Modelización de Series Temporales en Python

Alberto García-Galindo

Instituto de Investigación Biomédica de Salamanca (IBSAL)

PyCon ES ALICANTE 2019

Introducción

Describir la evolución de una variable estocástica a lo largo del tiempo y la modelización de dicho comportamiento delimitan todo un subcampo de la Estadística que se conoce como análisis de series temporales. Para este propósito, existen una gran cantidad de herramientas y paquetes estadísticos capaces de desempeñar esta tarea, y Python es una de ellas. El objetivo de esta ponencia es introducir al lector a los conceptos básicos de series temporales y las posibilidades que dicho lenguaje ofrece para trabajar y modelizar este tipo de datos, ya sea mediante técnicas clásicas basadas en modelos lineales como basadas en aprendizaje automático.

Almacenamiento de series temporales

pandas

La manera óptima de almacenar series temporales es a través de la clase `pandas.Series`, que permite indexar con frecuencias temporales nuestras observaciones.

```
> my_serie = read_csv("wolf_river.csv", sep = ";")
> my_serie["index"] = to_datetime(my_serie["month"])
> my_serie.set_index("index", inplace = True)
> my_serie.drop(["month"], axis = 1, inplace = True)
```

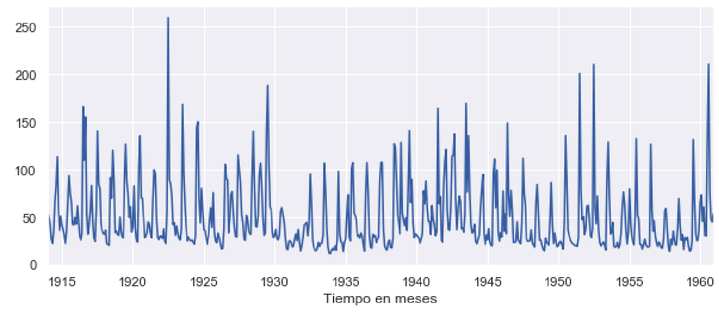

Figura 1. Nivel del río Wolf a su paso por New London (serie original)

Transformaciones

pandas, scipy.stats

Normalmente, los modelos estadísticos asumen que la serie temporal es estacionaria tanto en media como en varianza. En caso contrario, existen transformaciones que permiten corregir dicho comportamiento.

```
> serie_box = boxcox(my_serie) # Transformación de Box-Cox para estabilizar la varianza
> serie_diff = my_serie.diff(ndiff).iloc[ndiff:] # Diferenciación para corregir la tendencia
```

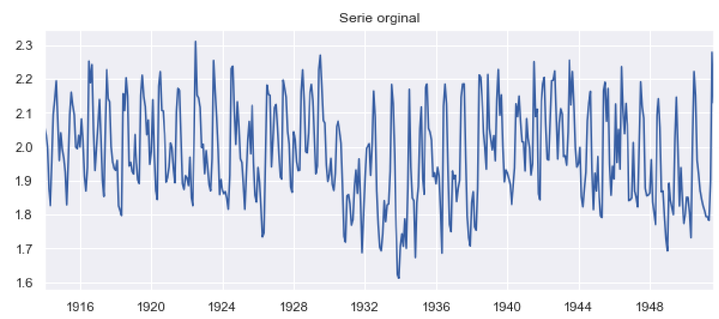

Figura 2. Serie bajo la transformación de Box-Cox ($\lambda = -0.3807$)

Descomposición clásica

statsmodels.tsa.seasonal

De forma general una serie temporal puede descomponerse para facilitar su comprensión y en algunos casos mejorar los resultados de ajuste y predicción. Se muestra útil especialmente para eliminar la posible estacionalidad que pueda presentarse en la serie.

```
> components = seasonal_decompose(serie_box, "add") # Descomposición aditiva
> trend, seasonal = components.trend, components.seasonal
> serie_deseas = serie_box - seasonal # Serie desestacionalizada
```

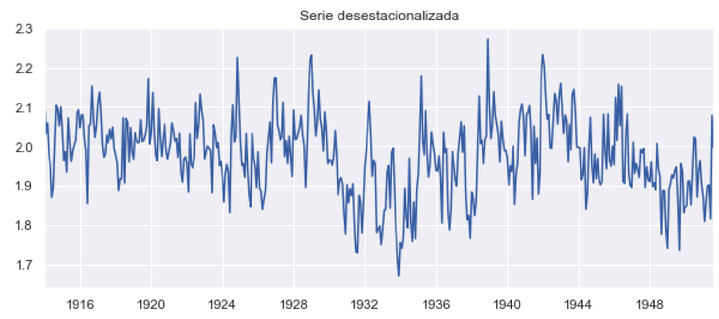

Figura 3. Serie desestacionalizada mediante descomposición clásica

Suavizado exponencial

statsmodels.tsa.holtwinters

Constituyen toda una familia de métodos que se fundamentan en la ponderación de las observaciones pasadas de la serie de manera que los pesos de las observaciones decaen exponencialmente hacia el pasado.

```
> ses_model = ExponentialSmoothing(serie_box) # Suavizado Exponencial Simple
> holt_model = ExponentialSmoothing(serie_box, trend = "add") # Método de la tendencia lineal de Holt
> hw_model = ExponentialSmoothing(serie_box, trend = "add", seasonal = "add", seasonal_periods = 12) # Método de Holt Winters
```

Modelos SARIMA

statsmodels.graphics, statsmodels.tsa.statespace.sarimax, pmdarima.arima, statsmodels.stats.diagnostic, tstoolbox

Parten del supuesto de que la serie temporal esta generada, quizás tras algún tipo de transformación, por un proceso estocástico estacionario determinado. Abarcan desde los modelos más simples hasta modelos muchos más complejos capaces de trabajar sobre series tendencias complejas y patrones estacionales.

```
> tsaplots.plot_acf(serie_box, lags = nlags) # Función de autocorrelación simple
> tsaplots.plot_pacf(serie_box, lags = nlags) # Función de autocorrelación parcial
> sarima_model = SARIMAX(serie_box, order = (p, d, q), seasonal_order = (P, D, Q, 12)).fit()
> sarima_model.summary() # Resumen del modelo
> autosarima_model = auto_arima(serie_box, seasonal = True, m = 12) # Identificación automática
```

Para contrastar si nuestro modelo es correcto y no presencia una falta de ajuste, podemos proceder a realizar un diagnóstico de los residuales.

```
> sarima_model = SARIMAX(serie_box, order = (1, 0, 0), seasonal_order = (1, 0, 0, 12).fit()
> sarima_model.resid # Residuales del modelo
> lb_test = acorr_ljungbox[1][6] # P-valor del test de Ljung-Box para los 6 primeros retardos
> resid_diag(sarima_model.resid)
```

Una vez definido el modelo, podemos proceder a realizar las predicciones.

```
> forecasts = sarima_model.forecast(steps = 7) # Predicción para los siguientes 7 instantes
```

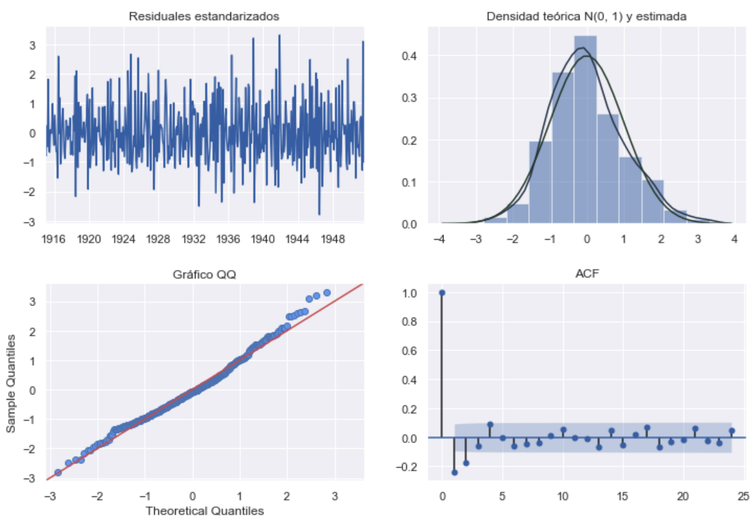

Figura 4. Diagnóstico de residuales para el modelo SARIMA (1, 0, 0) x (1, 0, 0)

Machine Learning

tstoolbox, sklearn.ensemble, sklearn.metrics

Los algoritmos de aprendizaje supervisado pueden utilizarse para modelar las dependencias que presente la serie temporal. No obstante, para ello es necesario considerar los retardos de la serie temporal como variables predictoras. Además, es necesario entrenar el algoritmo con las observaciones hasta un determinado instante y validar el modelo resultante con las observaciones desde ese mismo instante.

```
> train = ts_split(my_serie)[0] # Período de entrenamiento de algoritmos
> test = ts_split(my_serie)[1] # Período de validación de modelos
> serie_df = series_to_supervised(train, n_in = 12) # DataFrame con retardos como predictoras
> X_train, y_train = train.iloc[:,:-1], train.iloc[:,-1]
> rf_model = RandomForest().fit(X_train, y_train) # Entrenamiento de RandomForest
> rf_forecasts = iter_forecast(rf_model, y_train, 12, test) # Predicción iterativa
> rf_rmse = sqrt(mse(test, rf_forecasts)) # Evaluación con Error Cuadrático Medio
> rf_mae = mae(test, rf_forecasts) # Evaluación con Error Absoluto Medio
```

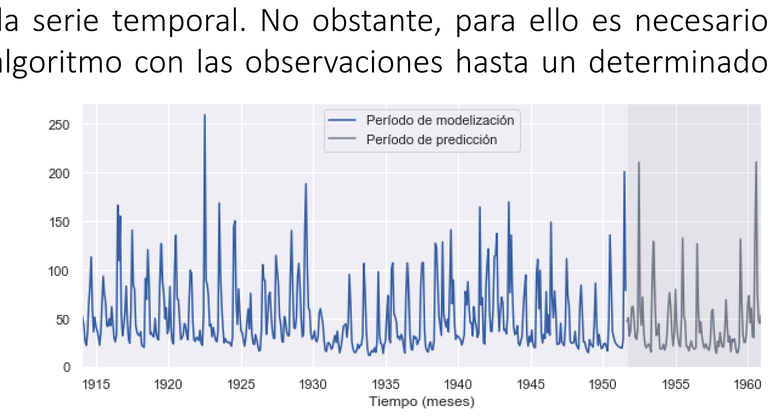

Figura 5. División temporal de la serie en período de modelización y predicción

Una explicación más detallada, con mayor fundamento matemático, metodologías de predicción, comparaciones tanto teóricas como prácticas y referencias podrás encontrarlas el repositorio de mi Trabajo Fin de Grado: **https://github.com/Alberto267/TFG**