exercicio6

January 22, 2021

1 Exercício 6

O objetivo deste exercício é utilizar as ELMs para resolver problemas multidimensionais, a partir de bases de dados reais.

A primeira base de dados a ser estudada é a base Breast Cancer (diagnostic). Para esta base, os alunos deverão dividir de forma aleatória os dados entre treinamento e teste comparar as acurácias de treinamento e teste para diferentes valores do hiperparâmetro que controla o número de neurônios. Os valores de acurácia devem ser apresentados na forma de ± _ para, pelo menos, cinco execuções diferentes.

Além das Extreme Learning Machines, será treinado, utilizando o perceptron, e avaliado seu desempenho na solução dos dois problemas, comparado às ELMs.

Como neste exercício será avaliada a performance do classificador para diferentes quantidades de neurônios na camada intermediária será utilizada a metodologia de validação cruzada para selecionar o melhor modelo.

1.0.1 Validação Cruzada

A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados.

O conceito central das técnicas de validação cruzada é o particionamento do conjunto de dados em subconjuntos mutuamente exclusivos, e posteriormente, o uso de alguns destes subconjuntos para a estimação dos parâmetros do modelo (dados de treinamento), sendo os subconjuntos restantes (dados de validação ou de teste) empregados na validação do modelo.

1.0.2 Método k-fold

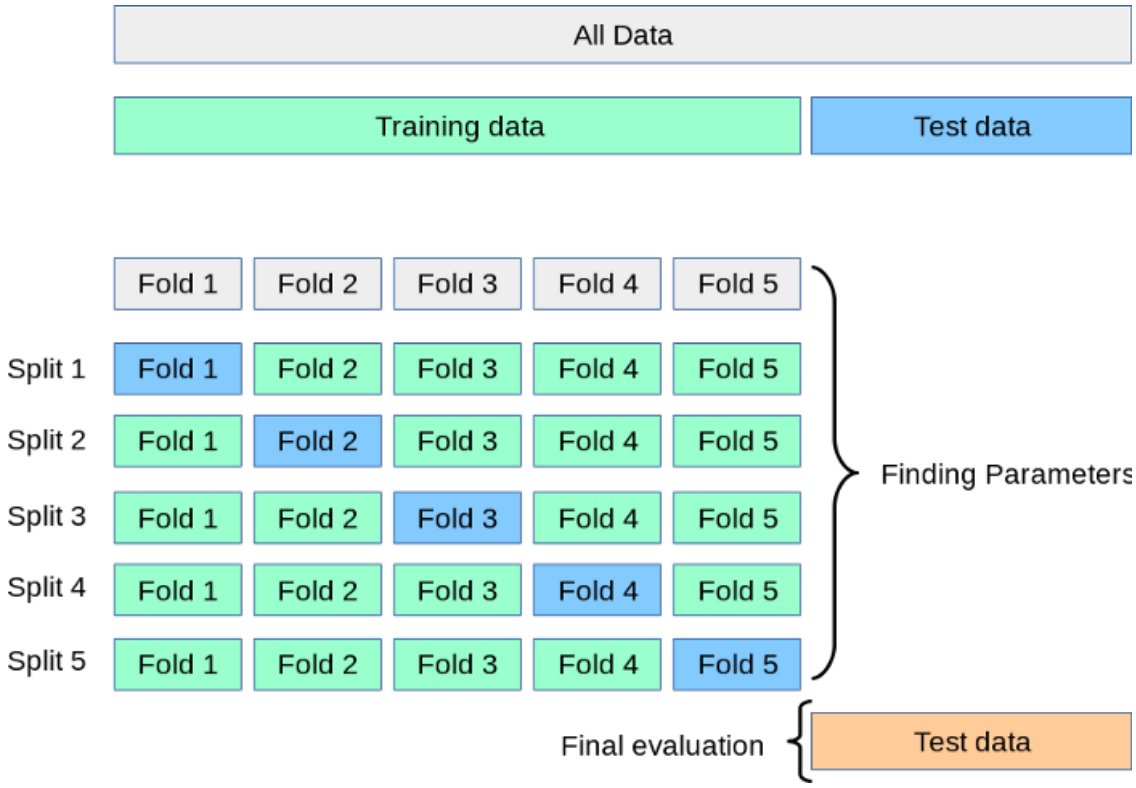
Na abordagem básica, chamada k-fold CV, o conjunto de treinamento é dividido em k conjuntos menores. O seguinte procedimento é seguido para cada uma dos k "folds":

- Um modelo é treinado usando $k - 1$ como dados de treinamento;
- O modelo resultante é validado na parte restante dos dados (ou seja, é usado como um conjunto de teste para calcular uma medida de desempenho, como precisão).

A medida de desempenho relatada pela validação cruzada k-fold é então a média dos valores calculados no loop. Essa abordagem pode ser computacionalmente cara, mas não desperdiça muitos dados (como é o caso ao corrigir um conjunto de validação arbitrário), o que é uma grande vantagem em problemas como inferência inversa, onde o número de amostras é muito pequeno.

[31]:
```
from IPython import display

# Particionamento k-fold (Fonte: source - Scikit-learn.org)
display.Image("./Cross-Validation.png")
```

[31]:



No código abaixo se encontra o classificador **ELM** que foi implementado no **Exercício 5**.

[7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, plot_roc_curve, roc_curve,␣
 ↪auc
from sklearn.metrics import plot_confusion_matrix, confusion_matrix

class ELM(BaseEstimator):
    """ Classificador ELM """
```

```
    def __init__(self, p):
        self.Z = None
        self.W = None
        self.p = p

    def _check_X_y(self, X, y):
        """ Validate assumptions about format of input data"""
        assert set(y) == {-1, 1}, 'Response variable must be ±1'
        return X, y

    def fit(self, X: np.ndarray, y: np.ndarray):
        """Controi um classificador otimizado a partir do conjunto de␣
↪treinamento (X, y).

         Parâmetros
         ----------
         X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
             Os exemplos de entrada de treinamento.

         y: Vetor de formato (n_samples,)
             Os valores alvo (rótulos de classe) do conjunto de treinamento.

         p: número de neurônios da camada intermediária

         Retorna
         -------
         self: objeto
             Estimador ajustado.
         """

        X, y = self._check_X_y(X, y)

        # Vetor pesos da camada escondida gerado de forma aleatória
        self.Z = np.random.uniform(low=-0.5, high=0.5, size=(X.shape[1] +1,␣
↪self.p))

        H = np.tanh(np.dot(X, self.Z[1:]) + self.Z[0])

        # Vetor pesos entre a camada de saída e a camada intermediária
        self.W = np.dot(np.linalg.pinv(H), y)

        return self

    def predict(self, X):
        """ Make predictions using already fitted model
```

```
        Parâmetros
         ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
             Os exemplos de entrada.
        """

        H = np.tanh(np.dot(X, self.Z[1:]) + self.Z[0])
        return np.sign(np.dot(H,self.W))

    def score(self, X, Y):
        """ Retorna a acurácia do classificador

        Parâmetros
         ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
             Os exemplos de entrada.

        Y: Vetor de formato (n_samples,)
             Os valores alvo (rótulos de classe) dos exemplos de entrada.
        """
        Y_pred = self.predict(X)
        return np.sum(Y_pred == Y)/Y.shape[0]
```

A seguir, se encontra o código do classificador **Perceptron** que foi implementado no **Exercício 4**.

[19]:
```
class Perceptron(BaseEstimator):
    """ Classificador Perceptron """

    def __init__(self, eta = 0.01, maxepocas=100):
        self.W = None
        self.eta = eta
        self.maxepocas = maxepocas

    def _check_X_y(self, X, y):
        """ Validate assumptions about format of input data"""
        assert set(y) == {-1, 1}, 'Response variable must be ±1'
        return X, y

    def fit(self, X: np.ndarray, y: np.ndarray):
        """Controi um classificador otimizado a partir do conjunto de␣
↪treinamento (X, y).

         Parâmetros
         ----------
```

```
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
            Os exemplos de entrada de treinamento.

        y: Vetor de formato (n_samples,)
            Os valores alvo (rótulos de classe) do conjunto de treinamento.

        Retorna
        -------
        self: objeto
            Estimador ajustado.
        """

        # Vetor pesos inicialmentee nulo
        self.W = np.zeros(X.shape[1] +1)

        for nepocas in range(self.maxepocas):
            for xi, yi in zip(X, y):
                yhati = self.predict(xi)
                self.W += self.eta * (yi - yhati) * np.insert(xi,0,1)

        return self

    def predict(self, X):
        """ Make predictions using already fitted model

        Parâmetros
         ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
            Os exemplos de entrada.
        """

        return np.sign(np.dot(X, self.W[1:]) + self.W[0])

    def score(self, X, Y):
        """ Retorna a acurácia do classificador

        Parâmetros
         ----------
        X: {tipo matriz, matriz esparsa} de forma (n_amostras,␣
↪n_características)
            Os exemplos de entrada.

        Y: Vetor de formato (n_samples,)
            Os valores alvo (rótulos de classe) dos exemplos de entrada.
        """
```

```
        Y_pred = self.predict(X)
        return np.sum(Y_pred == Y)/Y.shape[0]
```

1.1 Experimentos

1.1.1 *Breast Cancer*

A primeira base de dados a ser estudada é a base Breast Cancer (diagnostic). O código a seguir faz a importação e tratamento dos dados.

[20]:
```
# Carregar a base de dados, remover dados faltantes e normalizar os dados
data = load_breast_cancer()

X = data.data
Y = data.target

labels = data.target_names

count = 0
# Loop para verificar se há dados nulos/fatantes no vetor de rótulos Y
for i, j in zip(pd.DataFrame(Y).isnull().sum(), pd.DataFrame(X).isnull().sum()):
    if i!=0 or j!=0:
        count+=1

if count == 0:
    print('Não há dados faltantes!')


# Standardization - Todos os atributos agora terão média zero e variação␣
 ↪unitária
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Mapeia os rótulos binários de forma que negativo = -1 e positivo = +1
Y[Y==0] = -1

# Separa os dados de forma aleatória - 70% para treinamento e 30% para testes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

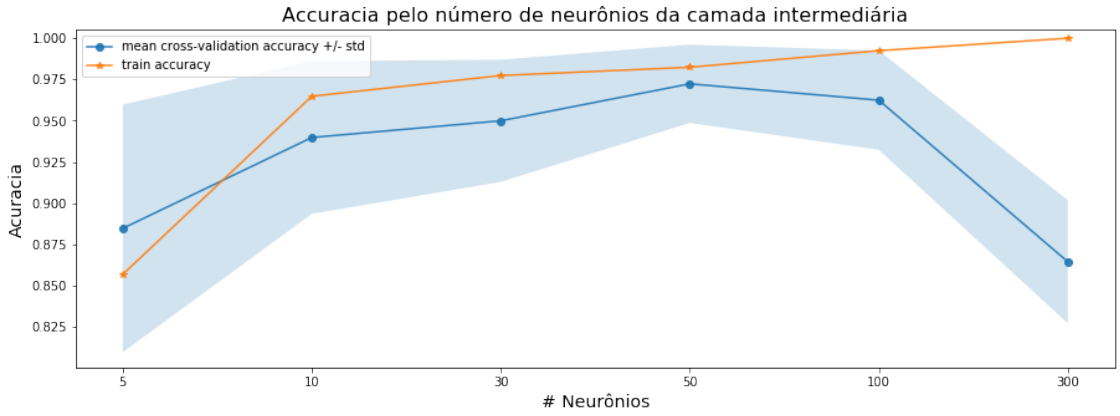
```
Não há dados faltantes!
```

ELM O código a seguir plota a performance estimada com +/- 1 desvio padrão para diferentes quantidades de neurônios na camada intermediária, utilizando validação cruzada com 10 folds.

[21]:
```
def run_cross_validation_on_elms(X, y, Nneuronios, cv=10):
    """ Método para ajustar ELMs com diferentes números de neurônios na camada␣
 ↪oculta
```

```
        sobre os dados de treinamento usando validação cruzada

    Parâmetros
        ----------
    X: {tipo matriz, matriz esparsa} de forma (n_amostras, n_características)
            Os exemplos de entrada.

    y: Vetor de formato (n_samples,)
             Os valores alvo (rótulos de classe) do conjunto de treinamento.

    Nneuronios: Lista com quantidade de neurônios (p) que serão avaliados

    cv: Número de folds que serão utilizados na validação cruzada

    """
    # Listas auxiliares que serão retornadas
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores_train = []

    for p in Nneuronios:
        ELM_model = ELM(p)
        cv_scores = cross_val_score(ELM_model, X, y, cv=cv, scoring='accuracy')
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        model = ELM_model.fit(X, y)
        accuracy_scores_train.append(model.score(X, y))
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    accuracy_scores_train = np.array(accuracy_scores_train)
    return cv_scores_mean, cv_scores_std, accuracy_scores_train

# function for plotting cross-validation results
def plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪accuracy_scores_train, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(list(map(str, Nneuronios)), cv_scores_mean, '-o', label='mean␣
 ↪cross-validation accuracy +/- std', alpha=0.9)
    ax.fill_between(list(map(str, Nneuronios)), cv_scores_mean-cv_scores_std,␣
 ↪cv_scores_mean+cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(list(map(str, Nneuronios)), accuracy_scores_train, '-*',␣
 ↪label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('# Neurônios', fontsize=14)
```

```
    ax.set_ylabel('Acuracia', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(list(map(str, Nneuronios)))
    ax.legend()

# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [5, 10, 30, 50, 100, 300]


cv_scores_mean, cv_scores_std, accuracy_scores_train =␣
 ↪run_cross_validation_on_elms(X_train,
                                                        Y_train,Nneuronios)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
 ↪accuracy_scores_train,
                               'Accuracia pelo número de neurônios da camada␣
 ↪intermediária')
```



[23]:
```
idx_max = cv_scores_mean.argmax()
best_Nneuronios = Nneuronios[idx_max]
best_score = cv_scores_mean[idx_max]
best_score_std = cv_scores_std[idx_max]
print('Ao utilizar {} neurônios na camada intermediária foi atingindo a melhor␣
 ↪acurácia média de validação cruzada sobre o conjunto de treinamento: {} +/-␣
 ↪{}%'.format(best_Nneuronios, round(best_score*100,5),␣
 ↪round(best_score_std*100, 5)))
```

```
Ao utilizar 50 neurônios na camada intermediária foi atingindo a melhor acurácia
média de validação cruzada sobre o conjunto de treinamento: 97.23718 +/-
2.37101%
```

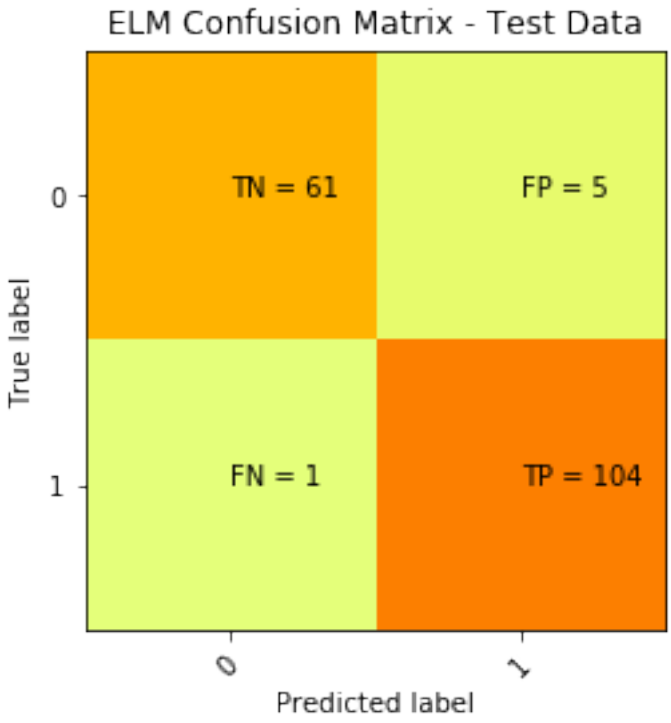
Ao utilizar mais de 50 neurônios a acurácia de treinamento aumento ao passo que a

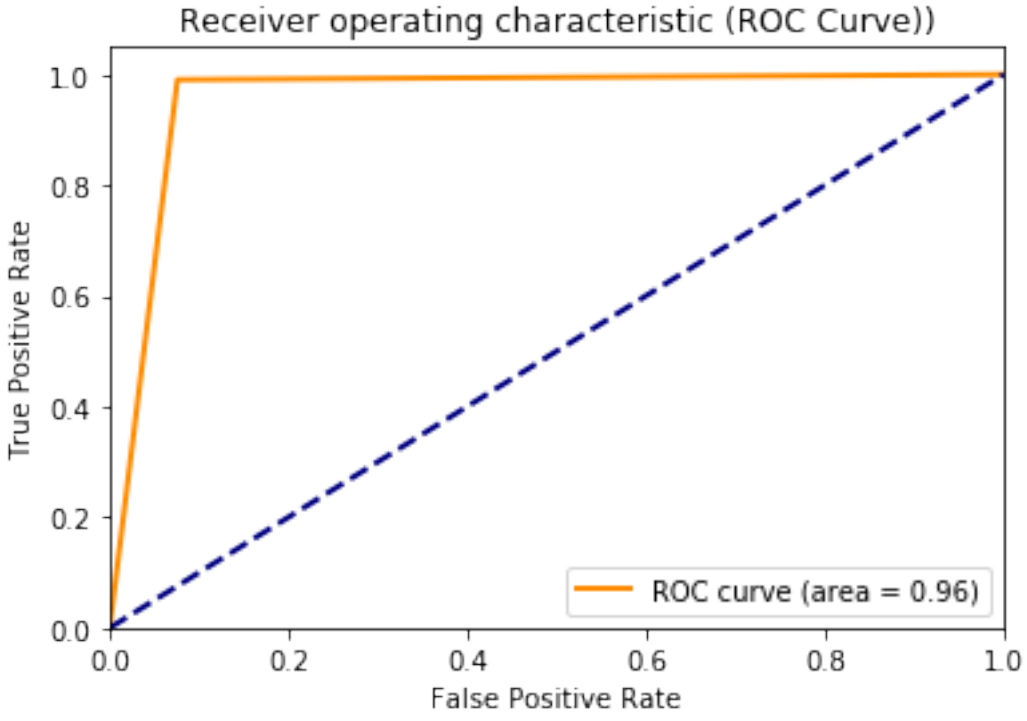
acurácia média de validação cruzada diminui, evidenciando que ocorre sobreajuste. Ao diminuir a capacidade do modelo, menos de 50 neurônios, ocorre subajuste com ambas as acurácias de treinamento e de validação cruzada menores quando comparadas ao modelo com 50 neurônios.

[24]:
```
# function for training and evaluating the ELM model
def run_single_classificator(X_train, y_train, X_test, y_test, labels,␣
↪classificator = 1, p = 10):

    if classificator == 1:
        model = ELM(p).fit(X_train, y_train)
        title = 'ELM Confusion Matrix - Test Data'
    else:
        model = Perceptron().fit(X_train, y_train)
        title = 'Perceptron Confusion Matrix - Test Data'

    accuracy_train = model.score(X_train, y_train)
    accuracy_test = model.score(X_test, y_test)
    Y_pred = model.predict(X_test)

    print('Accuracia - Conjunto de Treinamento: ', round(accuracy_train*100,5),␣
↪'%\n')
    print('Accuracia - Conjunto de Teste: ', round(accuracy_test*100,5), '%\n')
    print(classification_report(y_test, Y_pred, target_names=labels))

    # matriz de confusão
    cm = confusion_matrix(y_test, Y_pred)

    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    tick_marks = np.arange(len(np.unique(y_train)))
    plt.xticks(tick_marks, rotation=45)
    plt.yticks(tick_marks)
    s = [['TN','FP'], ['FN', 'TP']]

    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
    plt.show()

    # Curva ROC

    fpr, tpr, _ = roc_curve(y_test, Y_pred)
```

```
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.
 ↪2f)' % auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic (ROC Curve))')
    plt.legend(loc="lower right")
    plt.show()

    return accuracy_train, accuracy_test

# train and evaluate an ELM model
run_single_classificator(X_train, Y_train, X_test, Y_test, labels, 1,␣
 ↪best_Nneuronios)
```

```
Accuracia - Conjunto de Treinamento:  97.48744 %

Accuracia - Conjunto de Teste:  96.49123 %

              precision    recall  f1-score   support

   malignant       0.98      0.92      0.95        66
      benign       0.95      0.99      0.97       105

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.97      0.96      0.96       171
```


ELM Confusion Matrix - Test Data
TN = 61
FP = 5
FN = 1
TP = 104
True label
Predicted label
0
1
0
1

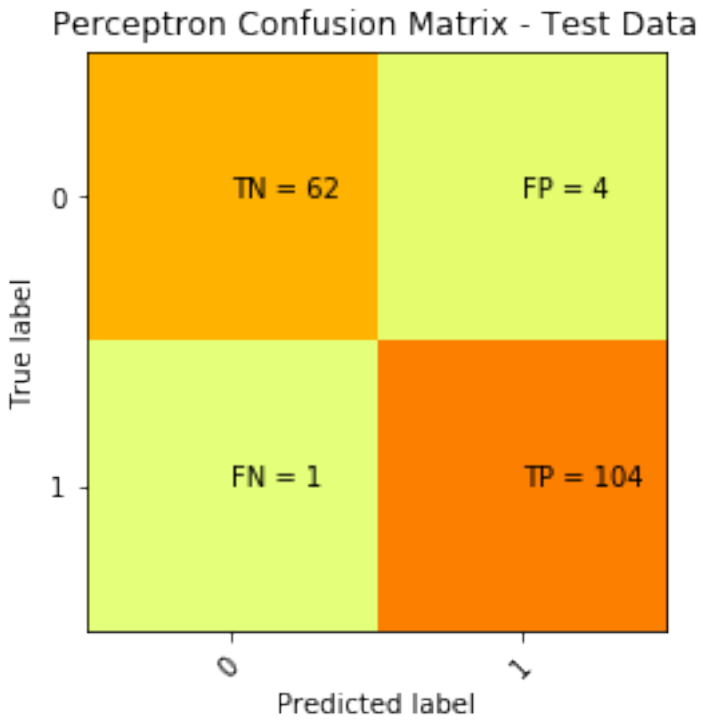

Receiver operating characteristic (ROC Curve))
True Positive Rate
False Positive Rate
ROC curve (area = 0.96)
0.0
0.2
0.4
0.6
0.8
1.0

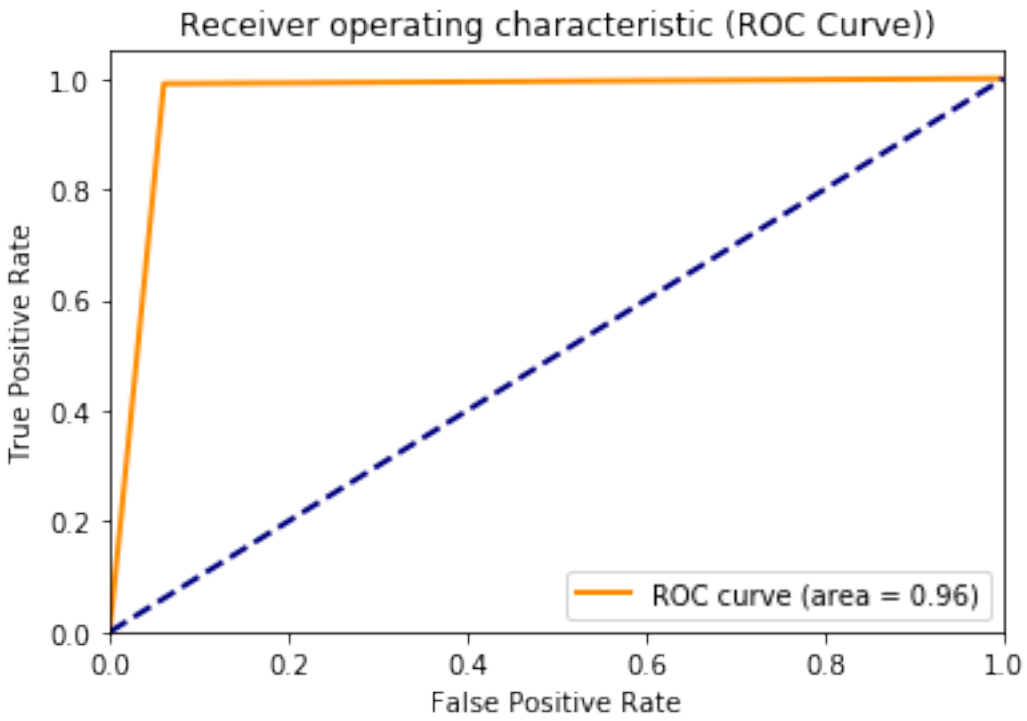
[24]: (0.9748743718592965, 0.9649122807017544)

Perceptron A seguir será avaliado a performance do perceptron (acurácia sore os dados de treinamento, testes, matriz de confusão e curva ROC). O resultado otdo será comparado com o ELM de 30 neurônios na camada intermediária.

[25]:
```
# train and evaluate an Perceptron model
run_single_classificator(X_train, Y_train, X_test, Y_test, labels, 0)
```

```
Accuracia - Conjunto de Treinamento:  100.0 %

Accuracia - Conjunto de Teste:  97.07602 %

              precision    recall  f1-score   support

   malignant       0.98      0.94      0.96        66
      benign       0.96      0.99      0.98       105

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```





[25]: (1.0, 0.9707602339181286)

Ambos os modelos apresentaram performances muito semelhantes, otendo o mesmo valor AUC igual a 0,96.

1.1.2 *Statlog (Heart)*

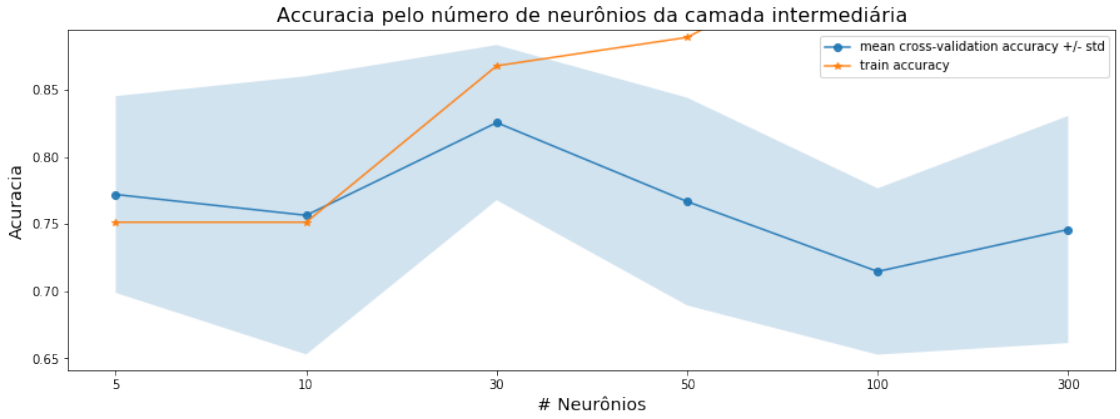
A segunda base de dados a ser estudada é a base Stalog (Heart). O código a seguir faz a importação e tratamento dos dados.

[26]:
```
data = []

with open('heart.dat','r') as token:
    for line in token:
        data.append(line.split())

data = np.asarray(data)

Y = np.squeeze(data[:,-1].astype(np.int))
X = data[:,0:-1]

# Standardization - Todos os atributos agora terão média zero e variação␣
 ↪unitária
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)

# Mapeia os rótulos binários de forma que negativo = -1 e positivo = +1
Y[Y==1] = int(-1)
Y[Y==2] = int(1)

# Separa os dados de forma aleatória - 70% para treinamento e 30% para testes
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

ELM O código a seguir plota a performance estimada com +/- 1 desvio padrão para diferentes quantidades de neurônios na camada intermediária, utilizando validação cruzada com 10 folds.

[27]:
```
# lista com quantidade de neurônios (p) que serão avaliados
Nneuronios = [5, 10, 30, 50, 100, 300]


cv_scores_mean, cv_scores_std, accuracy_scores_train =␣
↪run_cross_validation_on_elms(X_train,

                                                        ␣
↪Y_train,Nneuronios, cv=10)

# plotting accuracy
plot_cross_validation_results(Nneuronios, cv_scores_mean, cv_scores_std,␣
↪accuracy_scores_train,
                              'Accuracia pelo número de neurônios da camada␣
↪intermediária')
```



[28]:
```
idx_max = cv_scores_mean.argmax()
best_Nneuronios = Nneuronios[idx_max]
best_score = cv_scores_mean[idx_max]
best_score_std = cv_scores_std[idx_max]
```

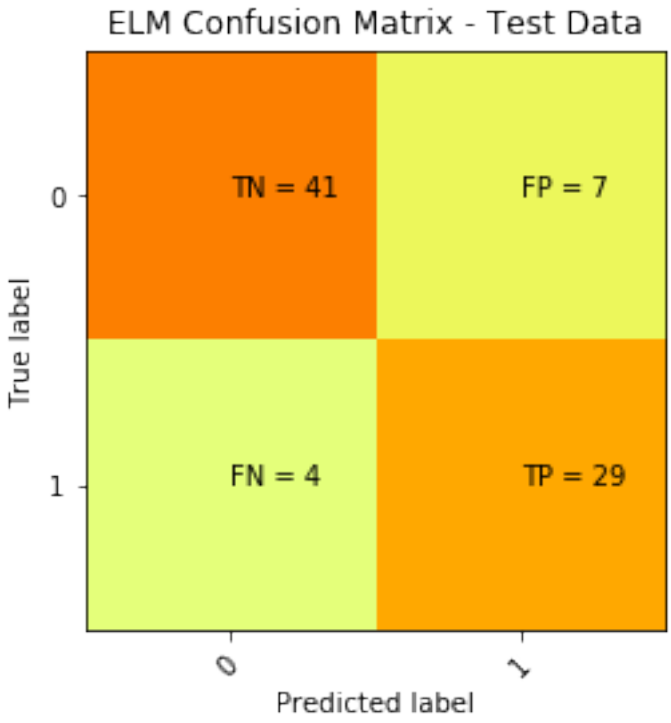
```
print('Ao utilizar {} neurônios na camada intermediária foi atingindo a melhor␣
 ↪acurácia média de validação cruzada sobre o conjunto de treinamento: {} +/-␣
 ↪{}%'.format(best_Nneuronios, round(best_score*100,5),␣
 ↪round(best_score_std*100, 5)))
```

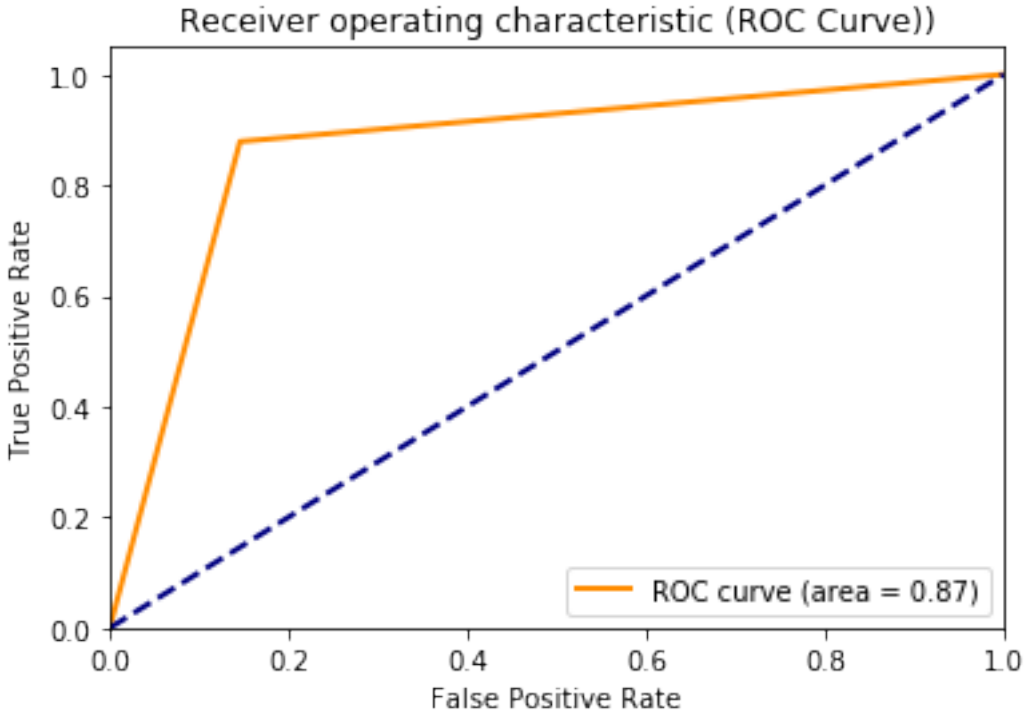
```
Ao utilizar 30 neurônios na camada intermediária foi atingindo a melhor acurácia
média de validação cruzada sobre o conjunto de treinamento: 82.54386 +/- 5.7715%
```

Ao utilizar mais de 30 neurônios a acurácia de treinamento aumenta ao passo que a acurácia média de validação cruzada diminui, evidenciando que ocorre sobreajuste. Ao utilizar 30 neurônios evita-se overfitting e há uma maior chance de se aproximar do erro esperado, generalizando o modelo nos dados de teste, conforme apresentado abaixo. Destaca-se a grande variância da acurácia de validação cruzada, apontando que há poucos dados nesse problema.

[29]:
```
# train and evaluate an ELM model
run_single_classificator(X_train, Y_train, X_test, Y_test, np.
 ↪array(['negative', 'positive']), 1, best_Nneuronios)
```

```
Accuracia - Conjunto de Treinamento:  86.77249 %

Accuracia - Conjunto de Teste:  86.41975 %

              precision    recall  f1-score   support

    negative       0.91      0.85      0.88        48
    positive       0.81      0.88      0.84        33

    accuracy                           0.86        81
   macro avg       0.86      0.87      0.86        81
weighted avg       0.87      0.86      0.86        81
```


ELM Confusion Matrix - Test Data
TN = 41
FP = 7
FN = 4
TP = 29
0
1
True label
0
1
Predicted label

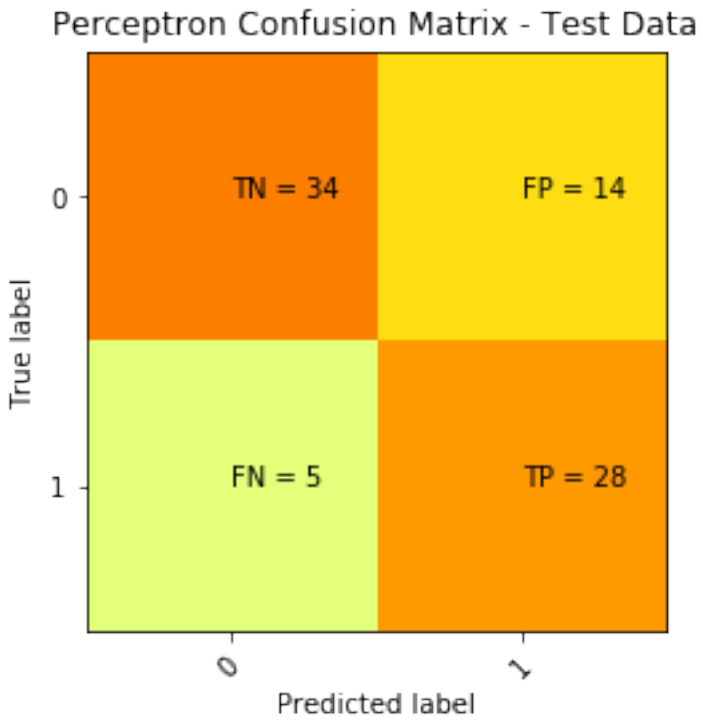

Receiver operating characteristic (ROC Curve))
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.87)

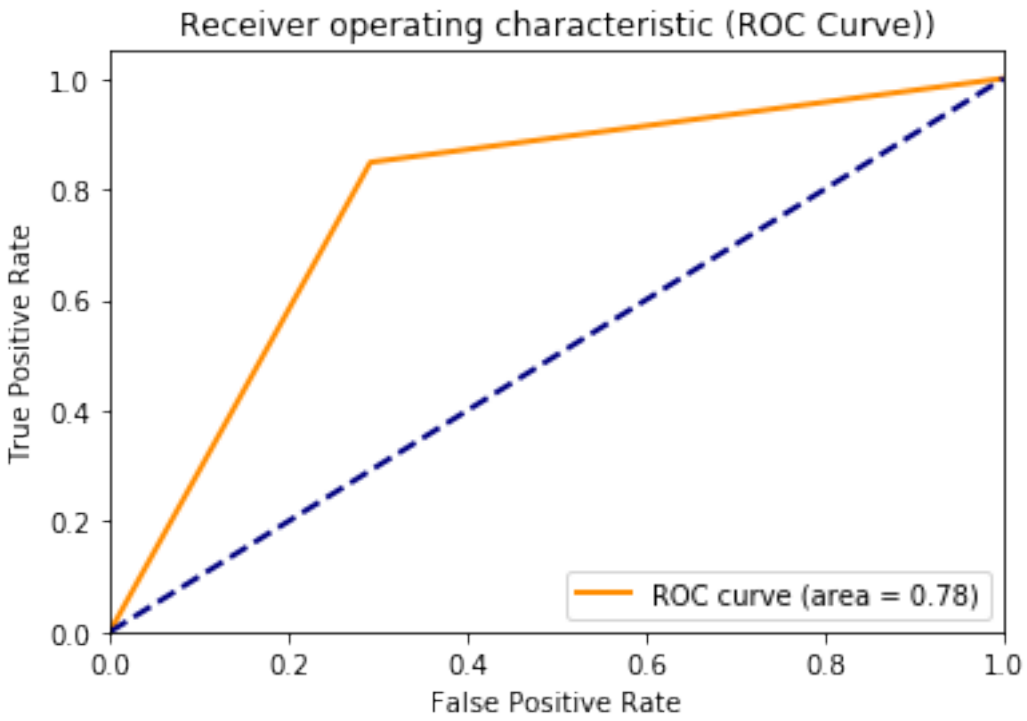
[29]: (0.8677248677248677, 0.8641975308641975)

Perceptron A seguir será avaliado a performance do perceptron (acurácia sore os dados de treinamento, testes, matriz de confusão e curva ROC). O resultado obtido será comparado com o ELM de 30 neurônios na camada intermediária.

[30]:
```
# train and evaluate an Perceptron model
run_single_classificator(X_train, Y_train, X_test, Y_test, np.
 ↪array(['negative', 'positive']), 0)
```

```
Accuracia - Conjunto de Treinamento:  77.24868 %

Accuracia - Conjunto de Teste:  76.54321 %

              precision    recall  f1-score   support

    negative       0.87      0.71      0.78        48
    positive       0.67      0.85      0.75        33

    accuracy                           0.77        81
   macro avg       0.77      0.78      0.76        81
weighted avg       0.79      0.77      0.77        81
```





[30]: (0.7724867724867724, 0.7654320987654321)

O modelo ELM com 30 neurônios na camada intermediária foi superior ao modelo Perceptron em todas as métricas de performance observadas, obtendo uma acurácia cerrca de 10% maior sobre o coinjunto de teste, evidenciando que há uma não linearidade na superfície de separação dos dados.