DESARROLLO DE UN COMPONENTE DE DEEP LEARNING PARA EL PROCESAMIENTO DE DATOS MEDIO AMBIENTALES PARA LA PLATAFORMA URB@NECOLIFE

CRISTIAN ALEJANDRO CHAPARRO CUADROS

Nota de Aceptación

Certificamos que el presente Trabajo de Grado
Satisface, en alcances y calidad, todos los requisitos
Que demanda un Trabajo de Grado de Maestría.

Claudia Liliana Zúñiga Cañón
Director

Diego Linares
Jurado

Gloria Ines Alvarez
Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en Ingeniería de Software.

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Cali 27/01/2021

Maestría en Ingeniería
Facultad de Ingeniería y Ciencias

Pontificia Universidad JAVERIANA Cali

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, marzo 03/2021

Autor: Cristian Alejandro Chaparro Cuadros

Título del Trabajo de Grado: "DESARROLLO DE UN COMPONENTE DE DEEP LEARNING PARA EL PROCESAMIENTO DE DATOS MEDIO AMBIENTALES PARA LA PLATAFORMA URB@NECOLIFE"

Director: Claudia Liliana Zúñiga Cañón

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Firma del director del Trabajo de Grado

Hoja de Vida.

Nombre: Cristian Alejandro Chaparro Cuadros
Dirrecion: CLL 4 N 12 -75
Celular: 301-348-62-53
Correo: chaparro.cuadros@gmail.com
Profesion: Ingeniero de Sistemas.
Universidad: Universidad Santiago de Cali.
Empresa: DreamCode.
Cargo: Arquitecto de soluciones.

ALCALDÍA DE
SANTIAGO DE CALI
DEPARTAMENTO ADMINISTRATIVO
DE GESTIÓN MEDIO AMBIENTE

Al contestar por favor cite estos datos:
Radicado No.: **20184133010008295l**
Fecha: **10-05-2018**
TRD: **4133.010.13.1.953.008295**
Rad. Padre: **20184173010055669l2**

CRISTIAN ALEJANDRO CHAPARRO CUADROS
Estudiante
Universidad Javeriana Cali
Correo electrónico cchaparro@javerianacaliedu.onmicrosoft.com

Asunto: Respuesta a radicado padre N° 2018-4173010-055669-2. Entrega de información histórica de datos de medición de calidad del aire

Reciba un cordial saludo

En atención a su solicitud se le hace entrega de manera oficial de los datos de promedios horarios de medición de contaminantes criterio de todas las estaciones de monitoreo que pertenecen al Sistema de Vigilancia de Calidad del Aire de Santiago de Cali –SVCASC con el fin de que puedan realizar los respectivos análisis para su investigación academica.

Es importante señalar que el Departamento Administrativo de Gestión del Medio Ambiente – DAGMA, conservará la propiedad intelectual de la información y análisis de los datos del Sistema de Vigilancia de Calidad del Aire de Santiago de Cali - SVCASC en forma independiente, por lo cual es necesario darle los créditos correspondientes a la entidad en las publicaciones y/o presentaciones que se realicen.

Saludos cordiales

HÉCTOR ALEJANDRO PAZ GÓMEZ
SUBDIRECTOR DE GESTIÓN DE CALIDAD AMBIENTAL
DAGMA

Proyectó: Jefferson Valdes Basto – Contratista
Revisó: Gisela Arizabaleta Moreno - Contratista
Carlos Fernando de Jesús Murillo - Contratista

FICHA RESUMEN
TRABAJO DE GRADO

TÍTULO: DESARROLLO DE UN COMPONENTE DE DEEP LEARNING PARA EL PROCESAMIENTO DE DATOS MEDIO AMBIENTALES PARA LA PLATAFORMA URB@NECOLIFE

1. **ÁREA DE TRABAJO:** Machine Learning, Deep Learning.
2. **TIPO DE PROYECTO:** Investigación.
3. **ESTUDIANTE:** Cristian Alejandro Chaparro Cuadros.
4. **CORREO ELECTRÓNICO:** cchaparro@javerianacali.edu.co
5. **DIRECCIÓN Y TELEFONO:** CLL 4 N 12-75 - 3182432017
6. **DIRECTOR:** Claudia Liliana Zúñiga Cañón
7. **VINCULACIÓN DEL DIRECTOR:** Profesor Titular – Directora Grupo de Investigación COMBA I+D de la Universidad Santiago de Cali.
8. **CORREO ELECTRÓNICO DEL DIRECTOR:** claudia.zuniga00@usc.edu.co
9. **CODIRECTOR (Si aplica):** N/A
10. **GRUPO O EMPRESA QUE LO AVALA (Si aplica**): Laboratorio de Computación Móvil y Banda Ancha (Grupo de Investigación COMBA I+D) de la USC.
11. **OTROS GRUPOS O EMPRESAS:**
12. **PALABRAS CLAVE:** *Machine Learning, Deep Learning, Data Analytics, Urban Data*.
13. **FECHA DE INICIO:** 21/11/2017
14. **DURACIÓN ESTIMADA (En meses):** 12 meses.
15. **RESUMEN:**

El Grupo de Investigación COMBA I+D de la Universidad Santiago de Cali, junto con la Universidad de Vigo y el Centro Universitario de la Defensa de España (CUD), han desarrollado el macroproyecto llamado Urb@nEcoLife. El cual busca, a través de una red de sensores móviles, capturar datos relacionados con la contaminación del aire, combinados con datos obtenidos a través de un sistema de *crowdsourcing*, para luego analizarlos y mediante *Deep Learning*, generar predicciones que ayuden a conocer y prevenir los problemas de contaminación medioambiental que afectan la calidad de vida de las personas en la ciudad de Cali.

El macroproyecto Urb@nEcoLife se estructura en cuatro componentes dependientes denominados así: Crowdsourcing (colaboración abierta distribuida), Big Data (datos masivos), Deep Learning (Aprendizaje profundo) y Visualización.

El componente de Deep Learning del macroproyecto Urb@nEcoLife se divide en dos etapas principales, la primera enfocada al modelo de aprendizaje y la segunda a la generación de alertas tempranas. Bajo este contexto, este proyecto de grado trabaja en la primera fase del componente de Deep Learning.

Resumen

El Grupo de Investigación COMBA I+D de la Universidad Santiago de Cali, junto con la Universidad de Vigo y el Centro Universitario de la Defensa de España (CUD), han desarrollado el macroproyecto llamado Urb@nEcoLife. El cual busca, a través de una red de sensores móviles, capturar datos relacionados con la contaminación del aire. En el siguiente trabajo se muestra la implementación de la metodología *Cross-Industry Standard Process for Data Mining* CRISDM, con el fin de resolver y predecir posibles problemas medio ambientales de la ciudad Cali – Colombia, usando como base un algoritmo de red neuronal recurrente para procesar las series de tiempo armadas de los datos de contaminación de los años 2010 – 2017. Dentro del proyecto se exploraron diferentes tipos de modelos para generar predicciones en diferentes escalas de tiempo.

Palabras clave

Machine Learning, Deep Learning, Data Analytics, Urban Data.

Summary

The COMBA R&D Research Group of the Universidad Santiago de Cali, together with the Universidad de Vigo and the Centro Universitario de la Defensa (CUD), are developing a macroproject called Urb@nEcoLife. Which seeks, through a network of mobile sensors, to capture data related to air pollution. In the following work shows the implementation of the Cross-Industry Standard Process for Data Mining CRISDM methodology, in order to solve predicting possible environmental problems in the city of Cali - Colombia using a recurring neural network algorithm as a basis to process the armed time series of the pollution data for the years 2010 - 2017, within the project different types of models were explored that generate predictions at different time scales.

Keywords

Machine Learning, Deep Learning, Data Analytics, Urban Data.

Vigilada Mineducación Res. 2333 del 2012

DESARROLLO DE UN COMPONENTE DE DEEP LEARNING PARA EL PROCESAMIENTO DE DATOS MEDIO AMBIENTALES PARA LA PLATAFORMA URB@NECOLIFE

Cristian Alejandro Chaparro Cuadros
Código 5572401

Proyecto de trabajo de grado para optar al título de
Magister en Ingeniería de Software

Director
Claudia Liliana Zúñiga Cañón, PhD

FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERIA DE SOFTWARE
SANTIAGO DE CALI, OCTUBRE DE 2020

CONTENIDO

LISTA DE FIGURAS 10
LISTA DE TABLAS 12
1. INTRODUCCIÓN 13
2. DESCRIPCIÓN DEL PROBLEMA DEL PROYECTO 14
2.1 Formulación del problema 14
2.2 Sistematización del problema 14
3. OBJETIVOS 16
3.1 Objetivo general 16
3.2 Objetivos específicos 16
4. MARCO TEÓRICO 17
4.1 Deep Learning 17
4.2 ¿Por qué se llama Deep Learning? 17
4.3 Diferencia entre la red neuronal y la red neuronal de aprendizaje profundo 17
4.4 Diferentes técnicas de aprendizaje profundo 18
4.5 Maldición de la dimensionalidad 19
4.6 Técnicas o arquitecturas de Deep Learning 19
4.7 Recurrent Neural Networks (RNN) 24
4.8 Long Short-Term Memory (LSTM) 27
4.9 Herramientas de Deep Learning 29
4.9.1 Theano 29
4.9.2 Caffe 29
4.9.3 Deeplearning4j 29
4.9.4 TensorFlow 30
4.9.5 Keras 30
5 METOLOGIA CRISP-DM (CRoss-Industry Standard Process for Data Mining) 31
6 ENTRENAMIENTO DEL MODELO BASADO EN DEEP LEARNING PARA DATOS DE LA PLATAFORMA URB@NECOLIFE 33
6.1 Compresión del negocio 33
6.1.1 Objetivos de negocio Urb@nEcoLife 33
6.1.2 Antecedentes o situación actual 34

6.1.3 Determinación de los objetivos del modelo 35
6.2 Compresión de los datos 35
6.2.1 Recolección de datos iniciales 36
6.2.2 Exploración de datos 38
6.2.3 Verificación de calidad de datos 39
6.3 Fase de preparación de los datos 40
6.3.1 Selección de datos 40
6.4 Modelamiento 47
6.4.1 Modelamiento estructura general 47
6.4.2 Modelo base 50
6.4.3 Modelo con el tamaño del parque automovilístico y las 12 características 51
6.4.4 Modelo con la población y las 12 características 51
6.4.5 Modelo con el parque automovilístico, población y las 12 características 52
6.4.6 Modelo de 3 meses 53
6.4.7 Modelo de 3 meses + PIB 53
7 EVALUACIÓN E INTEGRACION DEL MODELO 55
7.1 Evaluación del modelo 55
7.2 Integración del modelo 60
8 CONCLUSIONES Y TRABAJOS FUTUROS 61
9 REFERENCIAS BIBLIOGRÁFICA 63

LISTA DE FIGURAS

Ilustración 1. Censo Nacional de Población y Vivienda 2018 (DANE, 2020) 14
Ilustración 2. Diferencia entre Neural Network y Deep Learning Neural Network (xenonstack, 2017) 18
Ilustración 3. Maldición de la dimensionalidad (Courville, 2016) 19
Ilustración 4. Arquitectura de redes neuronales (Wiseman, 2020) 20
Ilustración 5. Uso de categorías por sector (Wiseman, 2020) 24
Ilustración 6. Secuencia (karpathy, 2020) 24
Ilustración 7. Red neural recurrente tiene ciclos (colah.github.io, 2020). 25
Ilustración 8. Un recorrido de una red neural recurrente (colah.github.io, 2020). 26
Ilustración 9. El problema de la decencia en un largo tiempo (colah.github.io, 2020) 26
Ilustración 10. Conectar información (colah.github.io, 2020). 26
Ilustración 11. La repetición del modelo estándar de RNN contiene una capa simple (colah.github.io, 2020) 27
Ilustración 12. Módulo de una LSTM (colah.github.io, 2020) 27
Ilustración 13. Notación LSTM (colah.github.io, 2020). 28
Ilustración 14. Cinta transportadora (colah.github.io, 2020). 28
Ilustración 15. Puertas LTSM (colah.github.io, 2020). 28
Ilustración 16. Framework de Deep Learning (Hale, 2020) 30
Ilustración 17. CRISP-DM) (crisp-dm.pdf, 2000) 31
Ilustración 18. Contaminación Atmosférica (Minambiente, 2020) 34
Ilustración 19. Archivos DAGMA 36
Ilustración 20. Datos UNIVALLE 36
Ilustración 21. Tasa de crecimiento población 37
Ilustración 22. Cali en cifras 37
Ilustración 23. Promedios Flora 39
Ilustración 24. Datos Validos Flora 40
Ilustración 25. Agrupación Flora 40
Ilustración 26 Motos y vehículos particulares 41
Ilustración 27 Población de la ciudad de cali 41
Ilustración 28 Carga de PIB 41
Ilustración 29 Flora +Población + vehiculos 41
Ilustración 30. Estructuración Flora 42
Ilustración 31. Limpieza Flora 42
Ilustración 32. Aplicación outliers 42
Ilustración 33. Función outliers 42
Ilustración 34. Ajuste outliers 43
Ilustración 35. Cambiar autlier 43
Ilustración 36. Aplicación outliers 43
Ilustración 37 Nulos por registro 44
Ilustración 38Numero de nulos por fila 44
Ilustración 39Borrar registros N/A 44
Ilustración 40. Función is float 45

Ilustración 41. Aplicación función float 45
Ilustración 42. Formateo de columna 45
Ilustración 43Flora total características 46
Ilustración 44. Partición de data 47
Ilustración 45. Separación flora modelo base 47
Ilustración 46 Separación modelo 3 meses 48
Ilustración 47. Funciones serie de tiempo 48
Ilustración 48. Diseño de red neurona 49
Ilustración 49. Ajuste modelo 49
Ilustración 50. Entrenamiento del modelo 49
Ilustración 51. Ajuste modelo base 50
Ilustración 52. Ajuste modelo 2 51
Ilustración 53. Ajuste modelo 3 52
Ilustración 54. Ajuste modelo 13 características 52
Ilustración 55. Ajuste modelo3 meses 53
Ilustración 56. Ajuste modelo 3 meses + PIB 54
Ilustración 57Error cuadrático medio 55
Ilustración 58. MAE y RMSE (medium, 2020) 55
Ilustración 59 Función ajuste r2 56
Ilustración 60. Resumen del modelo 56
Ilustración 61. Modelo 3 meses 56
Ilustración 62. Modelo meses + PIB 57
Ilustración 63. Predicción PIB PM10 50% 57
Ilustración 64. Predicción lluvia PM10 50% 58
Ilustración 65. Predicción población PM10 50% 58
Ilustración 66. Predicción parque automotor PM10 50% 59
Ilustración 67. Arquitectura integración 60
Ilustración 68. Código fuente 60

LISTA DE TABLAS

Tabla 1. Arquitectura de Deep Learning (Wiseman, 2020)..... 23
Tabla 2Implementación de la metodología..... 32
Tabla 3. Niveles de contaminantes..... 35
Tabla 4. Motos y vehículos particular..... 38
Tabla 5. Tipos de datos capturados por las estaciones de medición del aire en la ciudad de Cali 39

1. INTRODUCCIÓN

En la actualidad los problemas medioambientales dentro de las principales ciudades del mundo son algo que se ha vuelto recurrente, las ciudades cuentan con sistemas de monitoreo, como por ejemplo el "Sistema de Vigilancia de la Calidad del Aire de Cali – SVCASC", cuyo principal objetivo es medir continuamente los contaminantes criterio y la meteorología en diferentes puntos de la ciudad de Santiago de Cali.

El Grupo de Investigación COMBA I+D de la Universidad Santiago de Cali, junto con la Universidad de Vigo y el Centro Universitario de la Defensa de España (CUD), han desarrollado un macroproyecto llamado Urb@nEcoLife. El cual busca, a través de una red de sensores móviles, capturar datos relacionados con la contaminación del aire, combinados con datos obtenidos a través de un sistema de crowdsourcing, para luego analizarlos, y mediante *Deep Learning*, generar predicciones del nivel de PM10 de la ciudad de Cali.

El macroproyecto Urb@nEcoLife se estructura en cuatro componentes dependientes denominados así: Crowdsourcing (colaboración abierta distribuida), Big Data (datos masivos), Deep Learning (Aprendizaje profundo) y Visualización.

El componente de Deep Learning del macroproyecto Urb@nEcoLife se divide en dos etapas principales, la primera enfocada al modelo de aprendizaje y la segunda a la generación de alertas tempranas. Bajo este contexto, este proyecto de grado trabajará en la primera fase del componente de Deep Learning.

Para este componente se plantea usar técnicas de *Deep Learning*, (una rama de la Inteligencia Artificial) que permiten por medio de algoritmos, crear sistemas capaces de aprender por sí mismos con base en datos ya procesados. El objetivo de este documento es analizar y convertir estos datos en información relevante que permitan a la plataforma Urb@nEcoLife, realizar predicciones que servirán para la toma de decisiones en cuanto a los problemas medioambientales encontrados.

2. DESCRIPCIÓN DEL PROBLEMA DEL PROYECTO

En las últimas décadas los problemas medioambientales se han convertido en un escenario complejo, puesto que el ritmo de la sociedad ha cambiado. La concentración de las poblaciones en las grandes ciudades de Colombia según el Censo Nacional de Población y Vivienda del 2018 del Departamento Administrativo Nacional de Estadística (DANE), muestra que el 7.1% de la población de Colombia vive en centros poblados (DANE, 2020).

Ilustración 1. Censo Nacional de Población y Vivienda 2018 (DANE, 2020)

Así mismo, en estas zonas urbanas donde se concentran mayores tasas de población, se registra un crecimiento del parque automotor. Según el Registro Único Nacional de Transito (RUNT), actualmente en Colombia existe un parque automotor acumulado registrado a nivel nacional de 15.482.682 (RUNT, 2020), puntualmente en la ciudad de Cali el indicador de crecimiento del parque automotor fue del 4.8% para el 2017.

Es por esto, por lo que el proyecto Urb@nEcoLife con la ayuda del componente de Deep Learning, busca crear un modelo que permita conocer y predecir niveles de contaminación (PM10) para la ciudad de Cali.

Para que un sistema de información pueda realizar predicciones, es necesario que éste cuente con un histórico de datos almacenados y múltiples algoritmos diseñados explícitamente para cada situación que se desee predecir, este método elevaría los costos del sistema en cuanto a desarrollo, mantenimiento y escalabilidad del mismo. Por esta razón, el presente proyecto opta por utilizar algoritmos de *Deep Learning* que gracias a su capacidad de aprender de muchas y diferentes fuentes datos.

2.1 Formulación del problema

¿Cómo desarrollar un componente de Deep Learning para el procesamiento de datos medioambientales integrado a la plataforma Urb@nEcoLife?

2.2 Sistematización del problema

¿Cuáles son las técnicas de *Deep Learning* y herramientas de software que mejor se adaptan al análisis de datos medioambientales?

¿Cómo entrenar un modelo de *Deep Learning* para que sea capaz de predecir niveles de contaminación (PM10) para la ciudad de Cali?

¿Cómo evaluar el desempeño de un modelo basado en *Deep Learning* para datos de la plataforma Urb@nEcoLife?

¿Cómo integrar el modelo basado en *Deep Learning* a la plataforma Urb@nEcoLife para que permita generar las predicciones de posibles problemas medio ambientales?

3. OBJETIVOS

3.1 Objetivo general

Implementar una solución bajo el macroproyecto Urb@nEcoLife que pueda procesar la información generada por las fuentes de la plataforma y que permita generar predicción de posibles problemas medio ambientales usando técnicas de *Deep Learning*.

3.2 Objetivos específicos

- Realizar una revisión literaria de las técnicas y herramientas de *Deep Learning*, para el procesamiento de series temporales.
- Realizar el entrenamiento del modelo basado en *Deep Learning* para datos de la plataforma Urb@nEcoLife.
- Evaluar el desempeño del modelo basado en *Deep Learning* para datos de la plataforma Urb@nEcoLife.
- Integrar el modelo basado en *Deep Learning* a la plataforma Urb@nEcoLife para que permita generar las predicciones de posibles problemas medio ambientales.

4. MARCO TEÓRICO

El siguiente capítulo se desarrolla el marco teórico del proyecto, en el cual se presentan los principales referentes teóricos usados en desarrollo de éste.

4.1 Deep Learning

El aprendizaje profundo permite modelos computacionales que se componen de múltiples capas de procesamiento, para aprender representaciones de datos con múltiples niveles de abstracción. Estos métodos han mejorado drásticamente el estado del arte en reconocimiento de voz, reconocimiento de objetos visuales, detección de objetos y muchos otros dominios, como la farmacología y la genómica (NatureDeepReview, 2020).

4.2 ¿Por qué se llama Deep Learning?

Una red neuronal tradicional consta de un numero de capas reducido y este tipo de estructura de red neuronal no es adecuada para el cálculo de redes más grandes. Por lo tanto, se introduce una red neuronal que tiene más de 10 o incluso más de 100 capas (Deeplearning4, 2020).

Este tipo de estructura está destinada a trabajarse con *Deep Learning*. En la cual, se desarrolla una pila de la capa de neuronas. La capa más baja en la pila es responsable de la recolección de datos en bruto, como imágenes, videos, texto, entre otros.

Cada neurona de la capa más baja almacenará la información y la pasará a la siguiente capa de neuronas, y así sucesivamente. A medida que la información fluye dentro de las neuronas de las capas, se extrae la información oculta de los datos (Learning, 2020).

Por lo tanto, se puede concluir que a medida que los datos se mueven de la capa más baja a la capa más alta (que se ejecuta en el interior de la red neuronal), se recopila información más abstracta.

4.3 Diferencia entre la red neuronal y la red neuronal de aprendizaje profundo

La mayor parte de los métodos de aprendizaje emplean arquitecturas de redes neuronales, por lo que, a menudo, los modelos de aprendizaje profundo se denominan redes neuronales profundas. El término “profundo” suele hacer referencia al número de capas ocultas en la red neuronal. Las redes neuronales tradicionales solo contienen dos o tres capas ocultas, mientras que las redes profundas pueden tener hasta 150. Los modelos de *Deep Learning* se entrenan mediante el uso de extensos conjuntos de datos etiquetados y arquitecturas de redes neuronales que aprenden directamente a partir de los datos, sin necesidad de una extracción manual de características (mathworks, 2020).

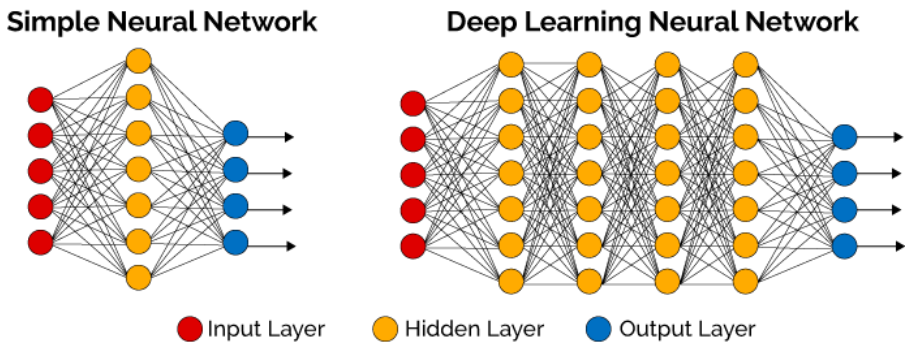


Ilustración 2. Diferencia entre Neural Network y Deep Learning Neural Network (xenonstack, 2017)

4.4 Diferentes técnicas de aprendizaje profundo

Redes neuronales convolucionales: Es un tipo de red que se basa en el aprendizaje del peso y los sesgos. Cada capa de entrada se compone de un conjunto de neuronas donde en cada entrada se realiza un producto de puntos y avanza con el concepto de no linealidad. Es un tipo de red totalmente conectada que usa la función SVM (Una Máquina de Vectores de Soporte) / Softmax (función exponencial normalizada) como función de pérdida (Udacity, 2020).

Máquina de Boltzmann restringida: Es una especie de red neuronal estocástica que consta de una capa de unidades visibles, una capa de unidades ocultas y una unidad de polarización. La arquitectura se desarrolla de tal manera que cada unidad visible se conecta a todas las unidades ocultas y las unidades de polarización se unen a todas las unidades visibles y ocultas. Durante el proceso de aprendizaje, la restricción se desarrolla de modo que ninguna unidad visible se conecta con ninguna unidad visible y ninguna unidad oculta se conecta con ninguna unidad oculta (Hinton, 2020).

Red neuronal recurrente*:* Es el tipo de red neuronal de aprendizaje profundo que utiliza los mismos pesos recursivamente para realizar predicciones de estructura sobre el problema. El gradiente estocástico se utiliza para el entrenamiento de la red mediante el algoritmo de retropropagación (Kim, 2020).

4.5 Maldición de la dimensionalidad

Muchos problemas de aprendizaje máquina se vuelven bastante complejos cuando la dimensión de los datos es alta. Más específicamente, el número de configuraciones distintas de un conjunto de variables aumenta exponencialmente con el número de variables. Esto se conoce como la maldición de la dimensionalidad, concepto introducido por Richard E. Bellman (springer, 2020) . Otra manera de decir esto es que conforme el número de variables aumenta, la cantidad de datos que se necesitan para generalizar crece de manera exponencial.

Ilustración 3. Maldición de la dimensionalidad (Courville, 2016)

4.6 Técnicas o arquitecturas de Deep Learning

Las arquitecturas de aprendizaje profundo son numerosas y a menudo, son una rama o variante de alguna arquitectura original, e incluyen una multitud de diferentes algoritmos asociados. Es importante tener en cuenta que comparar el rendimiento de múltiples arquitecturas es un reto, ya que sus capacidades a menudo se informan con base a diferentes conjuntos de datos (Wiseman, 2020).

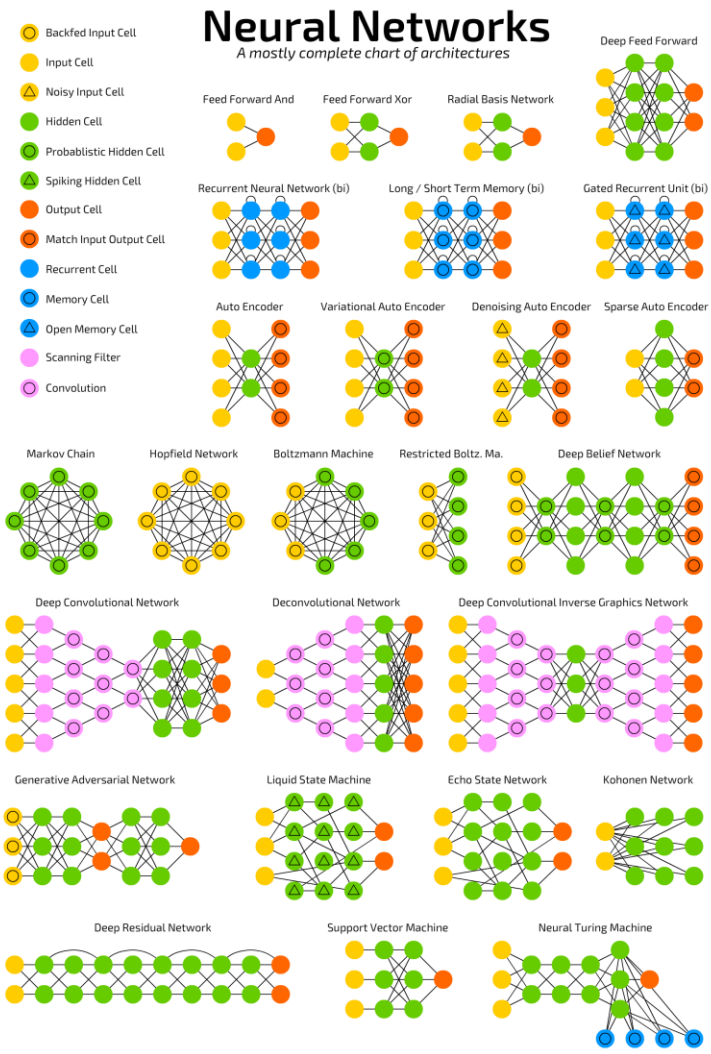
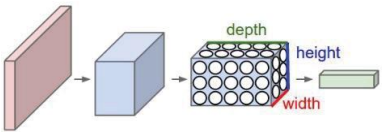
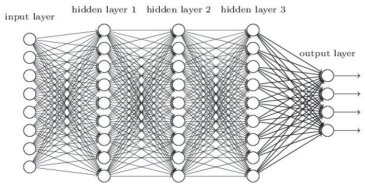
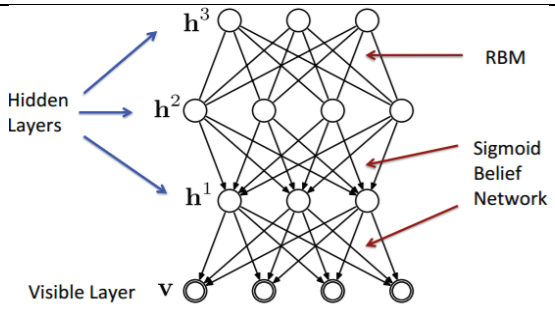
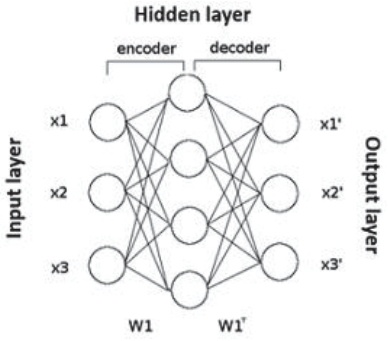
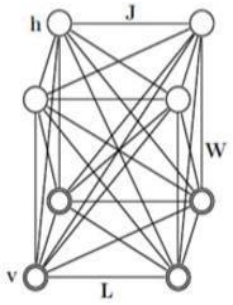
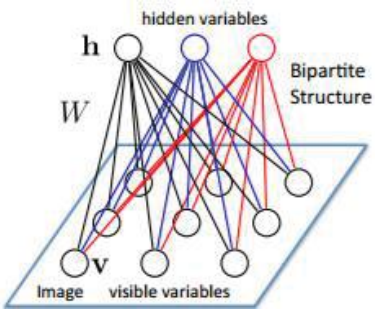
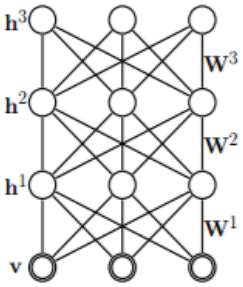
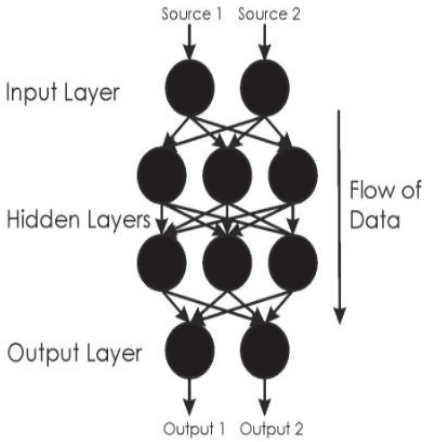


Ilustración 4. Arquitectura de redes neuronales (Wiseman, 2020)

A continuación, se presenta un listado de las arquitecturas de aprendizaje profundo y se hace un especial énfasis en las arquitecturas que serían la mejor opción para el objetivo del proyecto.

Arquitectura/Descripción	Fecha de introducción/Imagen
Redes neuronales convolucionales (CNN)	**1970s**
Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general se añade una función para realizar un mapeo causal no-lineal.	
Deep Neural Network (DNN)	**Mid -2000s**
Una red de múltiples capas con muchas capas ocultas, cuyos pesos están completamente conectados y a menudo, se inicializan (se entrenan previamente) utilizando máquinas Boltzmann restringidas apiladas (RBM) o *Deep Belief Networks* (DBN). (En la literatura, DBN se usa a veces para significar DNN).	
Deep Belief Network (DBN)	**2006**
Un RBM apilado. Modelos generativos probabilísticos compuestos de múltiples capas de variables ocultas y estocásticas. Las dos capas superiores tienen conexiones simétricas no dirigidas entre ellas. Las capas inferiores reciben conexiones dirigidas desde arriba hacia abajo desde la capa superior.	
Deep Autoencoder	**1986**
Un DNN cuyo objetivo de salida es la entrada de datos, a menudo se entrena previamente con DBN o se usan datos de entrenamiento distorsionados para regularizar el aprendizaje.	
Boltzmann Machine (BM)	**1986**

Una red de unidades similares a neuronas conectadas simétricamente que toman decisiones estocásticas sobre si estar encendido o apagado.	
Restricted Boltzmann Machine (RBM)	**1986**
Un BM especial que consiste en una capa de unidades visibles y una capa de unidades ocultas sin conexiones visibles-visibles u ocultas-ocultas.	
Deep Boltzmann Machine (DBM)	
Un BM especial donde las unidades ocultas se organizan de manera profunda en capas, solo las capas adyacentes están conectadas, y no hay conexiones visibles, visibles u ocultas dentro de la misma capa.	
Feedforward Neural Networks	
Una red de perceptrones organizados en capas, con la primera capa tomando entradas y la última capa produciendo salidas. Las capas medias están ocultas. Cada perceptrón en una capa está conectado a cada perceptrón en la siguiente	
Multilayer Perceptron (MLP)	**1974,1986**

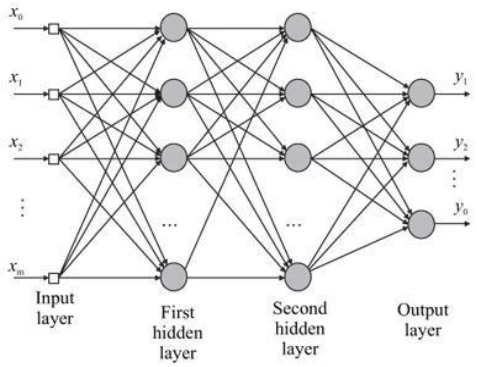
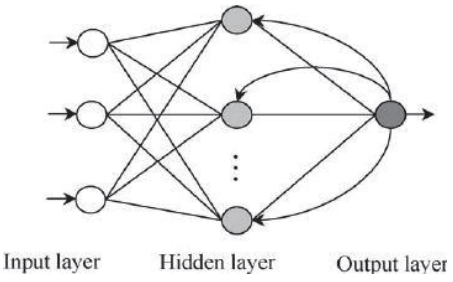
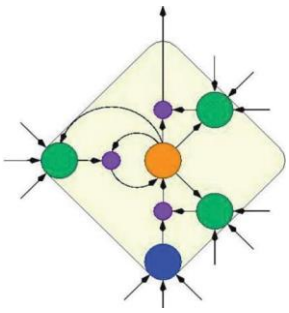
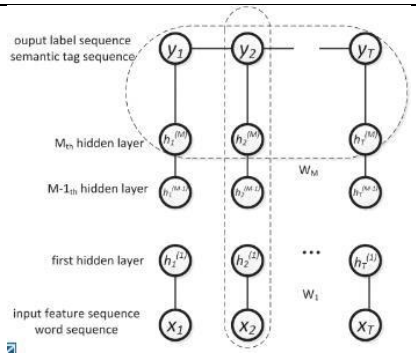
Un tipo de modelo de red neuronal de alimentación hacia adelante que tiene una capa o más de unidades ocultas y activaciones no lineales. Las redes MLP se usan generalmente en problemas de aprendizaje supervisado y se resuelven con algoritmos de propagación hacia atrás o con menor frecuencia para aprendizaje no supervisado utilizando estructuras autoasociativas	
Recurrent Neural Networks (RNN)	**1991**
Una red que contiene al menos una conexión de retroalimentación o un bucle que le permite realizar procesamientos temporales y aprender secuencias. La forma más simple de un RNN completo es un MLP con el conjunto anterior de activaciones de unidades ocultas que se retroalimenta en la red junto con las entradas. Los RNNs tienen memoria que les permite aprender secuencias	
Long Short-Term Memory (LSTM)	**1997**
Una variación eficiente y basada en gradientes de RNN que supera los desafíos de almacenar información en intervalos de tiempo prolongados (tareas de retraso prolongado) mediante la ejecución de un flujo de retorno de error constante.	
Conditional Random Field (CRF)	
Un método probabilístico popular para la predicción estructurada que se ha combinado con redes neuronales en una variedad de formas.	

Tabla 1. Arquitectura de Deep Learning (Wiseman, 2020)

La Ilustración 5 presenta una mirada más cercana a las arquitecturas utilizadas para el apoyo en las decisiones humanas. Donde se puede observar que las redes neuronales convolucionales CNN son la arquitectura más utilizada por todos los grupos analizados, excepto en los sistemas de recomendación y análisis de incertidumbre, que utilizan redes neuronales profundas DNN con una frecuencia ligeramente mayor. También es importante señalar que CNN y DNN se usan casi por igual en predicción / predicción, minería de datos, búsqueda, educación y toma de decisiones / apoyo (humano), por esta razón solo se va a profundizar en redes neuronales recurrentes y redes neuronales profundas.

Architectures

Subset	CNN	DNN	DBN	Auto encoder	RNN	RBM	BM	FNN	LSTM	MLP	DBM	CRF	DSN
Forecasting/ Prediction	165	133	72	45	51	37	28	23	15	18	6	13	1
Data mining	54	41	15	14	14	22	17	2	4	4	2	1	
Searching	88	71	12	6	8	5	4	9	5	4		1	2
Diagnostics (medical)	105	31	32	30		20	16	5		4	7	2	
Emotion/ Sentiment recognition	75	36	32	17	20	18	17	10	9	6	3	2	
Big data analytics	41	36	14	11	10	10	8	6		3	1	1	
Human activity recognition	94	37	18	16	18	8	8	7	8	4	2	1	
Social media analytics	45	22	15	9	5	9	8	2	1			3	
Education	22	19	1	6	7	2	3	2	2	2	2		
Decision making/support (hum..	16	14	13	8	2	2	1	1	2	2			
Visualization	43	21	3	10	1	3	1	3	1	1			
Event detection	28	18	12	5	9	6	5	1	5	2		1	
Security systems	41	10	4	4	2	3	2	2	2				
Surveillance	45	8	5	3	1	1	1	4	1				
Recommender systems	11	16	3	4	9	2	1	1	2	1			
Computer aided instruction	29	13	1	2	2	1	1		1	1			
Question-Answering	24	11	4		13		1	3	2		1		
Uncertainty analysis	5	8	5	2	1	5	4			2	2		
Data fusion	15	6	3	3	4			4	3		1		
Virtual/ Augmented reality	15	7	1	4		2	2						
Risk analysis	4	3	3	1		2	1			1			
Military	6	2	5			2	1	2					

Ilustración 5. Uso de categorías por sector (Wiseman, 2020)

4.7 Recurrent Neural Networks (RNN)

Cada rectángulo es un vector y las flechas representan funciones (por ejemplo, multiplicación de matriz). Los vectores de entrada están en rojo, los vectores de salida están en azul y los vectores verdes mantienen el estado del RNN. Secuencias:

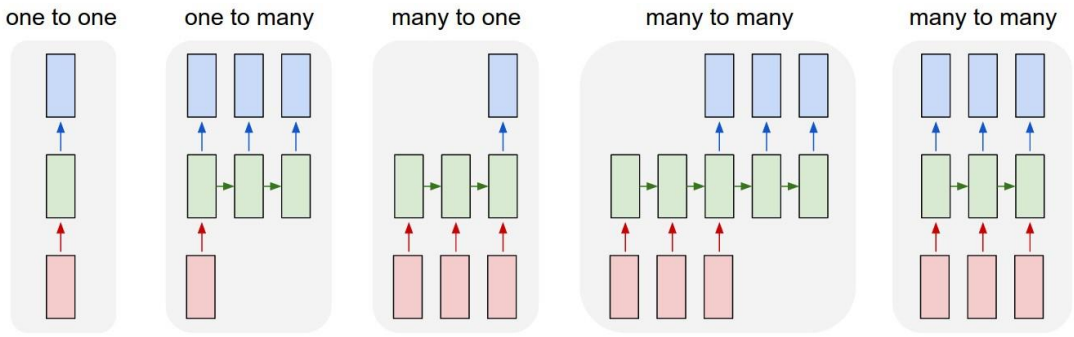


Ilustración 6. Secuencia (karpathy, 2020)

De izquierda a derecha:

- Modo de procesamiento de clásico sin RNN, desde entrada de tamaño fijo hasta salida de tamaño fijo (por ejemplo, clasificación de imágenes).
- Salida de secuencia (por ejemplo, los subtítulos de imágenes toman una imagen y generan una oración de palabras).
- Entrada de secuencia (por ejemplo, análisis de sentimientos donde una oración dada se clasifica como expresión de sentimientos positivos o negativos).
- Entrada de secuencia y salida de secuencia (por ejemplo, traducción automática: un RNN lee una oración en inglés y luego emite una oración en francés).
- Entrada y salida de secuencia sincronizada (por ejemplo, clasificación de video donde deseamos etiquetar cada cuadro del video).

La recurrencia es un proceso que se puede observar asociado al pensamiento humano, donde siempre se parte desde un pensamiento base. Por ejemplo, al realizar una lectura, esta es comprendida a medida que se va comprendiendo cada palabra anterior. Los pensamientos tienen persistencia. Las redes neuronales tradicionales no pueden hacer esto y es allí donde está su principal deficiencia. Las redes neuronales recurrentes abordan este problema. Son redes con bucles, permitiendo que la información persista (colah.github.io, 2020).

En la siguiente gráfica, una parte de la red neuronal, A, observa alguna entrada Xt y genera un valor Ht.

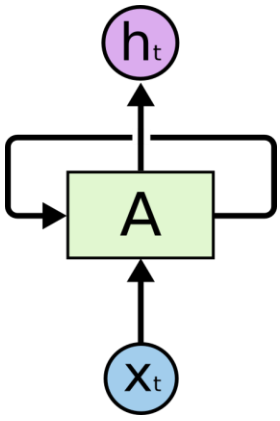


Ilustración 7. Red neural recurrente tiene ciclos (colah.github.io, 2020).

Un bucle permite que la información pase de un punto de la red al siguiente, una red neuronal recurrente se puede considerar como copias múltiples de la misma red, cada una de las cuales pasa un mensaje a un sucesor.

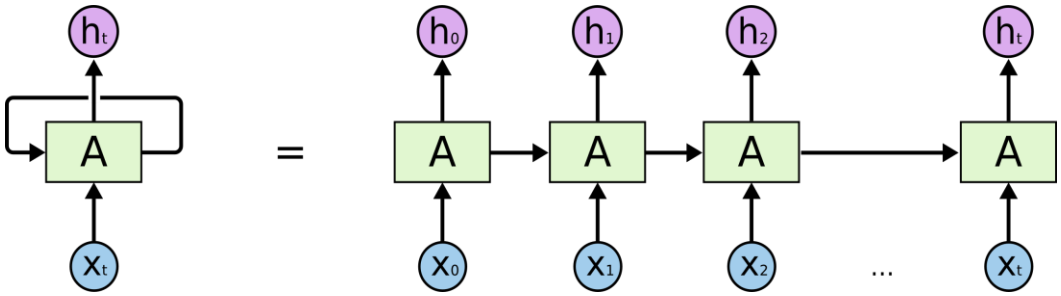


Ilustración 8. Un recorrido de una red neural recurrente (colah.github.io, 2020).

Una de las características de las RNN, es la idea de que podrían ser capaces de conectar información previa a la tarea presente, como el uso de información de estados anteriores podría informar la comprensión del estado actual.

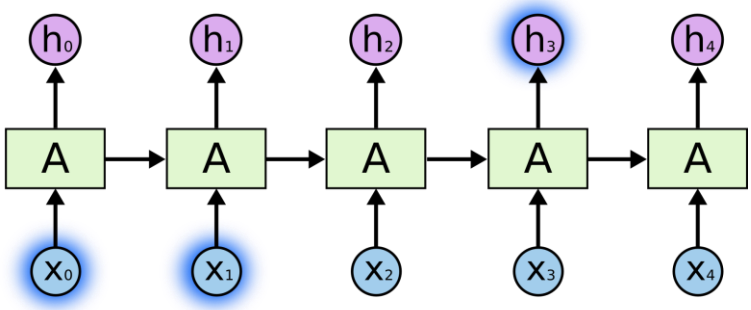


Ilustración 9. El problema de la decencia en un largo tiempo (colah.github.io, 2020)

Así mismo, hay casos en los que es necesario tener más contexto, el cual no es más que información de momentos pasados en la bandada de tiempo. Es totalmente posible que la brecha entre la información relevante y el punto en el que se necesita sea muy grande. Desafortunadamente, a medida que aumenta la brecha, los RNN no pueden aprender a conectar la información.

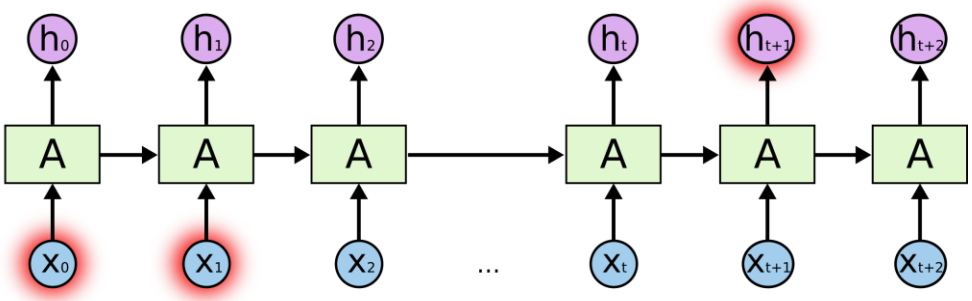


Ilustración 10. Conectar información (colah.github.io, 2020).

4.8 Long Short-Term Memory (LSTM)

Las redes de memoria a largo plazo, generalmente llamadas "LSTM", son un tipo especial de RNN, capaz de aprender dependencias a largo plazo. Fueron introducidas por Hochreiter y Schmidhuber (1997).

Las LSTM están diseñados explícitamente para evitar el problema de dependencia a largo plazo. Recordar información durante largos períodos de tiempo es prácticamente su comportamiento predeterminado, todas las redes neuronales recurrentes tienen la forma de una cadena de módulos repetitivos de la red neuronal. En las RNN estándar, este módulo de repetición tendrá una estructura muy simple, como una sola capa (tanh).

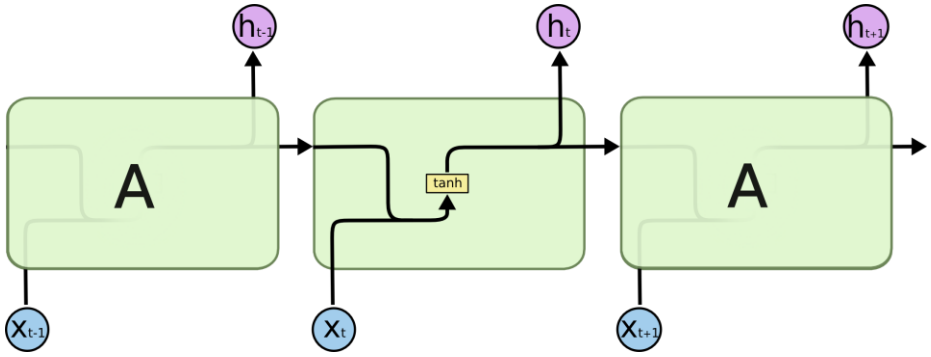


Ilustración 11. La repetición del modelo estándar de RNN contiene una capa simple (colah.github.io, 2020)

Las LSTM también tienen esta estructura de cadena, pero el módulo de repetición tiene una estructura diferente. En lugar de tener una sola capa de red neuronal, hay cuatro que interactúan de una manera especial.

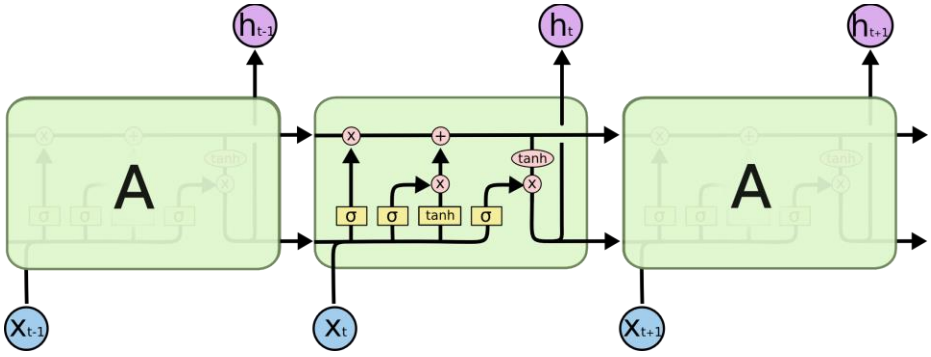


Ilustración 12. Módulo de una LSTM (colah.github.io, 2020)

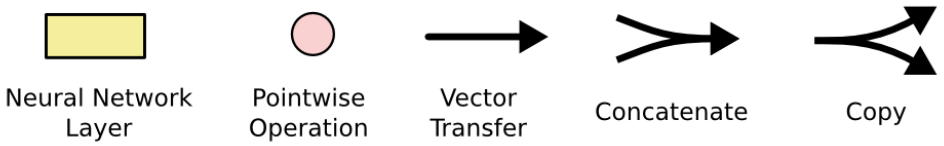


Ilustración 13. Notación LSTM (colah.github.io, 2020).

La clave para las LSTM es el estado de la celda, la línea horizontal que recorre la parte superior del diagrama corre directamente por toda la cadena, con solo algunas interacciones lineales menores. Es muy fácil que la información fluya sin cambios.

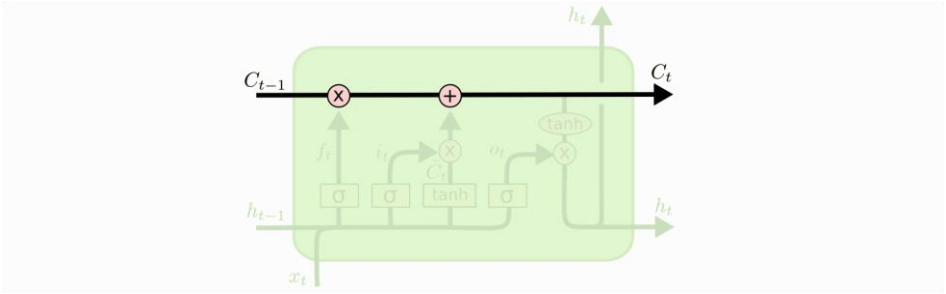


Ilustración 14. Cinta transportadora (colah.github.io, 2020).

Las LSTM tienen la capacidad de eliminar o agregar información al estado de la celda, cuidadosamente regulado por estructuras llamadas compuertas.

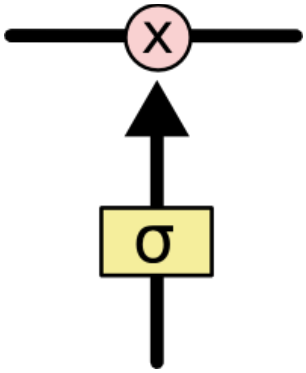


Ilustración 15. Puertas LTSM (colah.github.io, 2020).

La capa sigmoide produce números entre cero y uno, que describen la cantidad de cada componente que debe dejarse pasar. Un valor de cero significa "no dejar pasar nada", mientras que un valor de uno significa "dejar pasar todo".

4.9 Herramientas de Deep Learning

Implementar una arquitectura profunda realmente es posible, pero empezar de cero puede requerir mucho tiempo y también necesitan de tiempo para optimizar y madurar. En este capítulo se realiza una recopilación de las principales herramientas o infraestructuras de código abierto para implementar los algoritmos de aprendizaje profundo.

4.9.1 Theano

Theano fue un marco Python desarrollado en la Universidad de Montreal y administrado por Yoshua Bengio, para la investigación y el desarrollo de algoritmos de aprendizaje profundo. Que le permiten definir, optimizar y evaluar expresiones matemáticas con matrices multidimensionales de manera eficiente (cv-trick, 2020).

4.9.2 Caffe

Caffe se desarrolló originariamente como parte de una tesis doctoral y posteriormente se lanzó bajo la licencia de Distribución de Software de Berkeley. Caffe soporta una amplia variedad de arquitecturas de aprendizaje profundo, incluidas CNN y LSTM, pero destaca que no soporta RBMs ni DBMs. Caffe se ha utilizado para la clasificación de imágenes y para otras aplicaciones de visualización, soporta la aceleración basada en la GPU con la biblioteca NVIDIA CUDA Deep Neural Network. Caffe soporta Open Multi-Processing (OpenMP) para la paralización de algoritmos de aprendizaje profundo sobre un clúster de sistemas. Caffe y Caffe2 están escritos en C++ por causa del rendimiento y ofrecen una interfaz de Python y de MATLAB para la capacitación y la ejecución del aprendizaje profundo (Caffe, 2020).

4.9.3 Deeplearning4j

Deeplearning4j es una infraestructura popular de aprendizaje profundo que se centra en la tecnología Java, e integra interfaces de programación de aplicaciones para otros lenguajes, como: Scala, Python y Clojure. La infraestructura se libera bajo licencia de Apache e incluye soporte para RBMs, DBNs, CNNs y RNNs. Deeplearning4j también considera versiones paralelas distribuidas que funcionan con Apache Hadoop y Spark (infraestructuras de procesamiento de Big data).

Deeplearning4j tiene una amplia aplicabilidad, usándose incluso para la detección de fraudes en el sector financiero, sistemas de recomendación, reconocimiento de imágenes y seguridad cibernética (detección de intrusos en la red). La infraestructura se integra con CUDA por causa de la optimización de la GPU y se puede distribuir con OpenMP o Hadoop (deeplearning4j, 2020).

4.9.4 TensorFlow

TensorFlow ha sido desarrollado por Google como una biblioteca de código abierto y es un descendiente de DistBelief de código cerrado. Es posible utilizar TensorFlow para entrenar e implementar diferentes redes neurales (CNNs, RBMs, DBNs y RNNs) y se libera bajo la licencia Apache 2.0. TensorFlow se ha aplicado a varios problemas, como: la subtitulación de imágenes, la detección de malware, el reconocimiento de voz y la recuperación de información. Asi mismo, se tiene disponible una pila centrada en Android llamada TensorFlow Lite.

Es posible desarrollar aplicaciones con: TensorFlow en Python, C++, el lenguaje Java, Rust o Go (aunque Python es el más estable) y distribuir su ejecución con Hadoop. TensorFlow soporta CUDA además de interfaces especializadas en hardware (Tensorflow, 2020).

4.9.5 Keras

Keras es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque en permitir la experimentación rápida y asi poder pasar de la idea al resultado con el menor retraso posible (keras, 2020).

- Keras permite la creación de prototipos fácil y rápido (a través de la facilidad de uso, la modularidad y la extensibilidad).
- Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.
- Se ejecuta sin problemas en CPU y GPU.

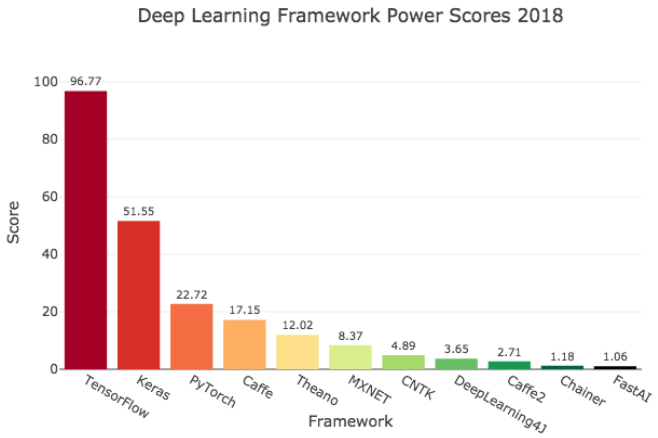


Ilustración 16. Framework de Deep Learning (Hale, 2020)

5 METOLOGIA CRISP-DM (CRoss-Industry Standard Process for Data Mining)

Para el desarrollo del proyecto se siguió como referencia la metodología de CRoss-Industry Standard Process for Data Mining CRISP-DM, la cual, sirve para definir y desarrollar un proyecto de minería de datos. Esta metodología se centra principalmente en un proceso interactivo que parte del entendimiento del negocio y la definición de sus objetivos a explorar. Una vez se tiene claro el objeto del negocio se procede a realizar una selección, agrupación y formateo de la data necesaria para el entrenamiento del modelo.

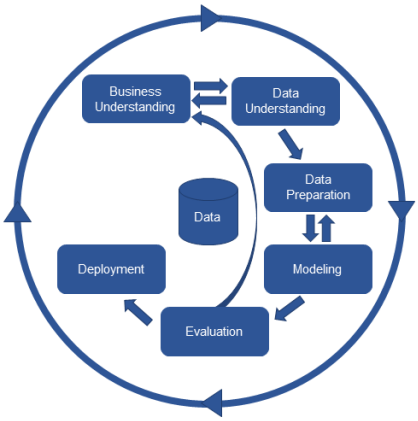


Ilustración 17. CRISP-DM) (crisp-dm.pdf, 2000)

Con la data en el formato necesario, el siguiente paso es definir el modelo o arquitectura de minería de datos a implementar. La elección podría depender de la calidad de la data, conocimiento del equipo de trabajo o el enfoque del caso de negocio. Después de la elección del modelo se procede a la evaluación e implementación. En este punto es posible que los objetivos del negocio se cumplan o sea necesario devolverse a una etapa anterior de la metodología, hasta cumplir los objetivos de negocio y poder llegar hasta la última fase, la cual es presentar los resultados del modelo en términos del negocio.

La metodología CRISP-DM se divide en 6 fases (ver Ilustración 17):

- **Compresión del negocio:** Esta fase inicial se enfoca en la comprensión de los objetivos de proyecto y exigencias desde una perspectiva de negocio, convirtiendo posteriormente este conocimiento de los datos en la definición de un problema de minería de datos y en un plan preliminar diseñado para alcanzar los objetivos.
- **Compresión de los datos:** La segunda fase, comprende la recolección inicial de datos, con el objetivo de establecer un primer contacto con el problema, familiarizándose con ellos, identificar su calidad y establecer las relaciones más evidentes que permitan definir las primeras hipótesis.
- **Preparación de los datos:** La fase de preparación de los datos cubre todas las actividades necesarias para construir una estructura de datos finales a partir de unos datos iniciales. Las tareas de preparación de datos usualmente son realizadas muchas veces y no en cualquier orden prescripto. Las tareas incluyen la selección de tablas, registros y atributos, así como, la transformación y la limpieza de datos para las herramientas que modelan.

- **Modelamiento:** En esta fase de modelamiento, se seleccionan las técnicas de modelado más apropiadas para el proyecto de *Data Mining* específico, las técnicas a utilizar en esta fase se eligen en función de los siguientes criterios:

    - Ser apropiada al problema.
    - Disponer de datos adecuados.
    - Cumplir los requisitos del problema.
    - Tiempo adecuado para obtener un modelo.
    - Conocimiento de la técnica.

- **Evaluación:** En esta fase se evalúa el modelo, teniendo en cuenta el cumplimiento de los criterios de éxito del problema. Debe considerarse, además, que la fiabilidad calculada para el modelo se aplica solamente para los datos sobre los que se realizó el análisis.

- **Implementación:** En la fase de implementación una vez que el modelo ha sido construido y validado, se transforma el conocimiento obtenido en acciones dentro del proceso de negocio, ya sea que el analista recomiende acciones basadas en la observación del modelo y sus resultados.

De esta manera, en el desarrollo del proyecto se hizo uso de la metodología CRISP-DM en sus seis fases. A continuación, se relaciona cada fase para cumplir el alcance del proyecto (Ver Tabla 2):

Fases de la Metodología CRISP- DM	
Entrenamiento del modelo	Fase 1: Compresión del negocio Fase 2: Compresión de los datos Fase 3: Preparación de los datos Fase 4: Modelamiento
Evaluación e integración del modelo	Fase 5: Evaluación Fase 6: Implementación

Tabla 2Implementación de la metodología

6 ENTRENAMIENTO DEL MODELO BASADO EN DEEP LEARNING PARA DATOS DE LA PLATAFORMA URB@NECOLIFE

Para realizar el entrenamiento del modelo basado en *Deep Learning* para los datos de la plataforma Urb@nEcoLife se siguió la metodología CRISP- DM, en este capítulo se van a aplicar las 4 primeras fases de la metodología:

- Comprensión del negocio.
- Compresión de los datos.
- Preparación de los datos.
- Modelamiento

6.1 Compresión del negocio

Agrupa las tareas de comprensión de los objetivos y requisitos del proyecto Urb@nEcoLife desde una perspectiva empresarial o institucional, con el fin de convertirlos en objetivos técnicos y en un plan de proyecto.

Actividades:

- **Determinar los objetivos del negocio:** Se determina cuál es el problema que se desea resolver en el ámbito del proyecto Urb@nEcoLife.
- **Evaluación de la situación:** Se realiza una descripción de la problemática actual de contaminación del aire de las ciudades colombianas.
- **Determinación de los objetivos del modelo:** Se busca representar los objetivos del negocio en términos de las metas del proyecto Urb@nEcoLife.
- **Realizar plan de proyecto:** Se desarrolla un plan para el proyecto, que describa los pasos a seguir y las técnicas a emplear en cada paso. Determinación de objetivos del negocio.

6.1.1 Objetivos de negocio Urb@nEcoLife

El objetivo principal es entrenar y validar un algoritmo de *Deep Learning* con los datos históricos de contaminación ambiental de la ciudad de Cali, de tal forma que pueda ser usado por la plataforma Urb@nEcoLife, con el fin de desarrollar una serie de servicios que permita a entes gubernamentales y a ciudadanos del común, la toma decisiones para disminuir la contaminación ambiental.

6.1.2 Antecedentes o situación actual

Las ciudades se convierten en uno de los principales centros de desarrollo de un país. Las ciudades han tenido un crecimiento poblacional que ha venido en aumento en los últimos años. De acuerdo con la Organización Mundial de la Salud (OMS), desde el año 2009 más de la mitad de la mitad de la población mundial ya vivía en centros urbanos, además, se espera que para el año 2050, el 70 % de la población mundial viva en las ciudades. De acuerdo con un análisis publicado por la OMS en mayo de 2018, el 90 % de la población urbana del mundo respira aire con niveles de contaminación que son mucho más altos que los umbrales recomendados (OMS, 2020).

Según los análisis realizados por el Ministerio de Ambiente y Desarrollo Sostenible de Colombia, la contaminación atmosférica en el país es uno de los problemas ambientales de mayor preocupación para los colombianos, por los altos impactos generados tanto en la salud como en el medio ambiente. Además, es el tercer factor generador de costos sociales después de la contaminación del agua y de los desastres naturales (Minambiente, 2020).

La sociedad tiene una gran oportunidad en abordar este tipo de problemáticas, a través del uso y aprovechamiento de las tecnologías de la información. Las Tecnologías de la Información y la Comunicación (TIC) pueden ayudar en el proceso proporcionando herramientas ya desarrolladas para: procesar gran cantidad de información, medir y evaluar el impacto, pronosticar e informar sobre niveles de contaminación. En el desarrollo de este trabajo se encontró una variedad de trabajos de referencia que hacen uso de las TIC para tratar de abordar problemas de contaminación ambiental. Las TIC constituyen una herramienta para que los ciudadanos y las autoridades trabajen juntos en el camino hacia la reducción de la contaminación atmosférica en las ciudades.

Como se muestra en la imagen (ilustración 18. Contaminación atmosférica), una de las principales fuentes de contaminación es la derivada por el alto tráfico de carros que se encuentran dentro de las ciudades. En Cali la tasa de motorización general tuvo un crecimiento en el 2017 del 4.8% con 630.478 vehículos (Tasa de Motorización, 2020).

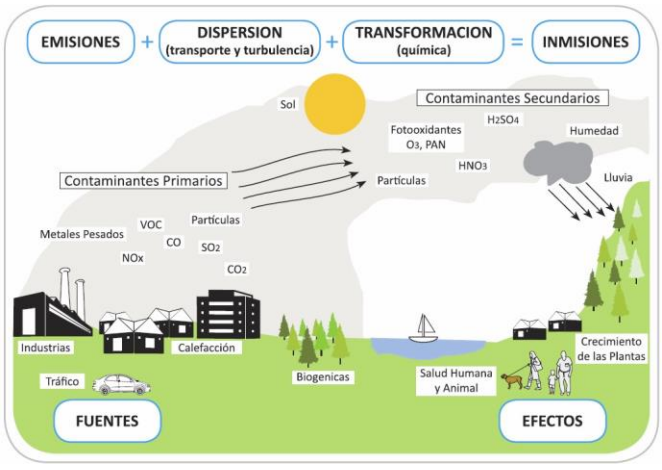


Ilustración 18. Contaminación Atmosférica (Minambiente, 2020)

EL Ministerio de Ambiente y Desarrollo Sostenible de Colombia determinó mediante la Resolución 2254 de 01 noviembre de 2017, que los niveles máximos permisibles de contaminación eran los siguientes:

Contaminante	Niveles máximos perdibles (ug, m3)	Tiempo de exposición
PM10	50	Anual
	10	24 horas
PM25	25	Anual
	50	24 horas
SO2	50	24 horas
	100	1 hora
NO2	60	Anual
	200	1 hora
O3	100	8 horas
CO	5.000	8 horas
	35.000	1 hora

Tabla 3. Niveles de contaminantes

6.1.3 Determinación de los objetivos del modelo

Criterios de éxito Urb@nEcoLife:

- Se necesita predecir el promedio de los siguientes tres meses del nivel de contaminación (PM10) de la ciudad de Cali.

6.2 Compresión de los datos

La fase de comprensión de los datos comprende la recolección inicial de datos. Esta fase tiene como objetivo establecer un primer contacto con el problema, familiarizándose con los datos, identificando su calidad y estableciendo relaciones más evidentes que permitan definir las primeras hipótesis para el proyecto Urb@nEcoLife.

Actividades:

- Recolección de datos iniciales.
- Descripción de datos.
- Exploración de datos.
- Verificar calidad de los datos.

6.2.1 Recolección de datos iniciales

Los niveles de contaminación históricos de la ciudad de Cali fueron entregados en un archivo por cada estación de monitoreo de la ciudad. En cada archivo se encuentra una pestaña por cada año de la información solicitada (2010 -2017). Esta información fue tramitada a través de un derecho de petición escrito soportado bajo el marco del proyecto. En las dos siguientes imágenes se muestra el número de archivos entregados y el resumen que tiene cada uno.

Ilustración 19. Archivos DAGMA

ALCALDÍA DE SANTIAGO DE CALI DEPARTAMENTO ADMINISTRATIVO DE GESTIÓN MEDIO AMBIENTE Grupo Calidad del Aire		Sistema de Vigilancia de Calidad del Aire de Santiago de Cali SVCASC **FORMATO DATOS VALIDOS (FINAL)** **ESTACION UNIVALLE**	SVCASC.FT.18	
			VERSIÓN	1
			FECHA DE APROBACIÓN	01/Mar/2016
Contenido		**Descripcion**		
0	Promedios Anuales	En esta hoja se encuentra el calculo de los promedios anuales		
1	2003 (ug-m3)	En esta hoja se encuentra los datos validos del año 2003 y la grafica de tendencia de los datos		
2	2004 (ug-m3)	En esta hoja se encuentra los datos validos del año 2004 y la grafica de tendencia de los datos		
3	2013 (ug-m3)	En esta hoja se encuentra los datos validos del año 2013 y la grafica de tendencia de los datos		
4	2014 (ug-m3)	En esta hoja se encuentra los datos validos del año 2014 y la grafica de tendencia de los datos		
5	2015 (ug-m3)	En esta hoja se encuentra los datos validos del año 2015 y la grafica de tendencia de los datos		
6	2016 (ug-m3)	En esta hoja se encuentra los datos validos del año 2016 y la grafica de tendencia de los datos		

Ilustración 20. Datos UNIVALLE

En el gráfico inferior se muestran los datos de crecimiento del parque automotor de la ciudad de Cali, los cuales fueron usados para entrenar el modelo.

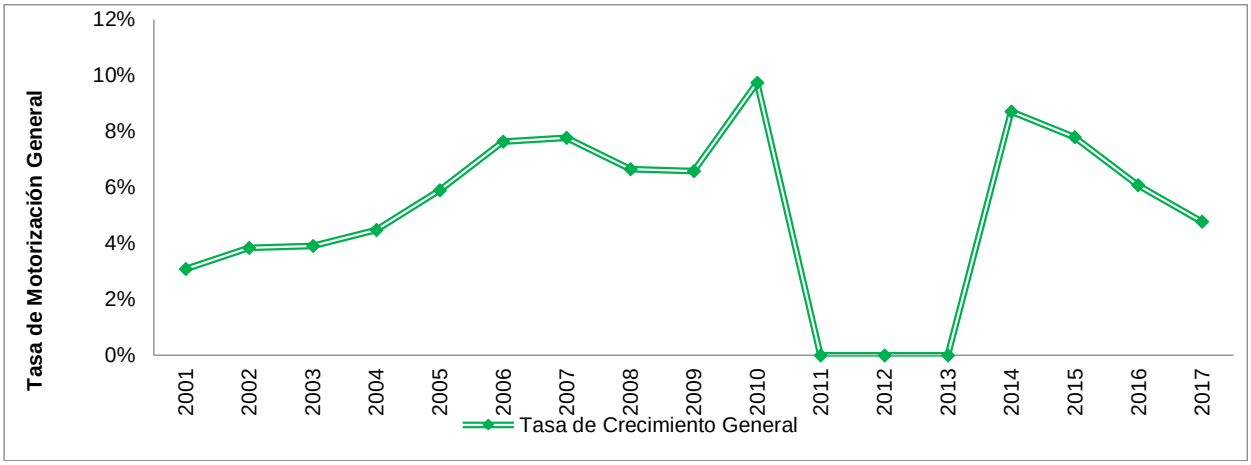
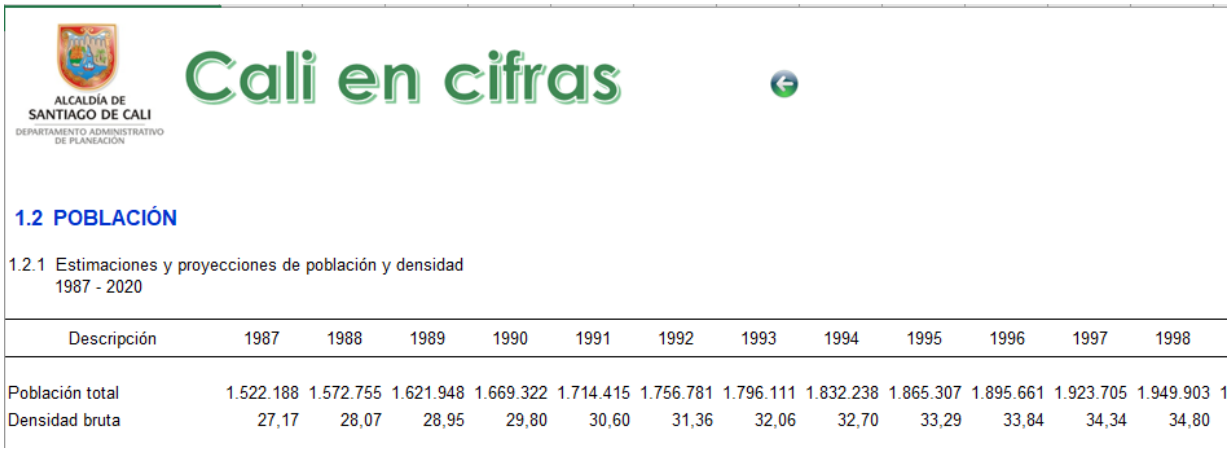


Ilustración 21. Tasa de crecimiento población

Cali cuenta con un portal donde se recopilan los datos más relevantes de la ciudad, expuestos para uso público. En este portal se encuentran los datos del censo poblacional de la ciudad, los cuales se muestran en la ilustración 22.



Descripción	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998
Población total	1.522.188	1.572.755	1.621.948	1.669.322	1.714.415	1.756.781	1.796.111	1.832.238	1.865.307	1.895.661	1.923.705	1.949.903 1
Densidad bruta	27,17	28,07	28,95	29,80	30,60	31,36	32,06	32,70	33,29	33,84	34,34	34,80

Ilustración 22. Cali en cifras

En la tabla inferior se presenta información sobre motos y vehículos particulares. Esta información fue tomada de los registros activos del Centro de Diagnóstico Automotor del Valle del Cauca para los años 2000 al 2017.

Año	Motos particulares	Vehículos particulares	Motos + Vehículos particulares	Tasa de Crecimiento General
2000	55.462	139.952	195.414	N/A
2001	56.045	145.394	201.439	3,1%
2002	56.498	152.672	209.170	3,8%
2003	57.038	160.323	217.361	3,9%
2004	58.267	168.837	227.104	4,5%
2005	60.679	179.831	240.510	5,9%
2006	64.309	194.554	258.863	7,6%
2007	66.650	212.359	279.009	7,8%
2008	69.871	227.705	297.576	6,7%
2009	77.256	239.913	317.169	6,6%
2010	90.693	257.373	348.066	9,7%
2011	111.422	279.905	391.327	12,4%
2012	138.262	300.507	438.769	12,1%
2013	162.626	321.493	484.119	10,3%
2014	181.174	345.165	526.339	8,7%
2015	196.934	370.419	567.353	7,8%
2016	206.592	395.202	601.794	6,1%
2017	211.538	418.940	630.478	4,8%

Tabla 4. Motos y vehículos particular

6.2.2 Exploración de datos

A continuación, se presenta de forma general los datos fuentes para el entrenamiento y validación del modelo.

- **Departamento Administrativo de Gestión del Medio Ambiente DAGMA** (Dagma, 2020)**:** El DAGMA cuenta con un Sistema de Vigilancia de la Calidad del Aire, que opera con nueve (9) estaciones automáticas. Estas estaciones se han venido fortaleciendo y actualizando continuamente con equipos nuevos para la medición de la contaminación del aire por emisión, tales como: Material particulado PM10 y PM2.5, Ozono troposférico, Dióxido de azufre, Dióxido de nitrógeno y condiciones meteorológicas (Resolucion N 4133.0.0 984 , Diciembre de 2013).

  En los datos provistos por el DAGMA desde enero del 2010 hasta diciembre del 2017, se incluye la medición de los diversos contaminantes, el material particulado y las condiciones del ambiente. Estos datos están contemplados en archivos de Excel (XLS) por cada estación de medición, es decir, en un

total de nueve archivos con los contaminantes captados por cada una de las estaciones de medición del aire ubicadas en la ciudad de Cali.

Campos	Unidad de Medida
Fecha & Hora	Personalizado
PM 10	$\mu g/m^3$
PM 2,5	$\mu g/m^3$
SO2	$\mu g/m^3$
NO2	$\mu g/m^3$
CO	$\mu g/m^3$
O3	$\mu g/m^3$
Velocidad del Viento	m/s
Dirección del Viento	Grados
Temperatura	Grados Centígrados
Humedad	Porcentaje
Radiación Solar	W/m^2
Lluvia	Milímetro

Tabla 5. Tipos de datos capturados por las estaciones de medición del aire en la ciudad de Cali

6.2.3 Verificación de calidad de datos

Todos los archivos .XLS suministrados por el DAGMA tienen una especificación de la calidad de datos en la pestaña "Promedios Anuales", la cual valida los siguientes puntos por cada una de las estaciones de la ciudad.

- Promedios Anuales.

Promedios Anuales												
Año	PM10 (ug/m3)	SO2 (ug/m3)	NO2 (ug/m3)	CO (ug/m3)	O3 (ug/m3)	H2S (ug/m3)	Vel Viento (m/s)	Dir Viento (Grados)	Temperatura (C°)	Humedad (%)	Radiacion Solar (Watt/M2)	Lluvia (mm)
2010	28,0	13,0	24,0	1423,8	21,3	-	0,8	160,3	24,0	75,0	155,2	0,4
2011	29,4	20,9	27,5	1459,8	26,0	-	0,8	168,2	24,4	72,1	151,2	0,4
2012	32,1	#¡DIV/0!	26,6	1879,5	#¡DIV/0!	-	0,8	167,5	25,1	68,3	160,7	0,3
2013	31,1	#¡DIV/0!	23,4	#¡DIV/0!	#¡DIV/0!	-	0,9	164,7	25,1	69,3	167,0	0,3
2014	49,5	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	-	0,9	162,3	25,2	68,9	167,4	0,3
2015	41,7	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	2,9	0,9	160,3	26,3	62,5	179,7	0,2
2016	33,4	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	20,0	2,5	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	0,2
2017	30,5	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	21,6	1,9	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	#¡DIV/0!	253,7	0,2

Ilustración 23. Promedios Flora

- Porcentaje de Datos Validos según el año.

Porcentaje de Datos Validos según el año												
Año	PM10 (ug/m3)	SO2 (ug/m3)	NO2 (ug/m3)	CO (ug/m3)	O3 (ug/m3)	H2S (ug/m3)	Vel Viento (m/s)	Dir Viento (Grados)	Temperatura (C°)	Humedad (%)	Radiacion Solar (Watt/M2)	Lluvia (mm)
2010	48,8%	52,8%	45,1%	47,2%	52,5%		56,0%	56,0%	56,0%	56,0%	54,1%	56,0%
2011	88,7%	86,8%	64,6%	50,4%	54,1%		91,9%	91,9%	91,9%	91,9%	91,9%	91,9%
2012	46,3%	0,0%	85,9%	18,9%	0,0%		89,7%	89,7%	89,7%	89,7%	89,7%	89,7%
2013	75,0%	0,0%	42,2%	0,0%	0,0%		94,4%	94,4%	94,4%	94,4%	94,4%	94,4%
2014	91,0%	0,0%	0,0%	0,0%	0,0%		92,6%	92,6%	92,6%	92,6%	92,6%	92,6%
2015	80,2%	0,0%	0,0%	0,0%	0,0%	51,1%	3,9%	3,9%	3,9%	3,9%	3,9%	89,5%
2016	92,9%	0,0%	0,0%	0,0%	21,7%	77,4%	0,0%	0,0%	0,0%	0,0%	0,0%	99,2%
2017	84,6%	0,0%	0,0%	0,0%	83,9%	87,5%	0,0%	0,0%	0,0%	0,0%	49,2%	89,5%

Ilustración 24. Datos Validos Flora

6.3 Fase de preparación de los datos

En esta fase se procesan las tres fuentes de información que se tienen disponibles en el proyecto Urb@nEcoLife y se desarrollan las siguientes actividades.

- Selección de datos.
- Limpieza de datos.
- Estructuración de datos.
- Integración datos.
- Formateo de datos.

El proceso en términos generales

6.3.1 Selección de datos

Se realizó la carga los datos de la estación la FLORA del DAGMA y se compilaron en único archivo.

```
Ruta="/Users/cristianchaparro/Documents/curso_data_science/Tesis/"

#Revision de los datos de la estacion la flora 2010-2017.

flora2010 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2010 (ug-m3)', header=0)
flora2011 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2011 (ug-m3)', header=0)
flora2012 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2012 (ug-m3)', header=0)
flora2013 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2013 (ug-m3)', header=0)
flora2014 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2014 (ug-m3)', header=0)
flora2015 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2015 (ug-m3)', header=0)
flora2016 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2016 (ug-m3)', header=0)
flora2017 = pd.read_excel(Ruta+'Originales/1.LAFLORA.xls',sheet_name='2017 (ug-m3)', header=0)

#Unifico todos los datos en solo archivo.
floraTotal=pd.concat([flora2010,flora2011,flora2012,flora2013,flora2014,flora2015,flora2016,flora2017], ignore_index=Tr
```

Ilustración 25. Agrupación Flora

Se realizo el cargue de la información sobre motos y vehículos particulares de Diagnóstico Automotor del Valle.

```
#Datos La información sobre motos y vehículos particulares de Diagnóstico Automotor del Valle

Tasa_de_motorizacion_general = pd.read_excel(Ruta+'Originales/Tasa_de_motorizacion_general.xlsx',sheet_name='Tasa_de_motorizacior
Tasa_de_motorizacion_general=Tasa_de_motorizacion_general.rename(columns={
  "Año":"Año",
    "Motos particulares":"Motos_particulares",
    "Vehículos particulares":"Vehiculos_particulares",
    "Motos + Vehículos particulares":"Motos_vehiculos_particulares",
    "Tasa de Crecimiento General":"Tasa_crecimiento_general"
})

Tasa_de_motorizacion_general.head(19)
```

Ilustración 26 Motos y vehículos particulares

Se realizo el cargue de la información de la población de la ciudad de Cali.

```
In [3]: #Datos de la poblacion de la ciudad de cali.

        Poblacion_cali = pd.read_excel(Ruta+'Originales/1_Generalidades.xls',sheet_name='C1.2.1', header=0)

        Poblacion_cali.head(19)
```

Ilustración 27 Población de la ciudad de cali

Se realizo el cargue de la información del producto interno bruto.

```
#Cargo la informacion del PIB

PIB_total_por_departamento= pd.read_excel(Ruta+'Originales/2018-provisional-PIB-total-por-departamento.xlsx',sheet_name='Cuadro 3
```

Ilustración 28 Carga de PIB

6.3.2 Estructuración de datos

Se unifico los datos de la estación de la FLORA con la población, PIB y el parque automotor para los años del 2010 – 2017.

```
#Agrupo los datos de contaminacion y el total de vehiculos.
flora2010['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[10,3]
flora2011['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[11,3]
flora2012['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[12,3]
flora2013['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[13,3]
flora2014['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[14,3]
flora2015['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[15,3]
flora2016['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[16,3]
flora2017['Motos_vehiculos_particulares']=Tasa_de_motorizacion_general.iloc[17,3]

#Agrupo los datos de contaminacion y la poblacion.
flora2010['Poblacion']=Poblacion_cali.iloc[15,24]
flora2011['Poblacion']=Poblacion_cali.iloc[15,25]
flora2012['Poblacion']=Poblacion_cali.iloc[15,26]
flora2013['Poblacion']=Poblacion_cali.iloc[15,27]
flora2014['Poblacion']=Poblacion_cali.iloc[15,28]
flora2015['Poblacion']=Poblacion_cali.iloc[15,29]
flora2016['Poblacion']=Poblacion_cali.iloc[15,30]
flora2017['Poblacion']=Poblacion_cali.iloc[15,31]
```

Ilustración 29 Flora +Población + vehiculos

Se consolidó toda la información bajo la siguiente estructura:

```
floraTotal=floraTotal.rename(columns={
  "Fecha & Hora":"Fecha",
    "H2S (ug/m3)":"H2S",
  "PM10 (ug/m3)":"PM10",
    "SO2  (ug/m3)":"SO2",
    "NO2  (ug/m3)":"NO2",
    "CO  (ug/m3)":"CO",
    "O3  (ug/m3)":"O3",
    "Vel Viento  (m/s)":"VelVien",
    "Dir Viento (Grados)":"DirVien",
    "Temperatura (C°)":"Temp",
    "Humedad (%)":"Humedad",
    "Radiacion Solar (Watt/M2)":"RSolar",
    "Lluvia (mm)":"Lluvia"
})
```

Ilustración 30. Estructuración Flora

6.3.3 Limpieza de datos

Se realizó la limpieza de datos, eliminado las columnas sin datos y cambiando los N/A por cero en todas las columnas.

```
floraTotal = floraTotal.drop(['Fecha'], axis=1, inplace=False)#Elimino la columna fecha
floraTotal=floraTotal.dropna(how='all')#Elimino la columnas que solo tienes nulos.
# Cambio los N/A por cero
floraTotal['H2S'].fillna(0, inplace=True)
floraTotal['PM10'].fillna(0, inplace=True)
floraTotal['SO2'].fillna(0, inplace=True)
floraTotal['NO2'].fillna(0, inplace=True)
floraTotal['CO'].fillna(0, inplace=True)
floraTotal['O3'].fillna(0, inplace=True)
floraTotal['VelVien'].fillna(0, inplace=True)
floraTotal['DirVien'].fillna(0, inplace=True)
floraTotal['Temp'].fillna(0, inplace=True)
floraTotal['Humedad'].fillna(0, inplace=True)
floraTotal['RSolar'].fillna(0, inplace=True)
floraTotal['Lluvia'].fillna(0, inplace=True)
```

Ilustración 31. Limpieza Flora

Un valor atípico (outlier) es una observación que es numéricamente distante del resto de los datos, con la función “outliers_col” se calculó el número de valores atípicos por columna.

```
def outliers_col(df):
    for columna in df:
        if df[columna].dtype != np.object:
            n_outliers =len(df[np.abs(stats.zscore(df[columna]))>3])
            print ("{}|{}|{}".format(
                df[columna].name,
                n_outliers,
                df[columna].dtype
            ))
```

Ilustración 33. Función outliers

```
: outliers_col(floraTotal)

CO|1750|float32
DirVien|5|float32
H2S|3|float32
Humedad|6|float32
Lluvia|8|float32
NO2|4|float32
O3|67|float32
PM10|8|float32
RSolar|463|float32
SO2|126|float32
Temp|6|float32
VelVien|6|float32
```

Ilustración 32. Aplicación outliers

Con la función “detect_outlier” se sacó una lista de estos valores para su posterior corrección.

```
outliers=[]
def detect_outlier(data_1):

    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)
    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

Ilustración 34. Ajuste outliers

Con la función “cambiar_outlier”: Se calculo la media corta al 30% de los valores de X, se buscaron los datos atípicos y se reemplazaron por la mediana corta.

```
            ))
#Busco los valores fuera de la distrucion y los cambio por media corta a 30%
def cambiar_outlier(data_1):
    threshold=3
    mean_1 = np.mean(data_1)
    mean_30 = stats.trim_mean(data_1, 0.3);
    std_1 =np.std(data_1)
    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            data_1=data_1.replace(y,mean_30)
    return data_1
```

Ilustración 35. Cambiar autlier

Se aplicó la función “cambiar_outlier” en todas las columnas y se generó el archivo final procesado.

```
In [59]: floraTotal['H2S']=cambiar_outlier(floraTotal['H2S'])
         floraTotal['NO2']=cambiar_outlier(floraTotal['NO2'])
         floraTotal['Temp']=cambiar_outlier(floraTotal['Temp'])
         floraTotal['O3']=cambiar_outlier(floraTotal['O3'])
         floraTotal['PM10']=cambiar_outlier(floraTotal['PM10'])
         floraTotal['SO2']=cambiar_outlier(floraTotal['SO2'])
         floraTotal['DirVien']=cambiar_outlier(floraTotal['DirVien'])
         floraTotal['H2S']=cambiar_outlier(floraTotal['H2S'])
         floraTotal['Humedad']=cambiar_outlier(floraTotal['Humedad'])
         floraTotal['Lluvia']=cambiar_outlier(floraTotal['Lluvia'])
         floraTotal['RSolar']=cambiar_outlier(floraTotal['RSolar'])
         floraTotal['VelVien']=cambiar_outlier(floraTotal['VelVien'])
         floraTotal['CO']=cambiar_outlier(floraTotal['CO'])
```

Ilustración 36. Aplicación outliers

6.3.4 Formateo de datos

Se agrego una columna que representa el número de N/A por cada registro, en esta agrupación se encuentran 70.152 registros.

```
In [17]: #Crea una columna con los numero de nulos.
         floraTotal['N_NA']=floraTotal.isnull().sum(axis=1)

In [32]: resultado=collections.Counter(floraTotal.N_NA)

In [38]: print(resultado.keys())
         print(resultado.values())

         dict_keys([12, 9, 7, 6, 5, 3, 4, 10, 2, 11, 1, 8, 0])
         dict_values([8216, 9645, 3922, 1968, 15283, 4723, 6785, 5481, 3498, 447, 5241, 4937, 6])
```

Ilustración 37 Nulos por registro

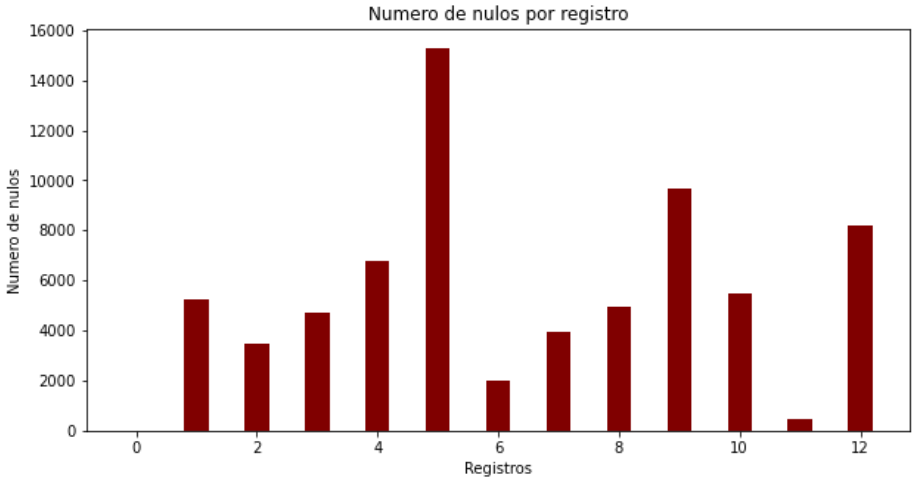


Ilustración 38Numero de nulos por fila

Se procede a borrar los registros cuya cantidad de N/A son menores a 3, después de eliminar quedan un total de 8.745 registros.

```
In [5]:
         #Solo dejo los registros con menos de 3 nulos.
         floraTotal=floraTotal[floraTotal.N_NA<3]

         #Borro la columna temporal.
         floraTotal=floraTotal.drop(['N_NA'], axis=1)
         floraTotal=floraTotal.drop(['Fecha'], axis=1)
```

Ilustración 39Borrar registros N/A

Con la función “is_float” se formateó todos los datos, algunas de las columnas tenían datos corruptos estos se remplazaron por 0.

```
#Funcion que valida si dato es un float,
#Si no es float lo vuelve 0 para que genere problemas con convercion de formato
def is_float(value):
  try:
    value= float("{0:.2f}".format(value))
    return value
  except:
    return 0.0
```

Ilustración 40. Función is float

Se aplicó la función “is_float” a todas las columnas para limpiar y formatear los datos.

```
#Aplico la funcion is_float en cada unos de la columnas.
floraTotal['H2S'] = floraTotal.apply(lambda x:is_float(x['H2S']), axis=1)
floraTotal['NO2'] = floraTotal.apply(lambda x:is_float(x['NO2']), axis=1)
floraTotal['O3'] = floraTotal.apply(lambda x:is_float(x['O3']), axis=1)
floraTotal['PM10'] = floraTotal.apply(lambda x:is_float(x['PM10']), axis=1)
floraTotal['SO2'] = floraTotal.apply(lambda x:is_float(x['SO2']), axis=1)
floraTotal['CO'] = floraTotal.apply(lambda x:is_float(x['CO']), axis=1)
floraTotal['DirVien'] = floraTotal.apply(lambda x:is_float(x['DirVien']), axis=1)
floraTotal['Humedad'] = floraTotal.apply(lambda x:is_float(x['Humedad']), axis=1)
floraTotal['Lluvia'] = floraTotal.apply(lambda x:is_float(x['Lluvia']), axis=1)
floraTotal['RSolar'] = floraTotal.apply(lambda x:is_float(x['RSolar']), axis=1)
floraTotal['Temp'] = floraTotal.apply(lambda x:is_float(x['Temp']), axis=1)
floraTotal['VelVien'] = floraTotal.apply(lambda x:is_float(x['VelVien']), axis=1)
```

Ilustración 41. Aplicación función float

Luego de realizar el formateo, se procedió a asignar el tipo de dato de cada una de las columnas.

```
#Vuelvo float32 todas las columnas.

data['H2S'] = data[['H2S']].astype('float32')
data['NO2'] = data['NO2'].astype('float32')
data['O3'] = data['O3'].astype('float32')
data['PM10'] = data['PM10'].astype('float32')
data['SO2'] = data['SO2'].astype('float32')
data['DirVien'] = data['DirVien'].astype('float32')
data['H2S'] = data['H2S'].astype('float32')
data['Humedad'] = data['Humedad'].astype('float32')
data['Lluvia'] = data['Lluvia'].astype('float32')
data['RSolar'] = data['RSolar'].astype('float32')
data['Temp'] = data['Temp'].astype('float32')
data['VelVien'] = data['VelVien'].astype('float32')
data['CO'] =  data['CO'].astype('float32')
```

Ilustración 42. Formateo de columna

6.3.5 Integración de datos

Para el desarrollo de los modelos se integraron los datos de diferentes formas, a contaminación se muestran las características usadas, el total de datos y el rango de valor de las características.

Modelo Base

La integración básica es donde se encuentra las 12 características de la estación de la FLORA, con un total de 8.745 registros donde cada registro representa un día de contaminación.

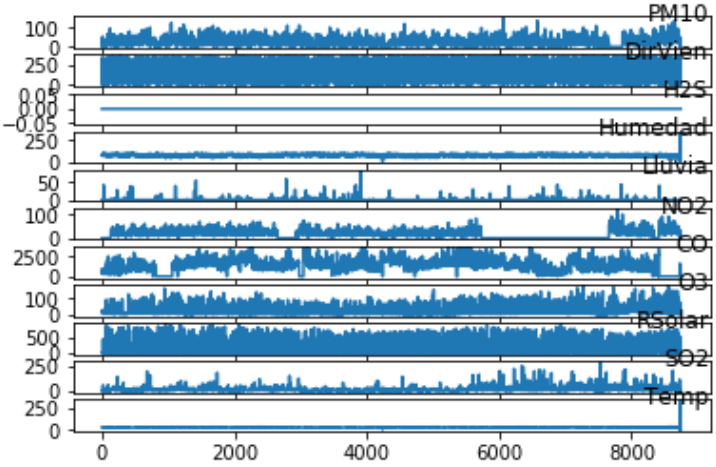
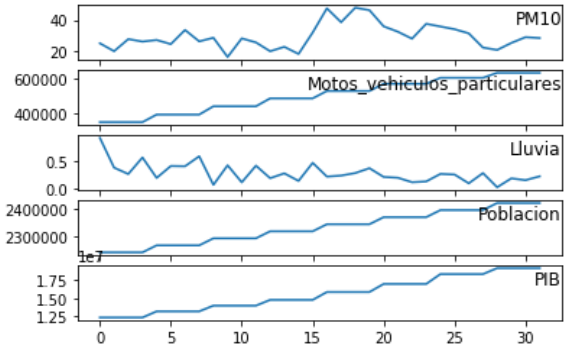


Ilustración 43Flora total características

Modelo 3 meses

La integración para modelos de 3 meses es donde se encuentran 5 características, con un total de 32 registros, donde cada registro representa un trimestre de contaminación de la ciudad de Cali.



6.4 Modelamiento

6.4.1 Modelamiento estructura general

Para el modelado y evaluación se decidió partir los datos en dos bloques, el bloque de entrenamiento y el bloque de evaluación. Posteriormente se realiza una validación cruzada entre lo que se predijo con el primer bloque y lo que se guardó en el segundo bloque de prueba.

Validación cruzada

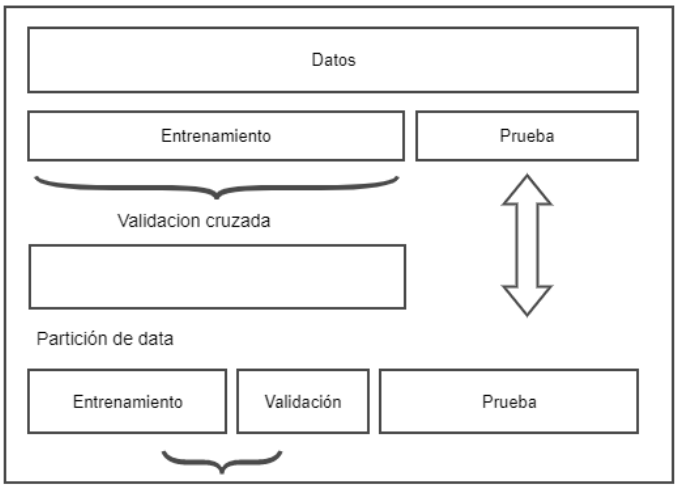


Ilustración 44. Partición de data

Para el desarrollo del modelo base se dividió en dos partes el total de datos de la estación de la FLORA (8.745). Con la parte (4.000) datos se entrenó el modelo y con el resto de los datos (4.744).

```
# Datos de entrada y de salida.
train_X, train_y = train[:, :-1], train[:, -1:]
test_X, test_y = test[:, :-1], test[:, -1:]


#Remodelar la entrada para que sea 3D [muestras, pasos de tiempo, características]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(4000, 1, 11) (4000, 1) (4744, 1, 11) (4744, 1)
```

Ilustración 45. Separación flora modelo base

Para el desarrollo del modelo de 3 meses se dividió en dos partes el total de datos (31). Con la parte (23) datos se entrenó el modelo y con el resto de los datos (8).

```
In [77]: values = reframed.values

         #Datos de entrenamiento y pruebas:
         n_train_hours = 23
         train = values[:n_train_hours, :]
         test = values[n_train_hours:, :]
```

Ilustración 46 Separación modelo 3 meses

El primer paso del modelado LSTM es enmarcar el problema en estructura de serie de tiempo. Con la estructura de serie de tiempo, se realiza el entrenamiento de modelo bajo un esquema de aprendizaje supervisado, para esto se usó la función "series_to_supervised" que tiene la siguiente estructura:

```
In [44]: def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
             n_vars = 1 if type(data) is list else data.shape[1]
             df = DataFrame(data)
             cols, names = list(), list()
         # observaciones pasadas  (t-n, ... t-1)
             for i in range(n_in, 0, -1):
                 cols.append(df.shift(i))
                 names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
         # tiempos futuros  (t, t+1, ... t+n)
             for i in range(0, n_out):
                 cols.append(df.shift(-i))
                 if i == 0:
                     names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                 else:
                     names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
             agg = concat(cols, axis=1)
             agg.columns = names
             if dropnan:
                 agg.dropna(inplace=True)
             return agg
```

Ilustración 47. Funciones serie de tiempo

La función toma cuatro argumentos:

- **datos**: Secuencia de observaciones como una lista o matriz NumPy 2D. Obligatorio.
- **n_in**: Número de observaciones de retraso como entrada (*X*). Los valores pueden estar entre [1...len(datos)] Opcional. El valor predeterminado es 1.
- **n_out**: Número de observaciones como salida (*y*). Los valores pueden estar entre [0..l en (datos) - 1]. Opcional. El valor predeterminado es 1.
- **dropnan**: Boolean si se eliminan o no las filas con valores de NaN. Opcional (El valor predeterminado es True).

La función devuelve un solo valor:

- **retorno**: Pandas DataFrame de series enmarcadas para aprendizaje supervisado.

Técnicamente, en la terminología de pronóstico de series de tiempo, el tiempo actual *(t)* y los tiempos futuros *(t + 1, t + n)* son tiempos pronosticados y las observaciones pasadas *(t-1, t- n),* se usan para hacer

pronósticos. Existen diferentes nombres para el problema dependiendo de la cantidad de pasos de tiempo para pronosticar:

- **Pronóstico de un paso**: Se predice el siguiente paso de tiempo (t + 1).
- **Pronóstico de pasos múltiples**: Se predice dos o más pasos de tiempo futuros.

Luego de tener los datos particionados, se procede a diseñar la red neuronal recurrente (LSTM), con cincuena (50) neuronas en la primera capa oculta y 1 neurona en la capa de salida para predecir la contaminación.

La forma de entrada será 1 paso de tiempo con doce (12) características. Se usa la función de pérdida de error absoluto medio (MAE) y la versión eficiente de Adam del descenso de gradiente estocástico (keras.io, s.f.) para realizar la compilación de este.

```
#Diseñando la red neuronal.
model = Sequential()
model.add(LSTM(50, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
```

Ilustración 48. Diseño de red neurona

El modelo se ajusta a cincuenta (50) épocas de entrenamiento con un tamaño de lote de setenta y dos (72), una época es un uso completo de todo el conjunto de entrenamiento. Es decir, durante cincuenta (50) épocas, se hace iteraciones aleatoriamente sobre todo el conjunto de entrenamiento cincuenta (50) veces.

```
# Ajustado el modelo
history = model.fit(x_train, y_train, epochs=50, batch_size=72,
                    validation_data=(x_test, y_test), verbose=2, shuffle=False)
```

Ilustración 49. Ajuste modelo

Ahora que el modelo se ha definido completamente, con su función de pérdida y su optimizador, será posible su entrenamiento, el cual puede realizarse con la función fit. Esta función toma matrices numpy y realiza un número dado de épocas (epochs) de entrenamiento utilizando el tamaño de lote de datos suministrado.

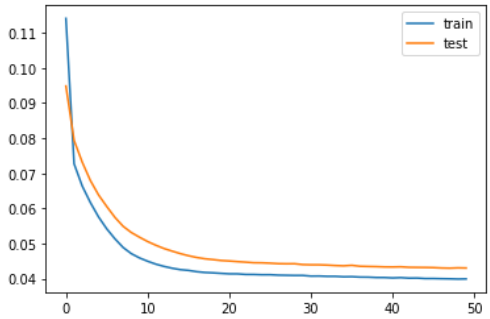


Ilustración 50. Entrenamiento del modelo

6.4.2 Modelo base

Dada las características del proyecto se desarrollaron varios modelos, en términos generales las variables con las que se trabajaron fueron las siguientes:

- **Datos:** Cantidad de registros usadas para entregar y validar el modelo.
- **Numero de características de entrada:** Se ajusta el número de características de entrada del modelo (T-1).
- **Numero de características de salida:** Se ajusta el número de características de salida del modelo (T).
- **Numero de paso de tiempo(N):** número de paso de tiempo entre las características (T-N).
- **Unidad de media:** La unidad de medida se puede trabajar agrupando una cantidad de datos (meses, trimestres, años).

El modelo base se desarrolló con las siguientes características:

- **Datos:** 8.745 registros de la ciudad de Cali estación la FLORA
- **Numero de características de entrada:** Las doce (12) características (sensores FLORA).
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** Horas

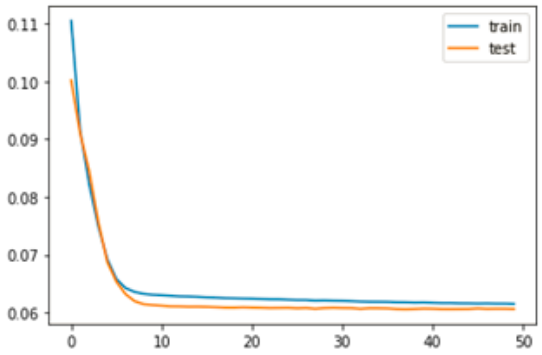


Ilustración 51. Ajuste modelo base

RSME: 12.880

6.4.3 Modelo con el tamaño del parque automovilístico y las 12 características

Se cambia la unidad de medida, se agrega una nueva característica de entrada y se genera el siguiente modelo:

- **Datos:** 8.745 registros de la ciudad de Cali estación la FLORA.
- **Numero de características de entrada:** Las doce (12) características + parque automovilístico (sensores FLORA).
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** Horas

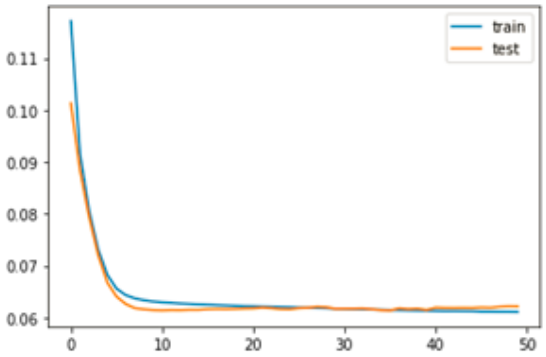


Ilustración 52. Ajuste modelo 2

RSME: 13.087

6.4.4 Modelo con la población y las 12 características

Se cambia la unidad de medida, se agrega una nueva característica en la entrada y se genera el siguiente modelo:

- **Datos:** 8.745 registros de la ciudad de Cali estación la FLORA.
- **Numero de características de entrada:** Las doce (12) características + población (sensores FLORA).
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** Horas

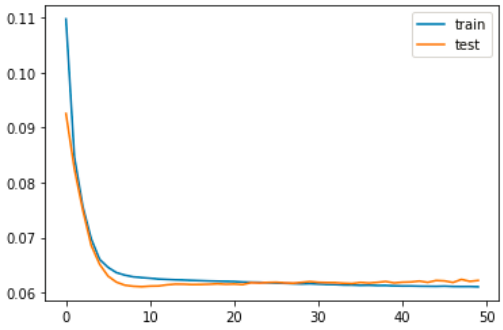


Ilustración 53. Ajuste modelo 3

RSME: 13.149

6.4.5 Modelo con el parque automovilístico, población y las 12 características

Se cambia la unidad de medida, se agrega dos nuevas características en la entrada del modelo:

- **Datos:** 8.745 registros de la ciudad de Cali estación la FLORA.
- **Numero de características de entrada:** Las doce (12) características + población + parque automovilístico (sensores FLORA).
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** Horas

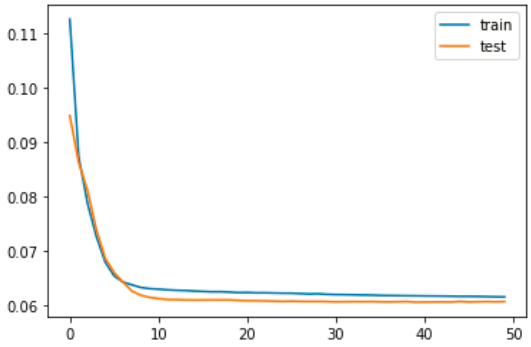


Ilustración 54. Ajuste modelo 13 características

RSME: 32.658

6.4.6 Modelo de 3 meses

Dado que la data no era la mejor, se definió un modelo diferente con cuatro (4) características que son las que mejor calidad tienen y se cambió la agrupación de datos por tres (3) meses.

- **Datos:** 32 registros donde el PM10, es el promedio de datos de contaminación de la Cali para los años 2010 -2017.
- **Numero de características de entrada:** Lluvia + población + parque automovilístico (sensores FLORA).
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** 3 meses

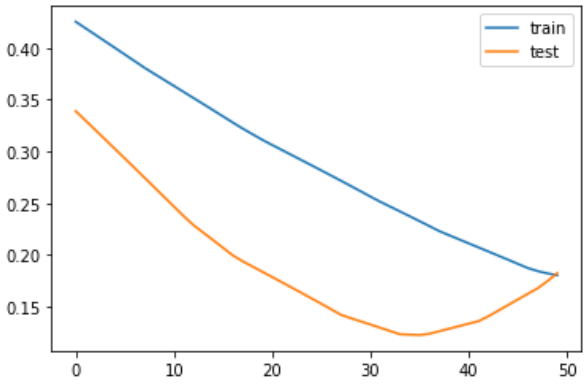


Ilustración 55. Ajuste modelo3 meses

RSME: 6.824

6.4.7 Modelo de 3 meses + PIB

En este modelo se usaron cinco (5) características, las cuales fueron las que mejor calidad de data tenían y se cambió la unidad de medida a tres (3) meses.

- **Datos**: 32 registros donde el PM10, es el promedio de datos de contaminación de la Cali para los años 2010 -2017.
- **Numero de características de entrada:** Lluvia + población + parque automovilístico (sensores FLORA) + PIB.
- **Numero de características de salida:** PM10 en T.
- **Numero de paso de tiempo:**1
- **Unidad de media:** 3 meses

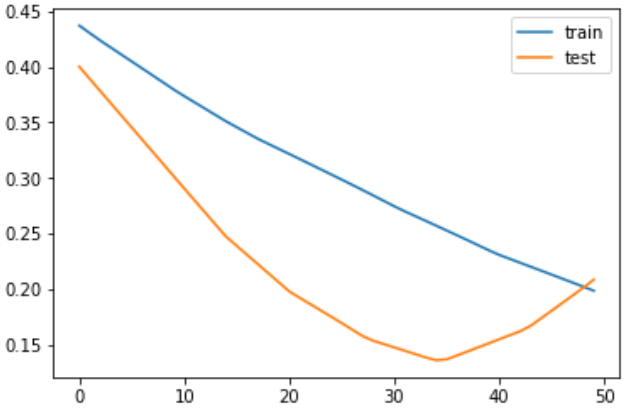


Ilustración 56. Ajuste modelo 3 meses + PIB

RSME: 6.894

7 EVALUACIÓN E INTEGRACION DEL MODELO

Para realizar el entrenamiento del modelo basado en *Deep Learning* para los datos de la plataforma Urb@nEcoLife se siguió la metodología CRISP- DM, en este capítulo se van a aplicar las 2 últimas fases de la metodología:

- Evaluación modelo.
- Integración modelo.

7.1 Evaluación del modelo

Para la realización de la evaluación del modelo se usaron las siguientes medidas.

Error absoluto medio (MAE): La media de las diferencias entre la variable objetivo y las predicciones sin el signo. Esta media se interpreta como unidades de la variable objetivo, en este caso la contaminación ambiental.

Error cuadrático medio (MSE): Funciona igual que MAE, pero la diferencia entre la variable objetivo y las predicciones se elevan al cuadrado.

Raíz del error cuadrático medio (RMSE): Es la raíz de MSE.

```
[4]: def rmse(objetivo, estimaciones):
         return np.sqrt(metrics.mean_squared_error(objetivo, estimaciones)
                        )
```

Ilustración 57Error cuadrático medio

CASE 1: Evenly distributed errors

ID	Error	|Error|	Error^2
1	2	2	4
2	2	2	4
3	2	2	4
4	2	2	4
5	2	2	4
6	2	2	4
7	2	2	4
8	2	2	4
9	2	2	4
10	2	2	4

MAE	RMSE
2.000	2.000

CASE 2: Small variance in errors

ID	Error	|Error|	Error^2
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	3	3	9
7	3	3	9
8	3	3	9
9	3	3	9
10	3	3	9

MAE	RMSE
2.000	2.236

CASE 3: Large error outlier

ID	Error	|Error|	Error^2
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	20	20	400

MAE	RMSE
2.000	6.325

Ilustración 58. MAE y RMSE (medium, 2020)

El coeficiente de determinación: Determina la calidad del modelo para replicar los resultados y la proporción de variación de los resultados que pueden explicarse por el modelo.

```
def  ajustested_r2(objetivo, estimaciones,n,k):
    r2=metrics.r2_score(objetico,estimaciones)
    return 1-(1-r2)*(n-1)/(n-k-1)
```

Ilustración 59 Función ajuste r2

```
def evaluar_modelo(objetivo,estimaciones,n,k):
    return {
        "rmse":rmse(objetivo,estimaciones),
        "mae": metrics.mean_absolute_error(objetivo,estimaciones),
        "ajustested_r2": ajustested_r2(objetivo,estimaciones,n,k)
    }
```

Ilustración 60. Resumen del modelo

En la Ilustración 55, se compara la predicción generada por el modelo (3 meses) VS la data original de los treinta y dos (32) trimestres usados para el entrenamiento y validación de este.

Modelo 3 meses:

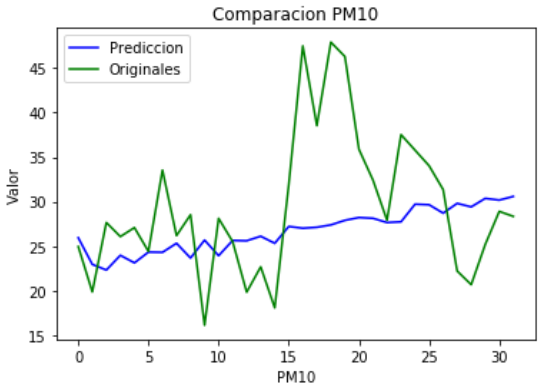


Ilustración 61. Modelo 3 meses

RSME: 8.116022

En la gráfica inferior se compara la predicción generada por el modelo (3 meses + PIB) VS la data original de los treinta y dos (32) trimestres usados para el entrenamiento y validación de este.

Modelo 3 meses + PIB:

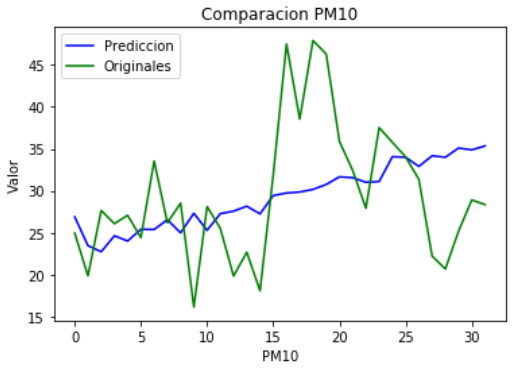


Ilustración 62. Modelo meses + PIB

RSME: 7.85167

Del modelo (3 meses + PIB) se generaron las siguientes predicciones:

En las siguientes predicciones se busca ver el comportamiento de las series de tiempo en (PM10) al agregar un dato sintético al final, al modelo se va a ingresar 8 registros de los cuales 7 son datos originales de los datos de la estación de la FLORA y 1 es un registro sintético que solo se va a modificar PIB, lluvia, población y parque automotor.

- En el eje X de la graficas se muestra el valor del PM 10 por cada registro.
- En el eje y se muestra los 8 trimestres.
- En las gráficas se resalta la última predicción es la muestra el comportamiento del modelo.

En la siguiente gráfica se muestra el cambio de la serie de tiempo al agregar un dato sintético al final. El dato agregado al final en este caso es el 50% del PIB del último trimestre, el cual muestra una baja en el PM10 en el dato 8 (sintético).

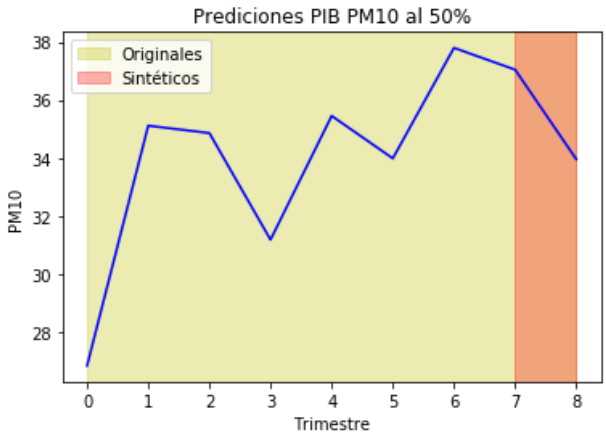


Ilustración 63. Predicción PIB PM10 50%

En la siguiente gráfica se muestra el cambio de la serie de tiempo al agregar un dato sintético al final. El dato agregado al final en este caso es el 50% de lluvia del último trimestre, el cual muestra una baja en el PM10 en el dato 8 (sintético).

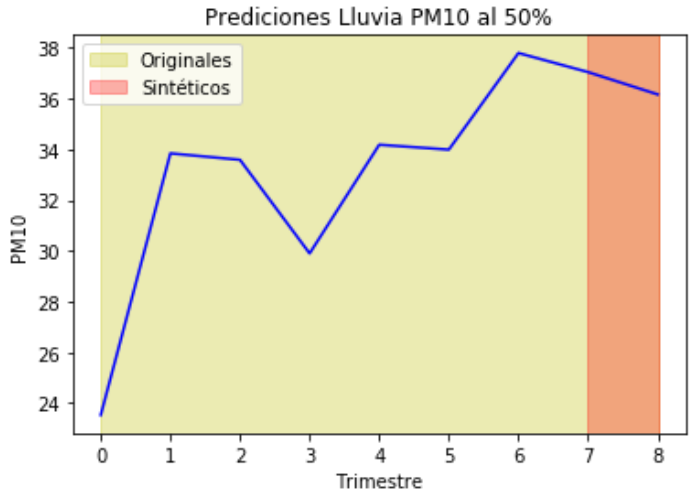


Ilustración 64. Predicción lluvia PM10 50%

En la siguiente gráfica se muestra el cambio de la serie de tiempo al agregar un dato sintético al final. El dato agregado al final en este caso es el 50% de población del último trimestre, el cual muestra una baja en PM10 en el dato 8 (sintético).

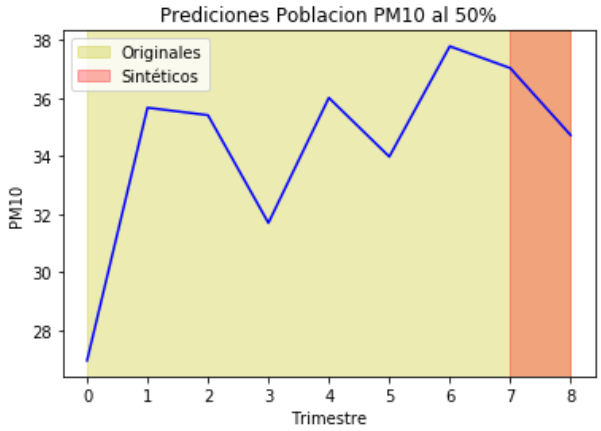


Ilustración 65. Predicción población PM10 50%

En la siguiente gráfica se muestra el cambio de la serie de tiempo al agregar un dato sintético al final. El dato agregado al final en este caso es el 50% de parque automotor del último trimestre, el cual muestra una baja en PM10 en el dato 8 (sintético).

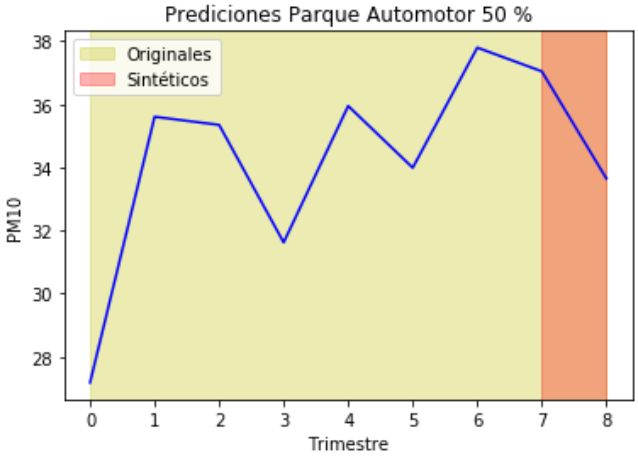


Ilustración 66. Predicción parque automotor PM10 50%

En las 4 predicciones se muestra que el PM10 disminuye en el trimestre final, estos nos permiten concluir que estas variables afectan el comportamiento de PM10 en la ciudad de Cali.

7.2 Integración del modelo

Dado que el desarrollo de este modelo hace parte del macroproyecto Urb@nEcoLife, la integración se realizó mediante la explosión de servicios REST con los siguientes métodos.

- Cargar el modelo.
- Generar predicción.

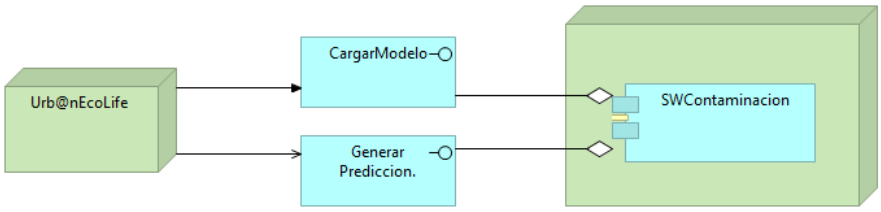


Ilustración 67. Arquitectura integración

El código fuente se encuentra en un repositorio en la nube, separada por cada una de las fases del desarrollo de la metodología del proyecto.

https://github.com/cchaparro/LTSM_Cali

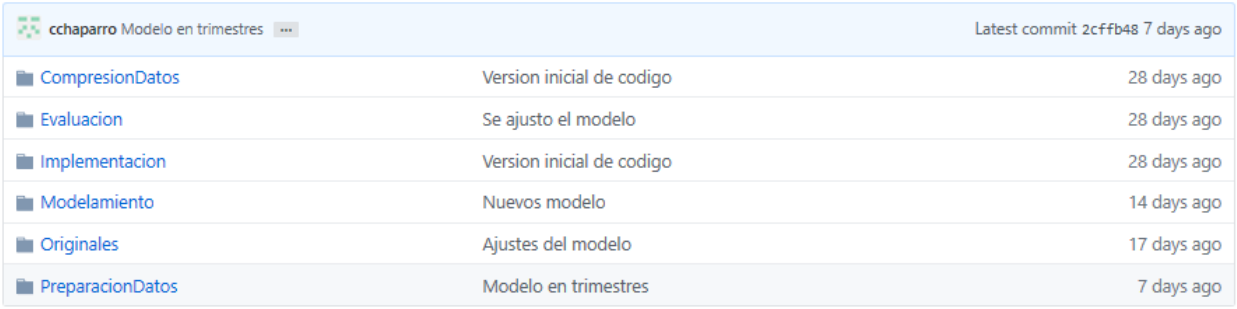


Ilustración 68. Código fuente

8 CONCLUSIONES Y TRABAJOS FUTUROS

La contaminación ambiental es un tema muy trabajado en el ámbito académico, ya que es uno de los mayores problemas de la actualidad. En las principales ciudades del mundo existen sensores que ayudan a medir la contaminación ambiental, pero esto no es más que una forma de detectar los problemas de contaminación cuando ya se tienen.

Puntualmente en la ciudad de Cali, la calidad de los datos entregada para el desarrollo del proyecto de grado no era la mejor, ya que existían zonas de tiempo donde no se encontraba información. Algunas características de entrada tenían gran cantidad de números en NaN y esto limitaba mucho la posibilidad de generar unas buenas predicciones. Sin datos de buena calidad es difícil realizar un buen entrenamiento y validar los resultados generados.

En el proyecto se desarrolló diferentes tipos de modelos, los primeros modelos se procesaban todas las características de la estación de la FLORA en una unidad de medida de una hora, dada la cantidad de registros con valores en NaN, se determinó borrar gran cantidad de registros completos, esto limito mucho la capacidad de generar predicciones acertadas, ya que era muy difícil asegurara la secuencia de los datos. Por recomendación de un asesor externo se determinó agrupar la información de la FLORA en trimestres (32) para los 7 años de información, se empezaron agregar características adicionales que permitieron generar modelos más precisos y se exploraron predicciones cuyo objetivo era ver el comportamiento de las series de tiempo agregando datos simulados al final de esta.

Lo que se pudo concluir del trabajo realizado dentro de este proyecto de grado, es que los casos de contaminación ambiental en términos generales son un perfecto escenario para trabajar con arquitecturas de Deep Learning, ya que existe una gran cantidad de variables o características que podrían afectar la calidad del aire en las ciudades. Además, es importante anotar que todas las características de entrada de los modelos realizados manifiestan patrones repetidos al convertirlas en series de tiempos. Así mismo, se pudo concluir que la ciudad de Cali es menos propensa a tener problemas de contaminación por la velocidad promedio del aire y su distribución geográfica.

La mejor forma de generar predicciones que generen un valor real es entendiendo el comportamiento de la ciudad o del sector en que se va a trabajar, esto permite identificar las características que son importantes para el entrenamiento del modelo, además estructurar de forma adecuada las series de tiempo que expresen de mejor forma las necesidades a explorar.

Este trabajo fue un reto personal y académico, ya que para su desarrollo se tuvo que abordar y aprender sobre técnicas de Machine Learning, así como de implementaciones que simplifican y abstraen un poco la complejidad de las diferentes arquitecturas de aprendizaje profundo o Deep Learning.

Para los trabajos futuros es importante buscar la forma de mejorar la calidad de los datos, se podría solicitar apoyo del DAGMA para generar una data con mayor calidad para periodos más cortos, ya que en algunos trabajos relacionados se revisaron predicciones en términos de días u horas.

En este trabajo se introdujeron nuevas características de entrada que generaron nuevos modelos de predicción, los cuales se podrían entrenar y generar predicciones con la información de los primeros tres meses de la pandemia de COVID 19 en Cali, lo que permitiría hacer una validación de la importancia de esas nuevas características en la contaminación ambiental de la ciudad.

9 REFERENCIAS BIBLIOGRÁFICA

Caffe. (2020). Obtenido de http://caffe.berkeleyvision.org/

colah.github.io. (2020). Obtenido de Understanding LSTM Networks: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Courville, I. G. (16 de 12 de 2016). *Deeplearningbook*. Obtenido de MIT Press: http://www.deeplearningbook.org

crisp-dm.pdf. (23 de 03 de 2000). *the-modeling-agency.* Obtenido de SPSS: https://www.the-modeling-agency.com/crisp-dm.pdf

cv-trick. (2020). Obtenido de https://cv-tricks.com/deep-learning-2/tensorflow-or-pytorch/

Dagma. (2020). Obtenido de http://www.cali.gov.co/dagma/publicaciones/38365/sistema_de_vigilancia_de_calidad_del_aire_de_cali_svcac/

DANE. (2020). *https://www.dane.gov.co.* Obtenido de https://www.dane.gov.co: https://www.dane.gov.co/index.php/estadisticas-por-tema/demografia-y-poblacion/censo-nacional-de-poblacion-y-vivenda-2018/donde-estamos

Deeplearning4. (2020). Obtenido de Deeplearning4: https://deeplearning4j.org/neuralnet-overview

deeplearning4j. (2020). Obtenido de https://deeplearning4j.org/

Hale, J. (2020). *towardsdatascience*. Obtenido de towardsdatascience: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Hinton, G. E. (2020). A fast learning algorithm for deep belief nets. *IEEEXplore*.

IBM. (s.f.). *IBM COM*. Obtenido de https://www.ibm.com/developerworks/ssa/library/cc-machine-learning-deep-learning-architectures/index.html

imagine, D. (27 de 05 de 2016). *Deep imagine.* Obtenido de Universidad de los andes: https://documentodegrado.uniandes.edu.co/documentos/9205.pdf

karpathy. (2020). Obtenido de karpathy.: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

keras. (2020). Obtenido de keras: https://keras.io/

keras.io. (s.f.). Obtenido de keras.io: https://keras.io/models/model/

Kim, J. (2020). Deeply-Recursive Convolutional Network for Image Super-Resolution. *IEEEXplore*.

Learning, I. t. (2020). *datascienceassn.* Obtenido de http://www.datascienceassn.org: http://www.datascienceassn.org/sites/default/files/Introduction%20to%20Machine%20Learning.pdf

machinelearningmastery. (s.f.). Obtenido de machinelearningmastery.com: https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

mathworks. (2020). *https://es.mathworks.com.* Obtenido de mathworks: https://es.mathworks.com/discovery/deep-learning.html

medium. (2020). Obtenido de medium: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

Minambiente. (2020). Obtenido de http://www.minambiente.gov.co/index.php/asuntos-ambientales-sectorial-y-urbana/gestion-del-aire/contaminacion-atmosferica

NatureDeepReview. (2020). *https://www.cs.toronto.edu/.* Obtenido de https://www.cs.toronto.edu/: https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

OMS. (2020). *www.who.int.* Obtenido de https://www.who.int/bulletin/volumes/88/4/10-010410/es/#:~:text=M%C3%A1s%20de%20la%20mitad%20de,unas%20165.000%20personas%20por%20d%C3%ADa.

Resolucion N 4133.0.0 984 . (Diciembre de 2013). Cali.

RUNT. (2020). *https://www.runt.com.co*. Obtenido de https://www.runt.com.co: https://www.runt.com.co/runt-en-cifras/parque-automotor

springer. (2020). *springer*. Obtenido de springer: https://link.springer.com/referenceworkentry/10.1007%2F978-1-4899-7687-1_192

Tasa de Motorización. (2020). Obtenido de www.cali.gov.co: http://www.cali.gov.co/observatorios/publicaciones/134358/movis_et_tti_tasa_motorizacion/

Tensorflow. (2020). Obtenido de https://www.tensorflow.org/

Udacity. (2020). *Udacity.* Obtenido de classroom.udacity.com: https://classroom.udacity.com

Wiseman, E. (2020). DEEP LEARNING FOR HUMAN DECISION SUPPORT. *Government of Canada*.

xenonstack. (27 de 04 de 2017). *xenonstack*. Obtenido de xenonstack: /www.xenonstack.com/blog/log-analytics-with-deep-learning-and-machine-learning