Assistive Guideline of Categorization for Competitive Programming Problems

by

Najmus Sakib Dhrubo
16101152
Md Samiul Islam
17101419

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
June 2021

© 2021. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

[illegible]	[illegible]
Najmus Sakib Dhrubo 16101152	Md Samiul Islam 17101419

Approval

The thesis/project titled "Assistive Guideline of Categorization for Competitive Programming Problems" submitted by

1. Najmus Sakib Dhrubo (16101152)
2. Md Samiul Islam (17101419)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on June 6, 2021.

Examining Committee:

Supervisor:
(Member)

[illegible]

Md. Tahmid Ul Islam Rafi
Lecturer
Department of Computer Science and Engineering
Brac University

Co-supervisor:
(Member)

Moin Mostakim

Moin Mostakim
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

[illegible]

Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

[illegible]

Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Programming is a very useful skill nowadays. Programming contests give people the opportunity to increase their programming skills. By solving programming contest problems contestants can increase not only their programming skills but also their mathematical and algorithmic knowledge. The competitive programming problems are presented in problem statements. Sometimes they are presented in the form of a story or sometimes directly. To solve the problem contestants must read the problem statement carefully. The problems can be of many categories. We have tried to classify number theory and graph theory problems. At first, we collected data from competitive programming problem statements. Then we used different machine learning algorithms such as fully connected neural network, naive bayes classifier, support vector machine on the data to predict if the category of the problem is either number theory or graph theory. With such machine learning approaches we achieved test accuracy of about 72%, 75% and 74%.

Keywords: Competitive Programming; Number Theory; Graph Theory; Neural Network; Naive Bayes Classifier; Support Vector Machine

Acknowledgement

Firstly, all praise to the Almighty Allah for whom our thesis have been completed without any major interruption.
Secondly, to our co-supervisor Moin Mostakim sir for his kind support and advice in our work. He helped us whenever we needed help.
Finally, to our parents who have always supported us.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

$BS4$	Beautiful Soup 4
$CFMC10$	Codeforces Multiclass-10
$CFMC5$	Codeforces Multiclass-5
$CFML10$	Codeforces Multilabel-10
$CFML20$	Codeforces Multilabel-20
CNN	Convolutional Neural Network
CSV	Comma Separated Value
$HTML$	HyperText Markup Language
LCM	Least Common Multiple
MLP	Multilayer Perceptron
$ReLU$	Rectified Linear Unit
SVM	Support Vector Machine
$TF-IDF$	Term Frequency - Inverse Document Frequency
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 Introduction

In competitive programming contestants are given a set of mathematical and algorithmic problems to solve. The problems can sometimes be presented as a story or sometimes they can be presented directly. Contestants have to carefully read the problem statement and gather information from it to understand the problem properly then they need to write code to solve the problem. These solutions need to be efficient so that they can be executed within a certain time limit and memory limit. The contestants have to generate the right output using their solutions. Usually, problems on topics from simple to high mathematical concepts, algorithms, data structures are given in these competitions. We used different machine learning algorithms to determine the category of a given problem. In order to classify competitive programming problems we used machine learning algorithms on the problem statement to build our classifier. We classified the competitive programming problems into two classes. One class is Number Theory and the other class is Graph Theory.

1.2 Problem Statement

Programming contest problems can be very challenging. Problem authors try to make the problems challenging for the contestants in many clever ways. The problem statement is very important for contestants. Contestants must read the problem statement carefully and use the information and hints given by the authors. Contestants use those information and hints to develop a solution for the problems. Then they write code according to that solution. The category of the problems can be of many types. Some categories are number theory, graph theory, geometry, advanced algorithms such as segment tree. Understanding the category of a problems is very important to solve that problem. Depending on the category the techniques and algorithms that need to be used can be very different. We have tried to determine if a category of a problem is number theory of graph theory using machine learning algorithms.

1.3 Research Objective

Our goal is to use machine learning algorithms to predict if the category of a competitive programming problem is either number theory or graph theory. Determining the category of a problem is very important for contestants in competitive programming. Problem authors try to make the problems challenging for the contestants. It can be challenging just to determine the category of a competitive programming problem. So, if someone tries to solve competitive programming problems they need to spend a lot of time in thinking about the problem and also reading and understanding the problem statement. Our objective is to simplify the process of determining the category of a problem by using machine learning algorithms. We have tried to determine if the category of a problem is either number theory or graph theory by using the problem statements written by the problem authors. We tried to extract relevant information from the problem statement and format that information in such a way so that we can use machine learning models to classify the problem into number theory and graph theory. We created a database by collecting number theory and graph theory problems. Then we used different machine learning algorithms to determine the problem.

Chapter 2

Literature Review

We found two papers where work on classifying programming contest problem was done. In [3] Athavale et al. used different algorithms on problem statements. In [2] Subramanian et al. classified programming contest problems by using CNN on the solution code of users that solved the problem. In both papers the data was collected from the online judge codeforces. Because of the difference in the type of data that was used in the papers there were different advantages and disadvantages for the authors. The datasets used in the papers had different features and the authors used different algorithms to use these features. Athavale et al. used algorithms that used the information given in the problem statements. Information in the problem statements can be given in a format that uses a story or in a format where problem statements can be short and information is given directly.

Subramanian et al. used the solution code of the people that have solved the problems already. There can be some similarity in the logic of the solution code written by different users. So, these similarities can be used in machine learning algorithms. Sometimes a competitive programming problem can be solved using different techniques. So, for the same problem there can be difference in logic in the solution code of different users

Athavale et al. created four datasets that had different types of competitive programming problems. They created two multilabel datasets and two multiclass datasets. The multilabel datasets had data that had data with multiple labels. There were 10 classes in one multilabel dataset. The other multilabel dataset had 20 classes. One multiclass dataset had 5 classes and another one had 10 classes. The data in the multiclass datasets had only one label.

At first, Athavale et al collected their data. After data collection they filtered the dataset and removed any unwanted data. Initially they created a dataset that had 4300 problems from codeforces. After that they removed problem sets with incomplete problem statements, problem sets that had no tags and also the problem sets that did not have any tags for algorithms. There were 4019 problems left after this filtering process. Now the problems in this filtered dataset had 35 different tags. In order to create the two multilabel dataset they created a list of tags with decreasing frequency. They created a dataset with problems that had the top 20 tags from the tag list. This dataset with 3960 problems was named Codeforces

Multilabel-20 (CFML20). This dataset is a multilabel dataset with 20 classes. With similar method the Codeforces Multilabel-10 (CFML10) dataset was created with the top 10 tags from the tag list. In order to create the multiclass datasets, they took problems from the CFML20 dataset. These problems had only 1 tag. Then they took problems with the 10 most common tags and created the Codeforces Multiclass-10 (CFMC10) dataset. This dataset had 1159 problems. Similarly, another multiclass dataset called Codeforces Multiclass-5 (CFMC5) was created by using the 5 most common tags. CFMC5 had 550 problems.

Athavale et al. used different algorithms on the four datasets. Some of the algorithms that they used were Convolutional Neural Network (CNN), CNN ensemble, Multilayer Perceptron (MLP). Athavale et al. achieved an accuracy of 62.7 % accuracy on CFMC5 dataset by using CNN Ensemble. On the CFMC10 dataset they achieved and accuracy of 54.7 % by using CNN. On the CFML10 dataset they were able to achieve F1 micro score of 45.32 and F1 macro score of 38 by using CNN Ensemble.

Chapter 3

Some Common Algorithms Used in Document Classification

Here are some common algorithms that are used in document classification.

3.1 Bag of Words and TF-IDF

Bag of words technique is used to select features from a text. Using bag of words technique, the selected features from a text can be formatted in a way that the data from the text can be used in machine learning models. At first, the bag of words technique creates a vocabulary consisting of known words. Then a score or value is set for each word. This value or score for a word can be set according the presence of the word in the text. The vocabulary can be built by using the unique words that appear in the text of the dataset. When processing each sentence, we can set a value for the words in the vocabulary to indicate if it is present in the sentence or not. To set the value a very simple binary scoring method can be used. A value of 0 can be used to represent the absence of a word in a sentence and a value of 1 can be used to represent the presence of a word in a sentence [4]. Using this representation, we can create vocabulary vectors. There will be a vector for each sentence that will represent the absence or presence of the words in the vocabulary. The values can be set using other methods as well. Another scoring method to set the values for the words can be used by using the frequency of a word relative to other words in the text. However, using frequency to set the values can create some problems. There can be some words that are common but do not contain relevant information. These words can appear more than other words that contain useful information. This problem can be solved using the TF-IDF (Term Frequency – Inverse Document Frequency) technique. This technique penalizes the frequent common words that do not have useful information by rescaling the values of the words.

3.2 Multilayer Perceptron (MLP)

Data that can be linearly separated can be classified using perceptron [9]. Perceptron faces limitations on the datasets that cannot be linearly separated. Data that cannot be linearly separated can be classified using multilayer perceptron. MLP can have 1 input layer, 1 output layer and multiple hidden layers. Each layer can have multiple nodes.

Below we can see a figure of a multilayer perceptron with 1 input layer, 1 hidden layer and 1 output layer. The input layer has 2 nodes, the hidden layer has 3 nodes, the output layer has 1 node.

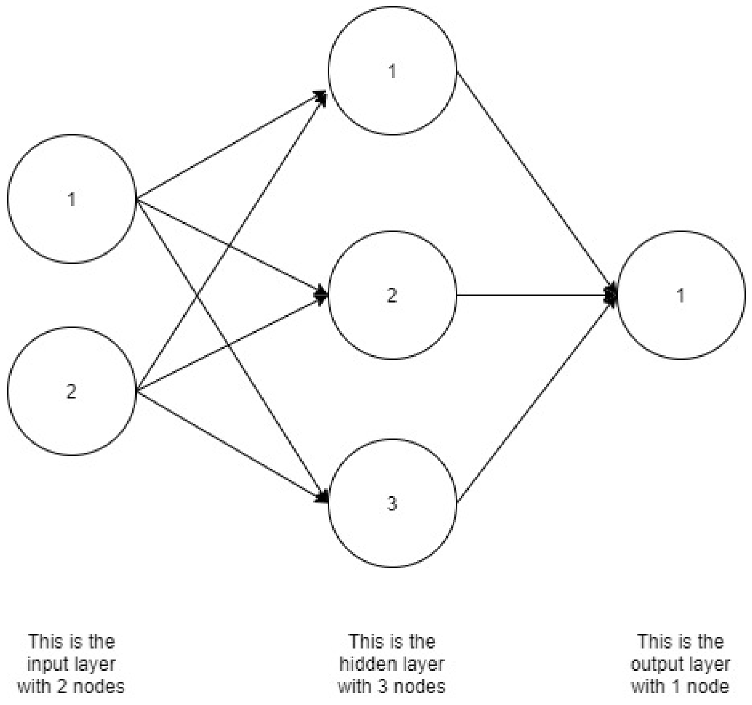

1
2
1
2
3
1
This is the input layer with 2 nodes
This is the hidden layer with 3 nodes
This is the output layer with 1 node

Figure 3.1: Multilayer Perceptron

We can see that there are connections between the nodes of one layer and the next layer. Each node in a layer is connected to all nodes in the next layer. The connections between the nodes have a weight value. At first, a dot product is calculated with the values from the input layer and the values of the weights between the input layer and the hidden layer. After that, the dot product is passed on to the hidden layer. The hidden layer nodes passes those values into an activation function. Another dot product value is generated between the values we get after using the activation function and the weights between the hidden layer and the output layer. Then that dot product value is sent to the output layer. The output layer uses the dot product values with an activation function and generates output value. Now the output value can be sent back through the neural network and the full process is repeated using backpropagation until the optimal output is generated. There are

many activation functions that can be used in the hidden layers and the output layer. ReLU (Rectified Linear Unit) and sigmoid are such functions. Formula for the sigmoid function is given below.

$$f(x) = \frac{1}{1 + e^{-x}}$$

3.3 Bayes Theorem and Naive Bayes

Bayes theorem is one of the most important formula in calculating conditional probability. It can be used to classify data. The formula for bayes theorem is:

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)}$$

From the equation given above we can see that we can calculate the probability of event A happening given that event B has already happened if we already know P(A) which the probability of event A happening independent of any other variables,P(B|A) the probability of event B happening given that event A has already happened, P(B) probability of event B happening independent of any other variables. Here,

P(A|B) is called the posterior.
P(A) is called prior.
P(B|A) is called likelihood.

Naive bayes simplifies Bayes theorem. Given the value of the class variable naive bayes assumes there is conditional independence between every pair of features [5]. So, if probability of event A happening depends on multiple variables then the Bayes theorem equation would look like the following:

$$P(A|B_1, B_2, B_3...B_n) = \frac{P(A) \times (B_1, B_2, B_3...B_n|A)}{P(B_1, B_2, B_3...B_n)}$$

Now naive bayes simplifies the equation into the following:

$$P(A|B_1, B_2, B_3...B_n) = P(A) \times (B_1, B_2, B_3...B_n|A)$$

Because of this simplified form it is possible to use the naive bayes formula to classify big datasets with many features. Naive bayes works very well in document classification. The algorithm works very fast.

3.4 Support Vector Machine (SVM)

Support vector machine is a very useful machine learning algorithm. It can be used for both regression and classification task [8]. SVM can represent multiple classes in multidimensional space using a hyperlane. Hyperplane is the space that is divided between the objects that are in different classes. Support vectors are those data points that are closest to the hyperplane. In order to minimize errors the hyperlanes are generated in iterations by support vector machine. Support vector machine tries to find a maximum marginal hyperplane by dividing the datasets into classes.

Support vector machine can transform a low dimensional input space into a high dimensional input space by using kernel. The process of transforming a low dimensional input space into a high dimensional input space is called kernel trick. By using kernel trick support vector machine can transform problems that were not separable into separable problems. The ability to perform kernel trick makes support vector machine a very strong and useful machine learning algorithm.

Chapter 4

Work Sequence

At first, we collected data consisting of problem name, problem statements and category of number theory and graph theory problems. Our dataset had 1056 data points. After collecting our data we discovered that we could not use the numerical values and mathematical symbols that were present in the problem statements of the data points. So we had to do some data cleaning.

To perform our data cleaning we removed all numerical values and mathematical symbols from the problem statements of our dataset. We used the label 0 for number theory problems and label 1 for graph theory problems.

Then we selected and extracted our required features from the dataset and split the dataset for training and testing. We used about 70% of the dataset for training and 30% for testing.

Then we used the dataset to train different machine learning models. We used a fully connected neural network with 1 input layer, 5 hidden layers, 1 output layer, a naive bayes classifier and support vector machine.

We tested the machine learning models on the part of the dataset that we reserved for testing and got test accuracy score.

With the neural network we achieved about 72% test accuracy score. With the naive bayes classifer we achieved about 75% test accuracy score and with support vector machine we achieved about 74% accuracy score.

Below a figure is given that shows the sequence of the work we have done. The figure provides a clear overview of our work.

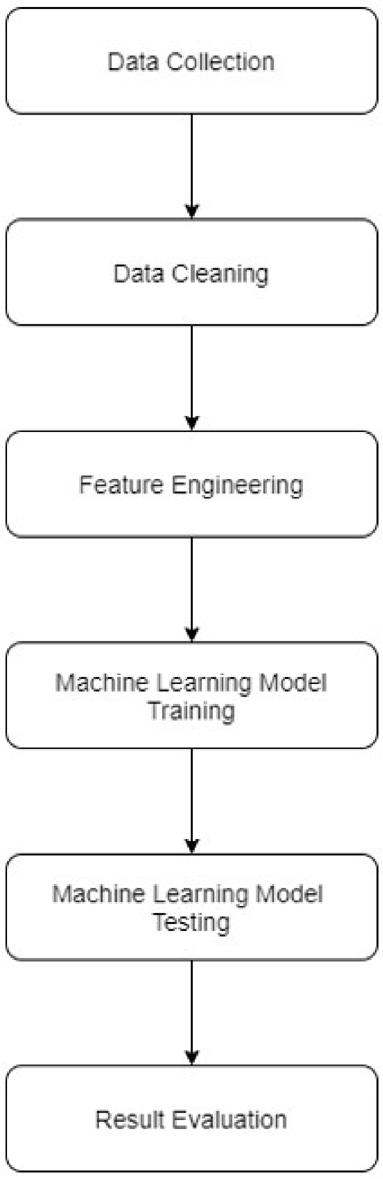


Figure 4.1: Work Sequence

Chapter 5

Dataset Collection and Processing

5.1 Problem Statement Structure

A programming contest's questions all follow some general pattern [10]. The following are a few of them:

(a) Problem name: To begin with, each problem has its own name. This name was chosen by the problem creators. They try to make it one-of-a-kind and relevant to the problem

(b) limit and memory limit: The time complexity and space complexity of the algorithms that must be preserved in order to pass this question are discussed in this section.

(c) Problem statement: This is the most important aspect of a question. The problem is listed here. In general, problem solvers tell a story to reflect the problem in order to make it more interesting. They often use graphs and often state the problem explicitly. To build the solution, the contestants read the problem statements and extract the main concept.

(d) Input: To begin, the contestants gather information and process it in order to arrive at a solution. That is why, for each question in the input segment, the format of input is defined so that contestants can better interpret it. In general, they will state what the maximum and minimum values are. Additionally, they discuss the datatype. It has been observed that they sometimes provide special instructions to deal with unusual situations.

(e) Output: This segment is also about format, but this time the output format is discussed. In a programming competition, a solution is accepted if the outcome is identical. For example, suppose the desired answer is 1.5, but someone submits a solution that yields a 1.50. The solution would then fail. As a result, each contestant must also preserve the performance format.

(f) Sample Input: We can see that they provide guidance about the format of the input in the input segment. They include the actual values in the sample input portion. So that the contestant can verify whether or not this solution is taking the feedback in the correct format. Furthermore, some contestants are unable to comprehend the input format.In that case, sample input can be extremely beneficial to contestants because it provides them with the real value with which to evaluate their solution.

(g) Sample Output: The sample output section, like the sample input section, shows the actual values that the correct algorithms would generate. This is extremely helpful since the contestants will measure whether or not the solution works.

5.2 Verdict Information

Let's speak about the verdict now, since it determines the outcome of a programming competition. First and foremost, one contestant must understand how the decision is made. Generally, the problem setters produce multiple or all possible test cases and their answers for each question. The responses are then saved in text files.As a result, when a contestant submits a solution, the test cases generated earlier are used to generate the output. Finally, the method, also known as an online judge, compares two outputs: one generated by the contestant and another generated by the problem setters or judges. One solution may be appropriate for a variety of verdicts [7].

The following are a few of them:

(a) Wrong Answer: When a contestant's solution does not fit the correct answer, the solution is labeled the "wrong answer."

(b) Time limit exceeded: When a contestant's solution fails to complete within the problem's time limit, a time limit exceeded judgment is given.

(c) Memory limit exceeded: As with the time limit, if a solution fails to execute within the memory allocated for that query, it is considered to have exceeded the memory limit.

(d) Run time error: The online judge will offer a runtime error if a solution compiles perfectly but throws errors during execution. Creating an array with a short length is a typical example of a run-time error. As a result, it functions perfectly during compile time but fails when the code attempts to allocate values to an index that is larger than the array's length.

(e) Presentation error: A presentation error is another intriguing conclusion. This verdict is given when the contestant's solution produces correct answer but not in the appropriate format.

(f) Accepted: This decision is straightforward. If a solution meets all of the conditions, the verdict is said to be accepted.

5.3 Web Scraping

Dataset is the fuel in the age of Machine Learning. Furthermore, people devote 90 % of their attention to a dataset. As a result, selecting the appropriate dataset is a critical component of our study. We needed a high-quality dataset after we decided on this subject. We were unable to locate a ready-to-use dataset, however. As a result, we've chosen to use codeforces.com to gather the info. We used the codeforces problem tag "number theory" to get number theory problems and problem tag "dfs and similar" to get graph theory problems.

Listing 5.1: Python Web Scraping Code

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl
import csv

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

# connecting url and parsing lxml
url = input('Enter - ')
html = urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, "lxml")

# creating CSV file
file_name = input('Enter File name:') + '.csv'
csv_file = open(file_name, 'w')
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['problem_statement', 'problem_categorey'])

# Main Part
for problem in soup.find_all('td',class_='id'):
  tags = problem('a')
  for tag in tags:
    n_url = tag.get('href', None)
    n_link = f'https://codeforces.com/{n_url}'
    n_html = urlopen(n_link, context=ctx).read()
    n_soup = BeautifulSoup(n_html, "lxml")

    # extracting problem statement variable
    prob_con = n_soup.find("div",class_="problem-statement")

    # getting problem category
    sidebar_tag = n_soup.find('div', style='padding: 0.5em;')
    catagory_list = []
    for sidetag in sidebar_tag.find_all('span', class_='tag-box'):
      sdtag = sidetag.text.strip()
      sdtag_list = sdtag.split()
      for content in sdtag_list:
        if content in catagory_list:
          continue
        else:
          catagory_list.append(content)
    problem_categorey = ' '.join([str(elem) for elem in
   catagory_list])
```

```

    # Assign problem statement
    problem_text = prob_con.text
    problem_text_list = problem_text.split("Input")
    m = problem_text_list[0]
    problem_statement = m


    print(problem_statement)
    print(problem_categorey)


    csv_writer.writerow([problem_statement,problem_categorey])
csv_file.close()
```

Python libraries: Here we have used **urlopen** to face data using url. We also have used **BeautifulSoup** as a web scraping tool. and finally used **ssl** and **csv** modules.

Listing 5.2: Python libraries

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import ssl
import csv
```

Taking URL: connecting url and parsing lxml: Here we are taking the url from the user then connecting the url using urlopen and finally grabbing data using BeautifulSoup [11].

Listing 5.3: URl manage

```
url = input('Enter - ')
html = urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, "lxml")
```

Creating CSV file: In this section, we have created a CSV file to store our dataset. Here we are taking input file from the user to assign the CSV file name.

Listing 5.4: Creating CSV file

```
file_name = input('Enter File name:') + '.csv'
csv_file = open(file_name, 'w')
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['problem_statement', 'problem_categorey'])
```

Main Part: Our main is done here. At first, using a nested loop we went through every section of a HTML page. Depending on the class name, id name and others unique names grab the info. After that, data filtration is done where some info is removed which is not necessary for the model.

Listing 5.5: Main Part

```python
# getting problem url
for problem in soup.find_all('td',class_='id'):
  tags = problem('a')
  for tag in tags:
    n_url = tag.get('href', None)
    n_link = f'https://c...content-available-to-author-only...s.com
   /{n_url}'
    n_html = urlopen(n_link, context=ctx).read()
    n_soup = BeautifulSoup(n_html, "lxml")

    # extracting problem statement variable
    prob_con = n_soup.find("div",class_="problem-statement")

    # getting problem category
    sidebar_tag = n_soup.find('div', style='padding: 0.5em;')
    catagory_list = []
    for sidetag in sidebar_tag.find_all('span', class_='tag-box'):
      sdtag = sidetag.text.strip()
      sdtag_list = sdtag.split()
      for content in sdtag_list:
        if content in catagory_list:
          continue
        else:
          catagory_list.append(content)
    problem_categorey = ' '.join([str(elem) for elem in
   catagory_list])
```

Assign Problem Statement: Last but not least, we populate the dataset. Finally, the dataset is written in CSV format and closes the file.

Listing 5.6: Assign Problem Statement

```python
# Assign problem statement
    problem_text = prob_con.text
    problem_text_list = problem_text.split("Input")
    m = problem_text_list[0]
    problem_statement = m


    print(problem_statement)
    print(problem_categorey)


    csv_writer.writerow([problem_statement,problem_categorey])
csv_file.close()
```

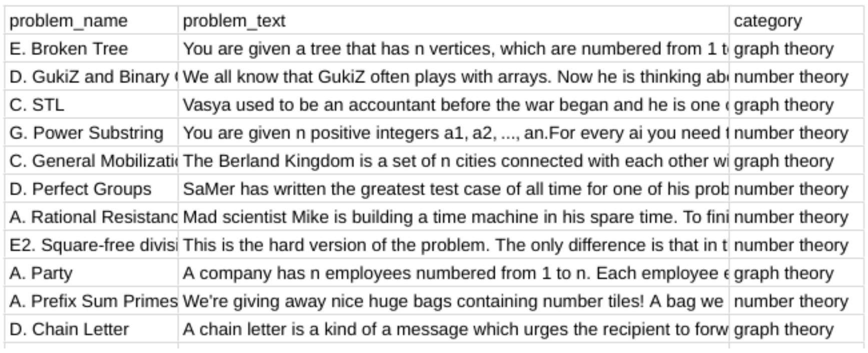
5.4 Initial Data Processing

Data Processing: We have only taken English letters using regular expressions [6]. Furthermore, set category for number theory as 0 (zero) and for graph theory as 1 (one).

Listing 5.7: Data Processing

```
f_num['problem_text'] = f_num['problem_text'].replace('[^a-zA-Z ]',
     ' ', regex=True).str.lower()
f_num['problem_text'] = (f_num['problem_text'].str.split()).str.
    join(' ')
cat = {'number theory': 0,'graph theory': 1}
f_num.category = [gender[item] for item in f_num.category]
```

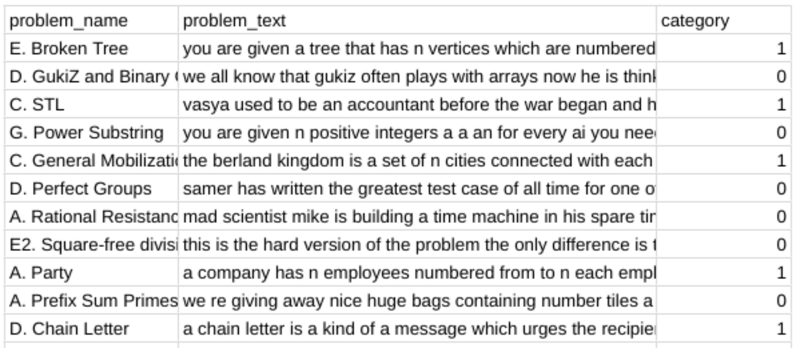
The raw dataset that we collect using web scraping is shown below: Finally after



problem_name	problem_text	category
E. Broken Tree	You are given a tree that has n vertices, which are numbered from 1 t	graph theory
D. GukiZ and Binary (	We all know that GukiZ often plays with arrays. Now he is thinking ab	number theory
C. STL	Vasya used to be an accountant before the war began and he is one (	graph theory
G. Power Substring	You are given n positive integers a1, a2, ..., an.For every ai you need t	number theory
C. General Mobilizati	The Berland Kingdom is a set of n cities connected with each other wi	graph theory
D. Perfect Groups	SaMer has written the greatest test case of all time for one of his prob	number theory
A. Rational Resistanc	Mad scientist Mike is building a time machine in his spare time. To fini	number theory
E2. Square-free divisi	This is the hard version of the problem. The only difference is that in t	number theory
A. Party	A company has n employees numbered from 1 to n. Each employee e	graph theory
A. Prefix Sum Primes	We're giving away nice huge bags containing number tiles! A bag we	number theory
D. Chain Letter	A chain letter is a kind of a message which urges the recipient to forw	graph theory

Figure 5.1: Dataset before cleaning

cleaning the dataset that looks like this one given below:



problem_name	problem_text	category
E. Broken Tree	you are given a tree that has n vertices which are numbered	1
D. GukiZ and Binary (	we all know that gukiz often plays with arrays now he is think	0
C. STL	vasya used to be an accountant before the war began and h	1
G. Power Substring	you are given n positive integers a a an for every ai you nee	0
C. General Mobilizati	the berland kingdom is a set of n cities connected with each	1
D. Perfect Groups	samer has written the greatest test case of all time for one o	0
A. Rational Resistanc	mad scientist mike is building a time machine in his spare tir	0
E2. Square-free divisi	this is the hard version of the problem the only difference is t	0
A. Party	a company has n employees numbered from to n each empl	1
A. Prefix Sum Primes	we re giving away nice huge bags containing number tiles a	0
D. Chain Letter	a chain letter is a kind of a message which urges the recipie	1

Figure 5.2: Dataset after cleaning

We saved the dataset in a file named “dataset.csv”. There are 955 data points in our dataset.

5.5 Feature Selection and Extraction

From the dataset we will use the columns “problem_text” and “category”. The problem_text column is for problem statement and the category column is for the label of the data point. We need to select and extract our features from the dataset. For this we selected some common words and terms used in number theory and graph theory problems.

For example, in number theory problems we may see the words such as factor, prime, remainder etc used more than in graph theory problems. Similarly in graph theory problems we may see words such as edge, connect etc used more than in number theory problems. We selected 19 words and terms. Some words and terms have the same meaning.

We converted those 19 words and terms into 16 features. Then counted how many times each word and term appear in the problem text of each data point and used the count as the values for the features.

The python code used to select and extract features is given below.

```python
import pandas as pd

feature_map = {"number": 0, "factor": 1, "prime": 2, "mod": 3, "
    remainder": 4, "gcd": 5, "greatest common divisor": 5, "divisor"
    : 6,  "lcm": 7, "least common multiple": 7, "vertex": 8, "
    vertices": 8, "edge": 9, "connect": 10, "city": 11, "road": 12,
    "graph": 13, "node": 14, "root": 15}

feature = []

df1 = pd.read_csv(r'dataset.csv')
problems = df1['problem_text']
label = list(df1['category'])


for a in problems:
    counter = [0] * 16

    for k in feature_map:
        counter[feature_map[k]]+=a.count(k)

    feature.append(counter)

df2 = pd.DataFrame(feature)
df2.to_csv('dataset_feature.csv', index=False)
df2 = pd.DataFrame(label)
df2.to_csv('datset_label.csv', index=False)
```

On line 1 we import the pandas library.

```
1 import pandas as pd
```

In the code given below we create a python dictionary of word and terms that we can find frequently in number theory and graph theory problems. Here for each word and term a number is assigned. For each data point we will create a list. The number that is assigned in the dictionary represents the index for that word or term in the list. We will use the index of a word to assign how many times that word or term has appeared in a particular data point. Some words and terms in the dictionary have the same index. This is because terms like “lcm” and “least common multiple” have the same meaning so they point to the same index.

```
1 feature_map = {"number": 0, "factor": 1, "prime": 2, "mod": 3, "
     remainder": 4, "gcd": 5, "greatest common divisor": 5, "divisor"
     : 6,  "lcm": 7, "least common multiple": 7, "vertex": 8, "
     vertices": 8, "edge": 9, "connect": 10, "city": 11, "road": 12,
     "graph": 13, "node": 14, "root": 15}
```

Using the code given below we create an empty list named feature and we load the dataset as pandas dataframe into a variable named df1. Then from the dataset we take the the problem texts as a list in the variable named problems and we store the labels as a list in the variable named label.

```
1 feature = []
2 
3 df1 = pd.read_excel(r'dataset.csv')
4 problems = df['problem_text']
5 label = list(df['category'])
```

In the section of code given below we count how many times each word and term in the feature_map appear in the problem text for each data point. Then we add that count in to a list named counter using the values in feature_map. Then we append that list to the list named feature that we created before.

```
1 for a in problems:
2     counter = [0] * 16
3 
4     for k in feature_map:
5         counter[feature_map[k]]+=a.count(k)
6 
7     feature.append(counter)
```

Using the code below we use the variable named df2 and save the feature list and label list in files named dataset_feature.csv and dataset_label.csv.

```
df2 = pd.DataFrame(feature)
df2.to_csv('dataset_feature.csv', index=False)
df2 = pd.DataFrame(label)
df2.to_csv('dataset_label.csv', index=False)
```

Chapter 6

Experiments We Have Done

6.1 Experiment With Neural Network

We have used multilayer perceptron to classify our dataset. Multilayer perceptron is a fully connected neural network. The python code is given below.

```python
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense

feature_df = pd.read_csv(r'dataset_feature.csv')
label_df = pd.read_csv(r'dataset_label.csv')

feature = []

for i in range(len(feature_df)):
    feature.append(list(feature_df.loc[i,:]))

label = list(label_df['0'])

feature_train = feature[:668]
label_train = label[:668]
feature_test = feature[668:]
label_test = label[668:]

classifier = Sequential()
classifier.add(Dense(25, input_dim = 16, activation = 'relu'))
classifier.add(Dense(20, activation = 'relu'))
classifier.add(Dense(15, activation = 'relu'))
classifier.add(Dense(10, activation = 'relu'))
classifier.add(Dense(5, activation = 'relu'))
classifier.add(Dense(1, activation = 'sigmoid'))

classifier.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
classifier.fit(feature_train, label_train, epochs=500, batch_size
    =5)
_, accuracy = classifier.evaluate(feature_train, label_train)

print('Train Accuracy: %.2f %%' %(accuracy*100))

test_results = classifier.predict_classes(feature_test)

```

```
36 counter = 0
37 for a, b in zip(test_results , label_test):
38     if a==b:
39         counter+=1
40
41 print('Test Accuracy: %.2f %%' %((counter/len(label_test))*100))
```

From line 1 to 3 we import the necessary pandas and keras libraries.

On line 5 and 6 we load the dataset features in a pandas dataframe named feature_df and labels in a pandas dataframe named label_df.

From line 8 to 13 the data in the dataframes features_df and label_df are converted to python lists and stored in a variables named feature and label.

From line 15 to 18 we split the dataset into training and testing dataset. About 70% of the data is used in training and the remaining 30% is used in testing. The lists feature_train and label_train hold the features and labels of the data points that are used in training and feature_test and label_test hold the features and labels of the data points that are used in testing

From line 20 to 26 we create a neural network with an input layer, hidden layers, output layer. The input layer has 16 nodes and the output layer has 1 node. There are 5 hidden layers.
The first hidden layer has 25 nodes, second hidden layer has 20 nodes, third hidden layer has 15 nodes, fourth hidden layer has 10 nodes, fifth hidden layer has 5 nodes. The hidden layers have ReLU (Rectified Linear Unit) function as their activation function and the output layer has sigmoid function as it's activation function.

From line 28 to 32 we selected binary cross-entropy as our loss function and adam as our optimization algorithm. Then we train the neural network using the data points in feature_train and label_train. The neural network will run for 500 epochs. We get training accuracy score of about 82%.

On line 34 we test our neural network using the data points in feature_test and store the test results in a variable named test_results.

From line 36 to 41 we calculate the testing accuracy using test_results and label_test and print the test accuracy score.

We get train accuracy score of about 82%.
We get test accuracy score of about 72%.

6.2 Experiment With Naive Bayes Classifier

We have used naive bayes classifier to classify our dataset. The python code is given below.

```
 1 import pandas as pd
 2 import nltk
 3 
 4 feature_df = pd.read_csv(r'dataset_feature.csv')
 5 label_df = pd.read_csv(r'dataset_label.csv')
 6 
 7 label = list(label_df['0'])
 8 data = []
 9 for a in range(len(label)):
10     t = list(feature_df.loc[a,])
11     feature_map = {}
12     for idx, b in enumerate(t):
13         if b>0:
14             feature_map[idx] = True
15         else:
16             feature_map[idx] = False
17 
18     data.append((feature_map, label[a]))
19 
20 train_data = data[:668]
21 test_data = data[668:]
22 
23 classifier = nltk.NaiveBayesClassifier.train(train_data)
24 
25 
26 print("Train Accuracy: %.2f %%" %(nltk.classify.accuracy(classifier
        , train_data)*100))
27 print("Test Accuracy: %.2f %%" %(nltk.classify.accuracy(classifier,
        test_data)*100))
```

Here from on line 1 and 2 we are importing pandas and nltk library.

On line 4 and 5 we are loading data on the pandas dataframe variables named feature_df, and label_df.

On line 7 the labels in the label_df dataframe in converted into a python list and stored in the variable named label.

On line 8 an empty list variable named data is declared. To use the naive bayes classifier from the nltk library we need to input the data points in the form of a list containing a tuple for each data point. The tuple will have two elements, the first one is a python dictionary and the second one is the label value for that data point. The dictionary will be a dictionary of features of the dataset. For each feature in the dictionary there can be one of two values, True of False. If a feature is found in a data point then we set the value of that feature to True otherwise we have to set it to False. So, for each data point there will be a tuple in the list. The tuples will have a python dictionary and a label. From line 8 to 18 we create that list.

On line 20 and 21 we divide the data into training data and testing data.
70% of the data is used in training and the remaining 30% of the data is used in testing.

On line 23 we create a variable named classifier that stores the naive bayes classifier after training.

On line 26 and 27 we calculate and print the train accuracy and test accuracy.

We get train accuracy score of about 78%.
We get test accuracy score of about 75%.

6.3 Experiment With Support Vector Machine

We used support vector machine (SVM) with linear kernel to predict classes for our dataset. The python code is given below.

```
 1 import pandas as pd
 2 from sklearn import svm
 3 from sklearn import metrics
 4 import numpy as np
 5 
 6 feature_df = pd.read_csv(r'dataset_feature.csv')
 7 label_df = pd.read_csv(r'dataset_label.csv')
 8 
 9 feature = []
10 label = list(label_df['0'])
11 
12 for i in range(len(label)):
13     feature.append(list(feature_df.loc[i,]))
14 
15 feature_train = np.array(feature[:668])
16 label_train = np.array(label[:668])
17 feature_test = np.array(feature[668:])
18 label_test = np.array(label[668:])
19 
20 classifier = svm.SVC(kernel='linear')
21 train_accuracy = classifier.fit(feature_train, label_train).score(
       feature_train, label_train)
22 test_results = classifier.predict(feature_test)
23 test_accuracy = metrics.accuracy_score(label_test, test_results)
24 
25 print("Train Accuracy: %.2f %%" %(train_accuracy*100))
26 print("Test Accuracy: %.2f %%" %(test_accuracy*100))
```

From line 1 to 4 we import the pandas, scikit-learn [1], and numpy libraries.

On line 6 and 7 we load the dataset. We load the features of the dataset into the pandas dataframe named feature_df and we load the labels of the dataset into the pandas dataframe named label_df.

On line 9 we create an empty python list named feature and on line 10 we convert the label values into a list and store it in a variable named label.

On line 12 and 13 we store the values of the feature_df dataframe into a python list by adding the features of each datapoint into the list named feature.

From the 15 to 18 we split the dataset into training data and testing data. 70% of the data will be used for training and 30% of the data will be used for testing. The feature_train variable holds the features of the datapoints that will be used for testing and label_train variable holds the labels of the datapoints that will be used to training. The feature_test variable holds the features of the data points that will be used for testing and label_test variable holds the labels of the datapoints that will be used for testing.

On line 20 we create a SVM classifier with linear kernel. On line 21 we train the classifier using the data in the variables feature_train and feature_label and store the train accuracy score in the variable named train_accuracy. On line 22 we test the classifier using variable feature_test and store the results in a variable named test_results. On line 23 we calculate the test accuracy of the classifier using label_test and test_results and store the accuracy score in a variable named test_accuracy.

On line 25 and line 26 we print the train and test accuracy scores.

We get train accuracy score of about 79%.
We get test accuracy score of about 74%.

6.4 Results

Algorithm	Train Accuracy	Test Accuracy
Neural Network (MLP)	82%	72%
Naive Bayes	78%	75%
Support Vector Machine	79%	74%

Table 6.1: Result Comparison of Different Algorithms

We can see that we get the highest test accuracy with the naive bayes algorithm and the lowest test accuracy with the neural network.

Chapter 7

Conclusion and Future Work

We can see that we get about 72-75% test accuracy with the algorithms we used. For the neural network we get about 72% test accuracy, for the naive bayes algorithm we get about 75% test accuracy and with support vector machine we get about 74% test accuracy. Difference in test accuracy is not very big.

In future we can try get a bigger dataset. With more careful feature selection and extraction, hyperparameter tuning and different algorithms we may get better results. In future we can try to classify competitive programming problems for more than two classes.

Bibliography

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] S. Subramanian, A. Kumar, M. Nagappan, and R. Suresh, "Classification and recommendation of competitive programming problems using cnn," in. Dec. 2018, pp. 262–272, ISBN: 978-981-10-7634-3. DOI: 10.1007/978-981-10-7635-0_20.

[3] V. Athavale, A. Naik, R. Vanjape, and M. Shrivastava, *Predicting algorithm classes for programming word problems*, 2019. arXiv: 1903.00830 `[cs.CL]`.

[4] J. Brownlee. (2019). "A gentle introduction to the bag-of-words model," [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model (visited on 05/26/2021).

[5] (). "1.9. naive bayes," [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html (visited on 05/26/2021).

[6] P. S. Foundation. (). "Regular expression howto," [Online]. Available: https://docs.python.org/3.3/howto/regex.html#matching-characters (visited on 05/26/2021).

[7] (). "Icpc live archive," [Online]. Available: https://icpcarchive.ecs.baylor.edu/index.php?option=com_content&task=view&id=14&Itemid=30 (visited on 05/26/2021).

[8] (). "Ml - support vector machine(svm)," [Online]. Available: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_support_vector_machine.htm (visited on 05/27/2021).

[9] (). "Multilayer perceptron," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron (visited on 05/26/2021).

[10] (). "Problem statement template," [Online]. Available: https://clics.ecs.baylor.edu/index.php?title=Problem_Statement_Template (visited on 05/26/2021).

[11] L. Richardson. (). "Beautiful soup documentation," [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visited on 05/26/2021).