Aprendizado de Máquina
Lista 04

Ricardo Marra - 19/0137576

I. Questão 01

Choose a toolbox and use it to train a Feedforward Neural Network for a multiclass classification task. Split your data into a training set, a validation set to define the training stopping condition (number of epochs) and a test set, for the final evaluation of the model. Softmax neurons must be used in the output layer, one for each class, and the training criterion must be cross-entropy minimization.

A. Describe your dataset, the network architecture (hidden units, dimension and number of layers) and discuss the overall performance (dataset splitting strategy, classification accuracy, confusion matrix). Suggestion of dataset repository: Kaggle, UC Irvine Machine Learning Repository— http://archive.ics.uci.edu/ml/

O conjunto de dados foi criado utilizando uma função do pacote *sklearn* chamada *make_blobs*, a função cria um cluster para cada classe desejada. A criação destes clusters se baseia em quatro argumentos: número de amostras, número de atributos, número de clusters (classes) e a distância entre entre as amostras e o centro do cluster.

Neste caso, criou-se 2000 amostras, 2 atributos, 3 classes e o desvio padrão do cluster é igual a 1. O conjunto de dados pode ser visualizado na Fig. 1.

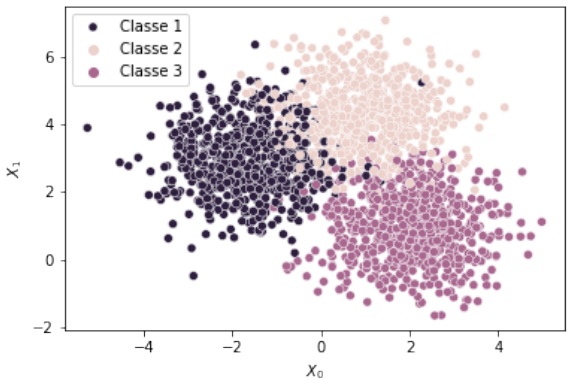

Fig. 1. Conjuto de dados.

É possível observar que as classes não são totalmente separáveis, visto que há uma certa sobreposição entre elas.

O conjunto de dados foi dividido em 60% (1200 amostras) para o conjunto de treinamento, e 20% (400 amostras) para os conjuntos de validação e teste.

Como o problema não é muito complexo, não é necessário uma grande rede neural para obter um bom resultado. Por isso, só foram utilizadas duas camadas de 10 neurônios cada. Essa escolha foi feita a partir do treinamento de 5 redes diferentes, com diferentes quantidades de neurônios (2, 5, 10, 15 e 20) nas duas camadas escondidas. O aumento do número de neurônios não impactou muito na performance da rede no conjunto de validação, como pode ser visto na Tabela I.

TABLE I
Acurácia das redes

Número de Neurônios	Acurácia
2	0.911
5	0.923
10	0.924
15	0.923
20	0.925

Para prevenir *overfit* no conjunto de dados se utilizou do *early stopping*, isso consiste na parada do treinamento quando o valor da função erro do conjunto de validação começa a divergir do valor do conjunto de treinamento. Desta forma, o modelo parou em 140 épocas. Ao avaliar no conjunto de teste, o modelo obteve uma acurácia de 0.95, não muito diferente do que foi avaliado no conjunto de validação, logo pode-se dizer que não aconteceu *overfit*.

A curva da função erro e acurácia dos conjuntos de treinamento e validação podem ser observadas nas Fig. 2 e Fig. 3. É possível perceber que não há muita divergência entre a curva de treinamento e validação.

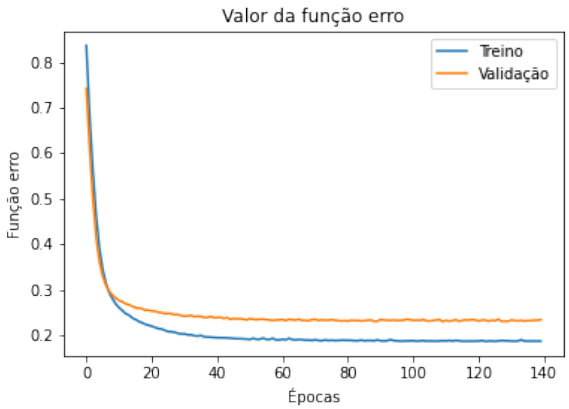

Fig. 2. Gráfico do valor da função erro.

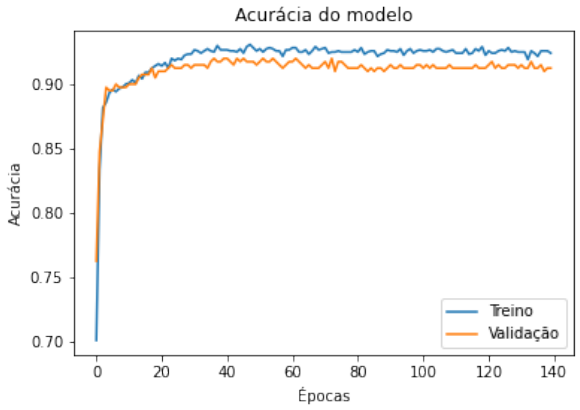

Fig. 3. Gráfico da acurácia do modelo.

A matriz de confusão pode ser observada na Tabela II, e assim pode-se perceber que a maior fonte de erro é na classificação das classes 1 e 3. A própria distribuição dos dados já indica essa dificuldade, já que as amostras das classes 1 e 3 estão mais entrelaçadas que a classe 2.

TABLE II
MATRIZ DE CONFUSÃO

	Classe 1	**Classe 2**	**Classe 3**
Classe 1	131	2	7
Classe 2	2	121	0
Classe 3	8	1	128

II. QUESTÃO 02

Use WEKA toolbox to apply a J48 decision tree **over the same classification problem**, previously chosen in question 1.

A. Present the resulting tree (dimension, rule-based description) and compare its performance with the results of question 1.

Para a aplicação na árvore de decisão, utilizou-se 80% do conjunto de dados para o treino e 20% para teste do modelo, deste modo, o conjunto de teste seria do mesmo tamanho que o conjunto utilizado em I-A.

Foi se utilizado todos os parâmetros padrões do ambiente WEKA para a criação da árvore J48. A árvore resultante pode ser vista na Fig. 4 e a matriz de confusão na Tabela III.

TABLE III
MATRIZ DE CONFUSÃO

	Classe 1	**Classe 2**	**Classe 3**
Classe 1	114	5	14
Classe 2	0	136	2
Classe 3	8	1	120

O modelo teve uma acurácia de 0.925 no conjunto de teste, valor pouco menor que o atingido utilizando a rede neural. A matriz de confusão também ficou parecida, a diferença se dá por um erro maior na classificação entre as classes 1 e 3, o que é esperado visto a menor acurácia. Outro fator é o balanceamento entre as classes, na divisão do conjunto de testes, visto que se tem mais amostras das classes 1 e 2.

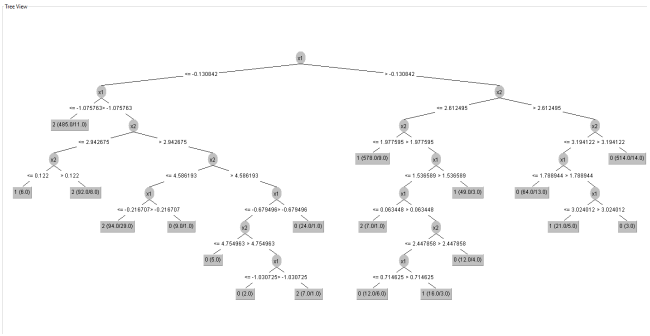

Fig. 4. Árvore de decisão.

Neste caso, a árvore de decisão ficou muito próxima da rede neural, visto que era um problema mais simples. Caso o problema fosse mais complexo, provavelmente a rede neural se sairia melhor. Porém, ao contrário da rede neural, é possível observar todo o caminho que a árvore de decisão percorre para poder fazer a classificação da observação, tornando ela uma opção mais viável para se ter uma melhor interpretação do modelo.

III. QUESTÃO 03

A. Pick SVM light (http://svmlight.joachims.org/), WEKA toolbox or a library/toolbox at your own choice, for an experiment testing different kernel functions and parameter setups in order to obtain the best SVM classifier for a two-class classification task, whose data is chosen by you. Don't forget to discuss the results.

O conjunto de dados utilizado vem da função do pacote *sklearn* chamada *make_moons*. A função cria um conjunto de amostras em formato de luas, como pode ser visto na Fig. 5. Um conjunto parecido foi criado para se testar os modelos de SVM.

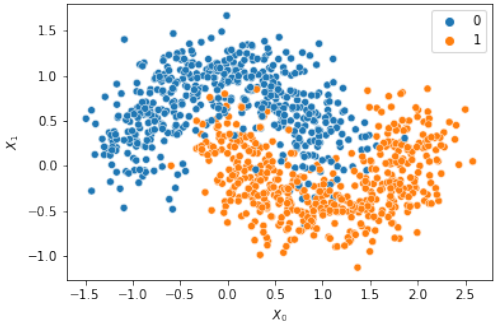

Fig. 5. Conjunto de dados.

As máquina de vetores suporte foram treinadas utilizando dois kernels diferentes, polinomial (1) e RBF (função de base radial) (2). Não se foi testado o kernel linear devido o comportamento dos dados. Para o estudo variou-se três parâmetros durante o treinamento. O parâmetro de regularização da função erro C, o grau do polinômio q e o parâmetro γ do kernel RBF.

1) Kernel Polinomial:

$$K_{poly}(a,b) = (a^T b + c)^q \tag{1}$$

A acurácia do modelo em cada iteração é mostrada na Tabela IV. É possível perceber que o aumento do parâmetro de regularização C tende a aumentar a acurácia do modelo. Porém, fazer isso pode fazer com que aconteça *overfit*, visto que, com o aumento do parâmetro de regularização, o viés inserido no modelo seja maior.

TABLE IV
ACURÁCIA DE CADA MODELO POLINOMIAL

C \ q	2	3	4	5	6	7
0.01	**0.6825**	0.775	0.6975	0.7475	0.695	0.7425
0.1	0.7575	0.835	0.7075	0.7875	0.71	0.77
1	0.75	**0.9**	0.7175	0.8825	0.715	0.79
10	0.7475	0.8925	0.7175	0.86	0.7175	0.855
100	0.7475	0.8925	0.72	0.8625	0.7025	0.8425
1000	0.7475	0.89	0.7225	0.865	0.7075	0.8325

Para este kernel, o modelo com maior acurácia foi com $C = 1$ e $q = 3$, ao se plotar a região de decisão do modelo, é possível perceber que não acontece *overfit*, visto que a região de decisão não está tão focada nos dados de teste, como pode ser visto na Fig. 6. Na Fig. 7. pode-se observar o modelo com a pior acurácia para este kernel, por questões de contraste.

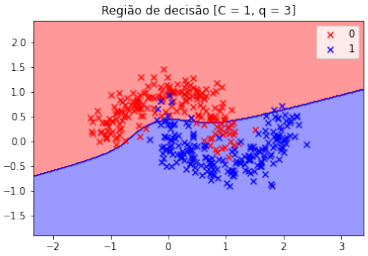

Fig. 6. Região de decisão (modelo polinomial [C = 1, q = 3]).

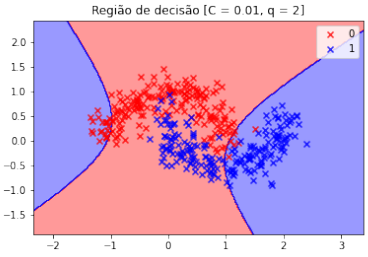

Fig. 7. Região de decisão (modelo polinomial [C = 0.01, q = 2]).

2) Kernel RBF:

$$K_{RBF}(a,b) = e^{-\gamma||a-b||^2} \tag{2}$$

Todos os valores de acurácia para o kernel RBF estão na Tabela V. Assim como visto com o kernel polinomial, o aumento do parâmetro de regularização tende a aumentar a acurácia do modelo, mas, ao contrário do outro kernel, o *overfit* neste caso é mais enfatizado. Isso pode ser visto no "melhor" modelo com o kernel RBF. A região de decisão deste modelo pode ser vista na Fig. 8.

TABLE V
ACURÁCIA DE CADA MODELO RBF

C \ γ	0.01	0.1	1	2	3	4
0.01	**0.8**	0.8	0.89	0.91	0.9375	0.94
0.1	0.8075	0.86	0.9425	0.955	0.955	0.955
1	0.8625	0.89	0.9575	**0.96**	0.9575	0.9575
10	0.8775	0.9075	0.9575	0.9575	0.9575	0.9575
100	0.88	0.9475	0.9575	0.955	0.955	0.955
1000	0.8925	0.955	0.9575	0.9525	0.955	0.95

Observando a região de decisão, pode-se ver que o modelo está muito adaptado ao conjunto de teste, mostrando um caso de *overfit*. Caso novas amostras fossem inseridas a acurácia do modelo poderia começar a cair. O modelo com kernel RBF que mais errou pode ser visualizado na Fig. 9. Apesar do kernel, a região de decisão é apenas uma linha separando os dados, mostrando o kernel linear poderia ser considerado.

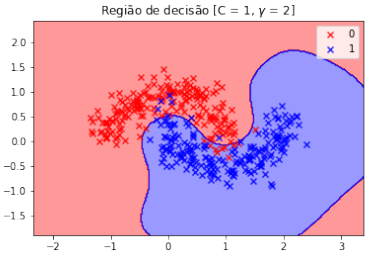

Fig. 8. Região de decisão (modelo RBF [C = 1, γ = 2]).

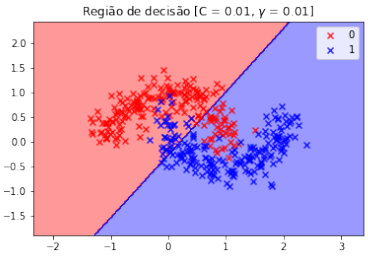

Fig. 9. Região de decisão (modelo RBF [C = 0.01, γ = 0.01]).

3) Conclusão: A variação entre os parâmetros promove a discussão de qual seria o melhor modelo. Em média, o kernel polinomial consegue 0.77 de acurácia, já o kernel RBF

consegue 0.92, porém, como foi visto, essa acurácia pode ser um fator de *overfit* do modelo em cima dos dados de teste, devido sua região de decisão.

Neste caso, a escolha do melhor modelo seria a do kernel polinomial com $C = 1$ e $q = 3$. Essa escolha se deve ao fato que, ao observar a região de decisão, percebe-se que não há muito indício de *overfit*, ao contrário do melhor kernel RBF, que pode não se ajustar a novas amostras.

IV. Apêndice

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from matplotlib.colors import ListedColormap

from tensorflow.python.keras import models
from tensorflow.python.keras import layers
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras.callbacks import
    EarlyStopping

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples = 2000, n_features = 2,
    centers = 3, cluster_std = 1, random_state = 0)
X = pd.DataFrame(X, columns = ['x1', 'x2'])
y = pd.DataFrame(y, columns = ['target'])

data = pd.concat([X, y], axis = 1)
data.to_csv('data.csv', index = False)

sns.scatterplot(x = 'x1', y = 'x2', data = data, hue
     = 'target')
plt.xlabel('$X_0$')
plt.ylabel('$X_1$')
plt.legend([f'Classe {x}' for x in range(1, y.
    nunique()[0] + 1)])

plt.savefig('images/plotq1a')
plt.show()

y_cat = to_categorical(y)
X_new, X_test, y_new, y_test =  train_test_split(X,
    y_cat, test_size = 0.2, random_state = 0)
X_train, X_val, y_train, y_val = train_test_split(
    X_new, y_new, test_size = 0.25, random_state =
    0)

def plot_loss_accuracy(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Valor da fun  o erro')
    plt.ylabel('Fun  o erro')
    plt.xlabel(' pocas ')
    plt.legend(['Treino', 'Valida  o'], loc='best'
    )
    plt.savefig('images/plotq1b')
    plt.show()

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Acur cia do modelo')
    plt.ylabel('Acur cia')
    plt.xlabel(' pocas ')
    plt.legend(['Treino', 'Valida  o'], loc='best'
    )
    plt.savefig('images/plotq1c')
    plt.show()

def define_model(neurons):

    model = models.Sequential()
    model.add(layers.Dense(neurons, activation = '
    relu', input_dim = X_train.shape[1]))
    model.add(layers.Dense(neurons, activation = '
    relu'))
    model.add(layers.Dense(y_cat.shape[1],
    activation = 'softmax'))
```

```

    model.compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = ['accuracy'])

    return model

history_dict = {}

for neurons in [2, 5, 10, 15, 20]:
    model = define_model(neurons)
    es = EarlyStopping(patience = 30,
    restore_best_weights = True)

    print(neurons)
    history_dict[neurons] = model.fit(X_train,
    y_train, validation_data = (X_val, y_val),
    epochs = 300, verbose = .1, callbacks = [es])
    history_dict[f'model_{neurons}'] = model

y_pred = np.argmax(history_dict['model_10'].predict(
    X_test), axis = -1)
y_ori = np.argmax(y_test, axis = -1)
confusion_matrix(y_ori, y_pred)
```

Listing 1. Códigos referente a Questão 01

```
def plot_decision_regions(X, y, classifier, test_idx
    =None, resolution=0.02):
    """Plot the 2D-decision region of a classifier
    with matplotlib along its first two dimensions X
    [:,0] and X[:,1].

    Args:
        X (np.Array): (n,p) dataset to classify
        y (np.Array): (n,) array of labels. Works
    well up to 5 unique labels.
        classifier (sklearn): fitted sklearn
    classifier.
        test_idx (int, optional):  Index of test
    datapoints within X to display with a larger
    mark style. Defaults to None.
        resolution (float, optional): Resolution of
    the meshgrid used to colorize regions. Defaults
    to 0.02.
    """

    # setup marker generator and color map up for up
     to 5 classes
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', '
    cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))
    ])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max
    () + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max
    () + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
     resolution),
                           np.arange(x2_min, x2_max,
     resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
    xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Plot samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl,
    1],
                    alpha=0.8, color=cmap(idx),
                    marker='x', label=cl)

    # Plot test samples if they exist
    if not test_idx is None:
        X_test, y_test = X[test_idx, :], y[test_idx]
        for idx, cl in enumerate(np.unique(y_test)):
            plt.scatter(x=X_test[y_test == cl, 0], y
    =X_test[y_test == cl, 1],
                        alpha=1, color=cmap(idx),
                        linewidths=1, marker='o', s
    =55, label=f'test {cl}')
    plt.legend()

from sklearn.datasets import make_moons
from sklearn.svm import SVC

X_train, y_train = make_moons(n_samples = 1000,
    noise = 0.25, random_state = 0)
X_test, y_test = make_moons(n_samples = 400, noise =
     0.2, random_state = 0)

sns.scatterplot(x = X_train[:, 0], y = X_train[:,
    1], hue = y_train)
plt.xlabel('$X_0$')
plt.ylabel('$X_1$')

plt.legend()
plt.savefig('images/plotq3a')
plt.show()

c_list = [0.01, 0.1, 1, 10, 100, 1000]
d_list = [2, 3, 4, 5, 6, 7]
acc_list = []
svms = {}
idx = 0

for C in c_list:
    for d in d_list:
        svm = SVC(kernel = 'poly', C = C, degree = d
    )
        svm.fit(X_train, y_train)

        svms[idx] = svm
        idx += 1
        acc_list.append(((svm.predict(X_test) ==
    y_test).sum() / y_test.shape[0]))

plot_decision_regions(X_test, y_test, classifier =
    svms[np.argmax(acc_list)])
plt.title('Regi o de decis o [C = 1, q = 3]')
plt.savefig('images/plotq3b')

plot_decision_regions(X_test, y_test, classifier =
    svms[np.argmin(acc_list)])
plt.title('Regi o de decis o [C = 0.01, q = 2]')
plt.savefig('images/plotq3c')

c_list = [0.01, 0.1, 1, 10, 100, 1000]
l_list = [0.01, 0.1, 1, 2, 3, 4]
rbf_acc_list = []
rbfs = {}
idx = 0

for C in c_list:
    for l in l_list:
        svm = SVC(kernel = 'rbf', C = C, gamma = l)
        svm.fit(X_train, y_train)

        rbfs[idx] = svm
        idx += 1
        rbf_acc_list.append(((svm.predict(X_test) ==
```

```
     y_test).sum() / y_test.shape[0]))

plot_decision_regions(X_test, y_test, classifier =
    rbfs[np.argmax(rbf_acc_list)])
plt.title('Regi o de decis o [C = 1, $\gamma$ = 2]
    ')
plt.savefig('images/plotq3d')

plot_decision_regions(X_test, y_test, classifier =
    rbfs[np.argmin(rbf_acc_list)])
plt.title('Regi o de decis o [C = 0.01, $\gamma$ =
     0.01]')
plt.savefig('images/plotq3e')
```

Listing 2. Códigos referente a Questão 03