Aprendizagem de Máquina
Prof. Tiago Buarque A. de Carvalho

Exercícios sobre Comparação de Classi cadores

Atenção: a palavra signi cativa é utilizada para indicar uma diferença grande o su ciente que é atestada através de um teste de hipótese.

1. (25 pontos) Realize 100-fold cross validation estratificado na base Skin Segmentation utilizando o classificador 1-NN com distância Euclidiana então realize os procedimentos abaixo.

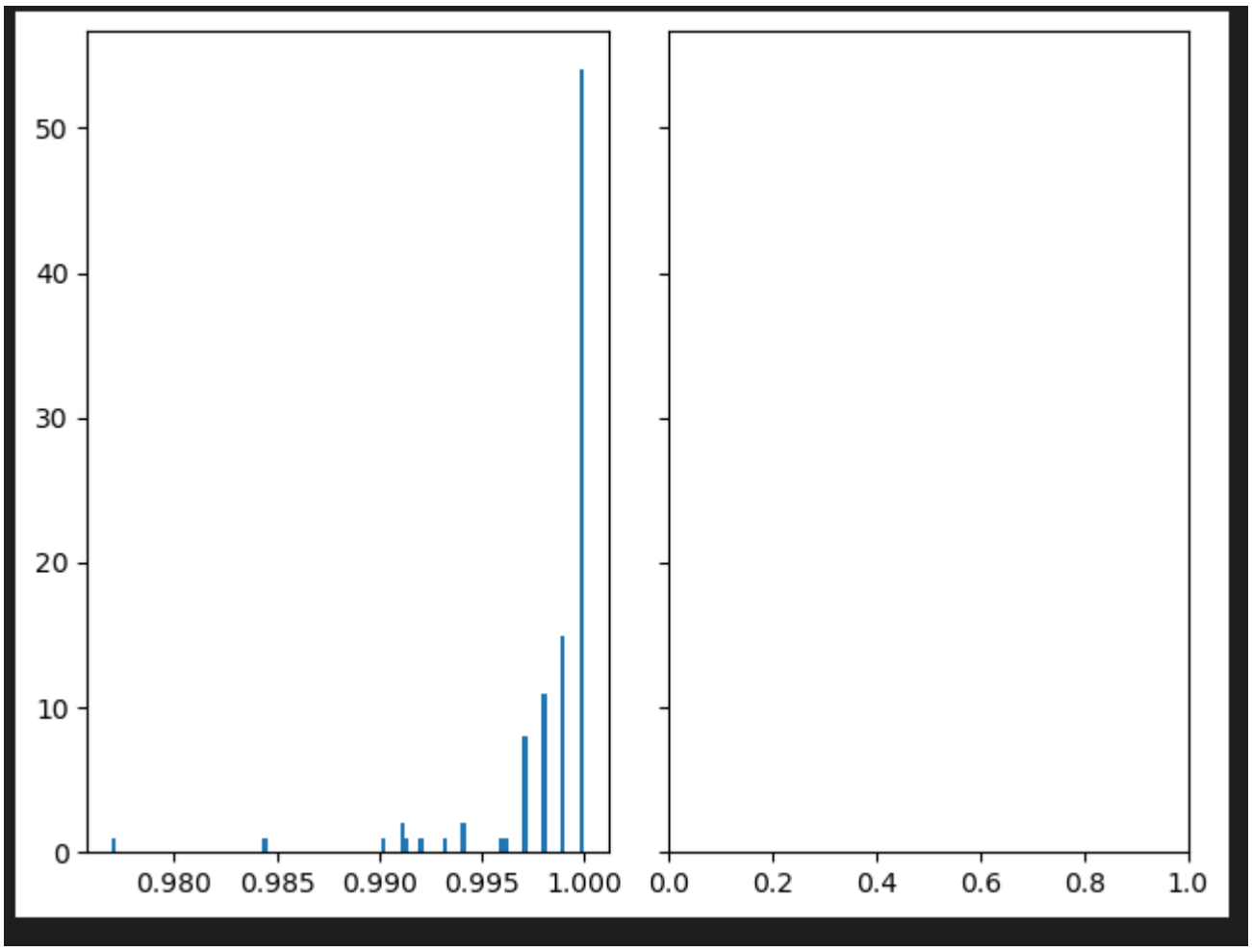
   (a) Mostre a média, o máximo e o mínimo da medida-F.

```
def calc_f_measure(classifier):
    list_rates = []
    list_f_measure = []

    for train_index, test_index in skf.split(x,y):
        # TREINO
        x_train = x[train_index]
        y_train = y[train_index]
        # TESTE
        x_test = x[test_index]
        y_test = y[test_index]

        classifier.fit(x_train, y_train)
        rate = classifier.score(x_test, y_test)
        list_rates.append(rate)

        y_pred = classifier.predict(x_test)
        recall = recall_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)

        f_measure = ((2*precision*recall)/(precision+recall))
        list_f_measure.append(f_measure)

    average_fMeasure = np.mean(list_f_measure)
    min_fMeasure = min(list_f_measure)
    max_fMeasure = max(list_f_measure)

    print("\n MÉDIA: \n")
    print("%.3f" % average_fMeasure)
    print("\n MÁXIMO: \n")
    print("%.3f" % max_fMeasure)
    print("\n MÍNIMO: \n")
    print("%.3f" % min_fMeasure)

    return list_f_measure, average_fMeasure
```

```
 MÉDIA:

0.998

 MÁXIMO:

1.000

 MÍNIMO:

0.977
```

(b) Mostre o histograma da medida-F.

```python
def histogram_f_measure(list_f_measure):
    n_bins = len(list_f_measure)
    fig, axs = plt.subplots(1, 2, sharey=True, tight_layout=True)
    axs[0].hist(list_f_measure, bins=n_bins)
```



(c) Calcule o intervalo de confiança da medida-F.

```python
def calc_confidence_interval(average_fMeasure, list_f_measure):
    desvio_padrao = np.std(list_f_measure)

    erro_padrao = desvio_padrao/np.sqrt(len(list_f_measure))

    multiplier = abs(stats.distributions.norm.ppf(0.025))

    confidence_interval = (average_fMeasure - multiplier*erro_padrao, average_fMeasure + multiplier*erro_padrao)
    print("\n INTERVALO DE CONFIANÇA \n")
    print(confidence_interval)

    return confidence_interval
```

```
 INTERVALO DE CONFIANÇA

(0.997643082377952, 0.9989729820531587)
```

(d) Qual a medida-F mínima que você espera ao aplicar este classi cador, sob as mesmas condições de treinamento, para dados nunca vistos?

```
O valor minimo do intervalo de confiança: 0.997643082377952
```

(e) Qual a medida-F esperada para o classi cador quando aplicada a dados nunca antes vistos.

A base Skin Segmentation (archive.ics.uci.edu/ml/datasets/Skin+Segmentation) tem três 4 colunas, as três primeiras são atributos e a última é a classe.

```
Valores dentro do intervalo de confiança: (0.997643082377952, 0.9989729820531587)
```

2. (25 pontos) Realize um experimento pareado com 100 repetições de Holdout 50/50 utilizando o classi cador 1-NN com distância Euclidiana. Utilize duas versões da base Wine archive. ics.uci.edu/ml/datasets/Wine para este experimento, a primeira versão é a base original, a segunda versão é a base sem a última coluna. Após calcular 100 taxas de acerto para cada uma das versões da base, realize os procedimentos abaixo.

(a) Calcule a diferença das 100 taxas de acerto.

```python
def calc_rating_n_columns(dataframe, n_columns, n_rates, porcent_test):
    list = []
    column = len(dataframe.columns)

    classificador = KNeighborsClassifier(n_neighbors=1)

    for n in range(n_columns):
        list_rates = []
        x = (dataframe[dataframe.columns[1:column:]])
        column -= 1

        for i in range(n_rates):
            x_treino, x_teste, y_treino, y_teste = train_test_split (x, y,test_size=porcent_test, random_state=i)
            classificador.fit(x_treino, y_treino)
            rates = classificador.score(x_teste, y_teste)
            list_rates.append(rates)

        list.append(list_rates)

    return list

lists_rates = calc_rating_n_columns(df, 2, 100, 0.5)
```

✓ 2.1s

```python
for k in range(len(lists_rates)):
    print(k, lists_rates[k], '\n')
```

✓ 0.1s

Taxas:

```
0 [0.7415730337078652, 0.6966292134831461, 0.7191011235955056, 0.7191011235955056, 0.8202247191011236, 0.6853932584269663, 0.7640449438202247, 0.6629213483146067,
0.7415730337078652, 0.6741573033707865, 0.7303370786516854, 0.7752808988764045, 0.7752808988764045, 0.7415730337078652, 0.797752808988764, 0.7191011235955056,
0.7078651685393258, 0.7078651685393258, 0.6741573033707865, 0.6404494382022472, 0.6966292134831461, 0.7752808988764045, 0.7191011235955056, 0.7640449438202247,
0.8089887640449438, 0.7191011235955056, 0.7415730337078652, 0.7415730337078652, 0.7303370786516854, 0.5955056179775281, 0.6179775280898876, 0.7528089887640449,
0.7303370786516854, 0.651685393258427, 0.8202247191011236, 0.7078651685393258, 0.7640449438202247, 0.6853932584269663, 0.7415730337078652, 0.7640449438202247,
0.7303370786516854, 0.6853932584269663, 0.6853932584269663, 0.7865168539325843, 0.7191011235955056, 0.6741573033707865, 0.7303370786516854, 0.7303370786516854,
0.651685393258427, 0.6853932584269663, 0.6853932584269663, 0.7191011235955056, 0.7415730337078652, 0.6629213483146067, 0.7303370786516854, 0.7528089887640449,
0.7303370786516854, 0.6966292134831461, 0.651685393258427, 0.7528089887640449, 0.7528089887640449, 0.6629213483146067, 0.7078651685393258, 0.6966292134831461,
0.7078651685393258, 0.7528089887640449, 0.6853932584269663, 0.7415730337078652, 0.7415730337078652, 0.7078651685393258, 0.7528089887640449, 0.6853932584269663,
0.6966292134831461, 0.6966292134831461, 0.797752808988764, 0.6966292134831461, 0.7528089887640449, 0.7415730337078652, 0.7415730337078652, 0.7640449438202247,
0.7303370786516854, 0.6966292134831461, 0.797752808988764, 0.7303370786516854, 0.651685393258427, 0.7078651685393258, 0.5955056179775281, 0.6966292134831461,
0.7303370786516854, 0.797752808988764, 0.6067415730337079, 0.651685393258427, 0.7303370786516854, 0.6404494382022472, 0.7640449438202247, 0.7303370786516854,
0.6966292134831461, 0.6741573033707865, 0.7415730337078652, 0.6741573033707865]

1 [0.8764044943820225, 0.8651685393258427, 0.8314606741573034, 0.8314606741573034, 0.8764044943820225, 0.9213483146067416, 0.8539325842696629, 0.8539325842696629,
0.797752808988764, 0.797752808988764, 0.797752808988764, 0.8089887640449438, 0.898876404494382, 0.8764044943820225, 0.898876404494382, 0.797752808988764,
0.8651685393258427, 0.797752808988764, 0.8426966292134831, 0.7640449438202247, 0.8539325842696629, 0.8651685393258427, 0.8314606741573034, 0.9101123595505618,
0.9101123595505618, 0.7415730337078652, 0.797752808988764, 0.8314606741573034, 0.8539325842696629, 0.8539325842696629, 0.797752808988764, 0.797752808988764,
0.8314606741573034, 0.9213483146067416, 0.8089887640449438, 0.797752808988764, 0.8426966292134831, 0.797752808988764, 0.8651685393258427, 0.797752808988764,
0.8876404494382022, 0.8089887640449438, 0.7865168539325843, 0.8651685393258427, 0.7752808988764045, 0.8651685393258427, 0.8202247191011236, 0.797752808988764,
0.8426966292134831, 0.8314606741573034, 0.8089887640449438, 0.8426966292134831, 0.8764044943820225, 0.8089887640449438, 0.8426966292134831, 0.7865168539325843,
0.7865168539325843, 0.797752808988764, 0.8202247191011236, 0.8651685393258427, 0.8426966292134831, 0.8651685393258427, 0.8202247191011236, 0.8089887640449438,
0.7752808988764045, 0.8876404494382022, 0.8314606741573034, 0.8089887640449438, 0.7865168539325843, 0.8202247191011236, 0.8764044943820225, 0.8651685393258427,
0.8539325842696629, 0.8539325842696629, 0.8651685393258427, 0.8089887640449438, 0.8539325842696629, 0.8764044943820225, 0.8202247191011236, 0.8202247191011236,
0.797752808988764, 0.8202247191011236, 0.8202247191011236, 0.7865168539325843, 0.7752808988764045, 0.797752808988764, 0.8651685393258427, 0.8314606741573034,
0.8876404494382022, 0.7865168539325843, 0.8202247191011236, 0.8089887640449438, 0.8876404494382022, 0.797752808988764, 0.8089887640449438, 0.8539325842696629,
0.8651685393258427, 0.8089887640449438, 0.8089887640449438, 0.8089887640449438]
```

```python
def calc_differences(lists_rates):
    total_differences = {}
    current_differences = []
    local_differences = []

    for x in range(len(lists_rates)-1):
        for y in range(len(lists_rates)-1):
            for z in range(len(lists_rates[y])):
                if y >= x:
                    # 0 lists_rates[0][0] - lists_rates[1][0]
                    # 1 lists_rates[0][1] - lists_rates[1][1]
                    # 1 lists_rates[0][...] - lists_rates[1][...]
                    current_differences.append(lists_rates[x][z] - lists_rates[y+1][z])

            if current_differences not in local_differences:
                if current_differences:
                    local_differences.append(current_differences)

            current_differences = []

        total_differences[x+1] = local_differences
        local_differences = []

    return total_differences

total_differences = calc_differences(lists_rates)
```

✓ 0.2s

```python
print(total_differences)
```

✓ 0.2s

Array de diferenças:


```
{1: [[-0.1348314606741573, -0.1685393258426966, -0.1123595505617978, -0.1123595505617978, -0.0561797752808989, -0.2359550561797753, -0.0898876404494382,
-0.1910112359550562, -0.05617977528089879, -0.1235955056179775, -0.0674157303370786, -0.0337078651685393, -0.1235955056179775, -0.1348314606741573, -0.101123595505618,
-0.0786516853932584, -0.1573033707865169, -0.0898876404494382, -0.1685393258426966, -0.1235955056179775, -0.1573033707865168, -0.0898876404494382, -0.1123595505617978,
-0.1460674157303371, -0.101123595505618, -0.022471910112359605, -0.05617977528089879, -0.0898876404494382, -0.1235955056179775, -0.2584269662921348, -0.1797752808988764,
-0.0449438202247191, -0.101123595505618, -0.2696629213483146, 0.011235955056179803, -0.0898876404494382, -0.0786516853932584, -0.1123595505617977, -0.1235955056179775,
-0.0337078651685393, -0.1573033707865168, -0.1235955056179775, -0.101123595505618, -0.0786516853932584, -0.0561797752808989, -0.1910112359550562, -0.0898876404494382,
-0.0674157303370786, -0.1910112359550561, -0.1460674157303371, -0.1235955056179775, -0.1235955056179775, -0.1348314606741573, -0.1460674157303371, -0.1123595505617977,
-0.03370786516853941, -0.0561797752808989, -0.10112359550561789, -0.1685393258426966, -0.1123595505617978, -0.0898876404494382, -0.202247191011236, -0.1123595505617978,
-0.1123595505617977, -0.0674157303370787, -0.1348314606741573, -0.1460674157303371, -0.0674157303370786, -0.0449438202247191, -0.1123595505617978, -0.12359550561797761,
-0.1797752808988764, -0.1573033707865168, -0.1573033707865168, -0.0674157303370787, -0.1123595505617977, -0.101123595505618, -0.1348314606741573, -0.0786516853932584,
-0.0561797752808989, -0.0674157303370786, -0.1235955056179775, -0.022471910112359605, -0.0561797752808989, -0.1235955056179775, -0.0898876404494382, -0.2696629213483146,
-0.1348314606741573, -0.1573033707865168, 0.011235955056179692, -0.2134831460674157, -0.1573033707865168, -0.1573033707865168, -0.1573033707865168, -0.0449438202247191,
-0.1235955056179775, -0.1685393258426966, -0.1348314606741573, -0.0674157303370786, -0.1348314606741573]]}
```


(b) Calcule o intervalo de confiança destas diferenças.

```python
def calc_confidence_interval(total_average, total_differences):
    for i in range(len(total_average)):
        for x in range(len(total_average[i+1])):

            standard_deviation = np.std(total_differences[i+1][x])
            standard_error = standard_deviation/np.sqrt(len(total_differences[i+1][x]))
            multi = abs(stats.distributions.norm.ppf(0.025))

            print(i+1, (total_average[i+1][x] - multi*standard_error, total_average[i+1][x] + multi*standard_error))

confidence_interval = calc_confidence_interval(total_average, total_differences)
```

✓ 0.1s

```
1 (-0.1257323180768185, -0.10438004147374327)
```


```
1 (-0.1257323180768185, -0.10438004147374327)
```

(c) Realize o teste de hipótese sobre estas diferenças para verificar se a diferença da taxa de acerto é significativamente maior entre as duas versões. Mostre sua conclusão para o teste.

Rejeita-se H0, pois o 0 não está no intervalo, portanto as médias de acerto dos classificadores são diferentes.

A primeira versão, com uma tabela a mais e mais informações, é maior do que a segunda versão.

(d) Calcule o intervalo de confiança da taxa de acerto para cada versão da base.

```
1 (0.7083788816487575, 0.7266772981265237)
2 (0.8252175672280455, 0.8399509720977969)
```

(e) Realize o teste de hipótese de sobreposição dos intervalos de confiança. Mostre sua conclusão para o teste.

1 (0.7083788816487575, 0.7266772981265237)

2 (0.8252175672280455, 0.8399509720977969)

0.7266772981265237 < 0.8252175672280455

Não existe sobreposição, portanto a segunda versão tem um intervalo de confiança maior

3. (25 pontos) Qual o número máximo de características que podem ser removidas da base Iris archive.ics.uci.edu/ml/datasets/iris sem reduzir significativamente a taxa de acerto? De na a metodologia utilizada para justificar sua resposta.

Não existe sobreposição entre os intervalos.

Se observarmos os intervalos de confiança, reduzindo 0 e 1 colunas, a taxa de acerto é bem próxima, já reduzindo 2 e por diante, a taxa de acerto cai drasticamente

```
1 (0.9392543625860171, 0.9514123040806496)
2 (0.917420609527016, 0.937690501584095)
3 (0.6997671170240668, 0.7286773274203776)
4 (0.5832407909731356, 0.6238703201379756)
```

4. (25 pontos) Utilizando o classificador *k*-NN na base Wine archive.ics.uci.edu/ml/datasets/Wine, teste os valores *k* = 1*, . . . ,* 15. Para qual valor de *k* o classificador apresenta uma taxa de acerto significativamente maior? De na a metodologia utilizada para justificar sua resposta.

O K-9 tem um intervalo de confiança maior, portanto uma taxa de acerto maior que todas as

outras.

```
int_conf:  1 (0.8278069811776297, 0.8590844045901608) 0.03127742341
int_conf:  2 (0.7517376782081853, 0.7883372281588561) 0.03659954995
int_conf:  3 (0.7871886077863689, 0.8240473472698112) 0.03685873948
int_conf:  4 (0.7637935844656589, 0.7972551046729177) 0.0334615202
int_conf:  5 (0.760634570546471, 0.7951706728992218) 0.03453610235
int_conf:  6 (0.7450133992562152, 0.7785821063617623) 0.0335687071
int_conf:  7 (0.742102976656216, 0.7755000195984655) 0.03339704294
int_conf:  8 (0.7268687241753458, 0.7630189162740926) 0.03615019209
int_conf:  9 (0.7221555790933124, 0.7602414246520058) 0.03808584555
int_conf:  10 (0.7047592570220123, 0.7394354995322949) 0.03467624251
int_conf:  11 (0.7006482820456401, 0.7345577104637231) 0.03390942841
int_conf:  12 (0.6925011245518988, 0.7224801488563413) 0.0299790243
int_conf:  13 (0.6855728748990733, 0.7204196344642229) 0.03484675956
int_conf:  14 (0.670194082046534, 0.7058358805002827) 0.03564179845
int_conf:  15 (0.6681356736485805, 0.7019017795349402) 0.03376610588
```