MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Avaliação Final

Aluno: Benício Ramos Magalhães

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Nessa avaliação será utilizado o dataset `creditcard.csv` que contém 31 colunas. Esse problema é o de detectar fraude em transações em cartões de crédito. Vamos assumir um cenário com alta disponibilidade de exemplos não rotulados, e baixa de exemplos rotulados. Para tal, pré-treinaremos camadas de uma rede neural com dados não anotados, a qual posteriormente será usada para compor um modelo inicial de classificação.

Você deverá criar um notebook (ipynb) com a solução. O Notebook deverá conter as saídas para visualização dos resultados. Fazer upload no Moodle de 2 arquivos:

1. Notebook com o código fonte (ipynb)
2. Versão em PDF com todos os resultados

Conforme código abaixo, use como características de entrada as colunas de posição 1 até 28 (marcadas no arquivo como V1 - V28), e como classe a última coluna (Class). Não utilize a coluna Amount.

As tarefas a realizar são as seguintes:

1. **Separe** os dados em:
   - conjunto S = 2,5% dos dados iniciais como treinamento com rótulo (assumiremos que temos rótulos apenas para esses 2,5%, ou 7120 exemplos), no formato par (x,y)
   - conjunto U = 50% dos dados iniciais como treinamento não anotado (note que S está contido em U),
   - conjunto T = o restante dos 50% para teste, no formato par (x,y).
2. **Modelo A**: denoising overcomplete autoencoder para pré-treinamento baseado em auto-supervisão
   - Arquitetura com as seguintes camadas:
     - entrada com 28 valores
     - normalização em batch
     - densa 32 neurônios, relu
     - densa 32 neurônios, relu
     - dropout 0.2
     - normalização em batch
     - densa 28 neurônios, relu (camada de código/bottleneck)
     - densa 32 neurônios, relu
     - densa 32 neurônios, relu
     - densa 28 neurônios, tanh
   - Inserção de ruído aleatório uniforme ponderado a 0.2 (insira ruído nos dados de treinamento fornecidos por entrada, mas mantenha a comparação com a saída sem ruído, como num denoising autoencoder)
   - Taxa de aprendizado inicial de 0.003 e com decaimento a partir da época 5, exponencial a -0.2
   - Treinar com perda MSE por 20 épocas com batch size 16 utilizando o conjunto **U**
3. **Análise de projeção das características**: visualize um scatterplot com os 2 principais componentes obtidos do PCA com as classes dos exemplos atribuídas com cores ou marcadores diferentes:
   - scatterplot com projeção PCA do conjunto de S original
   - scatterplot com projeção PCA do conjunto S após processado pelo "encoder", ou seja resultado da saída da camada de código
4. **Modelo B**: rede neural profunda densa, utilizando como base o encoder do modelo A, e inserindo uma nova camada densa de classificação com ativação sigmóide.
   - Taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas, exponencial a -0.3
   - Uso de pesos para as classes: 0.1 para classe 0 (majoritária), e 0.9 para a classe 1 (minoritária)
   - Treinar com perda MSE por 8 épocas com batch size 16
   - Compute como métricas, além da perda, precisão e revocação (precision / recall)
5. **Avalie a rede neural de classificação**:
   - Exiba o gráfico da precisão e revocação no treinamento calculada ao longo das épocas
   - Exiba precisão e revocação calculada no treinamento S e teste T
   - Exiba um scatterplot do conjunto S obtendo sua representação do código da rede de classificação (saída da camada com 28 exemplos)
6. **Bônus:** (+1 ponto extra) compare a solução com duas outras possibilidades que não envolvam uso do conjunto **U** não rotulado
   - Rede neural profunda com a mesma arquitetura e estratégias usadas no modelo B, mas sem usar os pesos pré-treinados, inicializando e treinando com os dados em S por 15 épocas. Avalie precisão e revocação no treinamento S e teste T.
   - Classificador SVM treinado nos dados originais S. Avalie precisão e revocação no treinamento S e teste T.
   - Classificador SVM treinado nos dados S obtendo sua representação do código da rede de classificação (modelo B). Avalie precisão e revocação no treinamento S e teste T.

In [1]:
```
#realizando as importações

import random
import numpy as np
import seaborn as sns
import tensorflow as tf
from sklearn.svm import SVC
from tensorflow import keras
from numpy.random import seed
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from sklearn.decomposition import PCA
from tensorflow.random import set_seed
from keras.models import Model, Sequential
from sklearn.metrics import precision_score, recall_score
from keras.layers import Input, Conv2D, AveragePooling2D, Flatten, Reshape, UpSampling2D, Conv2DTranspo
se, Dense, BatchNormalization, Dropout
```

In [2]:
```
#lendo dataframe
import pandas as pd
df = pd.read_csv("creditcard.csv")
df
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431
...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057

284807 rows × 31 columns

Parte 1: separar dados

In [3]:
```
#removendo coluna Amount
df = df.drop('Amount',axis=1)

#obtendo conjunto S = 2,5% dos dados iniciais como treinamento com rótulo
#assumiremos que temos rótulos apenas para esses 2,5%, ou 7120 exemplos), no formato par (x,y)
Sx, Sy = df.iloc[0:7120, 1:-1], df[["Class"]].iloc[0:7120]

#conjunto U = 50% dos dados iniciais como treinamento não anotado (note que S está contido em U),
U = df.iloc[0:int(len(df)/2), 1:-1]

#conjunto T = o restante dos 50% para teste, no formato par (x,y).
Tx, Ty = df.iloc[int(len(df)/2):, 1:-1],df[['Class']].iloc[int(len(df)/2):]

print('Tamanho das bases:')
print()
print('Sx:',len(Sx))
print('Sy:',len(Sy))
print('U :',len(U))
print('Tx:',len(Tx))
print('Ty:',len(Ty))
```

```
Tamanho das bases:

Sx: 7120
Sy: 7120
U : 142403
Tx: 142404
Ty: 142404
```

Parte 2: Modelo A

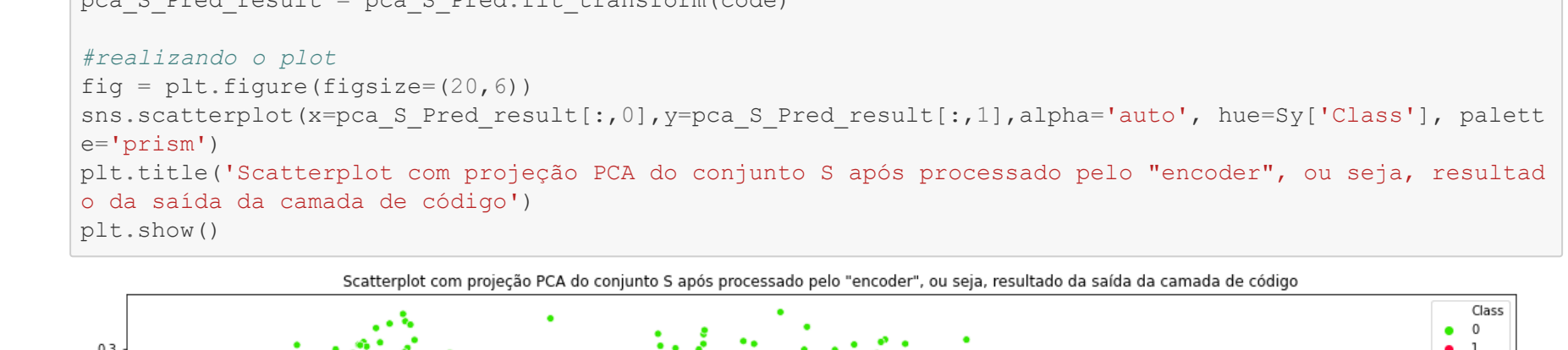
In [4]:
```
#modelo A: denoising overcomplete autoencoder para pré-treinamento baseado em auto-supervisão
def denoising_over_AE(input_shape):
    input = Input(shape=(input_shape))
    #encoder:
    encoder = BatchNormalization()(input)
    encoder = Dense(32, activation='relu')(encoder)
    encoder = Dense(32, activation='relu')(encoder)
    encoder = Dropout(0.2)(encoder)
    encoder = BatchNormalization()(encoder)
    encoder = Dense(28, activation='relu',name='code')(encoder)
    #decoder
    decoder = Dense(32,activation='relu',name='input_decoder')(encoder)
    decoder = Dense(32,activation='relu')(decoder)
    decoder = Dense(28,activation='tanh')(decoder)
    #autoencoder
    autoencoder = keras.models.Model(input, decoder)
    autoencoder.summary()
    return autoencoder

#inserção de ruído aleatório uniforme ponderado a 0.2
#(insira ruído nos dados de treinamento fornecidos por entrada,
# mas mantenha a comparação com a saída sem ruído, como num denoising autoencoder)
noiseFactor = 0.2
SxNoise = Sx + noiseFactor * np.random.normal(0,1,Sx.shape)
UNoise = U + noiseFactor * np.random.normal(0,1,U.shape)

#sementes:
seed(1)
set_seed(2)

#taxa de aprendizado inicial de 0.003 e com decaimento a partir da época 5, exponencial a -0.2
lr = 0.003

def scheduler_A (epoch, lr):
    if epoch < 5:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.2),4)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler_A)

#Treinar com perda MSE por 20 épocas com batch size 16 utilizando o conjunto U
batch_size = 16
epochs = 20

modelo_A = denoising_over_AE(28)

modelo_A.compile(loss='mse',
                 optimizer=keras.optimizers.Adam(lr=lr))

historyDenoising = modelo_A.fit(x=UNoise, y=U,
                        epochs=epochs,
                        batch_size = batch_size,
                        callbacks=[callbacklr],
                        verbose=1)
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28)]              0
_________________________________________________________________
batch_normalization (BatchNo (None, 28)                112
_________________________________________________________________
dense (Dense)                (None, 32)                928
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32)                128
_________________________________________________________________
code (Dense)                 (None, 28)                924
_________________________________________________________________
input_decoder (Dense)        (None, 32)                928
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 28)                924
=================================================================
Total params: 6,056
Trainable params: 5,936
Non-trainable params: 120
_________________________________________________________________
Epoch 1/20
8901/8901 [==============================] - 17s 2ms/step - loss: 0.6070
Epoch 2/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5477
Epoch 3/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5381
Epoch 4/20
8901/8901 [==============================] - 17s 2ms/step - loss: 0.5337
Epoch 5/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5312
Epoch 6/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5267
Epoch 7/20
8901/8901 [==============================] - 16s 2ms/step - loss: 0.5215
Epoch 8/20
8901/8901 [==============================] - 17s 2ms/step - loss: 0.5190
Epoch 9/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5166
Epoch 10/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5146
Epoch 11/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5127
Epoch 12/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5116
Epoch 13/20
8901/8901 [==============================] - 16s 2ms/step - loss: 0.5100
Epoch 14/20
8901/8901 [==============================] - 18s 2ms/step - loss: 0.5088
Epoch 15/20
8901/8901 [==============================] - 14s 2ms/step - loss: 0.5082
Epoch 16/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5073
Epoch 17/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5066
Epoch 18/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5067
Epoch 19/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5064
Epoch 20/20
8901/8901 [==============================] - 15s 2ms/step - loss: 0.5063
```

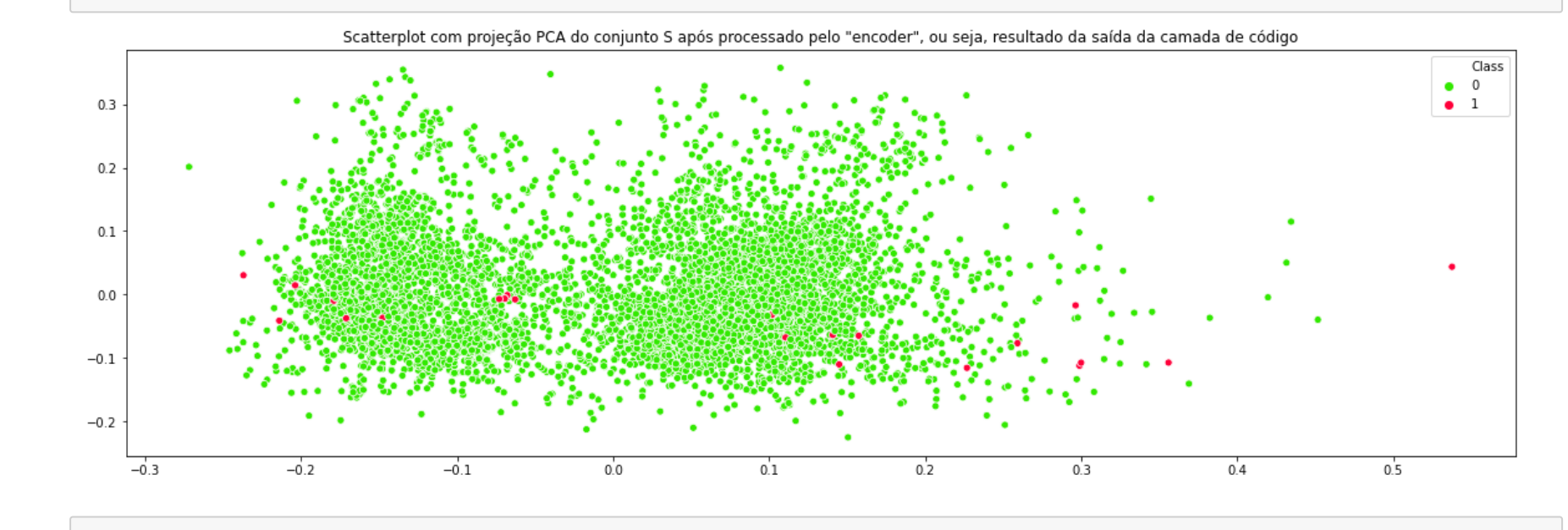
Parte 3: Análise da projeção das características

In [5]:
```
#função de normalização, para aplicar a normalização nos dados para melhor visualização e comparação do
s gráficos
def normalize(train):
    d_max = np.max(train)
    d_min = np.min(train)
    return (train - d_min) / (d_max-d_min)

#normalizando o dado original Sx (sem ruído)
SxNorm = normalize(Sx)

#apenas para curiosidade, fiz uma simulação normalizando o dado Sx com ruído, mas no final,
#o resultado apresentado é sem ruído
#SxNorm = normalize(SxNoise)
```

In [6]:
```
#realizando projeção PCA do conjunto original S
pca_S_Orig = PCA(n_components=2, random_state=1)
pca_S_Orig_result = pca_S_Orig.fit_transform(SxNorm)

#apenas para curiosidade, fiz um teste sem normalização para ver como ficaria o gráfico,
#mas no final, o resultado do gráfico está normalizado para melhor comparação dos resultados
#pca_S_Orig_result = pca_S_Orig.fit_transform(Sx)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_S_Orig_result[:,0],y=pca_S_Orig_result[:,1],alpha='auto', hue=Sy['Class'], palett
e='prism')
plt.title('Scatterplot com projeção PCA do conjunto de S original')
plt.show()
```



In [7]:
```
#extraindo a camada de código do modelo A para predição
extract = keras.models.Model(modelo_A.inputs, modelo_A.layers[-4].output)
code = extract.predict(SxNorm)

#apenas para curiosidade, fiz um teste sem normalização para ver como ficaria o gráfico,
#mas no final, o resultado do gráfico está normalizado para melhor comparação dos resultados
#code = extract.predict(Sx)

#realizando projeção PCA do conjunto S processado pelo encoder (camada de código)
pca_S_Pred = PCA(n_components=2, random_state=1)
pca_S_Pred_result = pca_S_Pred.fit_transform(code)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_S_Pred_result[:,0],y=pca_S_Pred_result[:,1],alpha='auto', hue=Sy['Class'], palett
e='prism')
plt.title('Scatterplot com projeção PCA do conjunto S após processado pelo "encoder", ou seja, resultad
o da saída da camada de código')
plt.show()
```



In [8]:
```
#verificando o balanceamento dos dados, apenas por curiosidade, pois nos gráficos tivemos poucos pontos
 da classe 1
#este item não faz parte das questões da prova.
mask0 = (Sy['Class'] == 0)
mask1 = (Sy['Class'] == 1)

print("Classe Majoritária: ", len(Sy[mask0].index.values)/len(Sy) * 100,'%')
print("Classe Minoritária: ", len(Sy[mask1].index.values)/len(Sy) * 100,'%')
```

```
Classe Majoritária:  99.64887640449437 %
Classe Minoritária:  0.35112359550561795 %
```

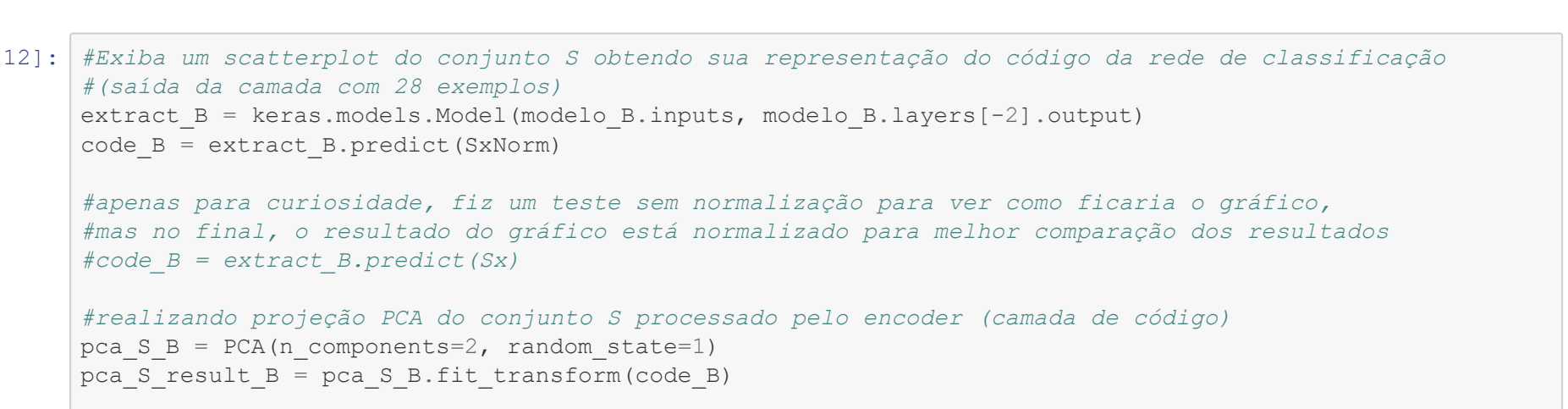
Parte 4: Modelo B

In [9]:
```
#modelo B: rede neural profunda densa, utilizando como base o encoder do modelo A,
#e inserindo uma nova camada densa de classificação com ativação sigmóide.

#obtendo saída da camada de código do modelo A
base_saida = modelo_A.layers[-4].output

#criando nova camada de saída que recebe a anterior
saida_nova = Dense(1, activation='sigmoid')(base_saida)

#modelo B, com a nova camada de saída
modelo_B = keras.models.Model(modelo_A.inputs, saida_nova)
modelo_B.summary()

#sementes:
seed(1)
set_seed(2)

#taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas, exponencial a -0.3
lr = 0.001

def scheduler_B (epoch, lr):
    if epoch < 1:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.3),4)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler_B)

#ponderar o total de cada classe e formar o peso, pois as mesmas estão muito desbalanceadas
#Uso de pesos para as classes: 0.1 para classe 0 (majoritária), e 0.9 para a classe 1 (minoritária)
peso_0 = 0.1
peso_1 = 0.9
class_weight = {0: peso_0, 1: peso_1}

#Treinar com perda MSE por 8 épocas com batch size 16
batch_size = 16
epochs = 8

#Compute como métricas, além da perda, precisão e revocação (precision / recall)
metrics = [
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
]

#compilando o modelo B
modelo_B.compile(loss='mse',
                 optimizer=keras.optimizers.Adam(lr=lr),
                 metrics=metrics)

#como não foi comentado nada nas questões referentes a este item, por padrão, fiz o treinamento com os
 dados sem ruído
#sem ruído
history = modelo_B.fit(x=Sx, y=Sy,
                       epochs = epochs,
                       batch_size = batch_size,
                       callbacks = [callbacklr],
                       class_weight = class_weight,
                       verbose=1)

#com ruído
#history = modelo_B.fit(x=SxNoise, y=Sy,
#                       epochs = epochs,
#                       batch_size = batch_size,
#                       callbacks = [callbacklr],
#                       class_weight = class_weight,
#                       verbose=1)
```

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28)]              0
_________________________________________________________________
batch_normalization (BatchNo (None, 28)                112
_________________________________________________________________
dense (Dense)                (None, 32)                928
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32)                128
_________________________________________________________________
code (Dense)                 (None, 28)                924
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 29
=================================================================
Total params: 3,177
Trainable params: 3,057
Non-trainable params: 120
_________________________________________________________________
Epoch 1/8
445/445 [==============================] - 1s 2ms/step - loss: 0.0035 - precision: 0.1111 - recall:
0.8800
Epoch 2/8
445/445 [==============================] - 1s 2ms/step - loss: 5.7737e-04 - precision: 0.8148 - recal
l: 0.8800
Epoch 3/8
445/445 [==============================] - 1s 2ms/step - loss: 3.9999e-04 - precision: 0.9200 - recal
l: 0.9200
Epoch 4/8
445/445 [==============================] - 1s 2ms/step - loss: 3.9380e-04 - precision: 0.8846 - recal
l: 0.9200
Epoch 5/8
445/445 [==============================] - 1s 2ms/step - loss: 3.4993e-04 - precision: 0.8846 - recal
l: 0.9200
Epoch 6/8
445/445 [==============================] - 1s 2ms/step - loss: 2.3740e-04 - precision: 0.8571 - recal
l: 0.9600
Epoch 7/8
445/445 [==============================] - 1s 2ms/step - loss: 2.9949e-04 - precision: 0.8846 - recal
l: 0.9200
Epoch 8/8
445/445 [==============================] - 1s 2ms/step - loss: 3.1542e-04 - precision: 0.8519 - recal
l: 0.9200
```

Parte 5: Avaliação da rede neural de classificação

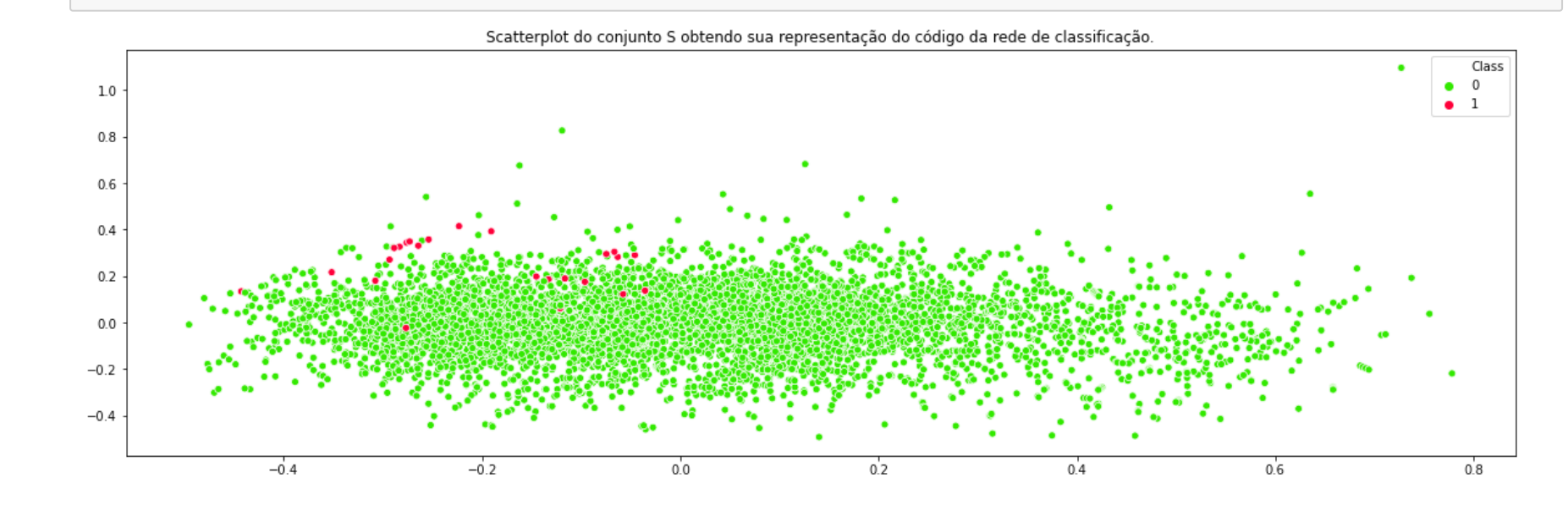
In [10]:
```
#Exiba o gráfico da precisão e revocação no treinamento calculada ao longo das épocas
fig = plt.figure(figsize=(20,6))
fig = plt.plot(history.history['precision'],'b')
fig = plt.plot(history.history['recall'], color='tab:orange', linestyle='--')
fig = plt.title('Gráfico de Precisão e Revocação Calculado ao Longo das Épocas')
fig = plt.xlabel('Épocas')
fig = plt.legend(["precision","recall"], loc="upper left")
```



In [11]:
```
#Exiba precisão e revocação calculada no treinamento S e teste T
score_S = modelo_B.evaluate(Sx, Sy, verbose = 0)
score_T = modelo_B.evaluate(Tx, Ty, verbose = 0)
print("S: Precisão = %.4f, Revocação = %.10f" % (score_S[1], score_S[2]))
print("T: Precisão = %.4f, Revocação = %.10f" % (score_T[1], score_T[2]))
```

```
S: Precisão = 0.8889, Revocação = 0.9599999785
T: Precisão = 0.8522, Revocação = 0.7757847309
```

In [12]:
```
#Exiba um scatterplot do conjunto S obtendo sua representação do código da rede de classificação
#(saída da camada com 28 exemplos)
extract_B = keras.models.Model(modelo_B.inputs, modelo_B.layers[-2].output)
code_B = extract_B.predict(SxNorm)

#apenas para curiosidade, fiz um teste sem normalização para ver como ficaria o gráfico,
#mas no final, o resultado do gráfico está normalizado para melhor comparação dos resultados
#code_B = extract_B.predict(Sx)

#realizando projeção PCA do conjunto S processado pelo encoder (camada de código)
pca_S_B = PCA(n_components=2, random_state=1)
pca_S_result_B = pca_S_B.fit_transform(code_B)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_S_result_B[:,0],y=pca_S_result_B[:,1],alpha='auto', hue=Sy['Class'], palette='pri
sm')
plt.title('Scatterplot do conjunto S obtendo sua representação do código da rede de classificação.')
plt.show()
```



Bônus: Tentando outros métodos para treinar com os poucos dados rotulados

Tentativa 1: DNN com a mesma arquitetura usada, mas sem pré-treinamento

In [13]:
```
#modelo DNN com a mesma arquitetura usada no modelo B, mas sem pré-treinamento
def dnn(input_shape):
    input = Input(shape=(input_shape))
    #encoder:
    encoder = BatchNormalization()(input)
    encoder = Dense(32, activation='relu')(encoder)
    encoder = Dense(32, activation='relu')(encoder)
    encoder = Dropout(0.2)(encoder)
    encoder = BatchNormalization()(encoder)
    encoder = Dense(28, activation='relu',name='code')(encoder)
    #decoder
    decoder = Dense(1,activation="sigmoid")(encoder)
    dnn = keras.models.Model(input,decoder)
    dnn.summary()
    return dnn

#modelo DNN:
modelo_DNN = dnn(28)

#Treinando com os dados em S por 15 épocas e usando as MESMAS ESTRATÉGIAS do modelo B.

#sementes:
seed(1)
set_seed(2)

#taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas, exponencial a -0.3
lr = 0.001

def scheduler_BNN (epoch, lr):
    if epoch < 1:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.3),4)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler_BNN)

#ponderaração
peso_0 = 0.1
peso_1 = 0.9
class_weight = {0: peso_0, 1: peso_1}

#Treinar com perda MSE por 15 épocas com batch size 16
batch_size = 16
epochs = 15

#Compute como métricas, além da perda, precisão e revocação (precision / recall)
metrics = [
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
]

#compilando o modelo
modelo_DNN.compile(loss='mse',
                   optimizer=keras.optimizers.Adam(lr=lr),
                   metrics=metrics)

#como não foi comentado nada nas questões referentes a este item, por padrão, fiz o treinamento com os
 dados sem ruído
#sem ruído
historyDNN = modelo_DNN.fit(x=Sx, y=Sy,
                       epochs = epochs,
                       batch_size = batch_size,
                       callbacks = [callbacklr],
                       class_weight = class_weight,
                       verbose=1)

#com ruído
#historyDNN = modelo_DNN.fit(x=SxNoise, y=Sy,
#                       epochs = epochs,
#                       batch_size = batch_size,
#                       callbacks = [callbacklr],
#                       class_weight = class_weight,
#                       verbose=1)
```

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28)]              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 28)                112
_________________________________________________________________
dense_5 (Dense)              (None, 32)                928
_________________________________________________________________
dense_6 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 32)                128
_________________________________________________________________
code (Dense)                 (None, 28)                924
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 29
=================================================================
Total params: 3,177
Trainable params: 3,057
Non-trainable params: 120
_________________________________________________________________
Epoch 1/15
445/445 [==============================] - 1s 2ms/step - loss: 0.0032 - precision: 0.2000 - recall:
0.5200
Epoch 2/15
445/445 [==============================] - 1s 2ms/step - loss: 5.4550e-04 - precision: 0.8846 - recal
l: 0.9200
Epoch 3/15
445/445 [==============================] - 1s 2ms/step - loss: 5.1146e-04 - precision: 0.7778 - recal
l: 0.8400
Epoch 4/15
445/445 [==============================] - 1s 2ms/step - loss: 2.8841e-04 - precision: 1.0000 - recal
l: 0.9200
Epoch 5/15
445/445 [==============================] - 1s 2ms/step - loss: 3.7671e-04 - precision: 0.8462 - recal
l: 0.8800
Epoch 6/15
445/445 [==============================] - 1s 2ms/step - loss: 2.1792e-04 - precision: 0.8571 - recal
l: 0.9600
Epoch 7/15
445/445 [==============================] - 1s 2ms/step - loss: 2.1484e-04 - precision: 0.9231 - recal
l: 0.9600
Epoch 8/15
445/445 [==============================] - 1s 2ms/step - loss: 2.7893e-04 - precision: 0.9200 - recal
l: 0.9200
Epoch 9/15
445/445 [==============================] - 1s 2ms/step - loss: 1.8849e-04 - precision: 0.8889 - recal
l: 0.9600
Epoch 10/15
445/445 [==============================] - 1s 2ms/step - loss: 2.3009e-04 - precision: 0.8846 - recal
l: 0.9200
Epoch 11/15
445/445 [==============================] - 1s 2ms/step - loss: 1.4469e-04 - precision: 0.9231 - recal
l: 0.9600
Epoch 12/15
445/445 [==============================] - 1s 2ms/step - loss: 2.9707e-04 - precision: 0.8800 - recal
l: 0.8800
Epoch 13/15
445/445 [==============================] - 1s 2ms/step - loss: 2.0294e-04 - precision: 0.9231 - recal
l: 0.9600
Epoch 14/15
445/445 [==============================] - 1s 2ms/step - loss: 1.4788e-04 - precision: 0.9231 - recal
l: 0.9600
Epoch 15/15
445/445 [==============================] - 1s 2ms/step - loss: 9.2967e-05 - precision: 0.9259 - recal
l: 1.0000
```

In [14]:
```
#Avalie precisão e revocação no treinamento S e teste T.
score_S_DNN = modelo_DNN.evaluate(Sx, Sy, verbose = 0)
score_T_DNN = modelo_DNN.evaluate(Tx, Ty, verbose = 0)
print("S: Precisão = %.4f, Revocação = %.10f" % (score_S_DNN[1], score_S_DNN[2]))
print("T: Precisão = %.4f, Revocação = %.10f" % (score_T_DNN[1], score_T_DNN[2]))
```

```
S: Precisão = 0.9231, Revocação = 0.9599999785
T: Precisão = 0.6899, Revocação = 0.7982062697
```

Tentativa 2: SVM nos dados originais

In [15]:
```
#Classificador SVM treinado nos dados originais S.

#aplicando ponderação das classes
peso_0 = 0.1
peso_1 = 0.9
class_weight = {0: peso_0, 1: peso_1}

#definindo modelo SVM
model = SVC(kernel='linear', C = 0.5,
            random_state=1,
            class_weight = class_weight)

#como não foi comentado nada nas questões referentes a este item, por padrão, fiz o treinamento com os
 dados sem ruído
#aplicando reshape do Sy, apenas para manter num formato sem warning do sklearn de array de 1d

#sem ruído
model.fit(Sx,np.asarray(Sy).reshape(-1))

#com ruído
#model.fit(SxNoise,np.asarray(Sy).reshape(-1))

#predizendo as classes para S e T:
predict_Sy = model.predict(Sx)
predict_Ty = model.predict(Tx)

#Avalie precisão e revocação no treinamento S e teste T.
print("S: Precisão = %.4f, Revocação = %.10f" % (precision_score(Sy, predict_Sy), recall_score(Sy, pred
ict_Sy)))
print("T: Precisão = %.4f, Revocação = %.10f" % (precision_score(Ty, predict_Ty), recall_score(Ty, pred
ict_Ty)))
```

```
S: Precisão = 0.9615, Revocação = 1.0000000000
T: Precisão = 0.5162, Revocação = 0.7847533632
```

In [16]:

```python
#Classificador SVM treinado nos dados S obtendo sua representação do código da rede de classificação (m
odelo B).

#como era necessário escolher 2 itens dos 3 apresentados, esta tentativa de resolução é apenas por ques
tão de aprendizado.
extract_svm_B = keras.models.Model(modelo_B.inputs, modelo_B.layers[-2].output)

#como não foi comentado nada nas questões referentes a este item, por padrão, fiz o treinamento com os
 dados sem ruído

#sem ruído
predict_Sy_svm_B = extract_svm_B.predict(Sx)

#com ruído
#predict_Sy_svm_B = extract_svm_B.predict(SxNoise)


#realizando projeção PCA do conjunto S processado pelo encoder (camada de código) para manter as mesmas
 dimensões de Sy
pca_S_B = PCA(n_components=1, random_state=1)
pca_S_result_B = pca_S_B.fit_transform(predict_Sy_svm_B)

#definindo modelo SVM
svm_S_B = SVC(C = 0.5,
              random_state=1,
              class_weight = class_weight)


#como não foi comentado nada nas questões referentes a este item, por padrão, fiz o treinamento com os
 dados sem ruído
#foi necessário converter para int, por conta de um erro falando que o tipo float não estava conseguind
o
#fazer a classificação de forma correta nos dados

#sem ruído
svm_S_B.fit(Sx, np.asarray(pca_S_result_B).reshape(-1).astype('int'))

#com ruído
#svm_S_B.fit(SxNoise, np.asarray(pca_S_result_B).reshape(-1).astype('int'))

#predizendo as classes para S e T:
predict_svm_Sy = svm_S_B.predict(Sx)
predict_svm_Ty = svm_S_B.predict(Tx)

#Avalie precisão e revocação no treinamento S e teste T.
print("S: Precisão = %.4f, Revocação = %.10f" % (precision_score(Sy,predict_svm_Sy, average='weighted'
), recall_score(Sy, predict_svm_Sy, average='weighted',zero_division=0)))
print("T: Precisão = %.4f, Revocação = %.10f" % (precision_score(Ty, predict_svm_Ty, average='weighted'
), recall_score(Ty, predict_svm_Ty, average='weighted',zero_division=0)))
```

```
S: Precisão = 0.9965, Revocação = 0.7908707865
T: Precisão = 0.9983, Revocação = 0.7316227072
```