TensorPRO (Tensorflow Privacy Remindful Optimization)

Alessio Proietti IN550 Final Exam

Abstract:

L' idea di base è capire se un cliente in un determinato contesto socioeconomico contattato dalla banca sottoscriverà o no un deposito. Il task è di apprendimento supervisionato, le label sono nella colonna 'y'.

Il dataset https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing) è stato preliminarmente esplorato. In una seconda fase è stato standardizzata ogni feature numerica, quelle categoriali sono state codificate con la strategia one hot encoding.

Il dataset era fortemente sbilanciato, nuove istanze per l' allenamento sono state generate con l' algoritmo ADASYN. In conclusione è stata allenata una rete con ottimizzazione ADAM in modalità differential privacy e si sono confrontate delle metriche con la versione non differential private di ADAM.

Fase Esplorativa

In [1]:

```python
# importo alcune libraries di cui avrò bisogno fin dall' inizio
import pandas as pd
import numpy as np

# libs per visualizzazione
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
```

In [2]:

```
df = pd.read_csv('https://raw.githubusercontent.com/alessio-proietti/2021-IN550-EXAM/main/data.csv', sep=';')
df.head(5)
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_w
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

5 rows × 21 columns

In [3]:

```
df.tail(5)
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	d
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	

5 rows × 21 columns

In [4]:

```
# Con questo posso avere un quadro colonne non numeriche
df.describe(include = 'object')
```

Out[4]:

	job	marital	education	default	housing	loan	contact	month	day_of_w
count	41188	41188	41188	41188	41188	41188	41188	41188	41
unique	12	4	8	3	3	3	2	10	
top	admin.	married	university.degree	no	yes	no	cellular	may	
freq	10422	24928	12168	32588	21576	33950	26144	13769	8

In [5]:

```
# Posso estrarre informazioni di base sulle colonne numeriche
df.describe()
```

Out[5]:

	age	duration	campaign	pdays	previous	emp.var.rate	c
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	٠
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	

In [6]:

```
# Stampo i tipi di tutte le variabili per capire con cosa ho a che fare
df.dtypes
```

Out[6]:

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
```

In [7]:

```
# Voglio contare i campi nulli o comunque capire se ce ne sono
print(df.isnull().sum())
```

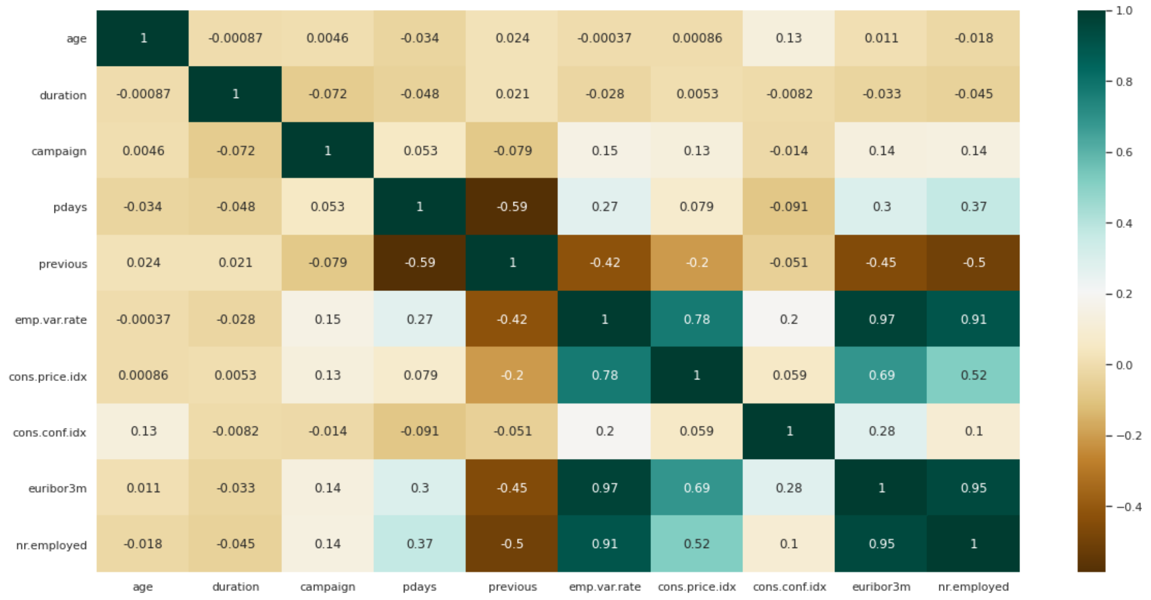
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

In [8]:

```
# È interessante calcolare e avere una visione delle correlazione tra le variabi
li numeriche
plt.figure(figsize=(20,10))
c = df.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c
```

Out[8]:

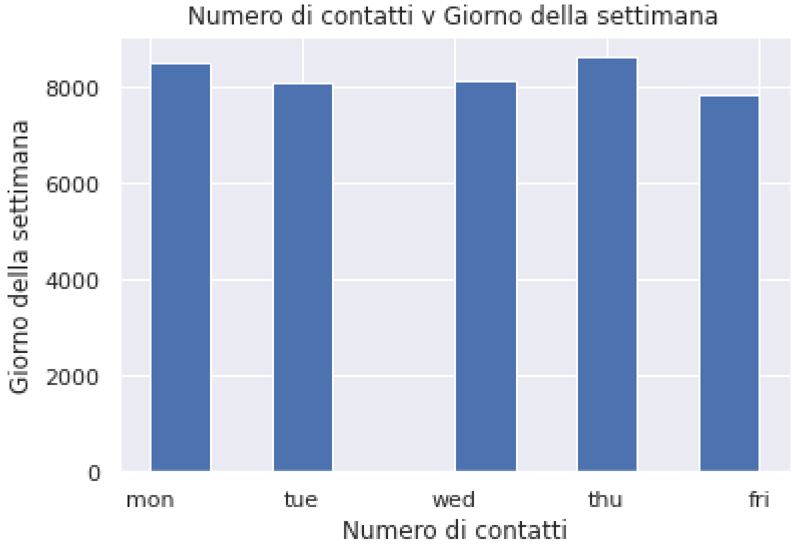
	age	duration	campaign	pdays	previous	emp.var.rate	cons.price
age	1.000000	-0.000866	0.004594	-0.034369	0.024365	-0.000371	0.000
duration	-0.000866	1.000000	-0.071699	-0.047577	0.020640	-0.027968	0.005
campaign	0.004594	-0.071699	1.000000	0.052584	-0.079141	0.150754	0.127
pdays	-0.034369	-0.047577	0.052584	1.000000	-0.587514	0.271004	0.078
previous	0.024365	0.020640	-0.079141	-0.587514	1.000000	-0.420489	-0.203
emp.var.rate	-0.000371	-0.027968	0.150754	0.271004	-0.420489	1.000000	0.775
cons.price.idx	0.000857	0.005312	0.127836	0.078889	-0.203130	0.775334	1.000
cons.conf.idx	0.129372	-0.008173	-0.013733	-0.091342	-0.050936	0.196041	0.058
euribor3m	0.010767	-0.032897	0.135133	0.296899	-0.454494	0.972245	0.688
nr.employed	-0.017725	-0.044703	0.144095	0.372605	-0.501333	0.906970	0.522



In [9]:

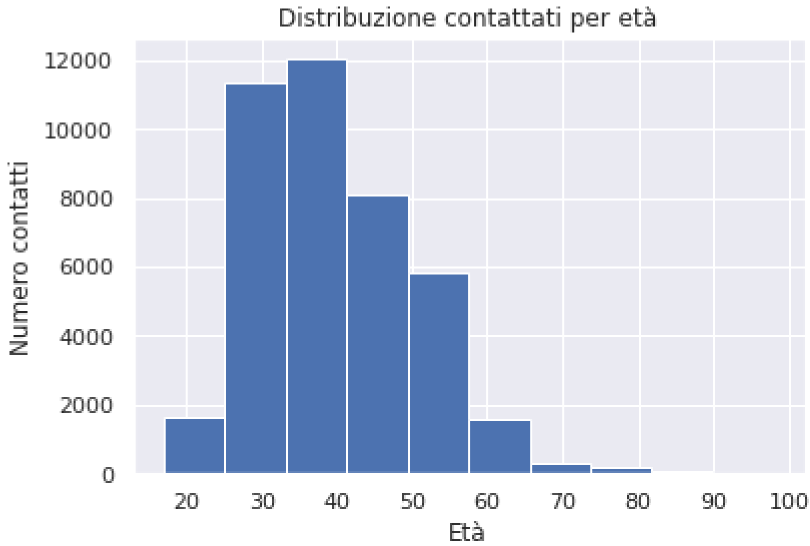
```
# Voglio vedere se la distribuzione dei contatti sulla settimana è uniforme
# TL;DR lo è in buona approssimazione

df['day_of_week'].hist().plot(kind="bar", figsize=(10,5))
plt.title("Numero di contatti v Giorno della settimana")
plt.ylabel('Giorno della settimana')
plt.xlabel('Numero di contatti');
```



In [10]:

```
# Come sono distribuiti contattati per età?
df.age.hist().plot(kind="bar", figsize=(10,5))
plt.title("Distribuzione contattati per età")
plt.ylabel('Numero contatti')
plt.xlabel('Età');
```

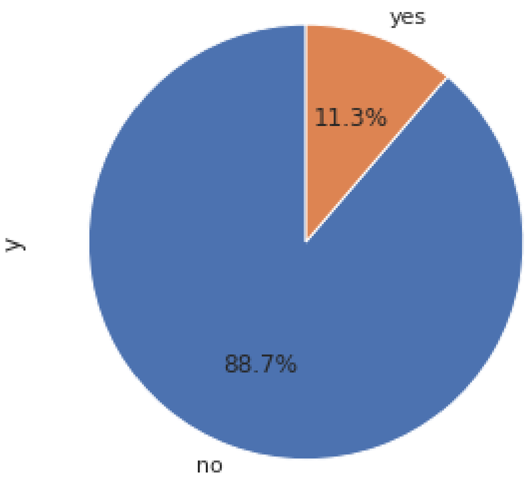


In [11]:

```
# Il dataset è sbilanciato, ecco un grafico a torta per capire le proporzioni
df.y.value_counts().plot(kind='pie', subplots=True,startangle=90,
figsize=(10,5), autopct='%1.1f%%')
```

Out[11]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f94c05fa0
40>],
      dtype=object)
```



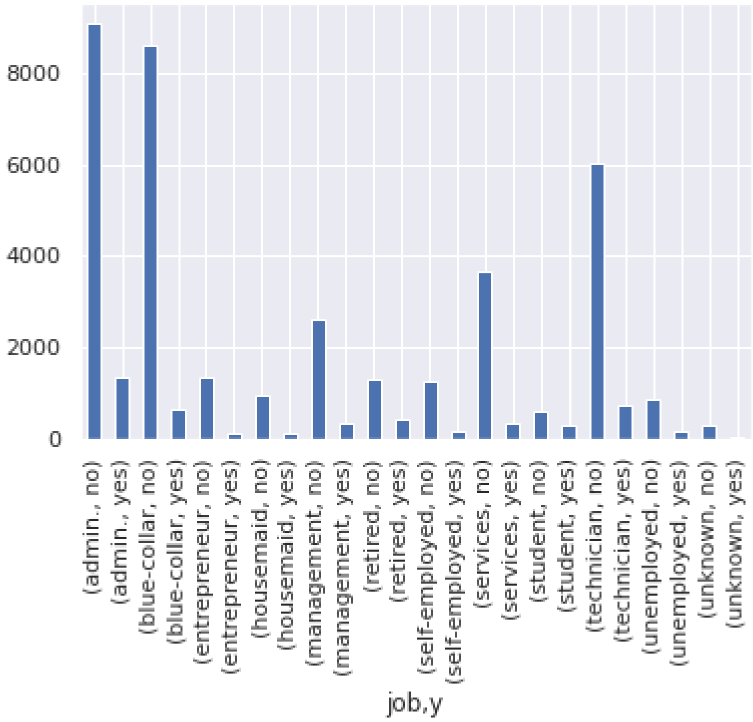
In [12]:

```
# Voglio vedere come cambia il rapporto YES-NO in base al lavoro
# Non sembra in effetti cambiare

df.groupby('job').y.value_counts().plot(kind='bar')
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94c05c53d0>
```



In [13]:

```
# Stampo scatterplot per le variabili numeriche
gsns=sns.pairplot(df, corner=True)

# non sembrano emergere dei trend
```

Preparazione del Dataset

In [14]:

```
# separo le label dal resto dei dati
labels = df[['y']]
df.drop(columns=['y'])
```

Out[14]:

	age	job	marital	education	default	housing	loan	contact	montl
0	56	housemaid	married	basic.4y	no	no	no	telephone	ma
1	57	services	married	high.school	unknown	no	no	telephone	ma
2	37	services	married	high.school	no	yes	no	telephone	ma
3	40	admin.	married	basic.6y	no	no	no	telephone	ma
4	56	services	married	high.school	no	no	yes	telephone	ma
...	...	...	...	...	...	...	...	...	..
41183	73	retired	married	professional.course	no	yes	no	cellular	no
41184	46	blue-collar	married	professional.course	no	no	no	cellular	no
41185	56	retired	married	university.degree	no	yes	no	cellular	no
41186	44	technician	married	professional.course	no	no	no	cellular	no
41187	74	retired	married	professional.course	no	yes	no	cellular	no

41188 rows × 20 columns

In [15]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [16]:

```
# inizializzo gli oggetti che mi permetteranno di codificare le colonne categori
ali
le = LabelEncoder()
ohe = OneHotEncoder()
```

In [17]:

```
#  estraggo le variabili categoriali e mostro un sample, la testa
categorical = df.select_dtypes(include=[object])
categorical.head(6)
```

Out[17]:

	job	marital	education	default	housing	loan	contact	month	day_of_week
0	housemaid	married	basic.4y	no	no	no	telephone	may	mon
1	services	married	high.school	unknown	no	no	telephone	may	mon
2	services	married	high.school	no	yes	no	telephone	may	mon
3	admin.	married	basic.6y	no	no	no	telephone	may	mon
4	services	married	high.school	no	no	yes	telephone	may	mon
5	services	married	basic.9y	unknown	no	no	telephone	may	mon

In [18]:

```
# One Hot Encoding delle variabili categoriali
categorical_le = categorical.apply(le.fit_transform)
categorical_sparse = ohe.fit_transform(categorical_le).toarray()

categorical_encoded = pd.DataFrame(categorical_sparse)
categorical_encoded
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	45	46	47	48	49	50	51	52
0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
41183	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
41184	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
41185	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
41186	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
41187	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.(

41188 rows × 55 columns

In [19]:

```
# Estraggo le variabili numeriche
numerical_not_scaled = df.select_dtypes(include=['float64', 'int64'])
numerical_not_scaled
```

Out[19]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx
0	56	261	1	999	0	1.1	93.994	-36.4
1	57	149	1	999	0	1.1	93.994	-36.4
2	37	226	1	999	0	1.1	93.994	-36.4
3	40	151	1	999	0	1.1	93.994	-36.4
4	56	307	1	999	0	1.1	93.994	-36.4
...	...	...	...	...	...	...	...	...
41183	73	334	1	999	0	-1.1	94.767	-50.8
41184	46	383	1	999	0	-1.1	94.767	-50.8
41185	56	189	2	999	0	-1.1	94.767	-50.8
41186	44	442	1	999	0	-1.1	94.767	-50.8
41187	74	239	3	999	1	-1.1	94.767	-50.8

41188 rows × 10 columns

In [20]:

```
# Per guadagnare performance apporterò una standardizzazione ai dati numerici
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [21]:

```
numerical_ndarray = scaler.fit_transform(numerical_not_scaled)
numerical = pd.DataFrame(numerical_ndarray)
numerical
```

Out[21]:

	0	1	2	3	4	5	6	7
0	1.533034	0.010471	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
1	1.628993	-0.421501	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
2	-0.290186	-0.124520	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
3	-0.002309	-0.413787	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
4	1.533034	0.187888	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
...	...	...	...	...	...	...	...	...
41183	3.164336	0.292025	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41184	0.573445	0.481012	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41185	1.533034	-0.267225	-0.204909	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41186	0.381527	0.708569	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41187	3.260295	-0.074380	0.156105	0.195414	1.671136	-0.752343	2.058168	-2.224953

41188 rows × 10 columns

In [22]:

```
# Riunisco i dati numerici riscalati con la parte codificata delle variabili cat
egoriali
features=numerical.join(categorical_encoded, lsuffix='_caller', rsuffix='_other'
)
features
```

Out[22]:

	0_caller	1_caller	2_caller	3_caller	4_caller	5_caller	6_caller	7_caller
0	1.533034	0.010471	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
1	1.628993	-0.421501	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
2	-0.290186	-0.124520	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
3	-0.002309	-0.413787	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
4	1.533034	0.187888	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.886447
...	...	...	...	...	...	...	...	...
41183	3.164336	0.292025	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41184	0.573445	0.481012	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41185	1.533034	-0.267225	-0.204909	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41186	0.381527	0.708569	-0.565922	0.195414	-0.349494	-0.752343	2.058168	-2.224953
41187	3.260295	-0.074380	0.156105	0.195414	1.671136	-0.752343	2.058168	-2.224953

41188 rows × 65 columns

In [23]:

```
# Trasformo il DataFrame in un ndarray
features_array = features.to_numpy()
features_array
```

Out[23]:

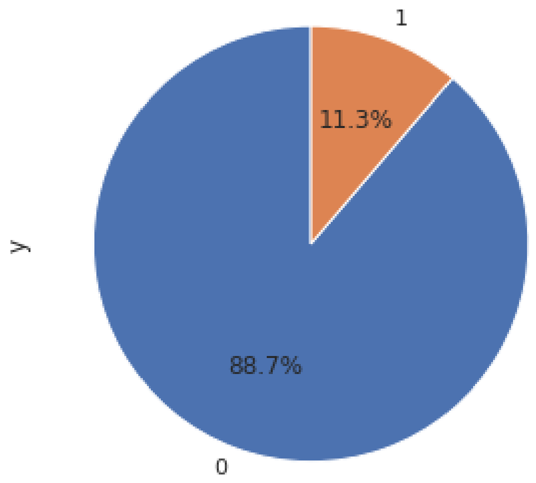
```
array([[ 1.53303429,  0.01047142, -0.56592197, ...,  0.        ,
         1.        ,  0.        ],
       [ 1.62899323, -0.42150051, -0.56592197, ...,  0.        ,
         1.        ,  0.        ],
       [-0.29018564, -0.12451981, -0.56592197, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [ 1.53303429, -0.26722482, -0.20490853, ...,  0.        ,
         1.        ,  0.        ],
       [ 0.38152696,  0.70856893, -0.56592197, ...,  0.        ,
         0.        ,  1.        ],
       [ 3.26029527, -0.07438021,  0.15610492, ...,  0.        ,
         1.        ,  0.        ]])
```

In [24]:

```
# Riporto di nuovo il grafico a torta per la label questa volta codificate con L
inearEncoder
labels_le = labels.apply(le.fit_transform)
labels_le.y.value_counts().plot(kind='pie', subplots=True,startangle=90,
figsize=(10,5), autopct='%1.1f%%')
```

Out[24]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f94b810ed
f0>],
      dtype=object)
```



In [25]:

```
# Divido il dataset nelle istanze di training e di test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features_array, labels_le, t
est_size=0.5)
```

ADASYN Oversampling

In [26]:

```
# SOLO sulle istanze di training agirò con un algoritmo di Oversampling, ADASYN
from imblearn.over_sampling import ADASYN
oversample = ADASYN()
```

In [27]:

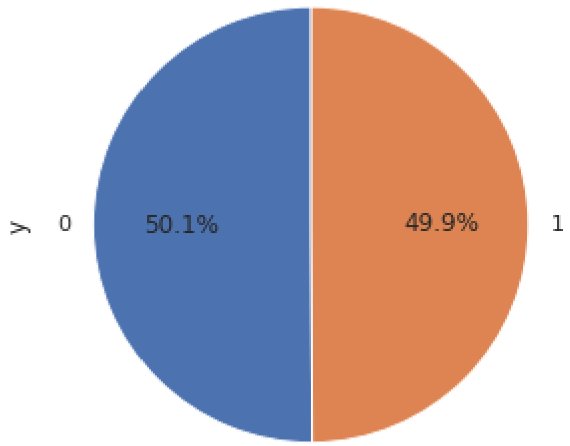
```
# Resample
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

In [28]:

```
# Come è cambiata la distribuzione?
y_train.y.value_counts().plot(kind='pie', subplots=True,startangle=90,
figsize=(10,5), autopct='%1.1f%%')
```

Out[28]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f94b09a54
60>],
      dtype=object)
```

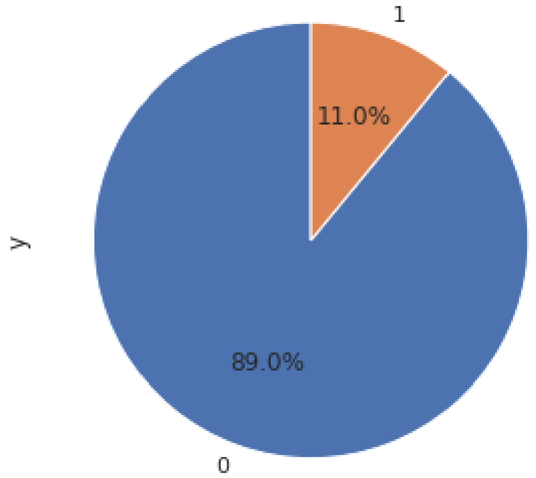


In [29]:

```
# Sul test set le proporzioni sono quelle originarie...
y_test.y.value_counts().plot(kind='pie', subplots=True,startangle=90,
figsize=(10,5), autopct='%1.1f%%')
```

Out[29]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f94b096d5
50>],
      dtype=object)
```



In [30]:

```
# Porto training set e test set in formato ndarray
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
print(type(y_train), type(y_test), type(X_train), type(X_test))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarra
y'> <class 'numpy.ndarray'>
```

A Machine is Learning ...

In [31]:

```
# È necessario avere TF >= 2 per utilizzare il pacchetto tensorflow-privacy
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import tensorflow as tf
print(tf.__version__)
```

```
2.3.1
```

In [32]:

```
# Definisco l' architettura dei una rete neurale "profonda" che alleneremo
model = Sequential([
  Dense(20, activation='relu', input_shape=(65,)),
  Dropout(.3),
  Dense(20, activation='relu'),
  Dropout(.3),
  Dense(20, activation='relu'),
  Dropout(.3),
  Dense(20, activation='relu'),
  Dense(1, activation='sigmoid'),
])
```

In [33]:

```
# Adam Optimizer con privacy learning
from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import DPKerasAdam
Optimizer
optimizer = DPKerasAdamOptimizer(
    l2_norm_clip=1.5,
    noise_multiplier=1.3,
    num_microbatches=1033,
    learning_rate=0.025)
```

In [34]:

```
model.compile(
  optimizer=optimizer,
  loss='binary_crossentropy',
  metrics=['accuracy','Precision', 'Recall']
)
```

In [35]:

```
history_dpadam=model.fit(
  X_train,
  y_train,
  epochs=10,
  validation_split=0.33,
  batch_size=24,
)
```

```
Epoch 1/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
328 - accuracy: 0.9887 - precision: 0.9840 - recall: 0.9709 - val_lo
ss: 8.6755e-07 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 2/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
237 - accuracy: 0.9954 - precision: 0.9888 - recall: 0.9932 - val_lo
ss: 7.6175e-10 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 3/10
1016/1016 [==============================] - 2s 1ms/step - loss: 0.0
025 - accuracy: 0.9994 - precision: 0.9982 - recall: 0.9993 - val_lo
ss: 1.9401e-12 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 4/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
348 - accuracy: 0.9975 - precision: 0.9926 - recall: 0.9974 - val_lo
ss: 0.0016 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_reca
ll: 1.0000
Epoch 5/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
652 - accuracy: 0.9867 - precision: 0.9531 - recall: 0.9966 - val_lo
ss: 5.1184e-06 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 6/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
130 - accuracy: 0.9960 - precision: 0.9849 - recall: 0.9993 - val_lo
ss: 7.6155e-15 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 7/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
106 - accuracy: 0.9982 - precision: 0.9937 - recall: 0.9992 - val_lo
ss: 6.2666e-19 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 8/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
090 - accuracy: 0.9973 - precision: 0.9900 - recall: 0.9992 - val_lo
ss: 5.7273e-10 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 9/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
046 - accuracy: 0.9988 - precision: 0.9955 - recall: 0.9997 - val_lo
ss: 1.0318e-23 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 10/10
1016/1016 [==============================] - 2s 1ms/step - loss: 0.0
075 - accuracy: 0.9984 - precision: 0.9940 - recall: 0.9995 - val_lo
ss: 7.0661e-24 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
```

In [36]:

```
print("Valuto il modello sui dati di test")
results = model.evaluate(X_test, y_test, batch_size=48)
print("test loss, test acc:", results)


print("Genero 10 predizioni")
predictions = model.predict(X_test[:10])
print("predizioni: \n", predictions)
y_test[:10]
```

```
Valuto il modello sui dati di test
430/430 [==============================] - 0s 804us/step - loss: 2.9
926e-20 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
test loss, test acc: [2.992560772279845 6e-20, 1.0, 1.0, 1.0]
Genero 10 predizioni
predizioni:
 [[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]]
```

Out[36]:

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0]])
```

Sui dati di test il modello raggiunge accuracy, precision, recall 1. È incoraggiante ma apre domande sulla correttezza metodologica della ricerca.

In [37]:

```
# Adam Optimizer con privacy learning
model.compile(
  optimizer='adam',
  loss='binary_crossentropy',
  metrics=['accuracy','Precision', 'Recall']
)
```

In [38]:

```
history_adam = model.fit(
  X_train,
  y_train,
  epochs=10,
  validation_split=0.33,
  batch_size=24,
)
```

```
Epoch 1/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
065 - accuracy: 0.9977 - precision: 0.9910 - recall: 0.9998 - val_lo
ss: 4.6750e-32 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 2/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
094 - accuracy: 0.9982 - precision: 0.9935 - recall: 0.9995 - val_lo
ss: 3.9171e-31 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 3/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
047 - accuracy: 0.9984 - precision: 0.9935 - recall: 1.0000 - val_lo
ss: 1.5085e-32 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 4/10
1016/1016 [==============================] - 1s 1ms/step - loss: 0.0
041 - accuracy: 0.9989 - precision: 0.9956 - recall: 0.9998 - val_lo
ss: 1.0664e-36 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 5/10
1016/1016 [==============================] - 2s 1ms/step - loss: 0.0
043 - accuracy: 0.9986 - precision: 0.9947 - recall: 1.0000 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 6/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
053 - accuracy: 0.9983 - precision: 0.9934 - recall: 1.0000 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 7/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
026 - accuracy: 0.9993 - precision: 0.9972 - recall: 1.0000 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 8/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
082 - accuracy: 0.9990 - precision: 0.9963 - recall: 0.9998 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 9/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
033 - accuracy: 0.9989 - precision: 0.9960 - recall: 0.9998 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
Epoch 10/10
1016/1016 [==============================] - 2s 2ms/step - loss: 0.0
115 - accuracy: 0.9992 - precision: 0.9971 - recall: 0.9997 - val_lo
ss: 0.0000e+00 - val_accuracy: 1.0000 - val_precision: 1.0000 - val_
recall: 1.0000
```

In [39]:

```
print("Valuto il modello sui dati di test")
results = model.evaluate(X_test, y_test, batch_size=48)
print("test loss, test acc:", results)


print("Genero 10 predizioni")
predictions = model.predict(X_test[:10])
print("predizioni: \n", predictions)
y_test[:10]
```

```
Valuto il modello sui dati di test
430/430 [==============================] - 0s 803us/step - loss: 7.9
621e-37 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
test loss, test acc: [7.962126754925632e-37, 1.0, 1.0, 1.0]
Genero 10 predizioni
predizioni: 
 [[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]]
```

Out[39]:

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0]])
```

Conclusioni

Si sarebbe potuto ipotizzare che usare usare una versione di ADAM dove SGD è sostituito da DP-SGD potesse impattarne le performance, almeno nel caso in esame ciò non sembra particolarmente vero anche se il valore della funzione di costo dopo all' ultima epoca è più alto.

Riporto il sommario di tutti gli eventi nel training dei due modelli

In [40]:

```
history_dpadam.history
```

Out[40]:


```
{'loss': [0.032797202467918396,
  0.023743323981761932,
  0.0024960124865174294,
  0.03479241579771042,
  0.06524933874607086,
  0.013013748452067375,
  0.010589821264147758,
  0.009014119394123554,
  0.004550587851554155,
  0.007533054798841476 4],
 'accuracy': [0.9886722564697266,
  0.9954442977905273,
  0.999384343624115,
  0.997455358505249,
  0.9867432713508606,
  0.9959778189659119,
  0.99819415807724,
  0.9972501397132874,
  0.9987687468528748,
  0.9983583092689514],
 'precision': [0.9840223789215088,
  0.9888349771499634,
  0.9982143044471741,
  0.9925602674484253,
  0.9530618786811829,
  0.984943151473999,
  0.9936964511871338,
  0.9900161027908325,
  0.9954684972763062,
  0.9940197467803955],
 'recall': [0.970908522605896,
  0.9931740760803223,
  0.9993498921394348,
  0.997399628162384,
  0.9965870380401611,
  0.9993498921394348,
  0.9991874098777771,
  0.9991874098777771,
  0.9996749758720398,
  0.9995124340057373],
 'val_loss': [8.675520462020359e-07,
  7.617529385051114e-10,
  1.9401481289593736e-12,
  0.001569235697388649,
  5.118351054989034e-06,
  7.61548942261386e-15,
  6.266587971803729e-19,
  5.727286978007839e-10,
  1.031805530454931e-23,
  7.06608666330242e-24],
 'val_accuracy': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
 'val_precision': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0],
 'val_recall': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]}
```

In [41]:

```
history_adam.history
```

Out[41]:

```
{'loss': [0.006491903215646744,
  0.009353882633149624,
  0.0046739354729652405,
  0.004108505789190531,
  0.004250656813383102,
  0.0053377775475382805,
  0.0025714579969644547,
  0.008214999921619892,
  0.00332126603461802,
  0.01148801390081644],
 'accuracy': [0.9976605772972107,
  0.9982351660728455,
  0.9983583092689514,
  0.9988508224487305,
  0.9986456036567688,
  0.9983172416687012,
  0.9993022680282593,
  0.9990149736404419,
  0.9989328980445862,
  0.9991791248321533],
 'precision': [0.9909793734550476,
  0.993537962436676,
  0.9935411214828491,
  0.9956303834915161,
  0.9946653842926025,
  0.9933806657791138,
  0.9972447156906128,
  0.9962753057479858,
  0.9959527254104614,
  0.9970821738243103],
 'recall': [0.9998374581336975,
  0.9995124340057373,
  1.0,
  0.9998374581336975,
  1.0,
  1.0,
  1.0,
  0.9998374581336975,
  0.9998374581336975,
  0.9996749758720398],
 'val_loss': [4.675048003206162e-32,
  3.9170506895812443e-31,
  1.5085249777848945e-32,
  1.0664158434296163e-36,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0],
 'val_accuracy': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
 'val_precision': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0],
 'val_recall': [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]}
```