Aprendizado de Máquina – Exercício 3

Thiago Amendola – 148062

Introdução

Foi feita a leitura de 1566 linhas de dados com 590 dimensões diferentes. Na etapa de pré processamento, os dados inválidos são substituídos pela média dos dados válidos daquela dimensão e são transformados tal que a média e desvio padrão de cada dimensão sejam, respectivamente, 0 e 1.

Preparados os dados, esta aplicação avalia a execução de 5 algoritmos distintos para a predição de respostas. Estes 5 algoritmos são: K Nearest Neighbors, SVM com kernel RBF, Rede Neural, Random Forest e Gradient Boosting Classifier. Para testar a acurácia de cada um destes algoritmos, é feito um 5 Fold externo no conjunto de dados, sendo a acurácia final a média das 5 acurácias geradas pelos folds. Dentro de cada Fold, é feito um 3 Fold interno com Grid Search, para encontrar a melhor configuração de hiperparâmetros que resolvem aquele conjunto de testes. O melhor conjunto de hiperparâmetros é utilizado para gerar a melhor acurácia do fold externo em execução.

Resolução

Começamos importando as bibliotecas que serão utilizadas nesta aplicação:

```
import numpy as np
import pandas as pd
import math
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
```

É então feita a leitura dos dados dos arquivos fornecidos:

```
raw_data = pd.read_csv('secom.data', ' ')
raw_labels = pd.read_csv('secom_labels.data', ' ')
datas = raw_data.values
labels = raw_labels.values[:,0]
```

Após a leitura, inicia-se a etapa de pré processamento dos dados. Inicia-se uma iteração pelas colunas da tabela de dados, representando as dimensões, visto que as operações de pré processamento são feitas em cada dimensão. Inicialmente, é computada a média dos valores válidos da dimensão, que substitui os valores inválidos da dimensão. Em seguida, os dados desta dimensão são normalizados, sofrendo transformações de modo

que a média da dimensão se torne 0 e o desvio padrão se torne 1:

```
    for col in range(datas.shape[1]):
        column = datas[:,col]
        #- Inputation by mean
        sumMean = 0
        divMean = 0
        for e in column:
            if (math.isnan(e)==False):
                sumMean += e
                divMean += 1
        sumMean /= divMean # has media
        #print(sumMean)
        for i in range(len(column)):
            if math.isnan(column[i]):
                column[i] = sumMean

        #- Mean and Standard Deviation
        var=0
        for i in range(len(column)):
            #- Mean equals 0
            column[i]-=sumMean
            #- Variance
            var += pow(column[i],2)

        var /= len(column)
        sd = math.sqrt(var) # is Standard Deviation

        #- Normalization
        for e in column:
            e = e/sd
```

A partir daqui, são aplicados os 5 algoritmos, que irão retornar suas respectivar acurácias:

```
    kNearNeighborsClsf(datas, labels)
    SVMkernelRBF (datas, labels)
    neuralNetwork (datas, labels)
    randomForest (datas, labels)
    gradientBoostingClassifier (datas, labels)
```

Em cada algoritmo, é feito um 5 Fold externo para o conjunto de dados. Dentro do fold externo, é aplicado um 3 Fold interno seguido de Grid Search, com o intuito de encontrar os melhores valores para os hiperparâmetros do algoritmo em questão. Dentro dos folds internos, executa-se o algoritmo de classificação usando os hiperparâmetros da iteração do Grid Search e é captada sua respectiva acurácia. Obtidas as 3 acurácias do 3 Fold interno, seleciona-se os valores de hiperparâmetros que garantiram a melhor acurácia e aplica-se o algoritmo de classificação com eles, gerando a acurácia do fold externo. Ao final do 5 Fold externo, é calculada a média dos 5 valores de acurácia obtidos

anteriormente, configurando a acurácia final do algoritmo.

Seguem os códigos para os algoritmos utilizados:

K Nearest Neighbors

Neste algoritmo, é calculado o PCA do conjunto de dados que mantém 80% da variância. Este algoritmo assume o valor K como hiperparâmetro, variando em [1, 5, 11, 15, 21, 25].

```
def kNearNeighborsClsf(datas, labels):
    pca = PCA()
    pca.fit(datas)
    var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4))
    x=0
    while (var[x] < 0.8):
        x+=1
    x+=1
    pca = PCA(n_components=x)
    datas_t = pca.fit_transform(datas)
    #Run kNN
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas_t,labels):
            d_tr, d_te = datas_t[ext_tr], datas_t[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_k = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for k in knn_neighbors:
                    knn = KNeighborsClassifier(n_neighbors=k)
                    knn.fit(tr, ltr.astype(int))
                    #Testing
                    pred = knn.predict(te)
                    ac = pred_accuracy(pred, lte)
                    #Check if result's the best
                    if ac > int_ac:
                        int_ac, int_k = ac, k
            #Training
            knn = KNeighborsClassifier(n_neighbors=int_k)
            knn.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = knn.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            aver_accur += ac
```

```
            if ac>best_accur:
                best_accur  = ac

    aver_accur /= 5
    print("Average accuracy="+str(aver_accur))
```

SVM com kernel RBF

Neste algoritmo, assumem-se como hiperparâmetros C e gamma, dentro dos valores [2**-5, 2**0, 2**5, 2**10] e [2**-15, 2**-10, 2**-5, 2**0, 2**5], respectivamente.

```
def SVMkernelRBF (datas, labels):
    aver_accur = 0
    best_accur = 0
    best_C = 1
    best_gamma = 1
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
        d_tr, d_te = datas[ext_tr], datas[ext_te]
        l_tr, l_te = labels[ext_tr], labels[ext_te]

        int_ac = 0
        int_C = 1
        int_gamma = 1

        int_skf = KFold(3)
        for int_tr, int_te in int_skf.split(d_tr,l_tr):
            tr, te = d_tr[int_tr], d_tr[int_te]
            ltr, lte = l_tr[int_tr], l_tr[int_te]

            for C in c_values:
                for gamma in gamma_values:
                    #Training
                    clf = SVC(C=C, kernel='rbf', gamma=gamma)
                    clf.fit(tr, ltr.astype(int))
                    #Testing
                    pred = clf.predict(te)
                    ac = pred_accuracy(pred, lte)
                    #Check if result's the best
                    if ac > int_ac:
                        int_ac, int_C, int_gamma = ac, C, gamma
        #Training
        clf = SVC(C=int_C, kernel='rbf', gamma=int_gamma)
        clf.fit(d_tr, l_tr.astype(int))
        #Testing
        pred = clf.predict(d_te)
        ac = pred_accuracy(pred, l_te)
        aver_accur += ac
        if ac>best_accur:
            best_accur, best_C, best_gamma = ac, int_C, int_gamma
    aver_accur /= 5
```

```
    print("Average accuracy="+str(aver_accur))
```

Redes Neurais

Neste algoritmo, o número de neurônios da camada escondida foi o hiperparâmetro adotado, variando em [10, 20, 30, 40].

```
def neuralNetwork(datas, labels):
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_hl = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for hl in hidden_layers:
                    nn = MLPClassifier(hidden_layer_sizes=hl)
                    nn.fit(tr, ltr.astype(int))
                    #Testing
                    pred = nn.predict(te)
                    ac = pred_accuracy(pred, lte)
                    #Check if result's the best
                    if ac > int_ac:
                        int_ac, int_hl = ac, hl
            #Training
            nn = MLPClassifier(hidden_layer_sizes=int_hl)
            nn.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = nn.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac
    aver_accur /= 5
    print("Average accuracy="+str(aver_accur))
```

Random Forest

Neste algoritmo, assumem-se como hiperparâmetros o número de features e o número de árvores, dentro dos valores [10, 15, 20, 25] e [100, 200, 300, 400], respectivamente.

```
def randomForest(datas, labels):
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_features = 0
            int_trees = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]
                for f in n_features:
                    for t in n_trees:
                        rf = RandomForestClassifier(n_estimators=t, crite
                        rf.fit(tr, ltr.astype(int))
                        #Testing
                        pred = rf.predict(te)
                        ac = pred_accuracy(pred, lte)
                        #Check if result's the best
                        if ac > int_ac:
                            int_ac, int_trees, int_features = ac, t, f
            #Training
            rf = RandomForestClassifier(n_estimators=int_trees, criterion=
            rf.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = rf.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac
    aver_accur /= 5
    print("Average accuracy="+str(aver_accur))
```

Gradient Boosting Machine

Neste algoritmo, assumem-se como hiperparâmetros o número de árvores e a taca de aprendizado do algoritmo, dentro dos valores [30, 70, 100] e [0.1, 0.05], respectivamente. Assume-se também como profundidade máxima da árvore igual a 5:

```
def gradientBoostingClassifier(datas, labels):
    tree_breadth = 5
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
```

```
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_lr = 0
            int_trees = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for lr in gbm_lr:
                    for t in gbm_trees:
                        gbm = GradientBoostingClassifier(loss='deviance',
                        gbm.fit(tr, ltr.astype(int))
                        #Testing
                        pred = gbm.predict(te)
                        ac = pred_accuracy(pred, lte)
                        #Check if result's the best
                        if ac > int_ac:
                            int_ac, int_trees, int_lr = ac, t, lr
            #Training
            gbm = GradientBoostingClassifier(loss='deviance',learning_rat
            gbm.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = gbm.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac
    aver_accur /= 5
    print("Average accuracy="+str(aver_accur))
```

Observação

Os algoritmos acima utilizam uma função de cálculo de acurácia denominada `pred_accuracy`, que calcula a acurácia de uma predição dados os valores preditos e os valores reais:

```
def pred_accuracy(pred, lte):
    hits = [None] * len(pred)
    for i in range(0,len(pred)):
        if pred[i] == lte[i]:
            hits[i] = 1
        else:
            hits[i] = 0
    return sum(hits)/len(pred)
```

Resultados

Foram obtidos os seguintes resultados:

Algoritmo	Acurácia
K Nearest Neighbors	0.9336358641460288
SVM com kernel RBF	0.9336358641460288
Redes Neurais	0.8576667141490812
Random Forest	0.9336358641460288
Gradient Boosting Machine	0.8985734926029181

Como é possível ver, os algoritmos de K Nearest Neighbors, SVM com kernel RBF e Random Forest obtiverama mesma acurácia de aproximadamente 93%. A pior acurácia provém do algoritmo de Redes Neurais, com acurácia aproximada de 86%

Código completo

```
import numpy as np
import pandas as pd
import math
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier

c_values = [2**-5, 2**0, 2**5, 2**10]
gamma_values = [2**-15, 2**-10, 2**-5, 2**0, 2**5]
hidden_layers = [10, 20, 30, 40]
n_features = [10, 15, 20, 25]
n_trees = [100, 200, 300, 400]
gbm_trees = [30,70,100]
gbm_lr = [0.1, 0.05]
knn_neighbors = [1, 5, 11, 15, 21, 25]

def pred_accuracy(pred, lte):
    hits = [None] * len(pred)
    for i in range(0,len(pred)):
        if pred[i] == lte[i]:
            hits[i] = 1
        else:
            hits[i] = 0
    return sum(hits)/len(pred)

def kNearNeighborsClsf(datas, labels):
    print(" => Starting K Nearest Neighbors Classifier with 5x3-Fold and
    #Apply PCA on datas, maintaining 80% of variance
    print("- Applying PCA on data, maintaining 80% of variance")
```

```
pca = PCA()
pca.fit(datas)
var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4))
x=0
while (var[x] < 0.8):
    x+=1
x+=1
pca = PCA(n_components=x)
datas_t = pca.fit_transform(datas)
#Run kNN
aver_accur = 0
best_accur = 0
ext_skf = KFold(5)
for ext_tr, ext_te in ext_skf.split(datas_t,labels):
        d_tr, d_te = datas_t[ext_tr], datas_t[ext_te]
        l_tr, l_te = labels[ext_tr], labels[ext_te]

        int_ac = 0
        int_k = 0

        int_skf = KFold(3)
        for int_tr, int_te in int_skf.split(d_tr,l_tr):
            tr, te = d_tr[int_tr], d_tr[int_te]
            ltr, lte = l_tr[int_tr], l_tr[int_te]

            for k in knn_neighbors:
                knn = KNeighborsClassifier(n_neighbors=k)
                knn.fit(tr, ltr.astype(int))
                #Testing
                pred = knn.predict(te)
                ac = pred_accuracy(pred, lte)
                #Check if result's the best
                if ac > int_ac:
                    int_ac, int_k = ac, k

        #Training
        knn = KNeighborsClassifier(n_neighbors=int_k)
        knn.fit(d_tr, l_tr.astype(int))
        #Testing
        pred = knn.predict(d_te)
        ac = pred_accuracy(pred, l_te)
        print("Accuracy="+str(ac)+"; K="+str(int_k))
        aver_accur += ac
        if ac>best_accur:
            best_accur  = ac

aver_accur /= 5
print("========================")
print("Average accuracy="+str(aver_accur))
print("========================")
print("")
```

```
def SVMkernelRBF (datas, labels):
    print(" => Starting SVM using RBF kernel with 5x3-Fold and C and Gamm
    aver_accur = 0
    best_accur = 0
    best_C = 1
    best_gamma = 1
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
        d_tr, d_te = datas[ext_tr], datas[ext_te]
        l_tr, l_te = labels[ext_tr], labels[ext_te]

        int_ac = 0
        int_C = 1
        int_gamma = 1

        int_skf = KFold(3)
        for int_tr, int_te in int_skf.split(d_tr,l_tr):
            tr, te = d_tr[int_tr], d_tr[int_te]
            ltr, lte = l_tr[int_tr], l_tr[int_te]

            for C in c_values:
                for gamma in gamma_values:
                    #Training
                    clf = SVC(C=C, kernel='rbf', gamma=gamma)
                    clf.fit(tr, ltr.astype(int))
                    #Testing
                    pred = clf.predict(te)
                    ac = pred_accuracy(pred, lte)
                    #Check if result's the best
                    if ac > int_ac:
                        int_ac, int_C, int_gamma = ac, C, gamma

        #Training
        clf = SVC(C=int_C, kernel='rbf', gamma=int_gamma)
        clf.fit(d_tr, l_tr.astype(int))
        #Testing
        pred = clf.predict(d_te)
        ac = pred_accuracy(pred, l_te)
        print("Accuracy="+str(ac)+"; C="+str(int_C)+"; gamma="+str(int_ga
        aver_accur += ac
        if ac>best_accur:
            best_accur, best_C, best_gamma = ac, int_C, int_gamma

    aver_accur /= 5
    print("========================")
    print("Average accuracy="+str(aver_accur))
    print("========================")
    print("")
```

```
def neuralNetwork(datas, labels):
    print(" => Starting Neural Networks with 5x3-Fold and Hidden layers' n
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_hl = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for hl in hidden_layers:
                    nn = MLPClassifier(hidden_layer_sizes=hl)
                    nn.fit(tr, ltr.astype(int))
                    #Testing
                    pred = nn.predict(te)
                    ac = pred_accuracy(pred, lte)
                    #Check if result's the best
                    if ac > int_ac:
                        int_ac, int_hl = ac, hl

            #Training
            nn = MLPClassifier(hidden_layer_sizes=int_hl)
            nn.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = nn.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            print("Accuracy="+str(ac)+"; hidden layers="+str(int_hl))
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac

    aver_accur /= 5
    print("=========================")
    print("Average accuracy="+str(aver_accur))
    print("=========================")
    print("")


def randomForest(datas, labels):
    print(" => Starting Random Forest with 5x3-Fold and nFeatures and nTr
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
```

```
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]

            int_ac = 0
            int_features = 0
            int_trees = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for f in n_features:
                    for t in n_trees:
                        rf = RandomForestClassifier(n_estimators=t, crite
                        rf.fit(tr, ltr.astype(int))
                        #Testing
                        pred = rf.predict(te)
                        ac = pred_accuracy(pred, lte)
                        #Check if result's the best
                        if ac > int_ac:
                            int_ac, int_trees, int_features = ac, t, f

            #Training
            rf = RandomForestClassifier(n_estimators=int_trees, criterion=
            rf.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = rf.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            print("Accuracy="+str(ac)+"; nFeatures="+str(int_features)+";
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac

    aver_accur /= 5
    print("=========================")
    print("Average accuracy="+str(aver_accur))
    print("=========================")
    print("")


def gradientBoostingClassifier(datas, labels):
    tree_breadth = 5
    print(" => Starting Gradient Boosting Classifier with 5x3-Fold and nT
    aver_accur = 0
    best_accur = 0
    ext_skf = KFold(5)
    for ext_tr, ext_te in ext_skf.split(datas,labels):
            d_tr, d_te = datas[ext_tr], datas[ext_te]
            l_tr, l_te = labels[ext_tr], labels[ext_te]
```

```
            int_ac = 0
            int_lr = 0
            int_trees = 0

            int_skf = KFold(3)
            for int_tr, int_te in int_skf.split(d_tr,l_tr):
                tr, te = d_tr[int_tr], d_tr[int_te]
                ltr, lte = l_tr[int_tr], l_tr[int_te]

                for lr in gbm_lr:
                    for t in gbm_trees:
                        gbm = GradientBoostingClassifier(loss='deviance',
                        gbm.fit(tr, ltr.astype(int))
                        #Testing
                        pred = gbm.predict(te)
                        ac = pred_accuracy(pred, lte)
                        #Check if result's the best
                        if ac > int_ac:
                            int_ac, int_trees, int_lr = ac, t, lr

            #Training
            gbm = GradientBoostingClassifier(loss='deviance',learning_rat
            gbm.fit(d_tr, l_tr.astype(int))
            #Testing
            pred = gbm.predict(d_te)
            ac = pred_accuracy(pred, l_te)
            print("Accuracy="+str(ac)+"; Number of trees="+str(int_trees)
            aver_accur += ac
            if ac>best_accur:
                best_accur  = ac

    aver_accur /= 5
    print("=========================")
    print("Average accuracy="+str(aver_accur))
    print("=========================")
    print("")



if __name__=='__main__':

    print(" => Reading data files...")

    raw_data = pd.read_csv('secom.data', ' ')
    raw_labels = pd.read_csv('secom_labels.data', ' ')
    datas = raw_data.values
    labels = raw_labels.values[:,0]

    print(datas.shape)
```

```
print(" => Preprocessing data...")
#Preprocess data
for col in range(datas.shape[1]):
    column = datas[:,col]
    #- Inputation by mean
    sumMean = 0
    divMean = 0
    for e in column:
        if (math.isnan(e)==False):
            sumMean += e
            divMean += 1
    sumMean /= divMean # has media
    for i in range(len(column)):
        if math.isnan(column[i]):
            column[i] = sumMean

    #- Mean and Standard Deviation
    var=0
    for i in range(len(column)):
        #- Mean equals 0
        column[i]-=sumMean
        #- Variance
        var += pow(column[i],2)
    var /= len(column)
    sd = math.sqrt(var) # is Standard Deviation

    #- Normalization
    for e in column:
        e = e/sd

#- Apply predition methods
kNearNeighborsClsf(datas, labels)
SVMkernelRBF (datas, labels)
neuralNetwork (datas, labels)
randomForest (datas, labels)
gradientBoostingClassifier (datas, labels)
```