NAIVE-BAYES-CLASSIFIER / model1.ipynb

KR-16 committed c3325ea · 28 minutes ago History

Preview Code Blame 304 lines (304 loc) · 8.51 KB Raw

Step 1: Load and Merge Data

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

Step 2: Build Vocabulary

In [ ]:
```
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 >= occurence]
    return vocabulary
```

In [ ]:
```
vocabulary = buildVocabulary(data['text'])
print(vocabulary)
```

Step 3: Create Reverse Index

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

In [ ]:
```
print(reverseIndex(vocabulary))
```

Step 4: Calculate Occurrence Probability

In [ ]:
```
def occurrenceProbability(word, all_documents):
    total_words = sum(1 for doc in all_documents if word in doc)
    total_documents = len(all_documents)
```

```
    total_documents = len(all_documents)
    return total_words / total_documents
```

Step 5: Calculate Conditional Probability

In [ ]:

```
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 {}
```

Step 6: Calculate Conditional Probability with Laplace Smoothing

In [ ]:

```
def conditionalProbabilitySmoothed(word, class_documents, all_documents, vocabulary_size, smoothing_parameter=1):
    total_words = sum(1 for doc in class_documents if word in doc)
    total_documents = len(class_documents)
    return (total_words + smoothing_parameter) / (total_documents + smoothing_parameter * vocabulary_size)
```

Step 7: Predict Class

In [ ]:

```
import math
def predict(document, vocabulary, human_occurence, ai_occurence, human_conditional, ai_conditional):
    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"
```

Step 8: Calculate Accuracy

In [ ]:

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

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

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

In [ ]:

```
human_train = data[data["generated"] == 0]["text"].tolist()
ai_train = data[data["generated"] == 1]["text"].tolist()

human_dev_essays = development_data[development_data['generated'] == 0]['text'].tolist()
ai_dev_essays = development_data[development_data['generated'] == 1]['text'].tolist()

dev_documents = human_dev_essays + ai_dev_essays
dev_labels = ["0"] * len(human_dev_essays) + ["1"] * len(ai_dev_essays)

vocabulary = buildVocabulary(human_train + ai_train)

human_occurence = occurrenceProbability("the", human_train)
ai_occurence = occurrenceProbability("the", ai_train)

vocabulary_size = len(vocabulary)

human_conditional = {word: conditionalProbability(word, human_train, human_train) for word in vocabulary}
ai_conditional = {word: conditionalProbability(word, ai_train, ai_train) for word in vocabulary}
```

In [ ]:

```
print(human_conditional)
print(ai_conditional)
```

Step 9: Compare the Effect of Smoothing

In [ ]:

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

for smoothing_parameter in smoothing_parameters:
    conditional_probs_human_smoothed = {word: conditionalProbabilitySmoothed(word, human_train, human_train, vocabulary
    conditional_probs_llm_smoothed = {word: conditionalProbabilitySmoothed(word, ai_train, ai_train, vocabulary_size, s
    accuracy = calculate_accuracy(dev_documents, dev_labels, vocabulary, human_occurence, ai_occurence, conditional_pro
    print(f"Smoothing Parameter: {smoothing_parameter}, Accuracy: {accuracy}")
```

```
print(f"Smoothing Parameter: {smoothing_parameter}, Accuracy: {accuracy}")
```

Step 10: Derive Top 10 Words Predicting Each Class

In [ ]:

```
def topWords(class_probability, vocabulary, top_n=10):
    sorted_words = sorted(vocabulary, key=lambda word: class_probability[word], reverse=True)
    return sorted_words[:top_n]
```

In [ ]:

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

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