

## Prueba técnica

**Proyecto:** Cementos Argos — Pronóstico de demanda y clasificación Alpha/Beta

**Autor:** Ricardo Moreno

**Repositorio:** <https://github.com/Ricardo-m-a/summa-sci-ml-test>

## Resumen ejecutivo

Se implementó una solución para (1) pronosticar la demanda de Cementos Argos y (2) clasificar registros como **Alpha** o **Beta** para apoyar decisiones de compra. La solución consta de: notebooks de análisis y entrenamiento, modelos serializados, un servicio REST (FastAPI) para inferencia funcional (backend).

---

### 1. Datos

- **dataset\_demand\_acumulate.csv:** demanda histórica mensual (2017-01 a 2022-04).  
Uso principal: construir y validar modelos de series de tiempo para pronosticar los meses 2022-05, 06 y 07.
- **dataset\_alpha\_betha.csv:** ~7000 registros con variables predictoras y etiqueta Class (Alpha/Beta). Uso: entrenamiento del clasificador.
- **to\_predict.csv:** 3 registros objetivo; contienen todas las variables excepto Demand y Class. Se completan con las predicciones obtenidas en el pipeline.

Limpieza básica aplicada en los notebooks:

- Conversión de Charges, Demand a numérico (pd.to\_numeric(..., errors='coerce')), limpieza de separadores de miles.
  - Mapeo de campos binarios (Yes/No, True/False, "1") a 1/0.
  - Tratamiento de faltantes vía SimpleImputer(strategy="median") en el pipeline de entrenamiento.
- 

### 2. Pronóstico de demanda (notebook: 02\_forecasting\_train\_test\_predict.ipynb)

#### Objetivo

Pronosticar la demanda para 2022-05, 2022-06 y 2022-07 y exportar un archivo con los pronósticos para completar to\_predict.csv.

## Modelos probados

Se exploraron varios enfoques clásicos y ML:

- **Modelos basados en series:** Prophet, SARIMAX/ARIMA (statsmodels).
- **Modelos supervisados para regresión:** XGBoost Regressor (XGBRegressor), RandomForest (regresor), SVR, regresión lineal.

## Selección final

Seleccioné **XGBoost** como modelo final de pronóstico y lo guardé en `../models/forecast/best_xgb_model.pkl`. XGBoost tras rendir mejor en los criterios de validación usados (RMSE / MAE / MAPE) tras la búsqueda de hiperparámetros y comparación con los modelos de series temporales.

## Preprocesamiento temporal

- Construcción de lags y ventanas móviles cuando aplica.
- Split temporal respetando la secuencia de tiempo (train / validation / test temporal).
- Métricas registradas: **RMSE, MAE, MAPE** (ver notebook / outputs).

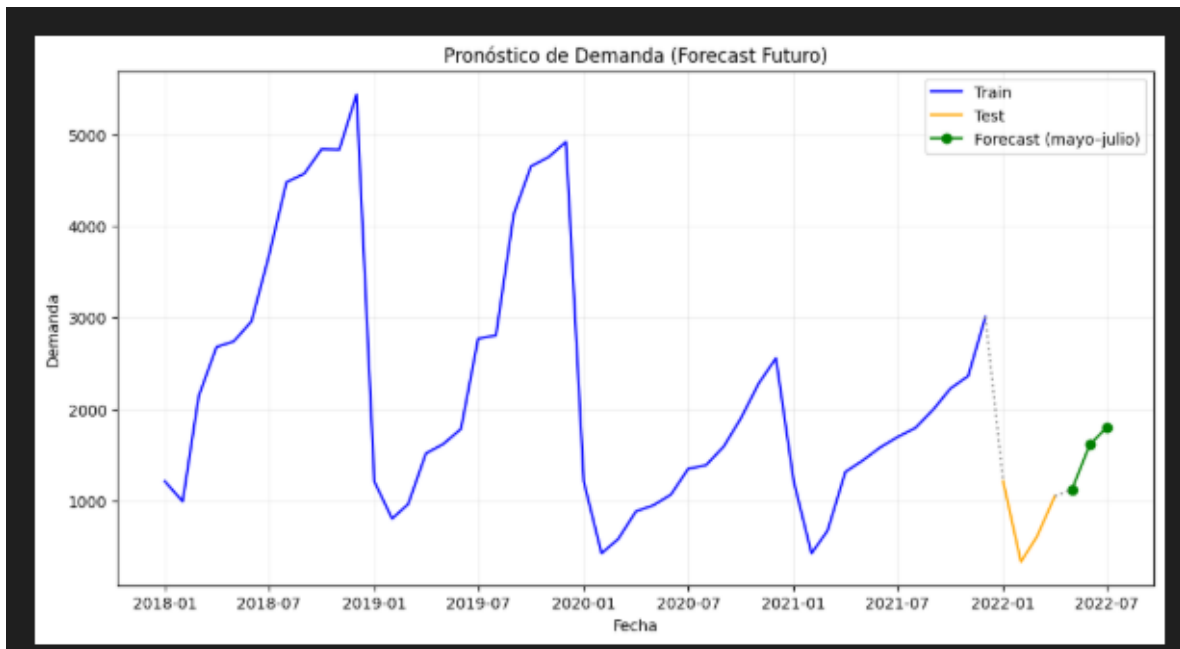
## Entregable

- Modelo guardado: `models/forecast/best_xgb_model.pkl`
- Output con los pronósticos y gráficos

Nota:

Si bien el gráfico de muestra, enviado en las indicaciones de la prueba, ubicaba la parte de test y predict en el mismo rango de tiempo, los separo tras la necesidad de simular un escenario real de incertidumbre (en el cual no podríamos hacer test en ese mismo rango, dado que no se tienen esos datos).

La elección del modelo de serie de tiempo se escogió por: ser mejor en términos relativos a los que se evaluaron con él; y, adecuarse mejor al fenómeno univariado (no tenemos features exógenas que expliquen la demanda, solo la demanda misma), si tuvieramos más características valdría la pena explorar ML no paramétrico, pues con solo una variable el  $r^2$  tiene valores buenos (no excelentes). En todo caso, es interesante notar que un 20% de la varianza, aprox., no la explica la demanda misma



### 3. Clasificación Alpha/Beta (notebook: 03\_class\_train\_test\_inference.ipynb)

#### Objetivo

Entrenar un clasificador que, dado el conjunto de variables (incluida la demanda — real o pronosticada), devuelva la etiqueta **Alpha** o **Beta** que indique el tipo de compra a realizar.

#### Pipeline de entrenamiento (resumen)

- **Feature engineering:**
  - `create_features(df)`: genera interacciones numéricas (ratios  $x_{div}y$  y productos  $x_{x_y}$ ) entre columnas numéricas.
  - Agregados por categoría: media y desviación estándar por categoría (`mean_by_{cat}`, `std_by_{cat}`) — calculados en entrenamiento y almacenados para inferencia.
- **Preprocesamiento (pipeline):**
  - `ColumnTransformer`:
    - Numéricas: `SimpleImputer(strategy='median') + StandardScaler()`.
    - Categóricas: `SimpleImputer(constant='missing') + OneHotEncoder(handle_unknown='ignore', sparse_output=False, max_categories=10)`.

- **Selección:** `SelectKBest(f_classif, k=min(50, n_features))`.
- **Modelos evaluados:** `RandomForestClassifier`, `XGBClassifier`, `LogisticRegression` (solver saga), con `class_weight="balanced"` cuando aplica.
- **Búsqueda de hiperparámetros:** `RandomizedSearchCV` con `StratifiedKFold(n_splits=5)` y scoring por `f1_macro`.
- **Ensemble final:**
  - Se seleccionaron los top-2 modelos según `test_f1_macro`.
  - `VotingClassifier(voting='soft')` + calibración de probabilidades con `CalibratedClassifierCV(method='isotonic')`.
  - Optimización de umbral (`optimize_threshold`) para mejorar F1 en la clase positiva (Beta).
- **Guardado de artefactos:**
  - `models/classifier/ensemble_model.pkl`
  - `models/classifier/label_encoder.pkl`
  - `models/classifier/optimal_threshold.pkl`
  - `models/classifier/feature_cols.pkl`
  - `models/classifier/numeric_cols.pkl`
  - `models/classifier/group_stats_map.pkl`

### Métricas y validación

- Validación con 5-fold estratificado durante el CV; holdout test 20% estratificado.
- Métricas finales calculadas: `accuracy`, `f1_macro`, `f1_beta` (F1 en la clase Beta), `auc`, `sensitivity` (recall Beta), `specificity`.
- Reportes y gráficas (confusion matrix, ROC) guardadas en `output/` (ej.: `confusion_matrix_ensemble.png`, `roc_curve_ensemble.png`) y métricas en `output/final_metrics.json` y `model_comparison.csv`.

---

### 4. Artefactos generados (ruta local del repo)

- `models/forecast/best_xgb_model.pkl` — modelo de forecasting seleccionado.

- `models/classifier/ensemble_model.pkl` — modelo ensemble calibrado para clasificación.
  - `models/classifier/label_encoder.pkl` — encoder de clases (Alpha/Beta).
  - `models/classifier/optimal_threshold.pkl` — diccionario `{'threshold': value}` usado en inferencia.
  - `models/classifier/feature_cols.pkl` — lista de columnas utilizadas por el modelo (orden crítico).
  - `models/classifier/numeric_cols.pkl` — lista de columnas numéricas (útil para coerción en la API).
  - `models/classifier/group_stats_map.pkl` — map de agregados por grupo (mean/std) calculados en train (se usa en inferencia para rellenar las columnas agregadas).
  - `output/` — métricas, plots y CSVs generados durante los notebooks (`final_metrics.json`, `model_comparison.csv`, `predictions_optimized.csv`, `confusion_matrix_ensemble.png`, `roc_curve_ensemble.png`, etc.)
- 

## 5. API y consumo (archivo `api/main.py`)

Se implementó una API con **FastAPI** en `api/main.py` que expone:

- `GET /` — health check.
- `POST /predict` — recibe JSON con las variables base (tal cual vienen del CSV) y devuelve:
  - `prediction` (Alpha/Beta),
  - `confidence` (score probabilístico transformado a confianza).

Flujo interno:

Convierte el JSON a DataFrame.

Aplica mapeos Yes/No → 1/0 para las columnas binarias guardadas en `mapped_cols.pkl`.

Limpia `Charges/Demand` (comma thousands) y fuerza tipos numéricos para las columnas guardadas en `numeric_cols.pkl`.

Ejecuta `create_features` (interacciones).

Rellena los agregados por grupo (`mean_by_*`, `std_by_*`) con `group_stats_map.pkl`.

Reindexa a `feature_cols.pkl` (orden y faltantes como NaN).

Llama `ensemble_model.predict_proba(...)`, aplica `optimal_threshold` y devuelve la etiqueta y la confianza.

Ejemplo de request (raw tal cual CSV — la API mapea internamente):

```
{
  "autoID": "9695-TERGH",
  "SeniorCity": 0,
  "Partner": "No",
  "Dependents": "No",
  "Service1": "Yes",
  "Service2": "No",
  "Security": "Yes",
  "OnlineBackup": "No",
  "DeviceProtection": "No",
  "TechSupport": "No",
  "Contract": "Month-to-month",
  "PaperlessBilling": "Yes",
  "PaymentMethod": "Electronic check",
  "Charges": "96.05",
  "Demand": "1124.42724609375"
}
```

Evidencia de prueba de API:

The screenshot shows the Swagger UI for the 'Cementos Argos Demand API' (version 0.1.0, OAS 3.1). The 'default' section is expanded, showing the 'POST /predict' endpoint. The 'Parameters' section is empty. The 'Request body' section is highlighted with a green box and contains a JSON object with the following fields: 'autoID' (string), 'SeniorCity' (integer), 'Partner' (string), 'Dependents' (string), 'Service1' (string), 'Service2' (string), 'Security' (string), 'OnlineBackup' (string), 'DeviceProtection' (string), 'TechSupport' (string), 'Contract' (string), 'PaperlessBilling' (string), 'PaymentMethod' (string), 'Charges' (string), and 'Demand' (string). The 'Execute' button is visible at the bottom of the interface.

Execute

Clear

Responses


Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "autoID": "5731-00308",
    "SeniorCity": 0,
    "Partner": "No",
    "Dependents": "No",
    "Service1": "Yes",
    "Service2": "No",
    "Security": "No internet service",
    "OnlineBackup": "No internet service",
    "DeviceProtection": "No internet service",
    "TechSupport": "No internet service",
    "Contract": "Two year",
    "PaperlessBilling": "No",
    "PaymentMethod": "Bank transfer (automatic)",
    "Charges": "19.7",
    "Demand": "1,004.23"
  }'
```

Request URL

http://127.0.0.1:8000/predict

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "prediction": "ALPHA",   "confidence": 0.9438958379050545 }</pre></div><div> <a href="#">Download</a></div></div> <div><pre>content-length: 54 content-type: application/json date: Wed, 03 Sep 2025 03:03:26 GMT server: uvicorn</pre></div>

Responses

Code	Description	Links
------	-------------	-------