Progetto di una rete fully connected che effettua classificazione binaria sul dataset “creditcardcsvpresent”

Matteo Brina

Il Dataset

Descrizione generale

- I campioni del dataset in esame constano di transazioni effettuate con carta di credito presumibilmente per l’acquisto di beni o servizi online in tutto il mondo.
- È composto da 6150 campioni di cui solo 896 positivi. Si può notare che il dataset è abbastanza sbilanciato con un rapporto positivi-negativi di 14,56% contro 85,44%

Il Dataset

Gli attributi dei campioni

- Merchant id
- Transaction date (tutti i campi sono vuoti)
- Average amount transaction per day
- Transaction amount
- Is declined
- Total number of declines per day
- Is foreign transaction

Il Dataset

Gli attributi dei campioni

- Is high risk country
- Daily chargeback average amount
- 6 month average chargeback account
- 6 month chargeback frequency
- Is fradulent (etichetta)

Lo scopo della rete

- Lo scopo del progetto è quello di creare una rete neurale che sia capace, apprendendo dal dataset, di classificare una transazione come fraudolenta o meno.

Operazioni preliminari sul dataset

Eliminazione campi

- La prima cosa che è stata fatta è quella di eliminare l'attributo transaction date da tutti i campioni in quanto tutti i valori erano 'NaN'.
- Successivamente è stato rimosso anche l'attributo 'Merchant_id' perché, dietro l'ipotesi che l'informazione non fosse rilevante, si è notato che rimuovendolo si raggiungevano punteggi migliori di val_loss più rapidamente.

```
data.pop('Transaction date')
data.pop('Merchant_id')
```

Operazioni preliminari sul dataset

Codifica dei campi booleani

- I campi booleani sono stati codificati in Yes=1 e No=0 per poter essere presi in input dalla rete.

```
code={
    "N":0,
    "Y":1
}
```

```
for obj in data.select_dtypes("object"):
    data[obj]=data[obj].map(code)
```

Operazioni preliminari sul dataset

Divisione degli attributi in input ed etichetta

- Gli attributi del dataset sono stati divisi in due gruppi. Il primo è quello i cui attributi andranno a costituire gli input della rete, il secondo rappresenta l'etichetta degli esempi.

```
X = data.iloc[:, :-1].values
```

```
y = data["isFradulent"].values
```

Operazioni preliminari sul dataset

Divisione del dataset in dati di training, validazione e test

- Il dataset viene diviso in un 75% di dati di training e un restante 25% a sua volta diviso in due in 12,5% di dati di test e un 12,5% di dati di validation.

```
X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.25, random_state=11111)
```

```
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.5, random_state=11111)
```

Operazioni preliminari sul dataset

Normalizzazione dei dati

- I valori degli attributi degli esempi vengono normalizzati in un range compreso tra 0 e 1 in modo da evitare fenomeni di exploding gradient causati da valori che eccedono l'unità.

```
y_train_r=y_train.reshape(-1, 1)
```

```
y_test_r=y_test.reshape(-1, 1)
```

```
y_val_r=y_val.reshape(-1, 1)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledy_train = scaler.fit_transform(y_train_r)
rescaledX_test = scaler.fit_transform(X_test)
rescaledy_test = scaler.fit_transform(y_test_r)
rescaledX_val = scaler.fit_transform(X_val)
rescaledy_val = scaler.fit_transform(y_val_r)
```

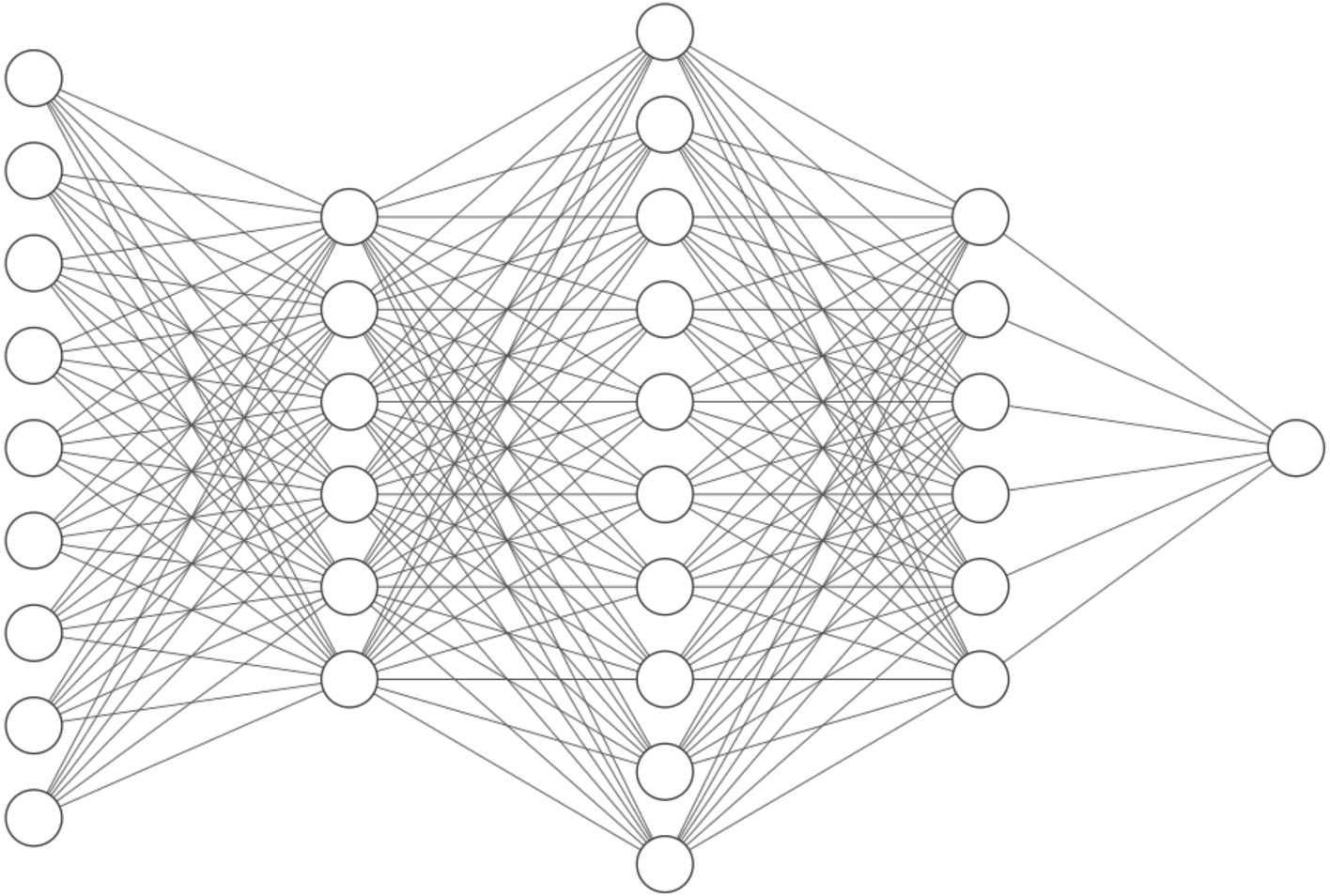
Generazione della rete

Struttura della rete

- Dopo aver compiuto vari test con diverse configurazioni di numero di strati nascosti, numero di neuroni per ogni strato e metodi di regolarizzazione, la rete che ha dato le prestazioni migliori è stata una rete composta da 3 strati nascosti con un numero di 6, 10 e 6 neuroni in ogni strato.

Generazione della rete

Struttura della rete



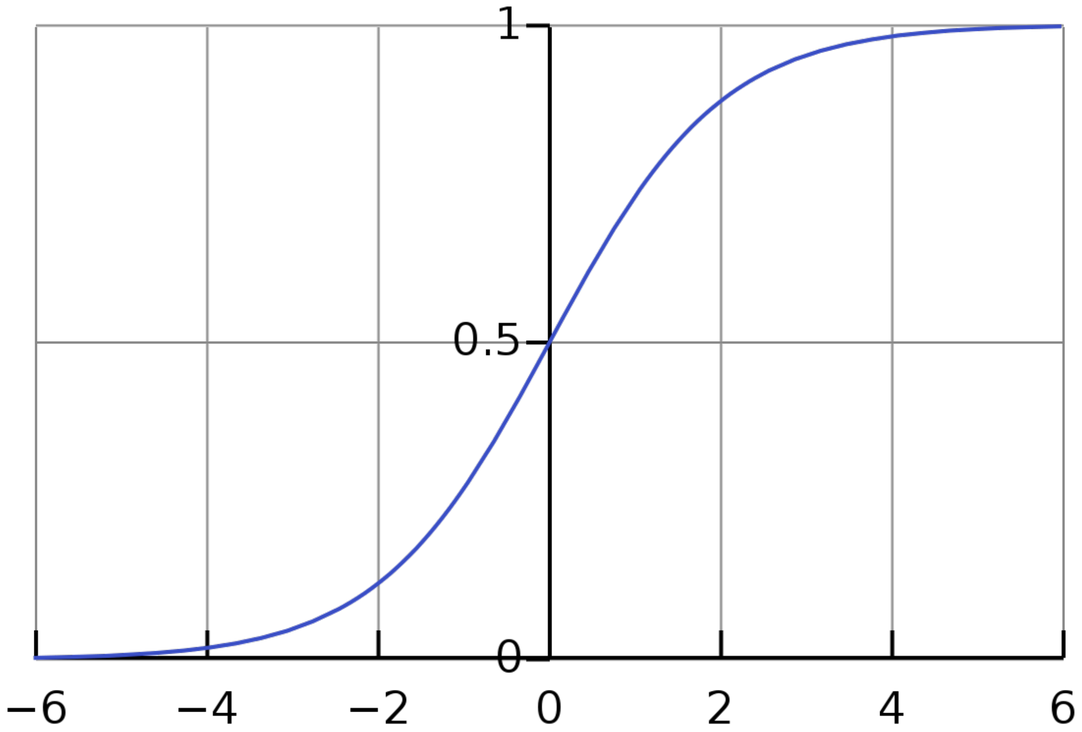
Generazione della rete

Funzioni di attivazione

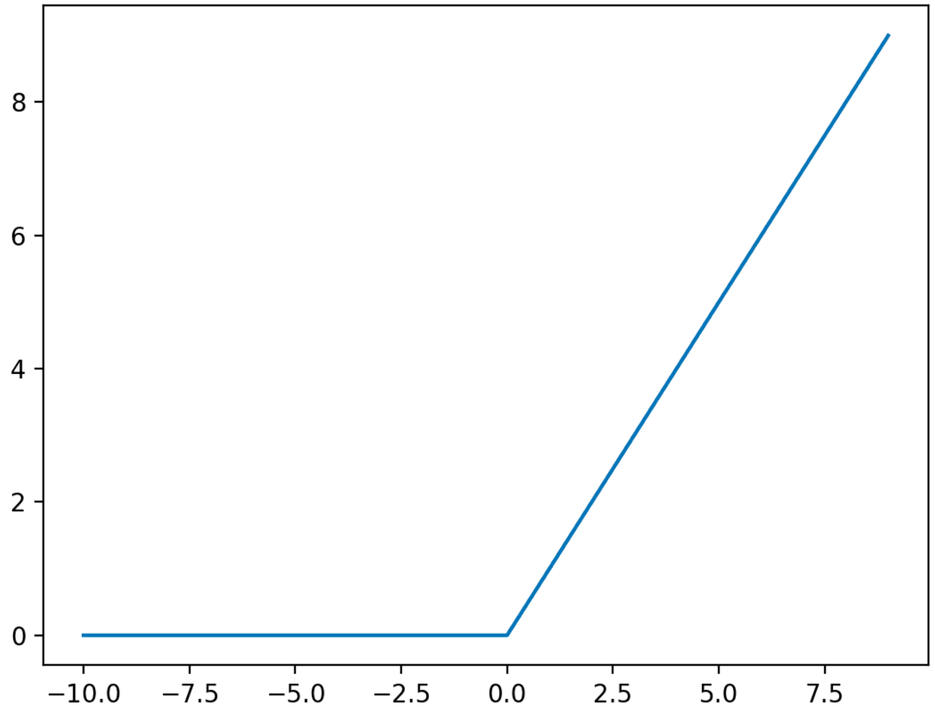
- La funzione di attivazione dei neuroni degli strati nascosti è ovviamente la relu, mentre quella del neurone di output è la sigmoide in quanto la rete deve effettuare classificazione binaria.

Generazione della rete

Funzioni di attivazione



Sigmoide



Relu

Generazione della rete

Metodo di ottimizzazione

- Come metodo di ottimizzazione è stato scelto Adam in quanto fornisce prestazioni migliori rispetto a rmsprop.

Generazione della rete

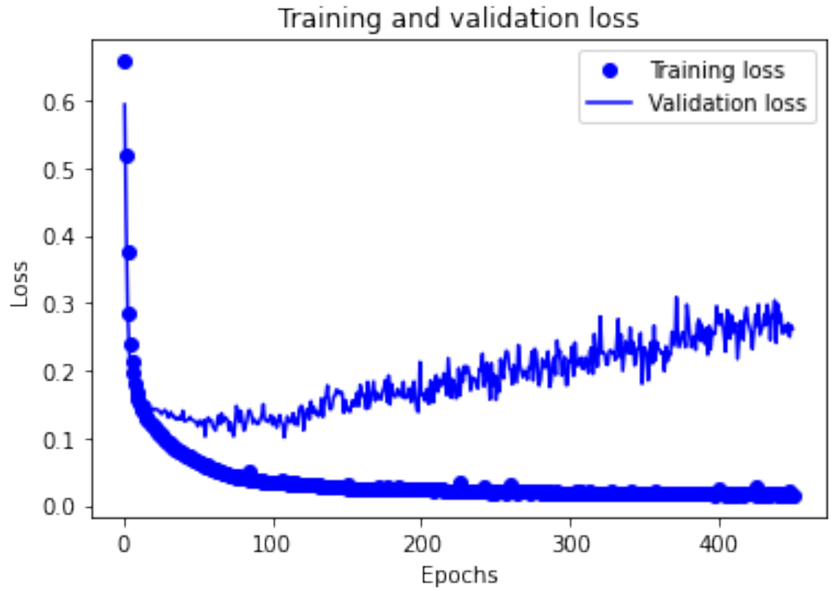
Metodi di regolarizzazione

- In seguito a numerosi test, i metodi di regolarizzazione che si sono dimostrati più efficienti sono stati la regolarizzazione l2 unita al dropout. La regolarizzazione l2 effettua una sorta di feature selection andando a portare a 0 i pesi delle feature che non sono rilevanti, mentre il dropout va ad allenare tutte le sottoreti che si possono ottenere rimuovendo stocasticamente neuroni dalla rete di partenza.

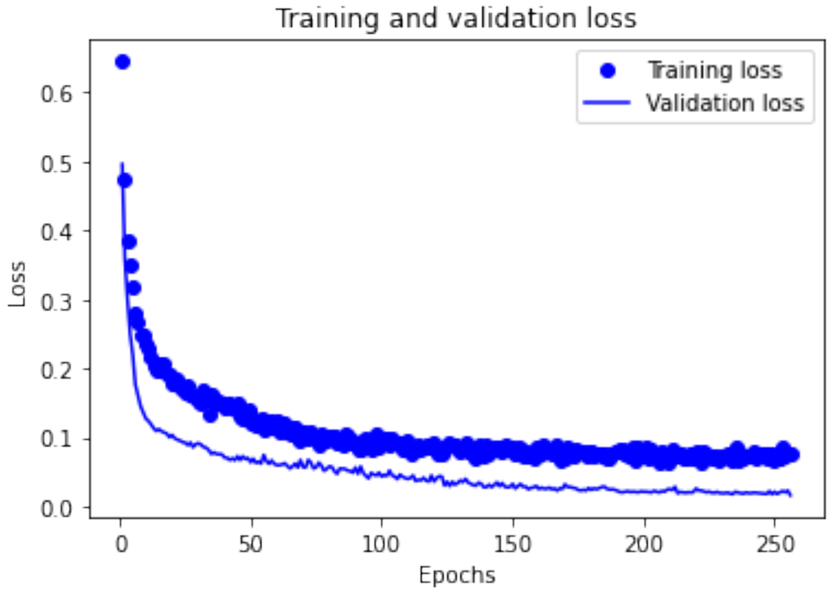
Generazione della rete

Metodi di regolarizzazione

- L’apporto della regolarizzazione in termini di capacità della rete, e quindi capacità di generalizzazione e velocità con cui si raggiunge l’overfitting è fondamentale.



Senza regolarizzazione



Con regolarizzazione

Generazione della rete

Funzione di loss

- Come funzione di loss è stata scelta la binary crossentropy in quanto è la più corretta da adottare nei casi di classificazione binaria.

$$-\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_1)$$

Generazione della rete

Metriche

- Le metriche che vengono calcolate durante l’apprendimento sono:
- PR AUC (mette in relazione precision e recall)
- ROC AUC (mette in relazione tpr e fpr)

Generazione della rete

Sommario

```
model1 = keras.Sequential([
     keras.layers.Dense(6, activation=tf.nn.relu, input_shape=(9,)),
     keras.layers.Dropout(0.5),
    keras.layers.Dense(10, kernel_regularizer=keras.regularizers.l2(0.01),
activation=tf.nn.relu),
    keras.layers.Dense(6,  activation=tf.nn.relu),
    keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

model1.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy', 'binary_crossentropy',
tf.keras.metrics.AUC(curve="PR"), tf.keras.metrics.AUC(curve="ROC")])

model1.summary()
```

Training della rete

Gradient descent

- Per addestrare la rete è stato scelto un gradient descent di tipo batch con una batch size di 32.
- Il numero di epoche è stato scelto arbitrariamente grande in quanto l'apprendimento viene fermato da una funzione di callback che ottimizza certi parametri.

Training della rete

Callback

- È stata scritta una funzione di callback personalizzata che fermi l’apprendimento nel momento in cui la PR-AUC e la ROC-AUC fossero uguali a 1 e la loss fosse minore di 0.02 (valore scelto empiricamente andando a valutare i minimi raggiungibili dopo un numero molto grande di epoche).

Training della rete

Callback

```
class haltCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if((logs.get('val_auc') >= 1) and (logs.get('val_loss') <= 0.02)
and (logs.get('val_auc_1') >= 1)):
            print("\n\n\nReached best value so cancelling training!
\n\n\n")
            self.model.stop_training = True


trainingStopCallback = haltCallback()
```

Risultati del training

Valori

- Come vincolato dalla callback, sui dati di validation ROC-AUC e PR-AUC sono uguali a 1 e la loss è pari a 0.016.
- Il training è avvenuto in 257 epoche

```
roc-auc is 1.000
average precision score is 1.000
pr-auc is 1.000
```

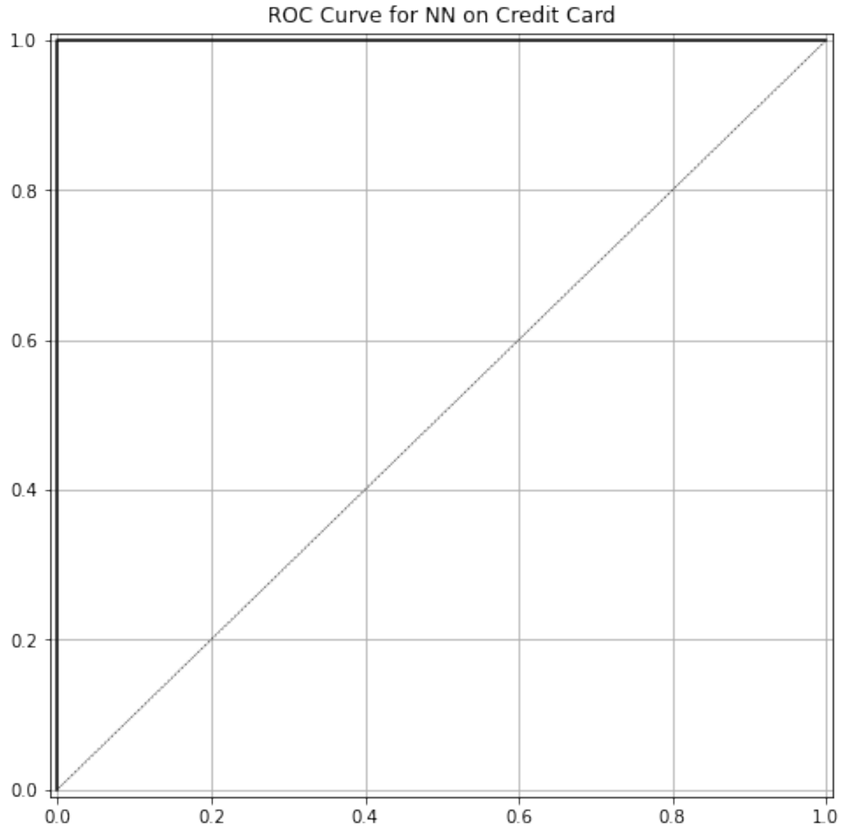
Risultati del training

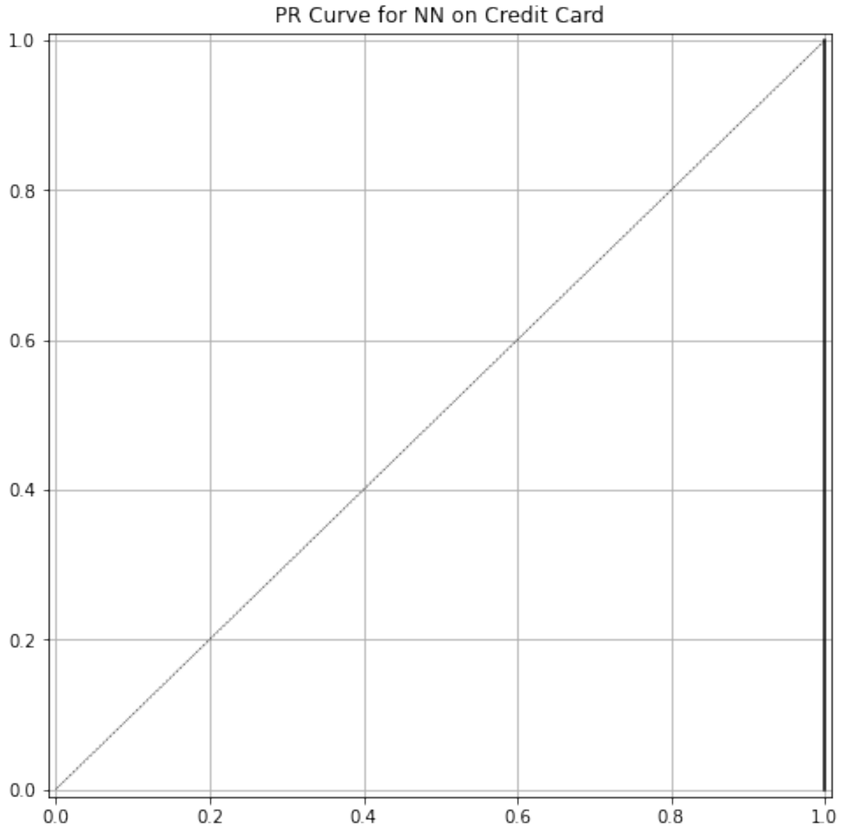
Codice

```
y_pred_prob_nn_1 = model1.predict(rescaledX_val)

y_pred_class_nn_1 = y_pred_prob_nn_1.round()

print('roc-auc is {:.3f}'.format(roc_auc_score(y_val,y_pred_prob_nn_1)))
print('average precision score is
{:.3f}'.format(average_precision_score(y_val,y_pred_prob_nn_1)))
x, y, z=precision_recall_curve(y_val, y_pred_prob_nn_1)
print('pr-auc is {:.3f}'.format(auc(y, x)))


plot_roc(y_val, y_pred_prob_nn_1, 'NN')
plot_pr(y_val, y_pred_prob_nn_1, ‘NN')
```

Risultati del training

Grafici





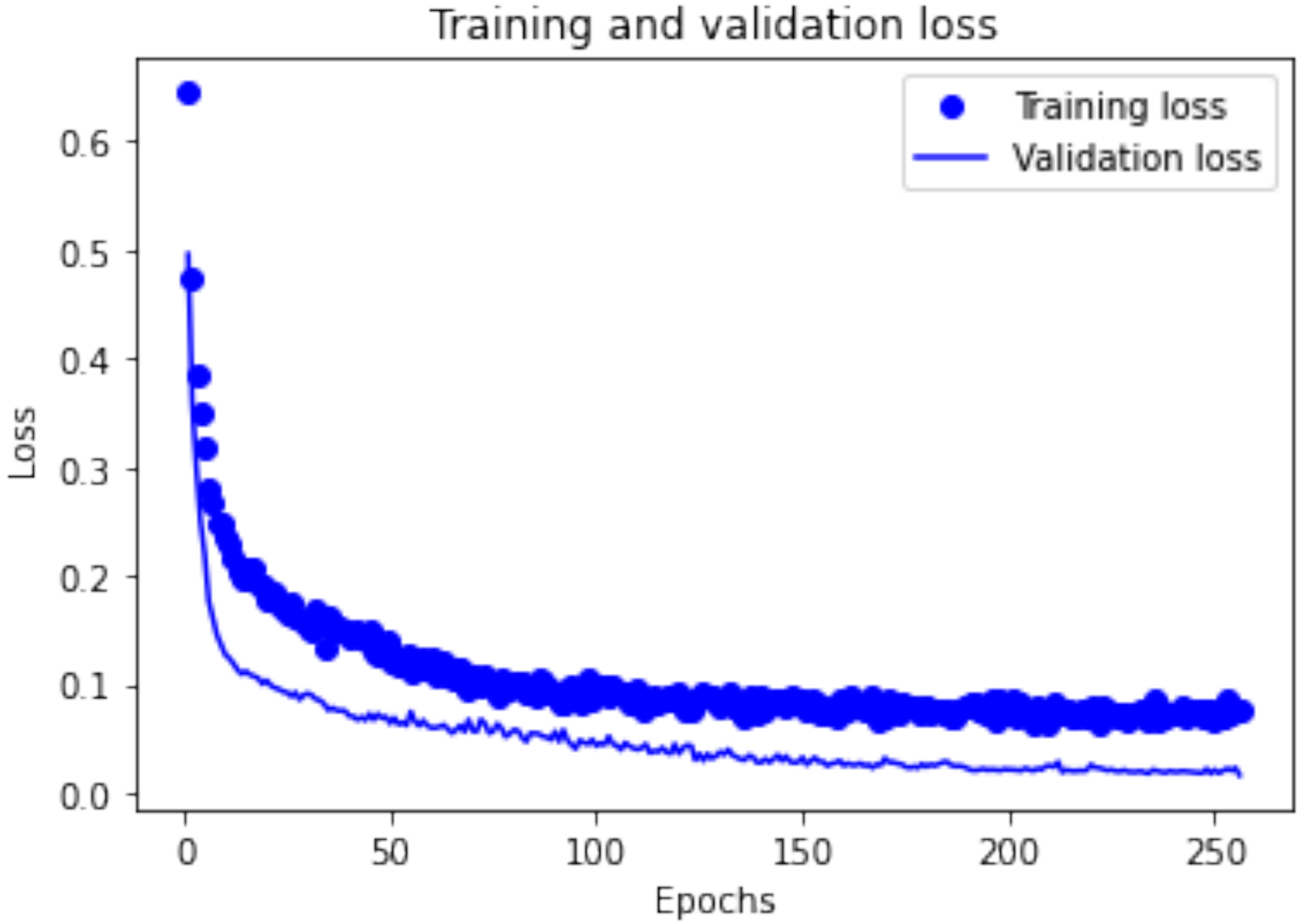
Risultati del training

Codice

```
loss = history_dict['loss']
val_loss = history_dict['val_loss']
binary_crossentropy = history_dict['binary_crossentropy']
val_binary_crossentropy = history_dict['val_binary_crossentropy']
pr_auc = history_dict['auc']
pr_val_auc = history_dict['val_auc']
roc_auc = history_dict['auc_1']
roc_val_auc = history_dict['val_auc_1']

epochs = range(1, len(acc) + 1)
```

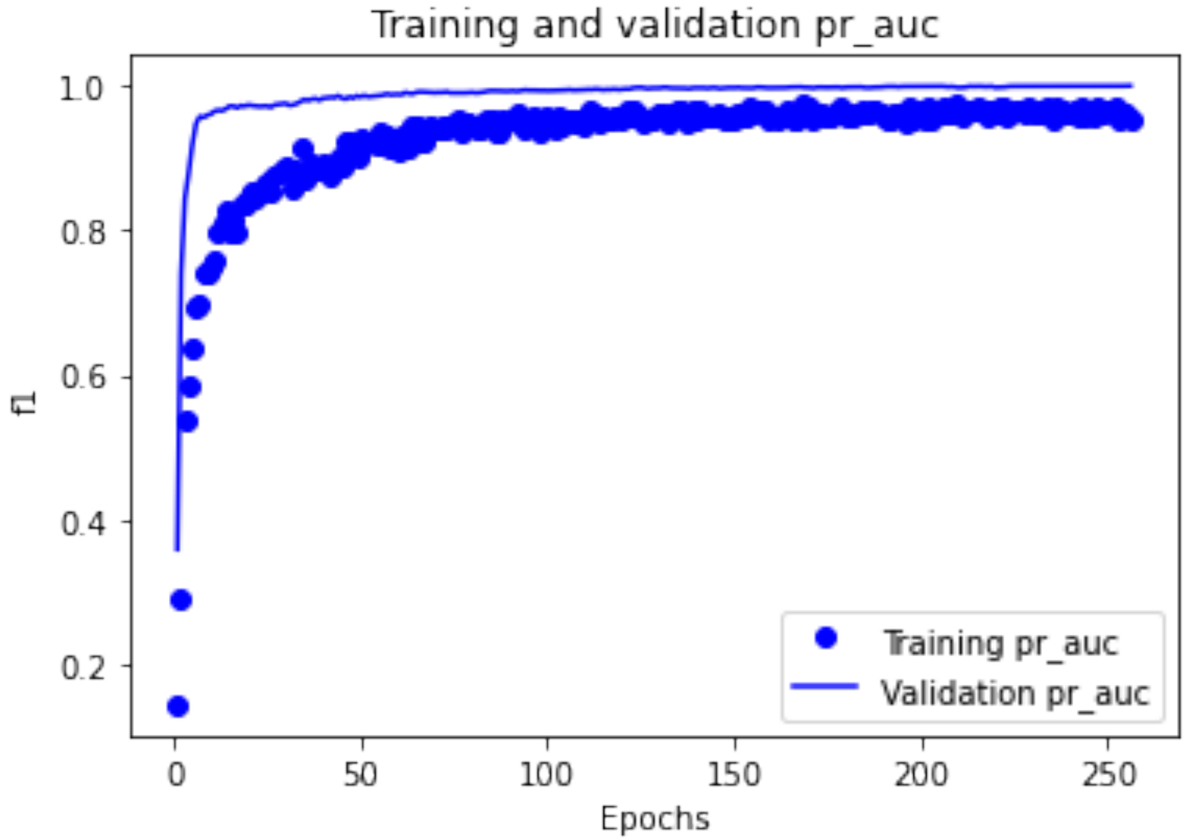
Risultati del training

Grafici



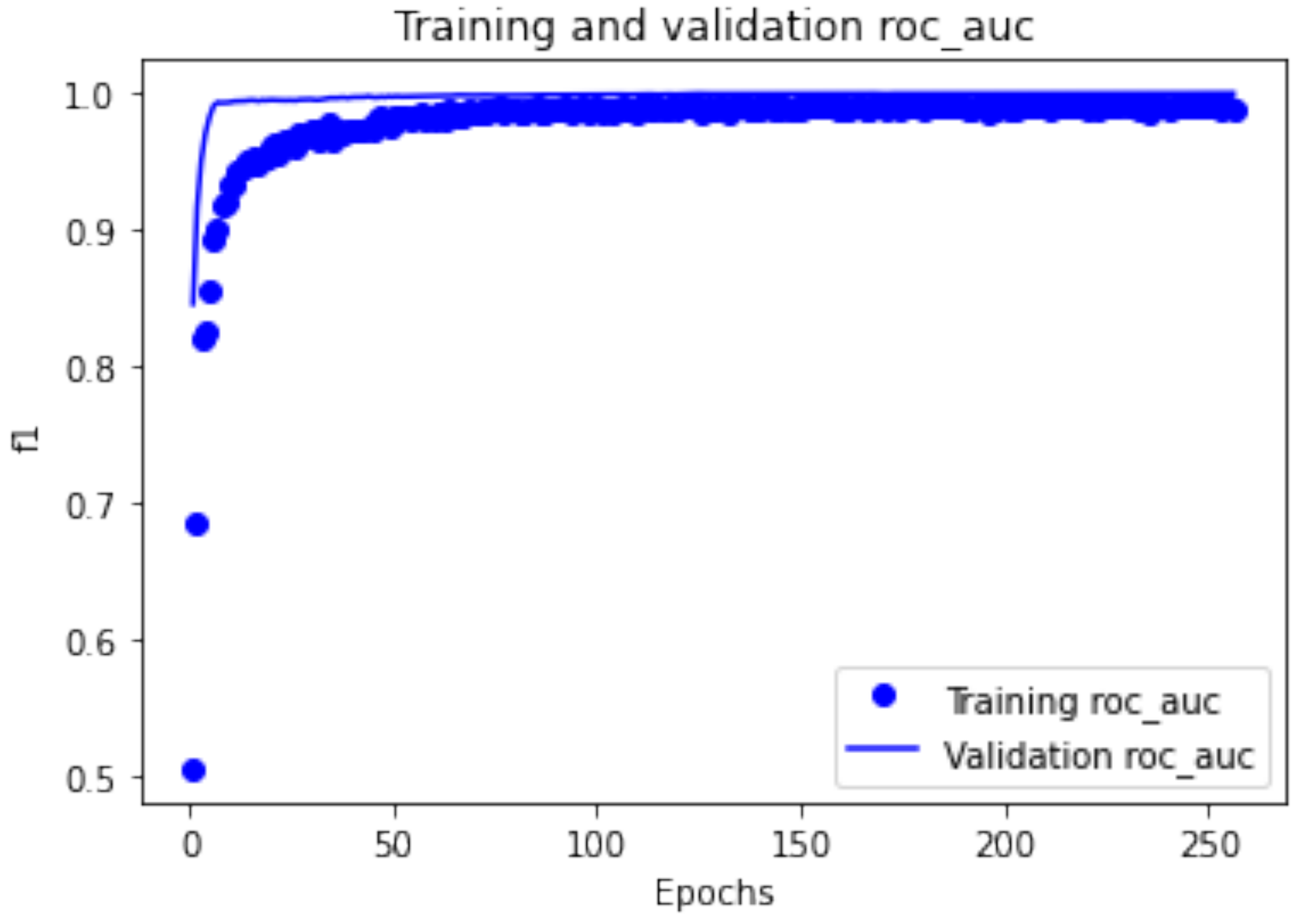
Risultati del training

Grafici



Risultati del training

Grafici



Salvataggio

Salvataggio modello

```
model1.save(‘/content/drive/MyDrive/modello/mio_modello21')
```

Salvataggio

Salvataggio history

```
with open(‘/content/drive/MyDrive/modello/trainHistoryDict12', 'wb') as
file_pi:
        pickle.dump(model1history.history, file_pi)
```

Valutazione modello

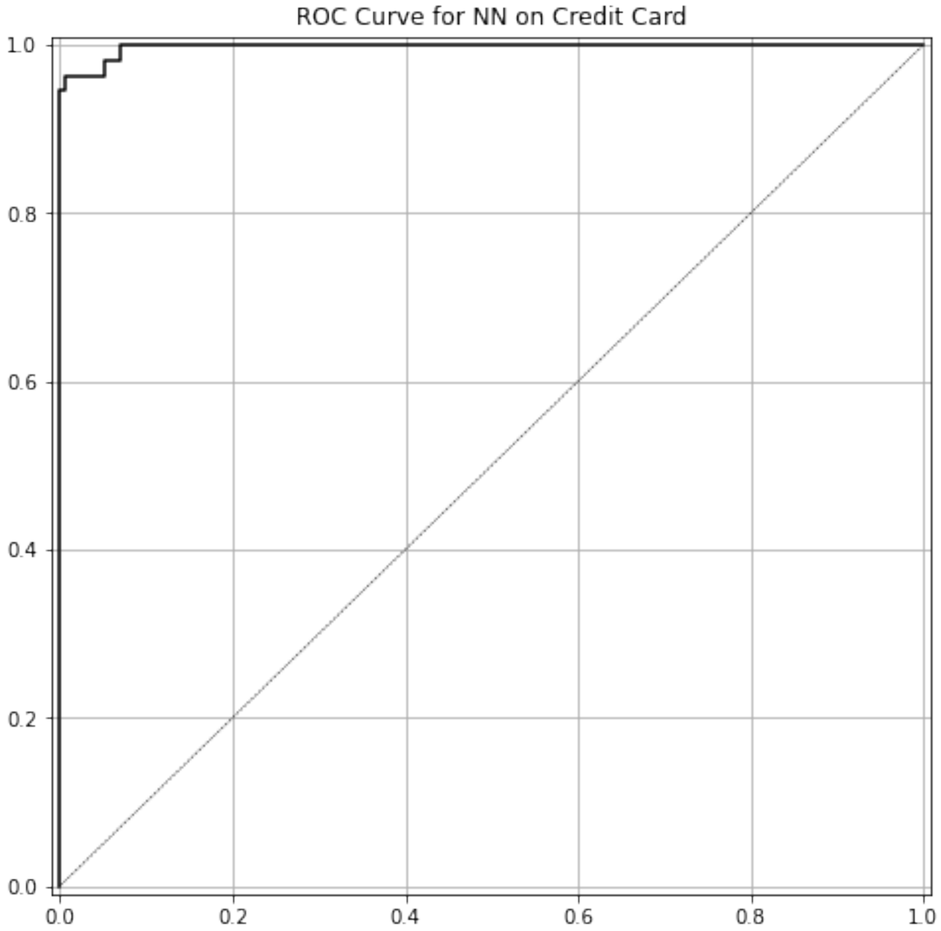
Valori

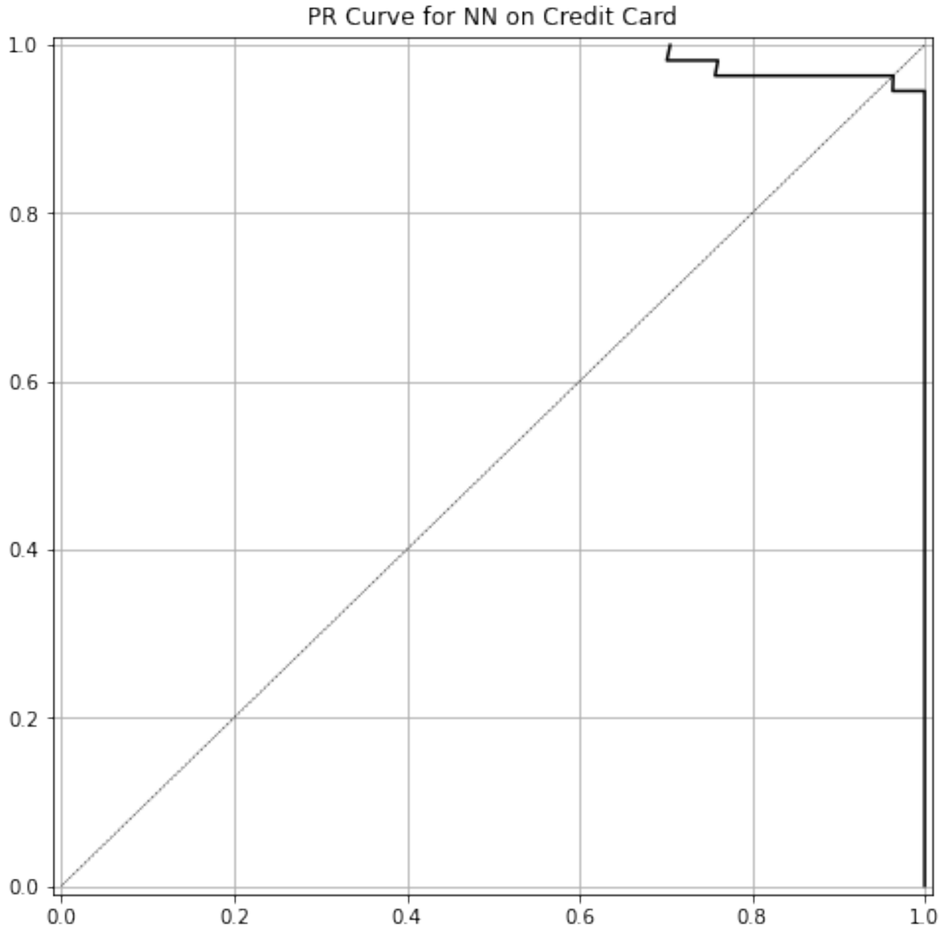
- Sui dati di test il modello ha conseguito i seguenti punteggi:

```
roc-auc is 0.998
average precision score is 0.990
pr-auc is 0.990
loss: 0.0534
```

Valutazione modello

Grafici





Valutazione modello

Considerazioni

- I risultati sui dati di test sono leggermente peggiori rispetto a quelli di validation probabilmente perché dati di training e test potrebbero avere una distribuzione leggermente diversa.

Link ai collab

- https://colab.research.google.com/drive/1HudxDLZ1k-NXTMZD3Y3zT313wzaq3WYY#scrollTo=JZrmbvJQlILl
- https://colab.research.google.com/drive/1fyTYbGvs6LlLq4kgERBmKVi7B22drYf0#scrollTo=rOX5QrQXl67s