Tuning degli iperparametri in Tensorflow/Keras

Marco Fiume

16/11/2022

Indice

- Caso di studio: classificazione di fake news
- Esplorazione dati
- Preparazione dati
- Modello di base
- Hyperband
- Ottimizzazione iperparametri
- Modello finale

Classificazione di fake news

Il caso preso in esame riguarda la classificazione supervisionata di fake news. Il dataset utilizzato è WELFake dal sito www.kaggle.com, esso consiste di 72.123 articoli di cui 35.028 reali e 37.106 notizie false. Per ogni riga sono presenti 2 feature: il titolo e il testo dell'articolo.

L'obiettivo è addestrare un modello in grado di distinguere tra notizie vere o false.

Esplorazione dei dati

Struttura del dataframe pandas:

Colonne	Non Nulli	Dtype
Unnamed:0	72134	int64
title	71576	object
text	72095	object
label	72134	int64

- Unnamed:0 è un id progressivo;
- label può assumere i valori 0 = fake news e 1 = notizia reale;
- sono presenti dati mancanti nelle colonne title e text.

Preparazione dei dati

- I dati mancanti sono stati riempiti utilizzando una stringa di testo vuota;
- sono stati concatenati titolo e testo degli articoli separandoli con uno spazio;
- le colonne contenenti l'id, il titolo e il testo sono state rimosse;
- il dataframe così ottenuto è stato suddiviso in training e test set riservando il 20% dei dati per il test;

```
df.fillna(value='', inplace=True)
df['full-article'] = df['title']+' '+df['text']
data=df.drop(['Unnamed: 0', 'title', 'text'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(data['full-article'], data['label'], test_size=0.20, random_state=42)
```

Modello di base - rete neurale in Tensorflow/Keras

```
max_features=50
vectorize_layer=TextVectorization(max_tokens=max_features, output_mode='binary')
vectorize_layer.adapt(X_train)

model=tf.keras.Sequential()
model.add(tf.keras.Input(shape=(), dtype=tf.string))
model.add(vectorize_layer)
model.add(tf.keras.layers.Dense(20, activation='relu', name='dense_1'))
model.add(tf.keras.layers.Dense(2, activation='softmax', name='dense_2'))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=tf.keras.metrics.SparseCategoricalAccuracy())

callbacks=[
    tf.keras.callbacks.EarlyStopping(
        monitor='val_sparse_categorical_accuracy',
        min_delta=0.01,
        patience=3,
        restore_best_weights=True
    )
]

history=model.fit(
    x=X_train,
    y=y_train,
    validation_split=0.2,
    epochs=10,
    callbacks=callbacks,
    batch_size=64
)
```

- vettorizzazione input;
- livello nascosto con 20 unità con funzione di attivazione Relu ;
- uscita con funzione di attivazione softmax;
- algoritmo di ottimizzazione: adam;
- early stopping;
- validation_split: 0.2

Modello di base - prestazioni

	precision	recall	f1-score	support
0	0.81	0.77	0.79	7089
1	0.79	0.83	0.81	7338
accuracy			0.80	14227
macro avg	0.80	0.80	0.80	14227
weighted avg	0.80	0.80	0.80	14227

Hyperband

L'algoritmo hyperband descritto in questo articolo (link) rappresenta un estensione del successive halving.

Successive halving

- Tutti i modelli candidati sono addestrati per un numero limitato di epoche. Nel caso in cui lo spazio ricerca sia troppo ampio viene selezionato un campione casuale di candidati come nel random grid search;
- si elimina metà dei modelli, in modo da conservare quelli che hanno ottenuto il punteggio migliore sui dati di validazione;
- nel round successivo l'addestramento dei modelli superstiti prosegue per un numero doppio di epoche;
- il processo si ripete finchè non è rimasto un solo modello.

Hyperband

Limiti di questo approccio

Nel punto 1, nel caso in cui lo spazio di ricerca sia troppo ampio è necessario selezionare n modelli. Considerando il budget totale di risorse a disposizione B, ogni modello potrà essere addestrato per B/n epoche.

- se n è grande il rischio è di eliminare immediatamente dopo una sola iterazione modelli buoni, ma lenti a convergere verso un risultato ottimale;
- se n è basso, quindi B/n è alto, si stanno testando meno candidati nello spazio di ricerca, potenzialmente lasciando fuori combinazioni migliori. Inoltre si rischia di allocare subito molte risorse su modelli scadenti.

Hyperband

Per ovviare a questi problemi Hyperband consiste nell'iterare succcessive halving per diversi valori di n:

- in ogni iterazione del ciclo esterno, denominata bracket, viene selezionato un valore n;
- per ogni B/n viene eseguito il successive halving.

Ottimizzazione iperparametri

```
vectorize_layer=TextVectorization(max_tokens=3000, output_mode='binary')
vectorize_layer.adapt(X_train)

def build_model(hp):
    model = keras.Sequential()
    model.add(tf.keras.Input(shape=(1), dtype=tf.string))
    model.add(vectorize_layer)
    for i in range(hp.Int('dense_layers', 1, 3)):
        units=hp.Int(f'units_{i}', min_value=10, max_value=100, step=10)
        if hp.Boolean('dropout'):
            model.add(keras.layers.Dropout(0.5))
        model.add(keras.layers.Dense(units=units, activation='relu', name=f'dense_{i}'))
    model.add(tf.keras.layers.Dense(2, activation='softmax', name='output'))


    model.compile(
        optimizer='adam',
        loss=keras.losses.SparseCategoricalCrossentropy(),
        metrics=keras.metrics.SparseCategoricalAccuracy()
   )

    return model

callbacks=[
    tf.keras.callbacks.EarlyStopping(
        monitor='val_sparse_categorical_accuracy',
        min_delta=0.01,
        patience=3,
        restore_best_weights=True
    )
]
tuner = kt.Hyperband(build_model,
                     objective='val_sparse_categorical_accuracy',
                     max_epochs=15,
                     factor=3,
                     seed=42,
                     directory='log',
                     project_name='keras_tuning'
                     )

tuner.search_space_summary()
```

- vettorizzazione input;
- da 1 a 3 livelli nascosti;
- da 10 a 100 unità per ogni livello;
- dropout 0.5 o 0;
- stessi parametri di addestramento del modello di base;
- “factor” è il fattore di riduzione dei candidati per ogni bracket;
- “max_epochs” stabilisce il massimo budget a dispozione per l’addestramento di un modello.

Modello finale

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization_2 (TextV  (None, 3000)             0
 ectorization)

 dense_0 (Dense)             (None, 90)                270090

 dense_1 (Dense)             (None, 20)                1820

 output (Dense)              (None, 2)                 42

=================================================================
Total params: 271,952
Trainable params: 271,952
Non-trainable params: 0
```

- 2 livelli nascosti;
- primo livello 90 unità;
- secondo livello 20 unità;
- nessun dropout

Modello finale - prestazioni

	precision	recall	f1-score	support
0	0.98	0.98	0.98	7089
1	0.98	0.98	0.98	7338
accuracy			0.98	14227
macro avg	0.98	0.98	0.98	14227
weighted avg	0.98	0.98	0.98	14227