

Kaunas University of Technology

Faculty of Informatics

Introduction to Artificial Intelligence (P176B101)

Report of Laboratory Work 4

Zeynep Eylul GULERYUZ, IFU-0

Student

**Doc. Dr. Germanas Budnikas**

Instructor

Kaunas, 2023

**Figure of contents**

[1. Work assignment 3](#_Toc135824531)

[2. Program code 3](#_Toc135824532)

[3. Investigating the performance 7](#_Toc135824533)

[4. Creating the model with best performing parameters 8](#_Toc135824534)

[5. Conclusions 8](#_Toc135824535)

# Work assignment

Create a classifier of SPAM using Bayes theorem. Investigate a dependency between classifier parameters and efficiency of the classifier output – dependency of number false positive and true negative number on

1. number N of analyzed lexemes,
2. spamicity value of unseen lexeme.

# Program code

The program consists of a class ‘SpamFilter’ and main program, containing the code for the tests.

*Spam Filter class:*

|  |
| --- |
| class SpamFilter:      def \_\_init\_\_(self, spamicityrate, N, threshold):          self.lexeme\_spam\_counts = {}          self.lexeme\_ham\_counts = {}          self.all\_counts = defaultdict(lambda: [0, 0])          self.lexeme\_spamicity\_probabilities = {}          self.spamicityrate = spamicityrate          self.N = N          self.threshold = threshold        def occurrences(self, folder\_path):      # Iterate over files in the folder          lexeme\_counts = defaultdict(int)          for filename in os.listdir(folder\_path):              file\_path = os.path.join(folder\_path, filename)              if os.path.isfile(file\_path):                    with open(file\_path, 'r', encoding='utf-8', errors='ignore') as file:                      # Read the file content                      content = file.read()                      # Tokenize the content into words (lexemes)                      words = re.findall(r'\b\w+\b', content.lower())                      # Count the occurrences of each lexeme                      for word in words:                          lexeme\_counts[word] += 1  # Increment SPAM count          return lexeme\_counts        def count\_occurrences(self, spam\_path, ham\_path):          # Count occurrences in the SPAM folder          self.lexeme\_spam\_counts= self.occurrences(spam\_folder)          # Count occurrences in the HAM folder          self.lexeme\_ham\_counts= self.occurrences(ham\_folder)          # Merge lexeme counts for SPAM and HAM into a single dictionary            for lexeme, counts in self.lexeme\_spam\_counts.items():              self.all\_counts[lexeme][0] += counts  # Update SPAM count          for lexeme, counts in self.lexeme\_ham\_counts.items():              self.all\_counts[lexeme][1] += counts  # Update HAM count      def print\_a\_lexeme(self, lexeme\_to\_print):          if lexeme\_to\_print in self.all\_counts:              spam\_count = self.all\_counts[lexeme\_to\_print][0]              ham\_count = self.all\_counts[lexeme\_to\_print][1]              print(f"Lexeme: {lexeme\_to\_print}")              print(f"SPAM Count: {spam\_count}")              print(f"HAM Count: {ham\_count}")              probability = self.lexeme\_spamicity\_probabilities[lexeme\_to\_print]              print(f"Spamicity: {probability:.2f}")      def calculate\_spamicity\_probability(self):          # Step 1: Calculate total number of lexemes in SPAM and HAM          total\_spam\_lexemes = sum(counts[0] for counts in self.all\_counts.values())          total\_ham\_lexemes = sum(counts[1] for counts in self.all\_counts.values())          # Step 2: Calculate P(lexeme|SPAM) for each lexeme          lexeme\_spam\_probabilities = {}          for lexeme, counts in self.all\_counts.items():              spam\_count = counts[0]              lexeme\_spam\_probabilities[lexeme] = spam\_count / total\_spam\_lexemes          # Step 3: Calculate P(lexeme|HAM) for each lexeme          lexeme\_ham\_probabilities = {}          for lexeme, counts in self.all\_counts.items():              ham\_count = counts[1]              lexeme\_ham\_probabilities[lexeme] = ham\_count / total\_ham\_lexemes          # Step 4: Calculate P(SPAM|lexeme) for each lexeme          for lexeme in self.all\_counts.keys():              # Boundary cases              if lexeme\_spam\_probabilities[lexeme] == 0:                  self.lexeme\_spamicity\_probabilities[lexeme] = 0.01              elif lexeme\_ham\_probabilities[lexeme] == 0:                  self.lexeme\_spamicity\_probabilities[lexeme] = 0.99              else:                  self.lexeme\_spamicity\_probabilities[lexeme] = lexeme\_spam\_probabilities[lexeme] / (lexeme\_spam\_probabilities[lexeme] + lexeme\_ham\_probabilities[lexeme])        def classify\_file(self,file\_path):          encodings = ['utf-8', 'latin-1']  # Add more encodings if needed          # Try different encodings to open the file and count lexeme occurrences          for encoding in encodings:              try:                  with open(file\_path, 'r', encoding=encoding) as file:                      content = file.read()                  # Tokenize the content into words (lexemes)                      words = re.findall(r'\b\w+\b', content.lower())                      new\_file\_spamicity = {}                      for word in words:                          if word in self.lexeme\_spamicity\_probabilities:                              new\_file\_spamicity[word] = self.lexeme\_spamicity\_probabilities[word]                          else:                              new\_file\_spamicity[word] = self.spamicityrate                  break  # Break the loop if the file is read successfully with an encoding              except UnicodeDecodeError:                      continue          # Mean spamicity          spamicity = self.lexeme\_spamicity\_probabilities.values()          mean\_spamicity = sum(spamicity) / len(spamicity)          # Calculate distances from the mean for each lexeme          new\_file\_spamicity\_distances = {}          for lexeme in new\_file\_spamicity:              lexeme\_spamicity = new\_file\_spamicity[lexeme]              distance\_from\_mean = abs(lexeme\_spamicity - mean\_spamicity)              new\_file\_spamicity\_distances[lexeme] = distance\_from\_mean          # Sort the new\_file\_spamicity\_distances dictionary by distances in descending order          sorted\_lexemes = sorted(new\_file\_spamicity\_distances, key=new\_file\_spamicity\_distances.get, reverse=True)          # Select the N lexemes with the largest distances from the mean spamicity          selected\_lexemes = sorted\_lexemes[:self.N]          dividor = reduce(lambda x, y: x \* y, [new\_file\_spamicity[lexeme] for lexeme in selected\_lexemes], 1)          divident = dividor + reduce(lambda x, y: x \* y, map(lambda lexeme: 1 - new\_file\_spamicity[lexeme], selected\_lexemes))          probability = dividor / divident          if probability >= self.threshold:              return 'SPAM'          else:              return 'HAM'          def evaluate\_performance(self, spam\_test\_folder\_path, ham\_test\_folder\_path, printt):          spam\_files\_total = len(os.listdir(spam\_test\_folder\_path))          ham\_files\_total = len(os.listdir(ham\_test\_folder\_path))          spam\_files\_as\_ham = 0          ham\_files\_as\_spam = 0          correctly\_classified\_files = 0          # Classify SPAM files in the SPAM test folder          for file\_name in os.listdir(spam\_test\_folder\_path):              file\_path = os.path.join(spam\_test\_folder\_path, file\_name)              classification\_result = self.classify\_file(file\_path)              if classification\_result == 'HAM':                  spam\_files\_as\_ham += 1              else:                  correctly\_classified\_files += 1          # Classify HAM files in the HAM test folder          for file\_name in os.listdir(ham\_test\_folder\_path):              file\_path = os.path.join(ham\_test\_folder\_path, file\_name)              classification\_result = self.classify\_file(file\_path)              if classification\_result == 'SPAM':                  ham\_files\_as\_spam += 1              else:                  correctly\_classified\_files += 1          # Calculate the ratios and print the metrics          false\_positive\_ratio = spam\_files\_as\_ham / spam\_files\_total \* 100          false\_negative\_ratio = ham\_files\_as\_spam / ham\_files\_total \* 100          correctly\_classified\_ratio = correctly\_classified\_files / (spam\_files\_total + ham\_files\_total) \* 100            if printt:              print("Number of SPAM files classified as HAM (false positive):", spam\_files\_as\_ham)              print("Number of HAM files classified as SPAM (false negative):", ham\_files\_as\_spam)              print("Ratio of correctly classified files (%):", correctly\_classified\_ratio)          return [correctly\_classified\_ratio, spam\_files\_as\_ham, ham\_files\_as\_spam] |

*Main program:*

|  |
| --- |
| import os  import re  from collections import defaultdict  from functools import reduce  import matplotlib.pyplot as plt  def test\_with\_spamicity\_value(values):      current\_directory = os.getcwd()      # Folder paths for SPAM and HAM folders      spam\_folder = os.path.join(current\_directory, 'SPAM')      ham\_folder = os.path.join(current\_directory, 'HAM')      performances = []      falsepositive =[]      falsenegative = []      for spamicityvalue in values:          # Training the model with data          NewSpamFilter = SpamFilter(spamicityvalue, 2, 0.5)          NewSpamFilter.count\_occurrences(spam\_folder, ham\_folder)          NewSpamFilter.calculate\_spamicity\_probability()          spam\_test\_folder = os.path.join(current\_directory, 'testspam')          ham\_test\_folder = os.path.join(current\_directory, 'testnotspam')          result = NewSpamFilter.evaluate\_performance(spam\_test\_folder, ham\_test\_folder, False)            performances.append(result[0])          falsepositive.append(result[1])          falsenegative.append(result[2])      return performances, falsepositive, falsenegative  values = (0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9)  result = test\_with\_spamicity\_value(values)  # Create a 2x2 grid of subplots  fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(14, 4))  axes[0].plot(values, result[0], '-o')  axes[0].set\_xlabel('New lexeme spamicity probability')  axes[0].set\_ylabel('Accurancy')  axes[1].plot(values, result[1], '-o')  axes[1].set\_xlabel('New lexeme spamicity probability')  axes[1].set\_ylabel('Number of false positive')  axes[2].plot(values, result[2], '-o')  axes[2].set\_xlabel('New lexeme spamicity probability')  axes[2].set\_ylabel('Number of false negative')  fig.suptitle('Testing the performance with different new lexeme spamicity probability values', fontsize=14, fontweight='bold')  plt.tight\_layout()  plt.show()  def test\_with\_N(values):      current\_directory = os.getcwd()      # Folder paths for SPAM and HAM folders      spam\_folder = os.path.join(current\_directory, 'SPAM')      ham\_folder = os.path.join(current\_directory, 'HAM')      performances = []      falsepositive =[]      falsenegative = []      for N in values:          # Training the model with data          NewSpamFilter = SpamFilter(0.4, N, 0.5)          NewSpamFilter.count\_occurrences(spam\_folder, ham\_folder)          NewSpamFilter.calculate\_spamicity\_probability()          spam\_test\_folder = os.path.join(current\_directory, 'testspam')          ham\_test\_folder = os.path.join(current\_directory, 'testnotspam')          result = NewSpamFilter.evaluate\_performance(spam\_test\_folder, ham\_test\_folder, False)            performances.append(result[0])          falsepositive.append(result[1])          falsenegative.append(result[2])      return performances, falsepositive, falsenegative  values = (2,4,16,32,64,128)  result = test\_with\_N(values)  # Create a 2x2 grid of subplots  fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(14, 4))  axes[0].plot(values, result[0], '-o')  axes[0].set\_xlabel('New N number')  axes[0].set\_ylabel('Accurancy')  axes[1].plot(values, result[1], '-o')  axes[1].set\_xlabel('New N number')  axes[1].set\_ylabel('Number of false positive')  axes[2].plot(values, result[2], '-o')  axes[2].set\_xlabel('New N number')  axes[2].set\_ylabel('Number of false negative')  fig.suptitle('Testing the performance with different N numbers', fontsize=14, fontweight='bold')  plt.tight\_layout()  plt.show()  def test\_with\_threshold(values):      current\_directory = os.getcwd()      # Folder paths for SPAM and HAM folders      spam\_folder = os.path.join(current\_directory, 'SPAM')      ham\_folder = os.path.join(current\_directory, 'HAM')      performances = []      falsepositive =[]      falsenegative = []      for threshold in values:          # Training the model with data          NewSpamFilter = SpamFilter(0.4, 2, threshold)          NewSpamFilter.count\_occurrences(spam\_folder, ham\_folder)          NewSpamFilter.calculate\_spamicity\_probability()          spam\_test\_folder = os.path.join(current\_directory, 'testspam')          ham\_test\_folder = os.path.join(current\_directory, 'testnotspam')          result = NewSpamFilter.evaluate\_performance(spam\_test\_folder, ham\_test\_folder, False)            performances.append(result[0])          falsepositive.append(result[1])          falsenegative.append(result[2])      return performances, falsepositive, falsenegative  values = (0.01, 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9)  result = test\_with\_threshold(values)  # Create a 2x2 grid of subplots  fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(14, 4))  axes[0].plot(values, result[0], '-o')  axes[0].set\_xlabel('New threshold value')  axes[0].set\_ylabel('Accurancy')  axes[1].plot(values, result[1], '-o')  axes[1].set\_xlabel('New threshold value')  axes[1].set\_ylabel('Number of false positive')  axes[2].plot(values, result[2], '-o')  axes[2].set\_xlabel('New threshold value')  axes[2].set\_ylabel('Number of false negative')  fig.suptitle('Testing the performance with different threshold values', fontsize=14, fontweight='bold')  plt.tight\_layout()  plt.show()  # Creating the model with the best performing parameters  NewSpamFilter = SpamFilter(0.4, 64, 0.01)  # Training the model  NewSpamFilter.count\_occurrences(spam\_folder, ham\_folder)  NewSpamFilter.calculate\_spamicity\_probability()  # Test folders  spam\_test\_folder = os.path.join(current\_directory, 'testspam')  ham\_test\_folder = os.path.join(current\_directory, 'testnotspam')  print('Best performance: ')  NewSpamFilter.evaluate\_performance(spam\_test\_folder, ham\_test\_folder, True) |

# Investigating the performance

The first test is conducted by changing the spamicity value of unseen lexeme. The parameters are:

NewSpamFilter = SpamFilter(spamicityvalue, 2, 0.5) where spamicityvalue is the value that wants to be tested. Results are as follows:

A picture containing text, line, screenshot, plot

Description automatically generated

Figure 1: Testing the performance with different new lexeme spamicity probability values

The second test is conducted by changing the number N of analysed lexemes. The parameters are:

NewSpamFilter = SpamFilter(0.4, N, 0.5) where N is the value that wants to be tested. Results are as follows:

A picture containing line, text, diagram, plot

Description automatically generated

Figure 2: Testing the performance with different N numbers

The third test is conducted by changing the threshold of spamicity value. The parameters are:

NewSpamFilter = SpamFilter(0.4, 2, threshold) where threshold is the value that wants to be tested. Results are as follows:

A picture containing text, line, plot, diagram

Description automatically generated

Figure 3: Testing the performance with different threshold values

# Creating the model with best performing parameters

According to the tests that have been done, the best performing parameters are:

1. **64** for the **number *N*** of analyzed lexemes,
2. **0.4** for the **spamicity** value of unseen lexeme,
3. **0.01** for the **threshold** value

With these parameters, model is created: NewSpamFilter = SpamFilter(0.4, 64, 0.01) The results are as follows:

Best performance:

Number of SPAM files classified as HAM (false positive): 10

Number of HAM files classified as SPAM (false negative): 1

Ratio of correctly classified files (%): 94.5

# Conclusions

The classifier is created using Bayes theorem. The results for the correct number of classifications are very promising and can be further improved by testing on more precise values and calculating the differences by more precise decimal points.