UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

ANÁLISIS Y RECONOCIMIENTO DE PATRONES DE CONDUCCIÓN TEMERARIA MEDIANTE ALGORITMOS DE APRENDIZAJE COMPUTACIONAL

TESIS

PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN

PRESENTA

RAMIRO IVÁN LUGO HERNÁNDEZ

DIRECTOR DE TESIS

LUIS CARLOS GONZÁLEZ GURROLA

CHIHUAHUA, CHIHUAHUA. DICIEMBRE 2016

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

ANÁLISIS Y RECONOCIMIENTO DE PATRONES DE CONDUCCIÓN TEMERARIA MEDIANTE ALGORITMOS DE APRENDIZAJE COMPUTACIONAL

TESIS

PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN

APROBADO:

DR. LUIS CARLOS GONZÁLEZ GURROLA, director

DR. FERNANDO MARTÍNEZ REYES, sinodal

DR. ALAIN MANZO MARTÍNEZ, sinodal

DICIEMBRE 2016

CHIHUAHUA, CHIHUAHUA.

30 de noviembre de 2016

I.S. RAMIRO IVÁN LUGO HERNÁNDEZ

Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de Maestro en Ingeniería en Computación, nos es grato transcribirle el tema aprobado por esta Dirección, propuesto y dirigido por el director **Dr. Luis Carlos González Gurrola** para que lo desarrolle como tesis, con el título: **"ANÁLISIS Y RECONOCIMIENTOS DE PATRONES DE CONDUCCIÓN TEMERARIA MEDIANTE ALGORITMOS DE APRENDIZAJE COMPUTACIONAL".**

ÍNDICE

Facultad de Ingeniería
Circuito No.1, Campus Universitario 2
Chihuahua, Chih. C.P. 31125
Tel. (614) 442-95-00 www.fing.uach.mx

4. **Experimentos y resultados**
   - 4.1 Aplicación de DTW sobre datos sin preprocesamiento
   - 4.2 Aplicación de Naive Bayes sobre datos proprocesados
   - 4.3 Aplicación de red neuronal artificial sobre datos preprocesados
   - 4.4 Generación de conjunto de datos para programación genética
   - 4.5 Obtención de función calificadora mediante programación genética
   - 4.6 Análisis de resultados
5. **Conclusiones**

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

A T E N T A M E N T E
"naturam subiecit aliis"

EL DIRECTOR

M.I. JAVIER GONZÁLEZ CANTÚ

FACULTAD DE INGENIERÍA U.A.CH.
CHIHUAHUA
Facultad de Ingeniería
DIRECCIÓN

EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO

DR. FERNANDO RAFAEL ASTORGA BUSTILLOS

Resumen

Hoy en día existe un aumento directamente proporcional entre la obtención de dispositivos tecnológicos y la generación de datos con ellos. La cantidad de datos generada día a día es muy, estos datos incluyen información de todos los ámbitos, como son los datos personales, información acerca del clima, difusión de noticias en tiempo real, entre muchos otros.

El tener a la mano toda esta información ha dado pie a que investigadores traten de utilizarlos en maneras que ayuden a solucionar distintas problemáticas. Dentro de estas problemáticas se encuentra el monitoreo de las vías de tránsito, donde el riesgo a sufrir un accidente ha permanecido desde que el automóvil comenzó a ser más accesible para la población en general. De esta manera, se ha abierto una gran puerta para abordar este problema haciendo uso de todos los datos que se pueden obtener referentes a las vías de tránsito. Asimismo, dado que hoy en día una gran mayoría de la población cuenta con algún tipo de dispositivo inteligente, esto serviría como una herramienta más para la obtención de datos.

Varios beneficios podrían obtenerse de esto, como tener un mejor control en las vías de tránsito, así como mitigar en gran manera el riesgo a propiciar y/o sufrir un accidente automovilístico, los cuales en gran parte son ocasionados debido a la conducción temeraria que practican algunos automovilistas.

Sin embargo, esto no es una tarea sencilla, ya que es necesario definir una estrategia que permita atacar este problema afectando en mínima medida la comodidad, seguridad y privacidad del conductor.

El presente documento plantea una estrategia para captura de datos y su posterior procesamiento mediante algoritmos de aprendizaje máquina para la clasificación de estos, como la red neuronal artificial, y así identificar si los patrones de conducción pertenecen a la clasificación temeraria en base a los datos obtenidos. Adicionalmente se propone una estrategia de calificación de conducción tomando como referencia los patrones identificados en la conducción de un automovilista.

Índice de Contenido

Índice de Contenido

Índice de figuras

Índice de tablas

1

Introducción

En este capítulo de establecen los antecedentes y se hace una revisión del estado del arte, resaltando los trabajos que se relacionan con la problemática abordada en este tema de investigación.

Asimismo se define el problema y la pregunta de investigación. Además, se define la justificación para realizar este trabajo.

Por último, se comenta la delimitación del trabajo y se expone el objetivo principal, así como los objetivos específicos que la investigación busca alcanzar.

1.1. Antecedentes

Hoy en día la tecnología juega un rol elemental en la vida de los seres humanos, con el paso de los años, los avances tecnológicos han sido demasiados, haciendo imposible el hecho de ignorarlos, es por ello que la sociedad se ha visto inmersa en el uso cotidiano de uno o más dispositivos que ayuden a facilitar tareas y/o actividades.

Si bien, estos avances pueden no solamente servir para facilitar actividades cotidianas. El hecho de que cada vez los dispositivos tecnológicos presenten más y mejores funcionalidades no ha pasado desapercibido para grupos de personas dedicados a la investigación, quienes al percatarse de esto han tratado de utilizarlos en distintas maneras sacando provecho de sus características, aunado a que presentan una gran accesibilidad para la mayoría de las personas.

Un claro ejemplo de esto son los smartphones, o teléfonos inteligentes; los cuales han tenido una evolución muy rápida en lo últimos años, así mismo han presentado un gran nivel de aceptación en la sociedad, pudiéndolos considerar como un objeto elemental para la vida de cualquier ser humano hoy en día.

Los avances en los smartphones han sido muy grandes, ya que los elementos de hardware que presentan

en la actualidad son equiparables a los de una computadora promedio, y en algunos casos mejores.

Gracias a esto, algunos investigadores ya han tratado de simular tareas y procesos en smartphones, que anteriormente solamente se podían considerar en una computadora o dispositivos especializados.

Entre esas tareas, una que destaca es la de simular sistemas avanzados de asistencia al conductor, *ADAS (Advanced Driver-Assistance Systems)*. Dichos sistemas tienen la funcionalidad de asistir y alertar a un conductor de un automóvil con el fin de prevenir potenciales accidentes de tránsito, y por ende, reducir en gran medida cualquier riesgo que pueda presentarse tanto para el conductor o terceros.

1.2. Trabajo Relacionado

Esta investigación cuenta con 3 pilares principales, que son primeramente la captura de datos mediante el acelerómetro de un teléfono inteligente [1], que son las señales de los distintos eventos de manejo, después se encuentra la utilización de algoritmos de aprendizaje máquina para la clasificación de dichos datos, y por último la implementación de una metodología para la calificación de recorridos de conducción, basada en la clasificación previa de los datos.

En el apartado de la obtención de datos de conducción, se pueden encontrar varios trabajos que abordan este tema utilizando un teléfono inteligente.

En [2] se utilizaron los datos obtenidos de los ejes X y Y del acelerómetro para la detección de varias maniobras de conducción temeraria. Para detectar la aceleración/desaceleración se hizo uso del eje Y, donde en base a varias pruebas se obtuvo que este evento realizado de manera segura, nunca sobrepasaba más de $0.3g$. Usando dicha información se estableció un umbral mínimo y uno máximo para la fuerza g, donde al exceder dicho límite, se consideraba un evento temeroso. El mismo principio se utilizó al momento de detectar los cambios de carril, solo que en este caso el eje utilizado fue el X, y el umbral establecido fue de $0.1g$.

En [3], se desarrolló una aplicación en android, la cual almacena los datos proporcionados por el acelerómetro y *GPS*, asimismo graba el sonido ambiente utilizando el micrófono del teléfono. Estos datos eran después combinados y analizados para detectar patrones de conducción temeraria. Entre los patrones que se detectaron fueron los correspondientes a cambios de carril, vueltas, frenones y acelerones.

En [4], además de capturar los datos del acelerómetro y *GPS*, también se realizaban grabaciones de video con la ayuda de la cámara integrada en el dispositivo. Esto con el fin de darle al conductor una posterior retroalimentación y así mejorar su desempeño al conducir. Los ejes utilizados para la detección de los patrones fueron el X y el Y, el primero detectaba aceleración y freno, mientras que el segundo detectaba vueltas,

volantazos y cambios de carril. Para todos los eventos se estableció un intervalo seguro situado entre los valores -3*g* y +3*g*.

En [5] además del acelerómetro, se utilizan el giroscopio, magnetómetro, *GPS* y cámara (para video), para luego fusionar datos relacionados entre si de cada uno. En este caso se utilizó un *iPhone* 4, y los eventos detectados fueron vuelta (derecha, izquierda, en u), acelerón y frenón.

En [6] y en [7] es utilizado el sensor de orientación. En el primero se usa en conjunto con el acelerómetro para detectar maniobras de manejo en estado de ebriedad (acelerón, frenón, volantazo, aceleración irregular, vueltas con radio ancho). La detección se realiza mediante ventanas de datos y variación de umbrales. En el segundo, el sensor de orientación es utilizado para detectar el movimiento tanto lateral como frontal. Mientras que el *GPS* se utiliza para obtener la localización del vehículo y su velocidad. Combinando la detección de las maniobras con la información proporcionada por el *GPS*, se usa una técnica de mapeo para mostrar donde se presenta el evento de manejo temeroso en una red de caminos real.

En [8] proponen un sistema que recolecta datos mediante el acelerómetro y sensor de orientación del dispositivo, cuyo análisis se realiza mediante el uso del algoritmo *DTW (Dynamic Time Warping)*. La característica más interesante de este sistema es que en base a la información obtenida con el sensor de orientación, se realiza una calibración para definir los ejes *x*, *y* y *z*, para luego comenzar a leer los datos del acelerómetro.

Claramente existe una fuerte tendencia al uso de tecnologías tanto para usuario, como para los investigadores, donde en muchas veces los recursos son limitados. De esta forma, ambas partes se benefician, el investigador podrá realizar más experimentos al contar con tecnologías económicas, así como el usuario, quien puede tener aplicaciones al alcance de su mano que pueden simular sistemas que de otra manera le sería dificil adquirir. Como es el caso de los Sistemas Avanzados de Asistencia al Conductor.

En cuanto a la clasificación de datos de esta índole usando técnicas de aprendizaje máquina, se encuentran trabajos, donde hacen uso de distintos algoritmos de aprendizaje máquina.

En [5], se propone un enfoque para predecir el estilo de conducción de una persona. La información recolectada mediante varios sensores del dispositivo se fusiona con otra información relacionada. Todo esto en un clasificador basado en el algortimo *DTW* (*Dynamic Time Warping*).

En [9] se presenta un enfoque de reconocimiento de patrones para caracterizar a los conductores en base a su nivel de habilidad. Para realizar este reconocimiento, usaron varios algoritmos de aprendizaje maquina, los cuales incluyen la redes neuronales artificiales multicapa, árbol de decisión, y máquina de vectores de soporte.

En [10] se propuso un análisis de comportamiento de conductor y reconocimiento de rutas mediante modelos ocultos de Markov (*HMM*), esto en dos enfoques. El primero toma el reconocimiento de maniobras

aisladas con concatenación de modelo para construir una ruta genérica, mientras que el segundo enfoque modela la ruta completa como una frase y refina el *HMM* para descubrir maniobras.

En [7], utilizando el sensor de orientación junto con el *GPS*, proponen una técnia que identifica cualquer patrón anormal y lo distingue de otros patrones mediante el uso de redes neuronales.

En [11], se propone una plataforma para evaluación de estilos de conducción. Usando datos obtenidos de la consola del automóvil así como de un *smartphone* y una unidad de medida de inercia, construyen un modelo para predecir si un conducor cuenta con un estilo agresivo de conducción, dicho modelo hace uso del algoritmo de clasificaión Naive Bayes con 5-bin de discretización.

En [12], se presenta un sistema de reconocimiento de estilo de conducción y clasificación de conductor, mediante el uso de los datos obtenidos desde la red de area de control del vehículo, para luego procesarlos mediante el algoritmo de máquina de vectores de soporte, y el algoritmo *k-means*.

En [13], propone un sistema que aplica un suavizado después de la adquisición de los datos, para luego realizar un preprocesamiento de ellos. Después se utiliza el algoritmo *endpoint* para la detección de los eventos, una vez que los eventos se detectan, se usa *DTW* para seleccionar la porción que contiene la perturbación del evento. Luego de estos pasos, se aplica un clasificador bayesiano para identificar un hábito temeroso de uno seguro.

Se puede ver que la dupla *smartphone*/aprendizaje maquina es algo que se está implementando cada vez más y más dentro de la clasificación de patrones de conducción automovilística. En la mayoría de los trabajos se usa el smartphone como herramienta de recolección de datos, y ya sea en el mismo o en un sistema aparte, aplicar algún algoritmo de aprendizaje máquina para determinar si los patrones detectados corresponden a hábitos de conducción temeraria o segura.

Algunos trabajos referentes a la detección y clasificación de patrones de conducción van más allá y se adentran a proponer un método de calificación basandose en los eventos detectados y la clasificación asignada. Trabajos como [14] y [11], han explorado esta área en formas interesantes.

En [14], la forma de calificación de recorridos se maneja mediante un puntaje inicial de 100, al cual se le van restando puntos cada vez que un evento temerario se presenta, la única flexibilidad que se le da a esta técnica es el hecho de tomar en cuenta el clima y el tiempo del dia para definir el puntaje adecuado para descontar.

En [11], la calificación se basa en cuestionarios, una vez que el recorrido ha finalizado, se entregan cuestionarios a algunas personas, las cuales tienen a su disposición la información de los eventos temerarios presentados, y en base a eso otorgar una calificación, donde al ser alta indica un estilo de conducción temerario, de lo contrario se considera seguro.

De esta manera se puede ver que la tripleta recolección/clasificación/calificación ha sido explorada muy poco de manera conjunta, en su mayoría solo se abordan una o dos problematicas. Lo cual da pie a muchas posibilidades para experimentar y diseñar estrategias que acoplen las 3 tareas de una manera fluida y eficaz.

1.3. Problema de Investigación

Actualmente existe un gran número de personas que cuenta con automóviles, lo cual hace que las vías de tránsito se vuelvan un lugar sumamente peligroso, más aún cuando conductores con hábitos de conducción temeraria no miden sus movimientos, aunado al ritmo tan rápido y estresante con el que se vive hoy en día, aumentando en gran medida el riesgo de propiciar y/o verse involucrado en un accidente.

Esta situación no mejora ya que tan solo en el año 2013, el INEGI (Instituto Nacional de Estadística y Geografía) tiene registrado un total de 28,291 accidentes vehiculares, de donde 26,297 (92.95 %) tienen como causa al conductor; mientras que en el año 2014, de 28,337 accidentes presentados, 27,073 (95.53 %) tienen la misma causa, reflejando así un aumento de 2.58 %.

Esto se puede mitigar con sistemas integrados de asistencia al conductor, los cuales analizan el entorno en el que se encuentra éste, así como la manera en que conduce, para después proveerle instrucciones y/o alertas que ayuden a prevenir potenciales accidentes. Sin embargo, estos sistemas son sumamente costosos y en la mayoría de los casos se presentan mediante equipo obstrusivo para el conductor.

Por esto es que su uso ha sido sumamente escaso, de tal manera que el apoyo que pudiesen brindar no se hace presente, por ende el peligro de estar propenso a un accidente de tránsito sigue siendo algo recurrente hoy en día.

1.4. Pregunta de Investigación

¿Cómo se puede simular un sistema de asistencia de conducción de manera económica y no obstrusiva, de tal forma que analice y defina si los patrones de conducción de un automovilista son seguros o temerarios, y por ende, ayudar a reducir el riesgo de propiciar un accidente?

1.5. Justificación

Los riesgos de una conducción temeraria no incluyen solamente a los conductores de automóviles particulares. El transporte público es un área muy afectada debido a los hábitos de conducción de un individuo, donde no solamente se pone en riesgo la integridad de una o dos personas, sino de más gente, ya que estos

medios de transportación son muy concurridos por personas que no fungen como conductores, si no como pasajeros. Así mismo los peatones tienen un alto riesgo de salir perjudicados por esto.

Es por ello que el llevar a cabo esta investigación, tiene como meta, hacer uso de las tecnologías que se encuentran al alcance de la mayoría de las personas para reconocer sus hábitos de conducción y clasificarla ya sea como conductor temerario o como conductor no temerario. Donde de ser clasificada dentro del patrón de conducción temeraria, pueda concientizar a la persona para cambiar sus hábitos de manejo.

Esto puede ayudar no solamente en una etapa posterior a la detección de los patrones, ya que también es posible utilizarse como herramienta disuasiva, donde el conductor al saber que se están monitoreando sus hábitos de manejo tratará de evitar que el acelerómetro registre señales que indiquen una clasificación de manejo temerario al ser analizadas.

Todo esto en conjunto tiene como meta el reducir de manera paulatina los accidentes automovilísticos donde el causante sea el conductor.

1.6. Delimitación

La investigación se realizará en la ciudad de Chihuahua, donde los datos se recolectarán de forma continua durante la duración de esta. La cual abarca desde octubre del 2014 a mayo del 2016.

Se tiene la limitante de que las pruebas solo pueden realizarse en autos propios o prestados por la institución.

Asimismo los teléfonos con los que se cuenta ya no son considerados de gama alta, por ende, de necesitarse algún tipo de hardware no presente en ellos para alguna prueba, se procedería a conseguir algunos de forma momentánea.

1.7. Objetivo General

Identificar patrones por medio de técnicas de *Machine Learning* mediante el uso de señales obtenidas a través del acelerómetro de *smartphones*, que clasifiquen si los hábitos de conducción de un automovilista son temerarios o seguros.

1.8. Objetivos Específicos

- Identificar y analizar los algoritmos que han probado ser eficaces en la detección de patrones de señales de acelerómetro.

- Proponer la utilización de un algoritmo que tenga un mejor resultado al momento de detectar patrones.
- Recolectar señales de acelerómetro de *smartphones* de manera automática para su posterior análisis
- Probar las señales recolectadas con el algoritmo propuesto
- Diseñar un sistema que analice automáticamente patrones de conducción con el algoritmo propuesto para clasificar el tipo de conductor
- Diseñar una estrategia de calificación de la conducción en base al análisis en los patrones de conducción de un automovilista

2

Marco Teórico

En este capítulo se explican los distintos algoritmos y técnicas utilizadas durante este trabajo. Yendo desde lo básico, que es aprendizaje máquina, pasando por la metodología de preprocesamiento elegida, la cual es la Bolsa de Palabras (*Bag of Words*), para luego pasar a los algoritmos de clasificación, donde se encuentran *Dynamic Time Warping*, *Naive Bayes* y Red Neuronal Artificial, por último se explica el funcionamiento de la programación genética, la cual ayudó a generar funciones de calificación de recorridos.

2.1. Aprendizaje Máquina

Una computadora realiza tareas mediante distintas acciones, dichas acciones son, comúnmente, definidas por un usuario humano. Sin embargo, conforme la computación se ha ido haciendo más compleja, aunado a las tareas que una computadora puede y se le es asignada realizar se vuelven más complicadas, la idea de encontrar alguna manera en que esta pueda decidir por si sola que acciones tomar ha ido creciendo gradualmente.

Es aquí donde entra en juego el aprendizaje máquina, el cual se centra en el concepto clave de aprender de la información, que es lo único que una computadora tiene a su disposición. Estableciendo el aprendizaje en términos de comportamiento humano o animal, se tiene que este se obtiene gracias a la experiencia. Una vez que el aprendizaje es adquirido, es entonces que un ente puede adaptarse o ajustarse ante nuevas circunstancias. Dividiendo el aprendizaje en fases, se tienen el recordar, adaptar, y generalizar: reconocer la última vez que se estuvo en una situación (ver la información), se intentó realizar una cierta acción (se dio un resultado) y funcionó (fue correcto), de tal manera que se intentará de nuevo, o en el caso de no haber funcionado, se intentará algo distinto. La generalización se refiere a reconocer la similitud entre dos situaciones de tal manera que lo que se aplicó en una, pueda ser aplicado en la otra.

El aprendizaje máquina, trata de hacer que las computadoras modifiquen o adapten sus acciones, de tal manera que estas se vayan volviendo más precisas, donde dicha precisión se mide en que tan bien las opciones elegidas, reflejan a las que son correctas. Conforme el aprendizaje máquina ha ido evolucionando a lo largo del tiempo, se han simulado elementos ya existentes en otras disciplinas con el fin de imitar su funcionamiento y adaptarlo a un algoritmo, el cual le dé a la computadora una manera específica de aprendizaje. Entre estas se encuentran la neurociencia, biología, estadísticas, matemáticas y física.

Entre los conceptos que se han tomado prestados de esas disciplinas con el fin de crear algoritmos de aprendizaje máquina, se encuentra la red neuronal, la cual realiza el aprendizaje mediante la imitación de las funciones elementales de una neurona cerebral humana. Otro ejemplo es la minería de datos, la cual busca extraer información significativa y valiosa de conjunto de datos enormes, como si se encontrara una piedra preciosa dentro de una mina.

El fin de aprender, es ir mejorando en una cierta tarea conforme a la práctica, la cual otorgará experiencia, sin embargo, la computadora no tiene una lógica propia que le indique si está mejorando o no. En algunos casos, al algoritmo se le puede decir la respuesta correcta para así obtenerla la próxima vez, donde la intención sería hacerlo un mínimo número de veces a manera de que el algoritmo fuera encontrando el solo las respuestas correctas. Una alternativa es también decirle al algoritmo si la respuesta es correcta o no, pero no decirle como encontrarla, a manera de que el la busque. Una variante a esto es asignarle una calificación a la respuesta dependiendo de qué tan correcta es, en vez de solo correcto o incorrecto. Por último, existen ocasiones en que no hay respuestas correctas, solo se busca que el algoritmo encuentre datos que tengan algo en común. Estas distintas formas de indicarle a los algoritmos como buscar las respuestas correctas han originado una clasificación de estos:

- Aprendizaje supervisado: Un conjunto de entrenamiento con respuestas correctas se provee y basado en este conjunto, el algoritmo generaliza para responder a todas las posibles entradas que se presenten posteriormente.

- Aprendizaje no supervisado: No se proveen las respuestas correctas, en vez de eso el algoritmo trata de identificar similitudes entre las entradas, de cual manera que tengan algo en común y puedan ser categorizadas juntas.

- Aprendizaje reforzado: Se encuentra entre el aprendizaje supervisado y no supervisado. Al algoritmo se le indica cuando la respuesta es incorrecta, mas no se le indica el cómo corregirla. De tal manera que debe de explorar e intentar distintas posibilidades hasta encontrar el cómo hacerlo.

2.2. Red Neuronal Artificial

Una red neuronal es un algoritmo de aprendizaje máquina, el cual trata de simular el funcionamiento de un cerebro real. El cual modela los elementos principales que se presentan en las conexiones entre neuronas reales, como las neuronas en sí, sus sinapsis, la manera en que se propagan y el cómo se activan [15].

2.2.1. Neuronas McCulloch and Pitts

El modelo más simple de una neurona es el *McCulloch and Pitts*, véase 2.1.

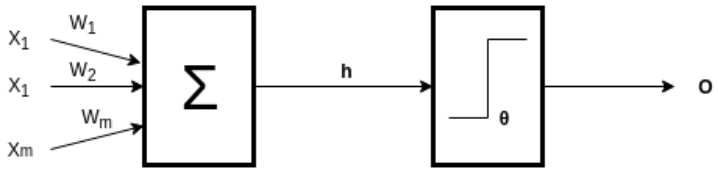

Figura 2.1: Neurona McCulloch and Pitts

Incluye 3 elementos:

- Un conjunto de entradas ponderadas, donde w_i indica la sinapsis.
- Una sumatoria, la cual suma todas las entradas.
- Una función de activación, la cual decide si una neurona se activa en base a las entradas proporcionadas, dependiendo si el resultado de la función sobrepasa cierto umbral.

El modelo describe una neurona simple, donde x_i denota una entrada dada, mientras que w_i indica la fuerza de la conexión, o de la sinapsis, dicha fuerza es llamada peso. Este peso influye directamente en la fuerza de la señal total, es decir, ayudará a indicar si la neurona se activa o no en base a las entradas proporcionadas. Para esto cada peso de cada sinapsis asociada a cada entrada es multiplicado por dicha entrada, $x_i * w_i$, cada multiplicación se irá sumando.

$$h = \sum_{i=1}^{m} w_i x_i \tag{2.1}$$

La ecuación 2.1 indica lo explicado anteriormente, el total de lo que la neurona procesará será la sumatoria de la multiplicación entre una entrada y su peso asociado, esto para todas las entradas proporcionadas.

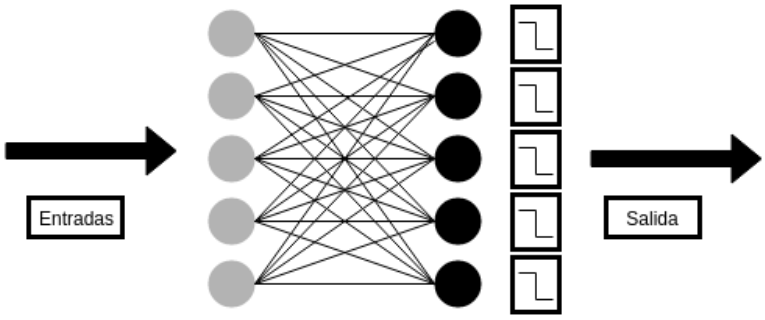

Figura 2.2: Estructura del perceptrón

La sumatoria resultante será entonces procesada por la neurona, donde habiendo definido un umbral anteriormente, dígase $\theta = 0$, se definirá si ésta se activará; suponiendo que el resultado hubiese sido 0.5, entonces $0.5 > 0$, por ende, la neurona se activaría, arrojando como salida 1, de lo contrario arrojaría 0.

2.2.2. Perceptrón

Una neurona tiene un funcionamiento, sin embargo, es muy díficil que una sola pueda solucionar un problema, ya que el resultado que esta arroje una vez que procese entradas variará muy poco, debido a la limitación de tener sólo una neurona realizando el proceso, asimismo, aunque los pesos asociados a las entradas se actualicen, el resultado de la clasificación no se verá realmente afectado. De tal manera que para que puedan usarse de manera útil, lo más viable es poner conjuntos de neuronas, también llamadas nodos.

Teniendo las neuronas en una red, estas ya tienen la posibilidad de aprender. Dicho aprendizaje es supervisado, es decir, se aprende de ejemplos, esto es que se aprenderá de un conjunto de datos que tiene entradas y salidas asociadas de manera correcta. El aprendizaje se obtiene cambiando o adaptando algo, en el caso de las redes neuronales, lo que se cambia comúnmente son los pesos, es decir, el aprendizaje se realiza entre neuronas.

El perceptrón es entonces un conjunto de neuronas *McCulloch and Pitts*, las cuales tienen un conjunto de entradas y otro de pesos (ver 2.2).

Del lado izquierdo se tienen las entradas, es importante señalar que los círculos grises son mera representación esquemática, ya que éstas no son consideraras neuronas. Caso contrario, los círculos negros si indican cada uno una neurona. Asimismo las líneas que conectan los círculos fungen como los pesos. Al tener varias entradas y varias neuronas, se deben identificar de formar correcta los pesos, de tal manera que cada uno se etiquete mediante w_{ij}, donde i correspondería una entrada y j correspondería la neurona a la que se está conectado.

Al tener las entradas y los pesos definidos, solo sería cuestión de aplicar la ecuación 2.1 y verificar cuales

neuronas sobrepasan el umbral definido. El esquema de la Figura 2.2 muestra 5 neuronas, de tal manera que la salida será un vector con 5 elementos, por ejemplo (0, 1, 0, 1, 0), donde cada 0 indicaría que esa neurona no se activó, de lo contrario un 1 indica que si lo hizo. Comparando este vector con el resultado esperado se puede entonces ver cual neurona arrojó una salida correcta y cual no. El resultado óptimo hubiera sido obtener los mismos valores que los del resultado esperado. Cuando esto no sucede entonces se tiene un error, el cual se puede calcular mediante la ecuación 2.2 que define la función de error.

$$y_k - t_k \tag{2.2}$$

Donde y_k es el resultado obtenido y t_k es el resultado esperado. Dicha función de error es útil para actualizar los pesos, esto con el fin de que en una nueva iteración, pueda ser posible obtener el resultado esperado en las neuronas donde se presentó error. La actualización de un peso se realiza con la ecuación 2.3.

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i \tag{2.3}$$

Donde al peso original se le suma el incremento en el peso multiplicado por un factor de aprendizaje, representado por η el cual indica que tanto cambiará el peso. Si el valor de η es muy alto, los valores de los pesos asociados a cada entrada aumentarán o disminuirán demasiado en cada iteración, haciendo que muy dificilmente se llegue a un valor que le permita a la red neuronal generar una clasificación adecuada de las entradas. Si por el contrario η es muy pequeño, puede que llegar al valor de los pesos que permita una correcta clasificación demore demasiado tiempo. Los valores recomendados son $0.1 < \eta < 0.4$. Existe otro elemento en la estructura de un perceptrón llamado bias, el cual ayuda a cuando las entradas no permiten que los pesos se actualicen, un ejemplo es cuando todas las entradas son 0, debido a que la función de actualización usa el incremento resultante, el cual a su vez para ser obtenido se debe multiplicar el error por la entrada asociada, significa que el incremento es de 0, por ende no hay actualización. De esta manera el bias, comúnmente con un valor fijo de -1 o 1, actua como una entrada más, con un peso propio asociado a la neurona. Así, independientemente de que las entradas sean todas 0, el bias permite que si haya un valor diferente de 0 que entre a la neurona, y por ende, puede existir una actualización posterior para los pesos (ver Figura 2.3).

2.2.3. Perceptrón Multicapa

Un perceptrón sencillo puede solucionar varios problemas, sin embargo hay ocasiones en que una estructura de este tipo no aprende lo suficiente y/o aprende de manera incorrecta, por ende, nunca se llega a

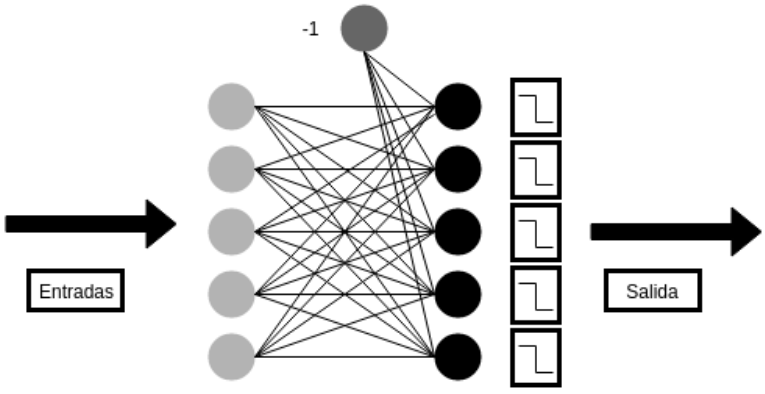

Figura 2.3: Bias en el perceptrón

la solución esperada. Para evitar esto, lo que se hace es agregar una capa más al perceptrón, convirtiéndolo entonces en un perceptron multicapa, como se ilustra en la Figura 2.4.

Esto hace de la red un modelo más complejo, y por ende, más fuerte. Este modelo es muy parecido al perceptrón simple, pero cuanta con varias diferencias. En este modelo hay dos etapas, una etapa que se ejecuta hacia adelante, y la segunda que se ejecuta hacia atrás. La primera se refiere a usar las entradas con los pesos asociados a cada una para así calcular los valores de activación de las neuronas. En este caso, como existe una capa más, llamada capa oculta, los valores de activación no serán en si la salida, si no de la capa oculta, los cuales fungen como las entradas para la capa de salida. La segunda se refiere a calcular el error de las neuronas y actualizar los pesos. Para calcular el error, en este caso se usa la función de error de suma de cuadrados, definida en la ecuación 2.4.

$$E(t,y) = \frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2 \tag{2.4}$$

Que en este caso es el error para todas las neuronas de la red. Es importante mencionar que la función de activación utilizada en el perceptrón simple no es la mejor en el caso del perceptrón multicapa, debido a que es discontinua, de tal manera que los valores para generar la activación son 0 o 1 siempre, nunca un valor intermedio. Una función de activación que es ampliamente usada es la sigmoide, la cual es muy parecida, pero es continua y por ende permite diferenciarse para obtener el gradiente. El cual sirve para saber en qué dirección se debe reducir el error a la hora de actualizar los pesos. Esta se ilustra en la ecuación 2.5.

$$a = g(h) = \frac{1}{1 + exp(-\beta h)} \tag{2.5}$$

Donde β comúnmente es un parámetro positivo, y h es la sumatoria obtenida de multiplicar cada par de entrada y peso asociado a esa neurona. Su gráfica se presenta en la Figura 2.5.

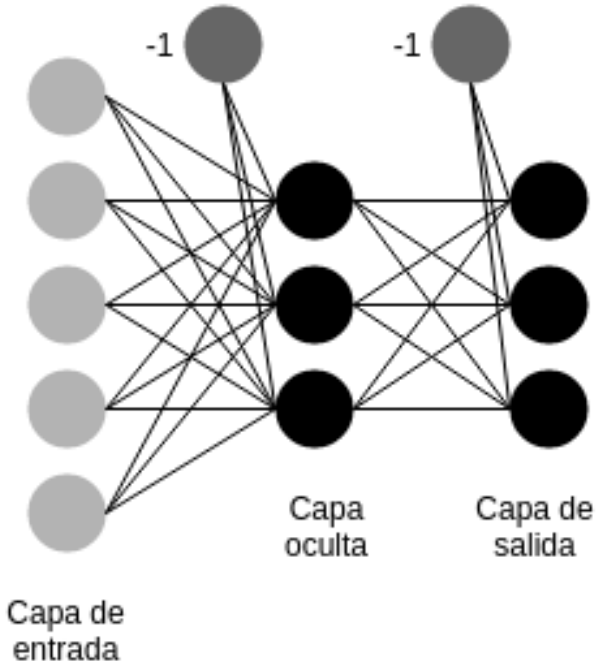

Figura 2.4: Estructura del perceptrón multicapa

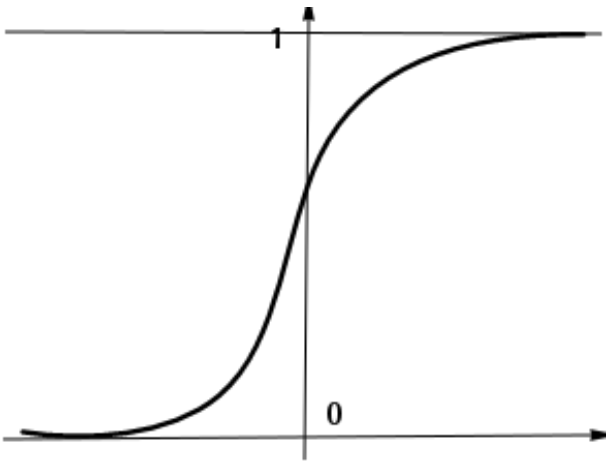

Figura 2.5: Gráfica de función sigmoidal

Teniendo esto en consideración, se procede a ejecutar la segunda etapa para la actualización de los pesos. En un principio se calcula el error en la salida usando la ecuación 2.6.

$$\delta(\kappa) = (y\kappa - t\kappa)y\kappa(1 - y\kappa) \tag{2.6}$$

Esto es, multiplicar a la resta de la salida obtenida menos la salida esperada, la salida obtenida, y todo esto multiplicarlo a la resta de 1 menos la salida obtenida. Después se calcula el error en la capa oculta mediante la ecuación 2.7.

$$\delta_h(\zeta) = a_\zeta(1 - a_\zeta)\sum_{k=1}^{N} w_\zeta \delta_o(k) \tag{2.7}$$

Esto es, multiplicar lo obtenido de la función de activación de la neurona que se esté tratando a la resta de 1 menos nuevamente esto obtenido, por último a eso se le multiplica la sumatoria de los pesos asociados multiplicados por el error de la capa de salida. Para finalmente actualizar los pesos de ambas capas, tanto de la de salida como de la oculta. Para la capa de salida se usa la ecuación 2.8.

$$w\zeta\kappa \leftarrow w\zeta\kappa - \eta\delta o(\kappa)a_\zeta^{hidden} \tag{2.8}$$

Esto es igual a como se actualiza en el perceptrón simple, solamente es restar al peso la multiplicación entre el error que es el obtenido de la capa de salida, lo que funge ahora como entrada es el valor que se obtuvo en la neurona como valor de activación y el factor de aprendizaje definido. Para la capa oculta, es prácticamente igual, solo que ahora el error es el que se obtuvo en la capa oculta, y esta vez sí se utilizan las entradas para la multiplicación, y no valores de activación. Esto se define en la ecuación 2.9.

$$v_t \leftarrow v_t - \eta\delta_h(\kappa)x_t \tag{2.9}$$

En todas estas ecuaciones para el cálculo de errores y actualización de pesos, las letras griegas indican índices fijos. Ambas etapas del perceptrón multicapa se ejecutarán un número de veces predefinido, de tal manera que se entrene hasta cierto punto, para después mediante datos de validación comprobar que tan bien aprendió la red en base a que tan bien está generalizando. Si los resultados son aceptables ahí se deja, de lo contrario se repite todo el proceso pero con un número mayor de iteraciones para entrenar [15]

2.3. Naive Bayes

El algoritmo de clasificación Naive Bayes tiene como elemento principal de su funcionalidad el uso del teorema de Bayes como en la ecuación que se muestra a continuación.

$$P\left(C_L|F_1,...F_n\right) \tag{2.10}$$

Su nombre, Naive Bayes (Bayes Ingenuo), se debe a que el algoritmo realiza una serie de suposiciones ingenuas sobre los datos. Asume que todas las características en el conjunto de datos tienen la misma importancia y son independientes, es decir, una característica no afecta a la otra [15].

El algoritmo hace uso de probabilidad condicional, donde para clasificar un evento donde es necesario calcular su probabilidad dada un determinado número de caracteristicas (ecuación 2.10).

La ecuación del Teorema de Bayes, donde el númerador se define en terminos de la distribución de probabilidad conjunta, lo que da como resultado la segunda ecuación y el denominador se descarta al ser constante.

$$P(C_L|F_1,...,F_n) = \frac{P(C_L) * P(F_1,...,F_n|C_L)}{P(F_1,...,F_n)} \tag{2.11}$$

$$P\left(C_L\right) * P\left(F_1,...,F_n|C_L\right) = P\left(C_L,F_1,...,F_n\right) \tag{2.12}$$

La ecuación anterior es redefinida con el uso de la regla de cadena con lo que se obtine la siguiente ecuación.

$$P\left(C_L,F_1,...,F_n\right) = P\left(F_i|F_{i+1},...,F_n,C_L\right) \tag{2.13}$$

Con el uso de la suposición de independencia entre variables la ecuación anterior se puede definir como:

$$P\left(F_i|F_{i+1},...,F_n,C_L\right) = P\left(F_i|C_L\right) \tag{2.14}$$

Finalmente la fórmula de Naive Bayes se puede definir como la ecuación que se muestra a continuación donde la probabilidad de *L* para una clase *C* dadas las evidencias de *F1* a *Fn* es igual a la multiplicación de cada parte de evidencia, por la probabilidad posterior, por el factor *1/Z* que convierte el resultado a probabilidad.

$$P(C_L|F_1,...,F_n) = \frac{1}{Z}p(C_L)\prod_{x=1}^{n}p(F_i|C_L) \tag{2.15}$$

Entre las ventajas que presenta está el ser simple, rápido y altamente efectivo, funciona bien incluso con conjuntos de datos que contienen ruido o perdida de información, aunado a esto, se desempeña correctamente

independientemente del tamaño del conjunto de entrenamiento, ya que requiere un número pequeño de datos para entrenarse, sin embargo, también presenta buenos resultados con un conjunto de mayor tamaño.

2.4. Dynamic Time Warping

La medida de distancia *DTW* es una técnica muy común en cuanto a reconocimiento de voz, ya que permite un mapeo no lineal de una señal a otra minimizando la distancia entre ambas.

Este método tiene una flexibilidad que le permite a dos series de tiempo que son similares pero localmente fuera de fase, alinearse de manera no lineal.

Suponiendo que se tienen dos series de tiempo, una secuencia Q de longitud n, y una secuencia C de longitud m.

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n \tag{2.16}$$

$$C = c_1, c_2, \ldots, c_i, \ldots, c_n \tag{2.17}$$

Para alinear estas dos secuencias, primero es necesario construir una matriz de n por m donde en dicha matriz el elemento (i^{th},j^{th}) corresponde a la distancia cuadrada, $d(q_i, c_j) = q(q_i, c_j)^2$, la cual es el alineamiento entre los puntos q_i y c_j

Para encontrar el mejor emparejamiento entre estas dos secuencias, se recupera un camino a través de la matriz que minimiza la distancia cumulativa total entre ellas. El camino óptimo es el camino que minimiza el costo de deformación.

$$DTW(Q,C) = min\left\{ \sum_{k=1}^{k} w_k \right. \tag{2.18}$$

Donde w_k se refiere al elemento $(i,j)_k$ de la matriz que también pertenece al elemento *k-esimo* de un camino de deformación W, un conjunto de elementos que representan un mapeo entre Q y C.

Este camino se puede encontrar usando programación dinámica para evaluar la siguiente recurrencia.

$$y(i,j) = d(q,c) + min\left\{ y(i-1,j-1), y(i-1,j), y(i,j-1) \right. \tag{2.19}$$

Donde $d(i,j)$ es la distancia encontrada en la celda actual, y $y(i,j)$ es la distancia cumulativa de $d(i,j)$ y las distancias cumulativas mínimas de las tres celdas adyacentes (ver Figura 2.6).

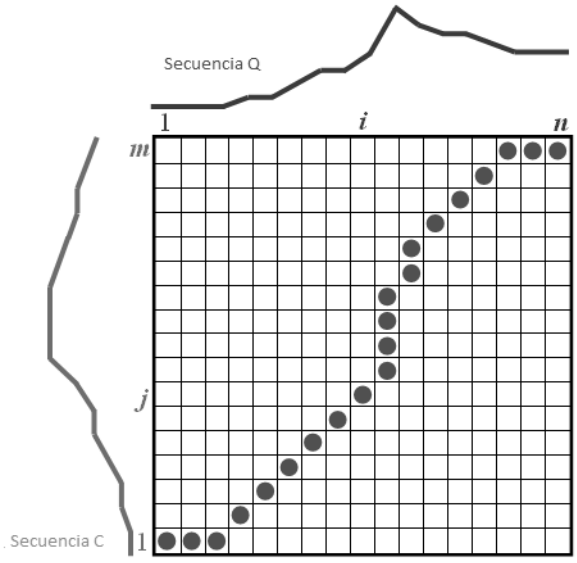

Figura 2.6: Alineación con DTW

2.5. Bolsa de Palabras para Series de Tiempo

Esta metodología, propuesta en [16] es usada en gran medida para el procesamiento de textos, de tal manera que ayude a clasificar el contenido de estos. La idea principal es guardar las palabras representativas de un determinado tema, con el fin de clasificar posteriormente contenidos basándose en la ocurrencia de dichas palabras. En el caso de series de tiempo, la técnica de *bag of words* se ha adaptado a manera de que se extraigan características representativas de estas. La metodología involucrada en esta técnica conlleva 4 tareas secuenciales:

- Suavizado de las series de tiempo (Smoothing)
- Segmentación de las series suavizadas
- Creación de codewords y codebooks
- Generación de histogramas

2.5.1. Suavizado

Las señales o series de tiempo en esporádicas ocasiones se presentan en una forma óptima, es sumamente común que contengan cierta cantidad de ruido, lo cual para fines de un algoritmo de clasificación no es

algo bueno. Para ello es entonces que usa el suavizado con el fin de eliminar la mayor cantidad de ruido posible, sin afectar el comportamiento de la señal, de tal manera que se puedan extraer características lo más significativas posibles. Técnicas como transformadas de *Wavelets* o *Fourier* han sido propuestas en la literatura para hacerse cargo de esta tarea. En las Figuras 2.7 y 2.8 se muestra una señal de datos crudos, y una señal de datos con un suavizado ya aplicado sobre ellos, respectivamente.

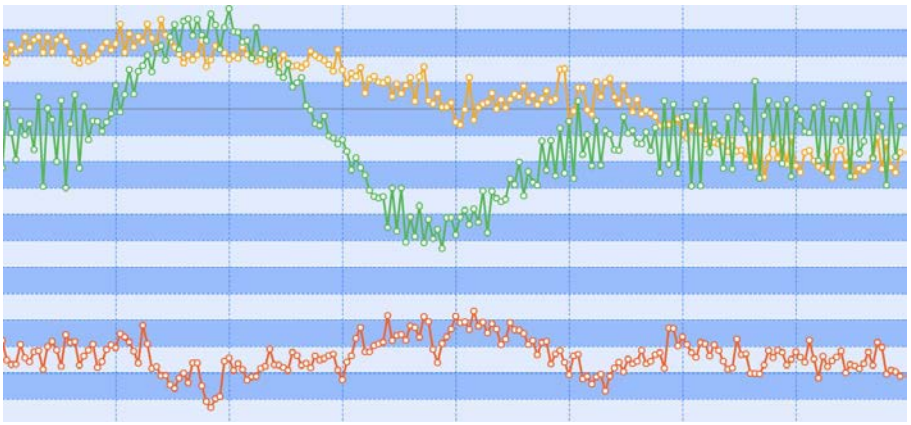

Figura 2.7: Señal sin suavizado

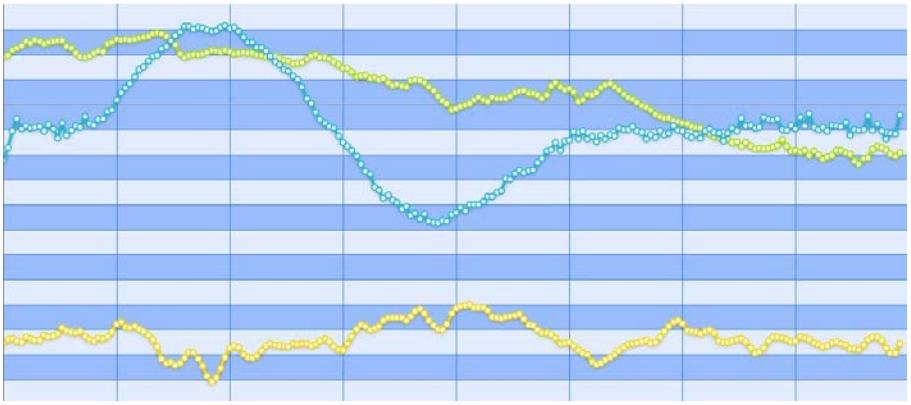

Figura 2.8: Señal con suavizado

2.5.2. Segmentación

Una vez teniendo las señales suavizadas, estas deben de segmentarse para generar la bolsa de palabras, donde cada uno de los segmentos ayude a proveer una noción del comportamiento de los distintos eventos presentes en cada una de las señales. Para realizar esto se define un valor de tamaño de ventana, en otras palabras, el tamaño que tomará cada uno de los segmentos, de tal manera que se va recorriendo toda la señal generando segmentos de igual tamaño hasta llegar al final de esta. En dado caso de no completar el

número necesario de puntos para un segmento en la última parte, dichos puntos se descartan. Es importante mencionar que el recorrido para la segmentación puede ser con traslape o sin él. La Figura 2.9 muestra un ejemplo de una segmentación utilizando una ventana deslizante de 10 puntos.

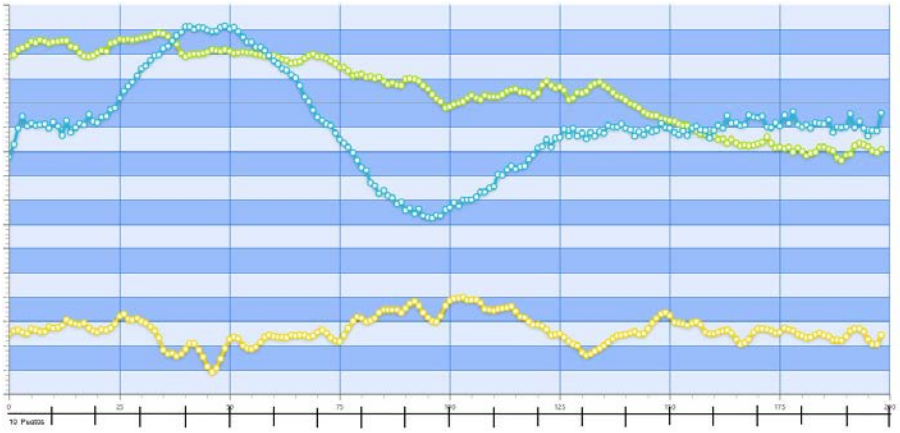

Figura 2.9: Segmentado de señal

2.5.3. Generación de Codewords y Codebook

Los segmentos son un paso anterior a obtener la representación que proveerá las características de los distintos eventos presentados en las señales. En otras palabras, se puede decir que los segmentos son los datos que se usarán de entrenamiento para tratar de darle una definición más certera a las distintas características de los distintos eventos. Para ello, se realiza un proceso de clustering sobre los segmentos, mediante el uso del algoritmo *k-means*, esto con el fin de obtener un promedio de las señales, es decir, agrupar todos los segmentos que se parezcan entre sí para que solo un codeword represente a ese grupo, o sea, el centro de este. La Figura 2.10 ilustra este funcionamiento. La cantidad de codewords que se generan es un numero definido con anterioridad, dicho número normalmente oscila entre 10 y 50, esto ayuda en una etapa posterior como de clasificación, donde esta variación en el número de codewords puede arrojar distintos resultados.

Debido a la naturaleza del problema y la forma en que se está atacando, las capturas del sensor acelerómetro se registran de manera indistinta en los 3 ejes y por lo tanto se agrupan de manera indistinta, logrando con ello una generación de codewords más efectiva, representado un comportamiento total de los eventos en 3 dimensiones que pueden ocurrir al conducir un vehículo. Al tener todos los codewords por clase, cuyo total seria la cantidad definida anteriormente, por tres debido a que se manejarían 3 ejes, por el número de tipos de eventos, tal como lo ilustra la Figura 2.11; solo es cuestión de unirlos para entonces generar un codebook, el cual es utilizado posteriormente para generar histogramas comparando señales prueba.

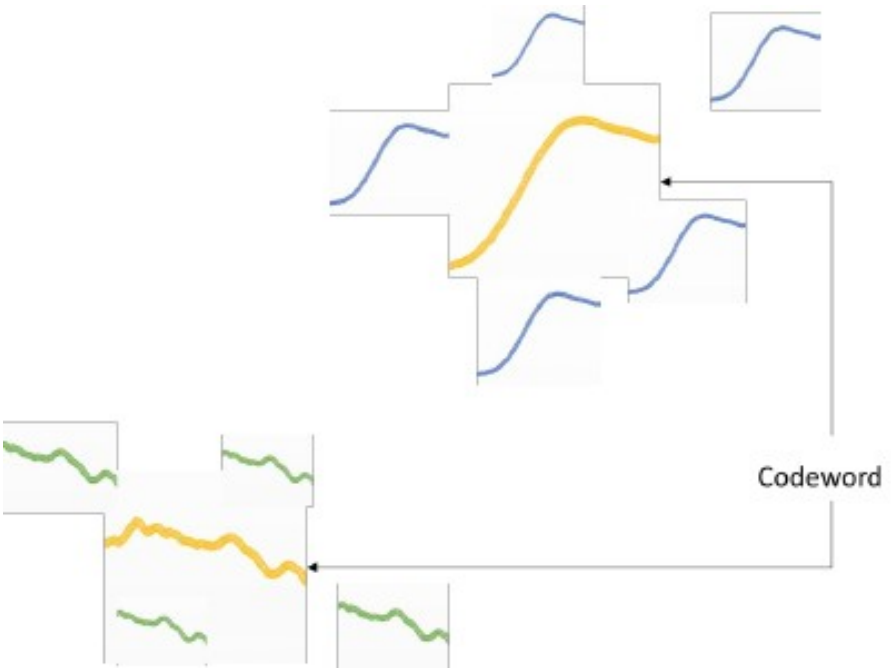

Figura 2.10: Generación de Codewords

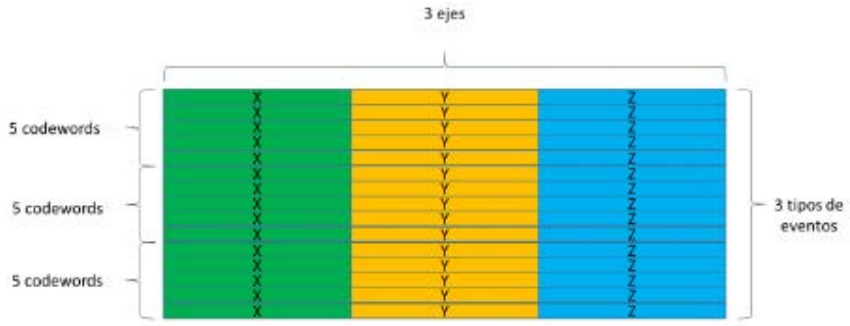

Figura 2.11: Formación de Codebook

2.5.4. Generación de Histogramas

Con el codebook generado, el cual contiene codewords representativas de cada uno de los posibles eventos presentes en las señales, se procede entonces a generar la comparación con las señales de prueba. Estas señales deben de ser segmentadas al igual que las señales de entrenamiento, esto para que cada uno de los segmentos se compare contra cada uno de los codewords presentes en el codebook. Es importante señalar que debido a que se tienen codewords de todos los eventos, asimismo de que la temporalidad ya no se hace presente durante la comparación, es probable que algunos segmentos lleguen a asemejarse a algún codeword que no pertenece al evento correspondiente. La comparación de los segmentos con los codewords se realiza mediante el cálculo de distancia euclidiana, donde de todas las comparaciones, se obtiene la que haya re-

sultado con la distancia mínima. Esto se realiza por cada segmento, donde en cada uno se le va agregando una unidad de incidencia al codeword que haya resultado el más parecido; de tal manera que al final de la comparación de todas las señales se habrá generado un histograma, donde a cada registro se le asignara el tipo de evento al que corresponde verdaderamente. La Figura 2.12 muestra este procedimiento.

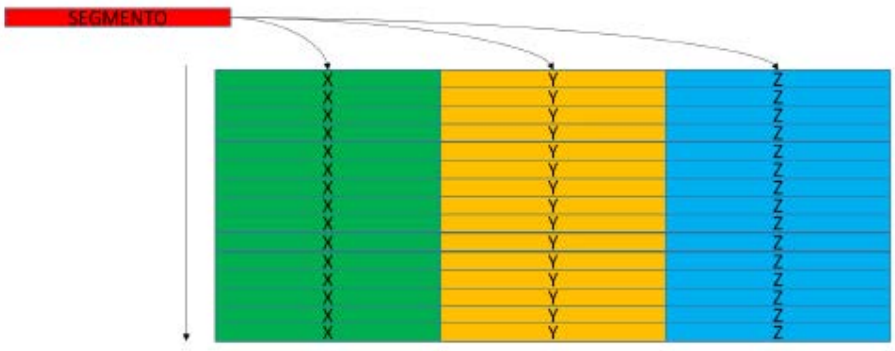

Figura 2.12: Comparación de segmentos con codebook

De este modo se tiene un conjunto de vectores de características, los cuales pueden ser usados como entrada para un algoritmo de clasificación que pueda diferenciar entre los distintos eventos. La Figura 2.13 muestra gráficamente el histograma para una sola señal contra un codebook que contiene codewords para dos posibles eventos, teniendo 50 codewords por eje.

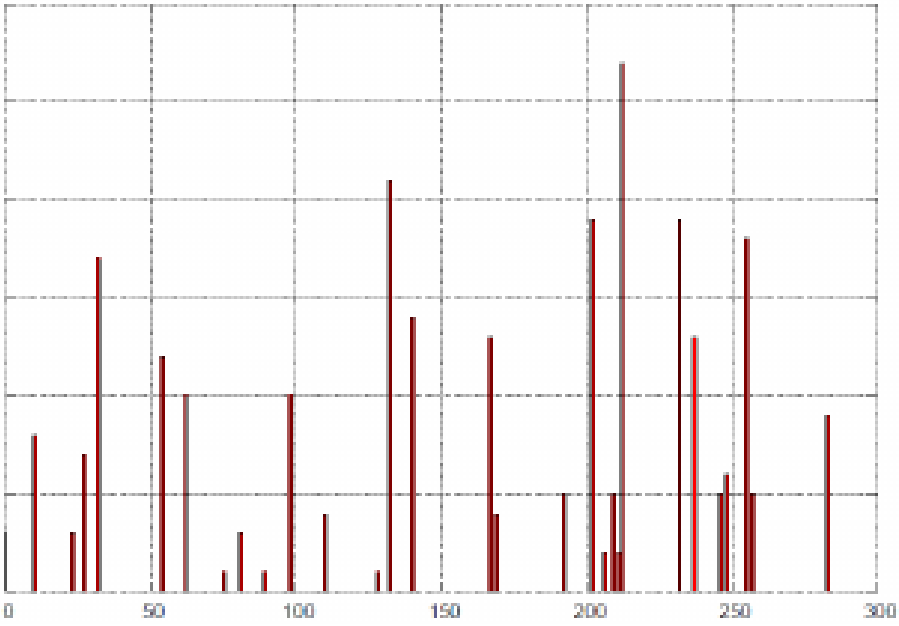

Figura 2.13: Histograma resultante

2.6. Programación Genética

La programación genética es una técnica de computación evolutiva, la cual resuelve automáticamente problemas sin la necesidad de que el usuario conozca o especifique la estructura de la solución previamente. Es un método sistemático e independiente de dominio para hacer que las computadores resuelvan problemas automáticamente comenzando desde la declaración de alto nivel, acerca de lo que es necesario hacerse [17] En un principio se cuenta con una población de programas de computadora, dicha población se evolu- cionará. Generación tras generación, la programación genética transforma las poblaciones de los programas a manera de obtener nuevas y mejores poblaciones. La programación genética es un proceso aleatorio, de tal manera que nunca puede garantizar resultados, sin embargo, esta aleatoriedad es lo que le permite escaparse de ciertas trampas en las que otros métodos deterministas pueden verse atrapados.

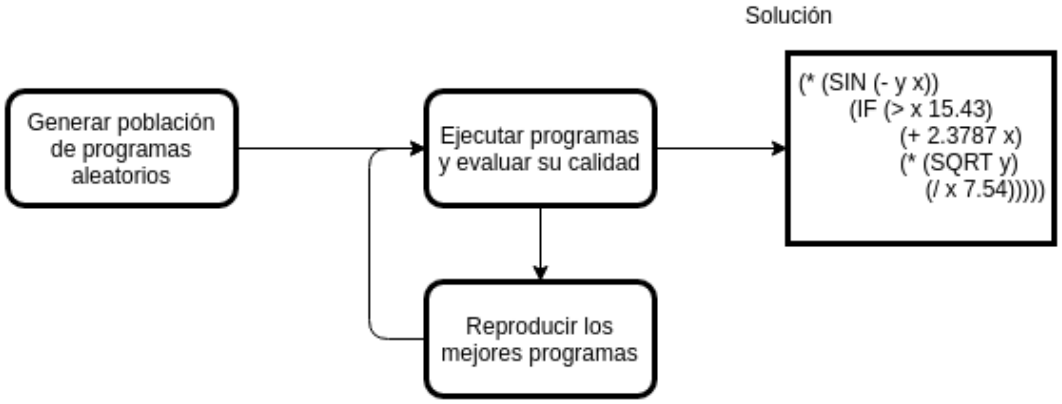

Figura 2.14: Flujo de programación genética

El proceso básico de un sistema de programación genética se puede ilustrar de manera simple con el siguiente algoritmo.

1. Crear una población inicial de programas
2. Repetir
3. Ejecutar cada programa y obtener su fitness
4. Seleccionar uno o dos programas de la población con una probabilidad basada en el fitness para participar en operaciones genéticas
5. Crear un nuevo programa individual aplicando operaciones genéticas con probabilidades especificadas

6. **hasta** que una solución aceptable sea encontrada o se llegue a alguna otra condición de parada (e.g., un número máximo de generaciones es alcanzado)

7. **retornar** el mejor individuo hasta ese momento

En un principio busca que tan bien un programa funciona al correrlo, para luego comparar su comportamiento con alguno ideal. Dicha comparación es cuantificada a manera de dar un valor numérico el cual es llamado fitness. Los programas que se desempeñen de mejor manera, son los que se reproducen y crean nuevos programas para la nueva generación. Las operaciones genéticas primarias que se usan para la creación de nuevos programas utilizando los existentes son:

- **Cruza**: La creación de un programa hijo mediante la combinación de partes aleatoriamente elegidas de dos programas padre.

- **Mutación**: La creación de un programa hijo mediante la alteración aleatoria de una parte elegida aleatoriamente de un programa padre.

2.6.1. Representación

Dentro de la programación genética, los programas son comúnmente expresados como árboles de sintaxis en vez de líneas de código. Un ejemplo de esto es el programa *max*(*x+x,x+3*y*). Las variables y las constantes en el programa (*x*, *y* y *3*) son hojas de un árbol. Dichos elementos son denominados terminales, mientras que las operaciones aritméticas (+, * y *max*) son nodos internos llamados funciones. Los conjuntos de funciones y terminales permitidas, juntos forman el conjunto primitivo de un sistema de programación genética.

En formas más avanzadas, los programas pueden componerse de múltiple componentes, como subrutinas. En este caso, la representación es un conjunto de árboles, cada uno siendo un componente, agrupados bajo un nodo raíz especial que actúa como pegamento. Estos árboles son llamados ramas.

La manera en que comúnmente se representan las expresiones dentro de la programación genética es en notación prefija, similar a la usada en Lisp o Scheme. De tal forma que max(x+x,x+3*y) se convertiría en (*max* (+ *x x*) (+ *x* (* *3 y*))). Esta notación otorga una mayor facilidad al momento de ver la relación entre expresiones y sus correspondientes árboles.

2.6.2. Inicialización de la Población

Al igual que en otros algoritmos evolucionarios, los individuos de la población inicial son comúnmente generados de manera aleatoria. Dos de los métodos más simples son el completo y el de crecimiento, y su

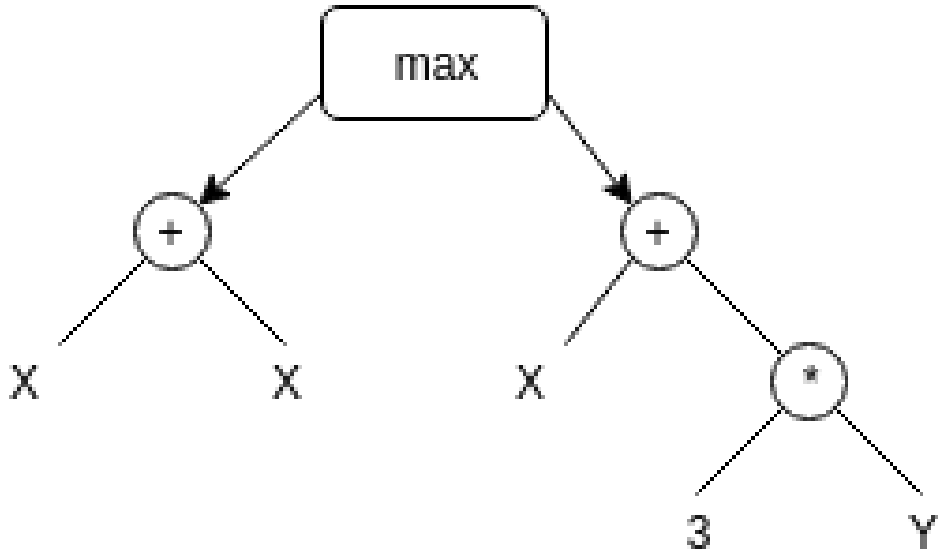

Figura 2.15: Árbol de programación genética

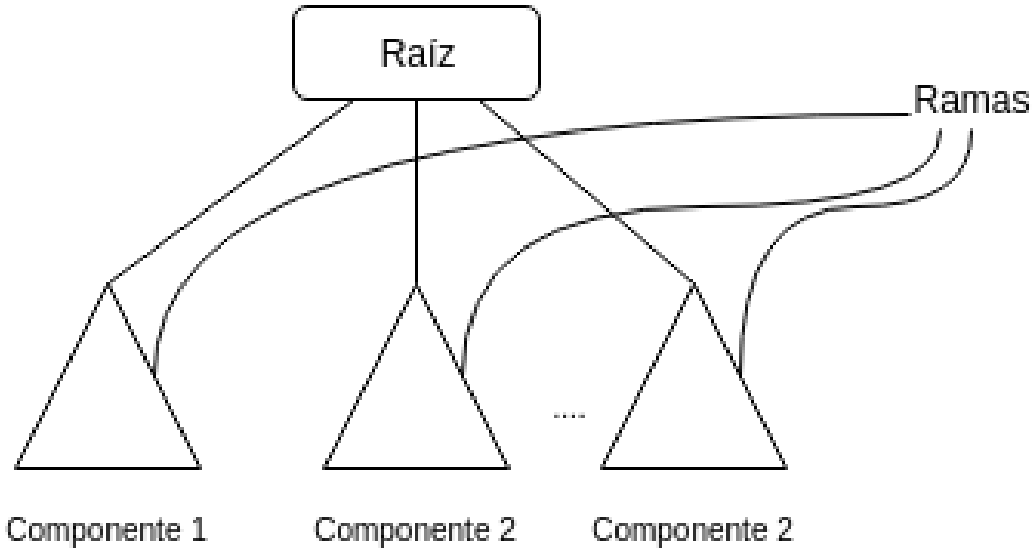

Figura 2.16: Árbol con ramas

combinación.

En ambos métodos, los individuos iniciales son generados a manera de que no excedan una profundidad máxima establecida con anterioridad. La profundidad de un nodo es el número de bordes que es necesario atravesar para llegar a él desde el nodo raíz. La profundidad de un árbol es la profundidad de su hoja más profunda.

En el método completo, los nodos son tomados de manera aleatoria del conjunto de funciones hasta que

la profundidad máxima del árbol es alcanzada. En el método de crecimiento, los nodos son seleccionados del conjunto primitivo completo (funciones y terminales) hasta que el límite de profundidad es alcanzado. Una vez que esto sucede, solamente terminales pueden ser elegidas.

2.6.3. Selección

Como en la mayoría de los algoritmos evolutivos, los operadores genéticos se aplican a individuos elegidos de manera probabilística basados en el fitness. De tal manera que mejores individuos son los que tendrán una mayor probabilidad de tener programas hijo.

El método más común es el de selección por torneo. En este método un número de individuos es elegido de manera aleatoria de la población, estos se comparan entre sí, para luego elegir al mejor para ser el padre. Cuando se está aplicando cruza, se necesitan dos padres, por ende, se realizan dos torneos de selección.

2.6.4. Recombinación y Mutación

La implementación de los operadores de cruza y mutación en la programación genética difiere en gran medida con otros algoritmos evolutivos. La forma más común de cruza es la de sub-árbol. Teniendo dos padres, la cruza de sub-árbol aleatoriamente selecciona un punto de cruza en cada árbol padre. Entonces, crea su descendiente reemplazando el sub-arbol arraigado en el punto de cruza en una copia del primer padre con una copia del sub-árbol arraigado en el punto de cruza del segundo padre (ver Figura 2.17).

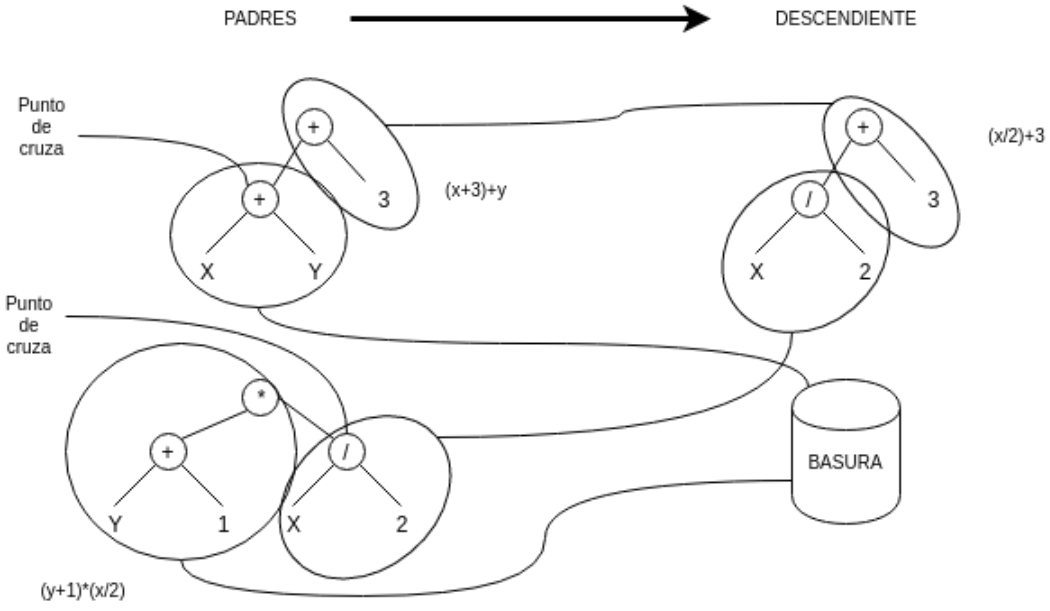

Figura 2.17: Operación de cruza

La forma más usada de mutación en programación genética, llamada mutación de sub-árbol, selecciona aleatoriamente un punto de mutación en un árbol y substituye el sub-árbol arraigado ahí con un sub-árbol generado aleatoriamente, véase 2.18

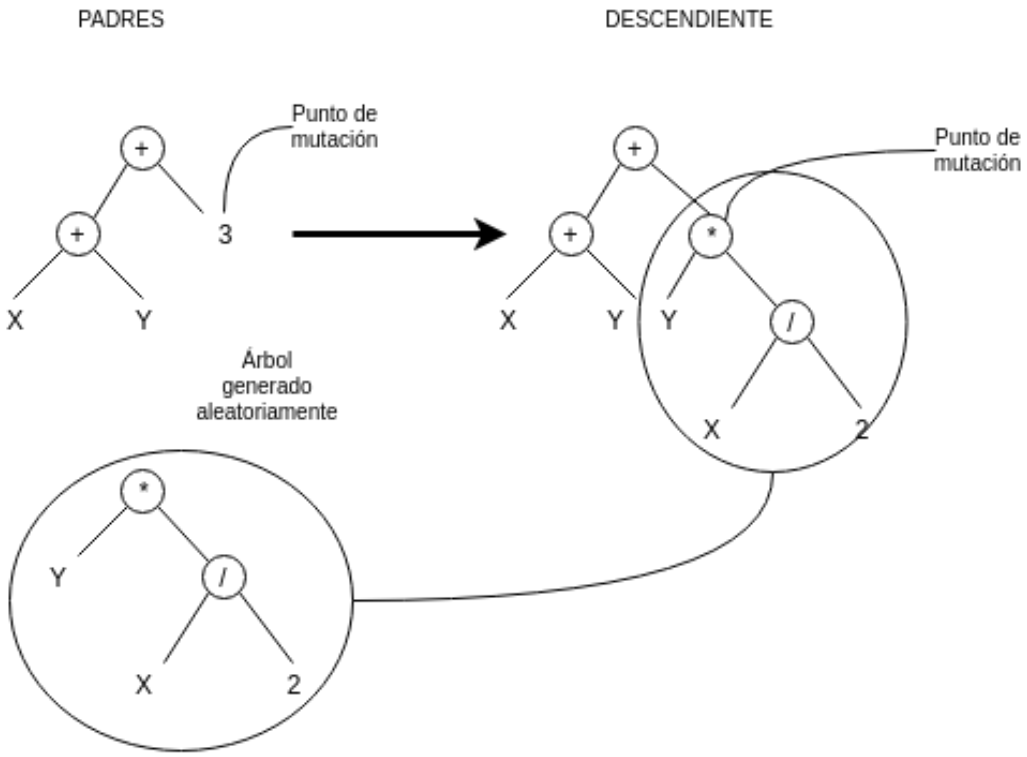

Figura 2.18: Operación de mutación

La elección de cuál de los operadores debe ser usado para crear el descendiente es probabilística. Los operadores en programación genética son normalmente mutuamente exclusivos. Sus probabilidades de aplicación son llamadas radios de operador. Comúnmente, la cruza es la que se aplica con una mayor probabilidad, donde el radio de cruza muy seguido es 90 % o mayor. Contrariamente, el radio de mutación es más pequeño, comúnmente situándose cerca del 1 %.

2.6.5. Ejecución de Programación Genética

Para aplicar un sistema de programación genética a un problema, varias decisiones se deben de hacer, a las cuales se les da la terminología de pasos preparatorios. Los puntos clave son:

- ¿Qué es un conjunto terminal?
- ¿Qué es un conjunto de funciones?

- ¿Qué es la medida del fitness?
- ¿Qué parámetros se utilizaran para controlar la ejecucion?
- ¿Cuál sera el criterio de terminación, y que será designado como el resultado de la ejecución?

2.6.6. Conjunto de terminales

Los primeros dos pasos preparatorios, la definición de los conjuntos de terminales, y de funciones, especifican un lenguaje. Esto es, juntos definen los ingredientes que se tendrán a disposición para que la programación genética pueda crear los programas.

El conjunto de terminales se conforma de entradas externas al programa, funciones sin argumentos, y constantes. Las entradas externas del programa usualmente toman la forma de variables con nombre (*x*, *y*). Aís como de funciones sin argumentos, que se incluyen porque retornan un valor diferente cada vez que son usadas, como la función rand() la cual retorna números aleatorios. Otra razón es debido a que la función produce efectos secundarios. Este tipo de funciones hacen más que solo retornar un valor, ya que pueden cambiar algunas estructuras de datos globales, imprimir o dibujar algo en la pantalla, entre otras cosas. Constantes, estas pueden ser pre-especificadas, aleatoriamente generadas como parte del proceso de creación del árbol, o creadas mediante mutación (ver Tabla 2.2).

2.6.7. Conjunto de Funciones

El conjunto de funciones utilizado en programación genética se define comúnmente por la naturaleza del problema que se está tratando. Si se estuviese tratando de un problema numérico simple, el conjunto de funciones se conformaría solamente de funciones aritméticas (+, -, *, /). La Tabla 2.1 muestra una muestra de las funciones más comunes dentro de la programación genética.

Tabla 2.1: Conjunto de Funciones

Tipo de Primitivo	**Ejemplo**
Aritmético	+, * /
Matemático	sin, cos, exp
Booleano	AND, OR, NOT
Condicional	IF-THEN-ELSE
Ciclico	FOR,REPEAT

Tabla 2.2: Conjunto de Terminales

Tipo de Primitivo	Ejemplo
Variable	x, y
Valor constante	3, 0.45
Funciones de 0 aridad	rand, go_left

2.6.8. Función Fitness

La medida del fitness es el mecanismo que tiene la tarea de indicar que elementos y regiones del espacio de búsqueda son buenos. El fitness puede ser medido en muchas maneras: la cantidad de error entre la salida y la salida esperada, la cantidad de tiempo requerida para llevar al sistema al estado deseado, la precisión del programa para reconocer patrones o clasificar objetos, entre otros.

A diferencia de otros algoritmos evolutivos, la funciones de fitness utilizadas en programación genética requieren ejecutar todos los programas en la población, comúnmente múltiple veces.

2.6.9. Parámetros de Programación Genética

Este paso especifica los parámetros de control para la ejecución. El parámetro más importante es el tamaño de población. Otros parámetros incluyen las probabilidades para aplicar las operaciones genéticas, el tamaño máximo de los programas y otros detalles de la ejecución.

En si no hay una recomendación general para los valores óptimos de los parámetros, ya que estos dependen mucho de los detalles de la aplicación. Sin embargo, la programación genética es robusta, y es probable que diferentes valores en los parámetros funcionen, de tal manera que no se requiere mucho tiempo buscando y definiendo los valores adecuados para que funcione correctamente.

2.6.10. Terminación y Designación de Solución

El quinto paso preparatorio consiste en especificar el criterio de terminación y el método para designar el resultado de la ejecución. El criterio de terminación puede incluir un número máximo de generaciones para la ejecución así como un predicado de éxito específico del problema. Comúnmente, el mejor individuo hasta ese momento es el que se designa como el resultado de la ejecución, aunque se pueden retornar individuos adicionales e información tan necesaria y apropiada para el dominio del problema.

3

Recolección y Preprocesamiento de los Datos

En el presente capítulo se explican los pasos que se llevaron a cabo durante el proceso de recolección de las muestras mediante acelerómetro, y del posterior formato y preprocesamiento de estos datos. Se explica la fase de recolección de datos, mencionando los aspectos de hardware y software utilizados durante esta. De igual manera se describen las características de los datos capturados, así como los requerimientos de formato para su preprocesamiento. En la fase de preprocesamiento se expone la metodología utilizada, la estructura interna de ésta y como funciona. Además, se explica la metodología que se empleó para asignar los valores iniciales de los parámetros utilizados en la fase de preprocesamiento y con qué valores, fueron los adecuados al momento de aplicarse. La Figura 3.1 muestra las tareas que se llevaron a cabo durante esta etapa.

3.1. Recolección de datos

El conjunto de datos utilizado durante la fase experimental se compone de 357 maniobras de conducción temeraria, divididos en 7 tipos de eventos, donde cada tipo de evento define una clase. Para el proceso de recolección de estos se utilizaron 5 teléfonos inteligentes marca Motorola de modelo Moto G. La recolección se realizó haciendo uso del acelerómetro incluido en cada dispositivo, el cual es un acelerómetro *ST Micro LIS3DH* de tres ejes, asimismo, cada dispositivo cuenta con un procesador a 1.2GHz y sistema operativo *Android* en su versión 4.4.4.

La recolección se enfocó en los eventos riesgosos más comunes al conducir un automóvil, los cuales son el arrancón, parada súbita, frenon, cambio de carril agresivo y volantazo. Para estos dos últimos se tomó

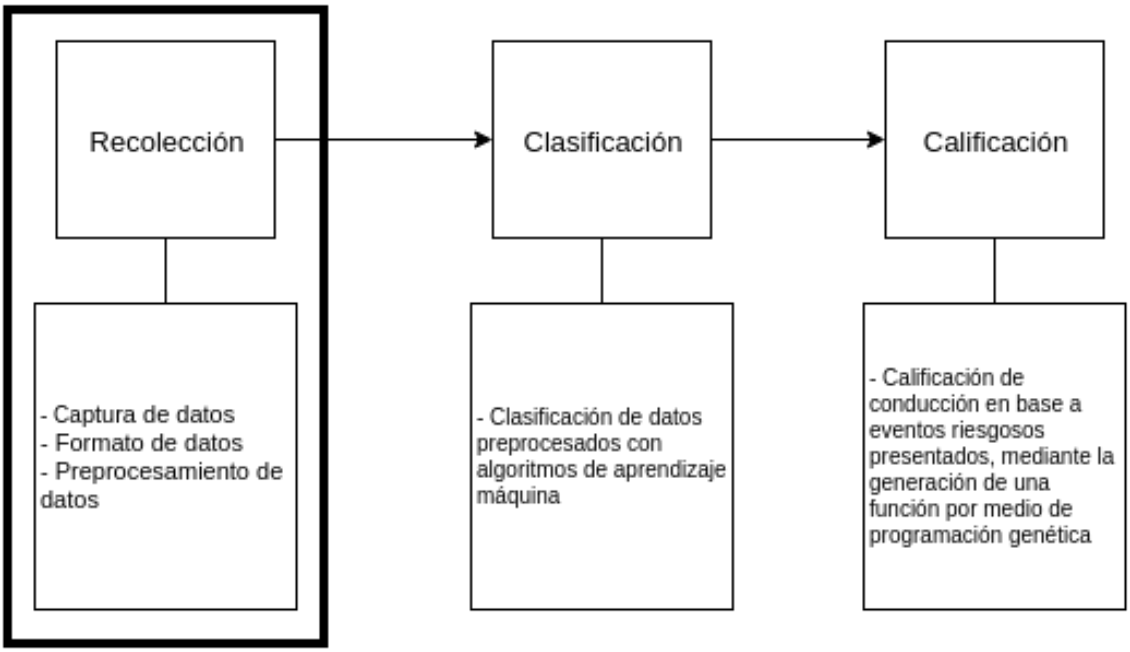

Figura 3.1: Etapa de recolección

en cuenta la dirección en la que se realiza la acción, ya sea a la izquierda o derecha. El número de eventos capturado para cada clase se presenta en la Tabla 3.1.

Tabla 3.1: Distribución de Eventos

Tipo de Evento (Clase)	**Número de Eventos**
Arrancón	43
Cambio Agresivo de Carril Izquierdo	91
Cambio Agresivo de Carril Derecho	87
Frenón	39
Parada Súbita	43
Volantazo Izquierdo	25
Volantazo Derecho	29

Algunas de las clases se comportan de manera similar, tal es el caso del cambio de carril y el volantazo, sin embargo si presentan diferencias notables para ser tomadas en cuenta de manera separada, su diferencia recae en que un volantazo presenta un cambio momentáneo de dirección de manera brusca y regresa a la

posición en la que estaba, ejemplo de esto es el esquivar un bache o un obstáculo en el camino, mientras que el cambio de carril, se genera el cambio en la dirección y se mantiene en la nueva posición, sin regresar a la anterior. La Tabla 3.2 presenta el comportamiento de cada uno de los distintos eventos.

Tabla 3.2: Descripción de Eventos

Tipo de Evento	**Descripción**
Arrancón	El automóvil presenta un aumento repentino en la velocidad desde reposo
Cambio Agresivo de Carril	Movimiento lateral de manera repentina mientras se encuentra en movimiento para luego continuar el curso en la nueva posición, comúnmente realizado en un espacio reducido y con alta velocidad
Frenón	Se presenta una desaceleración del automóvil de manera repentina por un momento para luego retomar la aceleración
Parada Súbita	El automóvil se detiene completamente de manera repentina
Volantazo	Movimiento lateral brusco mientras se encuentra en movimiento, para luego regresar a la posición en la que se encontraba y continuar el curso

Para detectar estos patrones de conducción, se posicionaron los smartphones en cinco posiciones diferentes dentro del vehículo, siendo estas las más comunes donde un conductor dejaría su dispositivo. Dichas posiciones fueron: el compartimiento de la puerta del conductor, el bolsillo de la camisa del conductor, el bolsillo frontal del pantalón del conductor, la consola central del automóvil, y dentro de una bolsa de mano/mochila posicionada en el asiento del copiloto (ver Figura 3.2).

Los dispositivos tienen los recursos de hardware para captar los patrones, sin embargo para obtener los datos de relevancia para el experimento fue necesario el diseño de una estrategia de captura, en conjunto con la utilización de una aplicación desarrollada anteriormente, la cual tiene la funcionalidad de extraer dichos datos y guardarlos en un archivo de texto (ver Figura 3.3). El sistema de captura consiste en la sincronización de dispositivos mediante una red local, en este caso 5, donde cada uno se coloca en posiciones del vehículo de uso común, así mismo, un dispositivo adicional realiza la tarea del control para el proceso de captura de los dispositivos así como el etiquetado de eventos para su respectiva clase. De los 5 dispositivos de captura, uno se utiliza como hotspot, donde mediante el uso de red inalámbrica Wi-Fi, todos los dispositivos restantes se conectan a él, incluyendo el de control. La conexión requiere especificar la velocidad de muestreo para

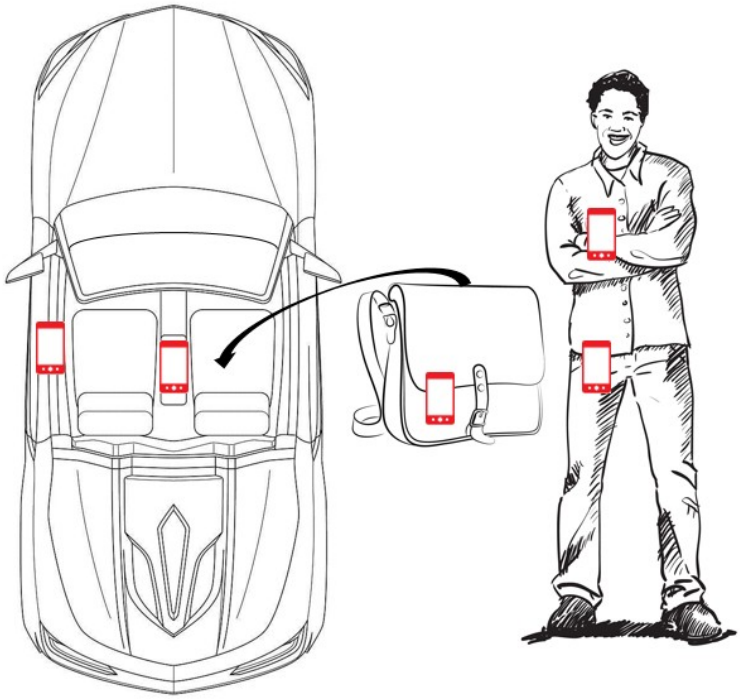

Figura 3.2: Posiciones donde fueron colocados los smartphones

la captura de los datos en la aplicación, donde los valores disponibles son 15, 35, 50, 100 y 200 Hz, de tal manera que la velocidad seleccionada, es el número de puntos por segundo que se capturan. Seleccionada la correcta velocidad de muestreo seleccionada, desde el dispositivo de control se elige la etiqueta de la clase que llevará el evento a capturar, para luego comenzar la captura de manera sincronizada entre todos los dispositivos, y al haber realizado el evento, esta se detiene. El archivo de texto generado por la aplicación contiene todos los puntos capturados hasta que se detiene la captura.

La recolección de los datos se organizó mediante rutas, donde cada una tendría un equipo designado, el cual constaría del piloto, copiloto y un pasajero asistente. Dichas rutas fueron planeadas con anterioridad, indicando que y cuantos eventos se realizarían durante ésta. Aunado al hecho de posicionar en distintas partes los smartphones y generar rutas diferentes, se utilizaron tres vehículos durante la recolección de los datos, la Tabla 3.3 muestra el modelo y año de estos.

Antes de iniciar el registro de los datos en cada ruta es necesario tener posicionados y sincronizados los dispositivos, así como el camino a recorrer, especificando el número y tipo de eventos a capturar. Cada uno de los elementos del equipo lleva a cabo una función específica. El piloto es el encargado de conducir y realizar las maniobras de conducción temeraria, el copiloto se encarga de que los dispositivos siempre se

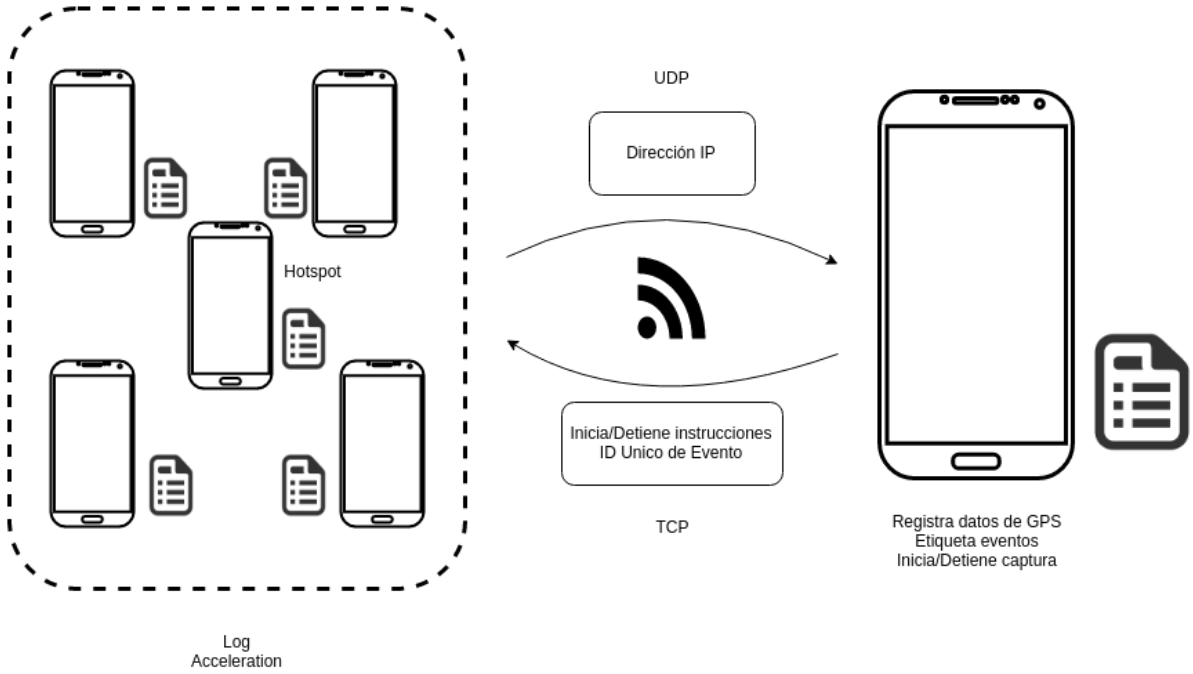

Figura 3.3: Plataforma de captura de datos

Tabla 3.3: Automóviles usados para captura de datos

Marca	Modelo	Año de Modelo	Tipo de Auto
Toyota	Camry	1999	Sedan
Nissan	Altima	2003	Sedan
Nissan	Sentra	2006	Sedan

mantengan en su posición y sin desconectarse, por último, el pasajero asistente, es quien lleva el control de la captura de los datos mediante el dispositivo de control, así como indicando que maniobra se debería de llevar a cabo. El siguiente paso consiste en establecer un hotspot para controlar todos los dispositivos de manera sincronizada, dicha sincronización se realizó conectando mediante wi-fi todos los dispositivos al que funge como hotspot, cuidando que la velocidad de captura fuera de 50Hz por segundo para cada uno, esto es, 50 puntos cada segundo.

Una vez establecida la conexión, se inicia la ruta, es decir, simular los eventos riesgosos. En cada uno de los eventos, se comienza la captura de los datos alrededor de un segundo antes de que ocurra el evento, asimismo, la captura se finaliza un segundo después de haber terminado la simulación del evento. Cada uno de los eventos capturados se almacena en un archivo de texto individual dentro de la memoria de cada dis-

positivo, de tal manera que por cada evento se generan cinco archivos de texto, uno por cada dispositivo de captura. La aplicación usada para la captura permite etiquetar el evento a capturar, de tal manera que cada uno de los eventos genera un archivo de texto, cuyo nombre consiste en el tipo de evento (clase) seleccionado a etiquetar, junto con la fecha y hora de captura (aceleron-2015-08-13-08-33-01-064.txt).

Teniendo cada uno de los eventos de cada una de las rutas, se procedió a extraerlos de los dispositivos y organizarlos por tipo de evento, independientemente de la posición en la que se encontraba el dispositivo de captura. Todo esto con el fin de aplicarles un formato necesario para trabajar con ellos.

3.2. Formateo de datos

Los distintos sensores de un smartphone capturan una gran variedad de datos, tales como aceleración, latitud, rotación, entre otros. Los datos que se consideraron para esta investigación desde un inicio, son los que en la literatura han sido usados para atacar este problema. Estos datos son los valores capturados de cada eje del acelerómetro, en si los valores de los ejes *x*, *y* y *z*.

El acelerómetro captura distintos datos, de los cuales solo se tomaron en cuenta los tres ejes de la aceleración con rotación. Los cuales manejan la rotación en cada una de las tres dimensiones que se perciben, de tal manera que se simule el movimiento que presenta un automóvil en la realidad. Dependiendo del posicionamiento del dispositivo, la detección de los ejes varia, es decir, el acelerómetro puede detectar en un eje movimientos que no pertenecen a éste, por ejemplo si el dispositivo se inclina lateralmente, detecta movimientos verticales en el eje *X* del acelerómetro, cuando debe de ser en el eje *Y*, sin embargo, de encontrarse en una posición fija, cada eje del acelerómetro capturaría correctamente los movimientos como se muestra en la Figura 3.4.

Los archivos generados por la aplicación de captura necesitan de un formato específico, ya que al momento de ser creados, se almacenan datos que no fueron contemplados utilizar, de tal manera que se desarrolló una aplicación que definiera nuevos archivos con el correcto formato a partir de los generados por la de captura, dicho formato consiste en extraer sólamente los valores de los ejes X, Y y Z del acelerómetro. Con el propósito de hacer un análisis visual del comportamiento de los datos para cada tipo de evento, la aplicación también grafica cada uno de estos nuevos archivos generados. En la Figura 3.5 se presenta el comportamiento de los distintos tipos de eventos (clases) de manera gráfica.

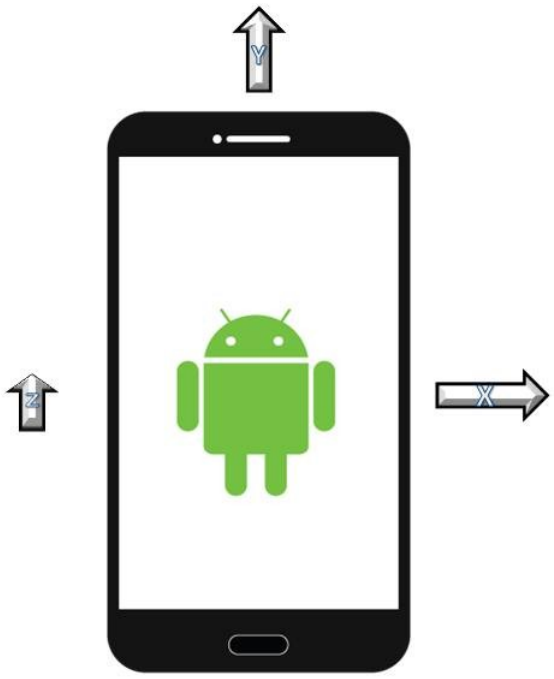

Figura 3.4: Ejes del acelerómetro de un smartphone

Como se puede observar entonces en la Figura 3.5, en algunos casos los ejes que detectan la perturbación de importancia para un evento, no coinciden con otros eventos, esto debido al distinto posicionamiento en el que se encontraban los dispositivos al momento de la captura de los datos. También se puede observar que independientemente de la diferencia en los ejes donde se presenta la perturbación, cada uno de los eventos tiene un patrón bien definido, de tal manera que la caracterización mediante alguna técnica de preprocesamiento si se presenta factible. De tal manera que, contando con los datos en un formato óptimo, donde solo se incluyen los datos de importancia para la investigación, el siguiente paso consistió en aplicar un preprocesamiento sobre ellos para obtener un vector de características, con el fin de usarse posteriormente en un algoritmo de clasificación.

3.3. Preprocesamiento de datos

Para la aplicación de algoritmos de clasificación es necesario contar con características que representen un evento de interés, donde se asume que eventos iguales contienen características similares y por lo tanto son distinguibles de otras clases.

Es aquí donde es necesario realizar un preprocesamiento de los datos anteriormente obtenidos y forma-

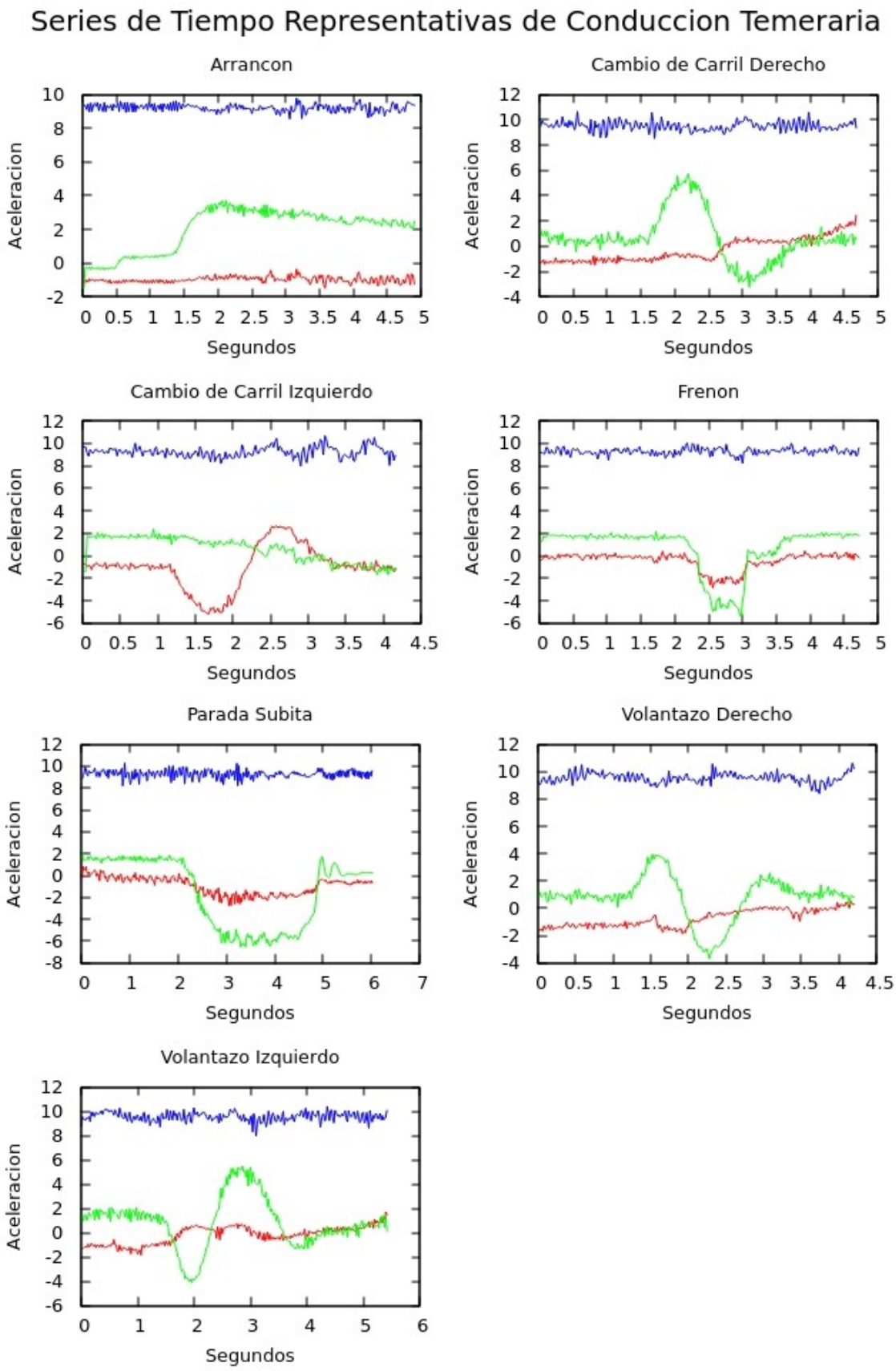

Figura 3.5: Representación gráfica de cada tipo de evento (clase)

teados, esto con el fin de encontrar ese vector de características que permita generalizar eventos similares, para luego aplicarles un algoritmo de clasificación y este pueda desempeñar su función de manera óptima.

El preprocesamiento de los datos para la obtención de una óptima caracterización consistió en aplicar la metodología *bag of words* sobre los datos formateados. Donde los pasos a realizar son los siguientes:

1. Segmentación (3 ejes): Segmentar cada una de la señales contenidas en el conjunto de entrenamiento, esto por cada clase

2. Creación de codewords (3 ejes): Una vez segmentadas las señales por clase, a cada conjunto de segmentos de cada una se le aplica el método de agrupamiento conocido como k-medias para generar codewords

3. Creación de codebook: Al tener los codewords generados, se procede a unir cada uno de ellos en una estructura que se define como codebook

4. Generación de histograma: Con el codebook generado, se procede a realizar una comparación con nuevos segmentos y así crear el histograma que fungirá como conjunto de entrada para un algoritmo de clasificación

Las tareas especificadas con la nota de 3 ejes, indica que dicha tarea se realizará para cada uno de los 3 ejes que manejan las señales capturadas (x, y y z).

Se diseñó una aplicación que realiza el proceso de muestreo automáticamente para cada una de las clases con las cuales se trabaja, donde los 357 registros disponibles se encuentran almacenados individualmente para las 7 clases que se pretenden clasificar y se procesan en la metodología con 50 % de los datos, con los que se genera un conjunto para entrenamiento (ver 3.6), de tal manera que si un tipo de evento contiene 100 datos, se utilizan 50 para el preprocesamiento.

Teniendo el conjunto de entrenamiento definido, el primer paso dentro de la metodología de *bag of words*, es la segmentación de cada uno de los datos contenidos en éste. Dicha segmentación se realiza seleccionando una ventana de puntos de la señal, la cual se corta para entonces generar un segmento. La cantidad de puntos seleccionados en la ventana, es un número especificado con anterioridad en los parámetros de la metodología. Así mismo, esta ventana es deslizante, esto significa que va recorriendo la señal desde el inicio hasta el final mientras los puntos de esta se lo permitan. El deslice puede o no tener traslape de puntos entre ventanas, de existir traslape, este es del tamaño de la ventana menos 1.

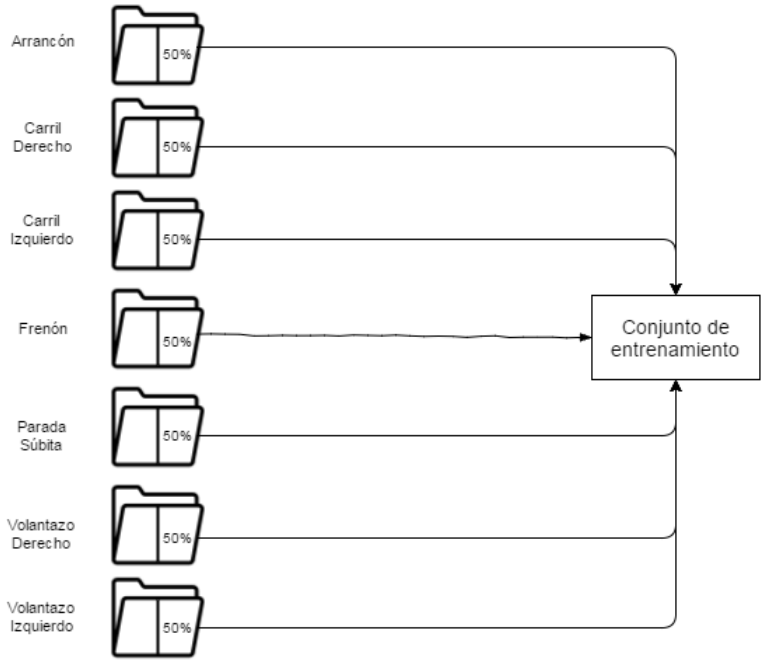

Figura 3.6: Composición de conjunto de entrenamiento

Cada uno de estos segmentos se almacena con todos los segmentos de su misma clase. Una vez realizada la segmentación de todas las señales de todas las clases, el siguiente paso es entonces generar los codewords para cada una de clases, el número de codewords a crear, al igual que el tamaño de ventana, se especifica en los parámetros *bag of words*.

El siguiente paso es la creación de los distintos codewords para cada clase. Para ello dentro de la metodología Bag of Words, se incluye la aplicación del algoritmo K-means. Este algoritmo, en su funcionamiento, crea centroides aleatorios en el mismo espacio que los puntos de los datos. Estos centroides buscan agrupar a todos los datos que se manejan en ese espacio, esto lo hace reajustándose durante cada iteración, donde una iteración conlleva a medir la distancia euclidiana de cada uno de los puntos con los distintos centroides, de tal manera que se asignara al grupo del centroide más cercano. Es entonces que el centroide se reajusta al centro del nuevo grupo definido, este ajuste es realizado debido a que en dicha iteración algunos puntos pueden ya no pertenecer a su grupo, o nuevos puntos se asignaron a su grupo, de tal manera que el centro de este ya no es el mismo. Esto se realiza hasta que los ajustes del centroide sean menores a cierto valor o en su defecto, ya no se mueva de lugar al querer ajustarse.

De esta manera es como se crean los codewords, que en el contexto del algoritmo, son los centroides. Mientras que los puntos son todos los segmentos obtenidos de las señales. Una vez que el algoritmo es aplicado sobre cada clase, los centroides resultantes son los codewords, los cuales son una representación

generalizada de todos los segmentos que fueron asignados a él.

Con codewords generados para cada clase en sus tres ejes se procede a estructurar el codebook, el cual contiene las señales segmentadas de referencia para todas las clases de interés donde se hace un solo conjunto de las señales en los 3 ejes, es decir, cada señal de prueba es comparada con las señales generadas en el codebook comparando sus segmentos de manera indistinta en relación al eje del cual provienen.

El último paso es la generación del histograma, para esto se generan nuevos segmentos, esta vez de todos los datos disponibles, no solo del conjunto de entrenamiento. Dicha segmentación se realiza con el mismo tamaño de ventana y traslape establecido para los segmentos de entrenamiento. Con los segmentos generados, cada uno de ellos es comparado, mediante distancia euclideana, con cada uno de los codewords presentes en el codebook. De las distancias obtenidas para cada segmento, se elige la menor, y se indica con cual codeword se obtuvo dicha distancia. El histograma se representa mediante un arreglo en el sistema, donde cada elemento es un numero inicializado en 0, asimismo, cada uno de estos elementos corresponde a un codeword, de manera que al realizar la comparación e indicar la distancia menor, el codeword con el cual se haya obtenido, aumentara su valor en 1, indicando que un segmento se acercó más a ese codeword. Al finalizar la comparación de todos los segmentos, se contará con un arreglo con distintos valores en cada uno de sus elementos, indicando el número de incidencias de cada codeword. Este arreglo es entonces el histograma que sirve como conjunto de entrada a algún algoritmo de clasificación.

El proceso de Bag of Words requería varios parámetros para realizarse, la Tabla 3.4 presenta los valores que se utilizaron en ellos. Es importante mencionar que se aplicaron varias combinaciones en los valores de los parámetros, esto con el fin de buscar una que generara el mejor vector de características posible para su clasificación. Para poder definir el mejor vector, cada uno de los vectores generados, habría de ser probado con un algoritmo de clasificación, en este caso una red neuronal artificial.

Tabla 3.4: Parámetros para Bag of Words

Parámetro	**Descripción**
Tamaño de ventana	Longitud de cada segmento en el paso de segmentación
Overlap	Si el paso de segmentación permitirá traslape o no (overlap es tamaño de ventana - 1)
Número de codewords	Cuantos codewords serán creados por clase para generar el histograma

Asimismo, la Tabla 3.5 presenta los valores de los parámetros que fueron probados durante el proceso de preprocesamiento utilizando la metodología de Bag of Words, donde los valores en negro indican el mejor valor para ese parámetro.

Tabla 3.5: Parámetros probados durante la aplicación de Bag of Words

Parámetro	**Valores**
Tamaño de ventana	5, 8, 10, 12, 15, **20**
Overlap	**Si**, No
Número de codewords	10, 20, 30, 40, **50**

4

Experimentos y Resultados

En este capítulo se mencionan los distintos experimentos que se llevaron a cabo una vez que la etapa de preprocesamiento de los datos concluyera. Primeramente se aborda el apartado de la clasificación de los eventos, mencionando tres algoritmos que fueron aplicados sobre los datos, los cuales fueron el *Dynamic Time Warping*, en el que se utilizaron los datos sin caracterización, simplemente formateados pero sin aplicárseles un preprocesamiento. Después se mencionan dos algoritmos que fueron aplicados sobre los datos ya preprocesados, Naive Bayes y Red Neuronal Artificial. Para cada uno se indican los parámetros utilizados y los mejores resultados obtenidos, así como una comparativa entre ellos. Posteriormente se explica lo relacionado a la etapa de calificación, esto es, después de haber clasificado los eventos mediante un algoritmo, como se calificaría un recorrido dados dichos eventos; para esta fase se utilizó programación genética. Se menciona el procedimiento realizado para la aplicación de programación genética, los distintos parámetros que fueron usados durante esta etapa, así como los resultados obtenidos y sus comparaciones. La Figura 4.1 muestra las tareas realizadas durante la etapa de clasificación.

4.1. Aplicación de DTW sobre Datos

Hoy en día existe una gran variedad de algoritmos y/o técnicas que permiten realizar clasificaciones, de tal manera que basarse solo en los resultados obtenidos por uno, no presentaría una buena fundamentación para dichos resultados, asimismo, no permitiría explorar otros que pudiesen ser más adecuados para la problemática que se está tratando y muy probablemente obtener mejores resultados de clasificación. Es por ello que antes de aplicar un algoritmo de clasificación sobre los datos preprocesados, se procedió a realizar una clasificación simple aplicando *Dynamic Time Warping* sobre los datos originales.

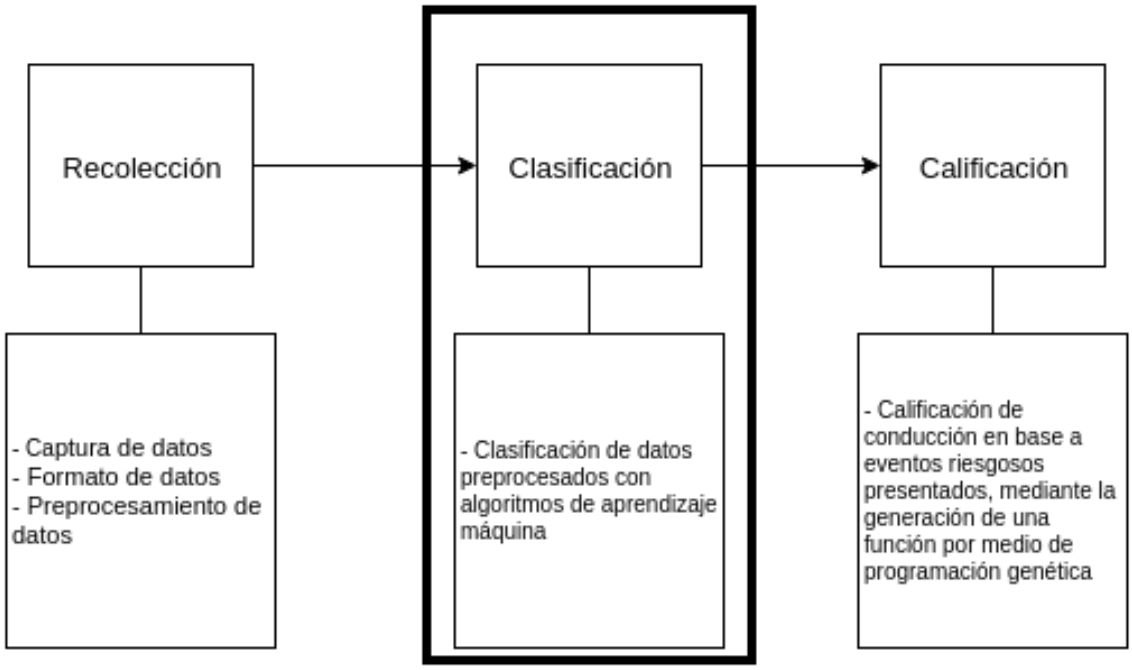

Figura 4.1: Etapa de clasificación

El procedimiento que se llevó a cabo para esta clasificación consistió dos pasos. Primeramente, de manera visual, se seleccionó la señal más representativa para cada tipo de evento, es decir, la señal que presentara de manera clara los movimientos de cada tipo de evento tomando como base una aplicación utilizada al momento de capturar los datos, la cual muestra en tiempo real como el acelerómetro detecta los movimientos. Dicha señal fungiría como señal ideal, esperando que la mayoría de los eventos que pertenecen a ese tipo, tengan un comportamiento muy similar.

El segundo paso consistió en, una vez seleccionadas las señales ideales, comparar cada una de las señales presentes en el conjunto de datos. La comparación es realizada mediante el alineamiento de la señal ideal con la señal a comparar, para luego calcular la distancia euclidiana entre ambas. Una vez que se comparó la señal con todas las ideales, se le asigna una clasificación, la cual corresponde al tipo de evento con el que se obtuvo la menor distancia. Si la clasificación asignada es la misma que la real de la señal entonces fue una clasificación positiva, de lo contrario, habría sido negativa. Los resultados de realizar esta clasificación se presentan en la Tabla 4.1

Como se puede observar, los resultados obtenidos para la clasificación mediante *DTW* mostraron un porcentaje de precisión sumamente bajo para cada uno de los eventos. Esto debido a que algunas señales no

Tabla 4.1: Porcentajes obtenidos por el DTW

Tipo de Evento	**Porcentaje de Precisión (%)**
Arrancón	57.14
Cambio Agresivo de Carril Izquierdo	8.88
Cambio Agresivo de Carril Derecho	25.58
Frenón	28.94
Parada Súbita	52.38
Volantazo Izquierdo	25
Volantazo Derecho	17.85

tienen la misma longitud y la perturbación donde el evento se presenta no es el mismo eje en todas, de tal manera que al alinearse, aunque la señal sea parecida, la distancia euclidiana obtenida indica algo totalmente distinto. Esto quiere decir que graficamente dos señales pueden ser sumamente parecidas, sin embargo, el evento se presenta en distintos ejes, de tal manera que al comparar la distancia entre ejes correspondientes, *X* con *X*, *Y* con *Y* y *Z* con *Z*, las distancias obtenidas no indican que las señales son parecidas.

4.2. Aplicación de Naive Bayes sobre Datos

Naive Bayes es un algoritmo de clasificación que pese a su simplicidad, se ha mostrado a la par en cuanto a potencia de clasificación en comparación con otros más complejos como la Máquina de Vectores de Soporte.

Se utilizaron los algoritmos del paquete scikit learn[1] de *Python*. Para el entrenamiento de la metodología se usaron 60 % de los datos para entrenamiento y 40 % para prueba, asimismo, con el fin de obtener resultados validos se crearon 30 modelos para cada conjunto de datos, donde en cada ocasión se mezclaban las instancias y al final se generaba un promedio del porcentaje de clasificación. La tabla 4.2 presenta los distintos porcentajes obtenidos por clase.

[1]http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

Tabla 4.2: Porcentajes obtenidos por Naive Bayes

Tipo de Evento	**Porcentaje de clasificación (%)**
Arrancón	93.24
Cambio Agresivo de Carril Izquierdo	78.22
Cambio Agresivo de Carril Derecho	79.95
Frenón	91.74
Parada Súbita	85.43
Volantazo Izquierdo	88.29
Volantazo Derecho	92.44

4.3. Aplicación de Red Neuronal Artificial sobre Datos Preprocesados

Dentro de los algoritmos seleccionados para generar la clasificación de los eventos se encuentra la red neuronal artificial. Dicho algoritmo ha mostrado ser efectivo al momento de identificar patrones en datos, más aun, se ha mostrado como la mejor opción al momento de extraer patrones significativos en problemas similares. La implementación utilizada fue la de Stephen Marsland[2] para el lenguaje de programación *Python*.

Con el fin de obtener el mejor modelo posible para un óptimo resultado, se probaron distintas configuraciones en los parámetros del algoritmo, específicamente en el número de neuronas presentes en la capa oculta de red neuronal, así como la función de activación utilizada en las neuronas. La Tabla 4.3 presenta los distintos parámetros probados para la selección del modelo.

Tabla 4.3: Parámetros probados durante la selección del modelo de Red Neuronal Artificial

Parámetro	**Valores**
Neuronas en capa oculta	5, 10, 15, 20, **25**, 30
Función de salida	Logística, **Softmax**

Al tener los datos del acelerómetro preprocesados mediante la aplicación de *Bag of Words*, se entrenó el

[2]http://stephenmonika.net/

modelo seleccionado con un 60 % de los datos, mientras que el resto (40 %) fungirían como datos de prueba. Este experimento fue realizado 10 veces, de manera que se crearon distintos dobleces de entrenamiento y prueba en cada ocasión. El porcentaje promedio de clasificación para cada clase se presenta en la tabla 4.4.

Tabla 4.4: Porcentajes obtenidos por la RNA

Tipo de Evento	**Porcentaje de clasificación (%)**
Arrancón	94.74
Cambio Agresivo de Carril Izquierdo	77.76
Cambio Agresivo de Carril Derecho	79.58
Frenón	93.05
Parada Súbita	89.63
Volantazo Izquierdo	91.68
Volantazo Derecho	93.65

Como se puede apreciar para la mayoría de los casos, los resultados muestran un gran poder de discriminación en 5 de las 7 clases a distinguir, con porcentajes de exactitud por encima del 90 %. Esto demuestra que el preprocesamiento realizado fue el correcto para obtener una buena caracterización de las señales y así poder clasificarlas, en su mayoría, de manera correcta. Sin embargo, dada la complejidad de las dos clases restantes (cambio agresivo de carril izquierdo y cambio agresivo de carril derecho) y si bien con la metodología se logran obtener resultados competitivos, esta no logra discriminar en tal grado a dichas clases.

4.4. Generación de Conjunto de Datos para Programación Genética

Dentro de los objetivos de esta investigación, estaba el diseñar e implementar una metodología que permitiese calificar un recorrido en base a los eventos temerarios presentados en él. Esto se ha tratado de abordar anteriormente, como en el caso de [14], donde se propone una metodología de calificación, la cual consistía en comenzar un recorrido con una puntuación de 100 e ir descontando de ella conforme el tipo de evento que se presentara. Dicha metodología es muy simple y podría generar calificaciones que no reflejaran realmente el nivel de riesgo que se presentó en el recorrido. Con el fin de generar una estrategia para crear un método de calificación más apegado a la realidad, se propuso el uso de programación genética. La Figura 4.2 muestra las tareas realizadas durante la etapa de calificación.

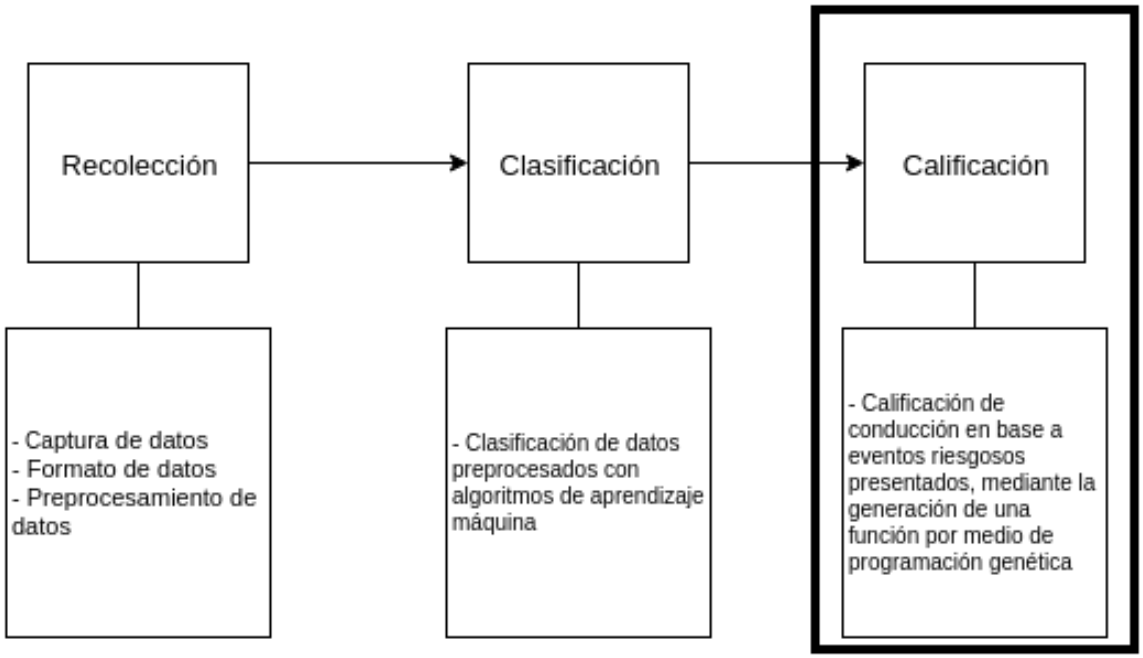

Figura 4.2: Etapa de calificación

La programación genética consiste en, mediante la simulación del comportamiento evolutivo del ser humano, encontrar o acercarse lo más posible a una solución de un problema específico. Se compone de una población inicial de individuos, los cuales son llamados programas, donde cada programa representa una solución; dicha población se genera aleatoriamente, donde el fitness promedio por lo general será muy alto. El fitness de un individuo, en este problema, se busca que sea bajo, entre más disminuya, mejores resultados se podrían esperar de él.

Al tener la población inicial, y siendo el objetivo tratar de encontrar un individuo que se acerque lo más posible a la solución del problema, entonces se evoluciona, dicha evolución se genera con operadores de mutación y cruza, donde en cada iteración, llámese generación, se definirá mediante un torneo en base al fitness de individuos seleccionados aleatoriamente, cuales se mutan y/o cruzan entre sí.

Los individuos son conjuntos de variables y operadores, en este caso aritméticos, los cuales en conjunto generan una función posfija, donde al sustituir las variables por valores reales se obtiene un valor, el cual puede ser exactamente el valor objetivo o muy cercano a él, esto dependiendo del nivel de fitness que tenga el individuo que se esté probando. El proceso concluye una vez que se encuentra la solución exacta o el número de generaciones definido anteriormente se alcanza. En este último caso se selecciona el mejor individuo como la mejor solución.

Para esto se procedió a generar un conjunto de datos, de donde se desprende una parte para entrenamiento, al cual se le aplica programación genética con el fin de obtener una función calificadora que tome todos los eventos temerarios como un todo y en base a ellos otorgar una calificación, sin asignarle un puntaje de descuento fijo a cada tipo de evento. Así mismo, del conjunto original, también se desprende un conjunto de prueba, esto para probar la función obtenida con datos no usados en su generación.

En primera instancia se crearon, en papel, 20 sesiones ficticias de conducción temeraria, donde cada una incluía un número de eventos temerarios, asimismo también se especificaba la longitud del camino recorrido y la velocidad promedio a la que se realizó. La Tabla 4.5 presenta un ejemplo de una sesión.

Tabla 4.5: Datos presentados en una sesión simulada

Dato	**Valor**
Longitud	10km
Velocidad Promedio	50km/h
Arrancón	1
Cambio Agresivo de Carril	3
Frenón	2
Parada Súbita	3
Volantazo	2
Calificación	85

Una vez que se crearon las 20 sesiones, se procedió a entregárselas a 6 personas de manera separada, a las cuales se les instruyó que asignaran una calificación a cada una en base a los eventos temerarios presentados, esto usando su propio criterio. Dichas calificaciones deberían de estar entre el rango de 0 a 100, siendo la mínima y la máxima respectivamente.

Al tener todas las calificaciones para todas las sesiones por cada uno de los participantes, de cada sesión se eliminó tanto la mínima como la máxima, para luego sacar el promedio de las restantes. Dichos promedios actuarían como un punto intermedio entre los distintos criterios de los participantes con el fin de brindar la mayor generalidad posible al momento de generar la función calificadora.

Teniendo el conjunto de 20 sesiones con sus respectivos promedios, se procedió a generar 10 conjuntos tanto de entrenamiento como de prueba, esto mediante la aplicación cross-validation. El fin fue generar 10 dobleces, cada doblez se distribuyó en 14 datos para entrenamiento y 6 datos para prueba, donde ninguno de estos dobleces fue idéntico.

Teniendo los 10 dobleces generados, el siguiente paso consistió en aplicarles programación genética a cada uno, para obtener 10 funciones, donde cada una sería probada con su respectivo conjunto de prueba, para así luego seleccionar la que mejor se desempeñara.

4.5. Obtención de Función Calificadora Usando Programación Genética

Teniendo los conjuntos de entrenamiento y prueba definidos para cada uno de los dobleces, el siguiente paso fue obtener una función por cada uno de ellos, con el fin de poder elegir la que mejor resultado arrojara al momento de probarla.

La programación genética consiste en simular la genética humana para encontrar una solución óptima a un problema específico. Al decir que se simula este proceso humano, quiere decir que utiliza sus principios para ir mejorando soluciones, las cuales se podrían considerar como los individuos de una población, dicha mejora se realiza mediante la mutación y la cruza de esos individuos para generar nuevos, los cuales conforman una nueva generación. Dicho proceso se repite un determinado número de veces o hasta que un individuo (solución) haya logrado solucionar el problema en cuestión de la mejor manera.

4.5.1. Aplicación de TinyGP

Para generar la función se utilizó una implementación ya existente de un algoritmo de programación genética, *TinyGP*, la cual fue hecha en *Python* por el Dr. Riccardo Poli[3]. Dicha implementación permite alimentar directamente el conjunto de entrenamiento mediante un archivo de texto, el cual debe contar con un formato específico, como lo muestra la Tabla 4.6 una vez teniendo esto, cuando se ejecuta esta aplicación automáticamente genera la función de salida..

La primera línea del archivo contiene datos de configuración para la aplicación, el primer valor indica el

[3]http://www.gp-field-guide.org.uk/

Tabla 4.6: Formato del conjunto de entrenamiento

Parametros	6		100		1		5	14
	2.0	1.0	6.0	2.0	0.0	4.0	66.8	
	0.0	2.0	1.0	3.0	0.0	1.0	69.0	
	11.0	1.0	7.0	9.0	4.0	2.0	39.6	
	4.0	0.0	2.0	9.0	1.0	6.0	47.8	
	10.0	6.0	9.0	8.0	6.0	7.0	32.0	
	2.0	1.0	0.0	1.0	2.0	2.0	73.0	
Recorridos	2.0	7.0	6.0	8.0	4.0	7.0	51.0	
	4.0	1.0	3.0	2.0	3.0	2.0	61.0	
	4.0	0.0	1.0	1.0	0.0	1.0	79.4	
	2.0	1.0	5.0	1.0	2.0	0.0	71.0	
	6.0	5.0	1.0	3.0	0.0	2.0	63.0	
	12.0	7.0	9.0	8.0	7.0	11.0	24.0	
	0.0	1.0	0.0	1.0	0.0	0.0	94.6	
	0.0	2.0	1.0	1.0	1.0	0.0	85.0	

número de variables que se maneja para la generación de la función, el segundo indica el número de constantes que se tienen a disposición en el conjunto primitivo, los dos siguientes valores indican el límite mínimo y máximo, respectivamente, de lo que puede valer una constante generada aleatoriamente, por último, se tiene el valor que indica el número de datos que se tienen en el conjunto. En cuanto a las siguientes líneas, cada una corresponde a un registro dentro del conjunto de datos, es decir a una sesión. Los primeros seis valores de cada registro son las variables, que en si son los distintos números de eventos presentados en cada recorrido. El último valor indica la calificación promedio, que en este caso funge como el objetivo.

La aplicación genera una función que contiene variables y operadores aritméticos, donde al asignarle valores a dichas variables con datos del conjunto de prueba, se intenta obtener el mismo resultado que el objetivo.

La aplicación contiene distintos parámetros que pueden ser modificados con el fin de obtener diferentes

resultados al momento de la generación de las funciones. La Tabla 4.7 presenta los parámetros que fueron modificados y que valores se les fueron asignados durante este proceso. Los valores resaltados indican los que ayudaron a generar mejores funciones

Tabla 4.7: Parámetros probados durante la generación de función calificadora

Parámetro	**Valores**
POP_SIZE	**100000**
DEPTH	5, 10, **15**
MAX_LEN	5, **10**
GENERATIONS	**100**, 150, 200
TSIZE	2, **3**
CROSSOVER_PROB	0.5, 0.7, **0.9**
PMUT_PER_NODE	0.05, 0.1, **0.4**, 0.5

El parámetro *POPSIZE* indica el tamaño de población inicial que maneja el algoritmo, este valor no se modificó ya que 100000 es una cantidad común que se maneja en algoritmos evolutivos. *DEPTH* indica la profundidad máxima que los programas iniciales pueden tener, mientras que *MAX_LEN* indica el máximo tamaño que un programa GP puede tener. Estos dos parámetros fueron modificados conjuntamente con la intención de generar funciones de longitud no muy extendida.

El parámetro *GENERATIONS* indica el máximo número de generaciones que correrá el algoritmo en caso de no encontrar una solución antes de eso. *TSIZE* indica el tamaño de torneo para elegir que individuo (solución) se mutará o cruzará. Por último los parametros *CROSSOVER_PROB* y *PMUT_PER_NODE* indican la probabilidad de que se presente crossover en una generación y la probabilidad de mutación cuando este operador de variación es elegido.

El algoritmo maneja 4 operadores aritméticos al momento de generar funciones, los cuales son suma (+), resta (-), multiplicación (*) y división (/); sin embargo, dado a la extensa longitud de algunas funciones generadas, fue necesario descartar el operador de división, ya que al no tener un control de la estructura de la función que se va generando ni de las variables con las que se prueba dicha función, se pueden presentar divisiones entre 0, generando un error instantáneo en el resultado arrojado por la función.

Una vez generadas todas las funciones por cada uno de los dobleces, se procedió a probarlas con su respectivo conjunto de entrenamiento. Para medir la efectividad de cada una de las funciones, se calculó el

RSE (*Relative Squared Error*), el cual indica las ventajas de usar un dado modelo sobre un enfoque más simple. Los valores esperados al calcular el *RSE* deben ser menores a 1 y acercarse lo más posible a 0. La Tabla 4.8 presenta el RSE para cada una de las funciones, asimismo, se presenta el promedio de los diez.

Tabla 4.8: RSE para cada función obtenida mediante TinyGP

Número de función	**RSE**
1	0.2746
2	0.2956
3	0.3601
4	0.4001
5	0.2144
6	0.6154
7	7.1535
8	0.1496
9	0.1263
10	0.4687
Promedio	1.0058

Los *RSE* presentados por cada función probada presentan un buen nivel, ya que no sobrepasan el valor de 1, a excepción de la número 7, la cual se disparó con un valor demasiado elevado, lo que afectó al promedio. Siendo la mejor la número 9, que presentó un *RSE* de 0.1263.

Con la finalidad de obtener una comparativa para los resultados obtenidos, se procedió a realizar el mismo proceso sobre los datos en [14], el cual cuenta con una estructura similar a las de esta investigación en las rutas de conducción realizadas. Al igual que en este trabajo, el número de rutas contenidas es de 20, la diferencia recae en que 10 son etiquetadas como agresivas, mientras que las restantes son tranquilas. En este caso, se realizaron 3 experimentos, uno correspondiente a la combinación de ambos, mientras que los otros dos, fue cada una de las etiquetas por separado. El número de dobleces fueron 10 nuevamente, con la única diferencia de que en los experimentos individuales el conjunto de entrenamiento contenía 7 datos, mientras que el de prueba 3, esto para mantener la proporción con el conjunto combinado. Las Tabla 4.9 muestra los *RSE* obtenidos junto con sus respectivos promedios.

Los resultados obtenidos para estos datos no fueron buenos, donde se mostró un error mayor fue en

Tabla 4.9: RSE para cada funcion obtenida mediante TinyGP (Castignani)

Número de función	Comb	Agresivo	Tranquilo
1	0.5245	20.7346	3.0571
2	0.8352	0.6785	0.3884
3	4.3838	1.1953	ERROR
4	1.2503	3.1990	4.9646
5	0.2532	6.2199	20.0936
6	1.2129	3.0120	36.6808
7	6.2434	2.3391	4.3008
8	66.7587	6.4008	9.3054
9	0.8417	0.7895	1.1728
10	0.9081	0.7468	0.4702
Promedio	8.3212	4.5515	8.0434

el conjunto combinado, ya que dados los datos que se tomaban en cuenta, las calificaciones otorgadas en dicha investigación variaban mucho o eran muy similares incluso cuando tenían un muy diferente número de eventos, esto debido a que también se tomaba en cuenta otros factores como la hora del día y el clima.

4.5.2. Aplicación de EvoDAG

Aunado a la utilización del *TinyGP*, se usó el algoritmo evolutivo EvoDAG, implementado por el Dr. Mario Graff[4], miembro del INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación.

EvoDAG presenta un sistema de programación genética de estado estable basado en proyecciones en el espacio fenotipo[18], el cual usa la idea de crear el mejor descendiente que puede ser producido por una combinación lineal de padres [19].

Así mismo, *EvoDAG* no evoluciona árboles, como su nombre lo indica, lo que evoluciona son grafos dirigidos acíclicos, ya que se basa en la cruza geométrica propuesta por Moraglio [20].

En el caso de la implementación del *EvoDAG*, solo se le alimentaron los datos para obtener las predicciones de las calificaciones. Una vez obtenidas dichas predicciones, se calcula el *RSE* de cada doblez y su

[4]https://github.com/mgraffg/EvoDAG

promedio. Para el *EvoDAG* se utilizaron los mismos conjuntos de datos generados para *TinyGP*, tanto en los datos de esta investigación, como en los contenidos en [14]. La tabla 4.10 presenta el *RSE* obtenido en cada doblez de los datos propios.

Tabla 4.10: RSE para cada función usando EvoDAG

Número de doblez	**RSE**
1	0.2954
2	0.6914
3	0.2802
4	1.0838
5	0.4620
6	0.6610
7	0.5109
8	0.9799
9	1.3957
10	0.6943
Promedio	0.7055

Como se puede observar, el *RSE* para cada uno disminuyó considerablemente en comparación con los obtenidos usando *TinyGP*. En el caso de *EvoDAG* dos dobleces sobrepasaron el valor de 1, a diferencia de solo uno en el experimento con *TinyGP*, sin embargo el promedio se mantuvo por debajo de este, ya que en *TinyGP*, el único doblez que sobrepasó el valor de 1 obtuvo un valor de 7.1535, lo cual aumentó en gran medida el promedio. La tabla 4.11 presenta los *RSE* obtenidos para los conjuntos de datos de [14]

Al igual que con los datos propios, se muestra una mejoría en cuanto a los *RSE* obtenidos, sin embargo los promedios para cada uno de los conjuntos se mantuvo por arriba del valor de 1. La principal razón de la inestabilidad presentada en estos conjuntos, es la subjetividad de las calificaciones otorgadas debido a tomar en cuenta no solamente los eventos presentados durante las rutas, sino también las condiciones del tiempo y el hecho de ser día o noche.

Tabla 4.11: RSE para cada función usando EvoDAG (Castignani)

Número de función	Comb	Agresivo	Tranquilo
1	0.5753	1.0671	3.0
2	0.4840	0.8691	3.0
3	0.4068	3.0100	ERROR
4	0.3251	2.8506	0.5221
5	2.6832	1.4797	0.9507
6	6.0669	1.2814	0.4324
7	0.4384	1.9677	3.0
8	0.4768	2.5540	0.6970
9	0.6401	1.7546	1.0
10	0.4387	1.7106	2.2359
Promedio	1.2535	1.8545	1.6480

4.6. Análisis de Resultados

A continuación se detallan las conclusiones obtenidas en base a los resultados de los experimentos realizados.

4.6.1. Clasificación de Eventos

Los experimentos de clasificación de los datos mostraron resultados interesantes. En primera instancia al generar una clasificación utilizando *DTW* sobre las señales sin preprocesar arrojó porcentajes de clasificación muy bajos, lo cual indica que el realizar un preprocesamiento es necesario, esto debido a que las señales son muy variadas en forma y longitud, haciendo que el *DTW* no pueda calcular correctamente la similitud entre ellas, generando así clasificaciones erróneas.

Habiendo hecho la clasificación sin preprocesamiento, se hizo evidente la necesidad de este. Una vez realizado, se procedió a aplicar una clasificación mediante el algoritmo Naive Bayes, el cual pese a que trata a todos los datos de manera independiente, muestra resultados que compiten con otros algoritmos de mayor complejidad. Dicha clasificación arrojó un nivel discriminatorio muy alto, donde el porcentaje más alto se situó en 93.34 %, mientras que el más bajo en 78.22 %, el cual sigue teniendo un alto nivel de discriminación. El último experimento en el apartado de clasificación fue el de la utilización de una red neuronal artifi-

cial, dicho algoritmo se muestra actualmente como uno de los más usados en cuestiones de clasificación, y específicamente en investigaciones relacionadas con la conducción temeraria, esto debido a su potencia de clasificación y a no presentar una gran complejidad para implementarlo.

Los porcentajes obtenidos, al igual que en Naive Bayes presentaron altos niveles de discriminación, donde el porcentaje más alto se posicionó en 94.74 %, mientras que el más bajo en 77.76 %.

En ambos algoritmos 5 de 7 clases presentaron un porcentaje mayor 85 %, lo cual demuestra que el preprocesamiento aplicado a los datos en un principio fue el correcto, así como la elección de los algoritmos de clasificación para atacar el problema. Obteniendo resultados que se posicionan en un nivel alto y que compiten con otros obtenidos en el estado del arte.

4.6.2. Calificación de Recorridos

Por último, una vez teniendo clasificados los datos, se buscaba generar una función que calificara los recorridos realizados durante la conducción de un automóvil en base a los eventos que se presentaran dentro de la clasificación de conducción temeraria. Habiendo aplicado tanto *TinyGP* y *EvoDAG* sobre los conjuntos de datos generados para estos experimentos, se pudo observar muy claramente que influye demasiado el cómo diseñar los conjuntos de datos, que elementos se tomarán en cuenta para la generación de los conjuntos. Tanto al aplicar *TinyGP* como *EvoDAG*, los errores fueron menores en los recorridos generados con datos de esta investigación, mientras que los que fueron conformados con el diseño utilizado en [14] presentaron errores mayores y desestabilidad, ya que se tomaban en cuenta factores que podrían o no afectar al momento de ir conduciendo, como el clima, mientras que en el conjunto propio, solamente se tomaron en cuenta factores que influyen directamente sobre el desempeño del conductor, como es la velocidad promedio y el número de eventos de conducción temeraria que se presentaban.

Asimismo, se observó que *EvoDAG* muestra un mejor desempeño al momento de predecir las calificaciones para los distintos recorridos, ya que el promedio de error en el conjunto propio no se sobrepasó el valor de 1. Esto indica que el uso de programación genética para calificar recorridos con eventos de conducción temeraria es un buen enfoque y permite una mayor flexibilidad y adaptabilidad sobre los métodos predefinidos basados en puntajes establecidos y descuentos por evento presentado.

5

Conclusiones

La conducción temeraria, especificamente, la identificación de los patrones temerosos para su posterior clasificación es un problema que presenta un gran número de retos, los cuales van desde el como posicionar un dispositivo para capturar los datos, definir que datos capturar, qué técnicas de preprocesamiento aplicarles, pasando por cómo definir un correcto vector de características, qué algoritmos utilizar para atacar el problema, hasta llegar a la problematica de cómo poder, una vez clasificados los patrones, dar una calificación a la conducción acorde los eventos presentados en ella.

Este trabajo abordó cada una de esas problematicas. En primera instancia, el definir cómo capturar los datos, dado que en la gran mayoría de los trabajos existentes en el estado del arte proponen la utilización de un dispositivo en una posición fija, la primer variante en este trabajo fue no predefinir una posición. De tal manera que la captura de datos, no sería dependiente de la posición del dispositivo.

Una vez capturados los datos, el siguiente problema a atacar, fue la definición de un vector de características. Dentro de toda la investigación, esta fue la tarea que más problemas presentó, ya que debido a la arbitrariedad en la posición del dispositivo al capturar los datos, las perturbaciones de los distintos eventos no se presentaban siempre en el mismo eje. Para buscar una representación funcional que fungiera como conjunto de entrada a un algoritmo de clasificación, se intentaron varias técnicas, desde los más simple que fue obtención de máximos y mínimos de cada una de las señales, hasta técnicas más avanzadas y no tan usadas como la metodología SAX [21].

Ninguna de estas técnicas/metodologías resultó exitosa, hasta llegar a la metodología de Bag of Words, la cual dada su naturaleza de generar palabras clave para luego completar un libro con ellas utilizando los distintos segmentos de las señales capturadas, no importaba el eje en el que se presentara la perturbación, ya que su palabra clave que representaría a ese y todo los segmentos con gran similitud, estaría incluída en el libro. Esto para luego comparar cada uno de los segmentos de todos los datos capturados con cada una

de esas generalizaciones contenidas en el libro y así poder definir que palabra clave representaría a dicho segmento. Esto permitió generar un histograma en base a las comparaciones realizadas, el cual sirvió como entrada a un algoritmo de clasificación.

Para la clasficación de los datos, se optó por utilizar primeramente un algoritmo sobre los datos no preprocesados, para indicar que tan necesario era el preprocesamiento de estos. Para ello se utilizó el algoritmo *DTW*, el cual cálcula distancias entre dos señales mediante distancia euclidiana. Sin embargo, debido a que las perturbaciones de eventos del mismo tipo podían presentarse en distinos de señal en señal, al algoritmo le era imposible realizar una buena clasificación.

Una vez con los datos preprocesados y el vector de características generado, se procedió a correr una red neuronal artificial sobre ellos. La red neuronal es un algoritmo potente y no muy complicado, ademas de ser ampliamente usado en problematicas similares, de tal manera que se decidió que era el adecuado para realizar la clasificación de los datos.

Los porcentajes obtenidos fueron muy buenos, en donde 5 de 7 clases presentaron valores alrededor de 90 %, los cuales se muestran altamente competentes con resultados obtenidos por otros trabajos en la literatura, indicando así que se realizó un preprocesamiento adecuado, así como una óptima elección del algoritmo de clasificación.

Con el fin de reafirmar los resultados obtenidos, se procedio a aplicar el algoritmo Naive Bayes, el cual pese a que toma cada uno de los datos de manera independiente, de igual manera mostró un alto nivel de discriminación, nuevamente en 5 de las 7 clases, el porcentaje de clasificación se situó alrededor del 90%.

Teniendo la idea de un trabajo redondeado, en donde se presentara recolección, clasificación y calificación, se decidió por diseñar una técnica de calificación de conducción en base a los eventos temerarios presentados en un recorrido. Intentando darle flexibilidad y adaptabilidad a este apartado, se descartó el calificar de manera predefinida, donde se tiene un porcentaje inicial y se va descontando en base a los eventos presentados.

Para ello se utilizó programación genética, con el fin de crear funciones que pudieran otorgar una calificación de la manera más objetiva posible tomando en cuenta los eventos temerarios presentados durante la conducción. Dos implementaciones fueron usadas, *TinyGP* y *EvoDAG*, donde este último mostró mejores resultados.

Como trabajo futuro, se pretende principalmente mejorar los resultados obtenidos durante la aplicación de programación genética, ya que aunque presentó resultados aceptables, aun muestra un poco de inestabilidad en cuanto a los errores obtenidos y el tamaño de las funciones generadas.

Asimismo se buscará reforzar el apartado de la clasificación mediante la implementación de más algorit-

mos, realizando una comparativa más a fondo para poder definir cuál es el más adecuado y qué mejoras se pueden realizar para aumentar aún más los porcentajes de clasificación.

Referencias

[1] N. Kalra and D. Bansal, "Analyzing driver behavior using smartphone sensors: A survey," *International Journal of Electronic and Electrical Engineering*, vol. 7, pp. 697–702, 2014.

[2] M. Fazeen, B. Gozick, R. Dantu, M. Bhukhiya, and M. C. González, "Safe driving using mobile phones," *IEEE Transactions On Intelligent Transportation Systems*.

[3] S. Pushpendra, N. Juneja, and K. Shruti, "Using mobile pones to detect driving behavior," *ACM DEV'14 Proceedings of the 3rd ACM Symposioum on Computing for Development*.

[4] S. Chigurupa, S. Polavarap, Y. Kancherla, and K. A. Nikhath, "Integrated computing system for measuring driver safety index," *International Journal of Emerging Technology and Advanced Engineering*.

[5] D. Johnson and M. Trivedi, "Driving style recognition using a smartphone as a sensor platform," *IEEE 14th International Conference on Intelligent Transportation system*.

[6] J. Dai, J. Tang, X. Bai, Z. Shen, and D. Xuan, "Mobile phone based drunk driving detection," *Proc. 4th Int Conf Pervasive Health*.

[7] H. Gazali, "Monitorin erratic driving behavior caused by vehicle overtaking using off-the-shelf technologies," 2010.

[8] D. Prarna, S. Shinde, N. Jadav, and A. Bhaduri, "Unsafe driving detection system using smartphone as sensor platform," *IEEE Transactions On Intelligent Transportation Systems*.

[9] Y. Zhang, W. Lin, and Y. K. Chin, "A pattern-recognition approach for driving skill characterization," *IEEE Trans. Intell. Transp. Syst.*

[10] A. Sathyanarayana, P. Boyraz, and J. H. L. Hansen, "Driver behavior analysis and route recognition by hidden markov models," *Vehicular Electronics and Safety, ICVES, IEEE International Conference on IEEE*.

[11] J. H. Hong, B. Margines, and A. K. Dey, “A smartphone-based sensing platform to model aggressive driving behaviors,” *Session: Driving Interfaces and Evaluations*.

[12] V. M. Ly, S. Martin, and M. M. Trivedi, “Driver classification and driving style recognition using imertial sensors,” *IEEE Intelligent Vehicles Symposium (IV)*.

[13] H. Eren, S. Makinist, E. Akin, and A. Yilmaz, “Estimating behavior by a smartphone,” *2012 Intelligent Vechicles Symposium*.

[14] G. Castignani, T. Derrmann, R. Frank, and T. Engel, “Driver behavior profiling using smartphones: A low-cost platform for driver monitoring,” *IEEE Intelligent Transportation Systems Magazine*.

[15] S. Marsland, *Machine Learning: An Algorithmic Prrspective, year = "1993", publisher = .Addison-Wesley", address = Reading, Massachusetts"*.

[16] J. Wang, P. Liu, M. F. H. She, S. Nahavandi, and A. Kouzani, “Bag-of-words representation for biomedical time series classification,” *Biomedical Signal Processing and Control*.

[17] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*, 2008. [Online]. Available: www.//gp-field-guide.org.uk

[18] M. Graff, E. S. Tellez, E. Villasenor, and S. Miranda-Jiménez, “Semantic genetic programming operators based on projections in the phenotype space,” *Research in Computing Science*, vol. 94, pp. 73–85, 2015.

[19] M. Graff, S. E. Tellez, S. Miranda-Jiménez, and H. J. Escalancte, “Evodag: A semantic genetic programming python library,” 2016.

[20] A. Moraglio, R. Poli, and R. Seehuus, “Geometric crossover for biological sequences,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3905 LNCS, 2006, pp. 121–132.

[21] P. Chaovalit, C. Saiprasert, and T. Pholprasit, “A method for driving event detection using sax with resource usage exploration on smartphone platform,” *EURASIP Journal on Wireless Communications and Networking*.