PetFinder

DiploDatos 2019

Práctico: Aprendizaje Profundo

Integrantes:
Alini, Walter
Salina, Noelia

Motivación y Objetivo

A partir del dataset de PetFinder que también se utilizó en la materia de Aprendizaje Supervisado, se busca predecir la velocidad de adopción de un conjunto de mascotas a través de modelos de MLP.

Partimos de un esqueleto de código provisto por la cátedra, con los siguientes objetivos:

- Entenderlo en su estructura, para evidenciar en la práctica la temática vista en la materia;
- Completarlo con un funcionamiento básico, para evidenciar el conocimiento de la estructura general;
- Analizar cómo la utilización de diversos parámetros e hiper parámetros modifican los resultados del experimento;
- Intentar mejorar los resultados del experimento, a través del conocimiento de dominio adquirido en Aprendizaje Supervisado;
- Hacer al menos un commit en la competencia de Kaggle en tiempo y forma, para evidenciar la capacidad de recorrido de todo el flujo del proyecto
- Realizar al menos una prueba con Redes Convolucionales o Redes Recurrentes

Descripción del dataset

El dataset cuenta con los siguientes campos:

- **Type:** tipo de animal (1 para perro; 2 para gato)
- **Age:** edad en meses
- **Breed1, Breed2:** raza (primaria y secundaria. Se cuenta con un archivo auxiliar para este mapeo)
- **Gender:** sexo (1 para macho, 2 para hembra, 3 si en el grupo hay de ambos)
- **Color1, Color2, Color3:** color (primario, secundario y terciario. Se cuenta con un archivo auxiliar para este mapeo)
- **MaturitySize:** tamaño de adulto (1 para pequeño, 2 para mediano, …)
- **FurLength:** largo de pelo (1 para corto, 2 para mediano, …)

- **Vaccinated:** si fue vacunado
- **Dewormed:** si fue desparasitado
- **Sterilized:** si fue castrado
- **Health:** nivel de salud (1 para saludable, 2 para lesiones pequeñas, …)
- **Quantity:** Cantidad de mascotas en la entrada
- **Fee:** Valor de la adopción
- **State:** estado de ubicación de Malasia (Se cuenta con un archivo auxiliar para este mapeo)
- **Description:** Texto ingresado por el usuario para el perfil
- **AdoptionSpeed:** La velocidad de adopción (y valor a predecir)
- **PID:** Clave única de mascota

Desarrollo

Disclaimers

- No estamos orgullosos de cómo quedó el código. Hicimos lo mejor que pudimos con el tiempo que le dedicamos
- Utilizamos la importancia de features que obtuvimos en Aprendizaje Supervisado (que no utiliza redes neuronales, con lo cual es un buen indicador, pero no necesariamente el mejor)
- En todos los casos, utilizamos *accuracy* como medición de resultado de los modelos que probamos

Pequeños cambios al esqueleto

- Agregamos el parámetro *test_size* para decidir en qué proporción de dividiría el dataset en train y test, y usamos ese parámetro para llamar a la función *load_dataset* que lo tenía hardcodeado y además tomaba un parámetro *batch_size* que no era utilizado
- Unificamos *one-hot length* para que tuviera el máximo de todo el set de datos y no sólo de *dataset* para el método de Keras *to_categorical* porque eso podría provocar que no se tomara el máximo posible, sino el máximo disponible para ese recorte de datos.
- Completamos la información necesaria para un submit básico de Kaggle

Agregado de Features

Este trabajo se inició con un baseline configurado de la siguiente manera:

- 30 épocas
- 1 capa oculta con 200 nodos
- 0,1 de dropout

- 20% de datos para test
- 2 features de one-hot-encoding (Gender y Color1)
- 1 feature embedded (Breed1)
- 2 features numéricos (Age y Fee)

Se fueron agregando features de a una a la vez para observar su impacto en la performance del modelo baseline.

Feature Added	Type	Dev loss	Accuracy
N/A	One Hot Encoding	1,539428485	0,3589985967
Color2	One Hot Encoding	1,489858015	0,3717524707
Color3	One Hot Encoding	1,521538167	0,3316013217
Type	One Hot Encoding	1,520742603	0,3509683609
MaturitySize	One Hot Encoding	1,50473652	0,3641946018
FurLength	One Hot Encoding	1,493235131	0,3731695712
Vaccinated	One Hot Encoding	1,551902036	0,3311289549
Dewormed	One Hot Encoding	1,510607739	0,3608880639
Sterilized	One Hot Encoding	1,497620769	0,3613604009
Health	One Hot Encoding	1,460687091	0,3623051345
Breed2	Embedded	1,566688516	0,3646669686
Quantity	Numeric	1,460005789	0,3547472954

De la tabla anterior se puede observar que el feature que mejores aportes realizó fue FurLength, ya que disminuyó la Dev Loss del baseline en un 3% y logró aumentar la Accuracy en un 3,95%.

Por otro lado, otros features que también lograron mejorar la performance del baseline, aunque en menor medida, fueron:

- Color2
- Breed2
- MaturitySize
- Health
- Sterilized
- Dewormed

Escalado de datos

Para los datos numéricos, probamos 2 métodos de escalado (vs. sin escalar): StandardScaler y MinMaxScaler, teniendo el primero una mejor performance por lo que lo elegimos entre los datos de escalado. Sin embargo, los números contra una versión sin escalar no difieren tanto.

Hidden layers

En primera medida probamos con varias capas ocultas para nuestro modelo, con el formato de parámetros que se nos entregó como esqueleto.
No encontramos la forma de probar una cantidad “dinámica” de hidden layers con el módulo skopt, con lo que en función de las pruebas manuales que hicimos, vimos que 3 capas ocultas era un número razonable y performaba bien entre los valores que testeamos.

Skopt module

La selección de los argumentos a parametrizar se basó en las pruebas realizadas manualmente para el práctico 1, en donde también se obtuvo información respecto a los rangos a utilizar en cada argumento.
Los argumentos seleccionados fueron los siguientes:

- ***epochs***: número de pasadas a ejecutar en cada iteración de entrenamiento
  - Entero entre 1 y 70
- ***batch_size***: tamaño de la porción de datos a procesar antes de actualizar el modelo
  - Entero entre 8 y 64
- ***test_size***: porción de dataset a utilizar para testear el modelo
  - Real entre 0.1 y 0.3
- ***hidden_layer_n***: tamaño de la capa oculta “n” (n entre 1 y 3)
  - Entero entre 5 y 100 (para las 3 capas)
- ***dropout_n***: probabilidad de apagado de nodos de una hidden_layer durante el entrenamiento
  - Real entre 0.0 y 0.4 (para las 3 capas)
- ***learning_rate***: tamaño del paso en el ajuste de los pesos de la red al optimizar la función de pérdida
  - Real entre 1e-4 y 1e-1

Respecto a la función objetivo, las pruebas se realizaron utilizando ***gp_minimize***, la cual es aproximada mediante un proceso Gaussiano y es la recomendada para estos casos. De todas maneras, sería interesante probar con otras como *gbrt_minimize* o *forest_minimize*.

Los mejores resultados fueron:
Run ID (MLFlow): 2bdf239967a74838bf17afc5a0705d64

Batch_size: 46
Dropout_1: 0.338
Dropout_2: 0.136
Dropout_3: 0.329
Embedded_columns: 'Breed1', 'Breed2
Epochs: 56
Hidden_layer_1: 95
Hidden_layer_2: 71
Hidden_layer_3: 20
Learning_rate: 0.00033422758239702973
Numeric_columns: 'Age', 'Fee', 'Quantity'
One_hot_columns: 'Gender', 'Color1', 'Color2', 'Color3', 'Type', 'MaturitySize', 'FurLength', 'State', 'Vaccinated', 'Dewormed', 'Sterilized', 'Health'
Test_size: 0.107
Accuracy: 0.407
Loss: 1.428

Overfitting

Consideramos que el modelo no overfittea, en primer lugar porque los resultados son relativamente razonables con respecto a lo esperado (y a los resultados vistos en la competencia), y en segundo lugar porque se denota una degradación pequeña de la precisión contra los datos de test, con lo cual sabemos que ante datos nuevos se comporta de una manera que no es drásticamente diferente.

Redes convolucionales

A partir del esqueleto del práctico 2 agregamos una red convolucional para tratar el campo descripción del dataset.
Como resultado, obtuvimos dos conclusiones:

- Creemos que el modelo overfittea. Los valores de train y test son similares, pero cercanos al 90%, algo que a priori pareciera exagerado. De la misma forma, al hacer un late commit en la competencia de Kaggle, vemos que el resultado es mucho menor (alrededor del 25% de accuracy), lo que hace suponer que la generalización del modelo no es buena.
- Pareciera ser además que los resultados no mejoran sensiblemente la performance obtenida en el práctico 1. De todas formas, los códigos son distintos y creemos, como mencionaremos en “Trabajo Futuro” que merece la pena unificar los criterios en una sola evaluación de modelos que contemple los dos prácticos.

Pendientes y trabajo futuro

- Mejorar el código y hacer que huela un poco menos, sobre todo el del práctico 2: comentarlo y modularizarlo cuanto menos.
- Agregar la práctica sobre redes convolucionales (práctico 2) a la práctica base (práctico 1), en una sola práctica
- Probar redes recurrentes
- Hacer que la cantidad de capas ocultas sea un parámetro a optimizar con skopt
- Shufflear los datos
- Probar otras funciones de minimización de skopt