



# INFORME LABO III

Forecasting avanzado

## Descripción breve

Se busca encontrar el mejor modelo para la predicción de ventas, en el contexto de la vida real. Se utilizaran varios modelos BaseLine – LGBM – Prophet – LSTM y Autogluon.

Tabla de contenido

Hipótesis Experimental ..... 2

Diseño experimental ..... 2

    2.1 Descripción del Dataset..... 2

    2.2 Modelo Base Line: ..... 3

    2.3 LightGBM ..... 3

    2.3 AutoGluon. .... 4

    2.3 Propeth..... 5

    2.4 LSTM ..... 5

Procedimientos y Recopilación de Resultados..... 7

Análisis de los resultados. .... 8

## Hipótesis Experimental

Utilizando como base una competencia Kaggle, experimentaremos con distintas herramientas de machine learning. Durante los experimentos se validarán los resultados de modelos ya conocidos como LightGBM con feature engineering, probaremos también soluciones más modernas como Autogluon. Vamos a poner a prueba algunos de los modelos vistos en la asignatura de series temporales como Propeth y LSTM. Se usará como modelo baseline el promedio histórico de 12 meses, que ofrece muy buenos resultados.

El objetivo es predecir las ventas para febrero 2020, teniendo información disponible hasta diciembre 2019.

## Diseño experimental

### 2.1 Descripción del Dataset

El set de datos original contiene 36 meses de ventas, los registros de ventas están abiertos por cliente y producto por cada periodo. Si un cliente no realiza compras de un producto para un periodo, no se tiene el registro de compra en cero para ese cliente y producto.

Se dispone también información de las categorías de productos, el stock por periodo y los productos a predecir.

- **Períodos:** Enero 2017 - Diciembre 2019 (36 meses)
- **Objetivo:** Predecir ventas totales por "product\_id" a febrero 2020
- **Productos:** 1188
- **Clientes:** 597
- **Ventas:** 2.293.481 registros
- **Ventas procesadas:** 17,021,654 registros si se completan los registros de compra en cero
- **Target:** tn
- **Productos a predecir:** 780

## 2.2 Modelo Base Line:

Este modelo asume que **la mejor predicción para febrero 2020 es el promedio mensual de ventas del año 2019** para cada producto. Es un enfoque simple que demostró ser efectivo.

## 2.3 LightGBM

Esta estrategia consiste en un solo modelo para todas las predicciones.

Previo a la corrida, se realizaron varias tareas para aumentar la información del dataset.

El primer paso fue completar con 0 los periodos para los que un cliente no registraba compras de un producto. Esto se hizo por cada cliente desde que realizó su primera compra. Es decir, si el cliente 1 realizó su primera compra en Julio 2017 y compró el producto 20001, desde Julio 2017 se comenzaron a completar los registros en 0 para los productos no comprados, no para los meses anteriores a la primera compra.

Luego se crearon nuevas variables a partir de los datos anteriores de ventas con cero, todo esto con el objetivo de capturar patrones temporales, estacionalidad, tendencias y comportamientos de clientes/productos que este tipo de modelos puedan explotar para realizar mejores predicciones. **Al finalizar este proceso, nuestro dataset tenía 98 variables.**

**Las mas importantes detectadas por nuestro modelo fueron:**

1. `ma_tn_12m` # Media móvil 12 meses
2. `coef_variacion_6m` # Volatilidad
3. `ma_tn_6m` # Media móvil 6 meses
4. `promedio_historico_mes` # Estacionalidad
5. `brand` # Marca del producto

El LGBM se corrió con los 17 millones de registros, se estandarizaron las variables numéricas mediante `StandarScaler`

Se realizaron varias corridas, se agregaron los hiperparametros “`simple_weight`”, “`linear_tree`”, y se realizó la optimización por optuna para evaluar los resultados.

```

'num_leaves': 31, # la corrida en en kaggle mejor tiene 31 leaves
'learning_rate': 0.05,
'feature_fraction': 0.9,
'bagging_fraction': 0.8,
'bagging_freq': 5,
'verbose': -1,
'random_state': 42,
#'sample_weight': True,
#'linear_tree': True,

#'num_leaves': 63,
#'learning_rate': 0.03738149205005092,
#'feature_fraction': 0.5576004633455672,
#'bagging_fraction': 0.8444560165276432,
#'bagging_freq': 1,
#'min_data_in_leaf': 83, # NUEVO - evita overfitting
#'lambda_l1': 0.0016447293604905321,
#'lambda_l2': 3.901069568054151e-06, # NUEVO - regularización

```

## 2.3 AutoGluon.

Esta estrategia consiste en implementar 780, uno por cada producto. Se implementaron varias estrategias, una de las configuraciones que mejor resultado fue la que se detalla a continuación.

```

# ⚙️ 5. Definir y entrenar predictor
# prediction_length: número de periodos futuros a predecir (aquí 2 meses: enero y febrero 2020)
# target: nombre de la columna objetivo a predecir, en este caso 'tn'
# freq: frecuencia temporal de los datos, 'MS' significa "Month Start" (inicio de mes)
predictor = TimeSeriesPredictor(
    prediction_length=2,
    target='tn',
    freq='MS', # Frecuencia mensual (Month Start),
    quantile_levels=[0.05, 0.25, 0.5, 0.75, 0.95]
)

predictor.fit(ts_data,
    num_val_windows=2,
    time_limit=60*60,
    #num_cpus=os.cpu_count() - 1,
    enable_ensemble=True
)

```

Python

Internamente AutoGluon utilizo los siguientes modelos:

	model	score_test	score_val	pred_time_test	pred_time_val	fit_time_marginal	fit_order
0	ChronosFineTuned[bolt_small]	-0.153592	-0.150235	0.416610	0.070560	129.696845	8
1	WeightedEnsemble	-0.153915	-0.133001	10.015458	3.903604	1.090130	13
2	TemporalFusionTransformer	-0.162828	-0.138842	0.340633	0.201673	240.356251	9
3	PatchTST	-0.168386	-0.149907	0.847054	0.355600	48.691051	11
4	ChronosZeroShot[bolt_base]	-0.174142	-0.155696	1.788336	1.546005	15.988574	7
5	AutoETS	-0.175008	-0.164182	1.422188	1.681127	1.629197	6
6	DeepAR	-0.178739	-0.153764	0.398808	0.464099	43.161346	10
7	DynamicOptimizedTheta	-0.181408	-0.166902	1.464091	0.394216	1.834912	5
8	DirectTabular	-0.185328	-0.176227	0.298558	0.196042	23.262159	3
9	RecursiveTabular	-0.191510	-0.182156	0.058866	0.039506	1.326468	2
10	TiDE	-0.195750	-0.168350	0.673404	0.742906	137.473103	12
11	SeasonalNaive	-0.202541	-0.187587	6.037529	0.196710	6.047851	1
12	NPTS	-0.269840	-0.251826	0.293823	0.261489	0.327731	4

Scripts:

02\_autoluon.ipynb

02\_autoluon\_duckdb.ipynb **(Este es el archivo que genera el modelo ganador)**

## 2.3 Propeth.

Para la utilización de propeth se utilizó un modelo por cada product\_id, es decir se corrieron 780 modelos para obtener las predicciones a febrero 2020, con la siguiente configuración.

Se tomó un mínimo de 12 meses para correr el modelo para un producto, si no se disponía de los 12 meses, se salteaba la predicción. Para los casos donde la predicción de tn fue cero, o no se pudo realizar, se completaron las predicciones con las de autogluon.

Tambien se intentó una hacer un promedio y un mínimo combinando las predicciones de Prophet y AutoGluon.

### Scripts:

02\_prophet.py

complete\_prophet\_predictions.py

## 2.4 LSTM

Para la prueba de este modelo se realizó una predicción por cada serie de producto y cliente, para la predicción final se sumaron las toneladas de cada modelo y cliente y así se obtuvo la predicción final.

Este experimento demora casi 10 horas en correr. Para algunos productos no fue posible hacer la predicción por falta de series. En estos casos se utilizó la predicción de AutoGluon.

Se intentaron varias estrategias utilizando otras variables numéricas como regresores, las pruebas iniciales se hicieron un un set de pruebas de 1-2-3 y hasta 10 productos. Para la corrida final solo se dejó como regresor el promedio histórico del mes, es el que más disminuía el valor de las predicciones.

Configuraciones para la corrida:

```
'''
'batch_size': 128,
'max_parallel_models': 4,
# 'n_cores': 4,
'epochs': 50,
'sequence_length': 6, # Reducido para 6 meses de datos
'hidden_size': 64,
'num_layers': 2,
'learning_rate': 0.001
```

```
class LSTMModel(nn.Module):
    """Modelo LSTM con PyTorch"""
    def __init__(self, input_size, hidden_size, num_layers, output_size=1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            dropout=0.2 if num_layers > 1 else 0
        )

        self.dropout = nn.Dropout(0.2)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # Inicializar estados ocultos
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Forward pass LSTM
        out, _ = self.lstm(x, (h0, c0))

        # Tomar la última salida
        out = self.dropout(out[:, -1, :])
        out = self.fc(out)

        return out
```

Scripts:

02\_pytorch\_lstm\_script.py

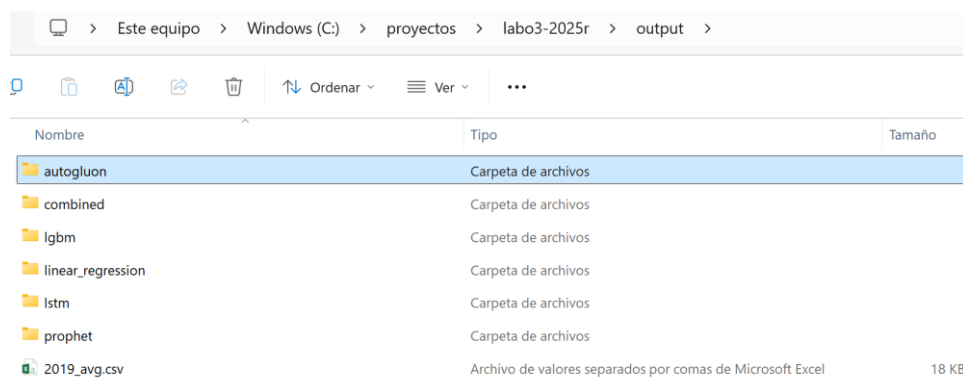
02\_pytorch\_lstm\_filtro\_780\_y\_alutogluon.py

## Procedimientos y Recopilación de Resultados

Para cada uno de los modelos, se realizó la predicción de los 780 productos, en los modelos donde no fue posible correr modelos por falta de series de tiempo (Prophet y LSTM) las predicciones faltantes se tomaron del script AutoGluon, hay scripts adicionales a los del modelo que se encaran de completar las predicciones.

Si bien la evaluación de los resultados se realizó en la plataforma Kaggle, previo a la subida a Kaggle, las predicciones se revisaron y compararon internamente, contra el promedio mensual y contra AutoGluon. Al no obtener valores razonables, algunos directamente se descartaron.

Los resultados de cada modelo se almacenan en la carpeta “output” donde hay una carpeta por cada uno de los modelos.



Nombre	Tipo	Tamaño
autogluon	Carpeta de archivos	
combined	Carpeta de archivos	
lgbm	Carpeta de archivos	
linear_regression	Carpeta de archivos	
lstm	Carpeta de archivos	
prophet	Carpeta de archivos	
2019_avg.csv	Archivo de valores separados por comas de Microsoft Excel	18 KB

Para los modelos en los que mejor resultado se observó en Kaggle, se utilizó un script que calculaba el promedio de las 2 predicciones o el valor mínimo. Los resultados de estas pruebas se guaran en la carpeta “combined” y se generan con el script “03\_combine.py”



## Análisis de los resultados.

Se observa que el modelo de AutoGluon fue superador respecto a al resto, algunos de los factores del éxito de este modelo radican en que se utilizan varios modelos diferentes, utiliza optimización de hiperparametros, cross-validation, ensamble y manejo de valores faltantes. Todo esto de manera automática.

El modelo es de muy simple implementación y de bajo costo computacional, ofreciendo los mejores resultados en el menor tiempo posible.

Modelo	Tiempo Entrenamiento	Kaggle Public	Complejidad	Kaggle
AutoGluon	20 minutos	0,242	Baja	predicciones_auto_numeric_WQL_val3_RF_Q10_mean.csv
LSTM PyTorch	10 horas	0,263	Alta	predicciones_combined_avg_20250810_084354.csv
LightGBM	< 1 hora	0,268	Alta	predicciones_productos_especificos_slr.csv
Promedio 2019	10 segundos	0,273	Muy Baja	2019_avg.csv
Prophet	10 minutos	0,304	Media	predicciones_prophet_autogluon_avg_20250810_093327.csv