

```
def conditionalProbability(word, class_documents, all_documents):
    total_words = sum(1 for doc in class_documents if word in doc)
    total_documents = len(class_documents)
    return {word: total_words / total_documents} if total_documents > 0 else {}
```

This function accurately calculates the number of documents (`num_class_documents_with_word`) that contain the given word within a given class. The conditional probability can be obtained by normalizing this value

by the total number of documents in that class. The end product is a dictionary that contains each word's conditional probability.

MODEL EVALUATION

It's time to evaluate our Naive Bayes Classifier's ability to discriminate between text created by AI and human authors now that it has probabilities from the training phase at its disposal. To evaluate the performance of the model, we will examine the crucial phases of prediction and accuracy computation in this section.

PREDICT CLASS

This function intelligently computes log probabilities for each class, considering the occurrence and conditional probabilities learned during training. The prediction is based on which class exhibits higher probability.

```
import math
def predict(document, vocabulary, human_occurence, ai_occurence, human_condition
    words = re.findall(r'\b\w+\b', document.lower())
    probability_human = math.log(human_occurence)
    probability_ai = math.log(ai_occurence)

    for word in words:
        if word in vocabulary:
            conditional_probs_word_human = human_conditional.get(word, 0)
```

```
            conditional_probs_word_llm = ai_conditional.get(word, 0)
            probability_human += math.log(conditional_probs_word_human)
            probability_ai += math.log(conditional_probs_word_llm)

    return "0" if probability_human > probability_ai else "1"
```

CALCULATE ACCURACY

```
def calculate_accuracy(dev_documents, dev_labels, vocabulary, human_occurence,
    correct_predictions = 0

    for doc, label in zip(dev_documents, dev_labels):
        predicted_class = predict(doc, vocabulary, human_occurence, ai_occurence
        if predicted_class == label:
            correct_predictions += 1

    accuracy = correct_predictions / len(dev_documents)
    return accuracy
```

This function iterates through the validation set, comparing the model's predictions against the actual labels. The accuracy is then calculated as the ratio of correct predictions to the total number of documents in the validation set.

HYPERPARAMETER TUNING

```
smoothing_parameters = [0.1, 1, 5, 10]

for smoothing_parameter in smoothing_parameters:
    conditional_probs_human_smoothed = {word: conditionalProbabilitySmoothed(wor
    conditional_probs_llm_smoothed = {word: conditionalProbabilitySmoothed(word,
    accuracy = calculate_accuracy(dev_documents, dev_labels, vocabulary, human_c
    print(f"Smoothing Parameter: {smoothing_parameter}, Accuracy: {accuracy}")
```

Hyperparameter tuning takes center stage as we explore the impact of Laplace smoothing on our model's accuracy. By varying the smoothing parameter across different values, we can observe how it influences the overall performance of the Naive Bayes Classifier.

TOP 10 WORDS PREDICTING EACH CLASS

```
top_words_human = topWords(human_conditional, vocabulary)
top_words_ai = topWords(ai_conditional, vocabulary)
```

```
print("Top words for Human-generated essays:", top_words_human)
print("Top words for AI-generated essays:", top_words_ai)
```

CONCLUSION

Finally, using probabilities as a guide and careful fine-tuning, our Naive Bayes Classifier performs exceptionally well in differentiating text written by AI/humans. The interpretability of distinguishing words broadens our comprehension and opens us new possibilities in the always changing field of text authorship identification.

Accuracy was average of 87% for alpha values [0.1, 1, 5, 10]. You can check for other alpha values by downloading the code from my Github link or code link given below.

```
Smoothing Parameter: 0.1, Accuracy: 0.8709981167608286
Smoothing Parameter: 1, Accuracy: 0.8662900188323918
```

CONTRIBUTIONS

1. Took help of reference codes to write the Naive Bayes Classifier from scratch.

2. Hyperparameter — running the code with multiple alpha values to check for accuracy, but there was only a minute difference.
3. Code was optimal for small size data, but when code was handling 100,000 rows it took nearly 50+ minutes to execute. Now I have reduced the dataset. Dataset was created from Chatgpt by giving prompts.

REFERENCE

1. I have referred the code on NaiveBayesClassifier from https://scikit-learn.org/stable/modules/naive_bayes.html
2. Thanks to https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/ they have explained the concepts in a easy way.
3. https://www.ibm.com/topics/naive-bayes — Basic Idea and implementation.

DATASETS

LLM - Detect AI Generated Text

Identify which essay was written by a large language model

www.kaggle.com

🤖 **LLM Generated Essays for the Detect AI Comp!**

Kaggle is the world's largest data science community with powerful tools and resources to help you achieve your data...

www.kaggle.com

LLM: 7 prompt training dataset

(for use in the LLM - Detect AI Generated Text competition)

www.kaggle.com

CODE

https://github.com/KR-16/NAIVE-BAYES-CLASSIFIER/blob/master/model1.ipynb

Naive Bayes Classifier | Probability | Machine Learning | Text Classification | Nltk

Written by Keerthi Raj

Edit profile

1 Follower

https://keerthiraj.netlify.app/

More from Keerthi Raj

Keerthi Raj

PETAL FLOWERS CLASSIFICATION WITH USING TPU

Kaggle Competition Petals to Metals Flowers Classification

Keerthi Raj

Titanic: Machine Learning from Disaster

Unraveling insights from the Titanic Dataset

14 min read · Nov 12

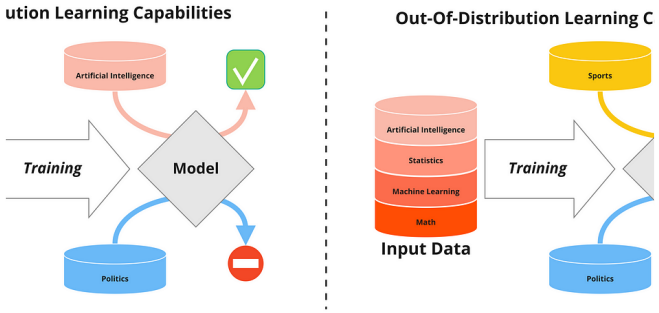
21 1

12 min read · Oct 28

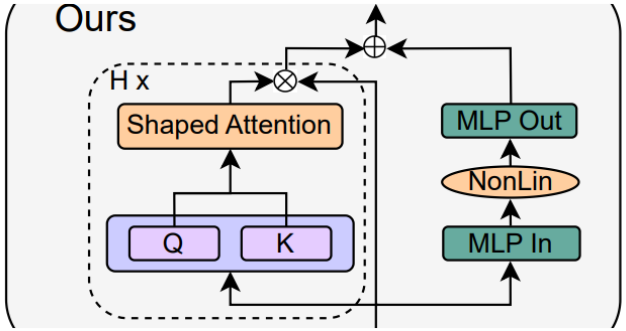
7

See all from Keerthi Raj

Recommended from Medium



Benjamin Thürer in Towards Data Science



Freedom Preetham in Autonomous Agents

Pre-Training Context is All You Need

The driving force behind modern transformer models stems to a large extent from its...

✦ · 6 min read · 6 days ago

161 4

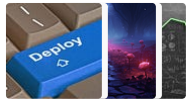
Simplifying Transformer Blocks — A Detailed Mathematical...

Large language models (LLMs) can expand their capabilities through various scaling...

16 min read · 4 days ago

166

Lists



Predictive Modeling w/ Python

20 stories · 659 saves

Practical Guides to Machine Learning

10 stories · 739 saves

Natural Language Processing

924 stories · 441 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 226 saves

Thomas Smith in The Generator

Did OpenAI Secretly Create a Brain-Like Intelligence After All?

What We Know About Q* and OpenAI's Potential AGI Breakthrough

✦ · 5 min read · Nov 26

3.2K 60

Anthony Alca... in Artificial Intelligence in Plain En...

The Silent Information Transformation with LLMs:...

As AI permeates across industries, a silent transformation is underway in how AI...

✦ · 14 min read · Nov 26

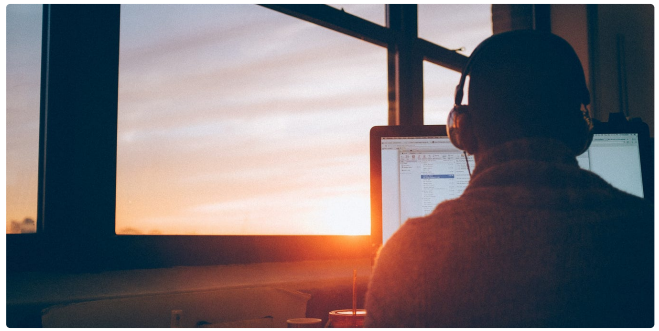
229 1

Oops!

Access is temporarily unavailable, please try again later.

Please contact us through our help center if this issue persists.

Jeremy Arancio in Towards AI



Alex Miguel Meyer in Better Marketing

Why are AI Products Doomed to Fail?

After one year of implementing AI features for various businesses, I share my perspective ...

· 17 min read · Nov 17

1.2K 34

How to Analyse Complex Business Problems Like Top Strategy...

A practical guide to planning and conducting analyses for a market expansion strategy

· 6 min read · 3 days ago

477 5

See more recommendations