Breve Informe Cloud Computing

Despliegue Modelo de Machine Learning con FastAPI y Github

Diplomado de Data Science – UAI

Integrantes:

Gerardo Rios

Washington Lizana

1. Preparación y Análisis de Datos

El presente trabajo busca Implementar un Modelo de Machine Learning que busca predecir el evento de un alza del tipo de cambio USD-CLP para el día siguiente. Este Modelo es una Red Neuronal que se desarrolló por el área de Trading de la Mesa de Dinero de BancoEstado y que actualmente está en uso.

El modelo utiliza como variables independientes diversas medidas macroeconómicas, como Tasas de Interés, Inflación, entre otros.

Dada la naturaleza del problema, se agregaron variables categóricas a las que se tenían en un principio.

Como mencionamos anteriormente, se toman variables que el mercado financiero considera relevante a la hora de predecir el precio del dólar (paridad USDCLP).

A fin de ocupar una fuente confiable, la base "bruta" fue obtenida de Bloomberg mediante una consulta automatizada a su API. La cual, a fin de no sobre cargar el nivel de consultas, el código se dirige a la API y obtiene solo los datos faltantes, desde última fecha cargada hasta hoy.

Para ver código, favor abrir "saca_datos_bbg_a_bd.py"

- **Metodología usada con datos faltantes y erróneos:**

Suele ocurrir que, al tener feriados locales, como por ejemplo feriado en USA, pero no en Chile, tenemos datos faltantes. En este caso, lo que consideramos correcto es "repetir" el último dato, para que el movimiento de este (en este caso 0%) no tenga incidencias en el precio del dólar.

- **Variables numéricas a categorías**

TPM (Chile y USA): separamos 4 categorías separadas en cuales consideramos nosotros como Expnsiva, Neutral y Restrictiva.

TPM (Chile y USA) 1 dummy que nos avisa si un día en específico cambio la TPM.

Equity: Máximos locales e históricos, como indicadores de Momentum (o fuerza) relativa del los precios en el Equity.

Tecnicos sobre índices: Como otras medidas de variables creadas por nosotros, construimos variables de análisis técnico para usar predictores típicos del Mercado Financiero (Indicadores de Momentum, regresión a la Media, ETC.)

Para este punto se necesita instalar una librería creada por nosotros (conexion-a-db), para más información ver README.txt

- **Metodología ocupada**

Se adjunta base da datos (db_data_bbg.db) la cual tiene la información necesaria para el modelo, en caso de querer actualizar la información, se puede encontrar el código de actualización de la data en (saca_datos_bbg_a_bd.py), es necesario una suscripción activa a Bloomberg Anywhere. Además, en este último podemos encontrar el tratamiento ocupado para datos faltantes y erróneos explicados con anterioridad.

Ejemplo da la data obtenida desde Bloomberg a la base de datos usando nuestra librería conexion-a-db disponible en GITHUB.

```
import conexion_a_db as db
```
✓ 1.4s

```
str = "./DATA/db_data_bbg.db"

equity = db.Descargar_DF(str, 'equity')
equity.head(10)
```
✓ 0.0s

	Fecha	Pais	Valor
0	2021-01-04 00:00:00	Eurozona EuroStoxx	3564.390
1	2021-01-04 00:00:00	China Shanghai	5267.720
2	2021-01-04 00:00:00	Alemania DAX	13726.740
3	2021-01-04 00:00:00	Canada S&P	17527.770
4	2021-01-04 00:00:00	Australia S&P	6684.247
5	2021-01-04 00:00:00	Chile IPSA	4257.440
6	2021-01-04 00:00:00	Chile IGPA	21360.310
7	2021-01-04 00:00:00	Mexico Mexbol	44703.000
8	2021-01-04 00:00:00	Brasil Ibovespa	118854.710
9	2021-01-04 00:00:00	Usa DowJones	30223.890

En data_getters.py podemos encontrar indicadores de Analisis técnicos y Otros categóricos descritos anteriormente,

Ejemplo 1: Análisis técnico

```python
def add_technical_indicators(var:str, df):
    import pandas as pd
    # Create all technical indicators at once for better performance
    ma_30 = df[var].rolling(window=30).mean()
    ma_360 = df[var].rolling(window=360).mean()
    std_30 = df[var].rolling(window=30).std()
    std_360 = df[var].rolling(window=360).std()
    high_30 = df[var].rolling(window=30).max()
    high_360 = df[var].rolling(window=360).max()
    low_30 = df[var].rolling(window=30).min()
    low_360 = df[var].rolling(window=360).min()
    range_30 = high_30 - low_30
    range_360 = high_360 - low_360
    zscore = (df[var] - ma_30) / std_30
    delta = df[var].diff()

    # Create new dataframe with all indicators
    new_cols = pd.DataFrame({
        f'{var}_MA_30': ma_30,
        f'{var}_MA_360': ma_360,
        f'{var}_STD_30': std_30,
        f'{var}_STD_360': std_360,
        f'{var}_HIGH_30': high_30,
        f'{var}_HIGH_360': high_360,
        f'{var}_LOW_30': low_30,
        f'{var}_LOW_360': low_360,
        f'{var}_Range_30': range_30,
        f'{var}_Range_360': range_360,
        f'{var}_ZScore': zscore,
        f'{var}_Delta': delta
    })

    # Concatenate with df_master
    df = pd.concat([df, new_cols], axis=1)
    return df
```

Ejemplo 2: TPM como variable categórica

```python
def add_tpm_category(df_macro):
    ranges_tpm_cl = [df_macro["tpm_cl"].min() + (i) * (df_macro["tpm_cl"].max() - df_macro["tpm_cl"].min()) / 3 for i in range(4)]
    ranges_tpm_us = [df_macro["tpm_us"].min() + (i) * (df_macro["tpm_us"].max() - df_macro["tpm_us"].min()) / 3 for i in range(4)]

    def category_tpm(tpm,ranges):
        if tpm <= ranges[1]:
            return 0 # Baja
        elif tpm <= ranges[2]:
            return 1 # Media
        else:
            return 2 # Alta
        #Agregamos columnas binarias de is in category
    df_macro["tpm_cl_cat"] = df_macro["tpm_cl"].apply(lambda x: category_tpm(x, ranges_tpm_cl))
    df_macro["tpm_us_cat"] = df_macro["tpm_us"].apply(lambda x: category_tpm(x, ranges_tpm_us))
    for cat in range(3):
        df_macro[f'tpm_cl_is_{cat}'] = (df_macro["tpm_cl_cat"] == cat).astype(int)
        df_macro[f'tpm_us_is_{cat}'] = (df_macro["tpm_us_cat"] == cat).astype(int)
    return df_macro
```

Ver data_getters.py para encontrar toda la información sobre nuevas variables.

2. Entrenamiento del Modelo

Como entrenamiento del modelo, hemos dejado un Jupyter Notebook con el detalle del modelo seleccionado y porque se seleccionó, además que se buscaba predecir con este.

En descripción de lo que se hizo, se busco predecir donde "va" el dólar en base a variables externas a este (multivariada) por lo que, definiendo direccionalidad como:

$$y_t = USDCLP_{t+1} - USDCLP_t > 0$$

i. Y ocupando un modelo de clasificación (sube, baja o se mantiene), creamos la variable:

```
# Definimos nuestra variable objetivo
isup = (data['usdclp'].shift(-1) - data['usdclp']) > 0

# Lo agregamos al dataframe
data['isup'] = isup

# Eliminamos filas con valores nulos (en este caso el último día)
data = data.dropna()
```

ii. Definimos variables predictoras y objetivo

```
Index(['Fecha', 'inflacion_cl', 'inflacion_us', 'tpm_cl', 'tpm_us', 'ipc_cl',
       'ipc_us', 'sp500', 'ipsa', 'usdclp', 'tpm_cl_cat', 'tpm_us_cat',
       'tpm_cl_is_0', 'tpm_us_is_0', 'tpm_cl_is_1', 'tpm_us_is_1',
       'tpm_cl_is_2', 'tpm_us_is_2', 'tpm_cl_change', 'tpm_us_change',
       'new_max_sp500', 'new_max_ipsa', 'new_max_usdclp', 'tpm_diff',
       'tpm_diff_pos', 'isup'],
      dtype='object')
```

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Definimos las features y la variable objetivo
features = [c for c in data.columns if c not in ['isup', 'usdclp', "Fecha"]]
X = data[features]
y = data['isup']
```

iii. Separamos la data como entrenamiento y validación de 80/20.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

iv. Entrenamos

```
# Se entrenan los modelos
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)

model_ab = AdaBoostClassifier(n_estimators=100, random_state=42)
model_ab.fit(X_train, y_train)
y_pred_ab = model_ab.predict(X_test)

model_mlp = MLPClassifier(hidden_layer_sizes=(50, 25), max_iter=500, random_state=42)
model_mlp.fit(X_train, y_train)
y_pred_mlp = model_mlp.predict(X_test)
```

v. Creamos y analizamos resultados:

```
print_metrics(y_test, y_pred_rf, "Random Forest")
print_metrics(y_test, y_pred_ab, "AdaBoost")
print_metrics(y_test, y_pred_mlp, "MLP Classifier")
```

✓ 0.0s

```
Metrics for Random Forest:
Accuracy: 0.5330
Precision: 0.3333
Recall: 0.0095
F1 Score: 0.0185
ROC AUC: 0.4966


Metrics for AdaBoost:
Accuracy: 0.5683
Precision: 0.5946
Recall: 0.2095
F1 Score: 0.3099
ROC AUC: 0.5433


Metrics for MLP Classifier:
Accuracy: 0.5022
Precision: 0.4726
Recall: 0.6571
F1 Score: 0.5498
ROC AUC: 0.5130
```

Resumen de Resultados:

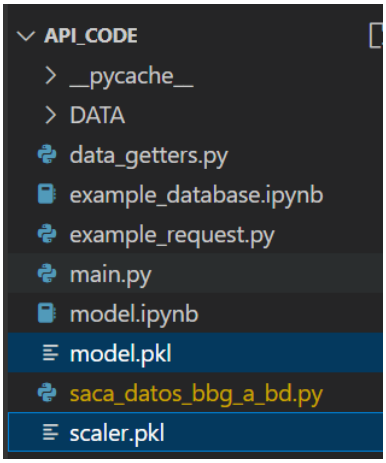
Al ser un modelo de clasificación, nuestro indicador clave será el F1 score, quien nos trega información clave acerca de precisión y recall.

En este caso, el modelo elegido será MLP Classifier (modelo de red neuronal de una dirección) o feed-forward siguiendo un random-forest.

vi. Guardamos modelo y scaler en formato pkl

```python
# Se guarda el MLP Classifier ya que es el que mejor resultado tuvo en términos de F1-Score
import joblib
joblib.dump(model_mlp, 'model.pkl')

# Se guarda el scaler
joblib.dump(scaler, 'scaler.pkl')
```

Como podemos ver, se crean los 2 .pkl. (Hacer caso omiso al .py con error, ya que en el PC que se tomaron los ejemplos, no hay un Bloomberg con licencia activa para la actualización automática de los datos, pero para nuestro ejemplo no es necesario).



3. Creación de la API con FastAPI

En la terminal corremos main.py (donde esta nuestra FastAPI)

Antes de empezar con la conexión a la API rejaremos una breve explicación de lo que hace (aunque como hemos recomendado anteriormente, es mejor ver el codigo fuente en main.py)

Hemos dejado una class con diferentes activos y tecnicos, los cuales podemos ir probando como un movimiento de estos puede provocar movimientos en el USDCLP (por ejemplo, si el IPSA sube 5%, como puede impactar en el precio del dólar).

i. Dejamos una lista de las variables que podemos ir "moviendo".

```python
class ModelInput(BaseModel):
    inflacion_cl: float
    inflacion_us: float
    tpm_cl: float
    tpm_us: float
    ipc_cl: float
    ipc_us: float
    sp500: float
    ipsa: float
    tpm_cl_cat: float
    tpm_us_cat: float
    tpm_cl_is_0: float
    tpm_us_is_0: float
    tpm_cl_is_1: float
    tpm_us_is_1: float
    tpm_cl_is_2: float
    tpm_us_is_2: float
    tpm_cl_change: float
    tpm_us_change: float
    new_max_sp500: float
    new_max_ipsa: float
    new_max_usdclp: float
    tpm_diff: float
    tpm_diff_pos: float
```

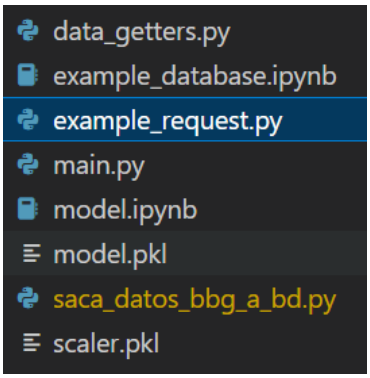
ii. Ahora si, volviendo con el codigo, para correr la Api con FAST API, corremos el siguiente comando en la terminal:

```
PS C:\Users\Gerandres\Desktop\Trabajo Cloud Computing\API_code> fastapi dev main.py
```

iii. Resultados:

```
FastAPI   Starting development server 🚀

          Searching for package file structure from directories with __init__.py files
          Importing from C:\Users\Gerandres\Desktop\Trabajo Cloud Computing\API_code

 module   main.py

   code   Importing the FastAPI app object from the module with the following code:

          from main import app

    app   Using import string: main:app

 server   Server started at http://127.0.0.1:8000
 server   Documentation at http://127.0.0.1:8000/docs

    tip   Running in development mode, for production use: fastapi run

          Logs:

   INFO   Will watch for changes in these directories: ['C:\\Users\\Gerandres\\Desktop\\Trabajo Cloud Computing\\API_code']
   INFO   Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
   INFO   Started reloader process [16860] using WatchFiles
   INFO   Started server process [5348]
   INFO   Waiting for application startup.
   INFO   Application startup complete.
```

iv. Para probar nuestra API, es recomendable abrir una nueva terminal, además hemos dejado como ejemplo un código llamado



v. El cual tiene valores como ejemplos de cómo debería moverse el CLP Curency, con los movimientos de otros activos;

```python
import requests

API_URL = "http://127.0.0.1:8000/predict"

# Sample payload matching all required model features.
PAYLOAD = {
    "inflacion_cl": 3.1,
    "inflacion_us": 2.4,
    "tpm_cl": 5.25,
    "tpm_us": 4.0,
    "ipc_cl": 120.5,
    "ipc_us": 130.8,
    "sp500": 4300.6,
    "ipsa": 5200.1,
    "tpm_cl_cat": 2,
    "tpm_us_cat": 1,
    "tpm_cl_is_0": 0,
    "tpm_us_is_0": 0,
    "tpm_cl_is_1": 1,
    "tpm_us_is_1": 0,
    "tpm_cl_is_2": 0,
    "tpm_us_is_2": 1,
    "tpm_cl_change": 0.15,
    "tpm_us_change": -0.1,
    "new_max_sp500": 0,
    "new_max_ipsa": 1,
    "new_max_usdclp": 0,
    "tpm_diff": 1.25,
    "tpm_diff_pos": 1,
}


def main() -> None:
    response = requests.get(API_URL, params=PAYLOAD, timeout=10)
    response.raise_for_status()
    print(response.json())


if __name__ == "__main__":
    main()
```

vi. Corremos desde la terminal (nueva)

```
PS C:\Users\Gerandres\Desktop\Trabajo Cloud Computing\API_code> python example_request.py
```

Resultados:

```
{'model_name': 'MLPClassifier', 'output': 0.0}
```

El modelo MLP nos dice que los movimientos de ese día, no debería generar un movimiento del tipo de cambio,