Universidad Autónoma De Querétaro

Facultad de Ingeniería

Maestría en Ciencias: Ingeniería Matemática

Predicción de Precios de Productos Agrícolas Mediante Redes Neuronales Recurrentes

Requisito indispensable para obtener el Grado de

Maestro en Ingeniería Matemática

Tesis presentada para lo obtención del grado académico

de Maestro en Ciencias con Línea Terminal en

Ingeniería Matemática

Querétaro, Qro., a __ de ____ de 2021

REPOSITORIO INSTITUCIONAL DGBSDI-UAQ

Dirección General de Bibliotecas y Servicios Digitales de Información

Predicción de Precios de Productos Agrícolas Mediante Redes Neuronales Recurrentes

por

Pedro Huitrón Tierrablanca

se distribuye bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional.

Clave RI: IGMAC-281632

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias

Tesis como requisito para obtener para obtener el grado de:
Maestro en Ciencias

Presenta:
Pedro Huitrón Tierrablanca

Dirigido por:
Mtro. Wilfrido Jacobo Paredes García

M en C. Wilfrido Jacobo Paredes García _____

Dr. Christopher Alexis Cedillo Jiménez _____

Dr. Roberto Augusto Gómez Loenzo _____

M. en C. Luisa Ramírez Granados _____

Dra. Rosalía Ocampo Velázquez _____

_____ _____

Dr. Manuel Toledano Ayala
Director de la Facultad

Dra. Ma. Guadalupe Flavia Loarca Piña
Directora de Investigación y Posgrado

RESUMEN

Objetivo: Construir una Red Neuronal Recurrente Elman que pueda superar en resultados al modelo predictivo SARIMA usando la información disponible en el Sistema de Información e Integración de Mercados (SNIIM).

Métodos: Se programó un modelo desde cero con el lenguaje de programación Python y las bibliotecas TensorFlow y Keras, usando la plataforma Google Colab para modelar Series de Tiempo a Ventanas de Comercialización y obteniendo como resultado una predicción con cierto porcentaje de error junto con algunas gráficas asociadas a dichas ventanas, al final se elaboró una prueba de proporciones para decidir si dicho modelo de Redes Neuronales Recurrentes (RNN) tipo Elman pudo superar al método SARIMA.

Resultados: A través de la prueba de proporciones, que tomó en cuenta los resultados de 23 productos, se puede sugerir que el método de RNN superó al Modelo autorregresivo integrado de media móvil (SARIMA) e incluso pudo modelar productos que SARIMA no de acuerdo a los datos proporcionados.

Conclusiones: La capacidad de predicción es muy significativa y, debido a la robustez del código para cambiar parámetros de manera relativamente sencilla, es posible encontrar la variante que pueda predecir mejor cierto producto en específico y, aunque se cumplió exitosamente el objetivo de la presente investigación, todavía es posible hacer mejoras al proceso actual para trabajos posteriores.

(Palabra Clave: redes neuronales, series de tiempo, aprendizaje automático, ventanas de comercialización)

ABSTRACT

Objetive: To build an Elman Recurrent Neural Network that can overperform the SARIMA predictive model in results using the information available in the Market Information and Integration System (SNIIM)

Methods: A model was programmed from scratch with the Python programming language and the TensorFlow and Keras libraries, using Google Colab platform to model Time Series to Market Windows and obtaining as a result a prediction with a certain percentage of error along with some graphs associated with these windows, in the final part a proportion test was elaborated to decide if the Recurrent Neural Network (RNN) model can overperform the SARIMA method.

Findings: Through a proportion test, which took into account results of 23 products, it can suggest that the Recurrent Neural Network (RNN) method (in its Elman variant) overperformed SARIMA, and it was even able to model products that SARIMA couldn't.

Conclusions: Predictability is very significant and, due to the robustness of the code to change parameters in a relatively simple way, it's possible to find the variant that can better predict a certain specific product and, although the objetive of the present research was successfully met, it's possible to make improvements to the current process.

(Keywords: neural networks, time series, machine learning, market windows)

DEDICATORIA

A las circunstancias pasadas y actuales que me llevaron a tomar esta decisión tan acertada en mi vida sobre otras cosas, algo de lo que nunca me arrepentiré jamás, y a esas personas y eventos que hicieron que tuviera el interés por el cual estoy haciendo este trabajo.

Más específicamente a mis papás porque en los momentos más difíciles tuve de ellos ese apoyo moral que me llevó no rendirme y seguir adelante, y a los valores que me inculcaron, los cuales han servido como herramientas esenciales en el éxito de mis proyectos de vida.

A mis hermanos que son mi segundo apoyo moral y al ser yo su hermano mayor, me motivan a ser el ejemplo para ambos, lo cual conlleva la responsabilidad de ser mejor cada día.

Al proyecto “Germinando” y a sus integrantes porque sin ellos no existiría esta investigación, ni hubiese ganado la formación profesional que obtuve de desarrollar esta pequeña contribución.

A mis profesores de esta maestría, porque además del enseñarme el contenido del programa, lo cual lo hicieron de manera excelente, me enseñaron la pasión que existe por el conocimiento y que uno puede tener la capacidad de indagar más.

A mis compañeros del programa, ya que sin ellos la maestría hubiese sido más complicada, y entre todos nos apoyamos en aquellos momentos más complicados y donde existía mayor incertidumbre.

AGRADECIMIENTOS

Me gustaría hacer una mención especial al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado, ya que con esta ayuda pude dedicarme a investigar a tiempo completo en este proyecto y sobre todo en lograr una formación más detallada y completa.

A mis papás nuevamente, ya que además de su apoyo moral, desinteresadamente me apoyaron en términos económicos los primeros meses durante el proceso de ingreso al programa, lo cual me ayudó a ingresar en menor tiempo.

A mis asesores por toda la paciencia que tuvieron conmigo y por darme mi lugar en una posición tan importante, lo cual me motivo a trabajar a esa altura.

A la facultad de ingeniería, funcionarios, instalaciones y demás servicios que fueron herramientas esenciales para desempeñar de manera más rápida y eficiente mí trabajo.

Y al coordinador de mi línea terminal de Ingeniería Matemática: Roberto Augusto Gómez Loenzo, cuya primer entrevista anterior a mi ingreso aún recuerdo y cómo esto me animó a presentarme al proceso de admisión del programa, en aquel entonces para mí fue un momento clave sobre qué hacer los siguientes años de mi vida.

CONTENIDO

1. INTRODUCCIÓN 9

1.1. Antecedentes 9

1.2. Planteamiento del Problema y Justificación. 12

1.2.1. Planteamiento del Problema 12

1.2.2. Justificación 15

1.3. Hipótesis 17

1.4. Objetivos 17

1.4.1. Objetivo General 17

1.4.2. Objetivos Específicos 17

2. MARCO CONCEPTUAL 18

2.1. Métodos Predictivos 18

2.2. Redes Neuronales Artificiales (ANNs) 23

2.3. Proceso para predicción en Aprendizaje Automático. 33

2.4. Proceso para predicción en RNAs. 34

2.4.1. Conceptos Básicos 34

2.4.2. Conceptos sobre Keras 37

2.4.3. Evaluar conjuntos de datos: Entrenamiento, Validación y Prueba 38

3.2.4. Preprocesamiento de datos 40

3.2.5. Subajuste y Sobreajuste 41

3.2.6. Reuniendo todos los conceptos 42

2.5. ¿Cómo preparar una predicción en RNA de Series de Tiempo?. 44

2.5.1. Convertir arreglos a lotes de ventanas. 45

2.5.2. Generar y Ejecutar modelos Predictivos en RNAs. 50

3. METODOLOGÍA 54

4. RESULTADOS Y DISCUSIÓN 59

5. CONCLUSIONES 63

6. BIBLIOGRAFÍA 64

7. ANEXOS 67

7.1. Resultados gráficos de las evaluaciones con el método de RNN. 67

7.2. Código Desarrollado. 80

ÍNDICE DE FIGURAS

Fig. 1.Volatilidad de los precios en alimentos básicos. Fuente: (FAO, 2010) 13
Fig. 2. Modelo explicativo. Fuente: (elaboración propia) 20
Fig. 3. Modelo predictivo. Fuente: (Elaboración Propia) 21
Fig. 4. Neurona Biológica. Fuente: (Torres Durán, 2013) 24
Fig. 5. Red con una neurona. Fuente: (elaboración propia) 24
Fig. 6. Función heaviside. Fuente: (elaboración propia) 25
Fig. 7. Función sigmoide. Fuente: (elaboración propia) 26
Fig. 8. Conexión entre neuronas. Fuente: (Rashid, 2016) 26
Fig. 9. Topología de la Red Neuronal. Fuente: (elaboración propia) 27
Fig. 10. Efectos de la función de error. Fuente: (elaboración propia) 29
Fig. 11. Descenso de gradiente en 2 y 3 dimensiones Fuente: (elaboración propia) 30
Fig. 12. Comparación entre tasa de aprendizaje alta y baja. Fuente: (Nagy et al., 2019) 31
Fig. 13. Variantes de RNN. Fuente: (Jones, 2017) 32
Fig. 14. Proceso básico de la red neuronal. Fuente: (Chollet, 2018) 35
Fig. 15. Modelado de una serie de tiempo. Fuente: (Chollet, 2018) 36
Fig. 16. Representación de un Proceso de Aprendizaje de ANN. Fuente: (Elaboración Propia) 41
Fig. 17. Ejemplo de datos de entrada en una serie de tiempo. Fuente: (Elaboración Propia) 45
Fig. 18. Estructura de datos en el proceso de predicción. Fuente: (Elaboración Propia) 46
Fig. 19. Generación de serie a lote de ventanas. Fuente: (Elaboración Propia) 47
Fig. 20. Ejemplo ventana como elemento de un lote. Fuente: (Elaboración Propia) 48
Fig. 21. Ejemplo de ventana dividida en entradas y etiquetas. Fuente: (Elaboración Propia) 49
Fig. 22. Método ".map" en un lote de ventanas. Fuente: (Elaboración Propia) 50
Fig. 23. Forma Final del Dataset. Fuente: (Elaboración Propia) 50
Fig. 24. Estructura de red neuronal utilizada. Fuente: (Elaboración Propia) 51
Fig. 25. Extracto del extracto del código de la red neuronal. Fuente: (Elaboración Propia) 52
Fig. 26. Variables utilizadas en la red neuronal. Fuente: (Elaboración Propia) 56
Fig. 27. Porcentajes de Error de SARIMA y RNN en términos de RMSE. 60
Fig. 28. Tres periodos de las diferencias entre predicciones (cruz) y etiquetas (punto). 62

ÍNDICE DE ECUACIONES

Ecuación 1. Ejemplo de Métricas para Evaluar el Error 23
Ecuación 2. Forma Algebraica de una Red Neuronal 27
Ecuación 3. Activación con la Función Sigmoide 28
Ecuación 4. Forma Algebraica del Descenso del Gradiente 29

1. INTRODUCCIÓN

1.1. Antecedentes

El reconocer la existencia de los derechos humanos implica saber la importancia que tiene el derecho a la alimentación, tanto en términos de cantidad como calidad, y el derecho a un trabajo que permita alcanzar una calidad de vida; ya lo mencionaba la ONU desde la proclamación de la Declaración Universal de los Derechos Humanos en 1948:

"Toda persona tiene derecho al trabajo y a la libre elección de este […] Toda persona tiene derecho a un nivel de vida adecuado que le asegure, así como a su familia, su salud y su bienestar, y en especial la alimentación…" (ONU, 2012)

En ambos casos el Estado Mexicano formalmente garantiza, como derecho constitucional, tanto una alimentación "nutritiva, suficiente y de calidad" como un empleo "digno y socialmente útil".(Constitución Política de Los Estados Unidos Mexicanos, 1917)

Esto también se ratifica en el artículo 6 de la Ley General de Desarrollo Social, y establece que "Son derechos para el desarrollo social la educación, la salud, la alimentación nutritiva y decorosa…" (Camara de Diputados de H. Congreso de la Union, 2004)

Esta situación no solo es una cuestión meramente formal, sino una posibilidad muy viable en México, ya que se caracteriza por sus recursos naturales, excelente localización y condiciones para la actividad económica; No obstante, la realidad es otra.

A pesar de que estos hechos han supuesto un avance para garantizar el derecho a la alimentación y reducir la pobreza, aún existen ciertas debilidades que no han sido erradicadas, como el acceso a una alimentación de calidad, la malnutrición o el sobrepeso y obesidad, sobre todo en áreas rurales (Cárdenas Elizalde et al., 2018).

La accesibilidad física de alimentos saludables y nutritivos es otro de los problemas comunes, ya que mientras las localidades urbanas tienen amplia disponibilidad, la mayoría de zonas rurales se ven en la necesidad de desplazarse para proveer su consumo, lo cual genera gastos adicionales tanto económicos como de tiempo. (Cárdenas Elizalde et al., 2018)

Un aumento de los precios agrícolas afecta a la mayoría de consumidores que son compradores netos (es decir que consumen más de lo que venden), también se refleja en un mayor impacto de personas que son pobres, ya que tienen que asignar una mayor proporción de ingresos a alimentos que el consumidor promedio. (Banco Mundial, 2008)

Se podría aseverar que un aumento de precio es una oportunidad para los productores, ya que un precio alto significa productos más rentables, lo cual ciertamente se da con grandes productores, pero no con los pequeños; la razón es que estos últimos no cuentan con insumos suficientes, su infraestructura no es la adecuada, además su acceso a la información de precios es limitada (sobre todo en mercados internacionales). (FAO, 2009)

Nuestro país pasa por una etapa de bajo desempeño del sector primario en los últimos años, esto debido a una orientación de la política económica basada en la manufactura, la cual ha incentivado a la economía, pero a costa de hacer a un lado a actividades como la agricultura; aun cuando existe evidencia de una relación causal entre el PIB de un país en desarrollo y el aumento del valor agregado de la agricultura. (Tiffin & Irz, 2006)

Vale la pena mencionar que el gobierno maneja diferentes indicadores para medir la pobreza donde se encuentran incluidos tanto la alimentación como el ingreso laboral, tal es el caso del indicador manejado por el CONEVAL estipulado en la Ley General de Desarrollo Social y por otro lado del que elabora SEDESOL con fundamento en la Ley de Coordinación Fiscal.

En este tiempo nos enfrentamos a un sector económico que está muy lejos de la autosuficiencia alimentaria, para ejemplificar basta notar que del total de la población económicamente activa solo el 4% se dedica a actividades agrícolas.(Monjarás, 2017)

Aunado a esto la producción del sector primario se encuentra concentrada solamente en 30 productos principales. Una posible causa de dicha reducción es que la agricultura se ha mantenido por debajo del PIB total, lo que lleva a reducir su participación. A pesar de este hecho, en años recientes se ha observado un saldo positivo en la balanza comercial agropecuaria y un incremento en las exportaciones. (Martínez et al., 2017)

Un claro ejemplo de producto exitoso es el jitomate, que además de satisfacer la demanda total interna, tiene el 25% de la participación en exportaciones mundiales (Rizo, 2018), otros buenos productos son el aguacate y la cebolla que también tienen un lugar destacado en la producción.

A pesar del potencial de estos productos existe una alta dependencia de productos importados, como es el caso del maíz, cuya compra a Estados Unidos ha generado una dependencia muy cercana al cien por ciento. Otros productos muy importados son la semilla de soya o el trigo. (Martínez et al., 2017)

El hecho es que ha habido varios cambios en el sector, tanto en la forma de cultivar como en las técnicas utilizadas para su producción, lo cual ha fortalecido a ciertos cultivos, pero también ha debilitado a otros; se puede citar a la frambuesa, a los espárragos y a las fresas, las cuales han alcanzado crecimientos considerables sin embargo otros productos como el maíz, el trigo y el frijol tienen tasas de crecimiento inferiores a las globales. (Díaz & Meade, 2019)

Con base a los argumentos anteriores se puede notar la importancia que tiene el conocer el precio a una fecha determinada de los productos agrícolas como medida para resolver problemas de costeo y rentabilidad en la comercialización (Díaz & Meade, 2019). Por lo mismo empezaron a surgir personas que elaboran metodologías para estimar dichos precios.

Los beneficios de predecir se dan cuando hablamos en términos de la planeación, es decidir contestar preguntas como ¿qué producir?, ¿cuándo hacerlo?, ¿se debe almacenar o no?, ¿se debería diversificar?, ¿cuáles serían los posibles costos? (FAO, 2001). Aunque quizá una pregunta para el proyecto que apoya este trabajo es: ¿cuál es la merma esperada?

Poniendo nuestra atención a la planeación, la innovación en procesos necesita nuevos roles; los empleos que fomentarán la nueva agricultura serán el economista agrícola o el experto en informática agrícola por mencionar algunos. (Buhigas, 2017)

Para poder evaluar el porqué del éxito o fracaso de un producto agrícola hay que tomar en cuenta diferentes factores, por ejemplo la influencia que tienen las condiciones climatológicas, los desastres naturales, las plagas, las enfermedades o el precio (Ayuso, 2018).

Los métodos de estimación en el sector agrícola inician en la India con el trabajo hecho por el estadístico P.V. *Sukhatme*, quien desarrolló las primeras aplicaciones de estas técnicas en productos agroindustriales, siendo él fundador de la *Indian Society of Agricultural Statistics* y llegando a ser la cabeza de la FAO (una división de las Naciones Unidas dedicada a la alimentación y agricultura) y cuyo trabajo se centró principalmente en nutrición. (Directorate of Economics and Statistics, Government of Puducherry, 2018)

En 1983 ya se buscaba combatir una alta volatilidad de los precios de bienes agrícolas comparando diferentes técnicas para estimar los mismos datos bajo diferentes enfoques cuantitativos (econométrico, ARIMA, suavización exponencial, y otros) y la opinión de un experto (es decir un enfoque cualitativo), donde queda claro que las técnicas de estimación cuantitativa son más precisas que la estimación por la experiencia.(Brandt & Bessler, 1983)

En el caso de México la estimación se ha hecho a través de series de tiempo con modelos como el Modelo autorregresivo integrado de media móvil estacional; por otro lado, su modelado a través de Redes Neuronales Artificiales (ANNs) es muy incipiente, del que casi no ha llegado a desarrollarse en la actualidad, apenas con algunos ejemplos como el modelado que hizo el Dr. Oscar Ariel Salazar con precios del maíz en la Universidad de Chapingo. (Zarecero, 2017)

Actualmente las mejoras en sistemas inteligentes del mercado agrícola se aprecian más en su producción como sistemas de riego o monitoreo de cultivo, sin embargo, no se ha hecho énfasis en la comercialización. Es aquí donde parte el presente proyecto de investigación que busca crear una metodología para la obtención de precios y compararlo con un método alterno ya desarrollado.

1.2. Planteamiento del Problema y Justificación.

1.2.1. Planteamiento del Problema

Es un hecho que en los últimos años la volatilidad en los precios agrícolas han incrementado de manera sustancial, tan solo en un lapso de 25 años los precios de la soya, maíz y trigo han aumentado de manera considerable tal como lo indica la Fig. 1.

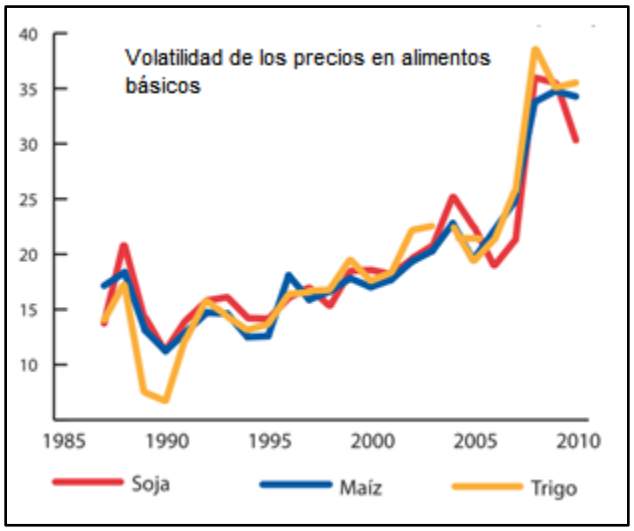


Fig. 1.Volatilidad de los precios en alimentos básicos. Fuente: (FAO, 2010)

Este fenómeno, junto con el hecho de que los mercados están integrados cada vez más en una escala mundial, contribuye a que dichas perturbaciones pueden pasar del nivel internacional al nacional con mayor velocidad. Esto afecta de manera especial a países en desarrollo donde, como se mencionó antes, gastan gran parte de sus ingresos en alimentos, y también en sus decisiones de inversión y planificación (FAO, 2010).

Tomando en cuenta este último punto podemos decir que una mala planeación nos puede llevar a mermas considerables, un tema del cual ya habíamos abordado anteriormente y que parece conveniente resaltar debido a los problemas que genera. Los factores causantes son variados ya sean tecnológicos, económicos, comerciales o las malas prácticas de las personas.

Actualmente esto se ve reflejado en el desperdicio del 30% de la producción en alimentos para el consumo humano que son $1,300 millones de toneladas en comida traducido en 1 billón de dólares en costos de producción, dicho de otra forma $700,000/$900,000 millones en costos ambientales y sociales respectivamente (FAO, 2020).

Si apreciamos mejor este tema de las exportaciones podemos argumentar que existen mercados con un potencial importante, debido a la demanda de otros países, pero a falta de eficiencia el beneficio se lo lleva otro competidor, típicamente un país con mejor desarrollo en el ramo.

Productos como la coliflor, el brócoli, el espárrago, el pimiento, la uva, el limón o la lima tienen diferenciales de precios que no son aprovechados (Wageningen Food & Biobased Research, 2014).

Una vez ya planteado el problema podríamos empezar a pensar qué sería conveniente en un modelo de estimación de precios que refleje la naturaleza del mercado agrícola (con modelos de pronóstico no lineales) y evaluar qué tan eficiente es como herramienta para proveer información a los agricultores, pero se abordarán más detalles de eso hasta las secciones correspondientes.

Un mercado agrícola posee características especiales como estacionalidad de producción, precio inelástico a la oferta y demanda y la naturaleza de la cosecha las cuales juegan un rol importante en el comportamiento del precio. (Jha & Sinha, 2013)

Dicho esto vale la pena mencionar que una predicción de este tipo se puede elaborar bajo dos enfoques, estructural o series, siendo el primer enfoque el más demandante en cuanto a datos con respecto al segundo, lo cual hace ideal a este último para países en desarrollo. (Jha & Sinha, 2013)

El estudio de series de tiempo se puede manejar tanto con modelos Box-Jenkins (SARIMA) y ANNs cada uno con sus ventajas y desventajas.

Se dice que el modelo de ANNs tiene la ventaja de poder predecir tendencias no lineales, de hecho se asume que los datos del sector agrícola tienen esta tendencia, debido a factores como desastres naturales o pestes, pudiéndose comprobar a través de un test de no linealidad. (Jha & Sinha, 2013)

La aplicación Redes Neuronales Artificiales Recurrentes (RNAs), una variante recurrente de las ANNs, en series de tiempo agrícolas es relativamente poca si se compara con estudios de series de tiempo en acciones o cualquier otro instrumento financiero, lo cual la hace un campo de estudio bastante atractivo considerando los beneficios que ofrece.

Es importante mencionar la existencia del estudio anterior donde se llevó a cabo la predicción de precios de productos agrícolas basado en el modelo Box Jenkins, una predicción conocida como *Modelo Autorregresivo Integrado de Media Móvil Estacional* (SARIMA), la ventaja de este es que este permite a los investigadores entender la dinámica del mercado.(Paredes-Garcia et al., 2019)

No obstante aún no queda claro qué modelo es más conveniente para este caso en específico ya que por un lado SARIMA es lineal y los datos son no lineales. Una ANN del tipo recurrente ofrece ventajas como la capacidad de predicción no lineal, convergencia rápida y para este trabajo en específico una capa recurrente que permita reconocer patrones tanto temporales como espaciales. (Wang & Wang, 2016)

1.2.2. Justificación

La ONU señala al sector alimentario y agrícola como una de las claves para la eliminación del hambre y una generación de ingresos aceptable (*United Nations*, 2020). Eso es posible ya que el potencial de la agricultura mexicana se refleja en el 3.55% del producto interno bruto (Banco Mundial, 2017), en una superficie sembrada de 21.6 millones de hectáreas (SIAP, 2017) y en 54.9 millones de personas trabajando en el campo.

Sin embargo, cerca de 18 millones 370 mil toneladas de la producción nacional pasan a ser mermas, las razones son varias, como empaques poco adecuados, falta o mal funcionamiento de las redes de enfriamiento, la falta de monitoreo de plagas, producción no vendida entre otras (Grupo Comunicación y Medios).

Todo esto causa una pérdida de 79 mil 695 millones de pesos, equivalente al 12.7% de la producción agrícola total, dichas prácticas se pudieron haber evitado con la correcta planeación, información oportuna, un adecuado almacenaje y un transporte eficiente (Fermoso Gómez, 2016).

El CONEVAL admite que aunque el problema de abasto no es inminente, se ha tenido que compensar con importaciones y su falta de infraestructura provoca desperdicio de alimentos, con lo cual en varias ocasiones no se puede contener el costo ni garantizar el abasto constante. (Cárdenas Elizalde et al., 2018)

Los eventos antes mencionados pueden llegar a reducir los precios de mercado como parte de un proceso de malbaratamiento lo cual no es benéfico, ya que aunque es conveniente para el consumidor pero el productor sufre las pérdidas generadas, lo que implica una mayor dificultad para invertir a largo plazo. (da Silva, 2016).

Otro motivo es la desigualdad alimentaria, la FAO alega que los países de bajos ingresos son los que tienen pérdidas más significativas debido a infraestructuras deficientes y obsoletas, así como conocimientos limitados e inversión nula.

La situación se agrava si consideramos los efectos del cambio climático y la huella ecológica que ha tenido en la agricultura sobre tierras y recursos hídricos ya de por sí escasos o su impacto en la biodiversidad. (FAO, 2015)

De ahí surge la necesidad de encontrar una manera que dé certidumbre a los productores sobre qué y cuándo producir, y aunque existen mecanismos de los cuales el gobierno se vale, como los precios en garantía (donde se ofrece un precio bajo de compra y un precio alto de venta, subsidiando el gobierno la diferencia) realmente esto es un premio a la deficiencia ya que no se ve mejoría en el proceso, además de que este mecanismo es muy propenso a manejarse con corrupción. (FAO, 2019)

Es por eso que se aconseja encontrar maneras más objetivas de asegurar la demanda a un precio que resulte rentable para los productores, con productos agrícolas que sean indicadores de la oferta y que su seguimiento constituya una base para la toma de decisiones (FAO, 2020), pero sin que esto resulte en un problema económico, además que pueda dar más poder al productor frente a otros participantes, como lo son los intermediarios.

Una forma es a través de un sistema de información que permita a los productores (sobre todo productores pequeños y medianos) tener una herramienta para la toma de decisiones (qué vender, cuándo vender y a qué precio), a través de una metodología que pueda ser confiable y útil para los usuarios del mercado agrícola.

Las ventajas de este sistema serían el hecho de que existirá certidumbre sobre los precios, más allá de solo hacer una estimación empírica, la cual servirá de sustento desde la decisión de un pequeño productor hasta la política de un organismo participante.

De esta necesidad surge el proyecto ‘Germinando’, cuyo objetivo es aprovechar la producción ya existente de los productores y ofrecer estabilidad en sus ventas a través de estimaciones clave obtenidas con un modelo de estimación de precios.

Con esto se justifica la razón por la que se necesita una metodología adecuada para dichas estimaciones, donde se analizarán datos sobre los precios de diferentes productos para revisar su viabilidad y encontrar posibles ventanas de comercialización con un nivel aceptable de confiabilidad.

1.3. Hipótesis

Los precios del mercado agrícola de Querétaro se pueden estimar con mayor exactitud usando el método de Redes Neuronales Recurrentes (RNN) comparado con el modelo Autorregresivo Integrado de Promedio Móvil Estacional (SARIMA).

1.4. Objetivos

1.4.1. Objetivo General

Construir una Red Neuronal Recurrente Elman que pueda superar en resultados al modelo predictivo SARIMA usando la información disponible en el SNIIM.

1.4.2. Objetivos Específicos

1. Determinar la variante de red neuronal que mejor se acople al problema de series de tiempo del mercado agrícola.
2. Determinar los datos que pueden modelarse del sistema nacional de información e integración de mercados de la Secretaría de Economía.
3. Construir el modelo RNN.
4. Medir los resultados a un criterio de evaluación apropiado

2. MARCO CONCEPTUAL

2.1. Métodos Predictivos

“Un pronóstico es una predicción del futuro de un evento o eventos ”, esto incluye datos históricos y el conocimiento de eventos futuros que pueden impactar dichas predicciones; generalmente mediante el uso de modelos que apoyen en el análisis (Montgomery et al., 2016).

Las actividades de predicción se encuentran en múltiples ámbitos, pudiendo ser negocios, política, economía, ciencias ambientales, ciencias sociales o medicina por mencionar algunos. (Montgomery et al., 2016).

En el modelado estadístico puede existir cierta ambigüedad tanto para el término “Predicción” como el término “Explicación” ya que ambos se derivan de este, sin embargo cada uno tiene identidad propia.

Mientras que en el término “Explicación” nos referimos a encontrar si la variable X provoca la variable Y (relación de causalidad) mediante un fuerte enfoque teórico con el que se justifica; en el término “Predicción” se toma un enfoque más empírico para encontrar valores futuros basado en datos históricos (Shmueli, 2010) .

Por otro lado el significado de “Predicción” también debe diferenciarse de otros términos como los son “Metas” o “Plan”; y aunque estén relacionados hay que resaltar que no son lo mismo. Una meta se define como los resultados deseados, pero que existe la posibilidad de que no ocurran, mientras que un plan son las acciones que se requieren para que una meta se convierta en dicha predicción (Richardson & Newbury, 2018).

Las predicciones de acuerdo a la longitud del tiempo pueden ser a corto, mediano o largo plazo; El corto plazo involucra periodos cortos (días, semanas o meses), el medio uno o dos años y el largo varios años en el futuro.

En el corto y mediano plazo se requieren actividades que van desde la administración de operaciones hasta la elaboración de presupuestos para nuevos proyectos; por otro lado el largo plazo se reserva para planes estratégicos. Son los plazos corto y mediano en los que se identifica, modela y extrapola los patrones que se encuentran en los datos históricos (Montgomery et al., 2016).

El horizonte a elegir dependerá de las necesidades de predicción de cada entidad en particular y lo que se predecirá, incluso se puede asumir que existe una relación entre la longitud del horizonte con el error producido (Smith & Sincich, 1991), lo cual se debe tomar en cuenta.

La cantidad de información encontrada es otra determinante para saber qué modelo se usará.

Un ejemplo claro son los modelos que se realizan cuando no hay suficiente información disponible; me refiero a los *modelos cualitativos*, los cuales no son solo conjeturas; se trata de enfoques bien estructurados de los que se obtienen buenos resultados aún sin información explícita (con ciertas limitaciones) pero que no serán cubiertos en el presente trabajo, ya que en este caso hay información suficiente para optar por un *modelo cuantitativo* (Richardson & Newbury, 2018), pero es importante mencionar que existen**.**

Con respecto a los modelos cuantitativos para que sea viable se necesitan dos condiciones: primero, la disponibilidad de la información numérica e histórica y segundo, que sea razonable asumir que algunos aspectos de los patrones del pasado continuarán en el futuro.

Abarcando el primer punto mencionado, con respecto a la disponibilidad de la información no solo se refiere a su existencia, sino a que esta sea relevante, suele pasar que los sistemas de almacenamiento cambien y que los datos ya no sean útiles o puede ser que sean útiles pero que se tenga que lidiar con dificultades como valores faltantes, valores atípicos, o una nueva metodología de captura (Montgomery et al., 2016).

Una vez listos para elegir una forma de trabajar y los datos, hay que tomar en cuenta de que existe una gran cantidad de modelos cuantitativos de predicción, muchas veces desarrollados para fines específicos, con particularidades, precisiones y costos que deben considerarse en el momento de su elección.

Casi todos los problemas de predicción con modelos cuantitativos recaen ya sea en *Series de Tiempo* (típicamente en intervalos de tiempo regulares) o en *Datos Transversales* (datos que se toman a un punto concreto en el tiempo); reitero que el presente trabajo se basa en Series de Tiempo.

Podemos definir una serie de tiempo como una sucesión de observaciones cronológica o una sucesión orientada por el tiempo de la variable de interés, tanto en intervalos regulares como irregulares (Montgomery et al., 2016);

Nuestro problema actual se encuentra en intervalos regulares (por ejemplo cada hora, día, semana, mes o año), lo cual facilita hasta cierto punto la forma de resolverlo, sin embargo existen formas lidiar con intervalos irregulares, pero no hay la necesidad de eso en este problema.

Con respecto al segundo punto, el de desarrollar un modelo cuantitativo, hacer una estimación no es otra cosa que extrapolar y esta actividad se puede definir como “averiguar o estimar el valor de una magnitud para valores de la variable que se hallan fuera del intervalo en que dicha magnitud es conocida” (*Extrapolar | Definición | Diccionario de La Lengua Española | RAE - ASALE*, n.d.).

Dicha extrapolación no solo debe continuar con los datos históricos sino que además debe hacerlo de manera que sea capaz de capturar los patrones subyacentes, el método más sencillo para hacer esto es usar únicamente la información de la variable a predecir sin descubrir los factores que lo provocan y contrario a lo que se podría pensar este tipo de análisis es capaz de encontrar patrones en la mayoría de los casos (Richardson & Newbury, 2018).

Podemos usar variables estimadoras para hacer predicción de series de tiempo, por ejemplo si queremos seguir con la predicción de precios agrícolas (PA) por semana se podría tomar de la siguiente semana:

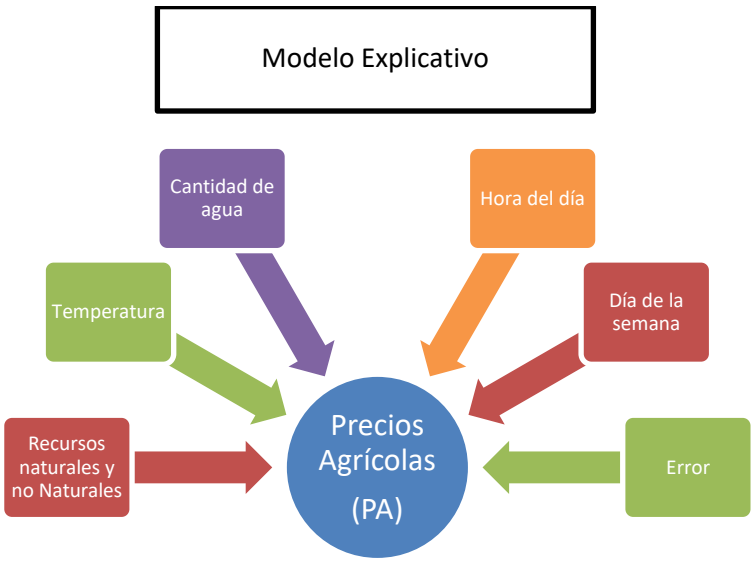


Fig. 2. Ejemplo de modelo explicativo. Fuente: (elaboración propia)

A esto se le denomina *modelo explicativo* ya que indica la variación del precio del producto agrícola referido exponiendo sus causas; dicho modelo contiene un término “error” al que indican los cambios en la demanda que no pueden ser explicados únicamente con las variables definidas, para compensar las variables que afectaron los resultados, es decir todo el modelo consiste en los datos definidos más el error (Shmueli, 2010).

También existe la opción de definir un modelo si los datos solo están representados como una serie de tiempo de la variable a definir, es decir que en este ejemplo la información estuviera en términos de periodos anteriores, en ese caso se consideraría como un modelo predictivo (Shmueli, 2010), y quedaría algo similar a lo que sigue:

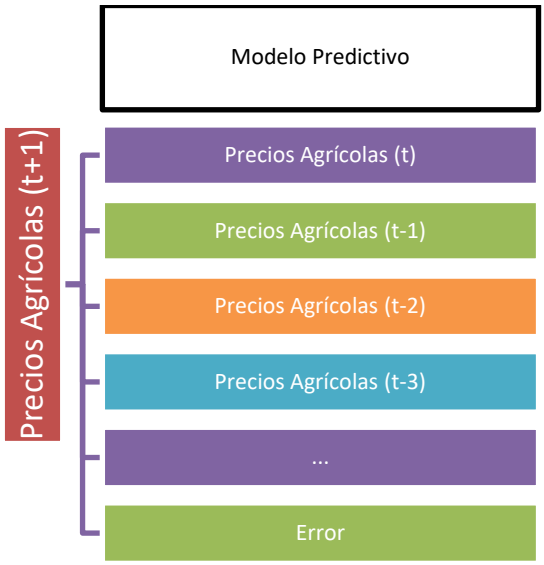


Fig. 3. Modelo predictivo. Fuente: (Elaboración Propia)

Otra opción posible es construir modelos híbridos que combinen modelos explicativos con modelos predictivos, pero no serán cubiertos en el presente trabajo;

Si bien es cierto que uno puede elegir entre las tres opciones descritas, existen razones que pueden llevarnos a elegir el modelo de series de tiempo sobre los otros dos; primero, los modelos explicativos e híbridos pueden no ser entendidos del todo, es decir, existe una dificultad asociada entre la relación de las variables (Richardson & Newbury, 2018) y lo que da lugar a su comportamiento lo cual puede reducir el provecho que puede ganar el usuario objetivo (personas comunes sin conocimientos técnicos).

Segundo, para conseguir la predicción de la variable de interés es necesario conocer sus predictores asociados y el resultado de estos, lo cual puede dificultar aún más el modelo; y tercero la mayor parte de las veces nos importa más predecir lo que pasó y no por qué pasó (Shmueli, 2010).

Entre las actividades que se llevan a cabo en la labor de predicción se encuentran: definir el problema, obtener información, análisis preliminar, elegir y ajustar modelos y finalmente el uso y evaluación del modelo.

Definir el problema; incluye conocer las expectativas del usuario, es aquí donde nos preguntamos cosas como la forma deseada de la predicción, su horizonte, la frecuencia de las actualizaciones, el nivel de exactitud, las particularidades del proceso a evaluar y entrevistas con los usuarios y tomadores de decisiones (Montgomery et al., 2016).

Obtener Información; casi siempre existen dos tipos de información que se pueden conseguir: datos estadísticos y la experiencia de personas especializadas; en el caso de los datos estadísticos hay veces en que los datos son muy antiguos y no son ajustables y, aunque un buen modelo predictivo debería ser capaz de lidiar con la evolución de los datos para no desperdiciar dicho recursos, en el peor de los casos se puede optar por usar solo los datos más recientes (Richardson & Newbury, 2018).

Elegir y ajustar modelos; Existen tres categorías para elegir: *Métodos Cualitativos* (Método Delphi, Estudio de Mercado, Panel de Consenso, Analogía Histórica), *Métodos Cuantitativos* (Los basados en métodos Box-Jenkins) y *Métodos Causales* (Los basados en modelos econométricos), cada uno con su precisión, campo de aplicación, datos requeridos, costos asociados y tiempo requerido para su desarrollo (Chambers et al., 1971).

Evaluación del modelo elegido; Primeramente hay que mencionar que para saber cómo se desempeñó un modelo lo ideal es hacerlo a través de nuevos datos que no se hayan usado para el proceso de ajuste, dicho esto la manera más sencilla es hacer una separación previa de los datos, uno para entrenar y otro para probar (Richardson & Newbury, 2018), los detalles se verán en secciones posteriores.

También hay que saber cómo medir la diferencia entre los datos observados y los datos predichos del modelo que comúnmente se le conoce como “error”, que básicamente es la parte que no pudo predecir el modelo, algunos ejemplos son aquellos *basados en escalas*, por ejemplo *Mean Absolute Error* “MAE” y *Root Mean Squared Error* “RMSE” o aquellos *basados en porcentajes* como *Mean Absolute Percentage Error* “MAPE”, (Richardson & Newbury, 2018)

Ejemplos de Métricas para Evaluar el Error

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

$$RMSE = \sqrt{\sum_{i=1}^{N}\frac{(y_i - \hat{y}_i)^2}{N}}$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Ecuación 1. Ejemplo de Métricas para Evaluar el Error

2.2. Redes Neuronales Artificiales (ANNs)

Se dice que la computación convencional se caracteriza por la solución de problemas matemáticos mediante la ejecución de un algoritmo, lo cual resulta muy efectivo para tratar tareas muy repetitivas, no obstante, dichos métodos empiezan a resultar poco efectivos cuando se tratan de problemas relacionados que requieren procesos más complejos y datos imprecisos (Navarrete García, 2003).

Es ahí donde herramientas ‘inteligentes’ salen a relucir; en este caso en concreto se hablará de las ANNs que son una estructura computacional inspirada en las neuronas biológicas y cuyo funcionamiento busca replicar un proceso de estas últimas, mediante la asignación de una entrada a una salida en una red que procesa datos altamente conectados (Oscar Ariel Zerecero Salazar, 2017).

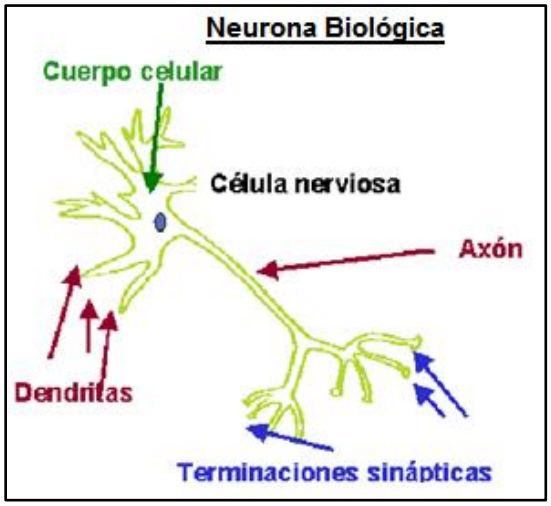


Fig. 4. Neurona Biológica. Fuente: (Torres Durán, 2013)

Las neuronas biológicas transmiten una señal eléctrica de una a otra, desde las dendritas hasta las terminaciones, estas señales pasan de una a otra (su forma se puede apreciar en la Fig. 4. Así es como nuestros cuerpos sienten la luz, el sonido, calor, presión, etc., es decir se toma una señal de entrada, dicha señal recorre el axón de la neurona y toma una salida la cual es la entrada de la siguiente neurona (Kaku, 2014).

Las RNAs comunican señales (números) a través de pesos y funciones de activación (para activar las neuronas). Usando un algoritmo de aprendizaje, estas redes ajustan pesos para resolver un problema dado.

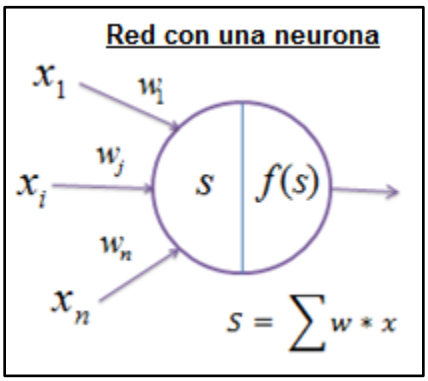


Fig. 5. Red con una neurona. Fuente: (elaboración propia)

La magnitud debe ser lo suficientemente considerable para superar cierto umbral y de esta manera desencadenar la señal. A una función de este tipo se le denomina función de activación. Dicha función se usa para calcular la salida de una neurona $f(s)$ como se describe en la Fig. 5 (Sivanandam & Deepa, 2006)

Existen diferentes tipos de funciones que pueden cumplir esta labor, la forma más sencilla sería la función escalón (función heaviside), la cual es cero inicialmente, pero toma la salida de uno a partir de cierto valor (como se ve en la Fig. 6).

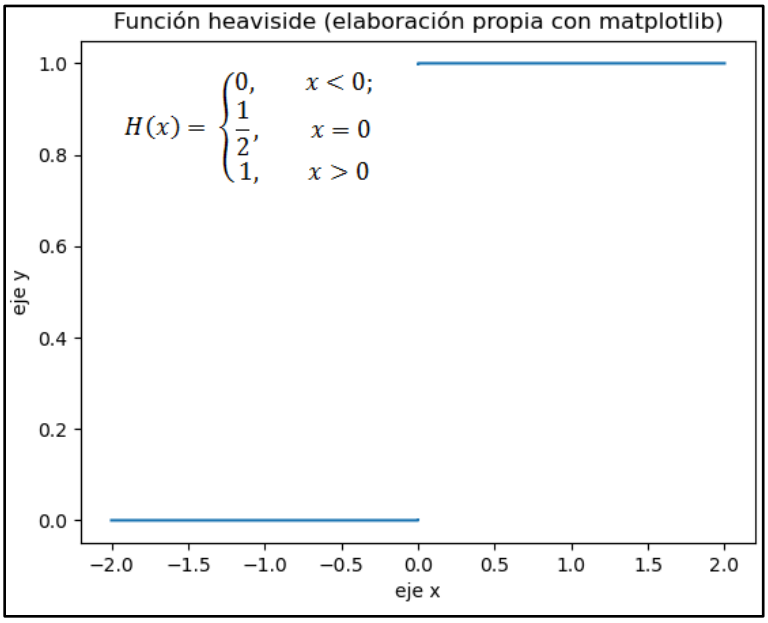


Fig. 6. Función heaviside. Fuente: (elaboración propia)

No obstante, dado la rigidez de esta función se busca otra que posea una forma de “S”, siendo la función sigmoide (también llamada función logística) una de las más usadas por la forma mencionada y por su simplicidad en los cálculos, la cual se puede apreciar en la Fig. 7 (Shouhua & Jingying, 2009).

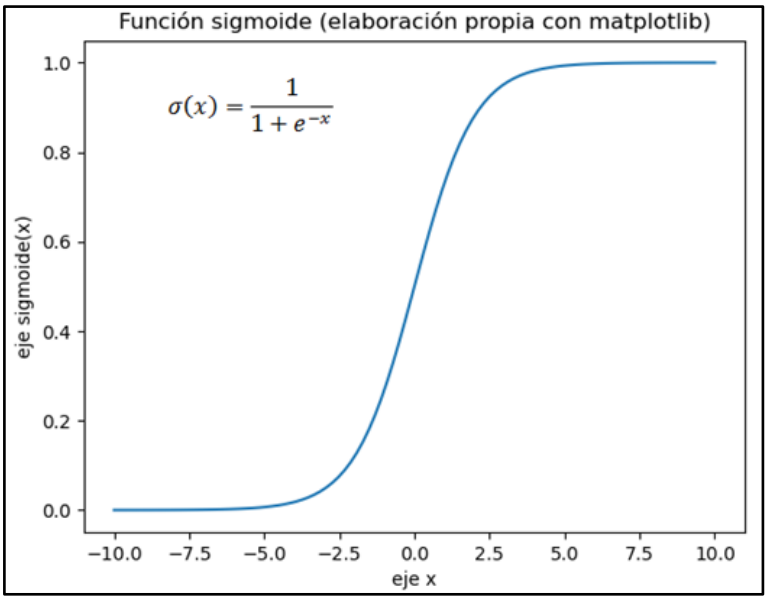


Fig. 7. Función sigmoide. Fuente: (elaboración propia)

Las neuronas biológicas tienen muchas entradas, las cuales se combinan mediante una suma ponderada, y a su vez esta se le aplica la función de activación (en este caso la función sigmoide). Si la señal que produce dicha combinación es lo suficientemente fuerte para superar cierto umbral (aplicado por la función de activación), entonces la neurona se activará intensamente, de otra manera dicha señal será suprimida en cierta medida (Shouhua & Jingying, 2009).

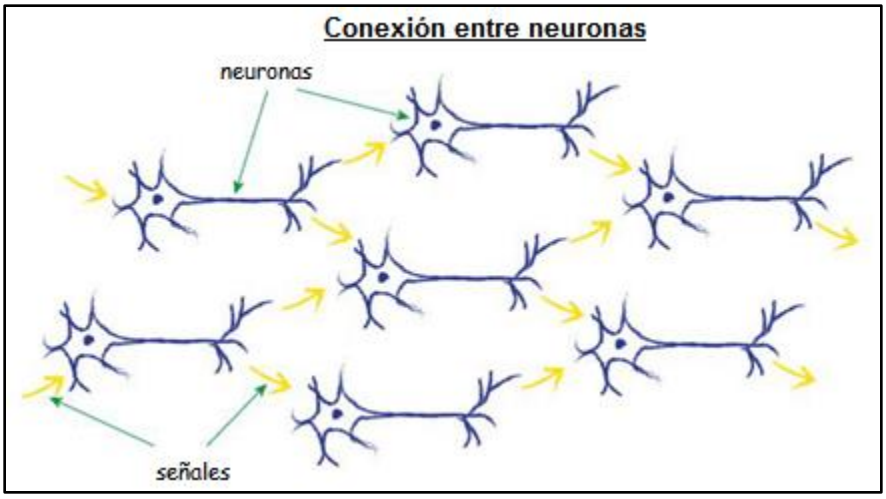


Fig. 8. Conexión entre neuronas. Fuente: (Rashid, 2016)

Dicha señal es a su vez transmitida entre varias neuronas interconectadas en un proceso llamado sinapsis como se ve en la Fig. 8 La forma en que se replica esto es través de capas, cada una conectada con la capa anterior.

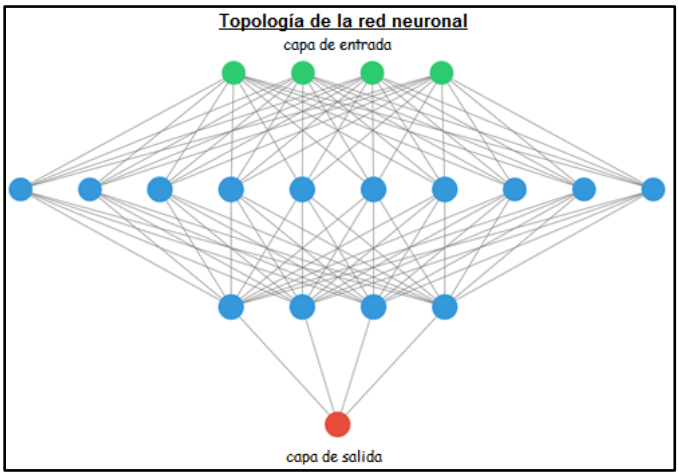


Fig. 9. Topología de la Red Neuronal. Fuente: (elaboración propia)

Las capas se describen en tres tipos, una capa de entrada, como la que hay al inicio, las capas ocultas que son las capas intermedias y la salida que se encuentra al final, en la Fig. 9 son la verde, azul y roja respectivamente.

Es decir, cada capa contiene diferentes neuronas o nodos los cuales se interconectan a cada uno de los nodos de la capa siguiente (capa de entrada), de la capa anterior (capa de salida) o ambas (capas intermedias) (Rashid, 2016).

Las líneas que conectan los nodos representan los pesos con los cuales se pondera la suma de cada una de las entradas de la capa anterior. Las cuales dependiendo del tamaño del peso pueden amplificar o disminuir la señal; típicamente estos pesos inician de manera aleatoria y van cambiando y ajustándose conforme empiezan a aprender (Du & S. Swamy, 2019). Dichas operaciones entre pesos y entradas se pueden compactar en la Ecuación 2)

$$\boldsymbol{X} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,k} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j,0} & w_{j,1} & \cdots & w_{j,k} \end{bmatrix} \begin{bmatrix} i_0^0 \\ i_1^0 \\ \vdots \\ i_n^0 \end{bmatrix} \rightarrow \boldsymbol{X} = \boldsymbol{WI}$$

Ecuación 2. Forma Algebraica de una Red Neuronal

Y aplicar la función de activación (sigmoide por poner un ejemplo) para obtener la Ecuación 3):

$$\boldsymbol{O = sigmoide(X)}$$

Ecuación 3. Activación con la Función Sigmoide

Donde cada uno de los elementos representa:

$\boldsymbol{w_{j,k}}$: Es el peso desde la capa **j** hacia la capa **k**

$\boldsymbol{I_n^k}$**:** Es cada entrada del nodo **n** a la capa **k**

$\boldsymbol{W}$: Representa la matriz de pesos

$\boldsymbol{I}$: Representa el vector de entradas

Una vez que generamos los valores de salida podemos comparar cómo el método estimó los resultados esperados, a través de su diferencia, a esto se le llama función de error o función de costo, la cual buscaremos minimizar ya que representa que tan mal se desempeñó (McClelland, Rumelhart, & Hinton, 1987), como se ve en la Fig. 10.

No obstante hacerlo de esta manera hace que al sumar los errores de cada uno de los nodos de salida los valores positivos se cancelen con los negativos, provocando un resultado que no refleja el desempeño de la red, otra manera de evaluar esto es a través del valor absoluto, tal como se muestra en el cuadro intermedio de la misma figura, con esto se solucionan las cancelaciones por problema de signo.

Sin embargo, es importante recordar que la función valor absoluto está sujeta a discontinuidades en ciertos puntos, haciendo que la función no sea diferenciable (Rudolf, Borgelt, & Braune, 2016), lo cual es un problema ya que más adelante se emplearán derivadas parciales.

La solución se encuentra en la forma cuadrática, es decir en la última opción de la **Fig. 10**. Con esto se evita la cancelación de errores del primer ejemplo y a diferencia del segundo, es diferenciable (Du & S. Swamy, 2019).

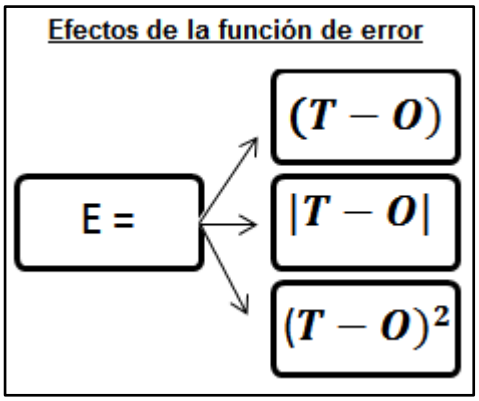


Fig. 10. Efectos de la función de error. Fuente: (elaboración propia)

Hasta el momento lo que se ha hecho es usar pesos para propagar señales hacia adelante desde la capa de entrada hasta la capa de salida de la red neuronal. Lo siguiente es usar esos pesos para propagar el error hacia atrás desde la salida hacia la red hasta los pesos entre la primer y segunda capa, a este método se le conoce como propagación hacia atrás (Farber, 1988).

La forma ideal de hacer esto sería propagar el error de cada nodo en base a la proporción de cada uno de sus pesos asociados y con la ayuda del álgebra resolver este problema, sin embargo, la expresión llega a ser demasiado compleja y poco práctica computacionalmente hablando (Rashid, 2016).

El enfoque que se toma es que no podemos encontrar una solución, no obstante, podemos aproximarla mediante una técnica llamada Descenso del Gradiente (la cual se aprecia en la ecuación 3). La idea de este método trata de que en vez de resolver algebraicamente para encontrar el mínimo se haga una estimación mediante un proceso iterativo reduciendo y llegando a un mínimo de la función de costo (la suma de los errores de cada neurona) (McClelland, Rumelhart, & Hinton, 1987).

$$w_{nuevo} = w_{antiguo} - \alpha \frac{\delta E}{\delta w}$$

Ecuación 4. Forma Algebraica del Descenso del Gradiente

Dónde:

- E: Es la función de error (o costo).

- w: es el peso de un entre dos nodos.

- α: Es la tasa de aprendizaje.

La forma de encontrar este mínimo es reduciendo cada uno de los pesos en la dirección donde se encuentra el descenso más empinado, dando cada vez pasos más pequeños en cada iteración mediante un coeficiente llamado tasa de aprendizaje (esto se puede apreciar gráficamente en la **Fig. 11**), este se multiplica por la derivada parcial del nodo correspondiente, el proceso se repite hasta llegar a cierto umbral o hasta terminar un número fijo de iteraciones (Shouhua & Jingying, 2009).

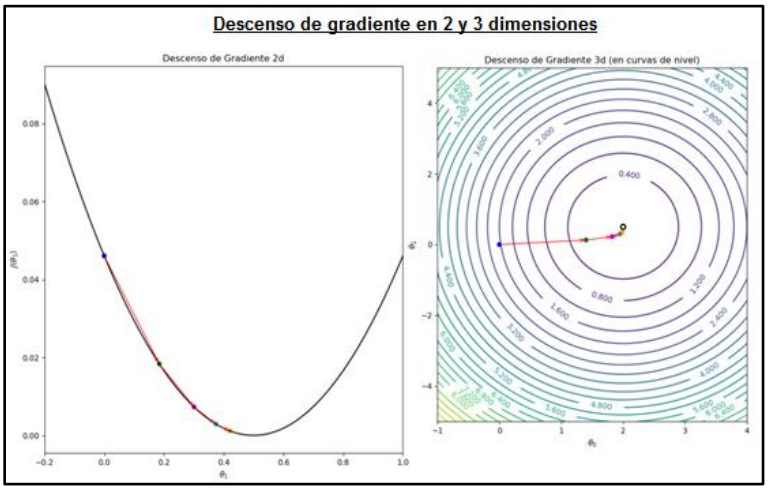


Fig. 11. Descenso de gradiente en 2 y 3 dimensiones Fuente: (elaboración propia)

La importancia de la tasa de aprendizaje radica en que el tamaño de las reducciones puede afectar el proceso de aprendizaje, tal como indica la Fig. 12 por un lado una tasa demasiado alta puede hacer que se alcance el mínimo varias veces pero que no se pueda llegar ahí, por otro lado, una tasa demasiada baja puede hacer que tarde más en converger dicho valor (Shouhua & Jingying, 2009).

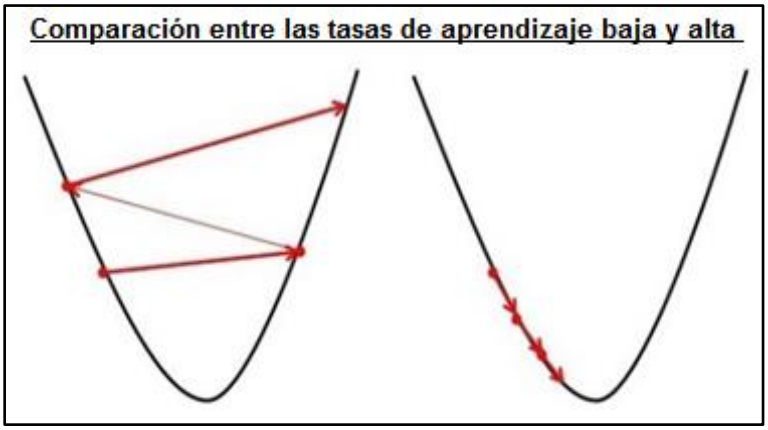


Fig. 12. Comparación entre tasa de aprendizaje alta y baja. Fuente: (Nagy et al., 2019)

Existen algunas consideraciones que deben tomarse en cuenta a la hora de preparar los datos y que pueden afectar los resultados si no se tienen en cuenta.

Uno de estas es el hecho de que si las entradas son muy grandes el gradiente será muy pequeño y será más lenta la convergencia a un valor, un fenómeno que se conoce como saturación prematura (Du & S. Swamy, 2019), por otro lado, si dichos valores son demasiados pequeños se pierde precisión; una recomendación es reajustar los valores entre 0.1 y 1.0 (se evita usar 0.0 ya que al hacer esto se acaba con el proceso de aprendizaje).

En el caso de las salidas hay que tomar en cuenta que si se usa la función sigmoide sus salidas se encuentran en el rango (0,1) lo que significa que no pueden tomar valores superiores o inferiores a ese rango (tampoco los mismos, ya que se trata de un rango abierto), por lo que necesario reajustar los valores deseados para que puedan ser comparados con las salidas (Rashid, 2016).

Los pesos iniciales se establecen de manera aleatoria, pero es necesario tomar ciertas consideraciones, por ejemplo, para evitar la saturación de la red neuronal podemos usar valores entre -1 y 1; otra medida importante es utilizar diferentes valores de entrada de forma aleatoria para reducir la posibilidad de caer en un mínimo local (Denz, 2013).

Con la topología que se ha establecido actualmente no es posible analizar series de tiempo, ya para estas últimas los estados anteriores es relevante, en ANNs esto se ha logrado con una capa adicional que almacena la información del estado anterior, la cual se le conoce como capa recurrente.

Existen diferentes variantes de la RNN, las más populares son la de *Elman* y la de *Jordan*, las cuales son diferentes por el nodo del cual almacenan la información, como se aprecia en la **Fig. 13**, la red de *Elman* toma los valores de la capa oculta, mientras que la de *Jordan* lo hace de la capa de salida.

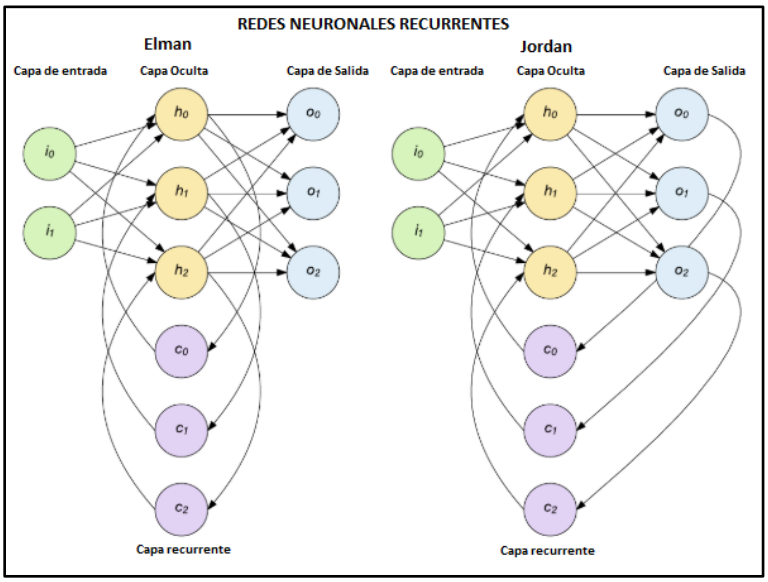


Fig. 13. Variantes de RNN. Fuente: (Jones, 2017)

El hecho del gran parecido en ambas redes hace que de manera muy habitual se busque comparar resultados en estas estructuras, sin embargo no hay información teórica concluyente que apoye esta idea, lo que sí existe son resultados empíricos que soporten la declaración de que en series de tiempo en las redes de tipo Elman el comportamiento el comportamiento sea mejor que en redes de tipo Jordan (Abdulkarim, 2016).

2.3. Proceso para predicción en Aprendizaje Automático.

Las ANNs, al ser una técnica de aprendizaje automático, parten de los mismos principios que este último (generar un comportamiento a través de los datos en lugar de programarlo explícitamente), por lo mismo a grandes rasgos la metodología es la misma (con particularidades que se irán detallando en secciones posteriores).

Por el momento podemos resumir al Aprendizaje Automático en tres cosas:

1. **Datos de entrada.** Pueden ser archivos de audio, imágenes, texto, videos, etc.

2. **Datos de salida esperados.** Un claro ejemplo son etiquetas ingresadas por una persona, como la descripción de cierto contenido.

3. **Una forma de retroalimentar.** Medidas que nos permitan saber qué tan alejado está el modelo del resultado que se espera, haciendo una comparación entre los datos de salida del modelo y los datos esperados.

Un buen modelo de aprendizaje automático busca, con útiles representaciones de datos de entrada, hacer transformaciones que generen salidas muy cercanas a las esperadas (Chollet, 2018).

Aquí se puede hacer énfasis a tres términos:

1. ***Representaciones:*** quiere decir las diferentes formas de ver a los datos, después se sabrá que antes de transformar habrá algunas formas que servirán más que otras dependiendo del contexto y del tipo de transformación a elegir.

2. ***Transformaciones:*** son las operaciones que nos permitirán convertir una entrada a una salida, la cual buscamos nos acerque más a la salida que esperamos.

3. ***Salidas:*** son los resultados que compararemos y que nos ayudarán junto con la métrica que elijamos para retroalimentar el proceso e ir acercándonos cada vez más a nuestra meta.

Dicho con otras palabras, el objetivo de cualquier algoritmo de aprendizaje automático es encontrar automáticamente transformaciones que conviertan los datos en representaciones útiles para una tarea específica.

Sin embargo lo que ha hecho tan popular a las RNAs, además de los resultados que han logrado en varios campos, son las siguientes características:

a) **Simplicidad.** Elimina en gran medida la necesidad de lo que se conoce como *Feature Engineering* (diseño de características), lo que es básicamente el proceso de convertir los datos brutos en representaciones que pueda entender un algoritmo en específico.

b) **Escalabilidad.** Es muy útil para paralelizar procesos, lo que permite agilizar el entrenamiento.

c) **Versatilidad y Reusabilidad.** El entrenamiento puede ser incremental, es decir, no hay que iniciar desde cero para retomar el entrenamiento anterior.

2.4. Proceso para predicción en RNAs.

2.4.1. Conceptos Básicos

Primero hay que definir qué significa “aprendizaje” en el contexto de RNA, lo cual se puede resumir como: “Encontrar en un modelo una combinación de parámetros que se ajusten a un conjunto muestras de datos de entrenamiento y sus correspondientes objetivos” (Chollet, 2018), a continuación se ejemplificará con un diagrama.

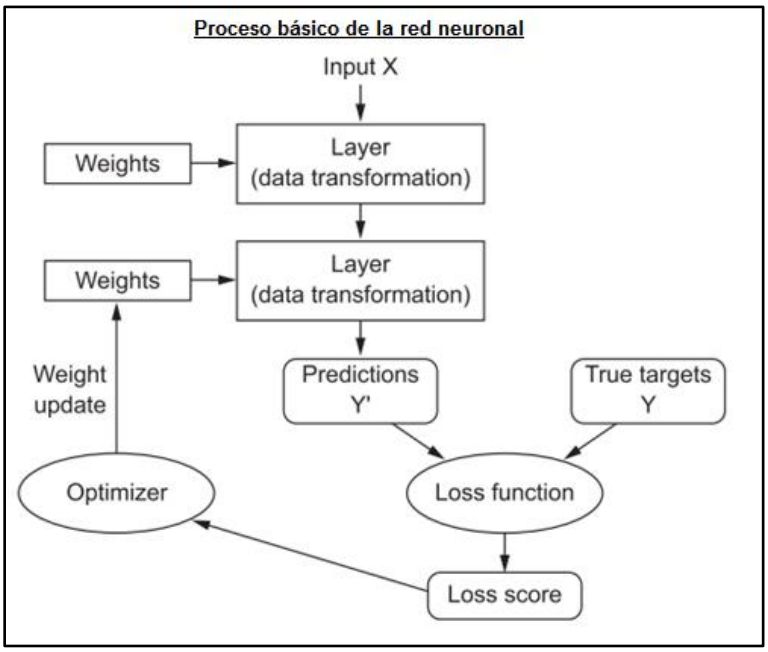


Fig. 14. Proceso básico de la red neuronal. Fuente: (Chollet, 2018)

Aquí podemos destacar cuatro elementos clave:

a) **Capas (***layers***).** Juntos conforman la red o modelo.

b) **Datos de entrada** (*inputs*)**, datos de salida** (*predictions/outputs*) y sus **metas** (*targets/labels*)**.**

c) **Función de error (***loss function***)**, para medir qué tan alejados estamos del resultado deseado.

d) **Optimizador (***optimizer***)**, que nos dice cómo aprender si necesitamos mejorar.

Al hablar de “muestras de datos de entrenamiento”, nos referimos a las entradas (*inputs*) del diagrama, los cuales como habíamos dicho tienen que ser *representaciones útiles*, sin embargo no hemos dicho qué es una representación útil dentro de la RNA.

La estructura de datos básica en redes son los tensores, los cuales son contenedores de datos (arreglos numpy o tensores tensorflow) que casi siempre son numéricos. El proceso de convertir nuestros datos a arreglos (tensores) se le conoce como *vectorización*, el cual puede tener múltiples dimensiones o ejes (Josh Patterson, 2017).

Para ejemplificar podemos decir que un escalar es un tensor de cero dimensiones, un vector es un tensor de una dimensión, una matriz un vector de dos dimensiones y así sucesivamente, también es bueno destacar algunas propiedades útiles de los tensores como lo son: el número de ejes, la forma y los tipos de datos.

Un punto muy importante es que la mayoría de los datos los podemos representar como un tensor, lo cual varía dependiendo del tipo de datos que sean (Josh Patterson, 2017), tales son los casos de:

a) **Datos vectores.** Se representan como tensor 2D (matriz)

b) **Datos en series de tiempo o en secuencia.** Tensor 3D.

c) **Imágenes.** Tensor 4D.

d) **Videos.** Tensor 5D.

Vale la pena resaltar el que en el "*caso b"* (series de Tiempo, el problema en cuestión de esta tesis) este puede ser visualizado como una colección de puntos (muestras) donde cada punto contiene un grupo de características que cambian cada cierto periodo.

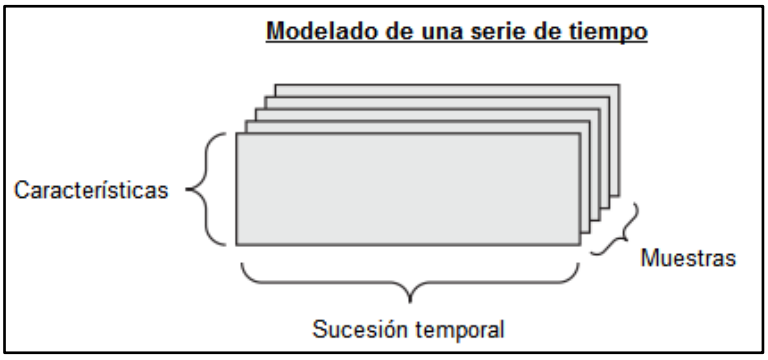


Fig. 15. Modelado de una serie de tiempo. Fuente: (Chollet, 2018)

Otros aspectos importantes son las operaciones que realizan los tensores como operaciones *element-wise*, *broadcasting*, producto punto, *reshaping*, entre otros (McKinney, 2017). Gracias a los cuales podemos generar *predicciones* (*predictions*) que podrán compararse con los *objetivos* (*labels*) y a su vez serán los parámetros de una función de pérdida (*loss function*) que nos arrojará una puntuación (*loss score*) la cual si disminuye deberá verse reflejada en una medida de éxito (*metric*).

Si dicho *loss score* no es el adecuado de acuerdo a cierto umbral, un algoritmo optimización (*optimizer*) modificará los parámetros iniciales (*weights*) de manera que durante cada iteración exista una mejoría. Cabe mencionar que tanto la *loss function* como el *optimizer* tienen diferentes variantes, mínimos cuadrados (mse) o *cross entropy* son ejemplos de *Loss Function* y *RMSProp* o *Adam* son ejemplos de *optimizers* (que no son sino variantes del descenso de gradiente estocástico visto anteriormente), aunque RMSProp es más recomendado para uso general (Chollet, 2018).

Ya hemos hablado de las capas (*layers*) como parte del proceso de aprendizaje profundo, sin embargo hay que resaltar su importancia como estructura fundamental en ANNs, podemos pensar que son funciones o módulos que toman uno o varios tensores de entrada y regresan uno o varios tensores de salida y la mayoría tienen un estado: los pesos de la capa.

Existen diferentes tipos de capas: las Densas (*Dense Class*), Recurrentes (como *SimpleRNN, LSTM*), *Flatten*, Embebidas (*embedding class*), etc., las cuales pueden apilarse una tras otra, siempre y cuando la salida de la anterior sea compatible con la entrada de la siguiente o incluso pueden hacerse capas personalizadas no lineales (Chollet, 2019).

2.4.2. Conceptos sobre Keras

Keras es una biblioteca que puede considerarse de alto nivel, ya que maneja bloques para desarrollar modelos de aprendizaje profundo, no maneja operaciones de tensores, al menos no directamente, para ello utiliza otras bibliotecas de soporte (como TensorFlow, Theano o CNTK), esto hace que sea más fácil usar keras que hacerlo directamente que con las bibliotecas mencionadas.

El flujo de trabajo de keras es el siguiente:

a) **Definir tus datos de entrenamiento**: tensores de entrada, tensores objetivo.

b) **Definir la red de capas (modelo).** Que mapee las entradas a los objetivos.

c) **Configurar el proceso de aprendizaje.** Eligiendo una función de pérdida, un optimizador y algunas métricas para monitorear. Usando el método *compile()*.

d) **Iterar sobre los datos de entrenamiento.** Usando el método fit() en dicho modelo.

Existen tres formas de definir un modelo, con la API *Sequential* (apilar linealmente las capas, lo cual es lo más común), usar la API *functional* (lo cual permite mayor libertad al definir el acomodo de las capas, pero requiere un dominio más avanzado), y hay una tercera forma aún más avanzada, definiendo clases y objetos. Una vez definido cualquier forma, lo siguiente ya no variará, independientemente como cuál método se haya hecho.

Para configurar el proceso de aprendizaje basta con usar el método *compile* e ingresar a los parámetros las funciones de optimización, pérdida y las métricas a usar. Finalmente se usa el método *fit()* y se ingresan los datos de entrada y los datos objetivo (*labels*), todo en forma de tensores.

2.4.3. Evaluar conjuntos de datos: Entrenamiento, Validación y Prueba

Se puede considerar que el objetivo de los modelos de Aprendizaje Automático (incluido el Aprendizaje Profundo) es incrementar la capacidad para *generalizar*, es decir poder predecir resultados con datos de entrada los cuales nunca ha visto dicho modelo.

Para ello existen dos fenómenos que deben superarse: subajuste y sobreajuste, el primero se da por no tener suficientes datos o el modelo adecuado para generalizar y el segundo por una excesiva exposición a un conjunto determinado de ejemplos que generaliza características muy específicas de dichos ejemplos como si se fueran a encontrar en cualquier dato, en ambos casos el poder de predicción se ve afectado.

Por lo que usar un solo conjunto para entrenar y probar es una mala práctica debido a que uno no puede estar seguro si el modelo simplemente memoriza los ejemplos o si realmente sabe generalizar (predecir); esta memorización se ve reflejada en los parámetros que ajusta el modelo (es decir los pesos).

Hasta el momento podría decirse que el problema está resuelto, no obstante hay que tomar en cuenta en que la búsqueda de mejorar las predicciones no solo es el ajuste de los pesos, otra actividad es ajustar parámetros para mejorar el rendimiento, como el número de capas y el tamaño de estas (conocidos como hiperparámetros), por lo que podemos caer en un segundo sobreajuste.

Dicho sobreajuste ahora surge por modificar constantemente la configuración de estos hiperparámetros, lo cual hace que el modelo funcione bien con los datos de validación pero no con datos nuevos, para ello se crea un nuevo conjunto, los datos de prueba y con eso se resuelve el problema.

Existen diferentes técnicas para particionar conjuntos, dependiendo de la situación es mejor elegir una técnica sobre otras.

1. **Validación Hold-out Simple**. Consiste en apartar del total de datos un conjunto de datos para testeo, dejando el resto para datos de entrenamiento y que dentro del mismo se está contando los datos de validación. Sin embargo nos podemos enfrentar a un problema, si los datos son muy pocos, los conjuntos de validación y prueba pueden ser tan pequeños que no sean estadísticamente representativos (Shmueli, 2010).

   Esto puede comprobarse si elegimos diferentes rondas aleatorias de datos antes de dividirlos, si dicho modelo resulta, tener medidas muy diferentes en cada ronda, es decir varianza alta, entonces existe dicho problema. Por lo que este método no sería el ideal, pero existen diferentes alternativas, como las siguientes.

2. **Validación *K-Fold*.** En este método dividimos las muestras en K particiones, para cada iteración ***i***, utilizamos las K – 1 restantes como conjunto de entrenamiento, y la partición ***i*** como conjunto de prueba, así repetimos K veces y obtenemos K resultados diferentes, los cuales al obtener su promedio tendremos un resultado más representativo (Sugiartawan & Hartati, 2019).

3. **Validación *K-Fold* iterativa modificando el orden.** Consiste en aplicar la validación *K-Fold* K veces como en el método anterior, la diferencia es que se repetirá este procedimiento P veces y en cada una de esas veces se modificará el orden de las muestras. (el promedio será entonces la suma de todos los elementos sobre el K por P veces).

3.2.4. Preprocesamiento de datos

Se había comentado anteriormente que es necesario preparar los datos brutos de entrada antes de ingresarlos a una red neuronal, dependiendo del tipo de datos que estemos manejando se requerirá hacer determinadas técnicas, las cuales se cubrirán en secciones posteriores, por ahora se mostrará solo de manera general.

a) **Vectorización.** Todas las entradas y valor objetivo de una red neuronal deben estar en términos de tensores con datos de punto flotante (o enteros en algunos casos), sin importar el tipo de dato (sonido, imagen, texto) la mayoría pueden representarse en términos de tensores.

b) **Normalización (tipificación).** Muchas veces cuando exista información con valores que representan diferentes características a diferentes escalas, se recomendará que los datos tengan una media de 0 y una desviación estándar de 1, por las siguientes razones.

- **i.** **Que los datos sean homogéneos.** Todas las características deberían tomar casi el mismo rango.
- **ii.** **Que los datos sean pequeños.** Datos más pequeños es más fácil que converjan.

c) **Valores faltantes.** Poner cero a un valor faltante es una práctica recurrente y segura, siempre y cuando esta no tenga un significado especial; la red aprenderá entonces que el cero significa “valor faltante”.

Sin embargo hay que considerar que si tenemos valores en el conjunto de entrenamiento pero no contamos con su correspondiente en el conjunto de prueba es mejor eliminarlos los del conjunto de entrenamiento también, ya que entonces la red no aprenderá ignorar esos valores.

d) **Diseño de características.** Se dijo anteriormente que las ANNs eliminan en gran medida la necesidad del diseño de características, pero en ciertas situaciones resulta conveniente, ya que le hacemos más fácil la labor al modelo, incluso puede ayudar cuando no disponemos de muchos datos disponibles y eliminar el sobreajuste del que se hablará a continuación.

3.2.5. Subajuste y Sobreajuste

Ya se había comentado antes que el sobreajuste es el principal problema con el que se lidia en Aprendizaje Automático, dicho problema se origina por la relación entre la *optimización* y la *generalización*, la primera refiriéndose al rendimiento que se obtiene con el conjunto de entrenamiento, y la segunda con respecto a datos nunca vistos o sea el conjunto de prueba (Chollet, 2018).

El objetivo es poder *generalizar*, pero esto no puede controlarse directamente, hay que recordar que no podemos hacer mejoras en base a los resultados del conjunto de prueba, si lo hacemos de esta manera pierde esa propiedad para evaluar las mejorías, por lo que lo único que se puede hacer es ajustar en base al conjunto de entrenamiento (o al de validación si tratamos con hiperparámetros).

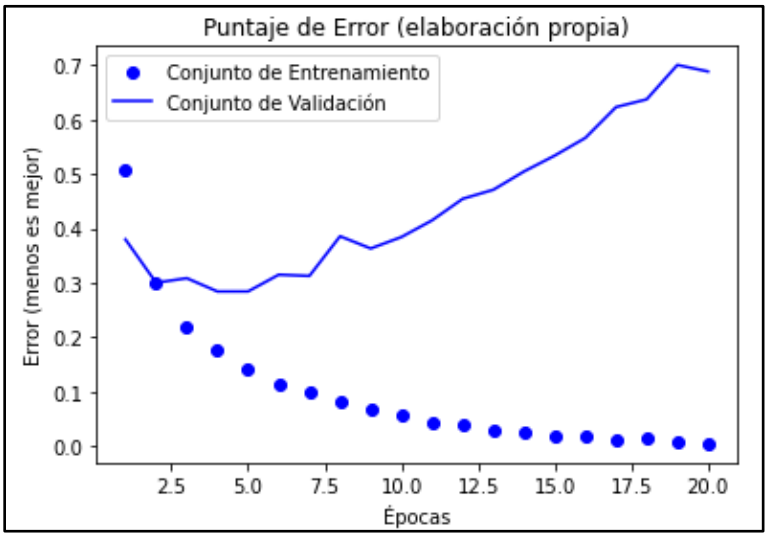


Fig. 16. Representación de un Proceso de Aprendizaje de ANN. Fuente: (Elaboración Propia)

En un inicio los procesos de optimización y generalización están correlacionados, si disminuye puntación de error en el conjunto de entrenamiento también lo hace en el conjunto de validación, a esto se le llama subajuste, un punto donde todavía hay mejoría.

No obstante llega un punto en el conjunto de validación se detiene y comienza a desmejorar (subir en este caso), mientras que en el conjunto de entrenamiento sigue disminuyendo, esta es la señal que nos indica que ya estamos en un punto de sobreajuste, donde el modelo está no aprendiendo características generales sino más bien específicas de ese grupo de datos.

Al proceso de eliminar el sobreajuste se le conoce como *regularización*, en dicho proceso nos encontramos con diferentes técnicas:

a) **Aumentar la cantidad de datos de entrenamiento.** Un modelo que se entrena con más datos puede generalizar mejor.

b) **Reducir el tamaño de la red.** Aumentar una red significa aumentar su capacidad de memorización, sin embargo esto debe ser compensado con una cantidad suficiente de datos, ya que una memorización excesiva nos lleva a un desempeño pobre para generalizar resultados.

c) **Regularizar los pesos de la red.** Si trabajamos con valores más pequeños es menos probable caer en el sobreajuste, por lo mismo si agregamos una penalización a pesos altos contaremos con valores más pequeños.

d)

3.2.6. Reuniendo todos los conceptos

a) **Definir el problema y construir el conjunto de datos.** ¿Qué tipo de datos de entrada serán? ¿Qué es lo que se quiere predecir? ¿Se cuentan con datos disponibles y son suficientes? ¿Cuál es el problema que se va a resolver? (*en el caso de esta tesis hablamos de regresión escalar*).

   En este punto estamos asumiendo dos cosas, primero que nuestros datos de salida pueden ser predichos con nuestros de entrada y segundo que contamos con suficiente información para establecer la relación entrada-salida.

b) **Elegir la métrica de éxito.** Para el caso de regresión podríamos elegir entre el Error Cuadrático Medio o el Error Absoluto Medio.

c) **Elegir el protocolo de evaluación.** Se trata de elegir entre el método *hold-out*, *K-Fold* o *K-Fold Iterado* que ya se mencionaron anteriormente.

d) **Preparando los datos (Preprocesamiento).** Es convertir los datos a tensores, escalar esos tensores de manera que tomen valores pequeños (tipificación), homogeneizar datos y hacer diseño de características.

e) **Desarrollar un modelo sencillo, pero mínimamente funcional.** Esto con la idea de saber si es posible construir de moderado pero con éxito (que tenga poder estadístico) y continuar con un modelo de mayor escala o si es necesario replantear algo en los pasos anteriores.

   Si este pequeño modelo es exitoso tendremos que hacer tres acciones.

   - **Activación de la última capa.** Es útil porque establece a través de restricciones la forma final de los datos, típicamente en problemas de regresión sin restricción de valor no se ingresa ninguno o en el caso de restricción de valor de 0 a 1 se utiliza la función sigmoide.

   - **Función de Pérdida.** Es necesario elegir la indicada de acuerdo al tipo de problema que nos encontramos, siendo el más común para regresión el error cuadrático medio (MSE)

   - **Ajustes del optimizador.** Consiste la elección del optimizador (generalmente *RMSprop* o *Adam* es ideal en la mayoría de los casos) y la tasa de aprendizaje (un valor entre 0 y 1).

f) **Desarrollar un modelo que provoque sobreajuste.** En el paso anterior se buscó que el modelo tuviera poder estadístico, sin importar si fuese poco, no obstante ahora se buscará maximizar dicho poder, para ello se buscará llevar al límite el modelo hasta provocar un sobreajuste.

   Hay que recordar que el problema universal del aprendizaje automático es lograr un equilibrio entre optimización y generalización, mismo que se encuentra en el umbral entre el subajuste y el sobreajuste, para esto podemos hacer lo siguiente:

   - **Agregar capas.**

   - **Agrandar capas.**

   - **Usar más épocas.**

   La idea es siempre monitorear el puntaje de error, tanto en entrenamiento como en validación, cuando se aprecie que los datos de validación comienzan a degradar (es decir, aumentar el puntaje de error), se ha llegado a sobreajuste (como se vio en la Fig. 16).

g) **Regularizar el modelo y ajustar hiperparámetros.** En este punto se buscará combatir el sobreajuste; para eso se usarán las técnicas de regularización antes mencionadas (aumentar el tamaño del conjunto de entrenamiento, regularizar el tamaño de la red, regularizar los pesos y la técnica de desprendimiento). Las técnicas de diseño de características también pueden ser útiles.

También es posible hacer uso del ajuste de hiperparámetros (por ejemplo ir cambiando el número de unidades por capas o la tasa de aprendizaje del optimizador), pero es importante no abusar demasiado, hacerlo demasiadas veces puede llevar a un sobreajuste con respecto al conjunto de validación.

Una vez desarrollada esta configuración solo queda entrenar el modelo final con todos los datos disponibles (entrenamiento y validación) y evaluarlo por última vez con el conjunto de prueba.

2.5. ¿Cómo preparar una predicción en RNA de Series de Tiempo?.

Siguiendo con el marco conceptual, el siguiente espacio nos ayudará a la elaboración de la metodología, hay que recordar que esta debe ir encaminada al objetivo general y esta a su vez a los objetivos específicos los cuales son:

“Construir una Red Neuronal Recurrente *Elman* que pueda superar en resultados al modelo predictivo SARIMA usando la información disponible en el SNIIM”.

Y los específicos:

1. “Determinar la variante de red neuronal que mejor se acople al problema de series de tiempo del mercado agrícola”.

2. “Determinar los datos que pueden modelarse del sistema nacional de información e integración de mercados de la Secretaría de Economía (establecer el conjunto para entrenar y para evaluar)”.

3. “Construir el modelo RNN”.

4. “Medir los resultados a un criterio de evaluación apropiado”.

2.5.1. Convertir arreglos a lotes de ventanas.

Para cumplir nuestro objetivo de análisis es necesario el desarrollo de un código (en este caso Python con bibliotecas adicionales) cuya finalidad sea construir ventanas de tiempo a partir de datos originales a través de los cuales se generen predicciones, dichos datos tienen la siguiente apariencia:

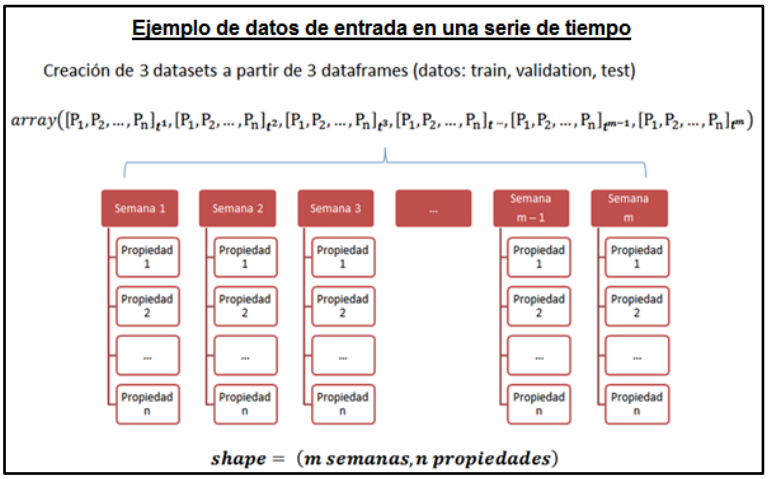


Fig. 17. Ejemplo de datos de entrada en una serie de tiempo. Fuente: (Elaboración Propia)

En la parte superior de la Fig. 17 podemos ver los datos de manera estructurada como un arreglo de *n propiedades* que se van midiendo a través del tiempo hasta el *tiempo m*; en este caso particular hablamos de semanas como medida de tiempo pero se puede generalizar a cualquier otra medida.

En la parte inferior de la misma figura se ve una representación gráfica de dicho arreglo, aquí también se aprecia que dicha estructura se ve de una forma de *m semanas por n propiedades* donde el convencionalismo es que lo más general (en este caso las semanas) se ordena a la izquierda y lo más particular (en este caso las propiedades) a la derecha.

Aquí hay que recalcar que mucho del código debe estar construido para dar cierta flexibilidad, es decir que para variar los parámetros de manera sencilla, y que genere nuevos análisis sin tantas líneas nuevas, aunque estandarizar ciertas variables también puede ser conveniente.

A diferencia de los problemas de clasificación en aprendizaje automático, en donde de manera explícita existe un conjunto de datos de entrada (*input*) y conjunto de etiquetas (*label*), en series de tiempo es necesario definir esta relación por cada ventana con el mismo conjunto de datos (Wu et al., 2020).

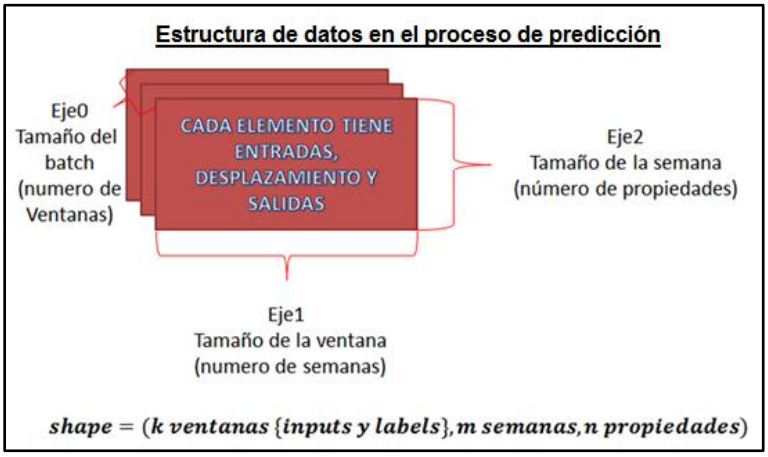


Fig. 18. Estructura de datos en el proceso de predicción. Fuente: (Elaboración Propia)

Vale la pena recordar que cuando se modela series de tiempo con tensores típicamente se usan tres ejes, donde el eje 0 es el eje del *batch* o lote, el eje 1 es el tiempo (u orden en el caso de lenguaje natural) y el eje 2 son las características (o propiedades), respetándose el convencionalismo de generalidad del lado izquierdo y particularidad en el derecho.

Dicho lo anterior los datos originales serán la materia prima para generar ventanas y guardarlas en un lote de ventanas con una estructura de datos especial y óptima para ser procesada (un *dataset* de TensorFlow que contendrá tensores), y a su vez estos *datasets* serán las entradas del modelo de predicción.

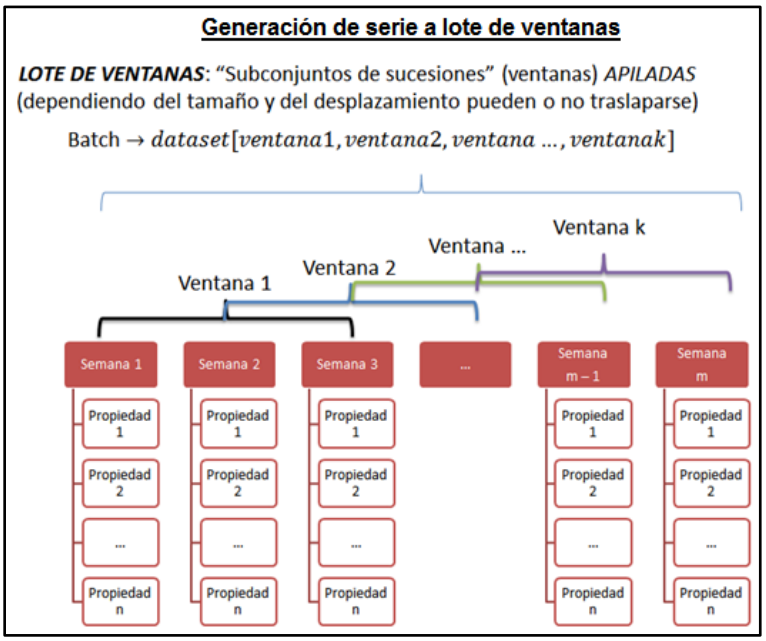


Fig. 19. Generación de serie a lote de ventanas. Fuente: (Elaboración Propia)

Podemos definir a las *propiedades* como cada una de las características a observar y medir durante uno o varios periodos de tiempo (regulares o irregulares). Recordar que cada elemento de manera individual se guarda en un registro y que los componentes de estos registros son lo que conocemos como "propiedades o atributos" (Saumyendra Sengupta, 1994).

Es por eso que en el proceso para transformar un conjunto de propiedades ordenadas por el tiempo a ventanas de tiempo con entradas y etiquetas es necesario desarrollar una función con tal lógica, afortunadamente la biblioteca de TensorFlow contiene una función predefinida que permite hacerlo solo llenando unos parámetros, esto nos facilitará el trabajo al integrarlo con el resto del código.

En la Fig. 19 podemos ver de manera gráfica cómo un arreglo de datos originales se puede convertir en un lote de k ventanas (en este ejemplo definimos ventanas de tres semanas por mera conveniencia, al no ser muchas y ejemplificar con precisión). Hay que recordar que debe existir una cantidad suficiente de datos dependiendo del tamaño total de la ventana y que cada ventana a su vez debe contener un conjunto datos de entrada y su etiqueta subyacente, de aquí surge la necesidad de crear otra función que se dedique a dividir (o rebanar) la ventana en dos conjuntos.

Para ponerlo más claro se establece un ejemplo gráfico con la transformación de una ventana, de un grupo de k ventanas, ingresando la ventana de 6 semanas y tres propiedades, como se ve a continuación:

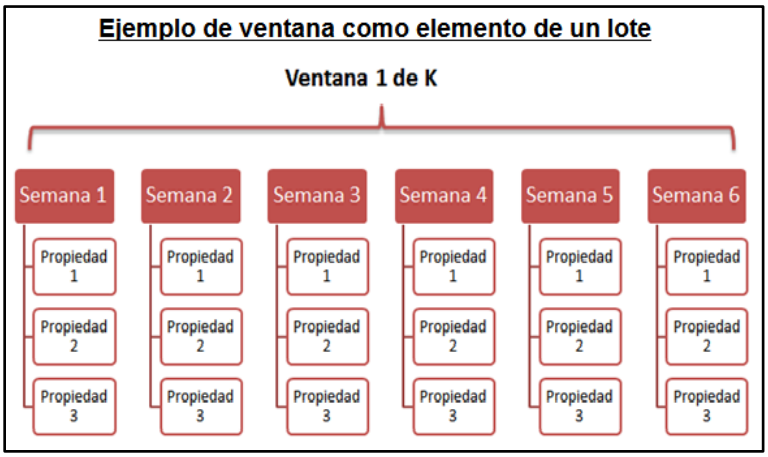


Fig. 20. Ejemplo ventana como elemento de un lote. Fuente: (Elaboración Propia)

Tiene sentido que si nuestra ventana tiene un tamaño de 6 semanas, es necesario tener al menos 6 semanas de datos para crear solo el primer elemento, y puesto que para realizar una estimación se requerirán varias ventanas entonces se necesitará un número mucho mayor de semanas.

Factores como el desplazamiento también tienen relación en la generación de ventanas; a mayor desplazamiento menor es el número de ventanas posibles para un número de semanas determinado; incluso si el desplazamiento es mayor también se afecta a otros factores como la incertidumbre (a mayor desplazamiento mayor incertidumbre).

Si definimos las primeras tres semanas como entrada, las siguientes tres como desplazamiento y retrocediendo última semana como etiqueta con la primera propiedad solamente el gráfico quedaría de la siguiente manera (Fig. 21):

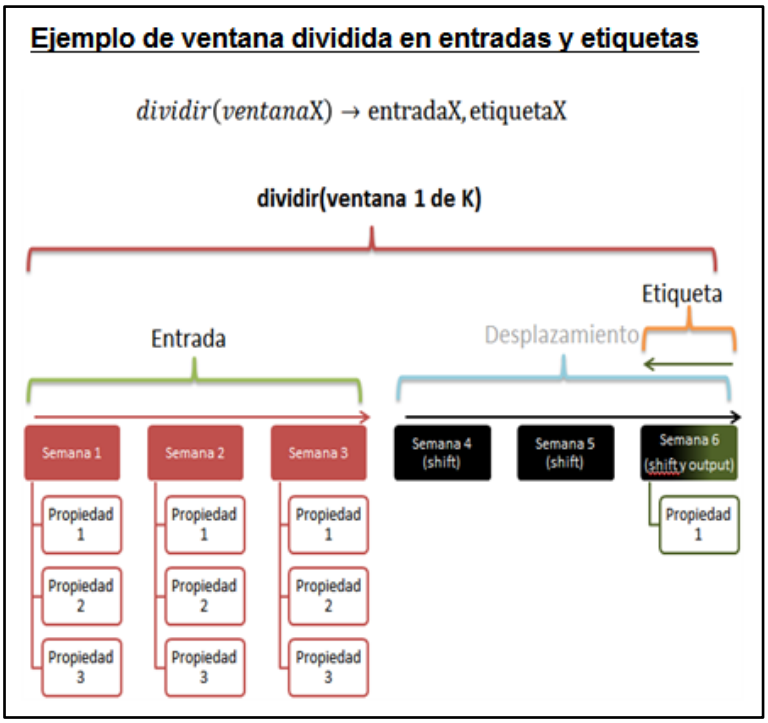


Fig. 21. Ejemplo de ventana dividida en entradas y etiquetas. Fuente: (Elaboración Propia)

Algo que hay que notar es que nuestra función divisora solo se aplicaría a *una sola ventana a la vez*, y puesto que como generaremos varias ventanas con un número variable dependiente de la cantidad de datos y nivel de desplazamiento, es necesario encontrar una estructura que nos ayude a automatizar el procesamiento para que en cada ventana de nuestro *dataset* pueda ser aplicada esta función divisora.

Afortunadamente los *datasets* de TensorFlow heredan un método de Python conocido como *".map"*, que nos permite aplicar a cada elemento de nuestra estructura (a cada elemento del *dataset* de TensorFlow) una función personalizada, siendo el proceso similar al siguiente:

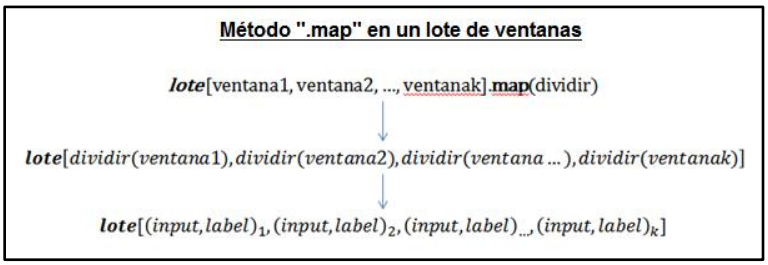


Fig. 22. Método ".map" en un lote de ventanas. Fuente: (Elaboración Propia)

Con esto ya tenemos listo los datos de entrada para los modelos predictivos.

2.5.2. Generar y Ejecutar modelos Predictivos en RNAs.

Como se definió en la sección anterior la idea es que a partir de datos iniciales (un conjunto de características medidas a través del tiempo) se crea un lote de ventanas (en estructura de *dataset* de TensorFlow) cuya cantidad es dependiente de ciertos parámetros (como el tamaño y el desplazamiento) y donde cada ventana generada a su vez contiene ciertos datos de entrada y etiquetas.

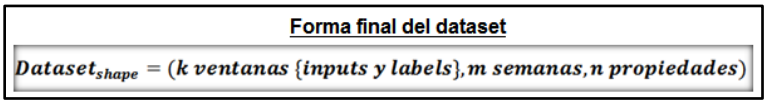


Fig. 23. Forma Final del Dataset. Fuente: (Elaboración Propia)

Antes de continuar hay que mencionar que las funciones son guardadas como funciones asociadas a un objeto (mejor conocido como métodos) y que los 3 diferentes lotes que se generan (los datos de entrenamiento, validación y prueba) son guardados como atributos (es decir tres atributos más uno extra que se añadió como ejemplo).

Aunque ya nos podemos dar cuenta, no está de más decir que el código está construido con el paradigma de Programación Orientada a Objetos (POO) de manera que el código sea reusable y no exista la necesidad de generar código redundante y que a su vez se pierda legibilidad.

Sabiendo que generamos objetos que tienen asociadas las funciones definidas (métodos) y los datos generados con estos (atributos), y que dichos datos son tensores de tres dimensiones o ejes, podemos decir que cumplen con la estructura necesaria para una RNN, la cual puede tener una estructura similar a la siguiente:

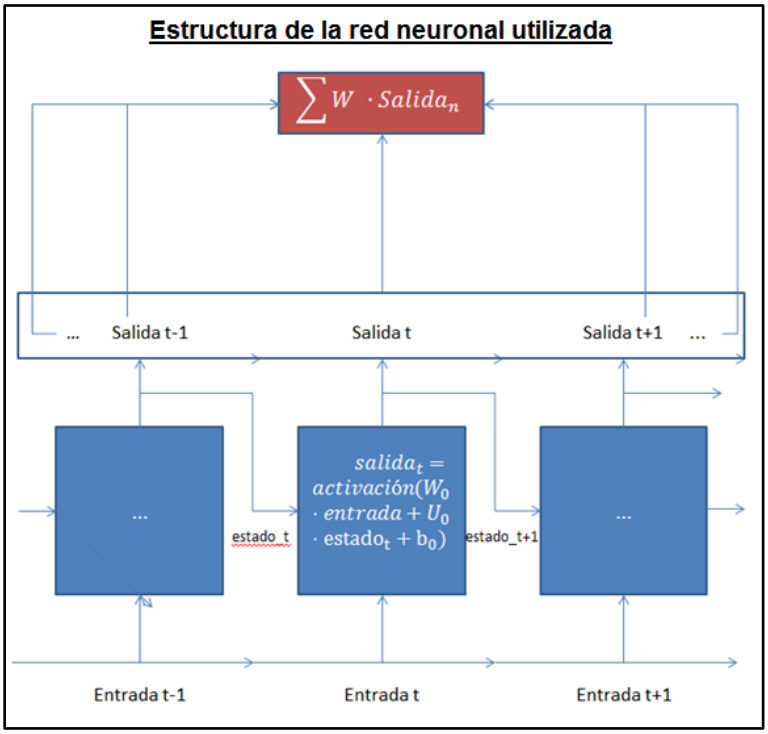


Fig. 24. Estructura de red neuronal utilizada. Fuente: (Elaboración Propia)

La finalidad de una RNN es generar diferentes estados utilizando un elemento de entrada multiplicado por un factor aleatorio (en forma de matriz), a eso se le suma el estado anterior multiplicado por otro factor aleatorio (matriz también) y finalmente se suma un sesgo (*bias*).

Es precisamente la Fig. 24 la representación de una RNN, que si la interpretamos con los datos que hemos conseguido hasta entonces, las entradas serían los elementos del *dataset*, los cuales sirven para calcular un estado al tiempo t donde “Entrada t” es el tensor de entrada, “W“ la matriz aleatoria por la cual se multiplicará la Entrada t, “Estado t” es el estado anterior del tiempo t, “U” la matriz aleatoria por la cual se multiplicará el Estado t y “b” es el sesgo al tiempo t.

Cada rectángulo azul representa el cálculo del estado, cuya dirección bifurca primero como entrada hacia el cálculo del siguiente estado, pero también hacia un vector que guarda dicho estado en una lista completa de estados, al final podríamos elegir qué queremos que nos devuelva, como en el siguiente ejemplo:

Extracto del código de la red neuronal

```
# #  Red Recurrente SimpleRNN # #
simple_rnn = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, simplernn_units]
    tf.keras.layers.SimpleRNN(10, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])
```

Fig. 25. Extracto del extracto del código de la red neuronal. Fuente: (Elaboración Propia)

La capa *SimpleRNN* contiene un parámetro llamado "*return_sequences*" el cual si es igual a False nos devolverá simplemente el último estado o si es igual True nos devolverá toda la lista de estados, esto cambia la forma en que se analiza el resto del a red, ya que si nos devuelve un solo valor se reduce una dimensión (la dimensión del tiempo).

Al final se optó por usar la capa conteniendo todos los estados junto con una capa extra del tipo *Dense* para reducir la dimensión a uno y generar una salida con la misma forma que si solo hubiéramos tomado el último estado.

La razón de tomar esta decisión fue porque aunque en teoría el último estado representa el cálculo de todos los estados anteriores, la proporción de cada estado es muy desigual, ya que por la forma en que se calcula, los últimos estados son los que tienen mayor influencia que los primeros y mientras más estados tenga la capa recurrente más grande será este problema, (porque son operaciones anidadas). Problema que no existe con la decisión que tomamos.

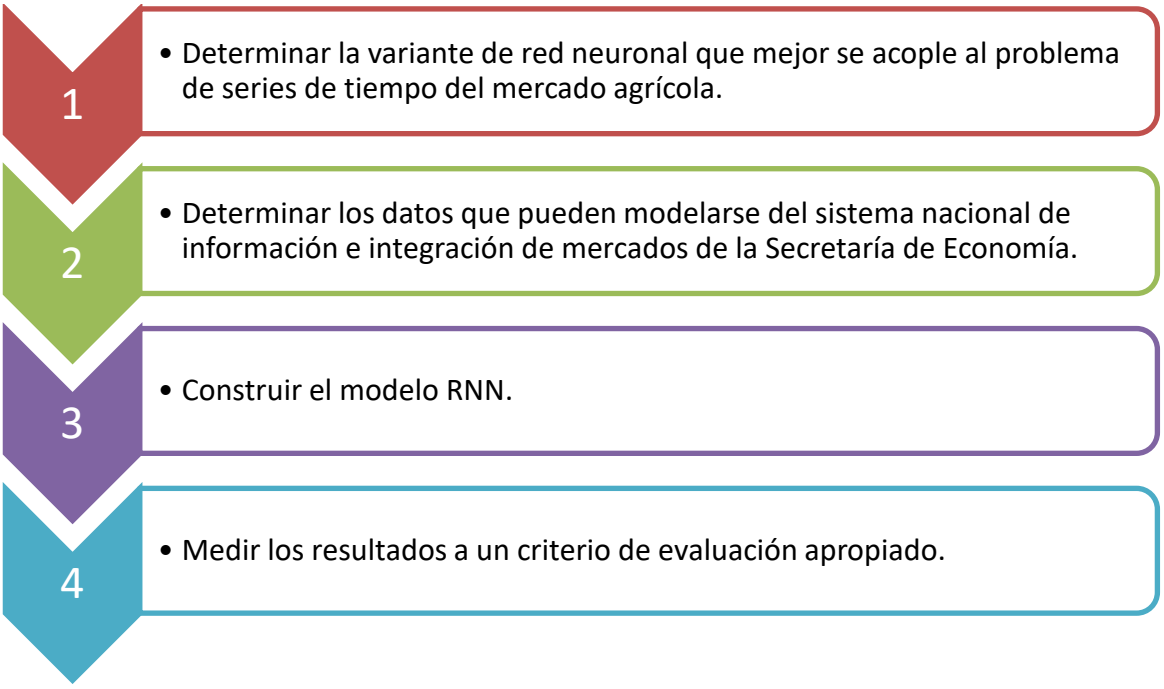
Alternativamente al uso de la capa recurrente se pudo haber optado por una red neuronal del tipo *feedforward*, es decir el uso de una capa aplanadora (*flatten*), que no es sino la multiplicación de las primeras dos dimensiones, la dimensión *batch* por la dimensión tiempo.

Analíticamente al hacer esto se asume que las entradas no tienen orden en específico, pero de ahí el problema de la *feedforward*, ya que en series de tiempo el orden que nos da el tiempo es esencial para la predicción.

Además del análisis con la RNN se hizo un análisis base por motivos de testeo y para ratificar que los números que produzca la RNN tengan sentido, esta base parte del razonamiento de que no hubiese cambios en el precio de un producto con respecto al periodo anterior, es decir que se mantenga el mismo precio y se mide el error con los datos de validación y prueba, el número tendría que estar relativamente cercano al de cualquier método de predicción.

3. METODOLOGÍA

De acuerdo con el planteamiento:



Lo primero para determinar la variante fue una revisión bibliográfica acerca de cuál red resultaría ser la más adecuada, tomando como criterio la naturaleza de los datos (predicción de precios agrícolas en forma de series de tiempo) y se concluyó en el marco conceptual que para este caso una RNN simple del tipo Elman resulta ser la mejor opción.

Seguidamente se busca hacer un análisis exploratorio con el conjunto de datos mediante el uso de un modelo ingenuo para empezar a descartar los datos que no fuesen factibles.

Para cumplir lo segundo del diagrama, es decir encontrar los datos que puedan modelarse con RNN, se plantearon los mismos criterios del ejercicio que se hizo con el método SARIMA (Paredes-Garcia et al., 2019) dando como prioridad a la comparación de resultados, pero además se agregó a la experimentación con redes algunos productos que fueron catalogados en aquel proyecto como ruido blanco y que en este ejercicio sí se pudieron evaluar.

El cumplimiento de la tercer parte representa la parte medular (aunque no la única) del presente trabajo y es la construcción del modelo de RNNs. Resumiendo, al principio se inició el proceso recabando los datos ya existentes de los productos en un único archivo CSV con fecha, nombre del producto y precio, la idea es filtrar solamente los datos útiles para dicho análisis.

De ahí todo lo demás se hizo dentro de Google Colab (por su rápida capacidad de procesamiento y el hecho de generar copias de seguridad), entonces se buscó generar código que tome el CSV como datos de entrada, transforme en un objeto dataframe que permita ser manipulable y por último se convierta a lotes de ventanas los cuales serán evaluados para generar cierto porcentaje de error en predicción, cuyos resultados a su vez serán comparados con la metodología SARIMA (dicha comparación ya formaría parte del cuarto objetivo específico).

Hay que aclarar que para lograr este objetivo existen muchos caminos y variantes que se pueden tomar, hasta el momento se describió a grandes rasgos lo que se hizo en el tercer objetivo, a continuación se describirán los detalles específicos para lograr los resultados obtenidos de las pruebas empíricas y qué fue lo que se hizo para mejorar el modelo.

Lo primero fue un proceso de carga de archivos CSV a dataframe, de ahí se generó una función “padre” cuyo rol entre otras cosas es definir variables, cambiar la muestra de periodicidad diaria a semanal, separar los datos en los grupos de entrenamiento, validación y prueba, crear las ventanas, y crear la predicción para las ventanas, a toda esta función padre se le llamó “aplicador de ventanas”.

Entrando en detalles y siguiendo con la lógica descrita en el marco conceptual sobre cómo generar ventanas se definieron las siguientes variables product_index, start_year, train_percent, val_percent, data, input_width, label_width, shift, label_columns, all_products, product_name, los cuales se detallan a continuación.

variable	rol	valor
product_index	Es el índice que sirve como selector de cada producto dentro del dataframe	Un valor entre 0 y n-1, por ejemplo dentro de prod_dataframe sería entre 0 y 16 (varió de acuerdo al producto a elegir)
start_year	Es el año que sirve como punto de partida para la creación de ventanas	Cualquier número entre 2009 y 2019, en varios casos específicos se hizo variar
train_percent	porcentaje de los datos destinados para entrenamiento	train + val <= 1, valor por defecto 0.6 (se complementa con los porcentajes de validación y prueba)
val_percent	porcentaje de los datos destinados para validación	train + val <= 1, valor por defecto 0.1
data	Es la fuente de datos, el dataframe que se generó con anticipación	Se definieron dos dataframes: prod_dataframe y prod_white con los datos disponible pero pueden definirse más dataframes
label_width	ancho del conjunto de la ventana de etiqueta	no puede ser mayor a la suma de input_width con shift; valor por defecto 8
input_width	ancho del conjunto de la ventana de entrada	menor al tamaño de la ventana y mayor cero; valor por defecto 8
shift	número de espacios de desplazamiento	mayor o igual a cero, mientras más grande más incertidumbre; valor por defecto 1
label_columns	las columnas con las que trabajará la ventana	El único valor es 'precio', pero puede extenderse a más valores si se agregan nuevos datos; valor por defecto = 'Precio'
all_products	Es una relación índice producto sobre los datos de la variable data	Se genera una por cada dataframe, en este caso hay dos 17 y 6 productos
product_name	Extrae el valor y lo imprime en una gráfica	El valor es extraido del dataframe

Fig. 26. Variables utilizadas en la red neuronal. Fuente: (Elaboración Propia)

Hay que recalcar que en varios de los parámetros los valores se definieron a través de ir probando qué valores funcionan mejor, por ejemplo el desplazamiento igual a uno tiene menor incertidumbre por lo tanto es más fácil de predecir, ventanas anchas son ventanas que predicen mejor (aunque si la información es muy poca puede haber errores) o que algunos años tienen mejores patrones de predicción que otros para ciertos productos.

Existe una sección definida para modificar los dataframes, se encarga de asignar al índice la columna de la fecha y cambiar la periodicidad “diaria laboral” (lunes a viernes) a semanas con el método “.resample” de los dataframes, después la muestra se recorta al año deseado, con la variable “start_year”, recordar que aunque el año inicial es 2009 cada producto puede elegir con qué año del rango predefinido comenzar (siempre que esté disponible), este puede ser benéfico para productos que presentan incertidumbre muy grande a largo plazo; en la práctica hubo varios años iniciales, siendo el parámetro con mayor variabilidad.

De ahí se separaron los datos en los grupos correspondientes (entrenamiento, validación y prueba), los cuales servirán a su vez como entradas para el siguiente bloque, lo cual consiste en una multiplicación de dataframes por porcentajes en decimales, los cuales sumados representan el cien por ciento de los datos, los valores por defecto son 0.6 para entrenamiento, 0.1 para validación y por lo tanto 0.3 para prueba, esto puede considerarse una separación hold-out simple tal y como se describió en la sección de marco teórico.

Con el fin de tipificar nuestros datos y una vez ya fragmentados (con el método hold-out) para avanzar hay que determinar su media y su desviación estándar (específicamente la del conjunto de entrenamiento), luego hay que restar la media y dividir sobre el la desviación estándar a cada uno de esos conjuntos para obtener los datos normalizados.

En la siguiente sección del código se nos pide aplicar las variables definidas a una función generadora de ventanas, la cual depende de la función aplicadora, podemos llamar a esta función generadora como “función hija” cuyo rol es recibir varios parámetros de la función padre, como lo son los tres dataframes construidos anteriormente (los que fueron divididos en entrenamiento, validación y prueba), anchura de entrada y anchura de etiquetas igual a 8, desplazamiento de 1 y columnas asignadas igual a “precio”.

Todo esto pasa por el proceso de generación de ventanas y predicción descrita en la sección de marco conceptual. La idea es que en este punto ya existan tres grupos de datos que puedan servir como materia prima de ventanas.

Para activar la red neuronal se probaron funciones sigmoide, relu, tangente hiperbólica y lineal, no obstante aunque muchas tuvieron resultados muy buenos una función lineal fue lo que dio mejores resultados de manera generalizada.

Otras decisiones tomadas sobre cómo debería ir la estructura de la red son el uso del algoritmo de optimización RMSProp para la red (por los motivos descritos en el marco conceptual), el error cuadrático medio (MSE) y la raíz del error cuadrático medio (RMSE) para la función de error y la métrica evaluadora respectivamente; la justificación sobre la MSE como función de error fue su simpleza y el hecho de que no es necesario que la unidades mantengan su magnitud original, por otro lado usar RMSE debe conservar su magnitud original (no solo proporcionalmente), además de que en el ejercicio SARIMA se evaluó con dicha métrica por lo que es necesario para siguiente paso.

Para describir el *cuarto y último objetivo específico*, se hizo una prueba de proporciones todo con el objetivo de comprobar el objetivo general de esta tesis.

Dato: De acuerdo a los resultados obtenidos del tercer objetivo 17 de 23 productos mejoraron con RNN en vez de SARIMA, la prueba se llevó de la siguiente manera.

- **Establecer hipótesis nula y alternativa**

$$\begin{cases} H_0: p_{redes} = 50\% \\ H_a: p_{redes} \neq 50\% \end{cases}$$

Estadístico z ≈ 1.97
Valor p ≈ 0.9756

Conclusión: Si se toma un nivel de significancia del 5% se puede rechazar la hipótesis nula y sugerir la alternativa, es decir que los pronósticos del método RNN pueden llegar a generar mejores resultados que los del método SARIMA.

4. RESULTADOS Y DISCUSIÓN

Como se ha mencionado anteriormente los precios fueron extraídos del Sistema Nacional de Información e Integración de Mercados (SNIIM), por parte de la Secretaría de Economía; con la metodología desarrollada se evaluaron en total 23 productos, de los cuales 17 ya eran evaluables con el método ARIMA (y se planteó una comparación de los resultados de ambos métodos) y los otros 6 no fueron evaluables con SARIMA pero si con el método RNN.

El producto final de este análisis se resume en dos partes: primero un porcentaje de error de predicción para cada producto analizado, el cual es comparable con el error de cada producto que se analizó con ARIMA y segundo en gráficas que se generaron gracias al desarrollo del código con ayuda de la biblioteca Matplotlib y que nos sirvió como indicador exploratorio para ver la eficiencia del modelo RNN.

Se puede decir que *en la mayoría de los datos (17 de 23 productos evaluados para ser exactos) los productos tienen un mejor desempeño con RNNs que con SARIMA* (aunque hay algunos casos concretos en los que no), se aprecia en la siguiente tabla cada caso con porcentajes en términos RMSE (para que sean comparables ambos métodos):

PORCENTAJES DE ERROR		
Producto	**ARIMA**	**Redes Neuronales**
Coliflor Grande	9.47%	12.05%
Nopal Grande	34.41%	10.16%
Calabacita Italiana	15.91%	15.01%
Chayote sin Espinas	41.39%	35.44%
Cacahuate	3.89%	7.24%
Tomate Saladette	8.09%	8.06%
Naranja Valencia Grande	3.89%	1.39%
Aguacate Hass	7.01%	1.06%
Kiwi	8.24%	2.19%
Limon	10.07%	7.25%
Lima #5	6.11%	2.49%
Plátano Macho	4.51%	12.81%
Plátano Tabasco	23.15%	19.64%
Piña Mediana	4.32%	4.70%
Toronja Roja	3.15%	4.52%
Epazote	6.51%	14.57%
Jícama	9.33%	1.67%
Pera D'Anjou	*no evaluable*	6.53%
Plátano Dominico	*no evaluable*	23.06%
Fresa	*no evaluable*	4.67%
Sandía Rayada	*no evaluable*	3.48%
Mazana Red Delicious	*no evaluable*	6.69%
Mazana Golden Delicious	*no evaluable*	5.21%

Fig. 27. Porcentajes de Error de SARIMA y RNN en términos de RMSE.

Como se dejó claro en la metodología los parámetros ingresados fueron los mismos para todos los productos, existiendo mínima variación y para dar congruencia a los resultados; por ejemplo las ventanas son de 8 semanas, el desplazamiento es igual a 1, los conjuntos de entrenamiento, validación y prueba son 60%, 10% y 30% respectivamente.

Antes de ingresar al modelo de predicción, los datos pasan por un proceso de normalización (unidades tipificadas) para evitar que la magnitud de los datos repercuta en los resultados, una vez analizados los datos regresan a su magnitud original (de unidades tipificadas a precio) para ser presentados y comparados.

El único parámetro que se varió bastante fue la fecha inicial para generar las ventanas; en un inicio se tomó como parámetro 2017, y aunque muchos productos (como el Nopal, la Calabacita y el Chayote) funcionaron muy bien con esta fecha, en algunos otros (como el limón o la lima) el cambio a una fecha anterior les favoreció tanto que le dieron la vuelta el análisis de peor a mejor, o en otros casos por lo menos se redujo el error (como el Plátano Macho o el Cacahuate).

Hay que recordar que la información disponible de los productos generalmente inicia del 1° de Enero de 2009 y termina al 28 de Febrero de 2019, aunque no en todos los casos, ya que hay veces en la que la disponibilidad datos de inicio fue menor. También nos concentramos en los datos más relevantes “fecha”, “producto” y “precio frecuente” y se descartaron algunos que eran más redundantes.

Con esta información se transformaron de objetos “CSV” a *arrays* de numpy y finalmente a objetos *dataset* de TensorFlow (contenedores de tensores). La columna cantidad fue la que más dificultades representó, debido a que la definición de cantidad varía mucho, ya sea por peso o por número de unidades.

Un ejemplo claro es “Nopal Grande” que se medía en cajas de 100 kilogramos y luego en kilogramos individuales, o el “Tomate Saladette” que durante diferentes periodos se llegó a medir por cajas de 17, 28, 30 y 32 kilogramos.

En estos casos se asume un precio unitario haciendo una división simple, aun con la pequeña variabilidad que nos da cualquier escala, incluso en los productos donde se manejan unidades (como manojos o piezas) no hay mayor problema siempre no se combinen ambas presentaciones (peso y unidades).

Aunque hay que reconocer que casos muy específicos esto último sí sucedió, como en el Nopal Grande y la Piña Mediana; lo que se hizo entonces fue buscar la equivalencia de piezas a kilogramos y afortunadamente existe muy poca variabilidad, pero aun así es importante mencionarlo.

Como se había mencionado además de los porcentajes de error se elaboraron gráficas por cada producto, para demostrar de manera puntual cómo puede haber variación con las predicciones; a continuación un ejemplo:

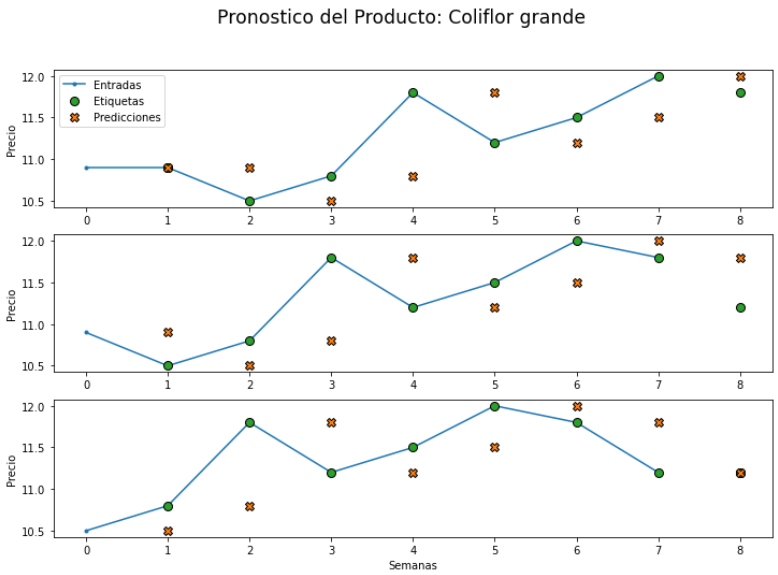


Fig. 28. Tres periodos de las diferencias entre predicciones (cruz) y etiquetas (punto).

Dicho ejemplo corresponde al producto “Coliflor” donde podemos ver que el eje horizontal representa las semanas y el vertical el precio que puede tomar; dentro de dicha gráfica se aprecia una línea azul que son los datos de entrada del producto, los puntos verdes las etiquetas y las cruces son las predicciones, mientras más cerca esté la cruz del punto verde la predicción es mejor.

El código es totalmente reproducible, aún con sus elementos aleatorios ya que siempre convergerán a resultados muy cercanos.

Por último y para cerrar esta sección de resultados me gustaría hacer una comparación objetiva entre los resultados del método ARIMA y RNN a nivel global remitiéndome a la prueba de proporciones.

Dicha prueba nos permitió asumir como hipótesis nula una cantidad menor o igual de predicción a 50% de nuestro método de predicción (RNN), al por fin conseguir rechazarla y asumir la hipótesis alterna pudimos (con un nivel de significancia de 5%) obtener evidencia de que RNN tiene mejor nivel de predicción.

Algunos hechos relevantes y relacionados que sirven como pistas adicionales son el hecho de que de los 23 productos 17 se comportaron mejor con RNN o que de la desviación estándar de los productos comparables de ambos grupos el que menor presenta variabilidad es otra vez RNN.

5. CONCLUSIONES

Con los resultados que se generaron gracias al código desarrollado y su posterior evaluación con la prueba de proporciones y algunos hechos que se definieron al final de la sección anterior, se cumple el objetivo de la tesis y se afirma la hipótesis planteada, *las RNNs pueden generar en su mayoría mejores resultados (con ciertas excepciones) que el método de SARIMA y existe evidencia que pueda dar soporte a esta afirmación (la prueba de proporciones).*

Dado que no todos los casos fueron positivos con RNN la primera sospecha desde el punto de vista de la investigación puede ser la metodología de partición de datos, la validación hold-out simple funcionó excelente, pero considerando que los datos fueron relativamente limitados quizás una partición diferente beneficiaría el proceso.

Algunas de las conclusiones que dejó el desarrollo de este proyecto son:

Al ejecutar más veces la RNN (aumentando el número de épocas) se mejora la predicción, pero esta mejoría es cada vez más pequeña hasta el punto que deja de ser relevante.

Aumentar el tamaño de la ventana es bastante benéfico en la predicción, aunque si la disponibilidad de datos es pequeña puede ser contraproducente e incluso no poder generar resultados.

La relación entre la distribución de los datos en conjuntos de entrenamiento, validación y prueba con respecto al tamaño de datos es algo que también puede causar un mejor o peor resultado, si el tamaño es muy pequeño y una de las tres secciones se le asigna un porcentaje bajo, puede ser que no genere ventanas para su análisis y no entregue resultados (o entregue uno muy malo).

La tipificación de datos es una práctica que mejora bastante los resultados, esto se comprobó con las pruebas, sin embargo una vez procesada la predicción, fue necesario que nuestro código regresara los resultados pero términos de precio, por motivos de presentación de resultados.

Hay que tomar en cuenta que a diferencia que el ejercicio que hizo con SARIMA (el cual construido con fluctuaciones, incluido el proceso de normalización) en el caso de RNN el procesamiento se hizo con precios completos, donde cada grupo de precios se obtuvo su media y desviación estándar con la finalidad de normalizar los precios correspondientes.

Para terminar me gustaría comentar que aunque esto signifique el final del presente trabajo aún existen bastantes áreas que pueden ser desarrolladas, por ejemplo extender la funcionalidad del código, agregar nuevas variables o hacer un nuevo diseño de características para mejorar la predicción, incluso se puede comentar que ambos métodos SARIMA y RNNs no son excluyentes, podría desarrollarse un modelo híbrido.

6. BIBLIOGRAFÍA

Abdulkarim, S. A. (2016). Time series prediction with simple recurrent neural networks. *Bayero Journal of Pure and Applied Sciences*, *9*(1), 19. https://doi.org/10.4314/bajopas.v9i1.4

Banco Mundial. (2008). *El alza de precios de los alimentos y sus efectos en América Latina y el Caribe*. http://siteresources.worldbank.org/INTLACINSPANISH/Resources/LCRFoodPricesBrochure_SP.pdf

Brandt, J. A., & Bessler, D. A. (1983). Price forecasting and evaluation: An application in agriculture. *Journal of Forecasting*, *2*(3), 237–248. https://doi.org/10.1002/for.3980020306

Buhigas, J. (2017). *Las nuevas profesiones agrícolas que salvarán el mundo*. Puentes Digitales. https://puentesdigitales.com/2017/12/07/las-nuevas-profesiones-agricolas-que-salvaran-el-mundo/

Constitución política de los estados unidos mexicanos, 319 (1917).

Camara de Diputados de H. Congreso de la Union. (2004). Ley General de Desarrollo Social. *Diario Oficial de La Federación*, 1–20. http://www.diputados.gob.mx/LeyesBiblio/pdf/264.pdf

Cárdenas Elizalde, M. del R., Alberto Cortés Cáceres El Colegio de México Agustín Escobar Latapí, F., Nahmad Sittón, S., Roberto Scott Andretta, J., María Teruel Belismelis, G., Hernández Licona Secretario Ejecutivo, G., Martínez Mendoza, É. A., Aparicio Jiménez, R. C., Gutiérrez Cruz, D., & Lafontaine Navarro Alice Zahí Martínez Treviño Alejandra Correa Herrejón Oscar David Mejía Arias Sandra Ramírez García Alma Verónica Corona García, L. (2018). *Estudio Diagnóstico del Derecho a la Alimentación Nutritiva y de Calidad 2018*. https://www.coneval.org.mx/quienessomos/InvestigadoresAcademicos/Paginas/Investigadores-Academicos-2014-2015.aspx

Chambers, J. C., Mullick, S. K., & Smith, D. D. (1971). How to choose the right forecasting technique. In *Harvard Business Review* (Vol. 49, Issue 4, pp. 45–70). http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:How+to+choose+the+right+forecasting+technique#0%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:HOW+TO+CHOOSE+RIGHT+FORECASTING+TECHNIQUE#0

Chollet, F. (2018). Deep Learning with Python, Manning. In *Manning*. http://faculty.neu.edu.cn/yury/AAI/Textbook/Deep Learning with Python.pdf

Chollet, F. (2019). *Making new layers and models via subclassing*. https://keras.io/guides/making_new_layers_and_models_via_subclassing/

Díaz, E., & Meade, G. L. (2019). *Retos y tendencias en el sector agropecuario en México*. https://www.ey.com/es_mx/consumer-products-retail/retos-y-tendencias-en-el-sector-agropecuario-en-mexico

Directorate of Economics and Statistics, Government of Puducherry, I. (2018). *PROF. P.V. SUKHATME - AN EMINENT STATISTICIAN*. http://ns.itestweb.in/pdy4/prof-pv-sukhatme-eminent-statistician

extrapolar | Definición | Diccionario de la lengua española | RAE - ASALE. (n.d.). Retrieved April 23, 2021, from https://dle.rae.es/extrapolar?m=form

FAO. (2010). *Perspectivas Económicas y Sociales*. http://www.fao.org/economic/es-policybriefs/es.

FAO. (2015). *El futuro de la alimentación y la agricultura - Tendencias y desafíos* (Vol. 4, Issue 3). http://www.fao.org/3/a-i6881s.pdf

FAO. (2020). *Pérdida y desperdicio de alimentos*. http://www.fao.org/policy-support/policy-themes/food-loss-food-waste/es/

Jha, G. K., & Sinha, K. (2013). Agricultural Price Forecasting Using Neural Network Model: An Innovative Information Delivery System. *Agricultural Economics Research Review*, *26*(2), 229--239. http://ageconsearch.umn.edu/bitstream/162150/2/8-GK-Jha.pdf

Jones, T. (2017). *Recurrent neural networks deep dive – Build Smart. Build Secure. IBM Developer*. https://developer.ibm.com/technologies/artificial-intelligence/articles/cc-cognitive-recurrent-neural-networks/

Josh Patterson, A. G. (2017). *Deep Learning: A Practitioner's Approach*. O'Reilly Media.

Martínez, A., Salgado, A., & Vázquez, S. (2017). Tendencias recientes del sector primario en México. *Observatorio Económico México*, 1–21. www.bbvaresearch.com

McKinney, W. (2017). Python for Data Analysis - Data Wrangling with Numpy, Pandas and IPython. In M. Beaugureau, K. Brown, & J. Kwityn (Eds.), *O'Reilly* (Second). https://doi.org/10.1097/00007890-200105270-00005

Monjarás, A. (2017). *¿En manos de quién está el campo mexicano? - Vía Orgánica*. Vía Orgánica. https://viaorganica.org/en-manos-de-quien-esta-el-campo-mexicano/

Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2016). *Introduction Time Series Analysis and Forecasting*. 671.

Nagy, M., Mohamed, M. A., & Aloufi, K. (2019). A Modified Method for Detecting DDoS Attacks Based on Artificial Neural A Modified Method for Detecting DDoS Attacks Based on Artificial Neural Networks. *The 1st International Conference on Information Technology*, *April*.

ONU. (2012). Declaración Universal de Derechos Humanos. In *Declaración Universal de Derechos Humanos: Vol. III* (Issue Iii, pp. 1–12). https://doi.org/10.18356/edcaa4d1-es

Paredes-Garcia, W. J., Ocampo-Velázquez, R. V., Torres-Pacheco, I., & Cedillo-Jiménez, C. A. (2019). Price forecasting and span commercialization opportunities for Mexican agricultural products. *Agronomy*, *9*(12). https://doi.org/10.3390/agronomy9120826

Rashid, T. (2016). *Make your own neural network*. CreateSpace Independent Publishing Platform. http://makeyourownneuralnetwork.blogspot.com/

Richardson, J. G., & Newbury, F. D. (2018). Business Forecasting: Principles and Practice. *Southern Economic Journal*, *19*(4), 531. https://doi.org/10.2307/1054108

Rizo, E. (2018). *Se consolida México como proveedor de tomate a nivel mundial - Hortalizas*. https://www.hortalizas.com/cultivos/tomates/se-consolida-mexico-como-proveedor-de-tomate-a-nivel-mundial/

Saumyendra Sengupta, C. P. K. (1994). *C++: Object-Oriented Data Structures*.

Shmueli, G. (2010). To Explain or to Predict? *Statistical Science*, *25*(3), 289–310. https://doi.org/10.1214/10-STS330

Smith, S. K., & Sincich, T. (1991). An Empirical Analysis of the Effect of Length of Forecast Horizon on Population Forecast Errors. *Demography*, *28*(2), 261–274. https://doi.org/10.2307/2061279

Sugiartawan, P., & Hartati, S. (2019). Time series data prediction using elman recurrent neural network on tourist visits in tanah lot tourism object. *International Journal of Engineering and Advanced Technology*, *9*(1), 314–320. https://doi.org/10.35940/ijeat.A1833.109119

Tiffin, R., & Irz, X. (2006). Is agriculture the engine of growth? *Agricultural Economics*, *35*(1), 79–89. https://doi.org/10.1111/j.1574-0862.2006.00141.x

Torres Durán, A. M. (2013). *Propuesta metodologica para la ensenanza del sistema nervioso en el grado octavo de la Institucion Educativa Francisco Miranda* [Universidad Nacional de Colombia]. http://bdigital.unal.edu.co/9720/7/28556444.2013.pdf

Wageningen Food & Biobased Research. (2014). *Programa Nacional de Agrologística*. https://edepot.wur.nl/373868

Wang, J., & Wang, J. (2016). Forecasting energy market indices with recurrent neural networks: Case study of crude oil price fluctuations. *Elsevier*, *102*, 365–374. https://doi.org/10.1016/j.energy.2016.02.098

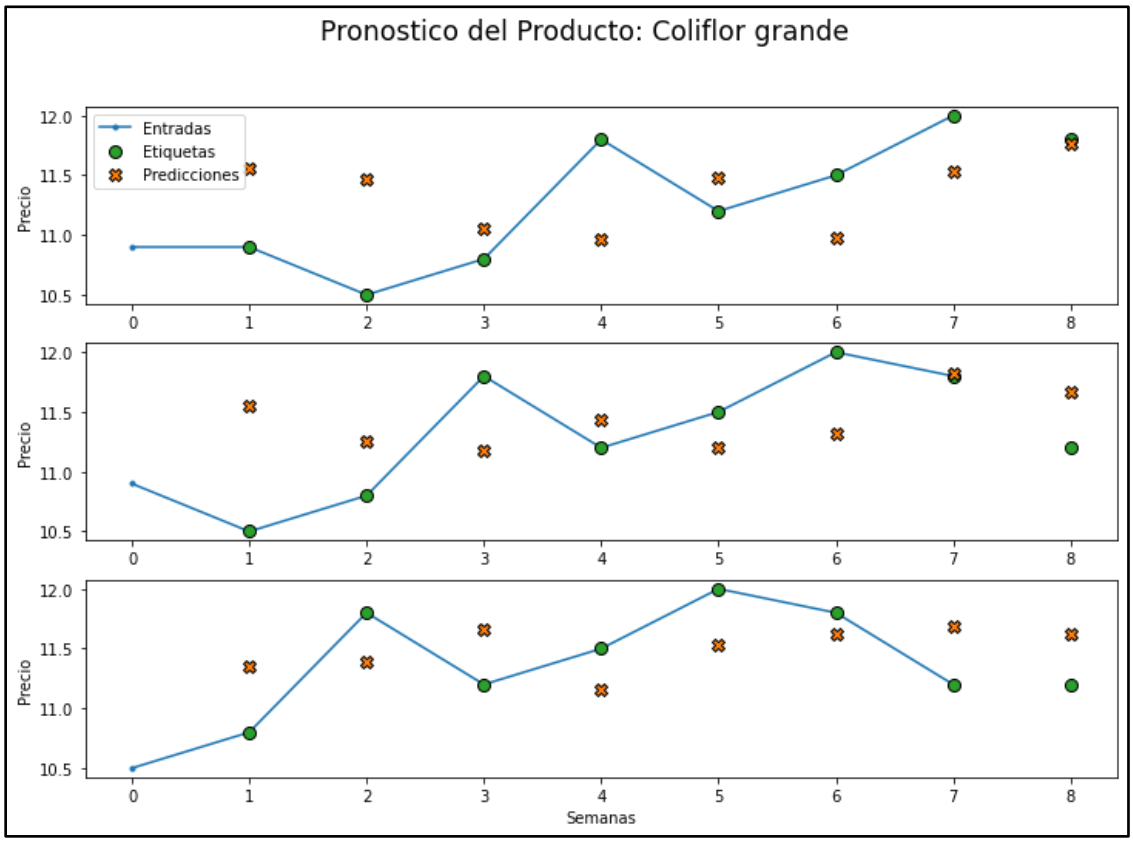
Wu, D., Wang, X., Su, J., Tang, B., & Wu, S. (2020). A labeling method for financial time series prediction based on trends. *Entropy*, *22*(10), 1–25. https://doi.org/10.3390/e22101162

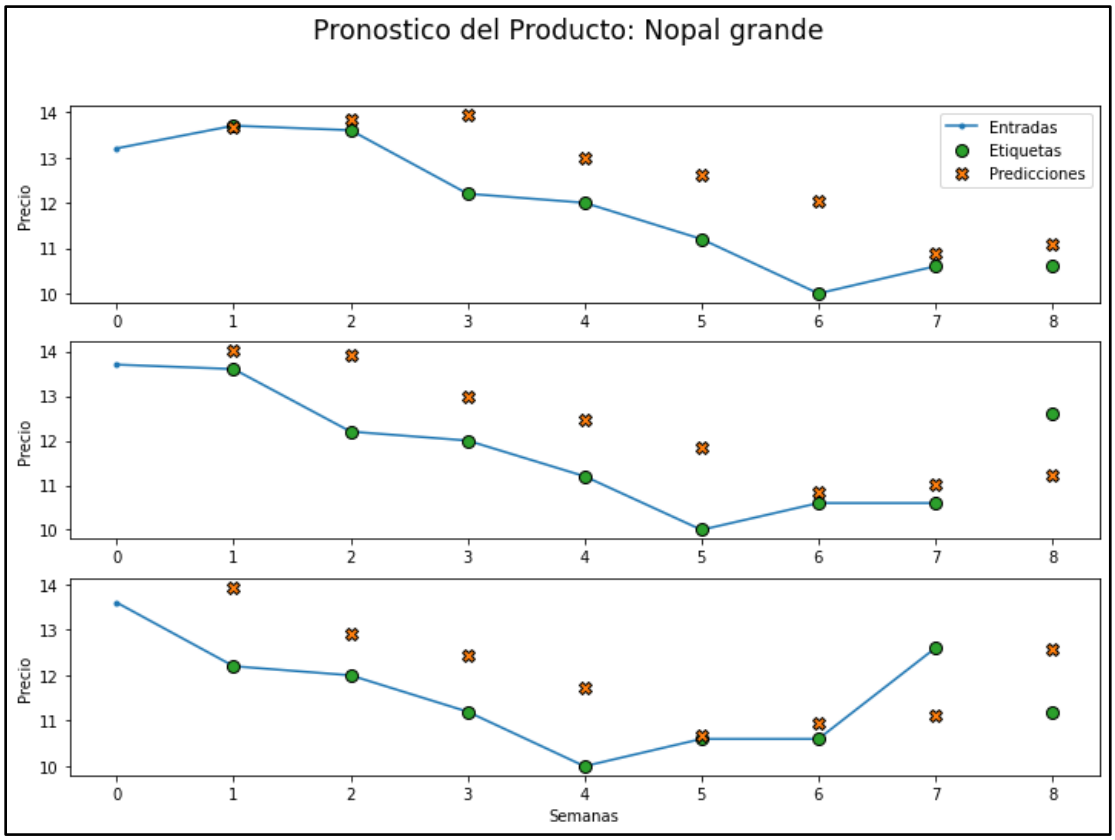
Zarecero, O. A. (2017). Análisis del comportamiento del precio del maíz [Universidad Autónoma de Chapingo]. In *Chapingo*. http://ri.uaemex.mx/bitstream/handle/20.500.11799/80159/TESIS DOCTORAL OSCAR.pdf?sequence=1&isAllowed=y

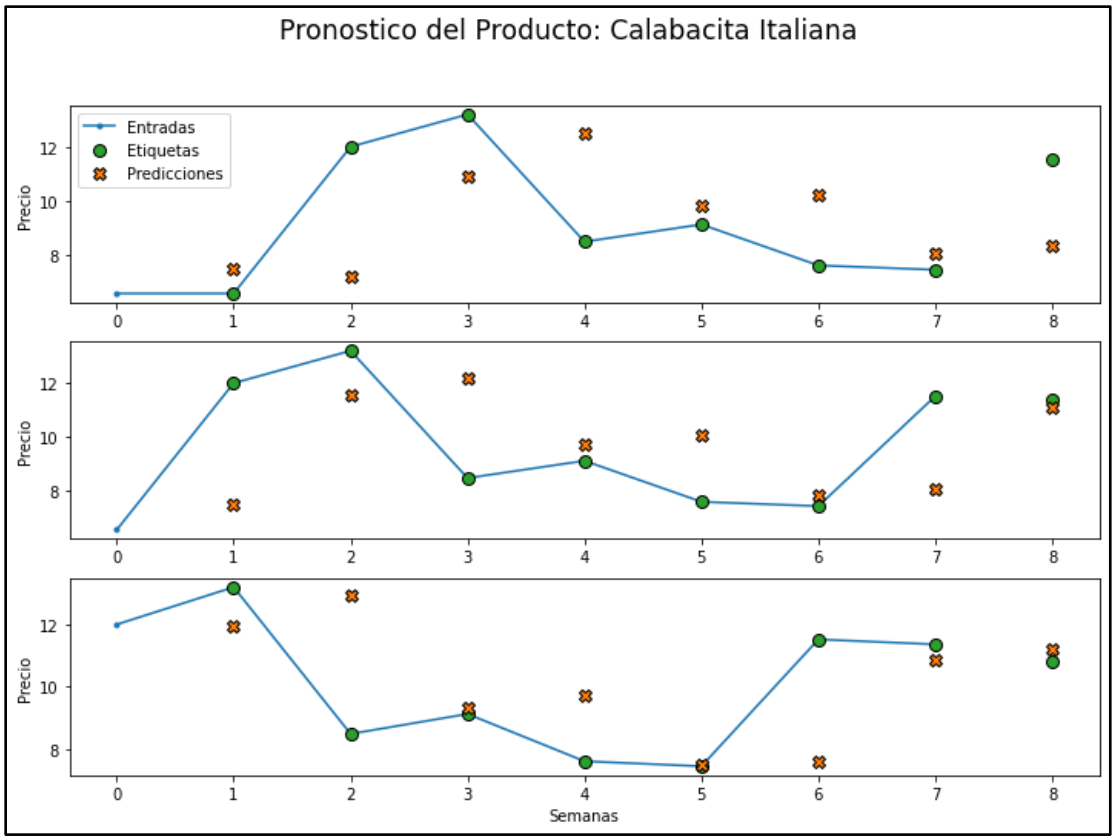
7. ANEXOS

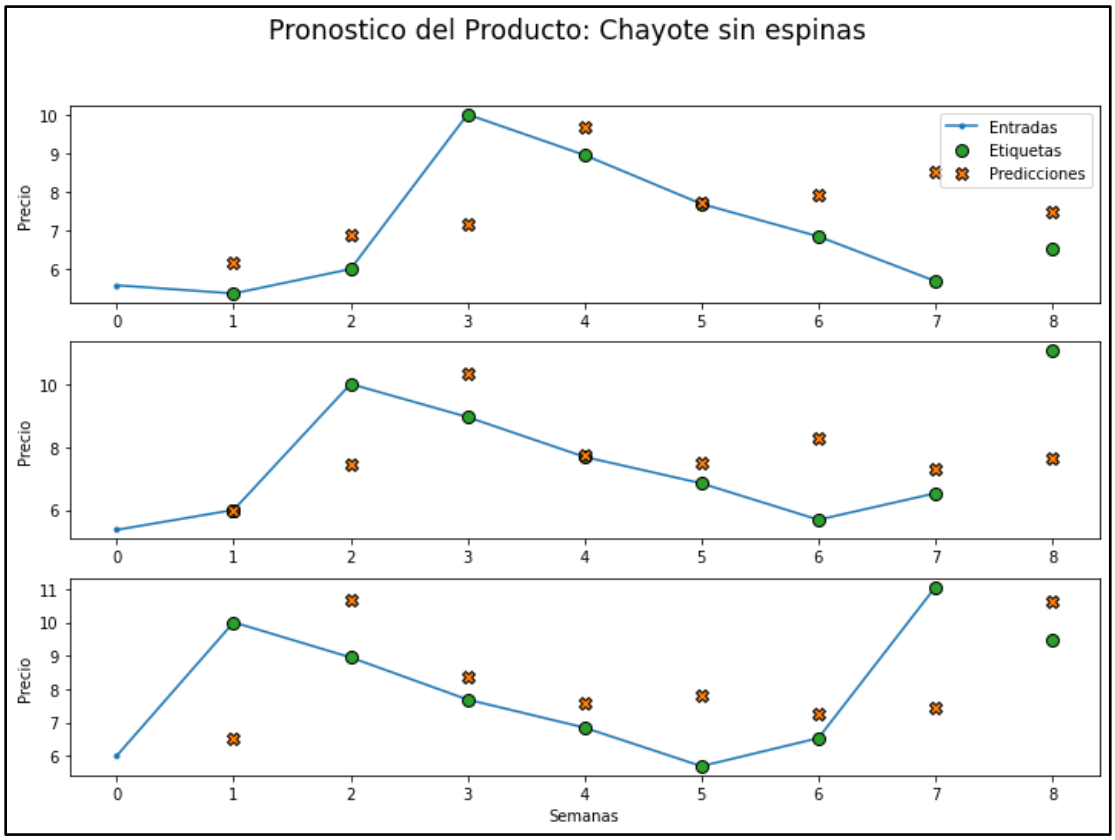
7.1. Resultados gráficos de las evaluaciones con el método de RNN.

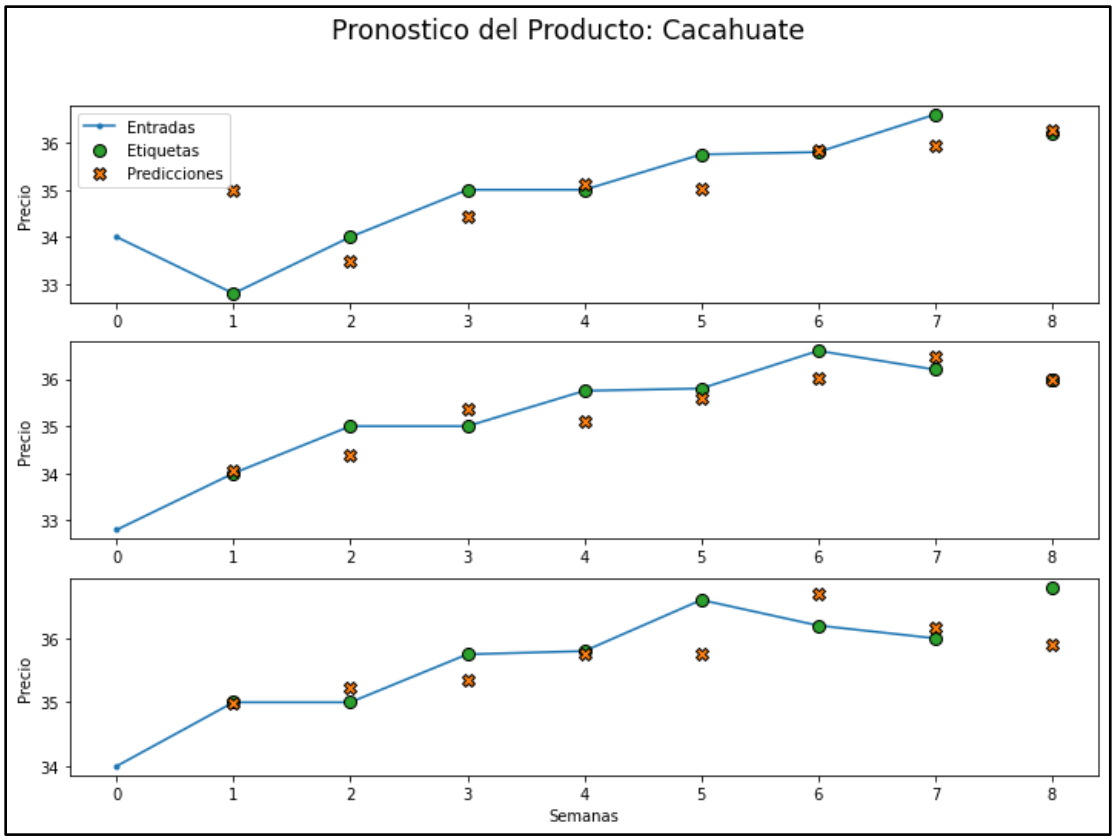
7.1.1. Productos que también son evaluables con ARIMA.

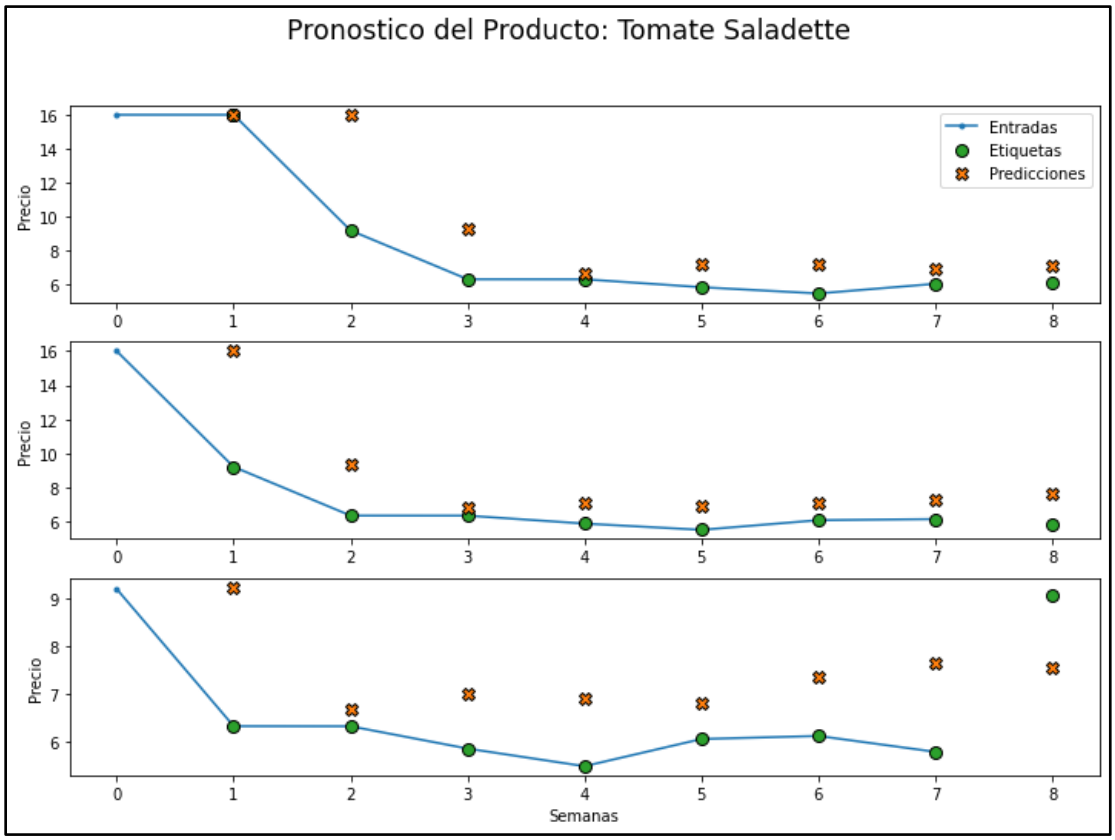


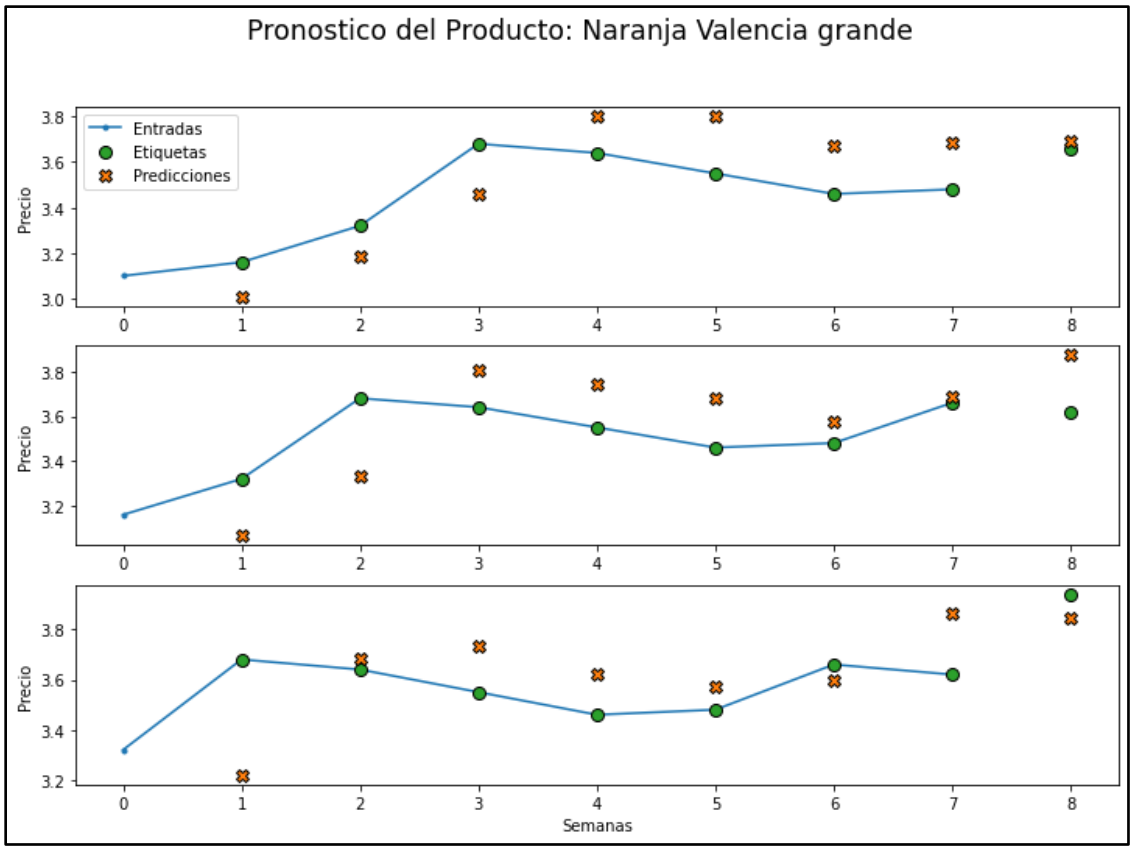

Pronostico del Producto: Nopal grande
Entradas
Etiquetas
Predicciones
Precio
Semanas

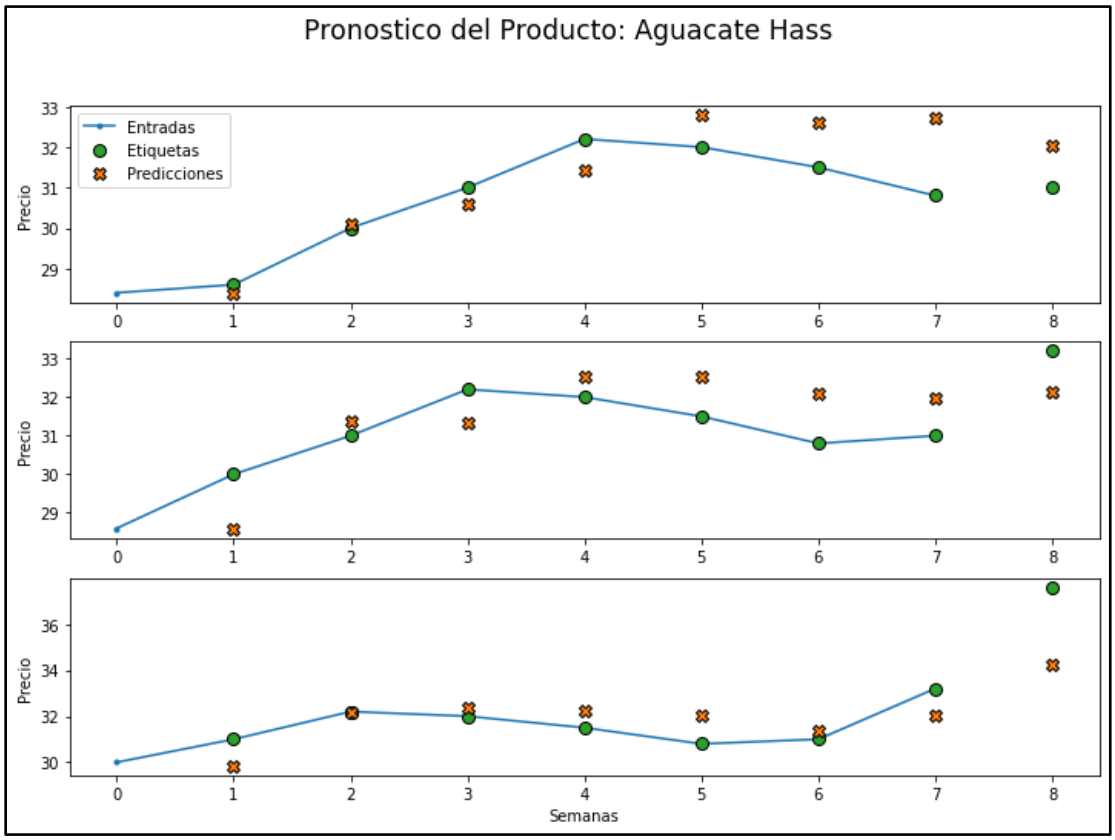

Pronostico del Producto: Calabacita Italiana
Entradas
Etiquetas
Predicciones
Precio
Semanas

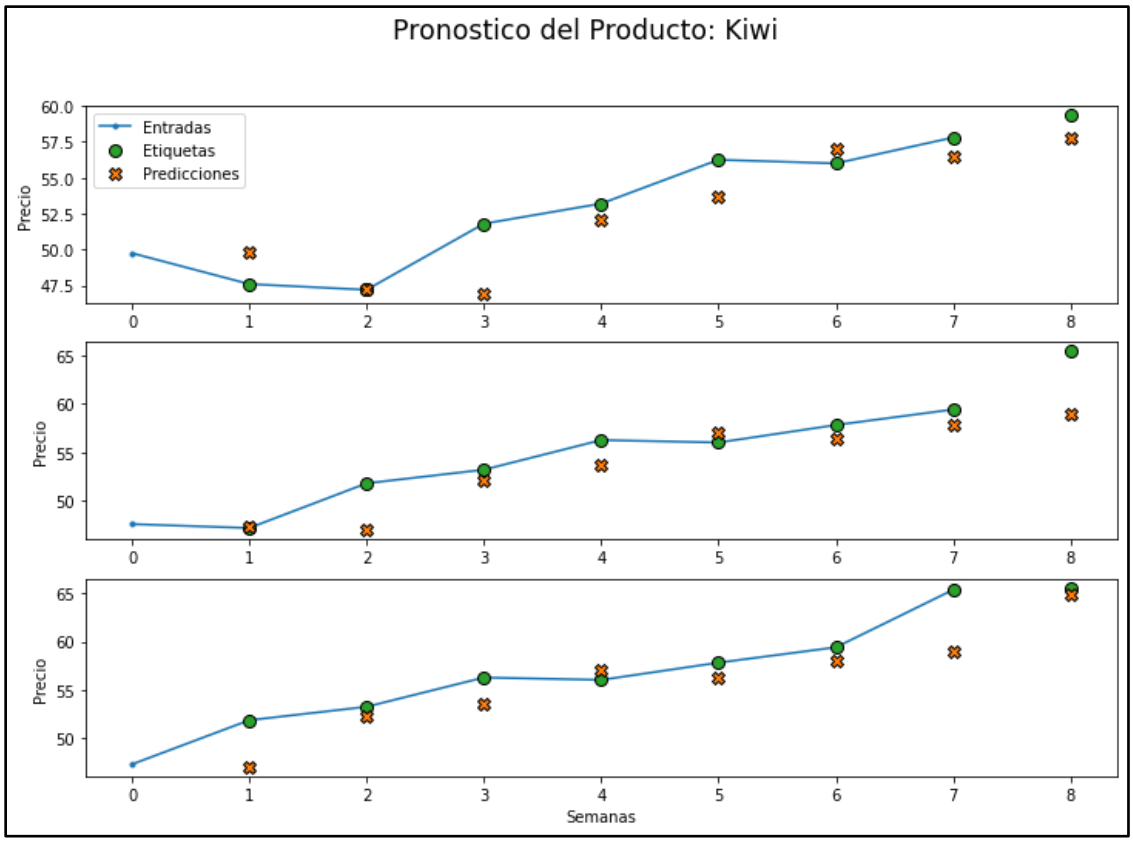

Pronostico del Producto: Chayote sin espinas
Entradas
Etiquetas
Predicciones
Precio
Semanas

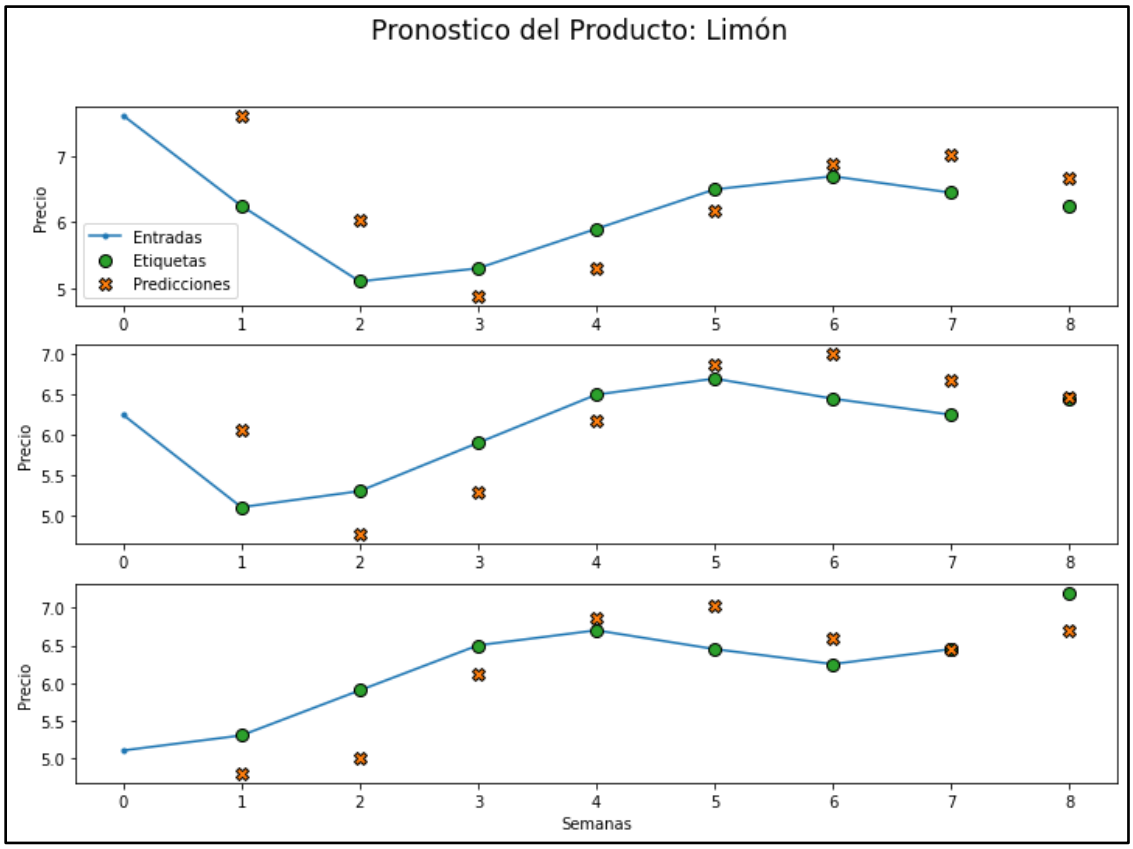

Pronostico del Producto: Cacahuate
Entradas
Etiquetas
Predicciones
Precio
Semanas

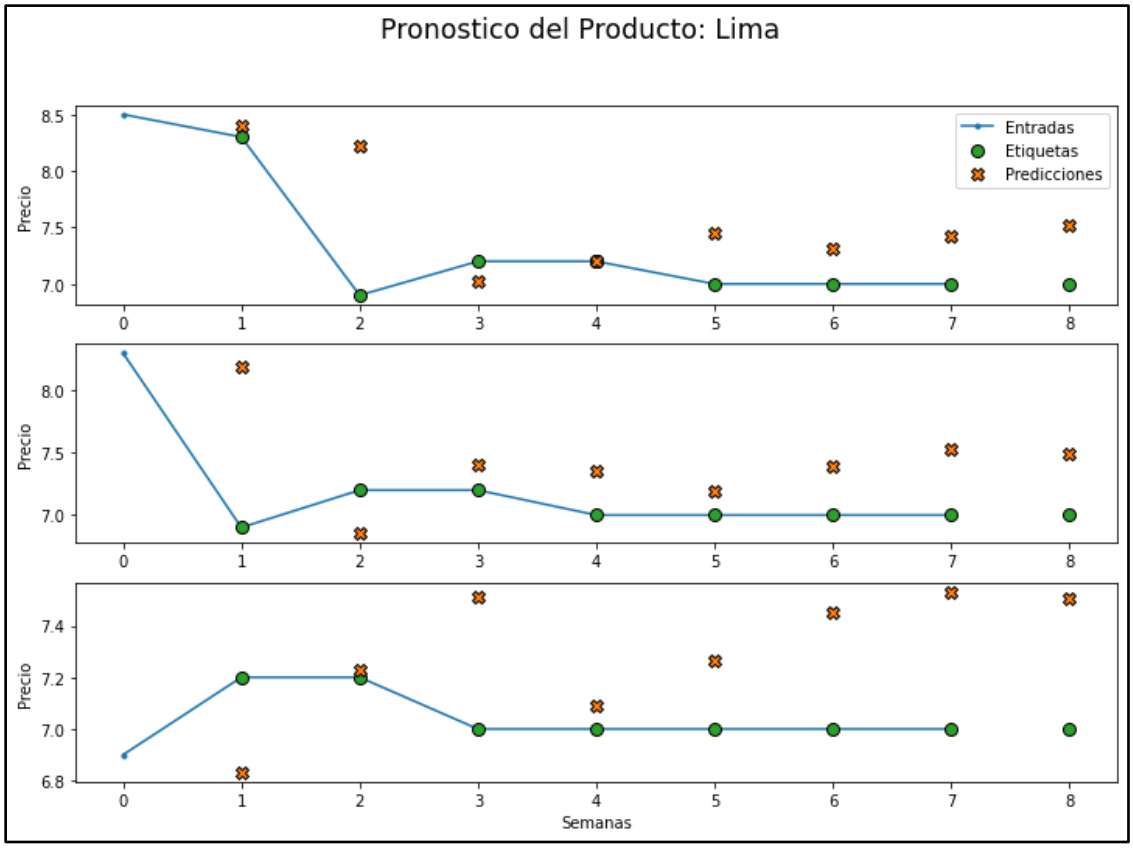

Pronostico del Producto: Tomate Saladette
Entradas
Etiquetas
Predicciones
Precio
Semanas

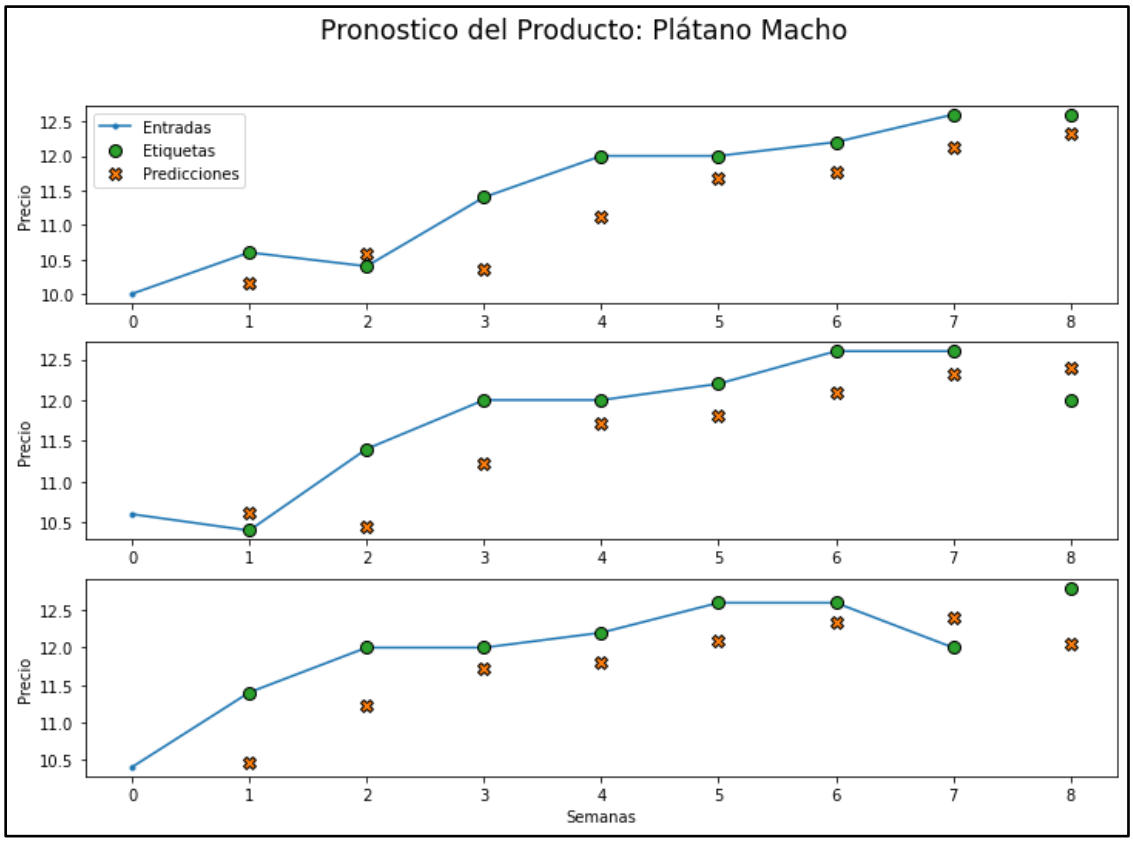

Pronostico del Producto: Naranja Valencia grande
Entradas
Etiquetas
Predicciones
Precio
Semanas

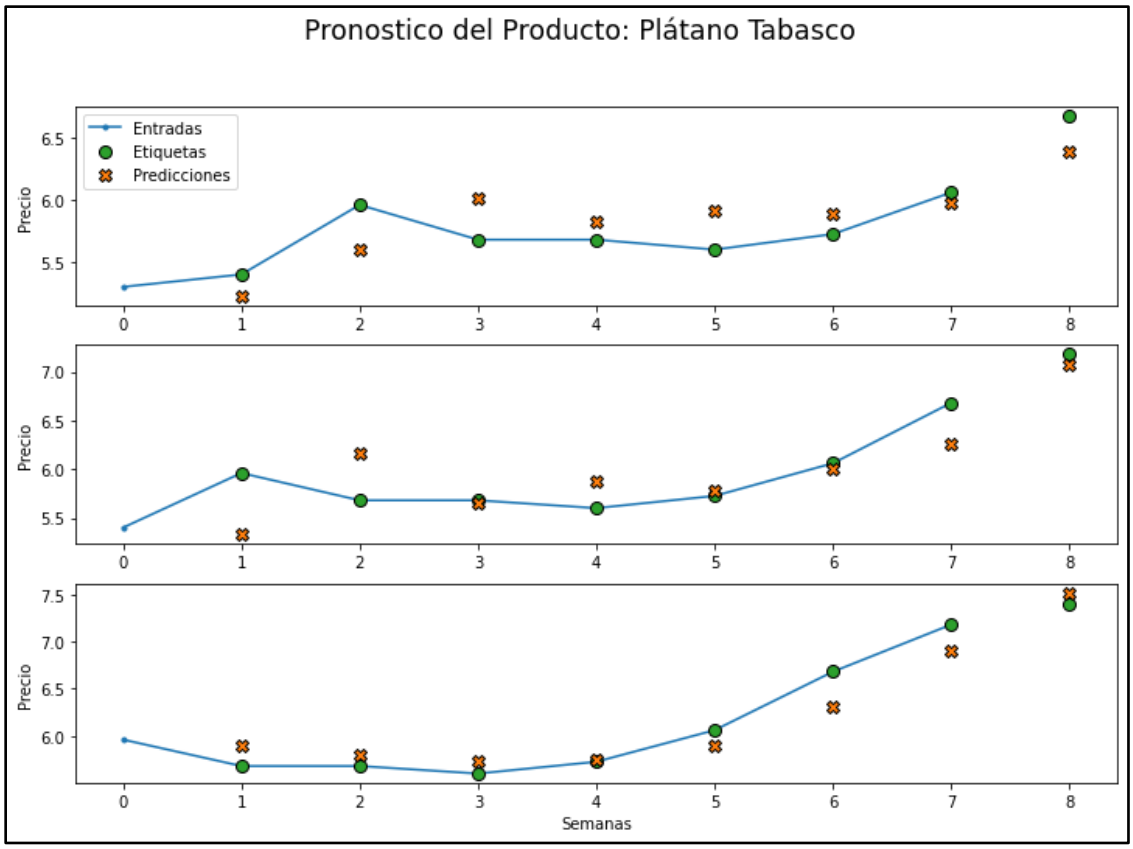

Pronostico del Producto: Aguacate Hass
Entradas
Etiquetas
Predicciones
Precio
Semanas

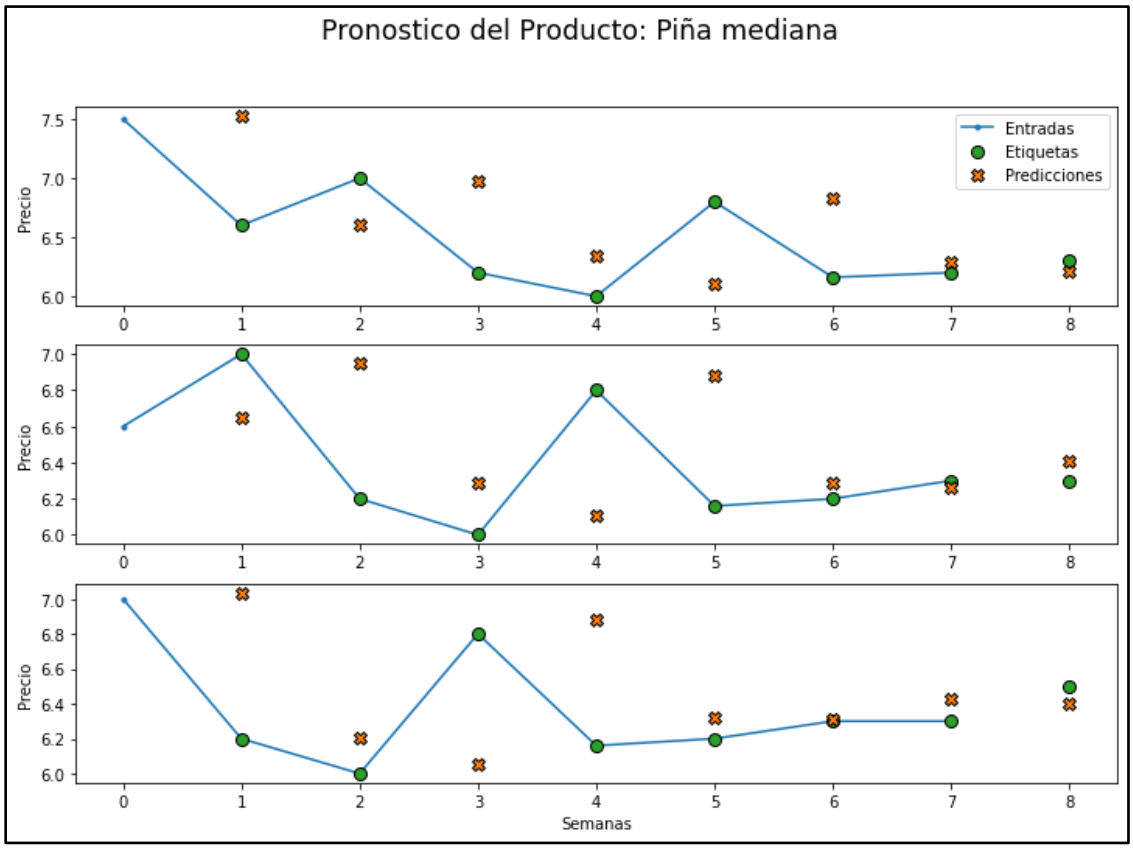

Pronostico del Producto: Kiwi
Entradas
Etiquetas
Predicciones
Precio
Semanas

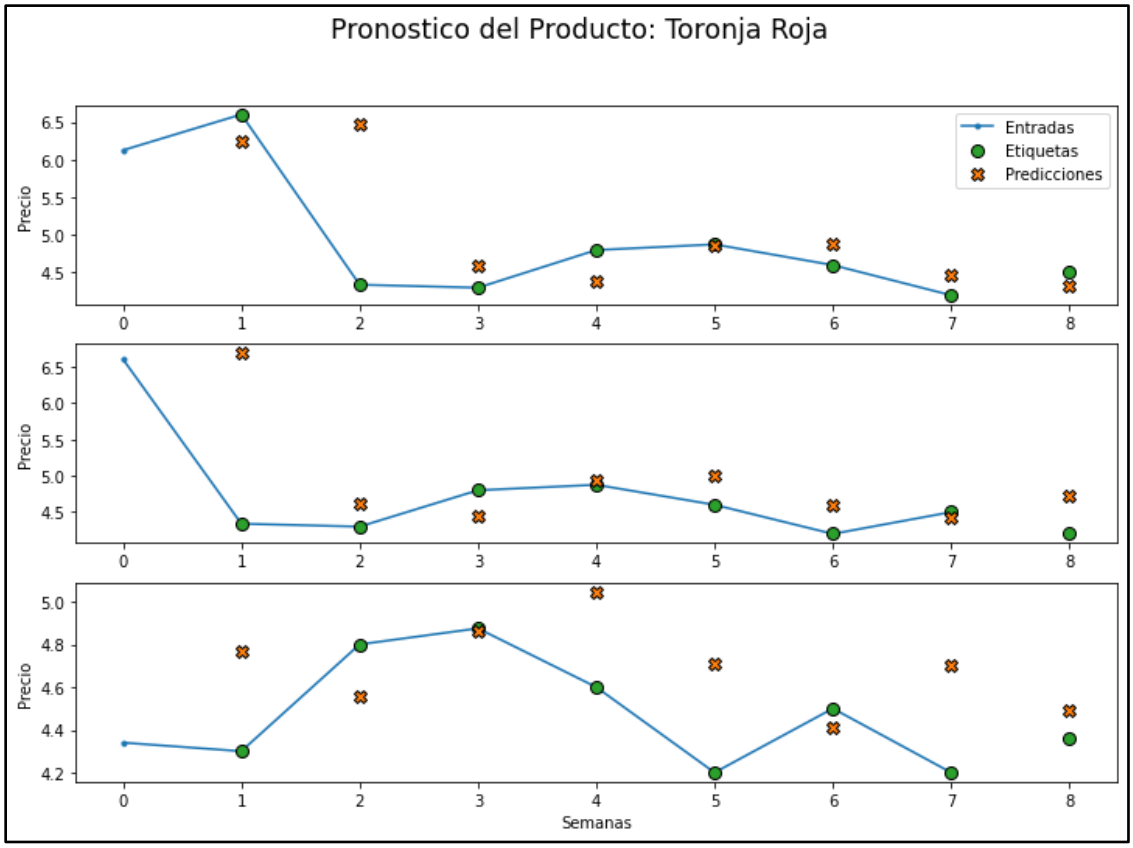

Pronostico del Producto: Limón
Entradas
Etiquetas
Predicciones
Precio
Semanas

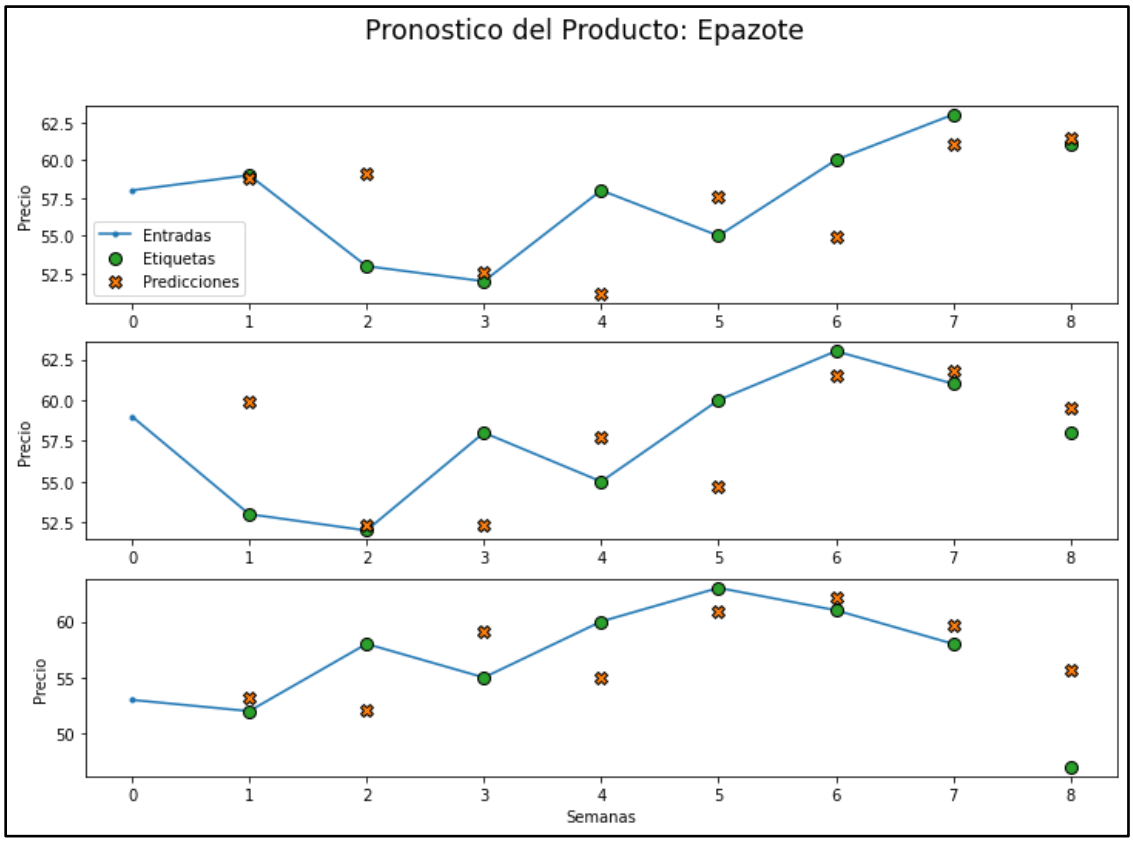

Pronostico del Producto: Lima
Entradas
Etiquetas
Predicciones
Precio
Semanas

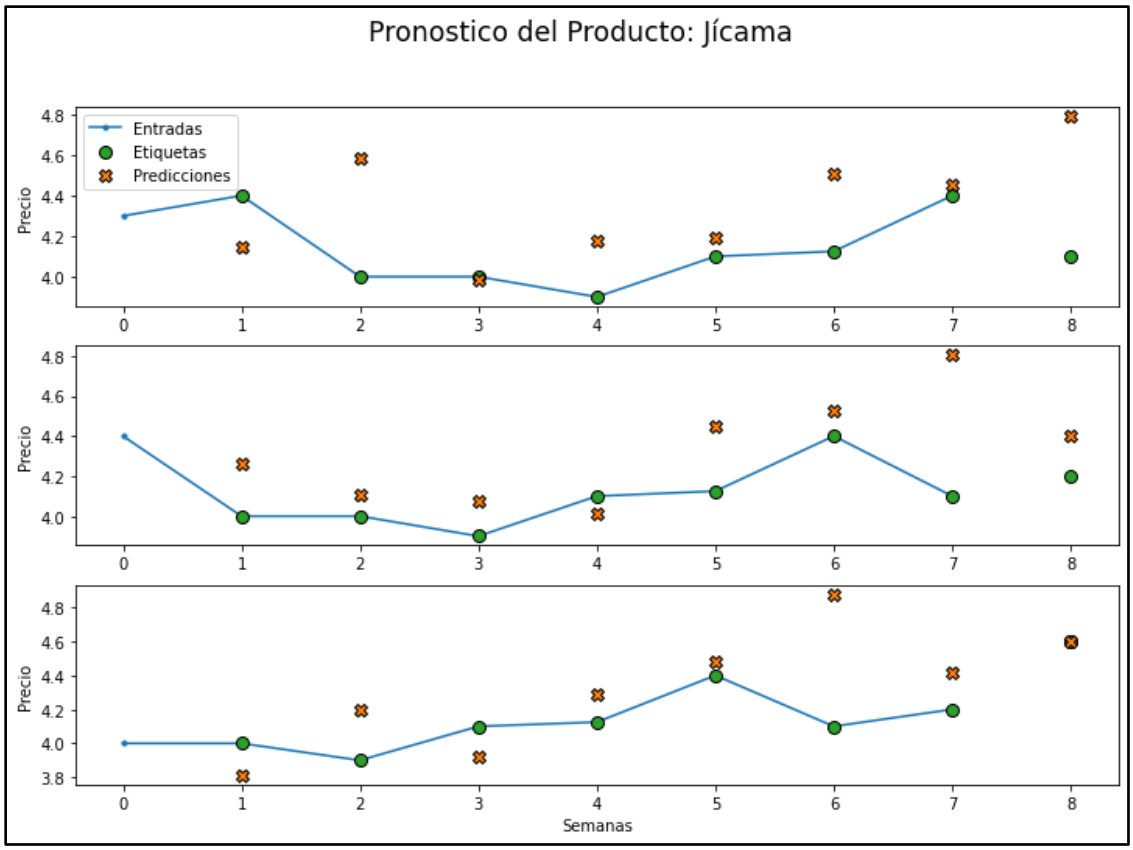

Pronostico del Producto: Plátano Macho
Entradas
Etiquetas
Predicciones
Precio
Semanas


Pronostico del Producto: Plátano Tabasco
Entradas
Etiquetas
Predicciones
Precio
Semanas

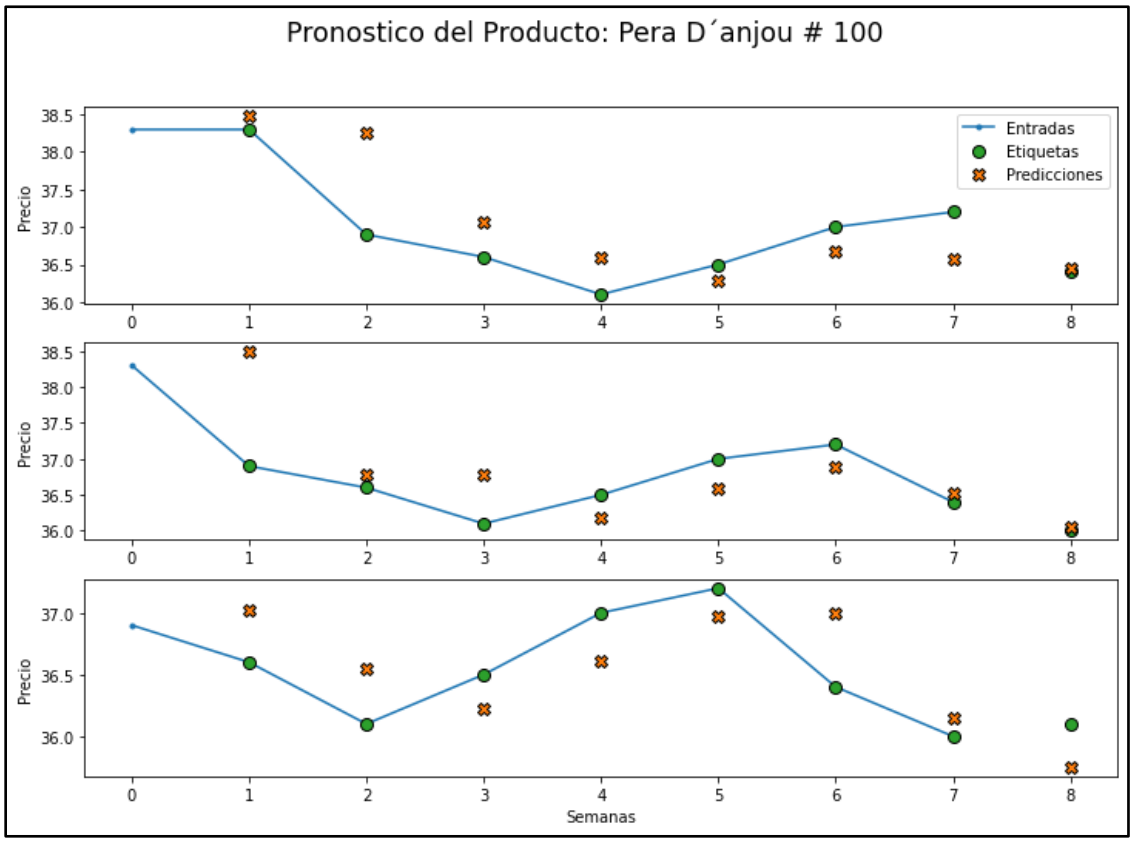

Pronostico del Producto: Piña mediana
Entradas
Etiquetas
Predicciones
Precio
Semanas

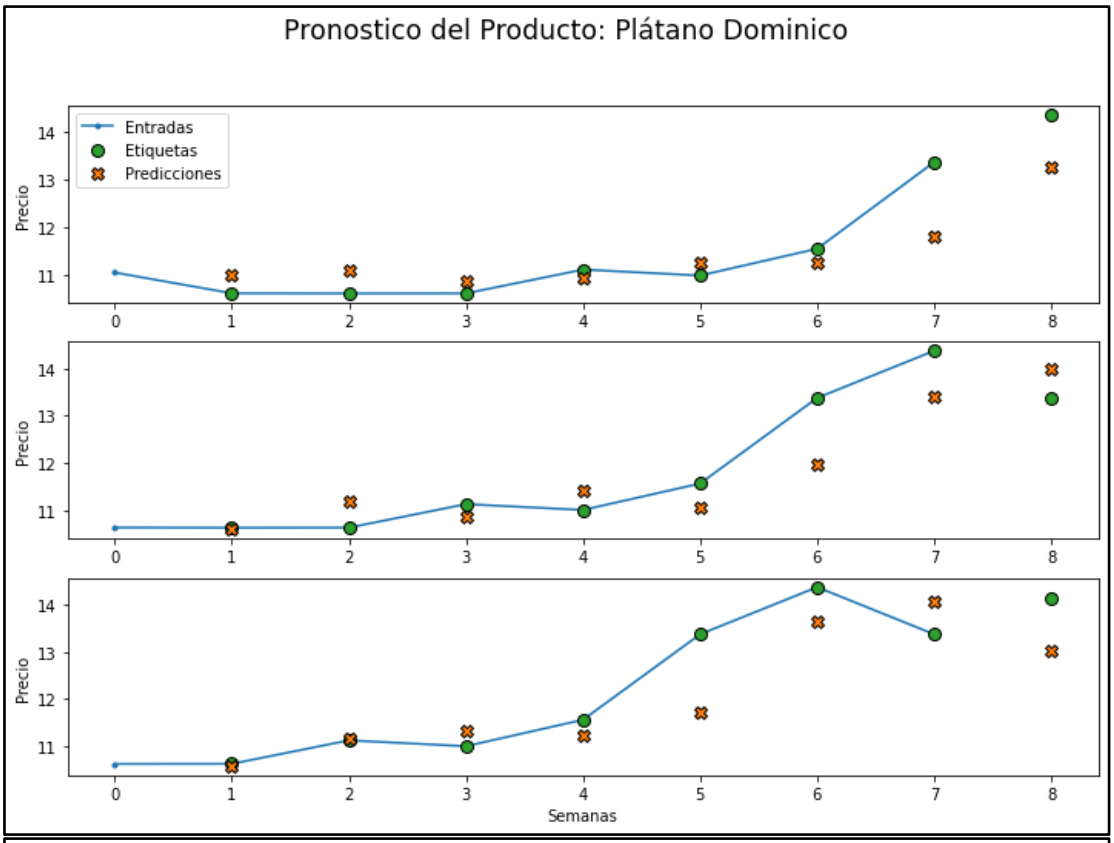

Pronostico del Producto: Toronja Roja
Entradas
Etiquetas
Predicciones
Precio
Semanas

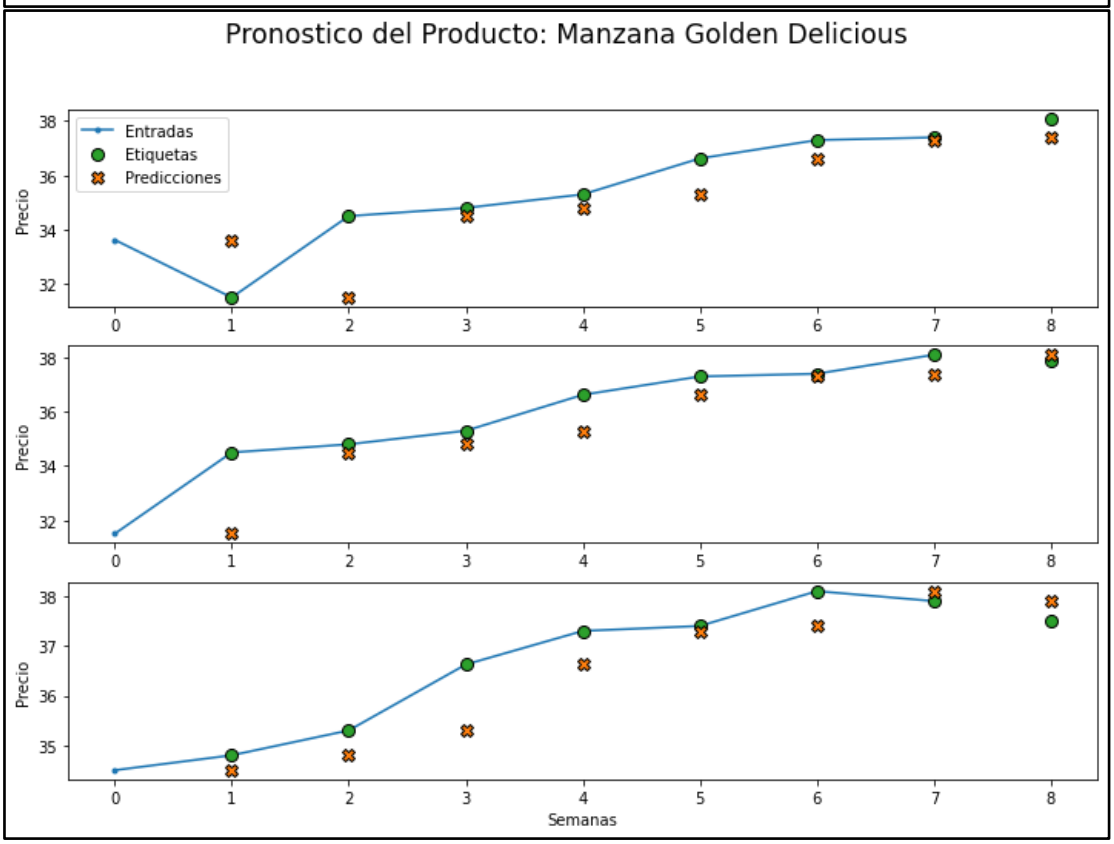

Pronostico del Producto: Epazote
Entradas
Etiquetas
Predicciones
Precio
Semanas

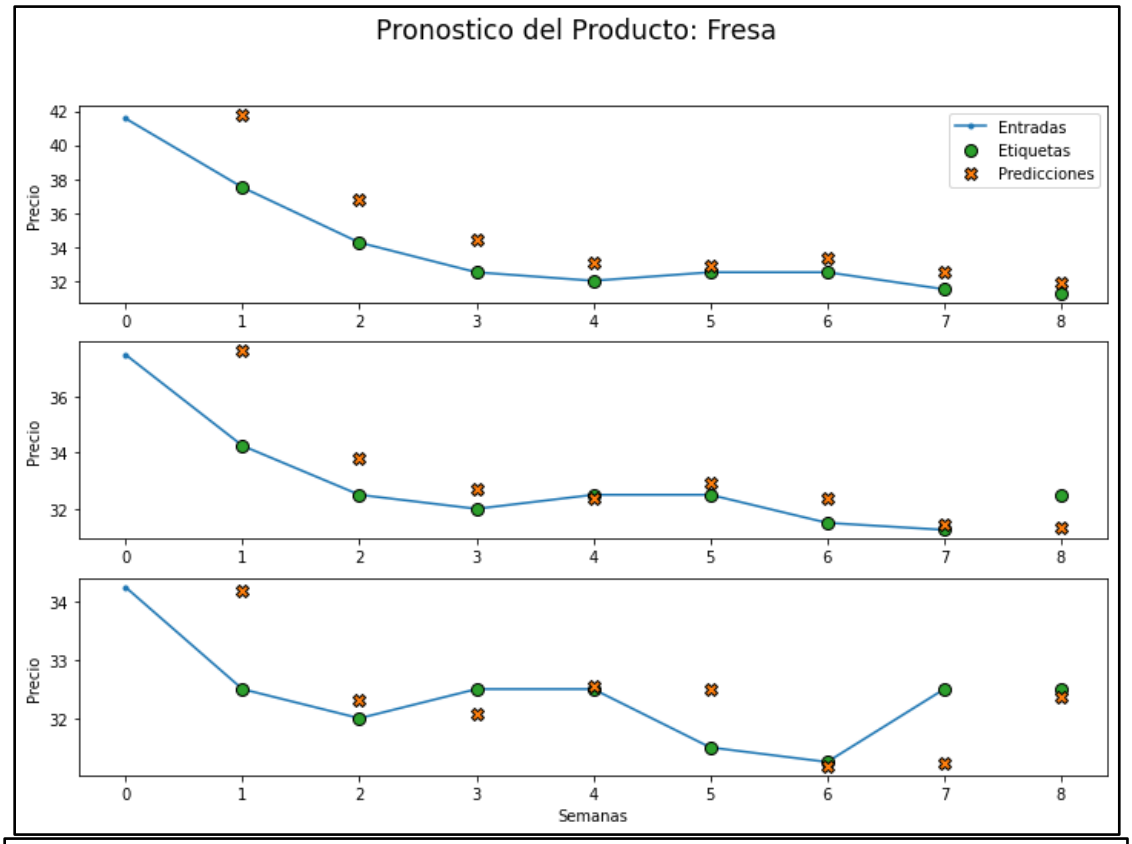

Pronostico del Producto: Jícama
Entradas
Etiquetas
Predicciones
Precio
Semanas

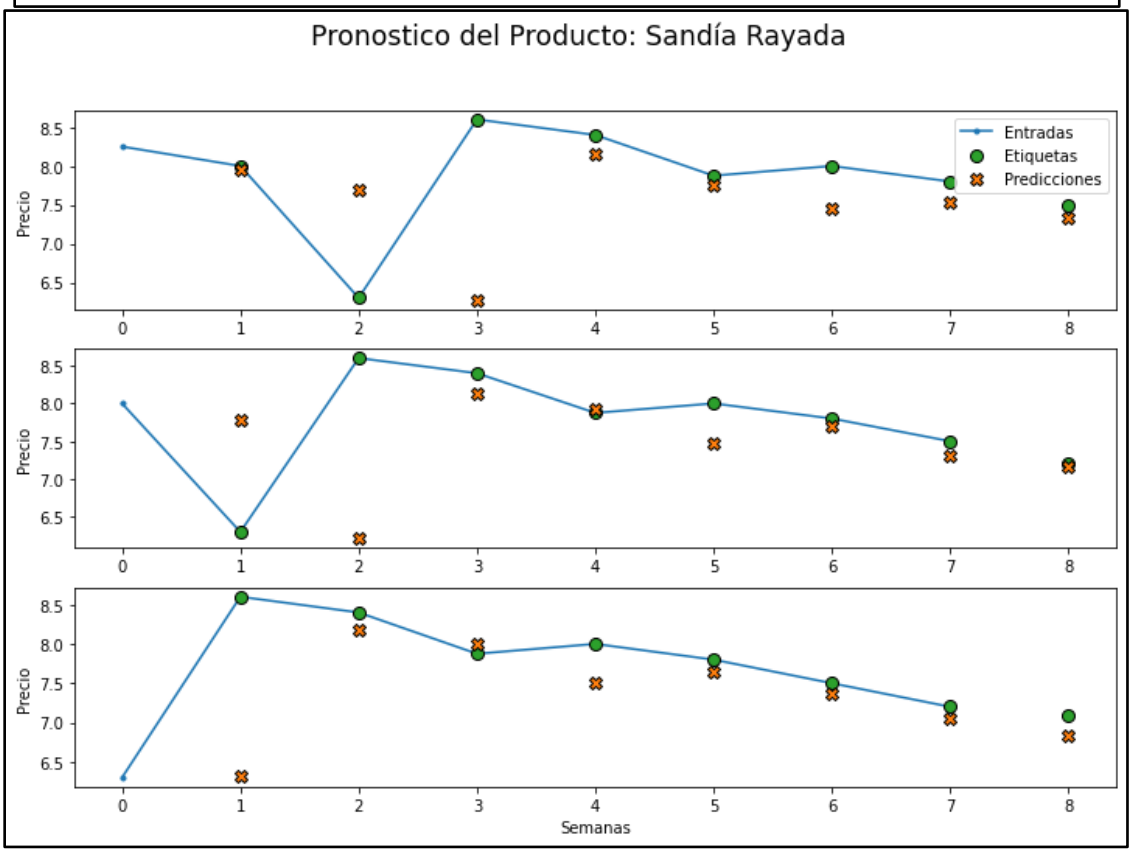
7.1.2. Productos que solo son evaluables con RNNs.

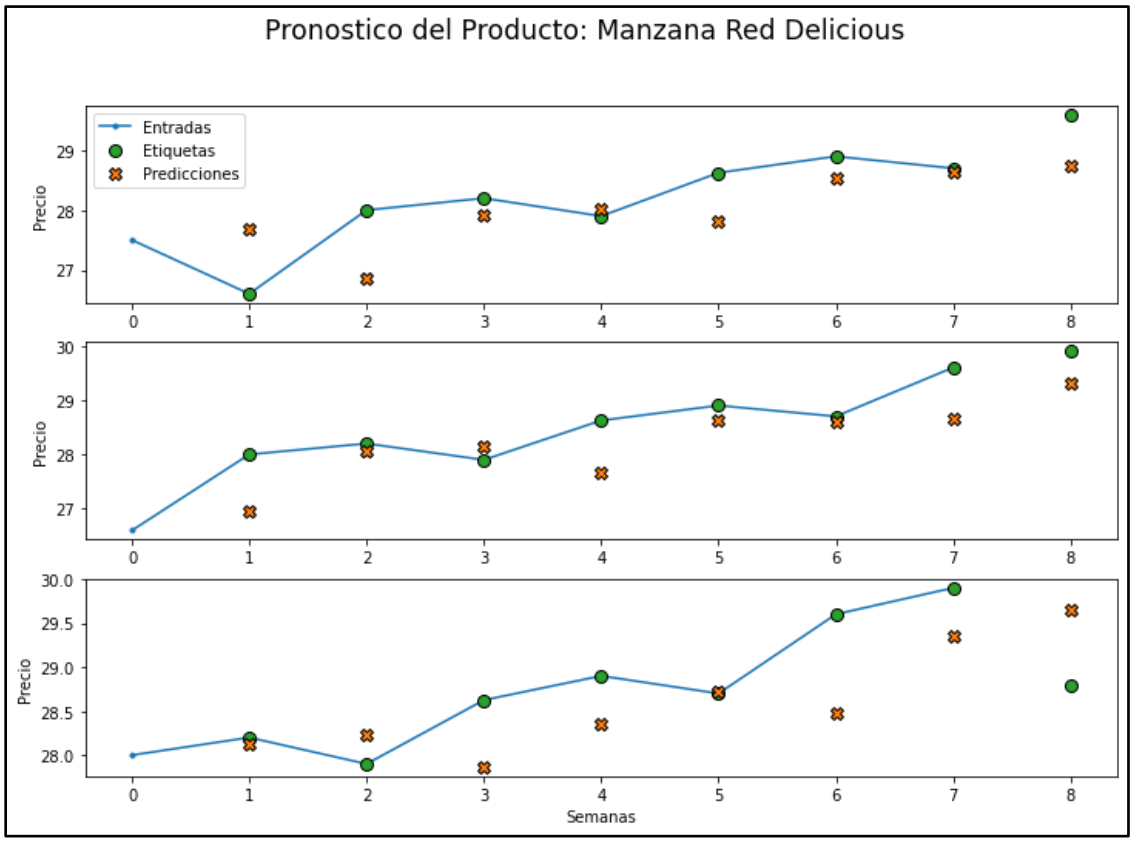



Pronostico del Producto: Plátano Dominico
Entradas
Etiquetas
Predicciones
Precio
Semanas


Pronostico del Producto: Manzana Golden Delicious
Entradas
Etiquetas
Predicciones
Precio
Semanas


Pronostico del Producto: Fresa
Entradas
Etiquetas
Predicciones
Precio
Semanas


Pronostico del Producto: Sandía Rayada
Entradas
Etiquetas
Predicciones
Precio
Semanas


Pronostico del Producto: Manzana Red Delicious
Entradas
Etiquetas
Predicciones
Precio
Precio
Precio
Semanas

7.2. Código Desarrollado.

7.2.1. Sección 1: Lógica del Proceso.

7.2.1.1. Bibliotecas y Atributos Iniciales.

```
## Importar Bibliotecas ##

import os
import datetime
import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
# II. Generador de Ventanas #
class WindowGenerator():
  def __init__(self, input_width, label_width, shift,
               train_df, val_df, test_df,
               label_columns=None):

    # Pasar los dataframes-parámetros como atributos
    self.train_df = train_df
    self.val_df = val_df
    self.test_df = test_df

    self.train_df_mean = self.train_df.mean()
    self.train_df_std = self.train_df.std()

    self.train_df = (self.train_df -
 self.train_df_mean) / self.train_df_std
    self.val_df = (self.val_df -
 self.train_df_mean) / self.train_df_std
    self.test_df = (self.val_df -
 self.train_df_mean) / self.train_df_std

    # Crea dos diccionarios nombre-
indice, uno con las etiquetas (si se ingresaron), y otro de las caract
erísticas del conjunto de entrenamiento.
```

```
    self.label_columns = label_columns
    if label_columns is not None:
      self.label_columns_indices = {name: i for i, name in
                                    enumerate(label_columns)}

    self.column_indices = {name: i for i, name in
                           enumerate(train_df.columns)}

    # Crea la anchura de la ventana de ENTRADA, el DESPLAZAMIENTO y la
 diferencia será la ventana de SALIDA (por lo tanto "shift >= 1" en SA
LIDA).
    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift

    self.total_window_size = input_width + shift

    self.input_slice = slice(0, input_width)
    self.input_indices = np.arange(self.total_window_size)[self.input_
slice]

    self.label_start = self.total_window_size -
 self.label_width # Evitar que de negativo respetando "label_width <=
total_window_size"
    self.labels_slice = slice(self.label_start, None)
    self.label_indices = np.arange(self.total_window_size)[self.labels
_slice]

  def __repr__(self):
    return '\n'.join([
        f'Total window size: {self.total_window_size}',
        f'Input indices: {self.input_indices}',
        f'Label indices: {self.label_indices}',
        f'Label column name(s): {self.label_columns}'])
```

7.2.1.2. Rebanador de Ventanas (Divisor).

```
 # Devuelve los datos en forma de ventana de acuerdo a los índices y c
olumnas LLAMÉMOSLE "EL REBANADOR" para UNA ventana
def split_window(self, features):
  inputs = features[:, self.input_slice, :]
  labels = features[:, self.labels_slice, :]
  if self.label_columns is not None:
    labels = tf.stack(
        [labels[:, :, self.column_indices[name]] for name in self.labe
l_columns],
        axis=-1)

  # Slicing no preserva información estática de la forma, hay que ajus
tar las formas
  # manualmente. De esta manera `tf.data.Datasets` son más fáciles de
inspeccionar.
  inputs.set_shape([None, self.input_width, None])
  labels.set_shape([None, self.label_width, None])

  return inputs, labels

WindowGenerator.split_window = split_window
```

7.2.1.3. Graficador.

```
# Graficar de manera individual la ventana
def plot(self, model=None, plot_col='Precio', max_subplots=3, mean=0,
std= 1, product_name='None'):

  # Extraer la ventana del conjunto de ejemplo
  inputs, labels = self.example

  plot_col_index = self.column_indices[plot_col] # Nos devuelve el índ
ice asociado a la columna (en el caso del precio es el cero)
  max_n = min(max_subplots, len(inputs)) # Representa el número de sub
graficos siendo 3 o menos (si las entradas (inputs) tienen menos de tr
es elementos)

  # Toda esta sección busca construir el gráfico: define el gráfico, l
os subgraficos, las etiquetas, etc.
  plt.figure(figsize=(12, 8)) # Crea la figura
  plt.suptitle(f'Pronostico del Producto: {product_name}', fontsize='x
x-large', weight='roman')
  for n in range(max_n):
```

```
    plt.subplot(max_n, 1, n+1) # Agrega un subgrafico a la figura actu
al (num_filas, num_columnas, índice)
    plt.ylabel(f'{plot_col}') # Etiqueta del eje y

    # Grafica en los ejes "x"(los indices de la ventana de entrada) y
"y" (precios correspondiente a "muestra n", y "columna plot_co_index")
    plt.plot(self.input_indices, (inputs[n, :, plot_col_index] * std)
+ mean,
             label='Entradas', marker='.', zorder=-10)

    # Aquí se busca asignar el índice de la columna a la variable "lab
el_col_index", primero busca validar si se especificó una etiqueta en
WindowGenerator mediante
    # el parámetro "self.label_columns", de ser así buscará la caracte
rística a graficar (el paramétro "plot_col") en el diccionario "label_
columns_indices" (creado al inicio del código),
    # en caso de contrario (que no haya etiqueta y por tanto no haya d
iccionario), "plot_col" buscará en el diccionario "label_index"

    if self.label_columns:
      label_col_index = self.label_columns_indices.get(plot_col, None)
    else:
      label_col_index = plot_col_index

    # Si no hay etiquetas (labels) porque no se específico en los pará
metros iniciales, el gráfico se salta a la siguiente etiqueta (para el
lo se usa el comando "continue")
    if label_col_index is None:
      continue

    # Ya una vez asegurado que existen datos para etiquetas, con las s
iguientes lineas se grafican, de esta manera se evitan errores de graf
icado
    plt.scatter(self.label_indices, (labels[n, :, label_col_index] * s
td) + mean,
                edgecolors='k', label='Etiquetas', c='#2ca02c', s=64)

    # Si se especifica un parámetro diferente de "None" en el modelo (
es decir que se esté evaluando un modelo), se grafican las prediccione
s de este
    # en caso contrario no grafican predicciones

    if model is not None:
      predictions = model(inputs)
```

```
      plt.scatter(self.label_indices, (predictions[n, :, label_col_ind
ex] * std) + mean,
                  marker='X', edgecolors='k', label='Predicciones',
                  c='#ff7f0e', s=64)

    # Crea una leyenda solo en la primer subgrafica.
    if n == 0:
      plt.legend()

  plt.xlabel('Semanas')

WindowGenerator.plot = plot
```

7.2.1.4. Generador de Ventanas (y aplicador del Divisor).

```
# Crear diferentes ventanas donde cada una será modificada por split_w
indow (EL REBANADOR)
def make_dataset(self, data):

  data = np.array(data, dtype=np.float32) # Convierte el DataFrame a N
umpy Array

  # Crea  ventanas COMPLETAS y las almacena en un lote (Dataset)
  ds = tf.keras.preprocessing.timeseries_dataset_from_array(
      data=data,
      targets=None,
      sequence_length=self.total_window_size,
      sequence_stride=1,
      sampling_rate=1,
      shuffle=False,
      batch_size=data.shape[0])

  # Rebana cada ventana (".map" aplica EL REBANADOR a cada elemento/ve
ntana) y los vuelve a guardar en el lote (Dataset)
  ds = ds.map(self.split_window)

  return ds

WindowGenerator.make_dataset = make_dataset

# Hacer tres DATASETS de Vetanas Rebanadas para los conjuntos entrenam
iento, validación y prueba (y uno extra de ejemplo para graficar)
@property
def train(self):
  return self.make_dataset(self.train_df)
```

```
@property
def val(self):
  return self.make_dataset(self.val_df)

@property
def test(self):
  return self.make_dataset(self.test_df)

@property
def example(self):
  """Obtiene y guarda un lote de ejemplos `inputs, labels` para grafic
ar."""
  result = getattr(self, '_example', None)
  if result is None:
    # Si el ejemplo no se encuentre se obtiene del dataset `.train`
    result = next(iter(self.train))
    # Lo guarda para la siguiente vez
    self._example = result
  return result


WindowGenerator.train = train
WindowGenerator.val = val
WindowGenerator.test = test
WindowGenerator.example = example
```

7.2.1.5. Modelo de Predicción.

```
## III. Creación de Modelos. ##

MAX_EPOCHS=1500
def compile_and_fit(model, window, patience=5):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss'
,
                                                    patience=patience,
                                                    mode='min')

  model.compile(loss=tf.losses.MeanSquaredError(),
                optimizer=tf.optimizers.RMSprop(),
                metrics=[tf.metrics.MeanAbsoluteError()])

  history = model.fit(window.train, epochs=MAX_EPOCHS,
                      validation_data=window.val,
                      callbacks=[early_stopping], verbose=False)
```

```
  return history


# #  Modelo Base (Modelo Ingenuo) # #
class Baseline(tf.keras.Model):
  def __init__(self, label_index=None):
    super().__init__()
    self.label_index = label_index

  def call(self, inputs):
    if self.label_index is None:
      return inputs
    result = inputs[:, :, self.label_index] # Filtra las característic
as a una
    return result[:, :, tf.newaxis] # Agrega una dimension extra al fi
nal



# #  Red Recurrente SimpleRNN # #
simple_rnn = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, simplernn_units]
    tf.keras.layers.SimpleRNN(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])
```

7.2.2. Sección 2: Experimentación con Datos.

7.2.2.1. Carga de Datos.

```
## I. Carga y preprocesamiento ##

# Abrir el archivo desde Drive
prod_dir = '/content/drive/MyDrive/Datos/Queretaro_Products4.csv'
white_dir = '/content/drive/MyDrive/Datos/white_noise.csv'
f = open(prod_dir, encoding='latin_1')
g = open(white_dir, encoding='latin_1')

# Leer el CSV, guardarlo como un dataframe, ponerle un nombre y elimin
ar columnas que no considere necesarias
prod_dataframe = pd.read_csv(f, header=None, sep=',', names=['Fecha',
'Producto', 'Precio', 'Stub'] , index_col='Producto')
prod_dataframe.drop(labels='Stub', axis=1, inplace=True)
prod_dataframe.columns.name = 'Product Batch'

prod_white = pd.read_csv(g, header=None, sep=',', names=['Fecha', 'Pro
ducto', 'Precio', 'Stub'] , index_col='Producto')
prod_white.drop(labels='Stub', axis=1, inplace=True)
prod_white.columns.name = 'White Batch'

# Guardar cada producto en un array
all_products = np.array(prod_dataframe.index.unique())
all_white = np.array(prod_white.index.unique())
print(pd.DataFrame(all_products, ))
print(pd.DataFrame(all_white))
```

7.2.2.2. Aplicador del Modelo.

```
# III. Aplicación del Modelo

class AppliedWindow:

  def __init__(self, product_index, start_year, train_percent, val_per
cent, data, input_width=8, label_width=8, shift=1, label_columns=['Pre
cio']):
    self.product_index = product_index
    self.start_year = start_year
    self.train_percent = train_percent
    self.val_percent = val_percent
    self.data = data
    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift
```

```
    self.label_columns = label_columns
    self.all_products = np.array(data.index.unique()) # Lista todos lo
s productos
    self.product_name = self.all_products[self.product_index]

  def __call__(self):

    # 1. Generar el DataFrame #
    self.product = self.data.loc[self.all_products[self.product_index]
]
    self.product.name = self.all_products[self.product_index] # Nombra
r el DataFrame
    dt = pd.to_datetime(self.product.pop('Fecha'), dayfirst=True) # Ex
traer la fecha en como objeto DateTime
    self.product.index = dt # Colocar el objeto DateTime como índice
    self.product = self.product.resample('W').mean() #Obtener los prom
edios semanales
    self.product = self.product[self.start_year:]
    self.column_indices = {name: i for i, name in enumerate(self.produ
ct.columns)} # Generar un diccionario con la relación índice/columna

    # 2. Separar los datos, Nota: es importante monitorear que el conj
unto de testeo sea lo suficientemente ancho para lo que requiere la ve
ntana#
    self.n_periods = len(self.product)
    self.product_train = self.product[0 : int(self.n_periods * self.tr
ain_percent)]
    self.product_val = self.product[int(self.n_periods * self.train_pe
rcent) : int(self.n_periods * (self.train_percent+self.val_percent))]
    self.product_test = self.product[int(self.n_periods * (self.train_
percent+self.val_percent)) : None]


    # 3. Normalizar los datos #
    self.product_train_mean = self.product_train.mean()
    self.product_train_std = self.product_train.std()

    self.product_train = (self.product_train -
 self.product_train_mean) / self.product_train_std
    self.product_val = (self.product_val -
 self.product_train_mean) / self.product_train_std
    self.product_test = (self.product_test -
 self.product_train_mean) / self.product_train_std
```

```
    # 4. Crear un Modelo de Desplazamiento #
    self.n_step_window = WindowGenerator(train_df=self.product_train,
val_df=self.product_val, test_df=self.product_test,
                                         input_width=self.input_width,
 label_width=self.label_width, shift=self.shift, label_columns=self.la
bel_columns)



    # 5. Rendimiento  con el modelo base #
    self.val_performance = {}
    self.performance = {}

    print('****MODELO BASE****')
    baseline = Baseline(label_index=self.column_indices['Precio'])
    baseline.compile(loss=tf.losses.MeanSquaredError(),
                     metrics=[tf.metrics.MeanAbsoluteError()])
    self.val_performance['Baseline'] = baseline.evaluate(self.n_step_w
indow.val, verbose=True)
    self.performance['Baseline'] = baseline.evaluate(self.n_step_windo
w.test, verbose=True)
    self.n_step_window.plot(baseline, mean=self.product_train_mean, st
d=self.product_train_std, product_name=self.product_name)

    #6. Rendimiento con la red recurrente ##
    print('****MODELO RECURRENTE****')
    history = compile_and_fit(simple_rnn, self.n_step_window)
    IPython.display.clear_output()
    self.val_performance['simple_rnn'] = simple_rnn.evaluate(self.n_st
ep_window.val, verbose=False)
    self.performance['simple_rnn'] = simple_rnn.evaluate(self.n_step_w
indow.test, verbose=False)
    self.n_step_window.plot(model=simple_rnn, mean=self.product_train_
mean, std=self.product_train_std, product_name=self.product_name)



    return print("El rendimiento del método de redes neuronales es:",
self.performance['simple_rnn'][0])
```