# Natural Language Processing

## Lab File

Name – Tushya Gupta
Roll number – UE233106
Group – 6

# Lab 1

**Q1.** To implement basic text processing operations like Tokenization, Normalization, Stemming, Lemmatization, Stop words removal, Sentence Segmentation etc. on text document.

Code –

```
#include <iostream>

using namespace std;
size_t max_string_size = 1024;

void shift_back(char **tokens, int index, int token_len) {
  for (int i = index; i < token_len - 1; i++) {
    tokens[i] = tokens[i + 1];
  }
}

void remove_stop_words(char **tokens, int &token_len) {
  int n;
  for (int i = 0; i < token_len; i++) {
    n = strlen(tokens[i]);
    if (tokens[i][n - 1] == '.' || tokens[i][n - 1] == ',') {
      tokens[i][n - 1] = '\0';
    } else if (strcmp(tokens[i], "is") == 0 || strcmp(tokens[i], "an") == 0 ||
               strcmp(tokens[i], "the") == 0) {
      shift_back(tokens, i, token_len);
      token_len--;
    }
  }
}

int get_index(char **tokens, char *token, int tokens_len) {
  for (int i = 0; i < tokens_len; i++) {
    if (strcmp(tokens[i], token) == 0) {
      return i;
    }
  }
  return -1;
}

void count_freq(int *freq, char **tokens, int tokens_len) {
  int index;
  for (int i = 0; i < tokens_len; i++) {
    index = get_index(tokens, tokens[i], tokens_len);
    if (index == -1) {
      cout << "Can't get index for " << tokens[i] << "\n";
```

```
      continue;
    } else {
      freq[index] += 1;
    }
  }
}

int main() {
  cout << "Enter a string: ";
  char inp[max_string_size];
  cin.getline(inp, max_string_size);

  int tokens_index = 0;
  char **tokens = new char *[max_string_size];
  int i = 0, start;
  while (inp[i] != '\0') {
    if (inp[i] == ' ') {
      i++;
      continue;
    } else {
      start = i;
      while (inp[i] != ' ') {
        i++;
      }
      char *result = new char[max_string_size];
      int j = 0;
      for (int k = start; k < i; k++) {
        result[j++] = inp[k];
      }
      tokens[tokens_index++] = result;
    }
  }

  remove_stop_words(tokens, tokens_index);

  int *freq = new int[tokens_index];
  for (int i = 0; i < tokens_index; i++) {
    freq[i] = 0;
  }
  count_freq(freq, tokens, tokens_index);

  cout << "=== Tokens ===" << "\n";
  for (int i = 0; i < tokens_index; i++) {
    cout << tokens[i] << "\n";
  }

  cout << "=== Frequency ===" << "\n";
  for (int i = 0; i < tokens_index; i++) {
    if (freq[i] == 0) {
      continue;
    }
    cout << tokens[i] << " = " << freq[i] << "\n";
  }
  return 0;
}
```

```
Enter a string: This is a sentence
=== Tokens ===
This
a
sentence
=== Frequency ===
This = 1
a = 1
sentence = 1
```

# Lab 2

**Q2.** To implement N-gram Language Model.

Code –

```
#include <ctime>
#include <fstream>
#include <iostream>

using namespace std;
size_t max_string_size = 1024;

void shift_back(char **tokens, int index, int token_len) {
  for (int i = index; i < token_len - 1; i++) {
    tokens[i] = tokens[i + 1];
  }
}

void remove_stop_words(char **tokens, int &token_len) {
  int n;
  for (int i = 0; i < token_len; i++) {
    n = strlen(tokens[i]);
    if (tokens[i][n - 1] == '.' || tokens[i][n - 1] == ',') {
      tokens[i][n - 1] = '\0';
    }
    else if (strcmp(tokens[i], "is") == 0 || strcmp(tokens[i], "an") == 0 ||
             strcmp(tokens[i], "the") == 0) {
      shift_back(tokens, i, token_len);
      token_len--;
    }
  }
}

int get_index(char **tokens, char *token, int tokens_len) {
  for (int i = 0; i < tokens_len; i++) {
    if (strcmp(tokens[i], token) == 0) {
      return i;
    }
  }
  return -1;
}

void count_freq(int *freq, char **tokens, int tokens_len) {
  int index;
  for (int i = 0; i < tokens_len; i++) {
```

```
    index = get_index(tokens, tokens[i], tokens_len);
    if (index == -1) {
      cout << "Can't get index for " << tokens[i] << "\n";
      continue;
    } else {
      freq[index] += 1;
    }
  }
}

int count_follwed(char *f, char *s, char **tokens, int tokens_len) {
  int c = 0;
  for (int i = 0; i < tokens_len; i++) {
    if (i + 1 < tokens_len && strcmp(tokens[i], f) == 0 &&
        strcmp(tokens[i + 1], s) == 0) {
      c++;
    }
  }
  return c;
}

float bigram(char *f, char *s, int *freq, char **tokens, int tokens_len) {
  float prob = 0;
  int index = get_index(tokens, f, tokens_len);
  if (index == -1) {
    return 0;
  }

  int c = freq[index];
  int c_2 = count_follwed(f, s, tokens, tokens_len);
  prob = (float)c_2 / c;

  return prob;
}

int main() {
  srandom(time(NULL));
  // cout << "Enter a string: ";
  char inp[max_string_size];
  // cin.getline(inp, max_string_size);

  int tokens_index = 0;
  char **tokens = new char *[max_string_size];
  ifstream file("temp.txt");
  while (file.getline(inp, max_string_size)) {
    int i = 0, start;
    while (inp[i] != '\0') {
      if (inp[i] == ' ') {
        i++;
        continue;
      } else {
        start = i;
        while (inp[i] != ' ' && inp[i] != '\0') {
          i++;
        }
        char *result = new char[max_string_size];
        int j = 0;
        for (int k = start; k < i; k++) {
          result[j++] = inp[k];
        }
```

```
        tokens[tokens_index++] = result;
      }
    }
  }

  remove_stop_words(tokens, tokens_index);

  int *freq = new int[tokens_index];
  for (int i = 0; i < tokens_index; i++) {
    freq[i] = 0;
  }
  count_freq(freq, tokens, tokens_index);

  // cout << "=== Tokens ===" << "\n";
  // for (int i = 0; i < tokens_index; i++) {
  //   cout << tokens[i] << "\n";
  // }
  //
  // cout << "=== Frequency ===" << "\n";
  // for (int i = 0; i < tokens_index; i++) {
  //   if (freq[i] == 0) {
  //     continue;
  //   }
  //   cout << tokens[i] << " = " << freq[i] << "\n";
  // }

  // char first[max_string_size], second[max_string_size];
  // cout << "First word: ";
  // cin >> first;
  // cout << "Second word: ";
  // cin >> second;

  char *word = tokens[random()%tokens_index];
  for (int i = 0; i < 5; i++) {
    float max_prob = 0;
    int max_index = 0;
    for (int j = 0; j < tokens_index; j++) {
      if (freq[j] == 0) {
        continue;
      }
      float prob = bigram(word, tokens[j], freq, tokens, tokens_index);
      if (prob > max_prob) {
        max_prob = prob;
        max_index = j;
      }
    }
    cout << tokens[max_index] << " ";
    word = tokens[max_index];
  }

  return 0;
}
```

```
❯ ./main.o
mat cat sat on mat ↵
```

# Lab 3

**Q3.** Write a program to extract features – TF, TF-IDF score from text.

Code –

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <ctime>
#include <fstream>

using namespace std;
size_t max_string_size = 1024;

void shift_back(char **tokens, int index, int token_len) {
  for (int i = index; i < token_len - 1; i++) {
    tokens[i] = tokens[i + 1];
  }
}

void remove_stop_words(char **tokens, int &token_len) {
  int n;
  for (int i = 0; i < token_len; i++) {
    n = strlen(tokens[i]);
    if (tokens[i][n - 1] == '.' || tokens[i][n - 1] == ',' ||
        tokens[i][n - 1] == '?') {
      tokens[i][n - 1] = '\0';
    } else if (strcmp(tokens[i], "is") == 0 || strcmp(tokens[i], "an") == 0 ||
               strcmp(tokens[i], "the") == 0) {
      shift_back(tokens, i, token_len);
      token_len--;
    }
  }
}

int get_index(char **tokens, char *token, int tokens_len) {
  for (int i = 0; i < tokens_len; i++) {
    if (strcmp(tokens[i], token) == 0) {
      return i;
    }
  }
  return -1;
}

void create_tokens(char **tokens, char *inp, int &tokens_index) {
  int i = 0, start;
  while (inp[i] != '\0') {
    if (inp[i] == ' ') {
      i++;
      continue;
    } else {
      start = i;
      while (inp[i] != ' ' && inp[i] != '\0') {
        i++;
      }
```

```
      char *result = new char[max_string_size];

      int j = 0;
      for (int k = start; k < i; k++) {
        result[j++] = inp[k];
      }

      tokens[tokens_index++] = result;
    }
  }
  remove_stop_words(tokens, tokens_index);
}

bool check_in(char **vocab, char *word, int vocab_index) {
  for (int i = 0; i < vocab_index; i++) {
    if (strcmp(vocab[i], word) == 0) {
      return true;
    }
  }
  return false;
}

void vocab(char ***sentences, int *tokens_index, int sentence_index,
           char **vocab, int &vocab_index) {
  for (int i = 0; i < sentence_index; i++) {
    for (int j = 0; j < tokens_index[i]; j++) {
      if (!check_in(vocab, sentences[i][j], vocab_index)) {
        vocab[vocab_index] = new char[max_string_size];
        strcpy(vocab[vocab_index], sentences[i][j]);
        vocab_index++;
      }
    }
  }
}

int get_freq(char **tokens, char *word, int tokens_index) {
  int res = 0;
  for (int i = 0; i < tokens_index; i++) {
    if (strcmp(tokens[i], word) == 0) {
      res += 1;
    }
  }
  return res;
}

void tf(char ***sentences, char **vocab, float **tf, int *tokens_index,
        int vocab_index, int sentence_index) {
  for (int i = 0; i < vocab_index; i++) {
    for (int j = 0; j < sentence_index; j++) {
      int w_freq = get_freq(sentences[j], vocab[i], tokens_index[j]);
      tf[i][j] = (float)w_freq / tokens_index[i];
    }
  }
}

int sentences_cotaining(char ***sentences, char *word, int *tokens_index,
                        int sentence_index) {
  int res = 0;
  for (int i = 0; i < sentence_index; i++) {
```

```
    for (int j = 0; j < tokens_index[i]; j++) {
      if (strcmp(sentences[i][j], word) == 0) {
        res += 1;
        break;
      }
    }
  }
  return res;
}

void get_idf(float *idf, char ***sentences, char **voc, int vocab_index,
             int sentence_index, int *tokens_index) {
  for (int i = 0; i < vocab_index; i++) {
    int c_sentences =
        sentences_cotaining(sentences, voc[i], tokens_index, sentence_index);
    idf[i] = log(sentence_index / (float)c_sentences);
  }
}

int main() {
  srandom(time(NULL));
  char inp[max_string_size];

  int sentence_index = 0;
  int *tokens_index = new int[max_string_size];
  char ***sentences = new char **[max_string_size];

  ifstream file("temp.txt");
  while (file.getline(inp, max_string_size)) {
    sentences[sentence_index] = new char *[max_string_size];
    tokens_index[sentence_index] = 0;
    create_tokens(sentences[sentence_index], inp, tokens_index[sentence_index]);
    sentence_index++;
  }

  // cout << "=== Sentences ===" << "\n";
  // for (int i = 0; i < sentence_index; i++) {
  //   printf("=== Tokens %d ===\n", i + 1);
  //   for (int j = 0; j < tokens_index[i]; j++) {
  //     cout << sentences[i][j] << "\n";
  //   }
  // }

  char **voc = new char *[max_string_size];
  int vocab_index = 0;
  vocab(sentences, tokens_index, sentence_index, voc, vocab_index);

  printf("=== Vocab ===\n");
  for (int i = 0; i < vocab_index; i++) {
    printf("%s\n", voc[i]);
  }

  float **term_freq = new float *[vocab_index];
  for (int i = 0; i < vocab_index; i++) {
    term_freq[i] = new float[sentence_index];
  }
  tf(sentences, voc, term_freq, tokens_index, vocab_index, sentence_index);

  printf("\n=== Term Frequency ===\n");
  printf("%-11s", "");
```

```
  for (int i = 0; i < sentence_index; i++) {
    printf("S%-3d ", i + 1);
  }
  printf("\n");
  for (int i = 0; i < vocab_index; i++) {
    printf("%-10s ", voc[i]);
    for (int j = 0; j < sentence_index; j++) {
      printf("%.2f ", term_freq[i][j]);
    }
    printf("\n");
  }

  float *idf = new float[vocab_index];
  get_idf(idf, sentences, voc, vocab_index, sentence_index, tokens_index);

  printf("\n=== IDF ===\n");
  printf("%-10s IDF", "");
  printf("\n");
  for (int i = 0; i < vocab_index; i++) {
    printf("%-10s %.2f\n", voc[i], idf[i]);
  }

  printf("\n=== TD-IDF ===\n");
  printf("%-11s", "");
  for (int i = 0; i < sentence_index; i++) {
    printf("S%-3d ", i + 1);
  }
  printf("\n");
  for (int i = 0; i < vocab_index; i++) {
    printf("%-10s ", voc[i]);
    for (int j = 0; j < sentence_index; j++) {
      printf("%.2f ", term_freq[i][j]*idf[i]);
    }
    printf("\n");
  }

  return 0;
}
```

```
=== Vocab ===
i
am
henry
like
college
do

=== Term Frequency ===
           S1   S2   S3   S4   S5   S6
i          0.33 0.33 0.00 0.67 0.33 0.33
am         0.33 0.00 0.00 0.33 0.00 0.00
henry      0.25 0.00 0.25 0.00 0.25 0.25
like       0.00 0.17 0.17 0.17 0.17 0.17
college    0.00 0.25 0.25 0.25 0.00 0.00
do         0.00 0.00 0.25 0.25 0.25 0.25

=== IDF ===
           IDF
i          0.18
am         1.10
henry      0.41
like       0.18
college    0.69
do         0.41

=== TD-IDF ===
           S1   S2   S3   S4   S5   S6
i          0.06 0.06 0.00 0.12 0.06 0.06
am         0.37 0.00 0.00 0.37 0.00 0.00
henry      0.10 0.00 0.10 0.00 0.10 0.10
like       0.00 0.03 0.03 0.03 0.03 0.03
college    0.00 0.17 0.17 0.17 0.00 0.00
do         0.00 0.00 0.10 0.10 0.10 0.10
```