

Despliegue de solución analítica para tracking bursátil TB500

Reporte de implementación y experimentos

Misael García, Carlos Cañas, David Castrillón y Karina Duran

Grupo 1

Contenido

1	Introducción y generalidades del prototipo.....	2
2	Rutinas para alimentar los con datos de entrenamiento y de prueba	2
3	Modelos de pronóstico	5
3.1	Modelo LSTM	5
3.1.1	Tratamiento previo de los datos	5
3.1.2	Proceso de entrenamiento, ajuste y pruebas	6
3.2	Modelo XGBoost	9
3.2.1	Tratamiento previo de los datos	9
3.2.2	Proceso de entrenamiento.....	9
3.3	Modelo RandomForest	12
3.3.1	Tratamiento previo de los datos	12
3.3.2	Proceso de entrenamiento.....	12
3.4	Modelo de optimización	15
4	Comparación de desempeño de modelos y potenciales ajustes por implementar	20
4.1	Desempeño del modelo LSTM	22
4.2	Desempeño del modelo XGBoost	23
5	Análisis de resultados y/o ajustes necesarios a los modelos, al esquema general de solución, o a los problemas de negocio o de analítica de datos	23
6	Avance y plan de implementación del prototipo	24
6.1	Avance del prototipo.....	24
6.2	Plan para entrega del prototipo.....	27

1 Introducción y generalidades del prototipo

El prototipo en desarrollo utiliza como fuente de datos Yahoo Finance API y FRED API con el fin de entregar al Grupo Stanley un portafolio con sus acciones ya definidas, pesos definidos y niveles de riesgo o retornos específicos para una fecha a corto plazo no mayor a 3 días. Si estos valores no son especificados, el prototipo entregará un portafolio que minimice el riesgo de las acciones manejadas por el grupo.

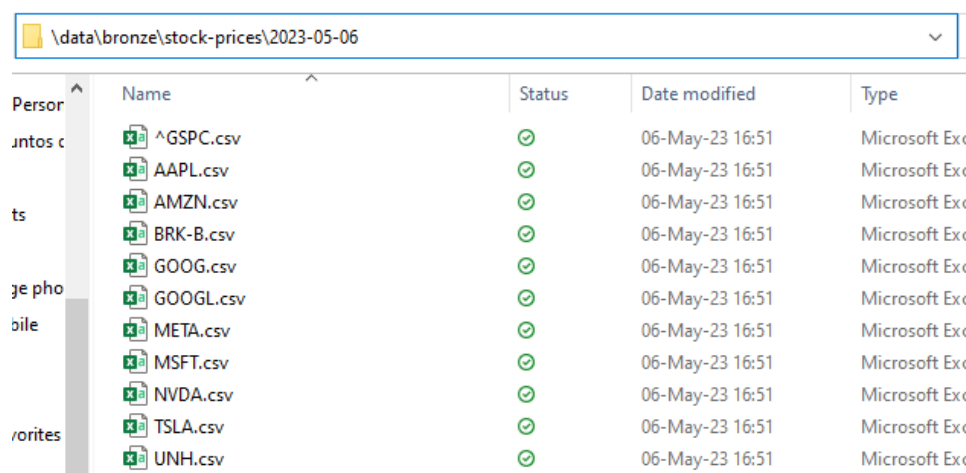
2 Rutinas para alimentar los con datos de entrenamiento y de prueba

El primer aspecto para considerar al implementar una solución analítica es identificar los datos fuente, definir de qué manera serán almacenados y habilitados para consumo del modelo o solución que se desea construir. Particularmente para el prototipo, las fuentes de datos principales que usaremos son Yahoo! Finance API y FRED API, las cuales serán consumidas y cargadas en dos momentos hacia el lago de datos y sus zonas propuestas: inicialmente se hará una primera carga como parte del proceso de carga histórica de datos (alrededor de 3 años de datos), posteriormente la data continuará actualizándose día a día a través del proceso de carga incremental construido para tal fin.

Para almacenar la data se propone hacerlo a través de tablas de tipo delta (Delta Lake), las cuales permiten aprovechar ampliamente las bondades de un lago de datos, manteniendo la integridad de los datos por medio de operaciones de tipo ACID; como parte del prototipo dichas tablas son emuladas como archivos de texto plano.

A continuación, se describen las zonas (o capas) del lago de propuestas para organizar la data, así como las tablas que almacenarán los datos en cada una de las capas:

- **Bronze:** Persistencia de los datos tal cual provienen de las APIs, organizada por fecha de ingestión



Name	Status	Date modified	Type
^GSPC.csv	✓	06-May-23 16:51	Microsoft Exc
AAPL.csv	✓	06-May-23 16:51	Microsoft Exc
AMZN.csv	✓	06-May-23 16:51	Microsoft Exc
BRK-B.csv	✓	06-May-23 16:51	Microsoft Exc
GOOG.csv	✓	06-May-23 16:51	Microsoft Exc
GOOGL.csv	✓	06-May-23 16:51	Microsoft Exc
META.csv	✓	06-May-23 16:51	Microsoft Exc
MSFT.csv	✓	06-May-23 16:51	Microsoft Exc
NVDA.csv	✓	06-May-23 16:51	Microsoft Exc
TSLA.csv	✓	06-May-23 16:51	Microsoft Exc
UNH.csv	✓	06-May-23 16:51	Microsoft Exc

- **Silver:** Limpieza, homologación de formatos entre fuentes y preprocesamiento: tratamiento de data faltante haciendo imputación estadística, eliminación de duplicados, homologación de formatos (fechas, números, etc.), identificación/tratamiento de outliers

\data\silver\stock-prices				
Name	Status	Date modified	Type	
stock_prices.csv	✓	07-May-23 17:01	Microsoft Ex	

- **Gold:** Data preparada y estructurada para alimentar al modelo en cuestión

\data\gold\portfolio-optimization				
Name	Status	Date modified	Type	
portfolio_optimization.csv	✓	07-May-23 18:37	Microsoft Ex	

A continuación, se muestra un adelanto de la estructura del ETL descrito anteriormente:

0. Imports and params

```
In [1]: import numpy as np
import pandas as pd
import quandl
import yfinance as yf
import os
import matplotlib.pyplot as plt

In [2]: hist_years = 3
bronze = './data/bronze/stock-prices/'
silver = './data/silver/stock-prices/'
gold = './data/gold/portfolio-optimization/'
ticker = ['AAPL', 'MSFT', 'AMZN', 'TSLA', 'GOOGL', 'GOOG', 'NVDA', 'BRK-B', 'META', 'UNH', '^GSPC']
start = (pd.to_datetime('today').normalize()+pd.DateOffset(years=-hist_years)).strftime('%Y-%m-%d')
end = pd.to_datetime('today').normalize().strftime('%Y-%m-%d')

In [3]: if not os.path.exists(bronze+end+'/'):
os.mkdir(bronze+end+'/')
if not os.path.exists(silver):
os.mkdir(silver)
if not os.path.exists(gold):
os.mkdir(gold)
```

1. Bronze Layer

```
In [4]: # Data extraction
for i in ticker:
    brz_data = yf.download(i, start=start, end=end)
    brz_data['Ticker']=i
    brz_data.to_csv(bronze+end+'/'+'i+'.csv')

[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
```

2. Silver Layer

```
In [5]: slv_data = pd.DataFrame()
for i in ticker:
    df = pd.read_csv(bronze+end+'/'+'i+'.csv')
    slv_data = pd.concat([slv_data, df])
slv_data.to_csv(silver+'stock_prices.csv', index=False)
```

Es importante aclarar que lo explicado anteriormente es la estructuración a la que esperamos llegar como parte del despliegue del prototipo, la cual logramos adelantar en esta etapa, sin embargo, para completar la ejecución de experimentos y pruebas sobre los modelos a evaluar, por practicidad, una parte de los datos fue descargada por medio de rutinas puntuales, que se ajustarán e integrarán posteriormente al proceso de ETL. Dichas rutinas se describen a continuación:

Ingesta de datos para la ejecución de pruebas

La función `datos` permite introducir dos listas, una para la lista de acciones que provienen de *Yahoo Finance* y la otra para incluir las variables macroeconómicas de Fred que el usuario requiera incluir o que por defecto se requieran para calcular el portafolio:

```
def datos(acc=["AMZN"],mac=["SP500"],start = '2022-01-01', end = '2023-03-31'):

    # variables macroeconomicas
    fred = fa.Fred(api_key='00dd681538883996c428f1dbb62e88a7')
    dfmac = pd.DataFrame()
    # listado variables diarias https://fred.stlouisfed.org/tags/series?t=daily
    for var in mac:
        dfmac[var] = fred.get_series(var, observation_start=start)
    dfmac.index = pd.to_datetime(dfmac.index.date).date

    # acciones
    df = pd.DataFrame()
    for ticker in acc:
        df[ticker] = yf.Ticker(ticker).history(start = start,end = end).Close
    df.index = pd.to_datetime(df.index.date).date

    # unir variables
    data = df.merge(dfmac, right_index=True, left_index=True)
    data.dropna(inplace=True)

    # seleccionar variables de interes
    return data
```

Para mostrar cómo funciona suponga que se quieren traer las 11 acciones más importantes del S&P500 desde enero de 2022 y las variables macroeconómicas de interés:

```
# identificación día actual
today = datetime.date.today().strftime('%Y-%m-%d')

# Acciones de interes
stk_yhf = ['AAPL','MSFT','AMZN','TSLA','GOOGL','GOOG','NVDA','BRK-B','META','UNH','^GSPC']

# Variables macroeconomicas
vme_fred = ['T10Y3M','DFF','DGS10','T5YIE','DFII10','SP500','DFEDTARU','DCOILWTICO','EFFR','DPRIME']

# ejecución de función datos
data = datos(stk_yhf,vme_fred,'2022-01-01',today)

print(data.shape)
data.head()
```

La ejecución de la función con los parámetros mencionados anteriormente genera un dataframe de 21 columnas (11 acciones y 10 variables macro) y 310 filas (periodo de días a evaluar)

(310, 21)

	AAPL float64	MSFT float64	AMZN float64	TSLA float64	GOOGL float64	GOOG float64	NVDA float64	BRK-B float64	Visualize
2022-01-03	180.68386840820312	330.8138732910156	170.4044952392578	399.9266662597656	144.9915008544922	145.07449340820312	300.8774719238281	300.7900085449219	
2022-01-04	178.39068603515625	325.1413879394531	167.52200317382812	383.1966552734375	144.39950561523438	144.41650390625	292.57666015625	308.5299987792969	
2022-01-05	173.64553833007812	312.65985107421875	164.35699462890625	362.7066650390625	137.77499389648438	137.65350341796875	275.7352600097656	309.9200134277344	

Hallazgos:

- La función es vulnerable al orden en que se solicitan las variables, es necesario ingresar los parámetros de la función con nombre o asegurar que el orden de los parámetros es [acciones, variables_macro, inicio, fin]
- Si bien existen variedad de variables macroeconómicas, para el entrenamiento de modelos en series de tiempo es necesario contar con valores en la misma periodicidad de las

acciones, por lo cual se debe limitar el uso de variables macroeconómicas a las que cuenten con información diaria.

- Si no se ingresan los parámetros de la función, se generan por defecto valores de 'AMZN' 'SP500' entre 2022-01-01 y 2023-03-01

Split de conjuntos para entrenamiento

La función `split` nos permite separar el conjunto de datos importados en la función ***datos*** en los diferentes conjuntos de datos de entrenamiento y prueba para las fechas, acciones y variables macroeconómicas. Teniendo en cuenta la restricción de variables macro económicas a utilizar y que las variables macroeconómicas están ubicadas al final del conjunto de datos se realiza un split entre variables según la existencia de las variables definidas, dentro del conjunto de datos.

```
def split(data, per=0.8):
    # creacion de dataframe de fechas provenientes del indice
    dataindex= pd.to_datetime(data.index)

    # identificación de elementos dentro del porcentaje elegido
    split = int(per*len(dataindex))

    # split de fechas
    date_train = dataindex[:split]
    date_test = dataindex[split:]

    # split de datos
    data_train = data.iloc[:split]
    data_test = data.iloc[split:]

    # Identificación de valores macro economicas
    c = [x in ['T10Y3M','DFF','DGS10','T5YIE','DFII10','SP500','DFEDTARU','DCOILWTICO','EFFR','DPRIME'] \
         for x in list(data.columns)].index(True)

    # split acciones
    y_train = data_train.iloc[:,c]
    y_test = data_test.iloc[:,c]

    # split exogenas
    ex_train = data_train.iloc[:,c:]
    ex_test = data_test.iloc[:,c:]

    return [date_train,date_test,y_train,y_test,ex_train,ex_test]
```

Para mostrar cómo funciona suponga que se quiere entrenar los modelos con el 80% del conjunto de datos, la función genera un índice de fechas, la separación de variables de acciones y de variables macroeconómicas.

```
# split de información para modelos
date_train, date_test, y_train, y_test, ex_train, ex_test = split(data, 0.8)

#Imprimimos la longitud de ambos conjuntos de datos:
print(len(y_train))
print(len(y_test))
```

3 Modelos de pronóstico

3.1 Modelo LSTM

3.1.1 Tratamiento previo de los datos

Para entrenar el modelo en cuestión, se parte de la premisa de que los datos insumo ya fueron cargados al lago de datos en capa Silver, es decir, se encuentran en un estado depurado, unificados y persistidos en un solo lugar, en este caso en una tabla de tipo de delta:

```
# Read data from data Lake
data = pd.read_csv(silver+silver_table)
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data.head()
```

	Open	High	Low	Close	Adj Close	Volume	Ticker
Date							
2020-05-07	75.805000	76.292503	75.492500	75.934998	74.454674	115215200	AAPL
2020-05-08	76.410004	77.587502	76.072502	77.532501	76.226830	133838400	AAPL
2020-05-11	77.025002	79.262497	76.809998	78.752502	77.426292	145946400	AAPL
2020-05-12	79.457497	79.922501	77.727501	77.852501	76.541443	162301200	AAPL
2020-05-13	78.037498	78.987503	75.802498	76.912498	75.617271	200622400	AAPL

Posteriormente, del dataset anterior se toma la métrica "Adj Close" generando un dataset nuevo para el ticker (o acción) que se desea procesar. El nuevo dataset se normaliza utilizando MinMaxScaler, escalando los datos entre 0 y 1 con el fin de preparar los datos antes de entrenar la red neuronal.

```
# Load the data
input_data = pd.DataFrame(data[data['Ticker']==ticker][metric_to_predict])

# Normalize data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(input_data[metric_to_predict].values.reshape(-1,1))
```

3.1.2 Proceso de entrenamiento, ajuste y pruebas

Variables

Para alistar las variables para el entrenamiento del LSTM iterando a través de los datos normalizados *scaled_data*, comenzando desde el índice *prediction_days*. Para cada iteración, se agrega una secuencia de longitud *pred_span_days* de valores previos a *x_train*, y se agrega el valor actual a *y_train*. Después, se transforman las listas *x_train*, *y_train* en arreglos y se reformatea *x_train* para que tenga las dimensiones apropiadas para el modelo LSTM.

```
# Initialize empty lists for training data input and output
x_train = []
y_train = []

# Iterate through the scaled data, starting from the pred_span_days index
for x in range(pred_span_days, len(scaled_data)):
    # Append the previous 'pred_span_days' values to x_train
    x_train.append(scaled_data[x - pred_span_days:x, 0])
    # Append the current value to y_train
    y_train.append(scaled_data[x, 0])

# Convert the x_train and y_train lists to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape x_train to a 3D array with the appropriate dimensions for the LSTM model
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

Para generar la muestra de test, se hace uso de la función *get_data_tail()*, la cual depende de dos parámetros: "input_data", que es el conjunto de datos históricos disponibles (del paso previo) de donde se tomará la porción de interés, partiendo de la última observación hacia atrás, y "backward_steps", que es el número de pasos de tiempo (o días) a recorrer hacia atrás para

delimitar la data a extraer. En esencia esta función toma los últimos datos disponibles en el dataset de entrada con el fin de probar y predecir usando la información más fresca.

```
# Extract the relevant portion of the dataset for model inputs
input_data_pred = get_data_tail(input_data, backward_steps, scaler)

# Initialize an empty List for test data input
x_test = []

# Iterate through the model inputs, starting from the pred_span_days index
for x in range(pred_span_days, len(input_data_pred)):
    # Append the previous 'pred_span_days' values to x_test
    x_test.append(input_data_pred[x-pred_span_days:x, 0])

# Convert the x_test List to a numpy array
x_test = np.array(x_test)

# Reshape x_test to a 3D array with the appropriate dimensions for the LSTM model
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

# Data to validate the predictions made
y_test = input_data.iloc[-backward_steps:,:].values
```

Parametrización

Trabajamos con dos topologías de red LSTM, una de 4 y otra de 5 capas. En general, en ambas topologías tenemos capas con `return_sequences=True` para devolver secuencias completas, salvo la penúltima, y la capa final o de la salida siendo de tipo Dense con una sola unidad conteniendo el dato predicho. Entre las capas LSTM, se agrega una capa Dropout para prevenir el sobreajuste y mejorar el rendimiento del modelo. Para entrenar y ajustar el modelo se hace uso de `RandomizedSearchCV()` para encontrar los mejores parámetros posibles. Luego se procede a generar predicciones para probar el desempeño del modelo.

```
# Define the hyperparameters and their ranges for random search
param_grid = {
    'units': units,
    'dropout_rate': dropout_rate,
    'learning_rate': learning_rate,
    'epochs': epochs,
    'batch_size': batch_size
}

# Define Keras Regressor
model = KerasRegressor(build_fn=create_model, verbose=0)

# Perform random search
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=5,
    cv=3,
    random_state=42
)
random_search.fit(x_train, y_train)

# Print the best hyperparameters and their corresponding score
print('Best Parameters:', random_search.best_params_)
print('Best Score:', random_search.best_score_)

# Generate price predictions using the LSTM model
y_pred = np.reshape(random_search.predict(x_test), (-1, 1))

# Invert the scaling applied to the predicted prices to obtain actual values
y_pred = scaler.inverse_transform(y_pred)
```

Se colocan algunos ejemplos de las salidas obtenidas:

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
lstm (LSTM)	(None, 60, 50)	10400
dropout (Dropout)	(None, 60, 50)	0
lstm_1 (LSTM)	(None, 60, 50)	20200
dropout_1 (Dropout)	(None, 60, 50)	0
lstm_2 (LSTM)	(None, 50)	20200
dropout_2 (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51
=====		
Total params: 50,851		
Trainable params: 50,851		
Non-trainable params: 0		

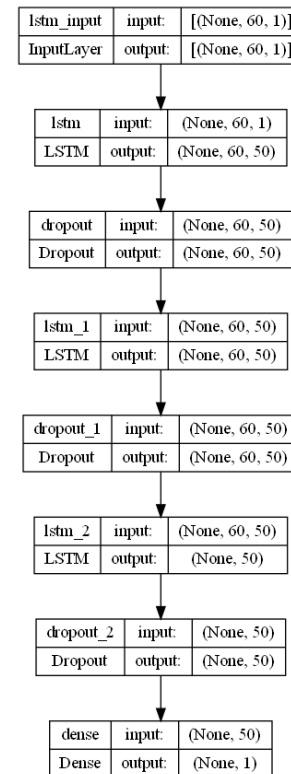


Imagen 1. Arquitectura de la red LSTM.

Definición de métricas adecuadas de desempeño

Para medir el desempeño del modelo usamos las siguientes métricas:

```

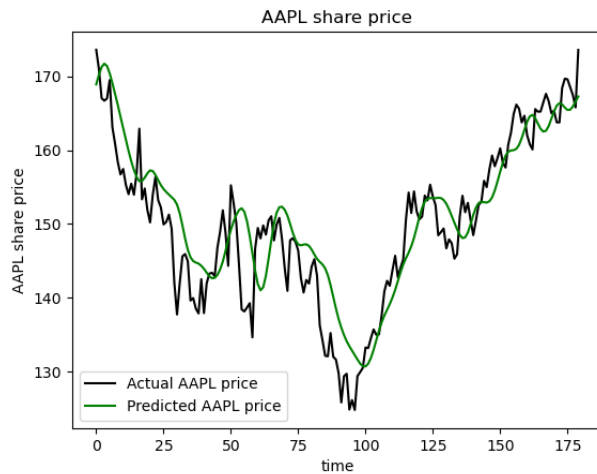
# Calculate performance metrics
mse = np.mean(np.power(y_test - y_pred, 2))
rmse = np.sqrt(np.mean(np.power(y_test - y_pred, 2)))
mae = np.mean(np.abs(y_test - y_pred))
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

```

Cálculo de métricas de desempeño

Como parte de las pruebas y medición de desempeño, usamos gráficos lineales para comprar la data real y la predicha en la ventana de tiempo establecida.

Best Parameters: {'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}
Best Score: -0.00350428675301373



Mean Squared Error: 25.9318
Root Mean Squared Error: 5.0923
Mean Absolute Error: 4.0945
Mean Absolute Percentage Error: 2.7991%

Imagen 2. Comparación precio real versus predicho.

3.2 Modelo XGBoost

3.2.1 Tratamiento previo de los datos

Para utilizar el modelo XGBoost se toman las acciones de interés de la compañía y las variables macroeconómicas. Se definen funciones para el tratamiento de los datos teniendo énfasis en el manejo de las fechas, pues para este modelo es requerido contar con fechas continuas.

```
df.index = pd.to_datetime(df.index.date).date  
df = df.asfreq('D')  
df = df.sort_index()
```

```
# split acciones  
y_train = data_train.iloc[:, :c].asfreq('D')  
y_test = data_test.iloc[:, :c].asfreq('D')  
  
# split exogenas  
ex_train = data_train.iloc[:, c:].asfreq('D')  
ex_test = data_test.iloc[:, c:].asfreq('D')
```

3.2.2 Proceso de entrenamiento

Variables

Para el entrenamiento se disponen de los datos de las acciones de interés y de las variables macroeconómicas en conjuntos distintos, con los índices tipo fecha estandarizados.

```
# split de información para modelos
date_train, date_test, y_train, y_test, ex_train, ex_test = split(data, 0.8)

#Imprimimos la longitud de ambos conjuntos de datos:
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
print("ex_train: ",ex_train.shape)
print("ex_test: ",ex_test.shape)
```



```
y_train: (264, 11)
y_test: (67, 11)
ex_train: (264, 10)
ex_test: (67, 10)
```

Para el modelo se entrenan las 11 variables de acciones y 3 variables exógenas de la información macroeconómica

Parametrización

Para la parametrización del modelo XGBoost se selecciona la librería SKForecast su función de ForecasterAutoregMultiSeries, el cual permite el ajustar el set de datos en diferentes lags.

```
forecaster_ms = ForecasterAutoregMultiSeries(
    regressor      = XGBRegressor(random_state=123),
    lags           = 14,
    #transformer_series = StandardScaler(),
)
```

El ajuste de parámetros se realiza mediante la función de grid_search_forecaster_multiseries en donde se iteran distintos valores de lags, iteraciones máximas, profundidad máxima y tasa de aprendizaje.

```
lags_grid = [7, 14, 21, 28]
param_grid = {
    'max_iter': [100, 500, None],
    'max_depth': [3, 10, None],
    'learning_rate': [0.01, 0.1, None]
}

results_grid_ms = grid_search_forecaster_multiseries(
    forecaster      = forecaster_ms,
    series          = y_train,
    exog            = ex_train[['T10Y3M', 'SP500', 'DCOILWTICO']],
    levels          = None, # If None all levels are selected
    lags_grid       = lags_grid,
    param_grid      = param_grid,
    steps           = 7,
    metric           = 'mean_absolute_error',
    initial_train_size = int(len(y_train)*0.5),
    refit            = False,
    fixed_train_size = False,
    return_best      = True,
    verbose         = False
)
```

Para el ajuste del modelo se utiliza la función backtesting_forecaster_multiseries que utiliza la información del set de entrenamiento para reducir el error según el error promedio absoluto.

```

multi_series_mae, predictions_ms = backtesting_forecaster_multiseries(
    forecaster      = forecaster_ms,
    series          = y_train,
    levels          = None, # If None all levels are selected
    steps           = 7,
    exog            = ex_train[['T10Y3M', 'SP500', 'DCOILWTICO']],
    metric          = 'mean_absolute_error',
    initial_train_size = int(len(y_train)*0.5),
    refit           = False,
    fixed_train_size = False,
    verbose         = False
)

```

Definición de métricas adecuadas de desempeño

Se define MSE como métrica de desempeño ya que este permite tener una idea de la magnitud del error en términos de la escala de los datos, así mismo permite comparar diferentes modelos entre sí.

Cálculo de métricas de desempeño

Resultado del grid search se obtienen los diferentes errores medios de los distintos modelos entrenados entre la iteración.

results_grid_ms							
	levels object	lags object	params object	mean_absolute...	learning_rate flo...	max_depth float...	max_iter float64
	[1 2 3 4 5 6 ... 25% [1 2 3 4 5 6 7] ... 25% 2 others 50%						
	['AAPL', 'MS...', 100%						
37	['AAPL', 'MSFT', 'AMZN', 'TSLA', ...]	[1 2 3 4 5 6 7 8 9 10 11 12 13 14]	{'learning_rate': 0.1, 'max_depth': 3, ...}	10.3731991949160... 24	0.1	3.0	500.0
38	['AAPL', 'MSFT', 'AMZN', 'TSLA', ...]	[1 2 3 4 5 6 7 8 9 10 11 12 13 14]	{'learning_rate': 0.1, 'max_depth': 3, ...}	10.3731991949160... 24	0.1	3.0	nan
36	['AAPL', 'MSFT', 'AMZN', 'TSLA', ...]	[1 2 3 4 5 6 7 8 9 10 11 12 13 14]	{'learning_rate': 0.1, 'max_depth': 3, ...}	10.3731991949160... 24	0.1	3.0	100.0
11	['AAPL', 'MSFT', 'AMZN', 'TSLA', ...]	[1 2 3 4 5 6 7]	{'learning_rate': 0.1, 'max_depth': 3, ...}	10.4023714722352... 26	0.1	3.0	nan
9	['AAPL', 'MSFT', 'AMZN', 'TSLA', ...]	[1 2 3 4 5 6 7]	{'learning_rate': 0.1, 'max_depth': 3, ...}	10.4023714722352... 26	0.1	3.0	100.0



Imagen 3. Precio predicho vs precio real.

Si bien el modelo presenta un valor de predicción cercano a los valores de prueba en los primeros lags, este valor tiende a alejarse a medida que los lags se alejan. Pues el modelo da prioridad a los valores mas recientes de la serie y en este caso se pronostica sobre una tendencia creciente que el modelo trata de simular según ciclos anteriores.

Lo anterior da prioridad a presentar proyecciones cercanas a la última fecha disponible de modo de no sobre estimar tendencias pasadas

3.3 Modelo RandomForest

3.3.1 Tratamiento previo de los datos

Para utilizar el modelo RandomForest se toman las acciones de interés de la compañía. Se definen funciones para el tratamiento de los datos teniendo énfasis en el manejo de las fechas, pues para este modelo es requerido contar con fechas continuas.

```
# Read data from data lake
data = pd.read_csv(silver+silver_table)
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

df_c = pd.pivot_table(data, values=metric_to_predict, columns=["Ticker"], index=data.index)
df_c = df_c[ticker]
```

3.3.2 Proceso de entrenamiento

Variables

Para el entrenamiento se disponen de los datos de las acciones de interés, con los índices tipo fecha estandarizados.

```
# Extract the relevant portion of the dataset for model inputs
df_train = df_c.iloc[-backward_steps:]
df_test = df_c.iloc[backward_steps:]
```

Para el modelo se entrenan las 10 variables de acciones

Parametrización

Para la parametrización del modelo RandomForest se selecciona la librería SKForecast su función de ForecasterAutoregMultiSeries, el cual permite el ajustar el set de datos en diferentes lags.

```
forecaster_ms = ForecasterAutoregMultiSeries(
    regressor      = RandomForestRegressor(random_state=123),
    lags           = 14,
)
```

El ajuste de parámetros se realiza mediante la función de grid_search_forecaster_multiseries en donde se iteran distintos valores de lags, iteraciones máximas, profundidad máxima y tasa de aprendizaje.

```
param_grid_RF = {
    'bootstrap': [True],
    'max_depth': [5, 10, 30],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [10, 20, 50]
}

results_grid_ms = grid_search_forecaster_multiseries(
    forecaster      = forecaster_ms,
    series          = df_train,
    levels          = None, # If None all levels are selected
    lags_grid       = lags_grid,
    param_grid      = param_grid_RF,
    steps           = 7,
    metric          = ['mean_squared_error', 'mean_absolute_error', 'mean_absolute_percentage_error'],
    initial_train_size = len(df_train)-30,
    refit           = False,
    fixed_train_size = False,
    return_best     = True,
    verbose         = False
)
```

Para el ajuste del modelo se utiliza la función backtesting_forecaster_multiseries que utiliza la información del set de entrenamiento para reducir el error según el error promedio absoluto.

```

multi_series_mae, predictions_ms = backtesting_forecaster_multiseries(
    forecaster      = forecaster_ms,
    series          = df_train,
    levels          = None, # If None all levels are selected
    steps           = 7,
    metric          = 'mean_squared_error',
    initial_train_size = len(df_train)-30,
    refit           = False,
    fixed_train_size = False,
    verbose         = False
)

```

Definición de métricas adecuadas de desempeño

Se define MSE como métrica de desempeño ya que este permite tener una idea de la magnitud del error en términos de la escala de los datos, así mismo permite comparar diferentes modelos entre sí.

Cálculo de métricas de desempeño

Resultado del grid search se obtienen los diferentes errores medios de los distintos modelos entrenados entre la iteración.

	levels	lags	params	mean_absolute_error	bootstrap	max_depth	min_samples_leaf	min_samples_split	n_estimators
69	[AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG, NVDA, BR...]	[1, 2, 3, 4, 5, 6, 7]	('bootstrap': True, 'max_depth': 30, 'min_samp...	6.697422	True	30	2	10	10
50	[AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG, NVDA, BR...]	[1, 2, 3, 4, 5, 6, 7]	('bootstrap': True, 'max_depth': 10, 'min_samp...	6.698381	True	10	4	5	50
47	[AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG, NVDA, BR...]	[1, 2, 3, 4, 5, 6, 7]	('bootstrap': True, 'max_depth': 10, 'min_samp...	6.698381	True	10	4	2	50
58	[AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG, NVDA, BR...]	[1, 2, 3, 4, 5, 6, 7]	('bootstrap': True, 'max_depth': 30, 'min_samp...	6.710453	True	30	1	5	20
74	[AAPL, MSFT, AMZN, TSLA, GOOGL, GOOG, NVDA, BR...]	[1, 2, 3, 4, 5, 6, 7]	('bootstrap': True, 'max_depth': 30, 'min_samp...	6.727368	True	30	4	2	50

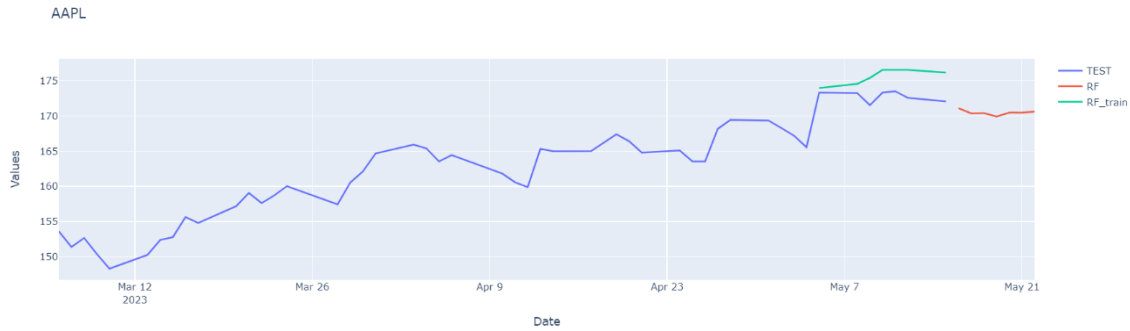


Imagen 4. Precio predicho vs precio real.

Comparación de modelos

Con los modelos anteriormente presentados se realizan distintas pruebas con la configuración de sus parametros para encontrar la mejor configuración de los mismos para su uso en la aplicación.

A continuación, se presenta una tabla resumen de las distintas configuraciones probadas.

Modelo	Configuración	MSE
LSTM Encoder-Decoder	{'units': 64, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	77.23
LSTM Encoder-Decoder-CNN	{'units': 128, 'epochs': 25, 'dropout_rate': 0.2, 'batch_size': 32}	25.10
LSTM	{'units': 128, 'epochs': 25, 'dropout_rate': 0.1, 'batch_size': 32}	508.97
RandomForest	{'lags': 7, 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 10}	102.51
RandomForest	{'lags': 7, 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 50}	103.01
XGBoost	{'learning_rate': 0.1, 'max_depth': 3, 'max_iter': 100}	93.87
XGBoost	{'learning_rate': 0.1, 'max_depth': 5, 'max_iter': 500}	94.24

Ensamblaje de modelo

Con los mejores modelos entrenados se realiza un ensamble con una regresión lineal entre sus predicciones de manera de ajustar posibles sesgos presentes en algunas acciones sobre el modelo el cual presento un MSE de 10.66

Coeficientes del modelo de ensemble

```
(-0.9671372722174851, array([0.28016131, 0.15137031, 0.57309055]))
```

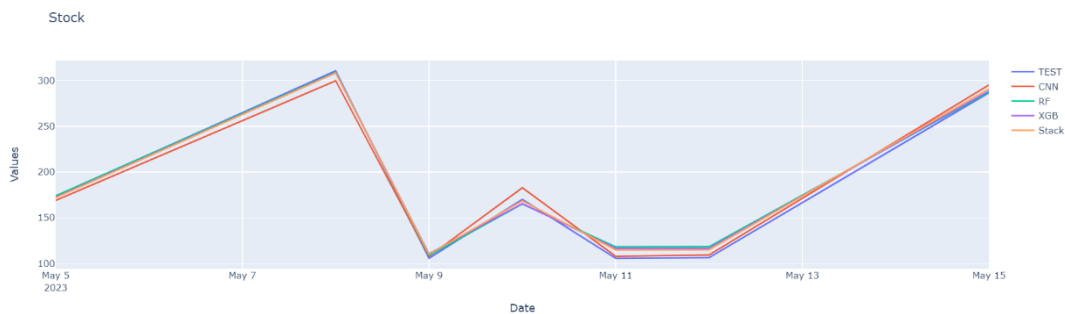
Predicción de ensemble

```
# Cambio de estructura de predicciones
RDM_FRST = list(RDM_FRST.values.reshape(-1))
XGB = list(XGB.values.reshape(-1))

# Agrupamiento de predicciones
preds = np.stack([
    CNN_LSTM, RDM_FRST, XGB
])
preds = pd.DataFrame(preds).transpose()

# Predicción regresión
STCK = stacker.predict(preds)
```

Ajuste de ensemble y predicciones



3.4 Modelo de optimización

La optimización de un portafolio es un proceso que implica la selección de una combinación de activos de inversión que maximiza el rendimiento esperado y minimiza el riesgo, en este caso estos activos corresponden al valor de cierre de las acciones. La selección de los activos que se incluirán en el portafolio es importante, se debe tener en cuenta la diversificación de este.

Para realizar dicha implementación, probaremos dos métodos:

- **Simulación de Monte Carlo:** Asignando pesos de manera aleatoria
- **Optimización matemática:** A través de una función de minimización del Sharpe Ratio

Simulación de Monte Carlo

La implementación realizada optimiza un portafolio utilizando simulación de Monte Carlo y el índice Sharpe, se considera que la cartera con el índice de Sharp máximo es la óptima.

Se calcula el rendimiento porcentual diario de cada acción y se calcula su correlación:

```
# Mean daily return
# Using the percent change function, the mean daily return of the stocks can be found.
df.pct_change(1).mean()
# Correlation between stocks
# Using the mean daily return, the correlation between the stocks can be determined.
df.pct_change(1).corr()
```

Se calcula el rendimiento aritmético y el rendimiento logarítmico de cada acción. Se utiliza el rendimiento logarítmico para calcular la covarianza:

```
# Arithmetic return
# The percentage change method gives the arithmetic return.
arith_returns = df.pct_change(1)
arith_returns.head()
# Covariance
# The covariance function gives the covariance.
log_returns.cov()
```

Se calcula la rentabilidad esperada, la volatilidad esperada y la relación de Sharpe para una cartera de tres activos (Facebook, Twitter y Apple) ponderados por un vector de pesos aleatorios. La rentabilidad esperada se obtiene multiplicando el promedio de los retornos logarítmicos por los pesos de la cartera y luego multiplicando por 252 (el número de días de negociación en un año). La volatilidad esperada se calcula mediante la raíz cuadrada de la multiplicación del producto punto del vector de pesos transpuesto por la matriz de covarianza de los retornos logarítmicos multiplicado por 252. La relación de Sharpe se calcula dividiendo la rentabilidad esperada por la volatilidad esperada.

```
expected_return = np.sum((log_returns.mean() * weights) * 252)
expected_vol = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov() * 252, weights)))
sharpe_r = expected_return/expected_vol
```

Se eligen pesos aleatorios para cada acción y se calcula el rendimiento esperado, la volatilidad esperada y la relación Sharpe para el portafolio. Este proceso se repite 7000 veces para crear múltiples portafolios aleatorios y se encuentra la cartera con el mayor índice Sharpe.


```

np.random.seed(200)

# Initialization of variables
portfolio_number = 7000
weights_total = np.zeros((portfolio_number, len(df.columns)))
returns = np.zeros(portfolio_number)
volatility = np.zeros(portfolio_number)
sharpe = np.zeros(portfolio_number)
for i in range(portfolio_number):
    # Random weights
    weights = np.array(np.random.random(3))
    weights = weights/np.sum(weights)
    # Append weight
    weights_total[i,:] = weights
    # Expected return
    returns[i] = np.sum((log_returns.mean()* weights) * 252)
    # Expected volume
    volatility[i] = np.sqrt(np.dot(weights.T,
    np.dot(log_returns.cov()*252, weights)))
    # Sharpe ratio
    sharpe[i] = returns[i]/volatility[i]

```

Se obtienen los resultados de la cartera con el mayor índice Sharpe, incluyendo sus pesos, el rendimiento y la volatilidad esperados. Se asigna a la variable "max_sharpe" el valor máximo del índice de Sharpe encontrado en la simulación. Luego, se identifica el índice de ese portafolio en la matriz de resultados y se asigna a "max_sharpe_index". Después, se recuperan los pesos de ese portafolio de la matriz "weights_total" y se asignan a "max_sharpe_weights". Finalmente, se calcula el retorno esperado de ese portafolio y se asigna a "max_sharpe_return", y se calcula la volatilidad esperada y se asigna a "max_sharpe_vol".

Estos resultados son utilizados posteriormente para informar la selección del portafolio óptimo de inversión de inversión.

```

max_sharpe = sharpe.max()
display(max_sharpe)

max_sharpe_index = sharpe.argmax()
display(max_sharpe_index)
max_sharpe_weights = weights_total[343,:]
display(max_sharpe_weights)

max_sharpe_return = returns[max_sharpe_index]
display(max_sharpe_return)

max_sharpe_vol = volatility[max_sharpe_index]
display(max_sharpe_vol)

```

Se genera un gráfico de dispersión para mostrar la relación entre la volatilidad y el retorno esperado para un conjunto de portafolios aleatorios compuestos por tres acciones: Facebook, Twitter y Apple. Cada punto en el gráfico representa un portafolio aleatorio y su color indica su ratio de Sharpe. El

punto rojo es el portafolio óptimo. Los ejes X e Y del gráfico representan la volatilidad y el retorno esperado, respectivamente.

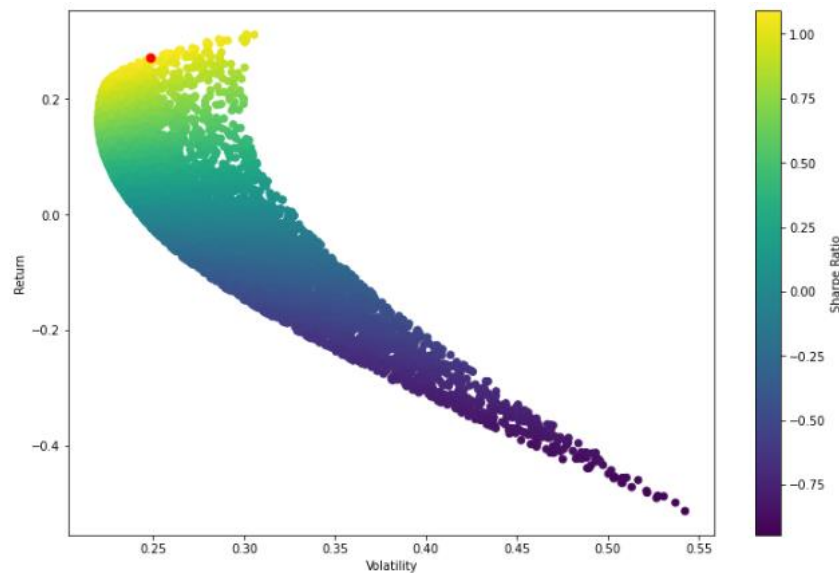


Imagen 5. Relación entre la volatilidad y el retorno esperado.

Optimización matemática

Hacer optimización matemática de un portafolio requiere de construir una serie de funciones que nos permitan procesar los resultados del módulo SciPy de Python que es el que usaremos para crear la función de optimización matemática que necesitamos.

Primero debemos definir una función que toma los pesos a probar y devuelve los retornos, la volatilidad y Sharpe Ratio asociados a la iteración.

```
# Get stats
def stats(weights):
    weights = np.array(weights)
    expected_return = np.sum((log_returns.mean()*weights) * 252)
    expected_vol = np.sqrt(np.dot(weights.T, np.dot(log_returns.cov()*252, weights)))
    sharpe_r = expected_return/expected_vol
    return np.array([expected_return, expected_vol, sharpe_r])
```

Dado que el objetivo de este modelo es minimizar el Sharpe Ratio, el cual es negativo, necesitamos crear una función que revierta el signo de dicho indicador.

```
# Minimize negative Sharpe Ratio
def sr_negate(weights):
    neg_sr = stats(weights)[2] * -1
    return neg_sr
```

Posteriormente debemos comprobar que la suma de los pesos sea igual a 1. Para ello, también se crea una función que valide lo anterior. Si el valor devuelto es 0, entonces la suma es 1.

```
# Check allocation sums to 1
def weight_check(weights):
    weights_sum = np.sum(weights)
    return weights_sum - 1
```

Lo siguiente es definir las restricciones, los límites y la suposición inicial sobre los pesos. Las restricciones se definen como una tupla de diccionarios donde el primer elemento describe el tipo de ecuación (ya que la función 'weight_check' devuelve un valor resolviendo una ecuación) y el segundo elemento describe la función como 'weight_check'. Dado que los pesos pueden estar en el rango de 0 a 1, los límites se establecen de esa manera, finalmente se establece la suposición inicial (los pesos del portafolio). Por ejemplo, para tres activos tendríamos:

```
constraints = ({'type':'eq', 'fun':weight_check})
bounds = ((0,1),(0,1),(0,1))
initial_guess = [0.3,0.3,0.4]
```

Luego, pasamos todo lo anterior como insumos de la función de minimización de SciPy, la cual toma la función que debe minimizarse, la suposición inicial y el *solver* que se utiliza para llevar a cabo la minimización. El *solver* utilizado es el de Programación Secuencial de Cuadrados Mínimos (SLSQP). Finalmente se asignan los límites y las restricciones definidas previamente.

```
# Model execution
results = minimize(sr_negate,initial_guess,method='SLSQP',bounds=bounds,constraints=constraints)
results

message: Optimization terminated successfully
success: True
status: 0
fun: -0.7669054113861363
x: [ 0.000e+00  3.683e-16  1.285e-16  0.000e+00  2.971e-16
     0.000e+00  1.000e+00  1.318e-16  6.939e-18  0.000e+00]
nit: 3
jac: [ 3.749e-01  3.468e-01  5.840e-01  1.315e+00  4.134e-01
       4.943e-01 -7.451e-09  8.451e-02  1.753e-01  2.224e-02]
nfev: 33
njev: 3
```

De los resultados de la función anterior se extraen los pesos óptimos del portafolio (los cuales se encuentran en 'x'), y el retorno, la volatilidad y Sharpe Ratio, los cuales se extraen a través de la función 'stats' definida previamente.

```
# Portfolio allocation weights
wt = results.x
weights = list(wt)
weights
```

```
[0.0,
 3.6828250552900784e-16,
 1.2846798601778002e-16,
 0.0,
 2.9706586121472033e-16,
 0.0,
 0.9999999999999999,
 1.3176270765581462e-16,
 6.938893903907228e-18,
 0.0]
```

```
# Model result
stats = list(stats(wt))
stats
```

```
[0.44270012004684706, 0.5772551783755088, 0.7669054113861363]
```

Finalmente, todo el proceso anterior se empaquetó en una función que permite procesar una solicitud particular y entregar los resultados mostrados en formato JSON:

```
# Model pipe execution
output = opt_model_pipe(data)
print(json.dumps(output, indent=4))

{
  "ticker": [
    "AAPL",
    "MSFT",
    "AMZN",
    "TSLA",
    "GOOGL",
    "GOOG",
    "NVDA",
    "BRK-B",
    "META",
    "UNH"
  ],
  "weights": [
    0.0,
    3.6828250552900784e-16,
    1.2846798601778002e-16,
    0.0,
    2.9706586121472033e-16,
    0.0,
    0.9999999999999999,
    1.3176270765581462e-16,
    6.938893903907228e-18,
    0.0
  ],
  "return": 0.44270012004684706,
  "volatility": 0.5772551783755088,
  "sharpe_ratio": 0.7669054113861363,
  "message": "Optimization terminated successfully",
  "success": "True"
}
```

Es importante mencionar que este modelo está diseñado para ser alimentado por los datos predichos por los modelos de regresión explicados previamente, los cuales (los datos) se encuentran persistidos en capa Gold.

```
# Read data from data Lake
data = pd.read_csv(gold+gold_table)
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data.head()
```

	Adj Close	Ticker
Date		
2022-05-06	156.562683	AAPL
2022-05-09	151.366486	AAPL
2022-05-10	153.805313	AAPL
2022-05-11	145.831848	AAPL
2022-05-12	141.909821	AAPL

4 Comparación de desempeño de modelos y potenciales ajustes por implementar

Experimentación con la selección de modelos y parámetros se realiza la comparación de dos métricas, el RMSE y el MAE, para la calibración que resulta victoriosa en cada uno de los tres modelos implementados. En la siguiente gráfica se puede observar el desempeño de la mejor calibración para la acción de APPLE y MICROSOFT:

	LSTM	XGBoost	Prophet		LSTM	XGBoost	Prophet
RSME	4.2131	5.794328	10.908346	RSME	7.258	10.017981	2.647911
MAE	3.3472	4.603448	8.898584	MAE	5.679	8.162889	2.165912

Imagen 6. a) Desempeño acción APPLE. B) Desempeño acción MICROSOFT.

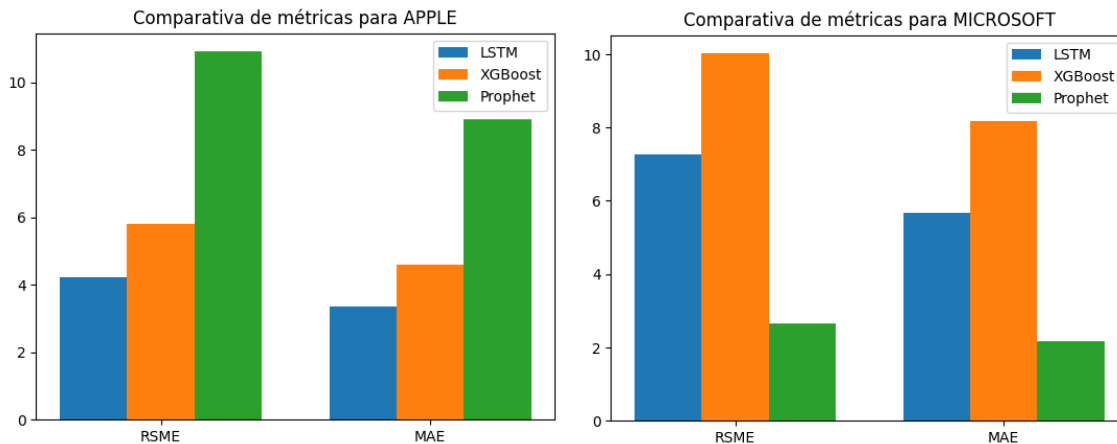


Imagen 7. Comparativa de desempeño de las acciones APPLE y MICROSOFT

Se podría decir que el modelo campeón es la red LSTM (azul), seguida del XGBoost (naranja) y por último en desempeño el modelo Prophet. Sin embargo, no es posible concluir que el uso de un modelo particular se ajuste a la predicción de todas las acciones. De las acciones revisadas es posible ver que se mantiene el desempeño del modelo obtenido con LSTM mejor que el desempeño de XGBoost, para el caso de la acción de Microsoft, el modelo obtenido con Prophet supera en un 73.65% el desempeño de XGBoost y en un 63.62% el desempeño del LSTM.

Como conclusión general de la comparativa se tiene que no hay un modelo único que permita predecir mejor en todas las series de tiempo de las acciones, por lo cual una salida es realizar un ensamble tipo promedio con los tres modelos implementados.

Como posible trabajo durante las próximas semanas estaría la exploración de la inclusión de más variables predictoras para la estimación del modelo XGBoost que incluya las características que describen la acción que se obtienen a través de Yahoo Finance como son los valores open, high, low volumen y close con el fin de ver si se obtiene una estandarización en el comportamiento de predicción del modelo entre las diferentes acciones.

Otra posible acción a realizar es revisar si para el modelo LSTM es posible incorporar variables adicionales a adjclose con el fin de agregar variables externas y las características de las acciones al conjunto de datos de entrenamiento de entrada de la red.

4.1 Desempeño del modelo LSTM

Notebook	Data histórica (años datos)	Ticker	Modelo	Grilla de hiperparámetros	Mejores hiperparámetros	MSE	RMSE	MAE	MAPE
9.1	1	AAPL	Topologia 1	Grilla 1	{'units': 128, 'learning_rate': 0.01, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	18.0272	4.2458	3.3962	2.30%
9.2	1	AAPL	Topologia 1	Grilla 2	{'units': 64, 'learning_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.2, 'batch_size': 32}	17.7501	4.2131	3.3472	2.26%
9.3	1	AAPL	Topologia 2	Grilla 1	{'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}	25.4527	5.0451	4.0375	2.74%
9.4	1	AAPL	Topologia 2	Grilla 2	{'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}	21.3768	4.6235	3.6312	2.47%
9.5	3	AAPL	Topologia 1	Grilla 1	{'units': 64, 'learning_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.2, 'batch_size': 32}	21.8768	4.6773	3.831	2.58%
9.6	3	AAPL	Topologia 1	Grilla 2	{'units': 64, 'learning_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.2, 'batch_size': 32}	17.7281	4.2105	3.4208	2.32%
9.7	3	AAPL	Topologia 2	Grilla 1	{'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}	25.9318	5.0923	4.0945	2.80%
9.8	3	AAPL	Topologia 2	Grilla 2	{'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}	19.6706	4.4352	3.5535	2.41%
9.1	1	MSFT	Topologia 1	Grilla 1	{'units': 128, 'learning_rate': 0.01, 'epochs': 25, 'dropout_rate': 0.1, 'batch_size': 64}	89.1456	9.4417	7.623	3.02%
9.2	1	MSFT	Topologia 1	Grilla 2	{'units': 128, 'learning_rate': 0.01, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	52.678	7.258	5.679	2.25%
9.3	1	MSFT	Topologia 2	Grilla 1	{'units': 256, 'learning_rate': 0.001, 'epochs': 50, 'dropout_rate': 0.1, 'batch_size': 128}	93.334	9.661	8.0206	3.18%
9.4	1	MSFT	Topologia 2	Grilla 2	{'units': 128, 'learning_rate': 0.001, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	83.0795	9.1148	7.4169	2.96%
9.5	3	MSFT	Topologia 1	Grilla 1	{'units': 128, 'learning_rate': 0.01, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	76.1458	8.7262	7.1521	2.83%
9.6	3	MSFT	Topologia 1	Grilla 2	{'units': 128, 'learning_rate': 0.01, 'epochs': 50, 'dropout_rate': 0.2, 'batch_size': 64}	55.5501	7.4532	5.88	2.35%
9.7	3	MSFT	Topologia 2	Grilla 1	{'units': 256, 'learning_rate': 0.001, 'epochs': 100, 'dropout_rate': 0.2, 'batch_size': 128}	84.309	9.182	7.505	2.99%
9.8	3	MSFT	Topologia 2	Grilla 2					

Tabla 1. Desempeño del modelo LSTM para las acciones AAPL y MSFT.

4.2 Desempeño del modelo XGBoost

Para verificar el desempeño del modelo XGBoost obtenido, se identifica el mejor modelo como el que presenta menor error (10.37) y se procede a pronosticar valores futuros con dicha configuración (lags = 14, learning_Rate=0.1, max_depth=3,max_iter=100).

Stock	MAE	RMSE
AAPL	4.60344782742587	5.794328495
MSFT	8.162888729211057	10.01798114
AMZN	11.79751968383789	14.68884296
TSLA	14.046327648740826	17.47174258
GOOGL	11.693109396732215	13.84513768
GOOG	11.373661272453539	13.48047189
NVDA	9.69188100641424	11.68712364
BRK	6.72775800300367	8.555982577
META	6.509180820349491	9.89094408
UNH	12.229493343468869	14.80313255
GSPC	92.6665575432055	113.3192482

Tabla 2. Desempeño modelo XGBoost.

5 Análisis de resultados y/o ajustes necesarios a los modelos, al esquema general de solución, o a los problemas de negocio o de analítica de datos

Con la implementación de los distintos modelos se pudo realizar una profundización en los diferentes parámetros y estructuras de estos, si bien cada uno presenta características distintas para el preprocesamiento de datos, las fuentes escogidas permiten el entrenamiento efectivo de los distintos modelos. Por lo que el proceso de ETL y preprocesamiento puede ajustarse para tratar de manera general la estructura de los datos y permitir que las subfunciones para cada modelo terminen de ajustarlas a sus necesidades.

Es claro que cada acción representa una serie de tiempo con características distintas y que la eficiencia del pronóstico de cada acción depende de las capacidades de cada modelo para identificar dichas características. Un ajuste necesario para el esquema general de la solución sería establecer un ensamble entre los modelos aquí evidenciados y optimizados para retener los aportes propios de cada modelo al entendimiento de las acciones.

En el entrenamiento de modelos que incorporan variables exógenas se identifica la importancia de una selección específica de variables a incluir, pues si bien es posible encontrar variables macroeconómicas con la dimensión de tiempo requerida para el análisis, algunas de estas no representan un peso considerable en el mercado de valores y su inclusión en los modelos agrega una variabilidad no deseada.

Con esto en mente el esquema general de la solución ajusta su alcance analítico y presentaría un portafolio optimizado según las acciones definidas por el usuario y un set de variables exógenas priorizadas por el experto del negocio. Creado mediante el pronóstico de valores de acciones con una ventana de 7 días en el futuro basado en el ensamblaje de una red neuronal, modelo xgboost y

model prophet. La solución es un API que retorna los pesos de las acciones en el portafolio optimizado, así como los valores esperados en el periodo.

6 Avance y plan de implementación del prototipo

6.1 Avance del prototipo

El avance de la implementación durante las semanas 1 a la 4 se condensa en la Tabla 3

Capa	Funcionalidad	Código	Hecho
Datos			
	Descarga y almacenamiento de datos mediante API (Yahoo Finance, Quandl, Fred)	DE - 1.1 - ETL Historical Data.ipynb	Si
Procesos			
	Acceso a datos financieros	DE - 1.1 - ETL Historical Data.ipynb	Si
	La persistencia de los datos se realizará por capas, como en un lago de datos, siendo las capas: - Bronze - Silver - Gold: Data preparada y estructurada para alimentar al modelo en cuestión	DE - 1.1 - ETL Historical Data.ipynb	Si
	Empaquetamiento en contenedor Docker	PENDIENTE	No
Modelos/Algoritmos			
	Implementación y pruebas Modelo predictivo LSTM	SPP - 1.8 - LSTM.ipynb	Si
	Implementación y pruebas Modelo predictivo XGBoost		Si
	Implementación y pruebas Modelo predictivo Prophet	SPP - 2.2 - Time Series (FBProphet).ipynb	Si
	Implementación y pruebas Modelo de optimización minimizando Sharpe Ratio	Math Method.ipynb	Si
	Implementación y pruebas Modelo de optimización con simulación de Montecarlo	Random Method.ipynb	Si
	Integración de modelo predictivo y prescriptivo (optimización)	PENDIENTE	No

Tabla 3. Avance de la implementación del prototipo.

El avance de la implementación se encuentra anexa con este documento en los siguientes archivos:

- DE - 1.1 - ETL Historical Data.ipynb
- PO - 1.3 - Math Method.ipynb
- SPP - 1.8 - LSTM.ipynb
- SPP - 2.2 - Time Series (FBProphet).ipynb

DE - 1.1 - ETL Historical Data.ipynb

Se encarga de realizar el proceso de adquisición de los datos y su almacenamiento en el data lake en sus diferentes zonas. Se definió para descargar datos históricos de un año (por ahora 3 años) desde la API Yahoo Finance. En la capa Bronze se encuentran los datos crudos extraídos por cada ticker. En la capa Silver están la totalidad de los datos depurados e integrados en una sola tabla.

PO - 1.3 - Math Method.ipynb

Ejecuta el modelo de optimización a través de un pipeline que consume el portafolio de acciones ubicado en la capa Gold. La función devuelve los pesos de asignación de cada acción en el portafolio, así como el rendimiento esperado, la volatilidad y Sharpe Ratio del portafolio optimizado.

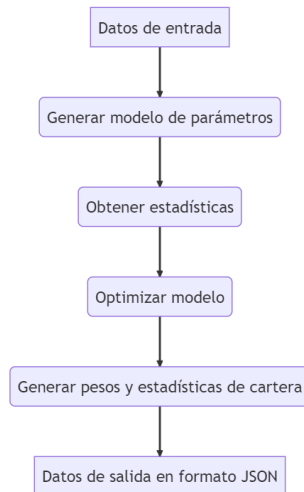


Imagen 8. Pipeline Optimización.

El proceso comienza con la obtención de los datos de entrada, que son los precios históricos de las acciones luego, se generan los parámetros del modelo a partir de los datos de entrada. Posteriormente, se obtienen las estadísticas de la cartera óptima utilizando la función stats, que calcula la rentabilidad esperada, la volatilidad y la relación de Sharpe de la cartera. Estas estadísticas se utilizan para optimizar el modelo, que minimiza el negativo de la relación de Sharpe. Una vez optimizado el modelo, se generan los pesos de la cartera óptima y las estadísticas de la misma. Por último, se devuelve un objeto JSON con los resultados del modelo, incluyendo los pesos de la cartera, la rentabilidad esperada, la volatilidad y la relación de Sharpe.

Salida en formato JSON:

```

{
  "ticker": [
    "AAPL",
    "MSFT",
    "AMZN",
    "TSLA",
    "GOOGL",
    "GOOG",
    "NVDA",
    "BRK-B",
    "META",
    "UNH"
  ],
  "weights": [
    0.0,
    3.6828250552900784e-16,
    1.2846798601778002e-16,
    0.0,
    2.9706586121472033e-16,
    0.0,
    0.9999999999999999,
    1.3176270765581462e-16,
    6.938893903907228e-18,
    0.0
  ],
  "return": 0.44270012004684706,
  "volatility": 0.5772551783755088,
  "sharpe_ratio": 0.7669054113861363,
  "message": "Optimization terminated successfully",
  "success": "True"
}

```

Imagen 9. Salida JSON optimización.

SPP - 1.8 - LSTM.ipynb

Contiene el pipeline para ejecutar el modelo de predicción LSTM para predecir los precios de acciones para una serie de valores específicos ticker. El proceso de entrenamiento de la red comienza por cargar los datos, normalizarlos y prepararlos para ser introducidos en el modelo. Luego se inicia un ciclo for para cada valor en ticker. Dentro del ciclo for, se extrae el conjunto de datos para el valor de ticker actual y se prepara para el entrenamiento y prueba del modelo. El conjunto de entrenamiento se divide en `x_train` y `y_train`, se convierten en arreglos de Numpy y `x_train` se reformatea a un arreglo 3D para poder ser utilizado por la capa de entrada de la red neuronal.

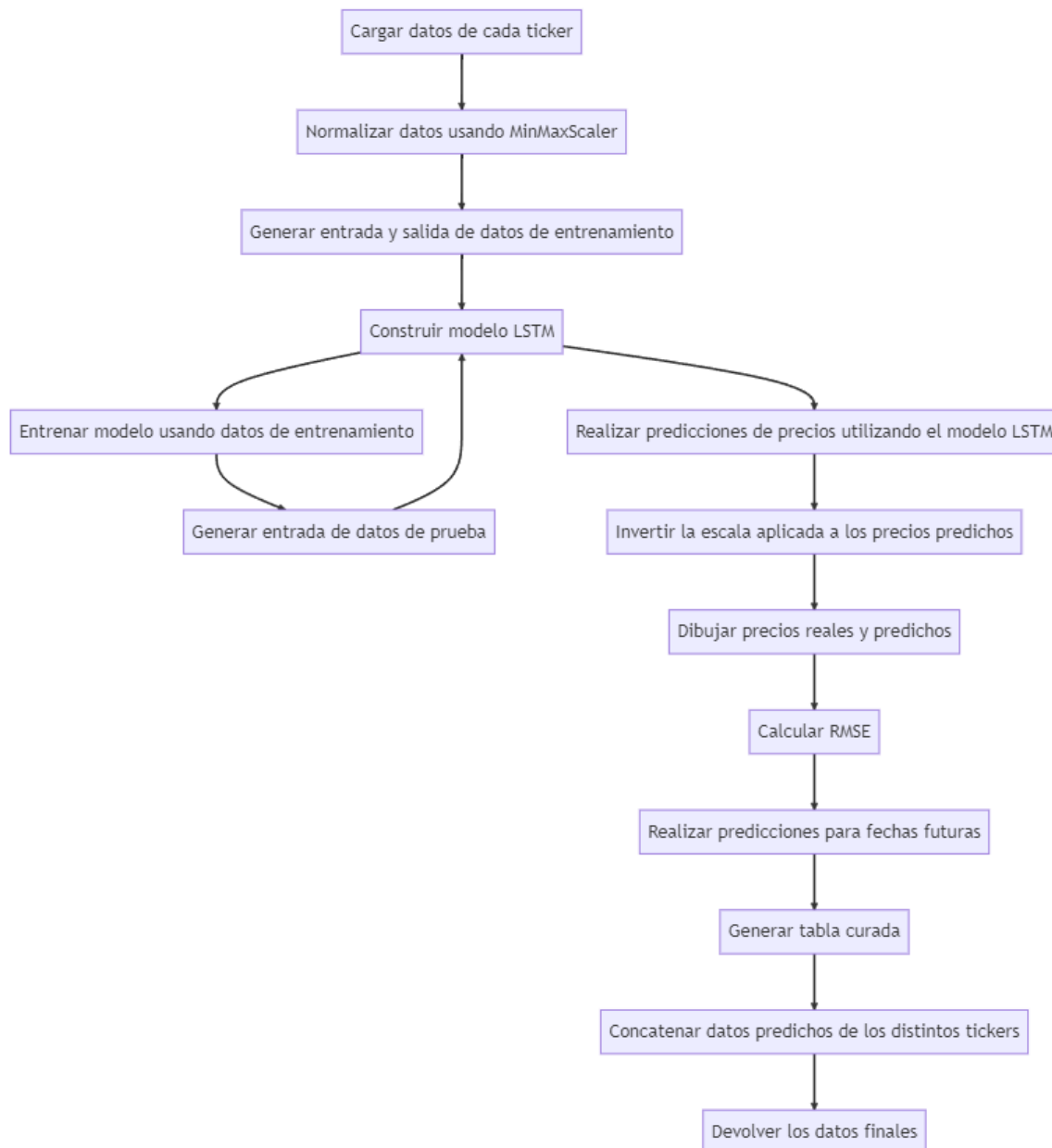


Imagen 10. Pipeline LSTM.

Para generar la salida de prueba del modelo, se extrae un conjunto de datos se utilizan los últimos de datos para iniciar las predicciones. La salida de prueba se genera con la función `model.predict()` y luego se invierten las transformaciones de escala para obtener los precios reales. Se genera una

gráfica que compara los precios reales con los precios predichos y se calcula el error cuadrático medio entre los precios reales y los precios predichos.

El modelo entrenado para hacer predicciones de precios futuros para un número específico de días. Finalmente, retorna una tabla que contiene los datos históricos y los datos de precios predichos para cada valor en ticker.

6.2 Plan para entrega del prototipo

Como parte de las actividades a ejecutar dentro de las siguientes 3 semanas están:

	Actividad	Semana 8 - 14 mayo	Semana 15 - 21 mayo	Semana 22 - 28 mayo
1	Seleccionar y mejorar la implementación del modelo predictivo			
2	Integrar el modelo de predicción seleccionado con el modelo de optimización			
3	Organizar y unificar el código para orquestar			
4	Empaquetar el modelo			
5	Generar la imagen Docker			
6	Crear la imagen en un contenedor AWS			
7	Realizar pruebas de funcionamiento de la API			
8	Documentar			
9	Preparación de la presentación			

Tabla 4. Plan de actividades para cierre de entrega del prototipo.