Implementación de Learning-MNL

Nicolás Herrera

January 20, 2025

Índice

1. Idea General del Código

El objetivo del presente informe es describir la implementación del modelo **Learning MNL** llamado **EnhancedMNL_extraInput** en el código, un modelo híbrido que combina la arquitectura clásica del Modelo Logit Multinomial (MNL) con una red neuronal profunda (DNN) para incorporar características adicionales en la predicción de elecciones discretas. El modelo utiliza dos entradas principales:

1. **Entrada Principal (`main_input`)**: Sigue la arquitectura MNL para calcular las utilidades de las alternativas.
2. **Entrada Extra (`extra_input`)**: Procesa características adicionales mediante una red neuronal para generar una nueva característica, llamada «data-driven», que se suma a las utilidades finales.

El código está implementado en **Keras** (una API de TensorFlow) y utiliza capas convolucionales (`Conv2D`), densas para implementar el MNL y así construir el modelo.

2. Flujo General del Código

2.1. 1. Definición de la Función `enhancedMNL_extraInput`

```
def enhancedMNL_extraInput(beta_num, choices_num, nExtraFeatures, networkSize,
hidden_layers=1, train_betas=True,
               minima=None, logits_activation='softmax'):
```

- **Parámetros**:
  - `beta_num`: Número de coeficientes (parámetros) en el modelo MNL.
  - `choices_num`: Número de alternativas de elección.
  - `nExtraFeatures`: Número de características adicionales.
  - `networkSize`: Tamaño de la red neuronal para procesar la entrada extra.
  - `hidden_layers`: Número de capas ocultas en la red neuronal (por defecto, 1).
  - `train_betas`: Indica si los coeficientes del MNL deben ser entrenables.
  - `minima`: Valores iniciales para los coeficientes del MNL (opcional).
  - `logits_activation`: Función de activación para las probabilidades de elección (por defecto, `softmax`).

2.2. Definición de las Entradas

```
main_input = Input((beta_num, choices_num,1), name='Features')
extra_input = Input((nExtraFeatures,1,1), name='Extra_Input')
```

- **Propósito**: Define las dos entradas del modelo:
  1. **`main_input`**: Entrada principal con dimensiones `(beta_num, choices_num, 1)`.
  2. **`extra_input`**: Entrada extra con dimensiones `(nExtraFeatures, 1, 1)`.

2.3. Cálculo de las Utilidades del MNL

```
if minima is None:
  utilities = Conv2D(filters=1, kernel_size=[beta_num,1], strides=(1,1),
padding='valid', name='Utilities',
    use_bias=False, trainable=train_betas)(main_input)
else:
  utilities = Conv2D(filters=1, kernel_size=[beta_num,1], strides=(1,1),
padding='valid', name='Utilities',
    use_bias = False, weights=minima, trainable=train_betas)(main_input)
```

- **Propósito**: Calcula las utilidades de las alternativas utilizando una capa convolucional (`Conv2D`).
- **Detalles**:
  - Si `minima` es `None`, los coeficientes se inicializan aleatoriamente.
  - Si `minima` se proporciona, se utilizan como valores iniciales para los coeficientes.

2.4. Procesamiento de la Entrada Extra con una Red Neuronal

```
dense = Conv2D(filters=networkSize, kernel_size=[nExtraFeatures, 1],
activation='relu', padding='valid', name='Dense_NN_per_frame')(extra_input)
dropped = Dropout(0.2, name='Regularizer')(dense)

x = dropped
for i in range(hidden_layers-1):
  x = Dense(units=networkSize, activation='relu', name="Dense{}".format(i))(x)
```

```
    x = Dropout(0.2, name='Drop{}".format(i))(x)
dropped = x
```

- **Propósito**: Procesa la entrada extra mediante una red neuronal.
- **Detalles**:
  - Se utiliza una capa convolucional (`Conv2D`) seguida de capas densas (`Dense`) y dropout (`Dropout`) para regularización.
  - El número de capas ocultas se define por el parámetro `hidden_layers`.

2.5. Generación de una Nueva Característica

```
new_feature = Dense(units=choices_num, name="Output_new_feature")(dropped)
new_featureR = Reshape([choices_num], name='Remove_Dim')(new_feature)
```

- **Propósito**: Genera una nueva característica a partir de la salida de la red neuronal.
- **Detalles**:
  - La salida de la red neuronal se transforma en una característica con dimensiones `(choices_num)`.

2.6. Cálculo de las Utilidades Finales

```
utilitiesR = Reshape([choices_num], name='Flatten_Dim')(utilities)
final_utilities = Add(name="New_Utility_functions")([utilitiesR, new_featureR])
```

- **Propósito**: Combina las utilidades del MNL con la nueva característica para obtener las utilidades finales.
- **Detalles**:
  - Las utilidades del MNL y la nueva característica se suman para obtener las utilidades finales.

2.7. Activación y Construcción del Modelo

```
logits = Activation(logits_activation, name='Choice')(final_utilities)
model = Model(inputs=[main_input,extra_input], outputs=logits)
return model
```

- **Propósito**: Aplica la función de activación (`softmax` por defecto) para obtener las probabilidades de elección y construye el modelo final.
- **Detalles**:
  - La salida del modelo son las probabilidades de elección (`logits`).

Bibliografía