FMKL

主要数据集：UCI Machine Learning Repository: Dorothea Data Set；

这是一个二分类问题。这个仓库已经有PCA处理过的特征矩阵与标签的csv，可以直接使用（PCA*800.csv与 *Labels.csv）。

由于数据集未公布测试标签，因而将验证集作为测试集，共有800+350=1150条数据。

可选batch size=115（80：35）

epoch=100

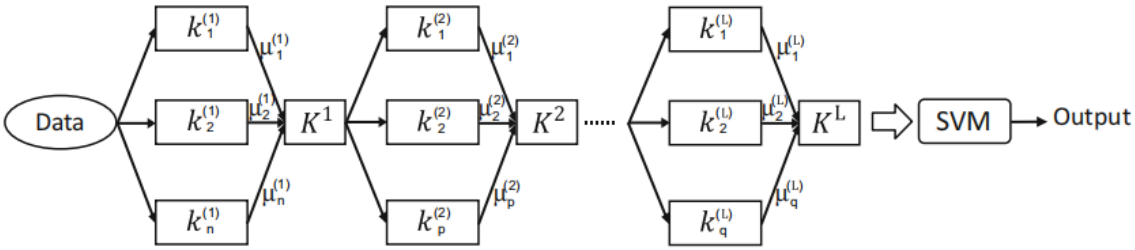
目标：

- 使用(dpmkl)相关函数改造trainFSVC与testFSVC,实现一个多层的多核学习网络模型；
- 支持自定义层数（最大3层）与外部自定义核函数接口，损失函数为交叉熵；

  ** 可以先实现1层rbf核，2层线性核的总计2层结构；
- 每次训练除了计算accuracy之外，也计算precision，recall，f1_score，roc_auc并打印；
- 每次训练检查损失变化，如果多个epoch后小于一个值，则立即终止程序；
- 以上程序使用matlab实现。

DMKL算法

代码仓库（Matlab）：ericstrobl/deepMKL: Deep Multiple Kernel Learning by Span Bound (github.com)



输入： 训练集D = {(xi, yi)|i = 1, ..., n}, $μ^1_k$为每个内核k的权重系数，层数为L，η ∈ [0..1]
输出： 学习到的权重向量μ，以及SVM系数向量α。

for i ← 1 to maxIterations do

- 使用μ1计算核K(L)；
- 用结果核K(L)求解SVM问题；
- 计算∇E；
- 更新权重向量：μi+1 = μi + η∇E；
- 如果达到停止标准，那么断开；

Algorithm 1: Multi-Layer Multiple Kernel Learning Algorithm: (μ, α)=ML-MKL(D, μ^1, L, η)

Input: Training set $D = \{(x_i, y_i)|i = 1, ..., n\}$, μ_k^1 for every kernel k, number of layers L, and $\eta \in [0..1]$

Output: The learned weight vector μ, and SVM coefficient vector α.

```
for i ← 1 to maxIterations do
    Compute the kernel K^(L) using μ^1;
    Solve the SVM problem with the resultant kernel K^(L);
    Compute ∇E (Eq. 16) by using the equation 18;
    Update the vector of weights: μ^{i+1} = μ^i + η∇E;
    if stopping criterion then
        Break;
```

Fuzzy SVC

fuzzy: 样本(***x***,*y*)→ (***x***,*y*,s), s_i = S(***x_i***) ∈ (0,1)

使用基于类中心均值的S(x)：1 - (欧几里得距离 / (组半径 + delta))（代码实现于：computeFuzzynumber.m）

trainFSVC.m

```
% /*--------------Min (1/2)*lamdai*lamdaj*yi*yj*K(xi,xj)-sum(lamdai)-------------
---------*/
% /*--------------subject to:-----------------------------------------------------
---------*/
% /*--------------sum(lamdai*yi)=0------------------------------------------------
---------*/
% /*--------------0 =< lamdai <= si*C---------------------------------------------
---------*/
% /*-----------------------------------------------------------------------------
---------*/

%---------------------------trainFSVC-------------------------------*/
function [lamda,boundary] = trainFSVC(train,fms,C,ker,para)

    %construct objective function
    [nrow,ncol] = size(train);

    %find X and Y
    X = train(:,1:ncol-1);
    Y = train(:,ncol);

    %objective function with linear part
```

```
    f=(-1)*ones(nrow,1);

    %define kernel matrix
    Kmatrix = zeros(nrow,nrow);
    Kmatrix = kernel(X,X,ker,para);

    a0 = zeros(nrow,1)+0.0001;

    %construct Hessian matrix
    H = (Y*Y').*Kmatrix;

    %Set LHS of constrains
    Aeq=Y';

    %Set RHS of constrains
    beq=0;

    %set boundary of variables
    lb=zeros(nrow,1);
    ub=C*fms;

    %Solve the optimal problem
    options = optimset;
    options.LargeScale = 'on';
    options.Display = 'on';
    [x,fval,exitflag,output,lambda] = quadprog(H,f,[],
[],Aeq,beq,lb,ub,a0,options);

    %output the results of classification
    lamda = x;

    %find the boundary of decision hyperplane
    epsilon = 1e-8;
    i_sv = find(abs(lamda)>epsilon);

    %[nrow,ncol] = size(train);

    %find X and Y
    %X = train(:,1:ncol-1);
    %Y = train(:,ncol);

    tmp = kernel(X,X(i_sv,:),ker,para)*(lamda.*Y);
    b = 1./Y(i_sv)-tmp;
    boundary = mean(b);
```

这段代码实现了基于模糊支持向量分类器（FSVC）的分类模型训练，其中输入参数包括：

- train：训练数据集，包括特征和标签
- fms：模糊隶属度向量，代表每个样本属于哪个类别的不确定性程度
- C：惩罚参数，用于控制模型的复杂度和过拟合程度
- ker：核函数类型，包括线性、多项式和高斯径向基等
- para：核函数参数，如多项式核的次数、高斯径向基核的带宽等

该函数的输出包括：

- lamda：拉格朗日乘子向量，代表每个样本被选为支持向量的程度
- boundary：决策超平面的截距，用于计算新的样本点的类别

该函数的实现过程包括：

- 首先根据输入的训练数据集计算出核矩阵（kernel matrix），其中每个元素代表两个样本之间的相似度
- 然后根据模糊隶属度向量和惩罚参数设置每个样本被选为支持向量的上界和下界
- 接着构造目标函数和约束条件，并使用二次规划方法求解拉格朗日乘子向量
- 最后根据拉格朗日乘子向量和支持向量计算出决策超平面的截距

testFSVC.m:

这是一个 Matlab 函数，用于进行支持向量分类器（FSVC）的测试。输入参数包括 lambda、boundary、train、test、ker 和 para，输出包括 predictedY 和 stat。具体实现如下：

首先，从输入参数 train 中提取特征和标签，分别存储在变量 X 和 Y 中。然后从输入参数 test 中提取特征，存储在变量 Xt 中。接下来，通过 kernel 函数计算训练数据和测试数据之间的核矩阵 Kmatrix。最后，利用训练出的模型和核矩阵 Kmatrix 对测试数据进行分类，得到预测标签 predictedY。

最后，根据预测标签 predictedY 和实际标签 test 计算各种统计指标，例如准确率、精度、召回率、敏感度、特异度等，存储在 stat 中并返回。

注释中还有一段代码被注释掉了，看起来是用于计算 ks 和 gini 指标的，但是没有被调用。

computeFuzzynumber.m:

```
function fms = computeFuzzynumber(trainset,delta)

    if isempty(trainset) == 1
        disp('The input dataset is null!');
        return;
    else
        [row1,col1] = size(trainset);

        group1 = trainset(trainset(:,col1) == -1,:);
        group2 = trainset(trainset(:,col1) == 1,:);

        row_g1 = size(group1,1);
        row_g2 = size(group2,1);

        mean_g1 = mean(group1(:,1:col1-1));
        mean_g2 = mean(group2(:,1:col1-1));

        max_g1 = 0;
        max_g2 = 0;

        for i=1:1:row_g1
            if sqrt(norm(group1(i,1:col1-1) - mean_g1)) >= max_g1
                max_g1 = sqrt(norm(group1(i,1:col1-1) - mean_g1));
            end
        end

        for j=1:1:row_g2
            if sqrt(norm(group2(j,1:col1-1) - mean_g2)) >= max_g2
                max_g2 = sqrt(norm(group2(j,1:col1-1) - mean_g2));
            end
        end

        fms = zeros(row1,1);
```

```
        for i=1:row1
            if trainset(i,col1) == -1
                fms(i,1) = 1 - (sqrt(norm(trainset(i,1:col1-1) -
mean_g1))/(max_g1 + delta));
            end
            if trainset(i,col1) == 1
                fms(i,1) = 1 - (sqrt(norm(trainset(j,1:col1-1) -
mean_g2))/(max_g2 + delta));
            end
        end
    end
```

这个函数的作用是为给定的训练集计算模糊数值，它采用了基于数据分布的方法来计算模糊数值，具体步骤如下：

1. 检查输入的训练集是否为空，如果为空则输出错误信息并返回；
2. 从训练集中分离出两个不同类别的组，根据类别标签将数据分为两组；
3. 计算每组数据的均值向量；
4. 对于每个数据点，计算其到组均值的欧几里得距离，并选择最大距离作为组的半径；
5. 对于每个数据点，根据其所属的类别，计算其模糊数值。

其中，模糊数值的计算方式为：1 - (欧几里得距离 / (组半径 + delta))，其中delta为一个常数，用于控制模糊数值的范围，防止出现负数。函数最终输出一个与输入训练集行数相同的向量，表示每个数据点的模糊数值。

FSVC.m:

```
% [trainset,testset] = partition(iris,41,41);
% fmst = computeFuzzynumber(trainset,0.001);
% cleanset = deleteoutliers(trainset,fmst,0.01);
% [tr1,tr2,tr3,tr4,fms1,fms2,fms3,fms4,vl1,vl2,vl3,vl4] =
createcorssvalidationset(cleanset,4);

% minmax std
iid10ptrn1new = iid10ptrn1(:,[24 85 102 103 106 107 108 111 117]);
[trainset,testset] = partition(iid10ptrn1new,200,200);
iid10ptsn1new = iid10ptsn1(:,[24 85 102 103 106 107 108 111 117]);
testset = iid10ptsn1new;
fmst = computeFuzzynumber(trainset,0.001);
[tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,fms1,fms2,fms3,fms4,fms5,fms6,fms7,fms
8,fms9,fms10,vl1,vl2,vl3,vl4,vl5,vl6,vl7,vl8,vl9,vl10] =
createcorssvalidationset([trainset fmst],10);


iid10ptrn1new = iid10ptrn1(:,[24 79 81 87 105 106 117]);
[trainset,testset] = partition(iid10ptrn1new,200,200);
iid10ptsn1new = iid10ptsn1(:,[24 79 81 87 105 106 117]);
testset = iid10ptsn1new;
fmst = computeFuzzynumber(trainset,0.001);
[tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,fms1,fms2,fms3,fms4,fms5,fms6,fms7,fms
8,fms9,fms10,vl1,vl2,vl3,vl4,vl5,vl6,vl7,vl8,vl9,vl10] =
createcorssvalidationset([trainset fmst],10);


% no minmax std
[trainset,testset] = partition(iid10ptrn1(:,[73 80 117]),150,150);
testset = iid10ptsn1(:,[73 80 117]);
```

```
fmst = computeFuzzynumber(trainset,0.001);
[tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,fms1,fms2,fms3,fms4,fms5,fms6,fms7,fms
8,fms9,fms10,vl1,vl2,vl3,vl4,vl5,vl6,vl7,vl8,vl9,vl10] =
createcorssvalidationset([trainset fmst],10);

[trainset,testset] = partition(iid10ptrn2(:,[3 85 109 117]),150,150);
testset = iid10ptsn2(:,[3 85 109 117]);
fmst = computeFuzzynumber(trainset,0.001);
[tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,fms1,fms2,fms3,fms4,fms5,fms6,fms7,fms
8,fms9,fms10,vl1,vl2,vl3,vl4,vl5,vl6,vl7,vl8,vl9,vl10] =
createcorssvalidationset([trainset fmst],10);
```

这段代码包含了多个步骤，以下是每个步骤的简要描述：

1. 通过 partition 函数将数据集 iris 分为训练集和测试集，并计算训练集的模糊数值（fmst），然后使用 deleteoutliers 函数从训练集中删除异常值，最后使用 createcorssvalidationset 函数生成交叉验证集。
2. 从 iid10ptrn1 数据集中选择一些特定的列作为特征，然后使用 partition 函数将数据集分为训练集和测试集，并计算训练集的模糊数值（fmst），最后使用 createcorssvalidationset 函数生成交叉验证集。
3. 从 iid10ptrn1 数据集中选择另一组特定的列作为特征，然后使用 partition 函数将数据集分为训练集和测试集，并计算训练集的模糊数值（fmst），最后使用 createcorssvalidationset 函数生成交叉验证集。
4. 从 iid10ptrn1 和 iid10ptsn1 数据集中选择特定的列作为特征，然后使用 partition 函数将数据集分为训练集和测试集，并计算训练集的模糊数值（fmst），最后使用 createcorssvalidationset 函数生成交叉验证集。
5. 从 iid10ptrn2 和 iid10ptsn2 数据集中选择特定的列作为特征，然后使用 partition 函数将数据集分为训练集和测试集，并计算训练集的模糊数值（fmst），最后使用 createcorssvalidationset 函数生成交叉验证集。

每个步骤的具体实现可能需要查看函数的定义以及数据集的结构才能深入分析。

```
testset = aus14std_ts;

trainset = ger24std_tr;
testset = ger24std_ts;

trainset = usa66std_tr;
testset = usa66std_ts;

fmst = computeFuzzynumber(trainset,0.001);
[tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,fms1,fms2,fms3,fms4,fms5,fms6,fms7,fms
8,fms9,fms10,vl1,vl2,vl3,vl4,vl5,vl6,vl7,vl8,vl9,vl10] =
createcorssvalidationset([trainset fmst],10);


%-----------------------------linear kernel--------------------------%
C = 100;

outputtable(1,:) =
{'model','totalg','totalb','gtog','gtob','btob','btog','gerro','berror','totaler
ror','accuracy','precision','recall','sensitivity','specificity','Fmeasure','cor
relation','ksscore','giniindex'};

for n=1:10
```

```
    eval(['[linear_tr' num2str(n) ',boundary] = trainFSVC(tr' num2str(n) ',fms'
num2str(n) ',C,''linear'',[]);']);
    eval(['[predicty_linear_tr' num2str(n) ', predict_linear_tr' num2str(n) '] =
testFSVC(linear_tr' num2str(n) ',boundary,tr' num2str(n) ',tr' num2str(n)
',''linear'',[]);']);
    outputtable(2*n,1) = {['predict_linear_tr' num2str(n)]};
    outputtable(2*n,2:19) = num2cell(eval(['predict_linear_tr' num2str(n)
'(:,1:18)']));
    eval(['[predicty_linear_ts' num2str(n) ', predict_linear_ts' num2str(n) '] =
testFSVC(linear_tr' num2str(n) ',boundary,tr' num2str(n) ',testset,''linear'',
[]);']);
    outputtable(2*n+1,1) = {['predict_linear_ts' num2str(n)]};
    outputtable(2*n+1,2:19) = num2cell(eval(['predict_linear_ts' num2str(n)
'(:,1:18)']));
end

xlswrite('D:\FSVCwithMatlab2\results20120126\outputFSVCforiid10pn120120522.xls',
outputtable,'sheet1','A1');

%[ks gini] = createksandgini(score_linear_ts1);```
```

这段代码中，首先将三个数据集 `aus14std_tr`、`aus14std_ts`、`ger24std_tr`、`ger24std_ts`、`usa66std_tr` 和 `usa66std_ts` 分别赋值给 `trainset` 和 `testset`，然后使用 `computeFuzzynumber` 函数计算一个模糊数 `fmst`。接着，使用 `createcorssvalidationset` 函数将 `trainset` 和 `fmst` 拆分为十份，分别赋值给 `tr1` 到 `tr10`、`fms1` 到 `fms10` 和 `vl1` 到 `vl10`。

接下来的代码是使用线性核函数进行十次交叉验证训练和测试，并将结果存储在 `predict_linear_tr1` 到 `predict_linear_tr10` 和 `predict_linear_ts1` 到 `predict_linear_ts10` 中。每次训练和测试结束后，将相关结果记录到 `outputtable` 中，并将 `outputtable` 存储到 Excel 文件中。最后，该代码注释掉了计算 `ks` 和 `gini` 的部分，可能是因为这部分代码没有用处或者被移到了其他地方。

DeepMKL.MainMethods[1]

[1]ericstrobl/deepMKL: Deep Multiple Kernel Learning by Span Bound (github.com)

deepMKL_train.m

这段代码实现了深度多核学习（deepMKL）的训练过程，使用的是交替最小化算法。具体实现如下：

1. 首先读取输入的训练数据和标签，以及模型的参数设置，包括网络的层数、学习率、最大迭代次数和SVM的惩罚参数。
2. 初始化网络的权重参数betas，以及多核学习模型的核函数。
3. 进行交替优化，每次交替包括以下步骤：

   a. 使用当前的权重参数betas，计算多核学习模型的核函数，并使用LIBSVM库训练SVM分类器；

   b. 基于训练好的SVM分类器，计算损失函数的梯度，并根据网络的层数选择不同的梯度计算方式；

   c. 根据梯度和学习率更新网络的权重参数betas，并进行投影到可行域内；

   d. 判断停止条件，如果网络的权重参数没有发生明显变化，或者迭代次数达到了最大值，则停止迭代。
4. 返回训练好的SVM模型以及网络的参数，包括权重参数betas、核函数的参数sig、网络的层数和训练数据的大小。

deepMKL_test.m

该函数实现了基于深度多核学习的测试过程。具体来说，它需要以下输入参数：

- `xA`：训练和测试数据矩阵，其中行是实例，列是特征。
- `y`：测试目标向量，其中行是实例。
- `model`：LIBSVM模型，包括训练好的支持向量机模型和相应的训练数据。
- `net`：深度多核学习网络的参数，包括信号，宽度和层数。

输出包括：

- `pred`：预测结果。
- `acc`：准确率。

具体地，该函数首先通过输入数据计算核矩阵，并根据指定的网络参数计算多个核矩阵。最后，它将最后一个核矩阵用于SVM分类器，以预测测试样本的标签，并返回预测结果和准确率。

DeepMKL.Utilities

normalizeKernel.m

这个函数将一个核矩阵进行归一化，返回一个归一化后的核矩阵。

具体来说，给定一个核矩阵K，对其进行归一化的过程为：

1. 计算核矩阵的对角线元素构成的列向量d，即$d*{i} = K*{ii}$。
2. 将K的每个元素除以$\sqrt{d*{i}d*{j}}$，即$kNorm*{ij} = K*{ij}/\sqrt{d*{i}d*{j}}$。

归一化后的核矩阵$kNorm$满足$kNorm*{ii}=1$，且*$0 \leq kNorm*{ij} \leq 1$。常用于核方法中的样本相似度度量，例如支持向量机（SVM）分类器的核函数。

DetermineSig.m

This function `DetermineSig` computes the median distance between pairs of points in the given data matrix `dot`, which is used as the Gaussian kernel parameter `sigma` in a kernel-based machine learning algorithm.

The function first computes a pairwise distance matrix `Dis` between all instances in the data using the `PairWiseDistance` function (which is not provided in the code snippet). It then extracts the upper triangular part of the distance matrix `Dis` (excluding the diagonal) and stores it in a one-dimensional vector `xDis`. The function then computes the median value of `xDis`, which is returned as the `sig` parameter.

If the median value is NaN, the function sets `sig` to 1. This is a common practice in some cases when the distance matrix has zero values, which can happen when the data points are identical or very close to each other.

SpanBoundDeriv.m

该函数计算了模型的Span Bound导数和Span值。Span Bound是一种正则化方法，它试图使得模型的预测函数在训练样本的局部邻域内尽可能平滑，从而提高其泛化性能。该函数的输入包括训练好的SVM模型、核矩阵Kn和Kd，以及训练数据Ytrain，输出包括Span Bound的导数dTdT和Span值Span。