Proyecto Final

DOCENTE	CARRERA	CURSO
PhD(c) Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Algoritmos y Estructura de Datos

1. Datos de los estudiantes

- Grupo: 07
- Integrantes:
  - Acuña Melo, Edgar
  - López Torres Herrera, Luis Rodrigo
  - Luna Flores, Julio Paolo
  - Vilela Arias, Roger

2. Competencias del proyecto

- Analizar e implementar algoritmos efícientes para la solución de problemas computacionales.
- Implementar estructuras de datos adecuadas, según el tipo de problema.

3. Equipos y materiales

- Javascript
- Navegador Web
- Cuenta en Github
- IDE de desarrollo

4. Trabajo de investigación

En el siguiente trabajo se realizó la implementación del clasificador KD-Tree teniendo como base a lo realizado en la practica 04 del curso, siendo el clasificador por frecuencia de palabras el seleccionado para el proyecto y desarrollado en su totalidad, orientandolo a la clasificacion de correos spam y no spam.
Para la compañia de antivirus ESET, el spam es cualquier forma de comunicación no solicitada que se envía de forma masiva (correo electrónico masivo no solicitado, o UBE). Su forma más frecuente es un correo electrónico de publicidad enviado a un gran número de direcciones (correo electrónico de publicidad no solicitado, o UCE), pero el "spamming"también existe a través de menajes instantáneos, de texto (SMS), redes sociales o incluso mensajes de voz. Enviar spam es ilegal en la mayoría de jurisdicciones.Una de las vías más comunes para propagar contenido no solicitado es a través de botnets, grandes redes de dispositivos "zombieïnfectados. A veces los correos en cadena y fraudulentos también se consideran spam, aunque difieren en que en general se reenvían por gente con buenas intenciones.(Fuente: www.eset.com)

Figura 01: Spam (Fuente: www.eset.com)

5. Desarrollo del proyecto

1. El proyecto final se encuentra desarrollado en el archivo TF-IDF.js el cual desarrolla el siguiente contenido.

```
let tokenize=(text)=>{
   const punctuation = ['!', '#', '"', '%', '$', "'", '&', ')', '(', '+', '*', '-', ',', '/', '.
   punctuation.forEach((e) => {
      text = String(text.replaceAll(e, ""));
   });
   return text.toLowerCase().split(/\s+/g);
}
let dictionary=(tokens,dict)=>{
   tokens.forEach((token)=>{
      if(!dict.includes(token)){
         dict.push(token);
      }
   })
   return dict;
};
let vsm=(tokens,dict)=>dict.map((token)=>tokens.reduce((acc,curr)=>curr== token?acc +1:acc,0));
let tf=(vsm,numberOfTokens)=>vsm.map((token)=>token/numberOfTokens);
let idf = (documentTokens,dict)=>dict.map((word)=>Math.log(documentTokens.length/documentTokens.
let tfidf=(tf,idf)=>tf.map((element,index)=>element*idf[index]);
let cosine=(tfIdf1,tfIdf2)=>tfIdf1.reduce((acc,curr,index)=>acc+curr*tfIdf2[index],0) / (Math.s
let dict = [];
let review = [];
let adtfidf = [];
let atk = [];
```

```
function readFile(input) {
   let file = input.files[0];
   let reader = new FileReader();
   reader.readAsText(file);
   reader.onload = function() {
      var content = reader.result;
      var lines = content.split("\r\n");
      for (var count = 1; count < lines.length; count++) {
         const obj = {};
         var rowContent = lines[count].split("|");
         obj.text = String(rowContent[1]);
         obj.classifier = String(rowContent[0]);
         review.push(obj);
      }
      construccionVectores();
      construccionKdTree();
   };
   reader.onerror = function() {
      console.log(reader.error);
   };
}

function construccionVectores() {
   console.table("review");
   atk = review.map((ele, index)=>{
      const obj = [];
      let dTokens = tokenize(ele.text);
      dict=dictionary(dTokens,dict);
      dTokens.forEach((e, i) => {
         obj[i] = e;
      });
      return obj;
   });

   console.log(dict);

   const adVms = [];
   adtfidf = review.map((ele, index)=>{
      const obj = [];
      let dTokens = tokenize(ele.text);
      let dVsm = vsm(dTokens,dict);
      adVms[index] = dVsm;
      let dtf = tf(dVsm,atk[index].length);
      let didf = idf(atk,dict);
      let dtfidf = tfidf(dtf,didf);
      dtfidf.forEach((e, i) => {
         obj[i] = e;
      });
      return obj;
   });
```

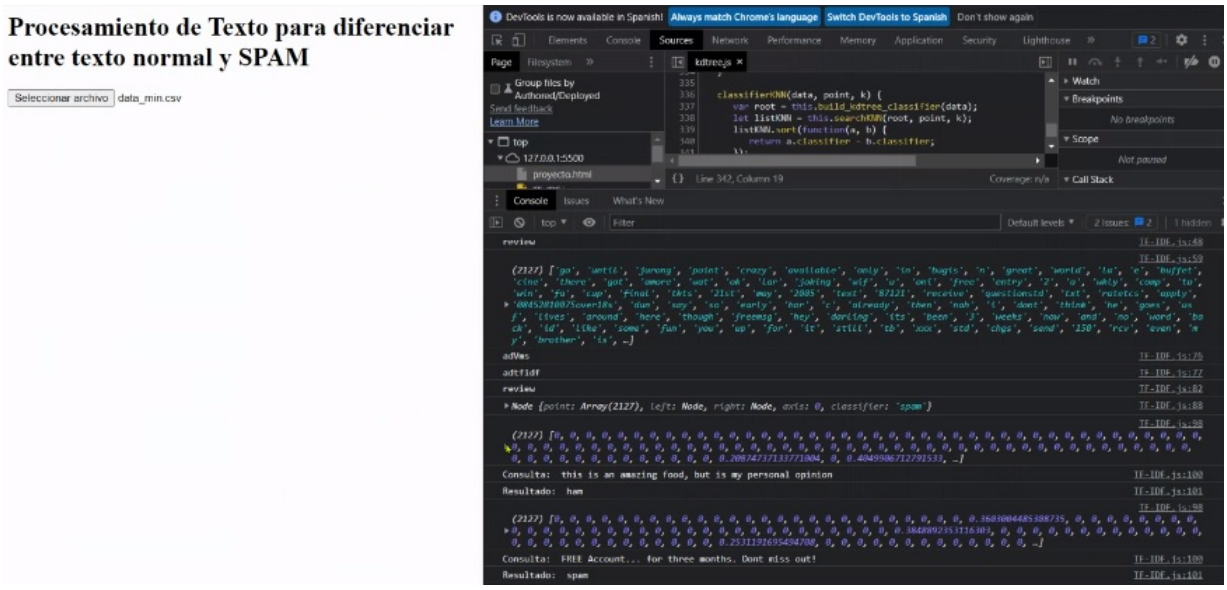
```
    console.table("adVms");
    console.table("adtfidf");

    review.map((ele, index) => {
       ele.vector = adtfidf[index];
    });
    console.table("review");
}

function construccionKdTree() {
    var oKdTree = new KdTree(adtfidf[0].length);
    var root = oKdTree.build_kdtree_classifier(review);
    console.log(root);

    let querys = ["this is an amazing food, but is my personal opinion", "FREE Account... for three months. Dont miss out!"];
    querys.forEach((query) => {
       let queryTokens = tokenize(query);
       let queryVsm = vsm(queryTokens,dict);
       let querytf = tf(queryVsm,queryTokens.length);
       let didf = idf(atk,dict);
       let querytfidf = tfidf(querytf,didf);

       console.log(querytfidf);

       console.log("Consulta: ", query);
       console.log("Resultado: ", oKdTree.classifierKNN(review, querytfidf, 3));
    });
```

2. Asi tambien se ha generado el archivo descriptor.py y se ha usado el archivo datamin.csv

```
import csv
import pandas as pd
import numpy as np
import string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

spam_or_ham = pd.read_csv("data.csv", sep='|', encoding='latin-1')[["v1", "v2"]]
spam_or_ham.columns = ["label", "text"]
punctuation = set(string.punctuation)
print punctuation

##Eliminamos los simbolos de puntuacion
def tokenize(sentence):
    tokens = []
    for token in sentence.split():
        new_token = []
        for character in token:
            if character not in punctuation:
                new_token.append(character.lower())
        if new_token:
            tokens.append("".join(new_token))
    return tokens

```

```
25  spam_or_ham.head()["text"].apply(tokenize)
26  train_text, test_text, train_labels, test_labels = train_test_split(spam_or_ham["text"],
27                                                                      spam_or_ham["label"],
28                                                                      stratify=spam_or_ham["label"])
29
30  print train_text
31  print "========================================="
32  print train_text.astype('U')
33
34  real_vectorizer = CountVectorizer(tokenizer = tokenize, binary=True)
35  train_X = real_vectorizer.fit_transform(train_text.astype('U'))
36  ##test_X = real_vectorizer.transform(test_text.astype('U'))
37
38  print(train_X)
```

3. Cargando el archivo .csv se puede comprobar el funcionamiento adecuado del proyecto



6. Implementación

El desarrollo de la practica se utilizo el mismo repositorio compartido para la practica 04 debido a que se tomo cmo base lo desarrollado en dicha practica, los archivos de codigo fuente del desarrollo de la presente practica, se encuentran alojados en el siguiente enlace:

https://github.com/JulioLunaUNSA/INFORME04.git

imagenes	PRUEBA VECTORIZACION IMAGENES
Practica 04 - Grupo 7.pdf	Informe 04
README.md	Initial commit
TF-IDF.js	Ajustes TF-IDF
data.csv	añadiendo descriptor y metodo de predicion
data_min.csv	CLASIFICADOR NLP + AJUSTE KDTREE
descriptor.py	CLASIFICADOR NLP + AJUSTE KDTREE
kdtree.js	Ajustes TF-IDF
main.html	Commit inicial
opencv.js	PRUEBA VECTORIZACION IMAGENES
p5.min.js	Commit inicial
predicion.js	añadiendo descriptor y metodo de predicion
proyecto.html	Ajustes TF-IDF
prueba.js	Ajustes TF-IDF
sketch.js	CLASIFICADOR NLP + AJUSTE KDTREE

7. Conclusiones

- De acuerdo a los objetivos planteados en el curso, con el desarrollo del proyecto final se logra un correcto analisis e implementacion de algoritmos con efíciencia demostrada para la solución de problemas computacionales.
- Se logro implementar estructuras de datos adecuadas manteniendo como base el KD-Tree, según el tipo de problema que se selecciono adicionalmente tomando como base las otras practicas desarrolladas.
- Se pudo discriminar los diferentes algoritmos teniendo en cuenta las prestaciones que se necesitan para la resolución de problemas computacionales.