

Un modelo de mezcla de marketing mejorado en Python

Hacer que mi modelo de mezcla de marketing mediocre sea mucho más poderoso



Dr. Robert Kübler · 23 de sep · 13 min de lectura ★



Foto de [Peter Nguyen](#) en [Unsplash](#)

Escena retrospectiva

En mi último artículo, les presenté el mundo del modelado de mezclas de marketing. Si no lo ha leído hasta ahora, hágalo antes de continuar.

Introducción al modelado de mezcla de marketing en Python

¿Qué inversión publicitaria está impulsando realmente sus ventas?

Allí, hemos creado un modelo de **regresión lineal** que es capaz de predecir las ventas en función de los gastos de publicidad en bruto en varios canales de publicidad, como TV, radio, banners web. Para mí, como practicante del aprendizaje automático, este modelo ya es bueno por sí solo. Aún mejor, también hace felices a los empresarios porque el modelo nos permite calcular el ROI, lo que nos permite juzgar qué tan bien se desempeñó cada canal.

¿Cómo e ver, nuestro sencillo modelo de regresión lineal a partir de la última vez tenía algunos **problemas que vamos a resolver en este artículo**. Deja parafrasearlos por ti:

1. **El rendimiento de nuestro primer modelo podría ser mejor.**
2. **Nuestro primer modelo se comporta de manera poco realista. Aumentar los gastos hasta el infinito también aumenta las ventas hasta el infinito, lo que no tiene sentido porque las personas solo pueden gastar una cantidad finita de dinero en nuestro producto.**
3. **La optimización se vuelve trivial, inútil y poco realista también. Para maximizar las ventas con un presupuesto fijo, ahora pondríamos todo el dinero en el canal con el coeficiente de regresión lineal más alto.**

Arreglando los problemas

Para evitar estos problemas, podemos realizar una **ingeniería de funciones inteligente** que nos permita incorporar algunos conocimientos del dominio de marketing en el modelo. No se preocupe, no se requiere experiencia en marketing para comprender estos conceptos; son bastante naturales y, por lo tanto, fáciles de entender. Las siguientes técnicas mejorarán el rendimiento y harán que el modelo sea más realista.

Publicidad Adstock

Esta ingeniería de funciones que haremos es un componente crucial llamado **publicidad publicitaria**, un término acuñado por Simon Broadbent [1]. Es una palabra elegante que encapsula dos conceptos simples:

1. Suponemos que cuanto más dinero gaste en publicidad, mayores serán sus ventas. Sin embargo, el aumento se debilita cuanto más gastamos. Por ejemplo, aumentar el gasto en TV de 0 € a 100.000 € aumenta mucho nuestras ventas, pero aumentarlo de 100.000.000 € a 100.100.000 € ya no hace mucho. Esto se llama **efecto de saturación o la efecto de rendimientos decrecientes**.
2. Si gasta dinero en la semana de publicidad T , a menudo la gente no comprará su producto inmediatamente, sino unas pocas (digamos x) semanas después. Esto se debe a que el producto puede ser caro y la gente quiere pensarlo detenidamente o compararlo con productos similares de otras empresas. Cualquiera que sea la razón, la venta en la semana $T + x$ se debe en parte a la publicidad que jugó en la semana T , por lo que también debería obtener algunos créditos. A esto se le llama **efecto de arrastre o retraso**.

Creo que ambos hechos son bastante fáciles de entender, y la gente de negocios también los investiga.

Nuestro nuevo modelo sigue siendo lineal, pero con características de anuncios publicitarios en lugar de gastos sin procesar como entrada. Esto hace que el modelo sea mucho más fuerte y al mismo tiempo lo mantiene interpretable.

Building Saturation and
Carry-Over Effects in
scikit-learn

Creación de efectos de saturación y arrastre en scikit-learn

Desafortunadamente para nosotros, scikit-learn no contiene estas transformaciones porque no son de interés para todas las industrias. Pero dado que ambas transformaciones no son tan complicadas, es una buena oportunidad para practicar la escritura de estimadores compatibles con scikit-learn.

Si nunca lo ha hecho antes, puede consultar mi otro artículo sobre ese tema. El artículo trata sobre regresores en lugar de transformadores, pero el enfoque no es muy diferente.

Escriba un regresor y benefíciase de todas las increíbles herramientas del ecosistema scikit-learn, como canalizaciones,...

haciadatascience.com

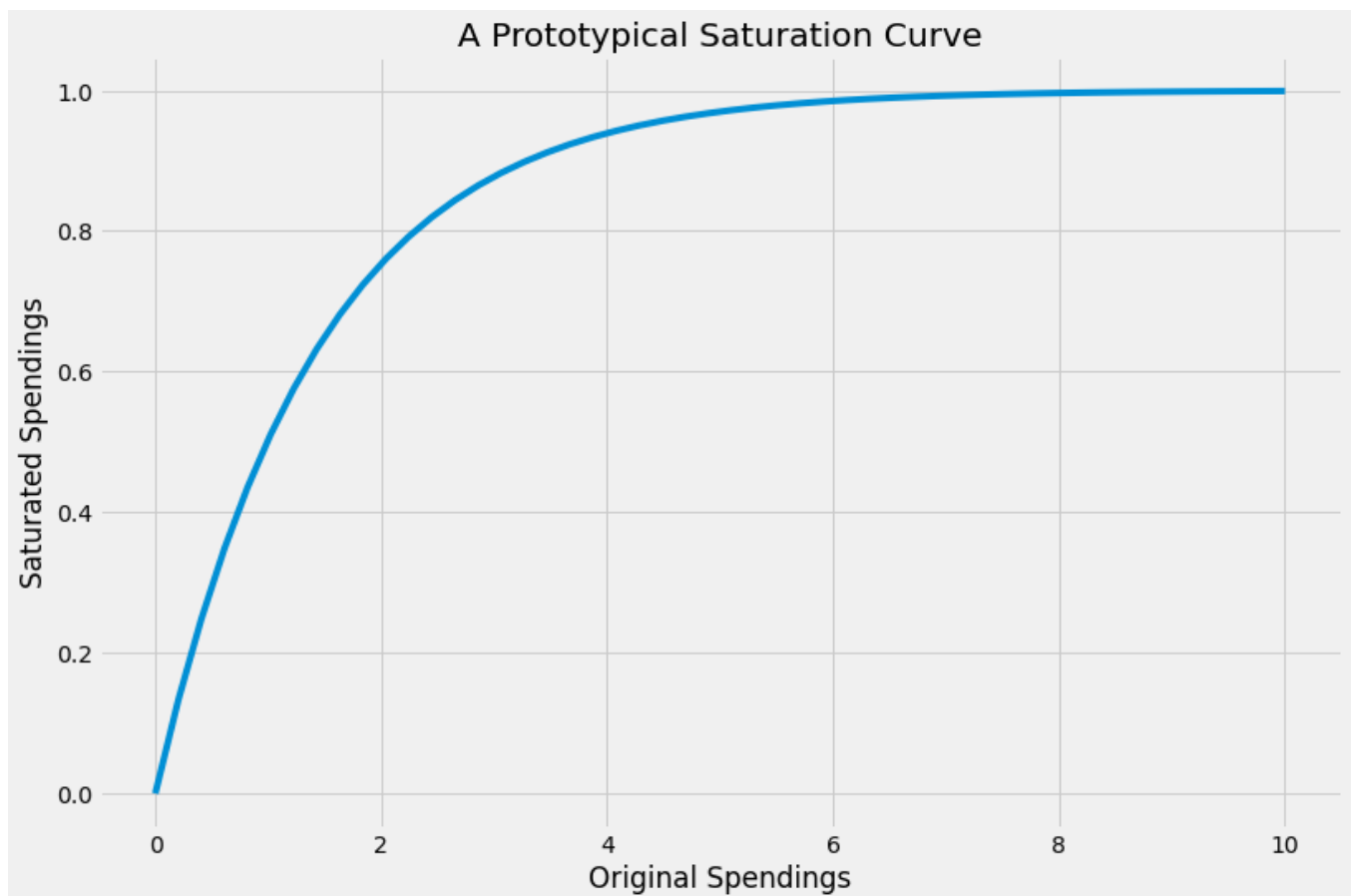
Entonces, comencemos con el más fácil: **el efecto de saturación**.

Creando un efecto de saturación

Queremos crear una transformación (= función matemática) con las siguientes propiedades:

1. Si los gastos son 0, los *gastos saturados* también son 0.
2. La transformación aumenta de manera monótona, es decir, cuanto mayor es el gasto en insumos, mayor es el gasto en producción saturada.
3. Los valores saturados no crecen hasta el infinito. En cambio, están delimitados en la parte superior por algún número, digamos 1.

En resumen, queremos algo como esto:



Hay muchas formas de obtener funciones como esta, en la imagen se ve la función $1 - \exp(-0,7x)$, por ejemplo. Así que usemos esta plantilla de función y generalicémosla a $1 - \exp(-ax)$ para algunos $a > 0$. a es un hiperparámetro que podemos ajustar después porque, por lo general, no conocemos la forma de la función de saturación.

Nota: Hay muchas funciones saturaciones más estándar, como los **Adbudg** y **colina de** funciones, ipero nos quedamos con la **exponencial de** la función para la simplicidad.

Un buen efecto secundario: podemos generar las curvas de saturación al final, por lo que sabemos si tiene sentido gastar más dinero o si *el canal ya está saturado*. De la imagen de arriba, por ejemplo, parece que invertir más de 8 es inútil.

Entonces, codifiquemos esto. De hecho, es solo esta pequeña clase simple:

```
class ExponentialSaturation:
    def __init__(self, a = 1.):
        self.a = a

    def transform(self, X):
        return 1 - np.exp(-self.a * X)
```

Sin embargo, agregaremos algunas verificaciones de cordura para las entradas para que sean compatibles con scikit-learn. Esto aumenta un poco el código, pero es un precio comparativamente pequeño que tenemos que pagar.

```
de sklearn.base importar BaseEstimator, TransformerMixin
de sklearn.utils.validation importar check_is_fitted, check_array

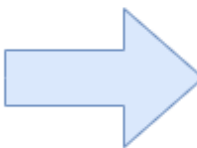
class ExponentialSaturation (BaseEstimator, TransformerMixin):
    def __init__(self, a = 1.):
        self.a = a

    def fit (self, X, y = None):
        X = check_array (X)
        self._check_n_features (X, reset = Verdadero) # de
BaseEstimator
```

volver a sí mismo

```
def transform (self, X):  
    check_is_fitted (self)  
    X = check_array (X)  
    self._check_n_features (X, reset = False) # de BaseEstimator  
  
    return 1 - np.exp (-self.a * X)
```

Todavía no es perfecto porque también deberíamos implementar una verificación para *un* ser mayor que cero, pero esto es algo que puede hacer fácilmente por su cuenta. Con el `ExponentialSaturation` transformador podemos hacer lo siguiente:



Spends	
0	100
1	50
2	30
3	120
4	9001
5	5
6	0
7	80

Saturated Spends	
0	0.632121
1	0.393469
2	0.259182
3	0.698806
4	1.000000
5	0.048771
6	0.000000
7	0.550671

Imagen del autor.

Este no estuvo tan mal, ¿verdad? Pasemos ahora al siguiente efecto.

Creación de un efecto de arrastre

Este es un poco más complicado. Permítanme usar un ejemplo para mostrarles lo que queremos lograr. Se nos da una serie de gastos a lo largo del tiempo, como

(16, 0, 0, 0, 0, 4, 8, 0, 0, 0),

lo que significa que gastamos 16 en la primera semana, luego no gastamos nada de la semana 2 a la 5, luego gastamos 4 en la semana 6, etc.

Ahora queremos que los gastos de una semana se transfieran parcialmente a las próximas semanas de forma **exponencial** . Esto significa: En la semana 1 hay un gasto de 16. Luego, llevamos más del 50%, lo que significa

- $0.5 * 16 = 8$ a la semana 2,
- $0.5^2 * 16 = 4$ a la semana 3,
- $0.5^3 * 16 = 2$ a la semana 4,
- ...

Esto introduce dos hiperparámetros: la **fuerza** (¿cuánto se transfiere?) Y la **duración** (¿cuánto tiempo se transfiere?) Del efecto de transferencia. Si usamos una **fuerza del 50%** y una **longitud de 2** , la secuencia de gasto de arriba se convierte en

(16, 8, 4, 0, 0, 4, 10, 5, 2, 0).

Creo que puede escribir algunos bucles para implementar este comportamiento, aunque una forma agradable y rápida es usar convoluciones. No lo explicaré en detalle, así que tome el código como un regalo. Destaqué de nuevo las líneas realmente importantes.

```
de scipy.signal import convolve2d
import numpy as np
```

```
class ExponentialCarryover (BaseEstimator, TransformerMixin):
    def __init__ (self, fuerza = 0.5, longitud = 1):
        self.strength = fuerza
        self.length = longitud

    def fit (self, X, y = None):
        X = check_array (X)
        self._check_n_features (X, reset = True)
        self.sliding_window_ = (
            self.strength ** np.arange (self.length + 1)
) . remodelar (-1, 1)

    volver a sí mismo
```

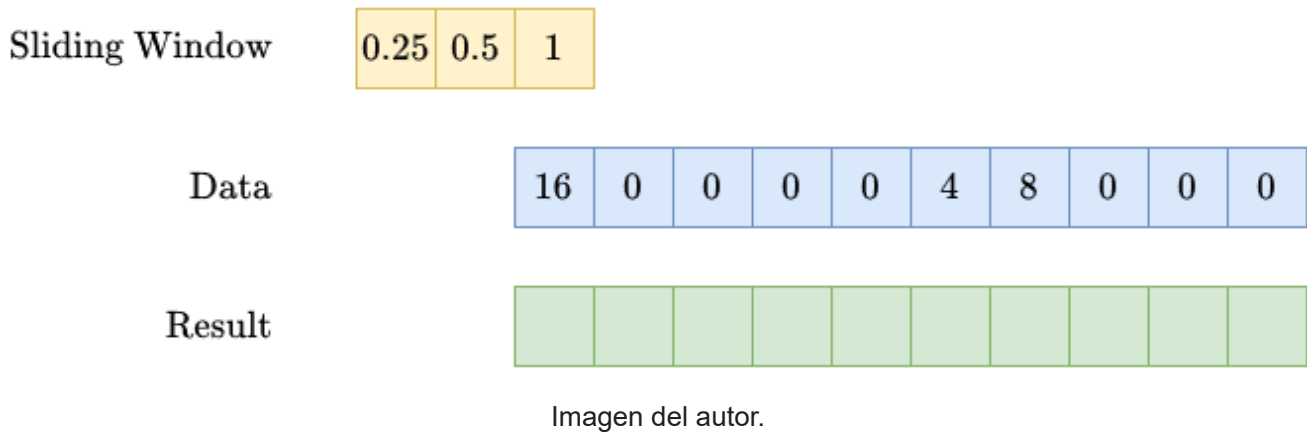
```
def transform (self, X: np.ndarray):
    check_is_fitted (self)
    X = check_array (X)
    self._check_n_features (X, reset = False)

    convolución = convolve2d (X, self.sliding_window_)

    if self.length> 0:
        convolution = convolution [: -self.length]

    vuelta convolución
```

Puedes ver que la clase toma fuerza y duración. Durante el ajuste, crea una ventana deslizante que es utilizada por la función convolve2d, haciendo mágicamente lo que queremos. Si conoce las capas convolucionales de las CNN, esto es exactamente lo que sucede aquí. Gráficamente, hace lo siguiente:



Nota: También hay muchas más formas de crear un remanente. La caída no tiene por qué ser exponencial. Y tal vez el pico del efecto publicitario no se alcance el día en que se gastó el dinero, sino siempre la semana siguiente. Puede expresar todas estas variaciones cambiando la ventana deslizante en consecuencia.

Combinemos los efectos de saturación y arrastre para crear un modelo de mezcla de marketing más realista.

El modelo final

Usaremos diferentes saturaciones y transferencias para cada canal. Esto tiene sentido porque, por lo general, un anuncio de televisión se queda más tiempo en su cabeza

que un banner que ve en línea, por ejemplo. Desde una perspectiva de alto nivel, el modelo se verá así:

The Model Pipeline

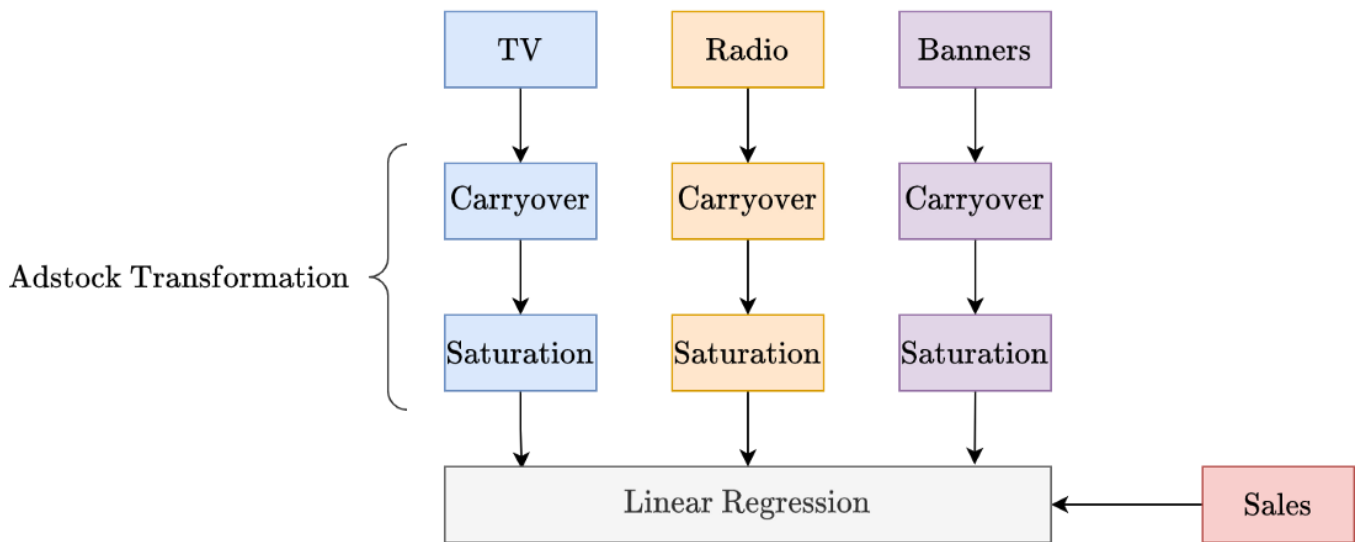


Imagen del autor.

Tenga en cuenta que la tubería azul es solo una función de los gastos de TV, la naranja de los gastos de radio y la púrpura de los gastos de pancartas. Podemos implementar esto de manera eficiente en scikit-learn usando las clases `ColumnTransformer` y `Pipeline`. El `ColumnTransformer` nos permite utilizar una transformación diferente para cada canal de anuncio mientras la `Pipeline` nos permite operaciones de la cadena para un solo canal. Tómese un segundo para comprender el siguiente fragmento:

```
de sklearn.compose importar ColumnTransformer
de sklearn.pipeline importar Pipeline
de sklearn.linear_model importar LinearRegression

adstock = ColumnTransformer (
    [
        ('tv_pipe', Pipeline ([
            ('carryover', ExponentialCarryover ()),
            ('saturation', ExponentialSaturation ())
        ]), ['TV']),
        ('radio_pipe', Pipeline ([
```

```

                ('carryover', ExponentialCarryover ()),
                ('saturation', ExponentialSaturation ())
            ]), ['Radio']],
            ('banners_pipe', Pipeline ([
                ('carryover', ExponentialCarryover ()),
                ('saturación ', ExponentialSaturation ())
            ]), [' Banners '])),
        ]
    )

model = Pipeline ([
    ('adstock', adstock),
    ('regression', LinearRegression ())
])

```

La parte más difícil es el `ColumnTransformer` , así que déjame explicarte cómo puedes leer el **bloque de TV** , marcado en negrita. Solo dice:

Aplique la tubería a la columna 'TV' y nombre esta parte 'tv_pipe'. La canalización es solo la transformación de anuncios publicitarios.

Allí no pasa nada más. Y al final, usamos este gran paso de preprocesamiento y agregamos un simple `LinearRegression` al final para tener un regresor real. Entonces, **carguemos los datos** nuevamente y hagamos un poco de entrenamiento.

```

importar pandas como pd
desde sklearn.model_selection importar cross_val_score,
TimeSeriesSplit

data = pd.read_csv (
    '
https://raw.githubusercontent.com/Garve/datasets/4576d323bf2b66c906d513c
    ',
    parse_dates = ['Fecha'],
    index_col = 'Fecha'
)

X = data.drop (columnas = ['Ventas'])
y = datos ['Ventas']

```

```
model.fit (X, y)

print (cross_val_score (modelo, X, y, cv = TimeSeriesSplit ()). mean
())

# Salida: ~ 0.55
```

Nota: No usamos la validación cruzada estándar k - veces aquí porque estamos tratando con datos de series de tiempo. `TimeSeriesSplit` es algo más razonable, y puede leer más al respecto [aquí](#).

¡Funciona! Sin embargo, el modelo sigue siendo bastante mala con una cruz-validado r^2 de alrededor de **0,55**, mientras que la edad, el modelo más simple tuvo una de 0,72. Esto se debe a que usamos los parámetros predeterminados y, por lo tanto, no óptimos para cada canal, a saber, $\alpha = 1$ para la saturación y una fuerza de arrastre de 0.5, y una longitud de 2.

Por lo tanto, sintonicemos todos los parámetros de anuncios publicitarios.

Ajuste de hiperparámetros

Voy a utilizar Optuna, una biblioteca avanzada para tareas de optimización. Entre muchas otras cosas, ofrece una `OptunaSearchCV` clase compatible con scikit-learn que puede ver como un reemplazo directo de scikit-learn `GridSearchCV` y `RandomizedSearchCV`.

En pocas palabras, `OptunaSearchCV` es una versión mucho más inteligente de `RandomizedSearchCV`. Mientras `RandomizedSearchCV` camina solo al azar, `OptunaSearchCV` camina de manera aleatoria al principio, pero luego verifica las combinaciones de hiperparámetros que parecen más prometedoras.

Consulte el código que se parece bastante a lo que está acostumbrado a escribir en scikit-learn:

```
de optuna.integration importar OptunaSearchCV
de optuna.distributions importar UniformDistribution,
IntUniformDistribution
```

```

tuned_model = OptunaSearchCV (
    estimador = modelo,
    param_distributions = {
        'adstock__tv_pipe__carryover__strength': UniformDistribution
(0, 1),
        'adstock__tv_pipe__carryover__length': IntUniformDistribution
(0, 6),
        'adstock__tv_pipe__saturation__a': UniformDistribution (0,
0,01),
        'adstock__radio_pipe__carryover__strength':
UniformDistribution ( 0, 1),
        'adstock__radio_pipe__carryover__length':
IntUniformDistribution (0, 6),
        'adstock__radio_pipe__saturation__a': UniformDistribution (0,
0,01),
        'adstock__banners_pipe__carryover__strength':
UniformDistribution (0, 1),

        'adstock__banners_pipe__carryover__length':IntUniformDistribution (0,
6),
        'adstock__banners_pipe__saturation__a': UniformDistribution
(0, 0.01),
    },
    n_trials = 1000,
    cv = TimeSeriesSplit (),
    random_state = 0
)

```

Le dice a Optuna que optimice `modelo`. Lo hace utilizando todos los parámetros que especifique en `param_distributions`. Debido a que nuestro modelo está bastante anidado, es decir, hay tuberías en un transformador de columna que está en una tubería nuevamente, tenemos que especificar exactamente qué hiperparámetro queremos ajustar. Esto se hace mediante cadenas

como `adstock__tv_pipe__carryover__strength`, donde **dos guiones bajos** separan diferentes niveles del modelo completo. A encontrar las palabras `adstock`, `tv_pipe`, `carryover` todo en la especificación del modelo, mientras que `strength` es un parámetro del `ExponentialCarryover` transformador.

Luego, encuentra algunas distribuciones. `UniformDistribution(0, 1)` significa que debe buscar parámetros **flotantes** entre 0 y 1. Por la misma lógica, `IntUniformDistribution(0, 6)` busca valores **enteros** entre 0 y 6 (ino 5!), por lo que le decimos al modelo que solo considere la longitud de arrastre menor o igual a seis semanas, lo que es solo una elección que hacemos.

Intentamos `n_trials=1000` diferentes combinaciones de parámetros y evaluamos usando a `de TimeSeriesSplit` nuevo. **Mantenga los resultados reproducibles configurando `random_state=0`.** ¡Hecho! Esto debería ser suficiente para comprender el código.

Comprobación de rendimiento

Comprobemos el rendimiento utilizando este modelo optimizado denominado `tuned_model`. **Tenga cuidado, esto lleva mucho tiempo.** Puede reducir el valor `n_trials` a 100 para obtener una solución peor mucho más rápido.

```
print (cross_val_score (tuned_model, X, y, cv = TimeSeriesSplit ()))  
  
# Salida: matriz ([0.847353, 0.920507, 0.708728, 0.943805, 0.908159])
```

La cruz-validado media r^2 es **0,87**, que es una gran mejora en comparación con el modelo no optimizado (0,55) y nuestro viejo modelo lineal normal desde el último artículo (0,72). Repasemos ahora el modelo y veamos qué ha aprendido.

```
tuned_model.fit (X, y)
```

Los hiperparámetros óptimos son los siguientes:

```
print (tuned_model.best_params_)  
print (tuned_model.best_estimator_.named_steps ['regression']. coef_)  
print (tuned_model.best_estimator_.named_steps ['regression'].  
intercept_)  
  
# Salida:  
# hiperparámetros = {  
# 'adstock__tv_pipe__carryover__strength': ,5248878517291329  
# 'adstock__tv_pipe__carryover__length': 4  
# 'adstock__tv_pipe__saturation__a': 1.4649722346562529e-05  
# 'adstock__radio_pipe__carryover__strength': ,45523455448406197  
# 'adstock__radio_pipe__carryover__length': 0  
# 'adstock__radio_pipe__saturation__a': ,0001974038926379962  
# 'adstock__banners_pipe__carryover__strength' : 0.3340342963936898  
# 'adstock__banners_pipe__carryover__length': 0
```

```
# 'adstock__banners_pipe_saturation__a': 0.007256873558015173
#}
#
# Coefficients = [27926.6810003 4114.46117033 2537.18883927]
# Intercepción = 5348.966158957056
```

Interpretando el modelo

Con los datos de arriba, podemos crear algunas imágenes bonitas que nos ayuden a obtener información del modelo.

Efectos de saturación

Si conectamos los valores de los transformadores de saturación, obtenemos lo siguiente:

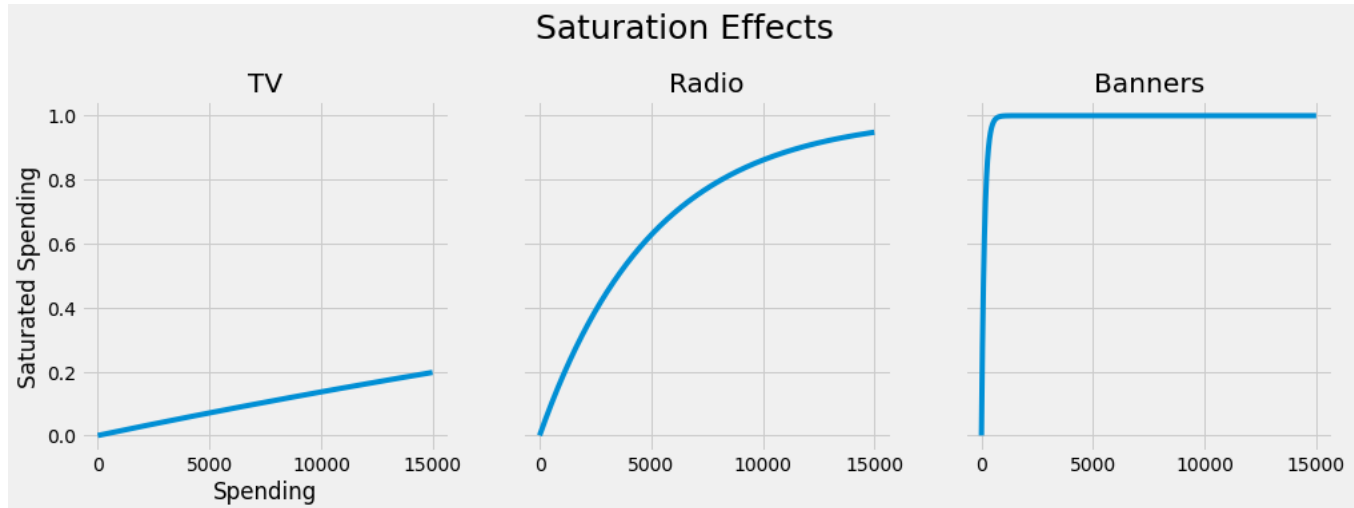


Imagen del autor.

- Podemos ver que el modelo piensa que el canal de televisión todavía está bastante *subaturado*; gastar más aquí podría ser beneficioso para las ventas. Tenga en cuenta que nuestro gasto máximo en televisión fue de aproximadamente 15000.
- La radio parece un poco más saturada, pero aumentar los gastos aquí todavía parece razonable. El gasto máximo en este canal fue de 7700.
- Las pancartas parecen *sobresaturadas*. El gasto máximo aquí fue de aproximadamente 2500, con un valor de función de casi 1 ya. Los gastos más elevados no parecen lograr mucho.

Efectos de arrastre

Si conectamos los valores de los transformadores de arrastre, obtenemos lo siguiente:

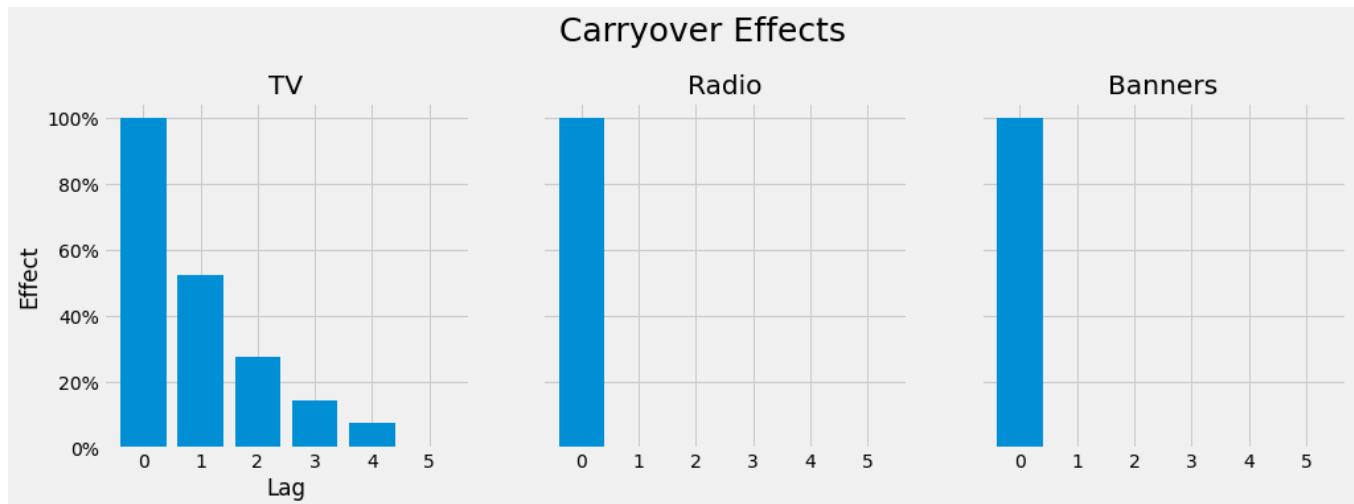


Imagen del autor.

Parece que los anuncios de televisión todavía tienen un efecto en las ventas 4 semanas después del gasto inicial. Esto es mucho más prolongado que el efecto de los gastos de radio y banners web que desaparecen rápidamente en la misma semana.

Contribuciones del canal

Como en el último artículo, podemos calcular las contribuciones de cada canal a las ventas de cada día. El código es un poco más complejo que antes porque el modelo también se volvió más complicado. De todos modos, aquí hay una versión funcional:

```
adstock_data = pd.DataFrame (
    tuned_model.best_estimator_.named_steps ['adstock']. transform
(X),
    columnas = X.columns,
    index = X.index
)

pesos = pd.Series (
    tuned_model.best_estimator_.named_steps ['regresión']. coef_,
    index = X.columns
)

base = tuned_model.best_estimator_.named_steps ['regresión'].
intercept_

unadj_contributions = adstock_data.mul (pesos) .assign (Base = base)
```

```

adj_contributions = (unadj_contributions
                      .div (unadj_contributions.sum (eje = 1), eje =
0)
                      .mul (y, eje = 0)
                      )

ax = (adj_contributions [['Base', 'Banners', 'Radio', 'TV']]
     .plot.area (
         figsize = (16, 10),
         linewidth = 1,
         title = 'Predicted Sales and Breakdown',
         ylabel = 'Ventas',
         xlabel = 'Fecha'
     )
     )

identificadores, etiquetas = ax.get_legend_handles_labels ()
ax.legend (
    identificadores [:- 1], etiquetas [:- 1],
    título = 'Canales', loc = " centro izquierda ",
    bbox_to_anchor = (1.01, 0.5)
)

```

La salida:

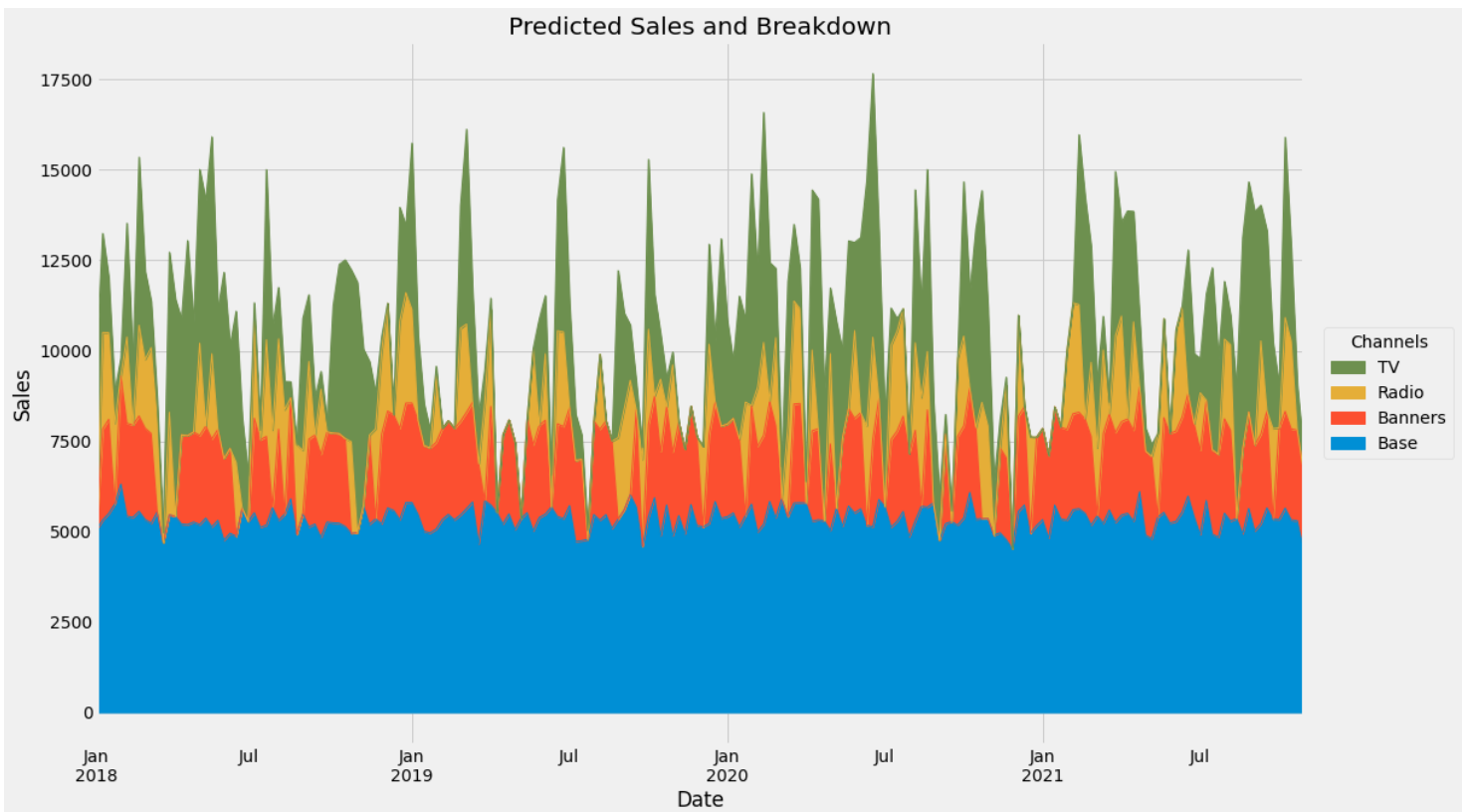
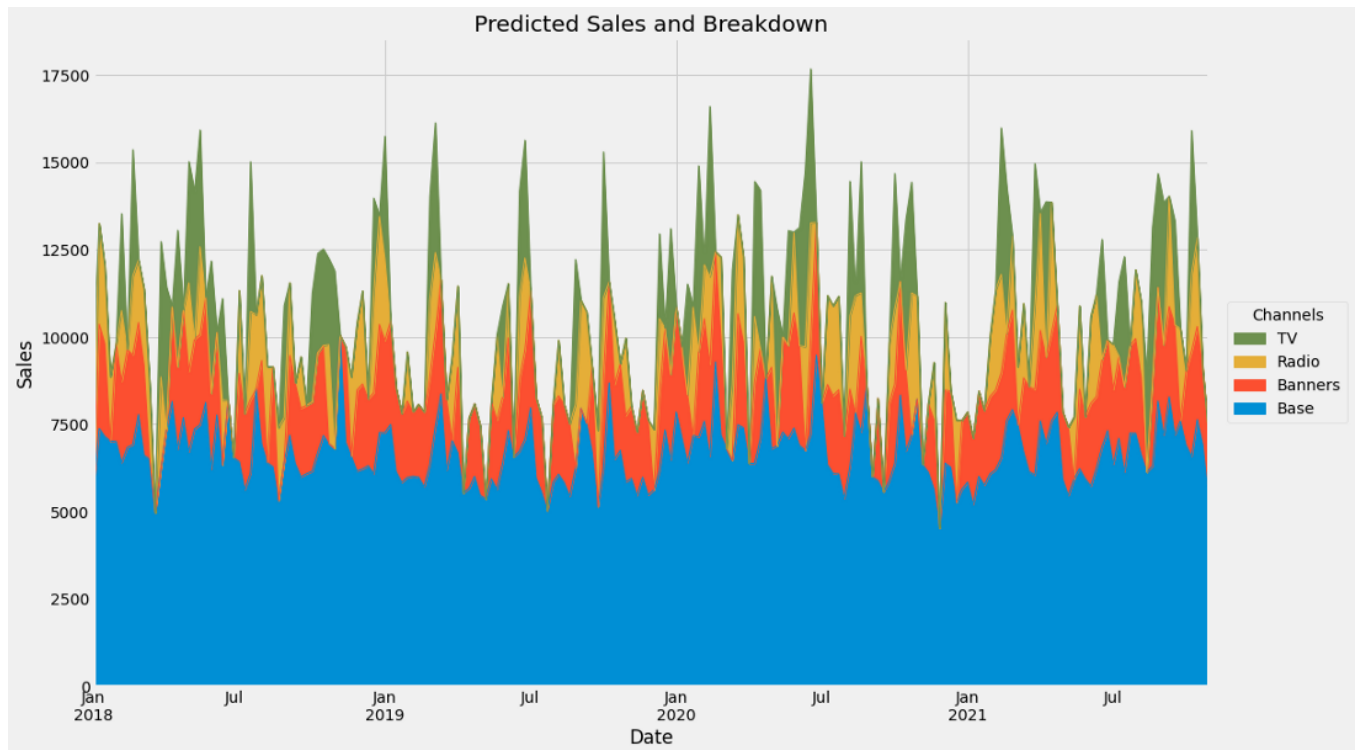


Imagen del autor.

En comparación con la imagen anterior, la línea de base ya no es tan ondulada porque el modelo puede explicar mejor las ventas con los gastos de canal dados. Aquí está el **viejo** :



El modelo antiguo . Imagen del autor.

Resumen y Outlook

En este artículo, hemos tomado nuestro modelo lineal antiguo y simple y lo hemos mejorado al no depender de los gastos de canal sin procesar, sino de los anuncios publicitarios. Los anuncios publicitarios son gastos transformados que reflejan el mundo real de manera más realista al introducir efectos de saturación y arrastre.

Incluso hicimos una implementación concisa de ambos conceptos que se pueden usar dentro del ecosistema scikit-learn de manera plug-and-play.

Lo último que hicimos fue ajustar el modelo y luego echar un vistazo a lo que había aprendido. Obtuvimos algunas ideas interesantes en forma de imágenes que las personas pueden entender fácilmente, lo que hace que este modelo sea preciso e interpretable.

Sin embargo, todavía quedan algunos cabos sueltos:

1. Hemos hablado de optimizar el gasto y cómo esto no era posible con el modelo anterior. Bueno, con el nuevo lo es, pero no entraré en detalles aquí. En resumen, trate nuestra `tuned_model` función a y optimícela con un programa de su elección, por ejemplo, **Optuna** o **scipy.optimize.minimize** . Agregue algunas restricciones presupuestarias a la optimización, como que la suma de los gastos debe ser menos de 1,000,000 en total.
2. La introducción **de** datos **no relacionados con el gasto** en el modelo podría mejorarlo aún más. Tome el **día de la semana** , el **mes** o tal vez incluso el **precio del producto** que queremos vender, por ejemplo. Especialmente el precio es un factor importante para las ventas: un iPhone normal ofrecido por 10000 € no generaría muchas ventas, mientras que uno por 100 € estaría agotado en un abrir y cerrar de ojos. Las características oportunas como el mes pueden resultar interesantes para productos de temporada, como abanicos o chocolate caliente. O días especiales como Navidad. ¡Reúna sus pensamientos y asegúrese de agregar todo lo que pueda influir en las ventas!