```
classdef MdlResults
    %MDLRESULTS Shared functions for ML/DL models
    %   Input a trained model and test data for access to shared functions
    %   Functions:
    %       classify
    %       metrics
    %
    %   Authors:
    %       Michael Caiola (Michael.Caiola@fda.hhs.gov)
    %       Meijun Ye (Meijun.Ye@fda.hhs.gov)
    %
    %   Disclaimer: This software and documentation (the "Software") were
    %   developed at the Food and Drug Administration (FDA) by employees of
    %   the Federal Government in the course of their official duties.
    %   Pursuant to Title 17, Section 105 of the United States Code,
    %   this work is not subject to copyright protection and is in the
    %   public domain. Permission is hereby granted, free of charge, to any
    %   person obtaining a copy of the Software, to deal in the Software
    %   without restriction, including without limitation the rights to
    %   use, copy, modify, merge, publish, distribute, sublicense, or sell
    %   copies of the Software or derivatives, and to permit persons to
    %   whom the Software is furnished to do so. FDA assumes no
    %   responsibility whatsoever for use by other parties of the Software,
    %   its source code, documentation or compiled executables, and makes
    %   no guarantees, expressed or implied, about its quality,
    %   reliability, or any other characteristic. Further, use of this code
    %   in no way implies endorsement by the FDA or confers any advantage
    %   in regulatory decisions. Although this software can be
    %   redistributed and/or modified freely, we ask that any derivative
    %   works bear some notice that they are derived from it, and any
    %   modified versions bear some notice that they have been modified.
    %   The Software is not intended to make clinical diagnoses or to be
    %   used in any way to diagnose or treat subjects for whom the EEG is
    %   taken.

    properties
        Mdl
        net
        testData
        testLabels
        YPred
        scores
        predictFcn
    end

    methods
        function obj = MdlResults(Mdl,testData,testLabels)
            %UNTITLED Construct an instance of this class
            %   Detailed explanation goes here
            if endsWith(string(class(Mdl)),"Network")
                obj.net = Mdl;
```

```
            elseif isstruct(Mdl)
                names = string(fieldnames(Mdl));
                obj.Mdl = Mdl.(names(startsWith(names,"Classifi","IgnoreCase",true)));
                obj.predictFcn = Mdl.predictFcn;
            else
                obj.Mdl = Mdl;
            end
            obj.testData = testData;
            obj.testLabels = testLabels;
            obj = makePrediction(obj);
        end

        function accuracy = classify(obj)
            figure;
            accuracy = round(mean(obj.YPred == obj.testLabels(1:length(obj.YPred)))*100);
            confusionchart(obj.testLabels(1:length(obj.YPred)), obj.YPred, "RowSummary",↙
"row-normalized");
            title("Accuracy: " + accuracy' + "%")
        end

        function obj = makePrediction(obj)
            if ~isempty(obj.net)
                a = gpuDevice();
                %obj.testData.MiniBatchSize = 1;
                if contains(a.Name,"NVIDIA")
                    try
                        [obj.YPred,obj.scores] = classify(obj.net,obj.testData,"↙
ExecutionEnvironment",'gpu');
                    catch
                        gpuDevice(1);
                        [obj.YPred,obj.scores] = classify(obj.net,obj.testData,"↙
ExecutionEnvironment",'gpu',MiniBatchSize=16);
                    end
                    if length(obj.YPred) ~= length(obj.scores)
                        [obj.YPred,obj.scores] = classify(obj.net,obj.testData,"↙
ExecutionEnvironment",'gpu',MiniBatchSize=1);
                    end
                else
                    [obj.YPred,obj.scores] = classify(obj.net,obj.testData,"↙
ExecutionEnvironment",'cpu',MiniBatchSize=1);
                    if length(obj.YPred) ~= length(obj.scores)
                        [obj.YPred,obj.scores] = classify(obj.net,obj.testData,"↙
ExecutionEnvironment",'cpu',MiniBatchSize=1);
                    end
                end
            elseif ~isempty(obj.predictFcn)
                [obj.YPred,obj.scores] = obj.predictFcn(obj.testData);
                if isnumeric(obj.YPred(1)) %old format
                    in_0 = obj.YPred==0;
                    in_1 = obj.YPred==1;
                    in_2 = obj.YPred==2;
```

```
                    obj.YPred = string();
                    obj.YPred(in_0) = "HEA";
                    obj.YPred(in_1) = "TBI";
                    obj.YPred(in_2) = "STR";
                    in_0 = obj.testLabels==0;
                    in_1 = obj.testLabels==1;
                    in_2 = obj.testLabels==2;
                    obj.testLabels = string();
                    obj.testLabels(in_0) = "HEA";
                    obj.testLabels(in_1) = "TBI";
                    obj.testLabels(in_2) = "STR";
                    obj.scores = obj.scores(:,[1 3 2]);
                end
            else
                [obj.YPred,obj.scores] = predict(obj.Mdl,obj.testData);
            end
        end

        function rocObj = metrics(obj)
            if isempty(obj.scores)
                error("Run 'classify' before metrics.")
            end
            if isempty(obj.predictFcn)
                figure;
                histogram(max(obj.scores,[],2),.34:.01:.99)
                title("Score Histogram")
                xlabel Scores
                acc=[];
                minacc=[];
                sens=[];
                prec=[];
                F1=[];
                numData = [];
                t=.33:.01:1;
                for i = t
                    score_in = max(obj.scores,[],2)>i;
                    C=confusionmat(obj.testLabels(score_in), obj.YPred(score_in));
                    acc = [acc 100*((C(1,1)+C(2,2)+C(3,3))/(sum(C,'all')))];
                    minacc = [minacc min(100*[C(1,1) C(2,2) C(3,3)]./(sum(C,2)'),[],"↙
includenan")];
                    sens = [sens 100*(mean(diag(C)./sum(C,2)))];
                    prec = [prec 100*(mean(diag(C)./sum(C,1)'))];
                    F1 = [F1 (2*sens(end)*prec(end))/(sens(end)+prec(end))];
                    numData = [numData 100*sum(score_in)/length(obj.scores)];
                end
                E = (2*F1 + acc + minacc + 10*log(numData))/5;
                [~,a] = max(E);
                conf = t(a);
                c = colororder();
                figure;
                plot(t,acc,'LineWidth',2,'Color',c(1,:))
```

```
                hold on
                plot(t,minacc,':','LineWidth',2,'Color',c(1,:))
                %plot(t,sens,'LineWidth',2,'Color',c(2,:))
                %plot(t,prec,'LineWidth',2,'Color',c(3,:))
                plot(t,F1,'LineWidth',2,'Color',c(2,:))
                plot(t,numData,'LineWidth',2,'Color',c(3,:))
                plot([conf conf],[0 100],'k--')
                legend(["Accuracy","Min Acc","F1","Data Amount"],'Location','southwest')
                title("Model Performance")
                xlabel Threshold
                score_in = max(obj.scores,[],2)>conf;
                disp("Data Remaining: " + num2str(sum(score_in)/length(obj.scores)))
                accuracy = round((sum(obj.YPred(score_in) == obj.testLabels(score_in)).↙
/numel(obj.testLabels(score_in)))*100);
                figure;
                confusionchart(obj.testLabels(score_in), obj.YPred(score_in),↙
"RowSummary", "row-normalized");
                title("Optimized Accuracy: " + accuracy' + "%")
            end
            figure;
            rocObj = rocmetrics(obj.testLabels(1:length(obj.scores)),obj.scores,["HEA","↙
STR","TBI"],AdditionalMetrics="accu");
            plot(rocObj,"AverageROCType","micro")
        end

        function [ba,acc] = BinaryPlot(obj,class)
            if nargin < 2
                class = "TBI";
            end
            if ischar(class)
                class = string(class);
            end
            switch class
                case "HEA"
                    cid = 1;
                case "STR"
                    cid = 2;
                case "TBI"
                    cid = 3;
            end
            rocObj = rocmetrics(obj.testLabels,obj.scores,["HEA","STR","TBI"],...
                AdditionalMetrics=["accu","prec"]);
            M = rocObj.Metrics;
            M.F1 = 2*(M.TruePositiveRate.*M.PositivePredictiveValue)./(M.TruePositiveRate↙
+ M.PositivePredictiveValue);
            M.BA = mean([M.TruePositiveRate,1-M.FalsePositiveRate],2);
            M.ClassName = string(M.ClassName);
            in = M.ClassName == class;
            M = M(in,:);
            [ba, n] = max(M.BA);
            thresh = M.Threshold(n);
```

```
            ascores = obj.scores;
            ascores(:,1) = ascores(:,1) - max(ascores(:,2),ascores(:,3));
            ascores(:,2) = ascores(:,2) - max(ascores(:,1),ascores(:,3));
            ascores(:,3) = ascores(:,3) - max(ascores(:,1),ascores(:,2));
            in = ascores(:,cid) >= thresh;
            newLabels = repmat("Other",length(obj.scores),1);
            newLabels(in) = class;
            newLabels = categorical(newLabels);
            labels = obj.testLabels;
            labels(labels ~= class) = "Other";
            labels = removecats(labels);
            acc = mean(labels==newLabels);
            figure;
            confusionchart(labels, newLabels, "RowSummary", "row-normalized","↙
ColumnSummary","column-normalized");
            title("Balanced Accuracy: " + (ba*100) + "%")
        end

        function mov = MoviePlot(obj,class)
            if nargin < 2
                class = "TBI";
            end
            if ischar(class)
                class = string(class);
            end
            switch class
                case "HEA"
                    cid = 1;
                case "STR"
                    cid = 2;
                case "TBI"
                    cid = 3;
            end
            rocObj = rocmetrics(obj.testLabels,obj.scores,["HEA","STR","TBI"],...
                AdditionalMetrics=["accu","prec"]);
            M = rocObj.Metrics;
            M.ClassName = string(M.ClassName);
            in = M.ClassName == class;
            M = M(in,:);
            f = figure;
            f.Position(3) = f.Position(3)*2;
            mov(length(M.Threshold)) = struct('cdata',[],'colormap',[]);
            for i = 1:length(M.Threshold)
                subplot(1,2,1)
                ax = gca;
                ax.NextPlot = 'replaceChildren';
                plot(rocObj,"ClassNames",class,"ShowModelOperatingPoint",false)
                %plot(M.TruePositiveRate,M.FalsePositiveRate)
                hold on
                %plot([0 1],[0 1],'k--')
                scatter(M.FalsePositiveRate(i),M.TruePositiveRate(i))
```

```
                legend([class,"Threshold"])
                subplot(1,2,2)
                %ax = gca;
                %ax.NextPlot = 'replaceChildren';
                ascores = obj.scores;
                ascores(:,1) = ascores(:,1) - max(ascores(:,2),ascores(:,3));
                ascores(:,2) = ascores(:,2) - max(ascores(:,1),ascores(:,3));
                ascores(:,3) = ascores(:,3) - max(ascores(:,1),ascores(:,2));
                in = ascores(:,cid) >= M.Threshold(i);
                newLabels = repmat("Other",length(obj.scores),1);
                newLabels(in) = class;
                newLabels = categorical(newLabels);
                labels = obj.testLabels;
                labels(labels ~= class) = "Other";
                labels = removecats(labels);
                acc = mean(labels==newLabels);
                confusionchart(labels, newLabels, "RowSummary", "row-normalized","↙
ColumnSummary","column-normalized");
                cm = confusionmat(labels, newLabels);
                title("Accuracy = " + acc + "/Sensitivity = " + num2str(cm(1,1)/sum(cm↙
(1,:))) + ...
                    "/Precision = " + num2str(cm(1,1)/sum(cm(:,1))))
                mov(i).cdata = print('-RGBImage');
                %mov(i) = getframe(gcf);
            end

        end

        function scoremap = CAM(obj,plotAvg)
            if nargin <2
                plotAvg = false;
            end
            ind = obj.YPred==obj.testLabels;
            Cdata = subset(obj.testData,ind);
            Cscores = obj.scores(ind,:);
            tit = ["HEA","STR","TBI"];
            f = figure;
            f.Position(3) = f.Position(3) * 2;
            f.Position(4) = f.Position(4) * 3;
            for i = 1:3
                [m,in] = max(Cscores(:,i));
                scores = subset(Cdata,in);
                scores.MiniBatchSize = 1;
                reset(scores)
                y = read(scores);
                l = y.Response(1);
                y = y.Predictors{1};
                scoremap = gradCAM(obj.net,y,l);
                t = (0:1:length(y)-1)./scores.newHz;
                subplot(3,1,i)
                plot(t,y,Color=[0,0,0,1/19]);
```

```
            ylim([mean(min(y,[],2)) mean(max(y,[],2))])
            ylabel("EEG Activity")
            ax = gca;
            ax.ColorOrderIndex = 2;
            yyaxis("right")
            plot(t,scoremap,LineWidth=2);
            yl = ylim;
            ylim([0 yl(2)*2])
            ylabel("gradCAM Score")
            title(tit(i) +" (Score = " + m + ")")
            xlabel("Time (s)")
        end
        if plotAvg
            f = figure;
            f.Position(3) = f.Position(3) * 2;
            f.Position(4) = f.Position(4) * 3;
            for i = 1:3
                in = obj.testLabels(ind)==tit(i);
                [~, sin] = sort(obj.scores(in,i),'descend');
                sin = sin(1:floor(length(sin)/10));
                scores = subset(Cdata,sin);
                scores.MiniBatchSize = 1;
                reset(scores)
                avgmap=[];
                for j = 1:length(sin)
                    y = read(scores);
                    l = y.Response(1);
                    y = y.Predictors{1};
                    avgmap(j,:) = gradCAM(obj.net,y,l);
                end
                avgmap = mean(avgmap);
                t = (0:1:length(y)-1)./scores.newHz;
                subplot(3,1,i)
                plot(t,avgmap,LineWidth=2);
                yl = ylim;
                ylim([0 yl(2)*2])
                ylabel("gradCAM Score")
                title("Average " + tit(i) + " gradCAM")
                xlabel("Time (s)")
            end
        end
    end

    function scoremap = CAM2(obj,neg)
        if nargin <2 || neg == false
            neg = 1;
        else
            neg = -1;
        end
        ind = obj.YPred==obj.testLabels;
        Cdata = subset(obj.testData,ind);
```

```
            Cscores = obj.scores(ind,:);
            tit = ["HEA","STR","TBI"];
            f = figure;
            f.Position(3) = f.Position(3) * 5;
            f.Position(4) = f.Position(4) * 3;
            rw = [1:3;4:6;7:9];
            for i = 1:3
                [m,in] = maxk(Cscores(:,i),3);
                scores = subset(Cdata,in);
                scores.MiniBatchSize = 1;
                reset(scores)
                for j = 1:3
                    y = read(scores);
                    l = y.Response(1);
                    y = y.Predictors{1};
                    scoremap = gradCAM(obj.net,y,l);
                    t = (0:1:length(y)-1)./scores.newHz;
                    subplot(3,3,rw(j,i))
                    plot(t,y,Color=[0,0,0,1/19]);
                    ylim([mean(min(y,[],2)) mean(max(y,[],2))])
                    ylabel("EEG Activity")
                    ax = gca;
                    ax.ColorOrderIndex = 2;
                    yyaxis("right")
                    %p = semilogy(t,neg.*scoremap,LineWidth=2);
                    %p.Color(4) = .5;
                    %plot(t,scoremap,LineWidth=2);
                    bar(t,movmean(scoremap,100))
                    yl = ylim;
                    ylim([0 yl(2)*2])
                    %ylim([10^-5 10^-2])
                    xlim([0 180])
                    ylabel("gradCAM Score")
                    %title(tit(i) +" (Score = " + m + ")")
                    xlabel("Time (s)")
                end
            end
%             if plotAvg
%                 f = figure;
%                 f.Position(3) = f.Position(3) * 2;
%                 f.Position(4) = f.Position(4) * 3;
%                 for i = 1:3
%                     in = obj.testLabels(ind)==tit(i);
%                     [~, sin] = sort(obj.scores(in,i),'descend');
%                     sin = sin(1:floor(length(sin)/10));
%                     scores = subset(Cdata,sin);
%                     scores.MiniBatchSize = 1;
%                     reset(scores)
%                     avgmap=[];
%                     for j = 1:length(sin)
%                         y = read(scores);
```

```
%                         l = y.Response(1);
%                         y = y.Predictors{1};
%                         avgmap(j,:) = gradCAM(obj.net,y,l);
%                     end
%                     avgmap = mean(avgmap);
%                     t = (0:1:length(y)-1)./scores.newHz;
%                     subplot(3,1,i)
%                     plot(t,avgmap,LineWidth=2);
%                     yl = ylim;
%                     ylim([0 yl(2)*2])
%                     ylabel("gradCAM Score")
%                     title("Average " + tit(i) + " gradCAM")
%                     xlabel("Time (s)")
%                 end
%             end
        end

        function scoremap = CAMpaper(obj,neg)
            if nargin <2 || neg == false
                neg = 1;
            else
                neg = -1;
            end
            ind = obj.YPred==obj.testLabels;
            Cdata = subset(obj.testData,ind);
            Cscores = obj.scores(ind,:);
            tit = ["HEA","STR","TBI"];
            f = figure("Units","inches");
            f.Position(3) = 7.5;
            f.Position(4) = f.Position(4) * 3;

            rw = [1:3;4:6;7:9];
            k=1;
            for i = [1,3,2]
                [m,in] = maxk(Cscores(:,i),3);
                scores = subset(Cdata,in);
                scores.MiniBatchSize = 1;
                reset(scores)
                y = read(scores);
                if i == 2
                    y = read(scores);
                end
                l = y.Response(1);
                y = y.Predictors{1};
                scoremap = gradCAM(obj.net,y,l);
                t = (0:1:length(y)-1)./scores.newHz;
                subplot(3,1,k)
                plot(t,y,Color=[0,0,0,1/19]);
                ylim([mean(min(y,[],2)) mean(max(y,[],2))])
                ylabel("EEG Activity")
                ax = gca;
```

```
                ax.ColorOrderIndex = 2;
                yyaxis("right")
                %p = semilogy(t,neg.*scoremap,LineWidth=2);
                %p.Color(4) = .5;
                %plot(t,scoremap,LineWidth=2);
                bar(t,movmean(scoremap,100))
                yl = ylim;
                ylim([0 yl(2)*2])
                %ylim([10^-5 10^-2])
                xlim([0 180])
                ylabel("gradCAM Score")
                %title(tit(i) +" (Score = " + m + ")")
                xlabel("Time (s)")
                k= k + 1;
            end
        end



        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```