Universidad Jesuita de Guadalajara
División de Física y Matermáticas
Maestría en Ciencia de Datos

ITESO, Universidad Jesuita de Guadalajara

Análisis de series de tiempo con base en datos publicos del servicio de transporte público de MIBICI

Proyecto que presenta
Marco Antonio Mendoza Ramos

INVESTIGACION, DESARROLLO E INNOVACION II

Guadalajara, Jalisco, Mayo 2023

1. Introducción

En el año de 2014, el gobierno del estado de Jalisco lanzó una iniciativa llamada MIBICI que buscaba ser un nuevo medio de transporte ecológico en el que las y los Jaliciences pudieran moverse por medio de la renta de bicicletas en estaciones alrededor de tres de las principales ciudades de la Zona Metropolitana de Guadalajara (ZMG).

La ZMG es una de las entidades con mayor crecimiento y desarrollo de México. A lo largo de los últimos 30 años, el crecimiento medio anual población ha sido de alrededor del 2 %, lo que significa un aumento de casi medio millón de personas anualmente en la actualidad. Con dicho crecimiento, muchos servicios de transporte, incluído el de MIBICI, han aumentado su demanda y buscado ampliarse para poder brindar el servicio a todos los transeuntes que buscan movilizarse todos los días.

En este proyecto, se hará un análisis detallado usando los datos públicos del servicio de MIBICI desde sus inicios (Diciembre 2014) hasta el mes de Marzo del 2023. Uno de los obtjetivos del análisis será localizar las áreas o estaciones con mayor demanda de este servicio. Posteriormente, con dichos datos y como objetivo principal, se entrenará una red neural de tipo LSTM que permitirá evaluar el crecimiento a futuro de forma global y localizada, brindando así una posible solución a la problemática del crecimiento exponencial y al desabasto de bicicletas en el futuro cercano.

2. Marco Teórico

2.1. Series de tiempo

Una serie de tiempo es una secuencia de datos numéricos recopilados en intervalos regulares a lo largo de un período de tiempo. En otras palabras, es una colección de observaciones tomadas de forma secuencial en el tiempo. Las series de tiempo son comunes en muchos campos, como economía, finanzas, meteorología, ciencias de la salud y ciencias sociales, entre otros.

El análisis de series de tiempo es una técnica estadística que busca identificar patrones, tendencias, ciclos y estacionalidades en los datos para predecir eventos futuros, comprender el comportamiento pasado y tomar decisiones informadas. Algunos de los componentes más comunes en las series de tiempo incluyen la tendencia (un patrón de crecimiento o decrecimiento a lo largo del tiempo), la estacionalidad (variaciones regulares que ocurren en un período específico, como las estaciones del año), y el ruido (variaciones aleatorias o impredecibles).

Existen diferentes métodos para analizar y modelar series de tiempo, como modelos autorregresivos, modelos de promedios móviles, modelos de suavizado exponencial y modelos ARIMA (Modelo autorregresivo integrado de promedios móviles), entre otros. Estos enfoques permiten a los investigadores y analistas obtener información valiosa de las series de tiempo y realizar pronósticos más precisos.

2.2. ARIMA y sus derivados.

El modelo ARIMA (Modelo Autorregresivo Integrado de Promedios Móviles) es una técnica estadística popular para analizar y predecir series de tiempo. El modelo ARIMA combina tres componentes principales: modelos autorregresivos (AR), modelos de promedios móviles (MA) e integración (I).

1. Autorregresivo (AR): El componente AR modela la relación entre una observación en el tiempo actual y sus observaciones pasadas. Un modelo AR(p) utiliza "p" observaciones anteriores para predecir la observación actual.

2. Promedios móviles (MA): El componente MA modela la relación entre una observación en el tiempo actual y los errores de pronóstico pasados. Un modelo MA(q) utiliza "q" errores pasados para predecir la observación actual.

3. Integración (I): El componente de integración se refiere al proceso de diferenciación para hacer que la serie de tiempo sea estacionaria. La estacionariedad es una propiedad importante en el análisis de series de tiempo, ya que muchos modelos estadísticos suponen que las series son estacionarias. Un modelo ARIMA(d) utiliza "d" diferencias para lograr

la estacionariedad.

El modelo ARIMA se denota como ARIMA(p,d,q), donde "p" es el orden del componente autorregresivo, "d" es el grado de diferenciación y "q" es el orden del componente de promedios móviles.

Hay varias extensiones y variantes del modelo ARIMA, que incluyen:

1. SARIMA (Modelo Autorregresivo Integrado de Promedios Móviles Estacionales): Este modelo extiende el ARIMA básico al incluir componentes estacionales en los términos AR y MA. Se denota como SARIMA(p,d,q)(P,D,Q)s, donde "P", "D" y "Q" son los órdenes estacionales de los componentes AR, I y MA, respectivamente, y "s" es la periodicidad estacional de la serie.

2. ARIMAX (Modelo Autorregresivo Integrado de Promedios Móviles con Variables Exógenas): Este modelo extiende el ARIMA básico al incluir una o más variables exógenas que pueden ayudar a mejorar el pronóstico de la serie de tiempo.

3. SARIMAX (Modelo Autorregresivo Integrado de Promedios Móviles Estacionales con Variables Exógenas): Este modelo combina las características de SARIMA y ARIMAX, incluyendo componentes estacionales y variables exógenas en el análisis y pronóstico de la serie de tiempo.

Estos modelos y sus derivados pueden aplicarse a una amplia variedad de series de tiempo, como datos económicos, financieros, climáticos y de ventas, para identificar patrones, tendencias y predecir eventos futuros.

2.3. Redes Neuronales

Una red neuronal es un modelo de aprendizaje automático inspirado en la estructura y funcionamiento del cerebro humano. Su objetivo es imitar la forma en que el cerebro procesa información, permitiendo a las máquinas aprender y adaptarse a partir de datos de entrada. Las redes neuronales son especialmente útiles para abordar problemas complejos y no lineales, como reconocimiento de imágenes, procesamiento del lenguaje natural, predicción y clasificación.

Una red neuronal está compuesta por unidades de procesamiento llamadas neuronas, que están organizadas en capas. Existen tres tipos de capas en una red neuronal:

1. Capa de entrada: Recibe los datos de entrada y los pasa a la siguiente capa. Cada neurona en esta capa representa una característica de los datos de entrada.

2. Capas ocultas: Estas capas se encuentran entre la capa de entrada y la capa de salida. Pueden haber varias capas ocultas en una red neuronal, y cada capa puede contener múltiples neuronas. En estas capas, se realizan transformaciones no lineales y se extraen características de alto nivel de los datos.

3. Capa de salida: Esta capa proporciona el resultado final de la red neuronal, como una predicción, clasificación o puntuación. El número de neuronas en la capa de salida depende del tipo de problema que se esté resolviendo (por ejemplo, clasificación binaria, clasificación multiclase o regresión).

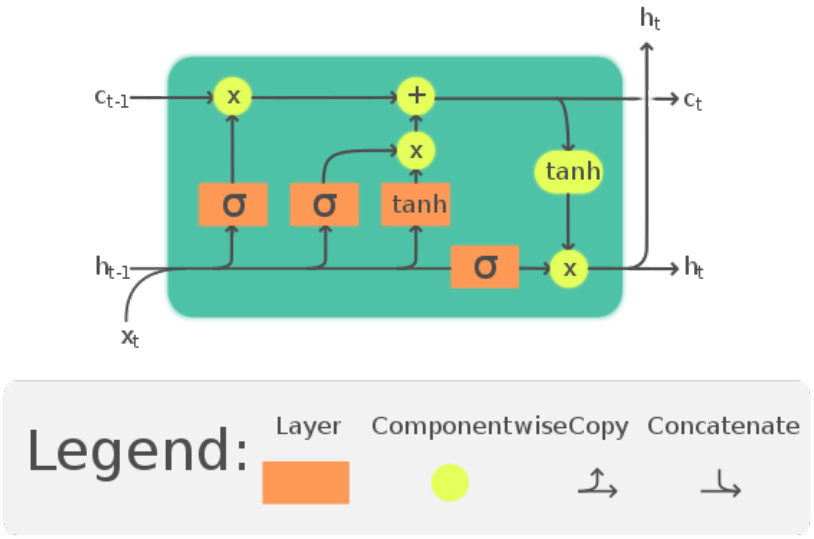
Cada neurona en una red neuronal está conectada a otras neuronas mediante enlaces ponderados, que representan la "fuerza" de la conexión entre las neuronas. Estos enlaces tienen pesos asociados, que son los parámetros que la red neuronal aprende durante el proceso de entrenamiento. El aprendizaje en una red neuronal implica ajustar los pesos de las conexiones para minimizar la diferencia entre las predicciones de la red y los valores objetivo reales.

El proceso de aprendizaje en una red neuronal generalmente implica el uso de un algoritmo de optimización, como el descenso de gradiente, y una función de pérdida que mide la discrepancia entre las predicciones de la red y los valores reales. A medida que la red neuronal se entrena, los pesos se actualizan iterativamente para reducir la función de pérdida y mejorar el rendimiento del modelo.

Existen varios tipos de redes neuronales, como las redes neuronales convolucionales (CNN), las redes neuronales recurrentes (RNN) y las redes neuronales profundas (DNN), cada una diseñada para abordar diferentes tipos de problemas y conjuntos de datos.

2.4. LSTMs

Una LSTM (Long Short-Term Memory) es un tipo especial de red neuronal recurrente (RNN) diseñada para abordar el problema del desvanecimiento del gradiente en el aprendizaje de secuencias largas. Las RNN convencionales pueden tener dificultades para aprender dependencias a largo plazo en secuencias debido a este problema, lo que limita su capacidad para modelar eficazmente información a lo largo de grandes intervalos de tiempo.



Las LSTM fueron introducidas por Sepp Hochreiter y Jürgen Schmidhuber en 1997 y han demostrado un rendimiento superior en tareas que involucran secuencias de datos, como el procesamiento del lenguaje natural, la generación de texto, la traducción automática, el reconocimiento de voz y la predicción de series de tiempo.

La principal innovación de las LSTM es la adición de una estructura de memoria llamada celda de memoria en cada unidad de la red. La celda de memoria contiene tres puertas (compuertas) que controlan el flujo de información dentro y fuera de la celda:

1. Puerta de entrada (input gate): Esta puerta determina cuánta información nueva de la entrada actual se incorporará a la celda de memoria.

2. Puerta de olvido (forget gate): Esta puerta controla cuánta información se descarta de la celda de memoria en función de la entrada actual y el estado oculto anterior.

3. Puerta de salida (output gate): Esta puerta determina cuánta información de la celda de memoria se utiliza para calcular el estado oculto siguiente y la salida de la unidad.

Estas puertas permiten a las LSTM mantener información relevante a lo largo de secuencias largas y descartar información irrelevante, lo que las hace más efectivas para aprender dependencias a largo plazo en comparación con las RNN convencionales.

Las LSTM se pueden utilizar en una variedad de arquitecturas de red, como secuencia a secuencia (sequence-to-sequence), codificador-decodificador (encoder-decoder) y redes bidireccionales (bidirectional), para abordar diferentes tipos de problemas que involucren secuencias de datos.

3. Implementación

3.1. Explración y análisis de los datos. (EDA)

Para este proyecto fueron usados los datos públicos de MIBICI, que constan de datos recabados desde Diciembre del 2014 con el último registro en Marzo del 2023. Así mismo, para los datos geoespaciales, fue utilizado un dataset de nomenclatura que contenía el identificador de cada una de las estaciones, así como las ubicaciones y estatus de cada una de ellas.

Para poder trabajar los datos como un completo primero se procedió a generar un dataset completo que contuviera toda la información, para ello se utilizó el código en python mostrado a continuación:

```python
import pandas as pd
import os

# Define la ruta base donde se encuentran los archivos CSV
ruta_base = "../datos/"

# Obtén la lista de subdirectorios en la ruta base
subdirs = os.listdir(ruta_base)
```

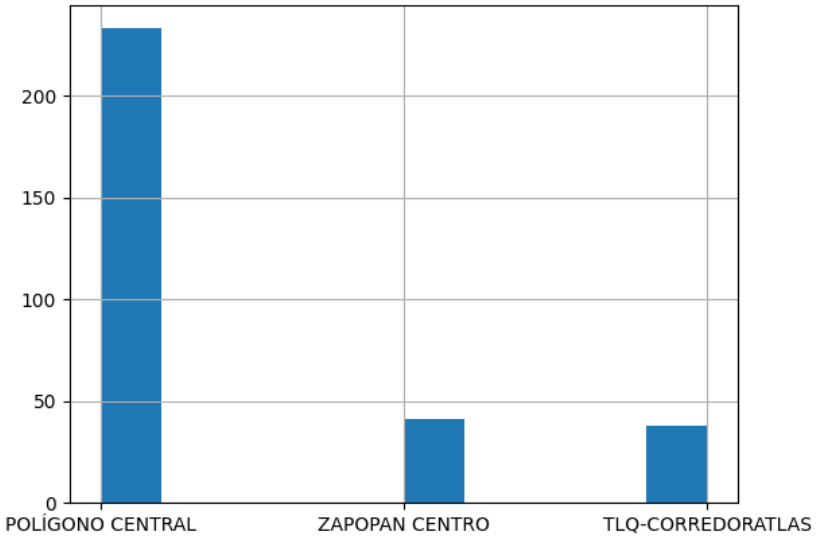
```
# Crea un DataFrame vacío para almacenar los datos combinados
df = pd.DataFrame()

# Para cada subdirectorio en la ruta base
for subdir in subdirs:
    # Construye la ruta completa al subdirectorio actual
    subdir_path = os.path.join(ruta_base, subdir)

    # Si la ruta actual es un directorio (ignora los
    # archivos en la ruta base)
    if os.path.isdir(subdir_path):
        # Obtén la lista de archivos CSV en
        # el subdirectorio actual
        csv_files = [f for f in os.listdir(subdir_path)
                     if f.endswith('.csv')]

        # Para cada archivo CSV en el subdirectorio actual
        for csv_file in csv_files:
            # Construye la ruta completa al archivo
            # CSV actual
            csv_path = os.path.join(subdir_path,
                                    csv_file)

            # Lee el archivo CSV en un DataFrame
            data = pd.read_csv(csv_path,
                               encoding='latin-1')

            # Agrega el DataFrame al DataFrame general
            df = pd.concat([df, data])
```

posterior a ello, fueron corregidos algunos elementos de una columna duplicada que contenía información relevante, esto debido al encoding en el que los datos fueron guardados. Finalmente, con los datos juntos y sin duplicados, se procedió a guardar el contenido del mismo en una archivo unificado.

```
df['Año_de_nacimiento'] =
    df['Año_de_nacimiento'].fillna(df['AÃ±o_de_nacimiento'])
df = df.drop('AÃ±o_de_nacimiento', axis=1)
```

```
df.to_csv("../datos/complete.csv", index=False)
```

tomando en cuenta los datos geospaciales, podemos observar que existen tres grandes zonas en las que las estaciones se encuentran divididas. La imagen a continuación mmuestra la proporción de dichas estaciones en las tres zonas.



para crear una mejor visualización de los mismos, se utilizó una paquetería de Python llamada `folium`, la cual dentro de un notebook puede interactivamente crear imágenes con punteros espaciales y que permite entre otras cosas, hacer zoom y seleccionar lugares sin la necesidad de utilizar una API directa de Google Maps. El código debajo genera dicha imagen interactiva que más adelante se muestra con fines meramente de ilustración.

```
import folium

df_nom = pd.read_csv("nomenclatura_2023_02.csv",
                     encoding='latin-1')

# Elimina las filas duplicadas basadas en la columna
# de latitud y longitud
```

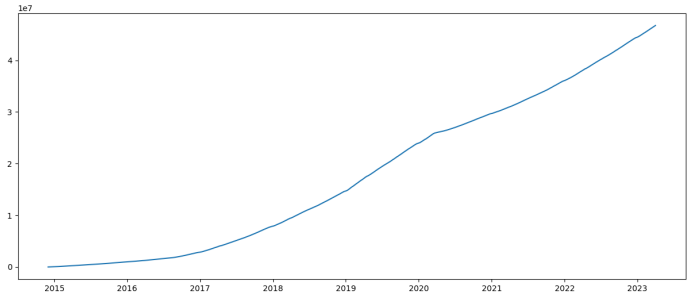
```
df_unique = df_nom.drop_duplicates(subset=['latitude',
                                           'longitude'],
                                   keep='first')

# Crea un mapa centrado en Guadalajara Jalisco
mapa = folium.Map(location=[20.659698, -103.349609],
                  zoom_start=12)

# Agrega marcadores para cada punto único en la serie
for i, row in df_unique.iterrows():
    folium.Marker(location=[row['latitude'],
                            row['longitude']],
                  popup=row['id']).add_to(mapa)

# Muestra el mapa
mapa
```

Así mismo, con el fin de poder eliminar aquellas estaciones que ya no se encuentran en servicio, es necesario crear una máscara que más tarde permitirá por identificador eliminarlos del dataset completo.

```
mask = df_nom["status"] == "NOT_IN_SERVICE"
bad_id = df_nom[mask]["id"].to_list()
```

Siguiendo con los datos, una buena forma de visualizar el crecimiento (si existe) es explorando la acomulación de eventos a lo largo del tiempo, para ello, concatenamos las columnas de inicio y final del viaje y después graficamos.

```
df["Inicio_del_viaje"] = pd.to_datetime(df["Inicio_del_viaje"])
df["Fin_del_viaje"] = pd.to_datetime(df["Fin_del_viaje"])
serie_tiempo = pd.concat([df['Inicio_del_viaje'],
                          df['Fin_del_viaje']])
serie_tiempo = serie_tiempo.sort_values()
```

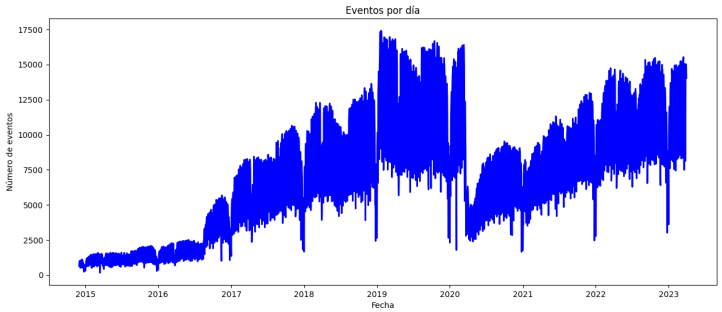


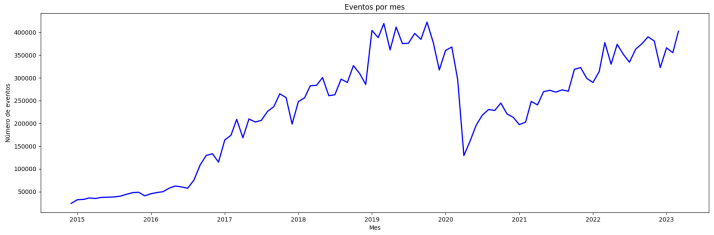
en la imagen anterior podemos observar como el crecimiento exponencial comienza desde el año 2014 para luego tener una decadencia en el año 2020, año del COVID, para finalmente seguir con dicho crecimiento de una forma un poco menos acelerada.

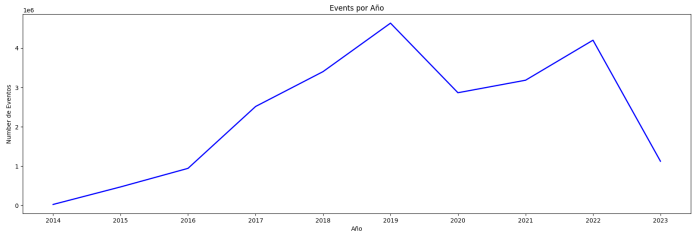
Así mismo, podemos observar el comportamiento por cada uno de los días, meses y años, para ello utilizamos el código siguiente, obteniendo tres visualizaciones.

```
df_events_per_day =
    df.groupby(df['Inicio_del_viaje'].dt.date)
     ['Inicio_del_viaje'].count()
     .reset_index(name='eventos_por_dia')

df_events_per_day =
    df.groupby(df['Inicio_del_viaje'].dt.to_period('M'))
     ['Inicio_del_viaje'].count()
     .reset_index(name='eventos_por_mes')
```

```
df_events_per_day =
    df.groupby(df['Inicio_del_viaje'].dt.to_period('Y'))
     ['Inicio_del_viaje'].count()
     .reset_index(name='eventos_por_año')
```







Como se puede observar en las tres imágenes, la tendencia muestra un crecimientos significativo que decrece abruptamente en tiempos de pandemia y luego continúa creciendo. Si bien la reducción de datos es considerable debido al conteo por días, se puede entrenar una red neural. En este caso, se optará por una LSTM.

3.2. Entrenamiento de una LSTM

Las redes neuronales LSTM (Long Short-Term Memory) son especialmente adecuadas para predecir series de tiempo debido a su capacidad para aprender y recordar dependencias a largo plazo en secuencias de datos. A diferencia de las redes neuronales recurrentes (RNN) tradicionales, las LSTM pueden capturar patrones temporales en series de tiempo y mantener información relevante a lo largo de secuencias largas sin verse afectadas significativamente por el problema del desvanecimiento del gradiente.

Para el caso de esta red, se optó por el uso de la paquetería de torch, creando así una serie de configuraciones con las que se evaluó el RMSE como función de pérdida. Debido a la cantidad de datos, se optó por la creación de una red pequeña con una cantidad de épocas considerables. Algunas de las configuraciones se pueden observar en la siguiente tabla:

Activador	Neuronas por capa	Capaz ocultas	Epocas	RMSE
lineal	100	2	200	12.1820
lineal	100	3	200	14.3430
lineal	150	2	300	13.5898
lineal	150	3	300	14.0987
lineal	200	2	200	13.3987
lineal	200	3	200	15.9870

como se puede observar en la tabla anterior, incluso con la menor configuración se obtiene un resultado menor de un mínimo local, incluso con menos épocas, razón por la cual consideramos esa relación como la final.

```
timeseries =
    df_events_per_day['events_per_day'].values.astype('float32'
timeseries = np.sqrt(timeseries)

# train-test split for time series
train_size = int(len(timeseries) * 0.67)
test_size = len(timeseries) - train_size
train, test = timeseries[:train_size], timeseries[train_size:]

import torch

def create_dataset(dataset, lookback):
    X, y = [], []
    for i in range(len(dataset)-lookback):
```

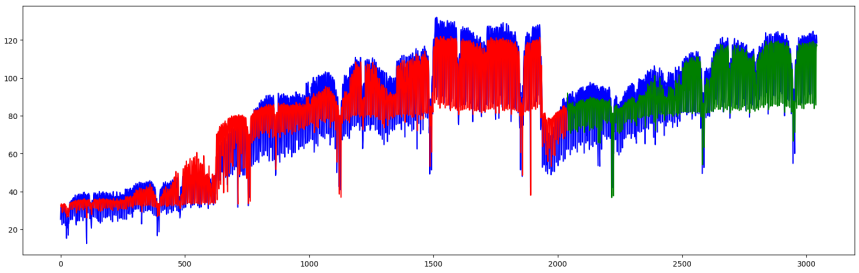
```
        feature = dataset[i:i+lookback]
        target = dataset[i+1:i+lookback+1]
        X.append(feature)
        y.append(target)
    return torch.tensor(X), torch.tensor(y)

lookback = 1
X_train, y_train = create_dataset(train, lookback=lookback)
X_test, y_test = create_dataset(test, lookback=lookback)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

#torch.Size([2037, 1]) torch.Size([2037, 1])
#torch.Size([1004, 1]) torch.Size([1004, 1])

import torch.nn as nn

class AirModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=1, hidden_size=100,
                            num_layers=2, batch_first=True)
        self.linear = nn.Linear(100, 2)
    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x

import numpy as np
import torch.optim as optim
import torch.utils.data as data

model = AirModel()
optimizer = optim.Adam(model.parameters())
loss_fn = nn.MSELoss()
loader = data.DataLoader(data.TensorDataset(X_train, y_train),
                         shuffle=True, batch_size=8)

n_epochs = 200
```

```
for epoch in range(n_epochs):
    model.train()
    for X_batch, y_batch in loader:
        y_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    # Validation
    if epoch % 10 != 0:
        continue
    model.eval()
    with torch.no_grad():
        y_pred = model(X_train)
        train_rmse = np.sqrt(loss_fn(y_pred, y_train))
        y_pred = model(X_test)
        test_rmse = np.sqrt(loss_fn(y_pred, y_test))
    print("Epoch %d: train RMSE %.4f, test RMSE %.4f"
          % (epoch, train_rmse, test_rmse))

with torch.no_grad():
    # shift train predictions for plotting
    train_plot = np.ones_like(timeseries) * np.nan
    y_pred = model(X_train)
    y_pred = y_pred[:, -1]
    train_plot[lookback:train_size] = model(X_train)[:, -1]
    # shift test predictions for plotting
    test_plot = np.ones_like(timeseries) * np.nan
    test_plot[train_size+lookback:len(timeseries)] =
        model(X_test)[:, -1]
```

Finalmente, para el entranamiento se optó por una configuración del 67 %, esto debido a los valores fuera de rango ocasionados por la pandemia, es de destacarse que dicha configuración fue elegida a prueba y error, empezando con la configuración estandar del 80 % y decreciendo hasta obtener una apreciación mejor de la predicción. El punto principal de realizar esto es entrenar al modelo con los outliers del modelo y que esto sirva para el crecimiento posterior a estos datos anómalos.

Para corroborar esto último, en la siguiente figura se muestra la serie de

tiempo de los eventos por día (azul) y la evaluación del modelo ocn el dataset de entrenamiento (rojo) y con el dataset de prueba (verde).



4. Conclusiones

Con base en los resultados podemos concluir que:

1. El modelo fue exitosamente entrenado aún con datos anómalos del COVID-19 en el periodo de tiempo 2020-2021.

2. El modelo entrenado es capaz de precedir incluso la estacionalidad de eventos, que como se puede observar en la última figura de la sección anterior, esta se repite anualmente al finalizar/iniciar cada año.

3. Con el modelo entrenado se puede predecir el crecimiento a corto y largo plazo del uso de las bicicletas en este sistema público de transportes. Así mismo, con el comportamiento de los datos se puede inferir que este seguirá creciendo en la misma proporción mostrada hasta el momento.

5. Pasos a futuro

Si bien este es un primer acercamiento a lo que es posiblemente un análisis más a fondo, existe una serie de pasos que pueden ser seguidos para futuras iteraciones:

1. Con base en los identificadores de las estaciones de las bicicletas, generar un análisis de aquellas con más visitas y usar tanto este modelo como uno particular por zona para determinar la cantidad de bicis a futuro.

2. Crear visualizaciones de calor con base en el análisis del punto anterior.

6. Referencias

1. https://www.jalisco.gob.mx/es/jalisco/guadalajara

2. Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time series analysis: forecasting and control. John Wiley & Sons.

3. Brockwell, P. J., & Davis, R. A. (2016). Introduction to time series and forecasting. Springer.

4. Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice. OTexts.

5. Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional Time Series Forecasting with Convolutional Neural Networks. arXiv preprint arXiv:1703.04691. https://arxiv.org/abs/1703.04691

6. Lai, G., Chang, W. C., Yang, Y., & Liu, H. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (pp. 95-104). https://dl.acm.org/doi/10.1145/3209978.3210006

7. Wang, Y., Wang, D., Wu, Q., Shi, J., & Ye, J. (2020). Deep Factors for Forecasting. In International Conference on Machine Learning (pp. 10035-10044). PMLR. http://proceedings.mlr.press/v119/wang20i.html