Assignment 1

Dataset of default Payments Information in Taiwan 2005

Informazioni iniziali

Il progetto è stato realizzato su google colab in python, servendosi delle librerie Keras, Keras_metrics (è già presente il comando pip per installarla) e Pandas.
I Path dei due file (train.csv e test.csv) vanno modificati prima dell'esecuzione: nel caso riportato essi sono presi direttamente dal mio Google Drive.

Breve analisi del Dataset

Il dataset è diviso in train (contenente le label da predire) e test (senza la label da predire).

- Shape del **train**: (27000, 24)
- Shape del **test**: (3000, 23)

Stampando l'header del train sembra esserci uno sbilanciamento sulla label da predire. Infatti, dopo una veloce verifica:

- **21.027** valori **0** (77,88% del totale)
- **5.973** valori **1** (22,1% del totale)

Sarà necessario prestare attenzione per evitare che il modello non tenda ad associare 0 a tutte le label: in quel caso dovrei notare una accuracy intorno al 77%.

Preprocessing dei dati

Binarizzazione degli attributi categorici

Ho individuato all'interno del dataset i seguenti **attributi categorici**:

SEX EDUCATION MARRIAGE AGE PAY_0 PAY_2 PAY_3 PAY_4 PAY_5 PAY_6

Essi sono stati **binarizzati** con l'istruzione:
pd.concat([train,pd.get_dummies(train['NOME_COLONNA'], prefix='NOME_COLONNA',drop_first=False)],axis=1).drop(["NOME_COLONNA"],axis=1)
che rimuove la colonna del dataset di partenza e la sostituisce con le colonne binarizzate'.

Questa istruzione sul dataset di **test** produce meno colonne poiché non presenta tutti i valori del dataset di **train**. Ho quindi **aggiunto manualmente** le suddette colonne mancanti e ho ordinato gli attributi in ordine alfabetico in entrambi i dataset, in modo da averli **simmetrici** (al netto del fatto che il dataset di train ha in più l'attributo da predire)

Rimozione delle label da predire

Per poter effettuare la fase di traning e rendere i dati di test e train identici dal punto di vista degli attributi ho **rimosso dal dataset di train** la colonna "**default.payment.next.month**" con l'attributo target e l'ho inserita in un array a parte.

Standardizzazione degli attributi

$$V_S = \frac{X - \mu_X}{\sigma_X}$$

Tramite la funzione **preprocess_data** presente nel source code ho standardizzato il dataset di train e il dataset di test.

Creazione dei dati di validazione

Tramite la funzione train_test_split ho **estratto dal dataset di traning i dati di validazione,** che ho deciso essere il 15% dei dati di train totali per non diminuire troppo quest'ultimi.

Prima della creazione del validation set:

- **Train** data shape: (27000, 91)
- **Test** data shape: (3000, 91)

Dopo della creazione del validation set:

- Train data shape: (22950, 91)
- Validation data shape: (4050, 91)
- Test data shape: (3000, 91)

Definizione del modello

```
model = Sequential()
model.add(Dense(128, activation='relu',input_shape=(dims,) ))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',
              metrics=['accuracy', keras_metrics.precision(), keras_metrics.recall()])
model.summary()
```

Il modello essendo di classificazione binaria è stato così definito.
Ho inoltre calcolato tramite la libreria esterna keras_metrics i valori di **precision** e **recall** per ogni epoca.

I **layer** hanno come attivatore la funzione **relu**, e sono rispettivamente di **128,64,1** neuroni. Infatti un numero relativamente alto di record mi potrebbe consentire di apprendere tutti i parametri che richiedono i neuroni.

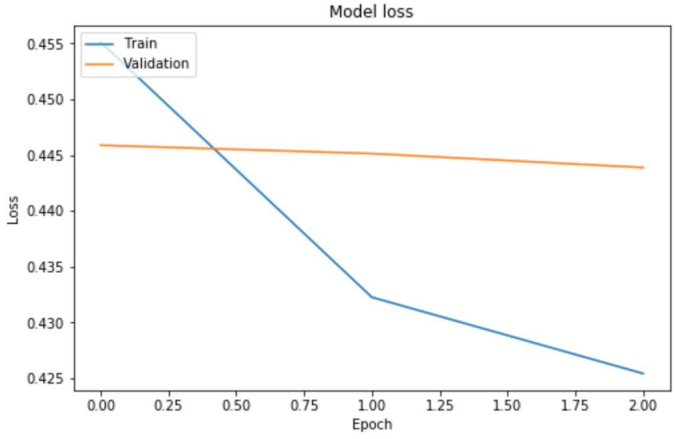
L'ultimo layer ha output 1 poiché stiamo trattando un problema **binario**.
La funzione di **loss** scelta è la **binary_crossentropy**, essendo un problema binario
L'ottimizzatore scelto è **adam** (https://arxiv.org/abs/1412.6980v8)

Non ho rilevato **grandi cambiamenti** modificando il numero di layer e il numero di neuroni per layer.
Stesso risultato provando a usare **Adamax**, **Nadam**, **SDG**: non si notano grandi cambiamenti e in alcuni casi (es. SDG) **sono necessarie più epoche** e quindi più tempo computazionale a parità di risultato, infatti **early_stop** agisce dopo più epoche rispetto a adam.

Prestazioni del Modello sui dati di train e validation

Eseguendo il modello con i dati di train e di validazione **ottengo una accuracy dell'82%** circa alla **3a epoca**: oltre la terza epoca la loss function della validation tende a salire e il modello si ferma tramite la callback **early_stop** impostata su val_loss.

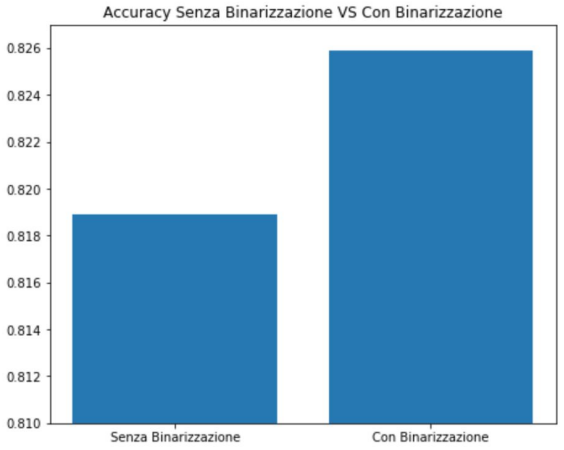
```
Train on 22950 samples, validate on 4050 samples
Epoch 1/10
22950/22950 [==============================] - 1s 27us/step - loss: 0.4602 - acc: 0.8080 - precision: 0.6297 - recall: 0.3050 - val_loss: 0.4408 - val_acc: 0.8210 - val_precision: 0.7008 - val_recall: 0.389
Epoch 2/10
22950/22950 [==============================] - 0s 19us/step - loss: 0.4314 - acc: 0.8225 - precision: 0.6850 - recall: 0.3552 - val_loss: 0.4343 - val_acc: 0.8178 - val_precision: 0.7143 - val_recall: 0.348
Epoch 3/10
22950/22950 [==============================] - 0s 19us/step - loss: 0.4265 - acc: 0.8238 - precision: 0.6917 - recall: 0.3565 - val_loss: 0.4374 - val_acc: 0.8215 - val_precision: 0.6952 - val_recall: 0.400
Epoch 00003: early stopping
```



Prestazioni alla terza epoca

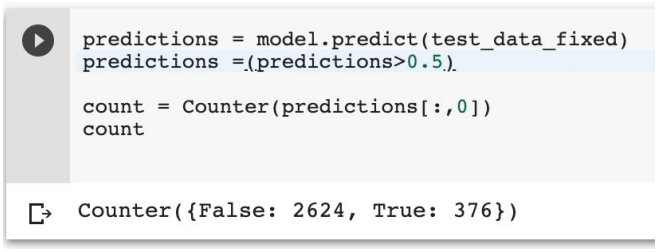
- **loss:** 0.4265
- **accuracy**: 0.8238
- **precision**: 0.6958
- **recall**: 0.3653

Per comprendere se la binarizzazione degli attributi ha dato effetti positivi sull'apprendimento del modello **ho provato a commentare le linee di codice contenenti la binarizzazione** e ho potuto verificare che l'incremento di accuracy nel modello è molto limitato ma comunque presente binarizzando gli attributi categorici.



Risultati del modello sui dati di test

Eseguendo il modello sul test set fornito ottengo i seguenti risultati



```
predictions = model.predict(test_data_fixed)
predictions =(predictions>0.5)

count = Counter(predictions[:,0])
count
```

```
Counter({False: 2624, True: 376})
```

- **False**: 2624
- **True**: 376

Il notebook che viene consegnato presenta già alcuni commenti: alcuni sono ripetitivi di quelli già mostrati qui, altri sono ragionamenti “miei” che mi scrivevo mentre lo realizzavo per aiutarmi ad intepretare il codice scritto.