Project_final

April 6, 2021

1 Machine Learning - Assignment 1

- **Group B**:
- Ana Raquel Maceiras (up200604342)
- Hélder Vieira (up201503395)
- Miguel Tavares (up200902937)
- Rui Vieira (up201403035)

1.1 Introdução

No mundo actual existem cada vez mais dados disponíveis sobre o dia a dia de um cidadão normal, quer seja através da utilização dos mais diversos dispositivos como através da utilização de serviços colectivos em grandes centros urbanos. De forma a ser possível prever comportamentos ou tendências da população em geral, a procura por modelos de aprendizagem tem aumentado. Grandes parte destes modelos baseiam-se na classificação de determinada categoria(s). Contudo, em situações reais, a linha real de separação entre as diferentes categorias alvo não é conhecida restando a tentativa de obtenção de uma linha o mais próxima possível desta. Assim, estes modelos oferecem-nos potenciais regiões de classificação com uma determinada confiança. Existem diversos métodos, muitos de aplicação directa, outros exigem o aprefeiçoamento de hiperparâmetros. Todo este processo implica um estudo prévio e posterior tratamentos dos dados, os quais servirão como base de treino do modelo. Cada modelo terá uma espressividade única, que poderá incorrer em overfitting ou underfitting, fortemente relacionado com a rácio bias/variance. Neste trabalho procuramos analisar o comportamento de diversos métodos de machine learning para dois casos distintos de dados e de observar o impacto do aprefeiçoamento dos hiperparametros na performance dos mesmos. Na primeira parte iremos observar que alguns modelos, mesmo sendo pouco fléxiveis podem oferecer uma boa resposta para situações específicas, e onde também os modelos mais fléxiveis convergem para complexidades similares, obtendo-se fronteiras de decisão semelhantes. Na segunda e última parte deste trabalho observaremos que, conjuntos de dados com classes pouco representadas irão dar lugar a modelos que não as consideram sem que para isso, seja investido um grande esforço no aprefeiçoamento.

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore')
from matplotlib import colors
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, StratifiedKFold,␣
↪cross_val_score, validation_curve, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier as kNN
from sklearn.linear_model import LogisticRegression as LogReg
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.tree import DecisionTreeClassifier as DTreeClass
from sklearn.metrics import accuracy_score, zero_one_loss, roc_auc_score,␣
↪roc_curve, classification_report, f1_score
from scipy import stats
```

[2]:
```
names = ['Sequence␣
↪name','mcg','gvh','alm','mit','erl','pox','vac','nuc','class'];
data_yeast = pd.read_csv('yeast.data', header = None, sep = '\s+', names =␣
↪names);
data_pima = pd.read_csv('diabetes.csv');
```

1.2 Question 1

[3]:
```
cols = data_pima.columns[:8]
fig, ax = plt.subplots(2,4, sharex=False, sharey=False, figsize=(20, 8))
txt="Figura 1. Box plots para as variáveis preditivas de acordo com a classe da␣
↪variável alvo."
plt.figtext(0.5, 0.01, txt, wrap=True, horizontalalignment='center',␣
↪fontsize=15)
for i in range(len(cols)):
    sns.boxplot(ax=ax[i//4, i%4], y=cols[i], x='Outcome', data=data_pima)
    ax[i//4, i%4].set(xlabel=cols[i])
plt.show();
```

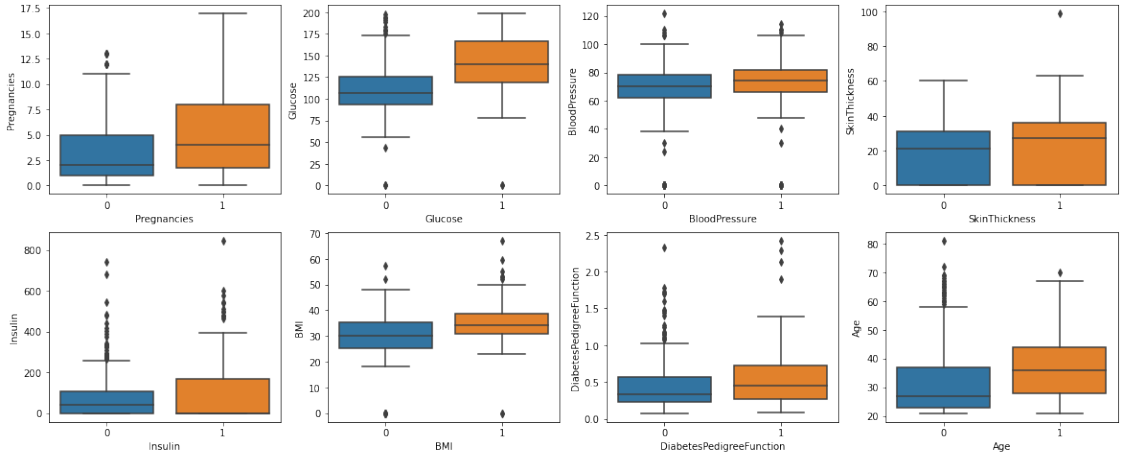

Figura 1. Box plots para as variáveis preditivas de acordo com a classe da variável alvo.

[4]:
```
data_pima_sel = data_pima.drop(columns = ['BloodPressure', 'SkinThickness',␣
 ↪'Pregnancies', 'DiabetesPedigreeFunction', 'Insulin'])
sns.pairplot(data_pima_sel, hue = 'Outcome', size=1.8)
txt="Figura 2. Gráficos de pares para três das variáveis preditivas (glucose,␣
 ↪BMI, age) em função das variável alvo"
plt.figtext(0.5, -0.03, txt, wrap=True, horizontalalignment='center',␣
 ↪fontsize=13)
plt.show();
```

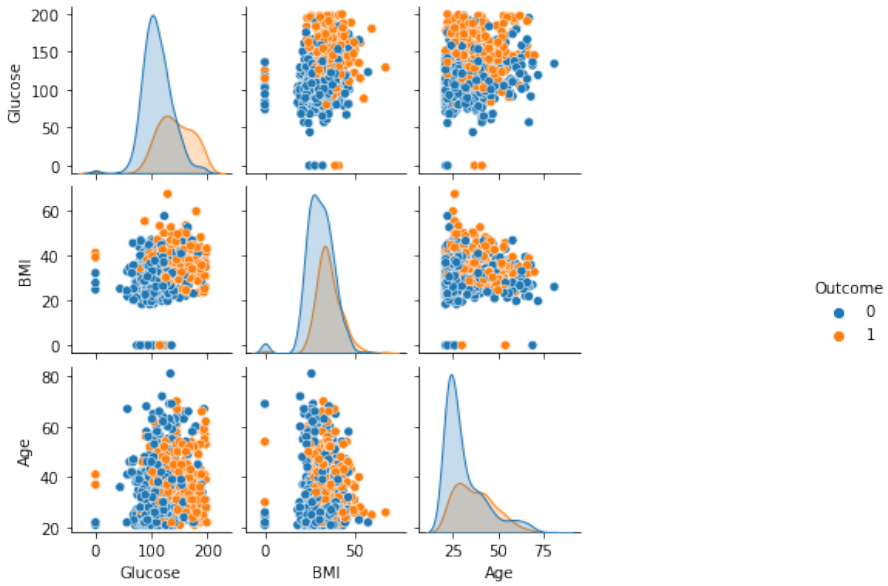

Figura 2. Gráficos de pares para três das variáveis preditivas (glucose, BMI, age) em função das variável alvo

Após análise aos gráficos apresentados, podemos observar rapidamente que as variáveis que melhor "separam" a variável alvo são *Glucose*, *BMI* e *Age*. Contudo, as duas escolhidas (como requisitado pelo exercicio) foram a *Glucose* e *BMI*. A variável *Age* foi descartada pois tinha mais outliers.

1.3 Methods comparison (kNN, Logistic Regression, QDA)

Após a escolha das duas variáveis preditivas, foram removidos alguns dados pois os mesmos representavam situação não reais (p.e. valores de glucose nulos). Para além disso, uma vez que um dos modelos depende do cálculo de distâncias (kNN), procedeu-se à normalização dos valores para as variáveis *Glucose* e *BMI*.

[5]:
```
data_pima_clean = data_pima_sel[(data_pima_sel['Glucose'] != 0) &␣
 ↪(data_pima_sel['BMI'] != 0)]
data_pima_clean = data_pima_clean.sample(frac=1).reset_index(drop=True)
data_pima_sel_final = data_pima_clean.drop(columns = 'Age')
```

Separation in train and test data and data normalization

[6]:
```
X = data_pima_sel_final.drop(columns = 'Outcome').values
y = data_pima_sel_final.loc[:,'Outcome'].values
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=123, stratify= y)

minmax_scaler = MinMaxScaler()
x_train = minmax_scaler.fit_transform(x_train)
x_test = minmax_scaler.transform(x_test)
```

[7]:
```
def plot_classifier_boundary(model,x,h = .05): #kindly provided in class
    cmap_light = colors.ListedColormap(['lightsteelblue', 'peachpuff'])
    x_min, x_max = x[:, 0].min()-.2, x[:, 0].max()+.2
    y_min, y_max = x[:, 1].min()-.2, x[:, 1].max()+.2
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=cmap_light)
    plt.xlim((x_min,x_max))
    plt.ylim((y_min,y_max))
cmap = colors.ListedColormap(['blue','orange'])
```

[8]:
```
def StratKF(model, splits,x_train, y_train, x_test, y_test): #fitted model␣
 ↪necessary
    SKF = StratifiedKFold(n_splits=splits, shuffle=True, random_state=1)
    score = []
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    for train_index, test_index in SKF.split(x_test, y_test):
        x_test1 = x_train[test_index]
        y_test1 = y_train[test_index]
        score.append(f1_score(y_test1, model.predict(x_test1)))
    print(f'F1 score on {splits}-fold test data: ',round(np.mean(score),4),'+/
 ↪-', round(np.std(score),4))
    print('F1 score on training set: ',round(f1_score(y_train,␣
 ↪y_pred_train),4), '\nF1 score on test set: ',round(f1_score(y_test,␣
 ↪y_pred_test), 4))
```

1.3.1 Logistic Regression

[9]:
```
modelLogReg = LogReg()
modelLogReg.fit(x_train, y_train)
StratKF(modelLogReg,5, x_train, y_train, x_test, y_test)
```

```
F1 score on 5-fold test data:  0.5016 +/- 0.1481
F1 score on training set:  0.5869
F1 score on test set:  0.5843
```

1.3.2 Quadratic Discriminant Analysis

[10]:
```
modelQDA = QDA()
modelQDA.fit(x_train, y_train)
StratKF(modelQDA,5, x_train, y_train, x_test, y_test)
```

```
F1 score on 5-fold test data:  0.5519 +/- 0.109
F1 score on training set:  0.6196
F1 score on test set:  0.6154
```

1.3.3 kNN

[11]:
```
fig = plt.figure(figsize=(30,10))
k = [1,10,50,100,200,300]
for i in range(len(k)):
    modelkNN = kNN(n_neighbors=k[i])
    modelkNN.fit(x_train, y_train)
    ax = fig.add_subplot(2, 3, i+1)
    plot_classifier_boundary(modelkNN, x_train)
    ax.scatter(x_train[:,0],x_train[:,1],color=cmap(y_train))
    ax.set_title(k[i], fontsize = 18)
    ax.set_xlabel('Glucose')
    ax.set_ylabel('BMI')
txt="Figura 3. Fronteiras de decisão obtidas com o modelo 'k Nearest␣
 ↪Neighbours' e usando diferentes valores de k: 1, 10, 50, 100, 200 e 300"
plt.figtext(0.5, 0.01, txt, wrap=True, horizontalalignment='center',␣
 ↪fontsize=22)
plt.show;
```

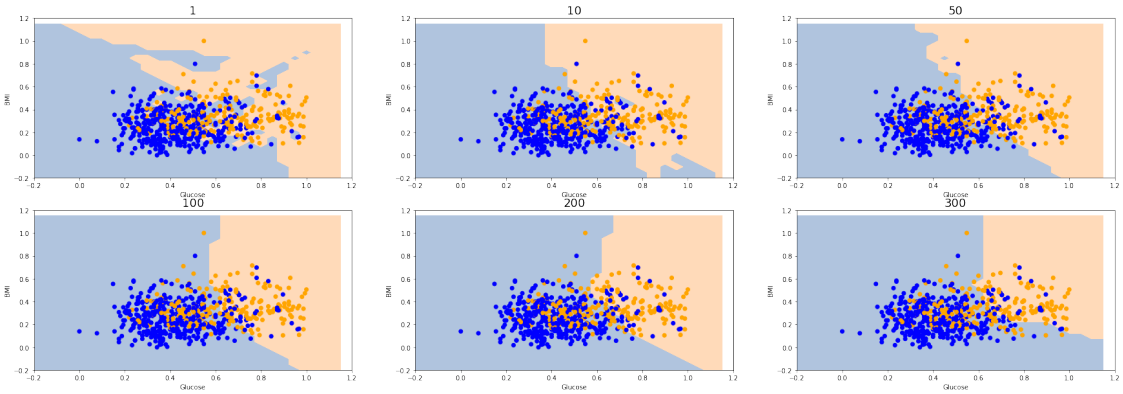

Figura 3. Fronteiras de decisão obtidas com o modelo 'k Nearest Neighbours' e usando diferentes valores de k: 1, 10, 50, 100, 200 e 300

Como esperado, e anteriormente referenciado, à medida que o k aumenta, a complexidade do modelo diminui. Observado a figura 3 podemos concluir que a fronteira de decisão vai se tornando mais simples.

[12]:
```
def valid_curve(model, x_train, y_train, n_jobs, scoring, param_range = np.
↪array([*range(1,301)]), param_name="n_neighbors"):
    np.random.seed(1)
    train_scores, test_scores = validation_curve(model, x_train, y_train,␣
↪param_name=param_name, param_range=param_range, scoring=scoring,␣
↪n_jobs=n_jobs)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.figure(figsize=(6, 3))
    plt.title("kNN Validation Curve")
    plt.xlabel("k Neighbors")
    plt.ylabel("Score")
    plt.plot(param_range, train_scores_mean, label="Training score",␣
↪color="darkorange")
    plt.fill_between(param_range, train_scores_mean - train_scores_std,␣
↪train_scores_mean + train_scores_std, alpha=0.1, color="darkorange")
    plt.plot(param_range, test_scores_mean, label="Cross-validation score",␣
↪color="navy")
    plt.fill_between(param_range, test_scores_mean - test_scores_std,␣
↪test_scores_mean + test_scores_std, alpha=0.1, color="navy")
    plt.legend(loc="best")
    print("Best K is %d" %param_range[np.where(test_scores_mean ==␣
↪max(test_scores_mean))][0])
    return param_range[np.where(test_scores_mean == max(test_scores_mean))][0]
```

[13]:
```
modelkNN = kNN()
k_best = valid_curve(modelkNN, x_train, y_train, 6, 'accuracy')
txt="Figura 4. Curva de validação para determinação do melhor k para o modelo␣
↪kNN para o dataset 'Pima'."
plt.figtext(0.5, -0.07, txt, wrap=True, horizontalalignment='center',␣
↪fontsize=13)
plt.show();
```

```
Best K is 63
```

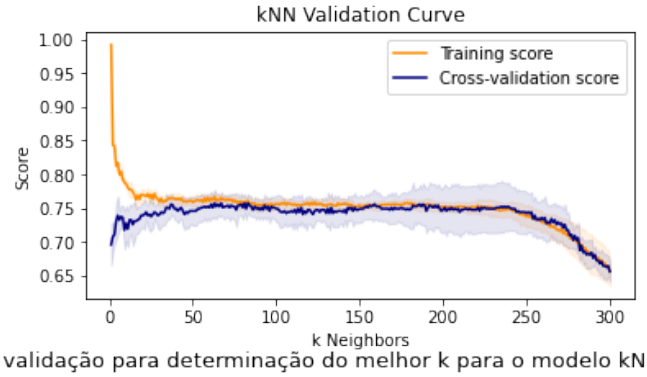

Figura 4. Curva de validação para determinação do melhor k para o modelo kNN para o dataset 'Pima'.

Como se pode visualizar na figura da fronteira de classificação e na curva de validação, inicialmente (aprox. até k igual 5) ocorre overfitting, seguidamente com o aumento do k existe uma tendência de estabilização da fronteira sendo que os scores de treino e CV são similares até um k de aproximadamente 250 em que começa a ocorrer underfitting. O k escolhido como ideal corresponde ao maximizante do score da CV.

[14]:
```
modelkNN = kNN(n_neighbors=k_best)
modelkNN.fit(x_train, y_train)
StratKF(modelkNN,5, x_train, y_train, x_test, y_test)
```

```
F1 score on 5-fold test data:  0.5858 +/- 0.0665
F1 score on training set:  0.6039
F1 score on test set:  0.5714
```

[15]:
```
#ROC AUC
ns_probs = [0 for _ in range(len(y_test))]
# probabilities for the positive outcome
LogReg_probs = modelLogReg.predict_proba(x_test)[:, 1]
QDA_probs = modelQDA.predict_proba(x_test)[:, 1]
kNN_probs = modelkNN.predict_proba(x_test)[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
LogReg_auc = roc_auc_score(y_test, LogReg_probs)
QDA_auc = roc_auc_score(y_test, QDA_probs)
kNN_auc = roc_auc_score(y_test, kNN_probs)
# summarize scores
#print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('LR ROC AUC=%.3f' % (LogReg_auc))
print('QDA ROC AUC=%.3f' % (QDA_auc))
print('kNN ROC AUC=%.3f' % (kNN_auc))
```

```
LR ROC AUC=0.836
QDA ROC AUC=0.830
kNN ROC AUC=0.828
```

[16]:
```
#ROC curves
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs, pos_label=None)
LogReg_fpr, LogReg_tpr, _ = roc_curve(y_test, LogReg_probs)
QDA_fpr, QDA_tpr, _ = roc_curve(y_test, QDA_probs)
kNN_fpr, kNN_tpr, _ = roc_curve(y_test, kNN_probs)
figs = plt.figure(figsize=(30,6))
# plot the roc curves
axs = figs.add_subplot(1, 5, 1)
axs.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
axs.plot(LogReg_fpr,LogReg_tpr, marker='+', label='Logistic Regression')
axs.plot(QDA_fpr,QDA_tpr, marker='+', label='QDA')
axs.plot(kNN_fpr,kNN_tpr, marker='+', label='kNN')
axs.set_xlabel('False Positive Rate')
axs.set_ylabel('True Positive Rate')
axs.set_title('ROC curves', fontsize = 18)
axs.legend();
##Barplot of AUCs
axs = figs.add_subplot(1, 5, 2)
sns.barplot(y=[LogReg_auc,QDA_auc,kNN_auc], x=['LogReg', 'QDA', 'kNN'], ax=axs)
axs.set_ylim(0,1)
axs.set_title('ROC AUC Comparison', fontsize = 18)
axs.set_ylabel('ROC AUC');
model_list = [(modelLogReg,'Logistic Regression Boundary') , (modelQDA, 'QDA␣
 ↪Boundary'), (modelkNN, 'kNN Boundary')]
for i in range(3,6,1):
    axs = figs.add_subplot(1, 5, i)
    plot_classifier_boundary(model_list[i-3][0],x_train)
    axs.scatter(x_train[:,0],x_train[:,1],color=cmap(y_train))
    axs.set_xlabel('Glucose')
    axs.set_ylabel('BMI');
    axs.set_title(model_list[i-3][1], fontsize = 18)
txt="Figura 5. Curvas ROC, gráfico de barras com os valores de ROC AUC, e␣
 ↪gráficos com as fronteiras de decisão para os três modelos testados."
plt.figtext(0.5, -0.02, txt, wrap=True, horizontalalignment='center',␣
 ↪fontsize=22)
plt.show();
```

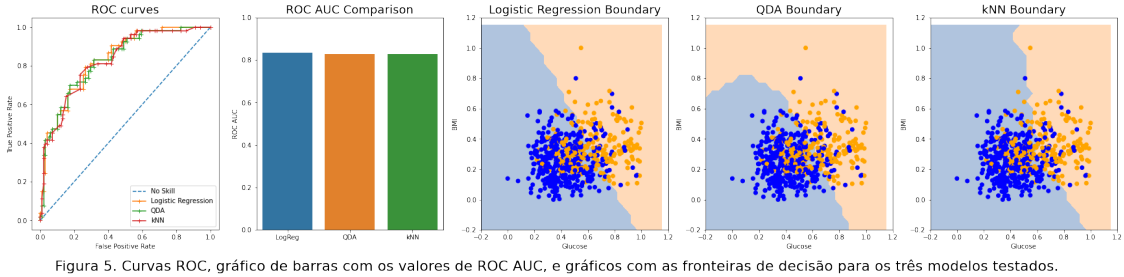

Figura 5. Curvas ROC, gráfico de barras com os valores de ROC AUC, e gráficos com as fronteiras de decisão para os três modelos testados.

Nesta análise comparámos os três modelos directamente (o kNN foi alvo de ajuste de hiperparâmetros, recorrendo à análise do gráfico acima representado). Podemos observar que ambos os modelos obtêm um valor de AUC semelhante, a rondar os 75%. Observando as "linhas de fronteira" obtidas rapidamente podemos observar a semelhança entre a regressão logistica e o kNN, o que seria de esperar dado o valor elevado do k, bem como o valor quase idêntico do AUC. É importante também referir, que dado o dataset escolhido o qual contêm uma boa separação das classes, um modelo como a regressão logistica (pouco flexível), consegue apresentar um resultado quase idêntico ao modelo utilizado mais flexível, o kNN. Isto é consequência de o valor óptimo de k ser bastante elevado, o que leva a uma fronteira de decisão bastante linear. Relativamente à QDA, a fronteira obtida não se assemelha a uma linearização, mas sim a uma quadratização da linha. Não é um modelo tão flexível como o kNN, mas em alguns casos é possível que tenha um melhor comportamento devido à abordagem quadrática.

1.4 Question 2

Separation in train data and test data and data normalization

[17]:
```
X = data_yeast.loc[:, ['mcg', 'gvh', 'alm', 'mit', 'erl', 'pox', 'vac', 'nuc']].
 ↪values
Y = data_yeast.loc[:,'class'].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 1, stratify = Y, shuffle=True)

minmax_scaler = MinMaxScaler()
X_train = minmax_scaler.fit_transform(X_train)
X_test = minmax_scaler.transform(X_test)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

[18]:
```
def StratKFW(model, splits, X_train, Y_train, X_test, Y_test):
    Y_pred_trainLogReg = model.predict(X_train)
    Y_pred_testLogReg = model.predict(X_test)
    print('Weighted F1 score on training set: ',round(f1_score(Y_train,␣
 ↪Y_pred_trainLogReg, average='weighted'),4),
          '\nWeighted F1 score on test set: ',round(f1_score(Y_test,␣
 ↪Y_pred_testLogReg, average='weighted'), 4))
    f1score = []
    accscore = []
    for train_index, test_index in SKF.split (X_test, Y_test):
        X_test1 = X_test[test_index]
        Y_test1 = Y_test[test_index]
        f1score.append(f1_score(Y_test1, model.predict(X_test1),␣
 ↪average='weighted'))
        accscore.append(accuracy_score(Y_test1, model.predict(X_test1)))
    print(f'Weighted F1 score on {splits}-fold test data: ',round(np.
 ↪mean(f1score),4),'+/-', round(np.std(f1score),4))
```

```
    print('\nClassification report:\n',classification_report(Y_test,␣
 ↪Y_pred_testLogReg, digits=3))
    return f1score, accscore
```

1.4.1 Logistic regression

[19]:
```
modelLR = LogReg()
modelLR.fit(X_train, Y_train)
f1score_LogReg, accscore_LogReg = [], []
f1score_LogReg, accscore_LogReg = StratKFW(modelLR, 5, X_train, Y_train,␣
 ↪X_test, Y_test)
```

```
Weighted F1 score on training set:  0.5564
Weighted F1 score on test set:  0.5435
Weighted F1 score on 5-fold test data:  0.5409 +/- 0.027

Classification report:
               precision    recall  f1-score   support

          CYT      0.507     0.753     0.606        93
          ERL      0.000     0.000     0.000         1
          EXC      0.000     0.000     0.000         7
          ME1      0.455     0.556     0.500         9
          ME2      0.000     0.000     0.000        10
          ME3      0.667     0.688     0.677        32
          MIT      0.617     0.592     0.604        49
          NUC      0.641     0.477     0.547        86
          POX      0.667     0.500     0.571         4
          VAC      0.000     0.000     0.000         6

     accuracy                          0.569       297
    macro avg      0.355     0.356     0.351       297
 weighted avg      0.541     0.569     0.544       297
```

1.4.2 kNN

[20]:
```
k_best = valid_curve(modelkNN, X_train, Y_train, 6, 'accuracy')
txt="Figura 6. Curva de validação para determinação do melhor k para o modelo␣
 ↪kNN para o dataset 'yeast'."
plt.figtext(0.5, -0.05, txt, wrap=True, horizontalalignment='center',␣
 ↪fontsize=12)
plt.show();
```

```
Best K is 17
```

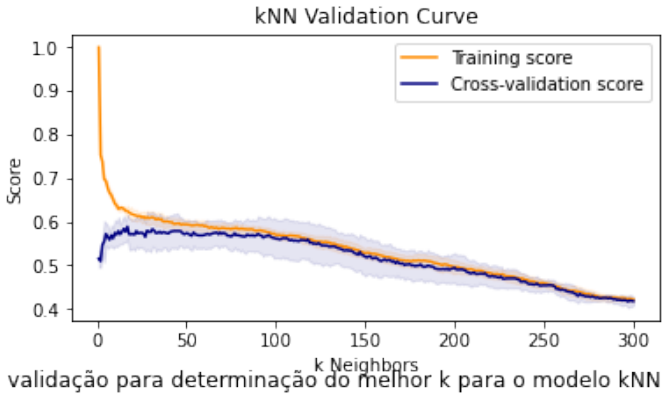

Figura 6. Curva de validação para determinação do melhor k para o modelo kNN para o dataset 'yeast'.

[21]:
```
modelknn = kNN(n_neighbors=k_best)
modelknn.fit(X_train, Y_train)
f1score_knn, accscore_knn = [], []
f1score_knn, accscore_knn = StratKFW(modelknn, 5, X_train, Y_train, X_test,␣
↪Y_test)
```

```
Weighted F1 score on training set:  0.6207
Weighted F1 score on test set:  0.5693
Weighted F1 score on 5-fold test data:  0.5664 +/- 0.0411

Classification report:
              precision    recall  f1-score   support

         CYT      0.530     0.656     0.587        93
         ERL      0.000     0.000     0.000         1
         EXC      0.667     0.571     0.615         7
         ME1      0.462     0.667     0.545         9
         ME2      0.333     0.200     0.250        10
         ME3      0.719     0.719     0.719        32
         MIT      0.681     0.653     0.667        49
         NUC      0.560     0.488     0.522        86
         POX      0.667     0.500     0.571         4
         VAC      0.000     0.000     0.000         6

    accuracy                          0.579       297
   macro avg      0.462     0.445     0.448       297
weighted avg      0.568     0.579     0.569       297
```

1.4.3 Decision tree

[22]:
```
modeltree = DTreeClass()
modeltree.fit(X_train, Y_train);
```

[23]:
```
def model_grid_search(model, param_grid, cv, scoring, n_jobs): # "cv - integer,␣
↪to specify the number of folds in a `(Stratified) KFold`,"
    model_grid = GridSearchCV(estimator = model, param_grid = param_grid, cv =␣
↪cv, refit = True, scoring= scoring, n_jobs = n_jobs)
    model_grid.fit(X_train, Y_train)
    print(f"Best estimator: {model_grid.best_estimator_} \n  Best score:␣
↪{model_grid.best_score_} \n Best Params: {model_grid.best_params_}")
    return model_grid.best_estimator_
```

[24]:
```
max_depth = [None] + [x for x in np.arange(1,20,4)]
min_sample_split= np.arange(2, 10,2)
min_sample_leaf = np.arange(1,5)
ccp_alpha = np.arange(0.01,100, 10)
param_grid_tree = {"criterion": ['gini', "entropy"],\
              "splitter": ['best',"random"],\
              "max_depth": max_depth,\
              "min_samples_split": min_sample_split,\
              "min_samples_leaf": min_sample_leaf, \
              "ccp_alpha": ccp_alpha}
```

[25]:
```
tree_grid = model_grid_search(modeltree, param_grid_tree, 5, "f1_weighted", 2)
```

```
Best estimator: DecisionTreeClassifier(ccp_alpha=0.01, criterion='entropy',
max_depth=5,
                       min_samples_leaf=4)
  Best score: 0.5669393207369835
 Best Params: {'ccp_alpha': 0.01, 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'best'}
```

[26]:
```
modeltree = tree_grid
f1score_tree, accscore_tree = StratKFW(modelLR, 5, X_train, Y_train, X_test,␣
↪Y_test)
```

```
Weighted F1 score on training set:  0.5564
Weighted F1 score on test set:  0.5435
Weighted F1 score on 5-fold test data:  0.5409 +/- 0.027

Classification report:
              precision    recall  f1-score   support

         CYT      0.507     0.753     0.606        93
         ERL      0.000     0.000     0.000         1
         EXC      0.000     0.000     0.000         7
```

```
          ME1       0.455     0.556     0.500         9
          ME2       0.000     0.000     0.000        10
          ME3       0.667     0.688     0.677        32
          MIT       0.617     0.592     0.604        49
          NUC       0.641     0.477     0.547        86
          POX       0.667     0.500     0.571         4
          VAC       0.000     0.000     0.000         6

     accuracy                          0.569       297
    macro avg       0.355     0.356     0.351       297
 weighted avg       0.541     0.569     0.544       297
```

1.4.4 Achieved results comparison

[27]:
```
stats_f1, pvalue_f1 = stats.f_oneway(f1score_LogReg, f1score_knn, f1score_tree)
print('p-value =',round(pvalue_f1,3), ': Não há diferenças estatisticas␣
 ↪significativas entre as médias de "weighted F1-score" para os três modelos.')
stats_acc, pvalue_acc = stats.f_oneway(accscore_LogReg, accscore_knn,␣
 ↪accscore_tree)
print('p-value =',round(pvalue_acc,3), ': Não há diferenças estatisticas␣
 ↪significativas entre as médias de "accuracy" para os três modelos.')
```

```
p-value = 0.461 : Não há diferenças estatisticas significativas entre as médias
de "weighted F1-score" para os três modelos.
p-value = 0.888 : Não há diferenças estatisticas significativas entre as médias
de "accuracy" para os três modelos.
```

[28]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 3), sharey=True)
#sns.set_theme(style="whitegrid")
ax[0].set_title('Models F1 Score', fontsize = 16)
ax[1].set_title('Models Accuracy Score', fontsize = 16)
sns.stripplot(y=f1score_LogReg + f1score_knn + f1score_tree,␣
 ↪x=['LogReg']*5+['kNN']*5+['Tree']*5, ax=ax[0])
sns.boxplot(y=f1score_LogReg + f1score_knn + f1score_tree,␣
 ↪x=['LogReg']*5+['kNN']*5+['Tree']*5, saturation=0.3, ax=ax[0])
sns.stripplot(y=accscore_LogReg + accscore_knn + accscore_tree,␣
 ↪x=['LogReg']*5+['kNN']*5+['Tree']*5, ax=ax[1])
sns.boxplot(y=accscore_LogReg + accscore_knn + accscore_tree,␣
 ↪x=['LogReg']*5+['kNN']*5+['Tree']*5, saturation=0.3, ax=ax[1]);
txt="Figura 7. Gráficos com os valores de 'F1 score' e 'accuracy' para os três␣
 ↪modelos testados."
plt.figtext(0.5, -0.05, txt, wrap=True, horizontalalignment='center',␣
 ↪fontsize=12)
plt.show();
```

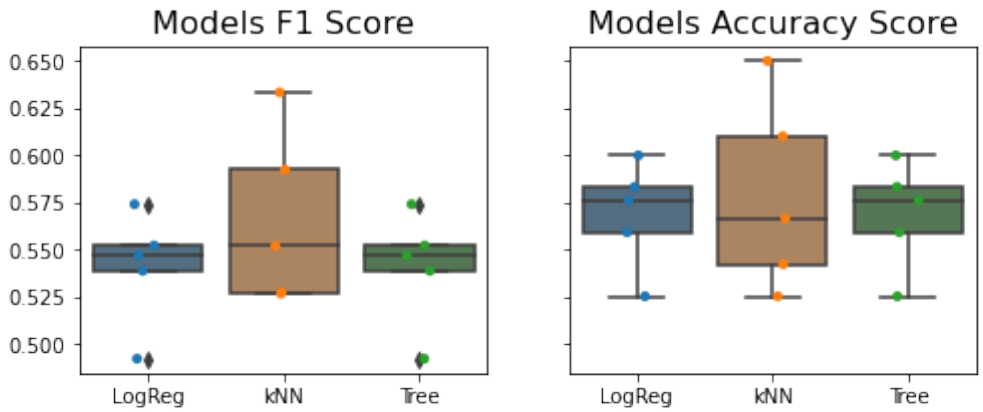

Figura 7. Gráficos com os valores de 'F1 score' e 'accuracy' para os três modelos testados.

Nesta questão foi nos proposto comparar como 3 modelos se comportavam com um dataset com várias classes alvo com uma representatividade reduzida (small classes). Um dos maiores problemas devido ao problema anteriormente referido foi os modelos não serem capazes de detectar algumas dessas classes. No modelo kNN foi realizada uma optimização do valor de k, e nas árvores de decisão, um grid search. No entanto, nenhum dos modelos conseguiu destacar-se dos restantes, obtendo-se resultados similares.

1.5 Conclusão

Dos resultados apresentados é possível concluir que os diversos modelos utilizados tem comportamentos distintos. No entanto, mesmo alguns sendo alvo de optimizações, verifica-se que a sua performance está muito dependente dos dados fornecidos. Podiam ter sido abordados outros modelos de diferentes características e respectivas suposições mais adequados aos dados fornecidos, tais como as random forest que tiram partido do bagging, o qual provavelmente detectaria também as classes menos representadas. Para além disso, métodos de sampling, que permitem balancear as classes presentes durante o treino iriam permitir um melhor treino dos modelos e, consequentemente, melhorar a performance destes; é o caso de oversampling, que permitiria uma melhor representatividade de todas classes através da duplicação das classes minoritárias.

1.6 Referências

- Towards Data Science
- The Elementals of Statistical Learning
- Pattern Recognition and Machine Learning