Linear Regression

The purpose of these coding vignettes is to explore how Linear Regression for classification works. As with the Perceptron Learning Algorithm implementation, I have created my own target function f and data set $\mathcal{D}$. We take d = 2 so we can visualize the problem, and assume $\mathcal{X}$ = [-1, 1]×[-1, 1] with uniform probability of picking each $\mathbb{x} \in \mathcal{X}$. In each run, we choose a random line in the plane as our target function f (we do this by taking two random, uniformly distributed points in [1, 1]×[1, 1] and taking the line passing through them), where one side of the line maps to +1 and the other maps to -1. Then we choose the inputs $\mathbb{x}_n$ of the data set as random points (uniformly in $\mathcal{X}$), and evaluate the target function on each $\mathbb{x}_n$ to get the corresponding output y_n.

```
(* Clear globals *)
Clear[Generate𝒳, DoLinearRegressionExperiment, Ein,
 GetYInterceptForm, LinearTarget, NoFeature, RiskNoiseFlip]
```

```
(* Generate our 𝒳=[-1,1]×[-1,1] space with 𝒹 uniformly distributed
   points and a random 𝒳-partitioning solution function 𝒻. *)

Generate𝒳[OptionsPattern[]] :=
 Module[{
   𝒟, 𝒻, 𝒹 = 2,
   f1, f2,
   m, a, b, c,
   t1, t2,
   dotTest1, dotTest2, DotTest
  },
  f1 = RandomReal[{-1, 1}, {1, 𝒹}][[1]];
  f2 = RandomReal[{-1, 1}, {1, 𝒹}][[1]];

  m = (f1[[2]] - f2[[2]]) / (f1[[1]] - f2[[1]]);
  a = -m;
  b = 1;
  c = m f1[[1]] - f1[[2]];
  𝒻 = {{c}, {a}, {b}};

  (* f should not dot to zero for our two original points! *)
  t1 = {1, f1[[1]], f1[[2]]};
  t2 = {1, f2[[1]], f2[[2]]};

  DotTest[v_, s_] := If[Abs[v] > 0.00000000001, Throw[s], 0];
  DotTest[t1.𝒻, "t1 dot test failed"];
  DotTest[t2.𝒻, "t2 dot test failed"];

  𝒟 = RandomReal[{-1, 1}, {OptionValue[𝒟Size], 𝒹}];

  {𝒟, 𝒻}
 ]
Options[Generate𝒳] = {𝒟Size → 100};
```

```
LinearTarget[f_, 𝕏_] := Sign[𝕏.f];

NoFeature[𝕏_] := 𝕏;

RiskNoiseFlip[percent_] := If[RandomReal[] ≤ percent/100, -1, 1];

DoLinearRegressionExperiment[𝒳_, OptionsPattern[]] :=
 Module[{
   𝒟, f, 𝕪,
   𝕏, 𝕏f, 𝕏fdag,
   𝕨
  },
  {𝒟, f} = 𝒳;
  𝕏 = Function[x, Prepend[x, 1]] /@ 𝒟;
  𝕏f = OptionValue[DataFeature][𝕏];
  𝕪 = RiskNoiseFlip[OptionValue[Noise]] * OptionValue[TargetFunction][f, 𝕏];
  𝕏fdag = PseudoInverse[𝕏f];
  𝕨 = 𝕏fdag.𝕪;
  {𝕨, 𝕏, 𝕪, 𝒟, f}
 ]
Options[DoLinearRegressionExperiment] =
  {TargetFunction → LinearTarget, DataFeature → NoFeature, Noise → 0};

ClassificationEin[𝕏_, 𝕨_, 𝕪_, OptionsPattern[]] :=
 Module[{
   N, misses, sumOfMisses
  },
  N = Length[𝕪];
  (*misses=MapThread[(If[#1≠#2,1,0])&,{Sign[𝕏.𝕨],𝕪}];*)
  misses =
   MapThread[(If[#1 ≠ #2, 1, 0]) &, {OptionValue[TargetFunction][𝕨, 𝕏], 𝕪}];
  sumOfMisses = Total[misses];
  1/N sumOfMisses
 ]
Options[ClassificationEin] = {TargetFunction → LinearTarget};

RegressionsEin[𝕏_, 𝕨_, 𝕪_] :=
 Module[{
   N
  },
  N = Length[𝕪];
  1/N Norm[Sign[X.𝕨] - 𝕪]^2
 ]

(*
 AltEin[𝕏_,𝕨_,𝕪_]:=
  Module[{
```

```
    N
   },
   N=Length[y];
   1/N ∑_{n=1}^N (Transpose[w].X[[n]]-y[[n]])^2
   ]
*)

GetYInterceptForm[w_] :=
 Module[{
   a, b, c, M, B
  },
  a = w[[2]];
  b = w[[3]];
  c = w[[1]];
  M = -a/b; (*If[b≠0,-a/b,10000];*)
  B = -c/b; (*If[b≠0,-c/b,10000];*)
  {M, B}
 ]

(* I don't think we're going to need this,
but who knows so I'll keep it around until we
 get the perceptron reimplemented in Mathematica. *)
(*
GetBoundaryPoints[w_]:=
 Module[{
   a,b,c,
   xAtYmax,xAtYmin,
   yAtXmax,yAtXmin,
   m
  },
  a=w[[2]];
  b=w[[3]];
  c=w[[1]];
  xAtYmax=(-c-b)/a;
  xAtYmin=(-c+b)/a;
  yAtXmax=(-c-a)/b;
  yAtXmin=(-c+a)/b;
  m=Sign[-a/b];
  {x1,y1,x2,y2}=
   Switch[
    m≥0,True,Switch[

    ]*)
```

Finding g when N = 100

Take N = 100. Use Linear Regression to find g and evaluate E_{in}, the fraction of in-sample points which got classified incorrectly. Repeat the experiment 1000 times and take the average (keep the g's as they will be used again later). Note the average E_{in}.

```
(* Clear globals *)
Clear[Experiment1]
Clear[e1avEin, e1fs, e1gs, e1D, e1runs, n]
Clear[DoInteractiveLR, DoAutoLR]
```

```
(* The interactive mode gives us the option of stepping through
 randomly generated data/f/g just to see what they look like. *)
DoInteractiveLR[] :=
 DynamicModule[{
   runs = 1000, N = 100,
   g, gs,
   X, y,
   D, f,
   avEin,
   i,
   fm, fb,
   sm, sb
  },
  gs = {};
  i = 1;
  {g, X, y, D, f} = DoLinearRegressionExperiment[GenerateX[]];
  EventHandler[Dynamic[(
     (*avEin=Mean[Function[x,Ein[X,x,y]]/@gs];*)
     avEin = ClassificationEin[X, g, y];
     {fm, fb} = GetYInterceptForm[f];
     {sm, sb} = GetYInterceptForm[g];
     Show[ListPlot[D, PlotLabel → StringForm[
         "Linear regression results\n(runs=``, N=``):\naverage (class.) Ein=``",
         runs, N, avEin]],
      Plot[{fm x + fb + 0.05, sm x + sb}, {x, -2, 2}]
     ]
    )],
   {"KeyDown" :→ ({g, X, y, D, f} = DoLinearRegressionExperiment[GenerateX[]]; i++)}
  ]
 ]
```

```
(* Auto mode implements what the problem asks for:
  it runs the tests and then outputs a fomatted string bearing the results. *)
DoAutoLR[] :=
 Module[{
   runs = 1000, N = 100,
   g, gs,
   y, ys,
   X, Xs,
   f, fs,
   D,
   avEin,
   i, n
  },
  fs = {};
```

```
  𝕘s = {};
  𝕪s = {};
  𝕏s = {};
  For[i = 1, i ≤ runs, i++, (
    {𝕘, 𝕏, 𝕪, 𝒟, 𝑓} = DoLinearRegressionExperiment[Generate𝒳[𝒟Size → 100]];
    AppendTo[𝑓s, 𝑓];
    AppendTo[𝕘s, 𝕘];
    AppendTo[𝕪s, 𝕪];
    AppendTo[𝕏s, 𝕏]
   )];
  avEin =
   Mean[MapThread[Function[{x, y, z}, ClassificationEin[x, y, z]], {𝕏s, 𝕘s, 𝕪s}]];
  n = Length[𝒟];
  {𝑓s, 𝕘s, 𝒟, avEin, runs, n}
 ]
```

```
Experiment1[interactive_] := If[interactive, DoInteractiveLR[], DoAutoLR[]]
{e1𝑓s, e1𝕘s, e1𝒟, e1avEin, e1runs, e1n} = Experiment1[False];
(*Experiment1[True]*)
StringForm[
 "Linear regression results\n(runs=``, N=``):\naverage (class.) Ein=``",
 e1runs, e1n, e1avEin × 1.]
```

```
Linear regression results
(runs=1000, N=100):
average (class.) Ein=0.03898`
```

Evaluating E_{out} with N = 1000

Now, generate 1000 fresh points and use them to estimate the out-of-sample error E_{out} of g that we got above (number of misclassified out-of-sample points / total number of out-of-sample points). Again, we run the experiment 1000 times and take the average and note the average E_{out}.

```
(* clear globals *)
Clear[Experiment2]
Clear[e2𝑓s, e2𝕘s, e2avEout, e2runs, e2n]
```

```
DoEoutTest[𝑓s_, 𝕘s_] :=
 Module[{
   runs = 1000, N = 1000,
   𝕘,
   𝕪, 𝕪s,
   𝕏, 𝕏s,
   𝑓,
   𝒟,
   avEout,
   i
  },
  𝕪s = {};
  𝕏s = {};
  For[i = 1, i ≤ runs, i++, (
    {𝕘, 𝕏, 𝕪, 𝒟, 𝑓} =
     DoLinearRegressionExperiment[{Generate𝒳[𝒟Size → N][[1]], 𝑓s[[i]]}];
    AppendTo[𝕪s, 𝕪];
    AppendTo[𝕏s, 𝕏];
   )];
  avEout =
   Mean[MapThread[Function[{x, y, z}, ClassificationEin[x, y, z]], {𝕏s, 𝕘s, 𝕪s}]];
  n = Length[𝒟];
  {𝑓s, 𝕘s, avEout, runs, n}
 ]

Experiment2[] := DoEoutTest[e1𝑓s, e1𝕘s];
{e2𝑓s, e2𝕘s, e2avEout, e2runs, e2n} = Experiment2[];
StringForm[
 "Linear regression results\n(runs=``, N=``):\naverage (class.) Eout=``",
 e2runs, e2n, e2avEout × 1.]
```

```
Linear regression results
(runs=1000, N=1000):
average (class.) Eout=0.047891`
```

Using linear regression to determine initial weights for PLA, N=10

Using N = 10, we find the weights using Linear Regression and then use them as a vector of initial weights for the Perceptron Learning Algorithm. We then run PLA until it converges to a final vector of weights that completely separates all the in-sample points. We can then examine the average number of iterations (over 1000 runs) that PLA takes to converge.

```
(* clear globals *)
Clear[DoPLAExperiment, DoInteractivePLAExperiment, DoLRtoPLASeries]
Clear[e3runs, e3n, e3AveIterations]
Clear[Experiment3]
```

```
DoPLAStep[w_, 𝑓_, 𝒟_, Converged_, Misses_] :=
 Module[{
   𝕩, h, chk,
   missedPoints,
   misses,
```

```
    numSamples,
    converged,
    w,
    i, ri
   },
   converged = Converged;
   misses = Misses;
   w = 𝕨;
   If[converged == 0, (
     converged = 1;
     missedPoints = {};
     numSamples = Length[𝒟];

     For[i = 1, i ≤ numSamples, i++, (
       (* cycle through the samples looking for misses *)
       𝕩 = Transpose[ArrayReshape[Prepend[𝒟[[i]], 1], {1, 3}]];
       h = Sign[(Transpose[w].𝕩)[[1, 1]]];
       chk = Sign[(Transpose[𝒻].𝕩)[[1, 1]]];

       If[chk ≠ h, (
         (* if we have a miss...*)
         converged = 0;
         AppendTo[missedPoints, 𝕩];
         (*iterMisses++;*)
         0), 0]
      )];

     If[converged == 0, (
       (* if we've missed any points,
       then pick one of them at random for the hypothesis update *)
       misses++;
       ri = RandomInteger[{1, Length[missedPoints]}];
       𝕩 = missedPoints[[ri]];
       h = Sign[(Transpose[w].𝕩)[[1, 1]]];
       chk = Sign[(Transpose[𝒻].𝕩)[[1, 1]]];
       w = w + chk * 𝕩; (* avoiding "cannot assign to raw object 0" *)
       (*𝕨=q*)(* ...maybe because it was self-
        assigning and the last value in a "block" thing? *)
       (*ah no I think it's because it's declared as a parameter!*)
      )];
    )];
   {w, converged, misses}
  ]

DoPLAExperiment[𝒳_, OptionsPattern[]] :=
 Module[{
    𝒟, 𝒻, w, 𝕘,
    misses = 0,
    converged = 0
   },
   {𝒟, 𝒻} = 𝒳;
   w = OptionValue[𝕨];

   While[converged == 0, (
     {w, converged, misses} = DoPLAStep[w, 𝒻, 𝒟, converged, misses];
```

```
    )];

   𝕘 = 𝕨;
   {𝕘, Length[𝒟], misses}
  ]
Options[DoPLAExperiment] = {𝕨 → {{0}, {0}, {0}}};

DoInteractivePLAExperiment[𝒳_] :=
 DynamicModule[{
   𝒟, f, 𝕨, 𝕘,
   𝒟pos, 𝒟neg,
   fm, fb, sm, sb,
   misses = 0,
   converged = 0,
   i = 0
  },
  {𝒟, f} = 𝒳;
  𝕨 = {{0.000000001}, {0.000000001}, {0.000000001}};
  {𝕨, converged, misses} = DoPLAStep[𝕨, f, 𝒟, converged, misses];

  EventHandler[Dynamic[(
      {fm, fb} = GetYInterceptForm[f];
      {sm, sb} = GetYInterceptForm[𝕨];
      𝒟pos = Select[𝒟, Sign[ArrayReshape[Prepend[#, 1], {1, 3}].f][[1, 1]] ≥ 0 &];
      𝒟neg = Select[𝒟, Sign[ArrayReshape[Prepend[#, 1], {1, 3}].f][[1, 1]] < 0 &];
      Show[
       ListPlot[{𝒟pos, 𝒟neg}, PlotMarkers → {"+", "-"}, PlotLabel → StringForm[
          "i=``, misses=``", i, misses]], Plot[{fm x + fb, sm x + sb}, {x, -2, 2}]
      ]
     )],
   {"KeyDown" :→ ({𝕨, converged, misses} = If[converged == 0,
        DoPLAStep[𝕨, f, 𝒟, converged, misses], {𝕨, converged, misses}]; i++)}
  ]
 ]

DoLRtoPLASeries[runs_] :=
 Module[{
   𝕘, 𝕏, 𝕪, 𝒟, f, i, n,
   iterations, iterationses, avIterations
  },
  iterationses = {};
  For[i = 1, i ≤ runs, i++, (
    {𝕘, 𝕏, 𝕪, 𝒟, f} = DoLinearRegressionExperiment[Generate𝒳[𝒟Size → 10]];
    {𝕘, n, iterations} = DoPLAExperiment[{𝒟, f}, 𝕨 → 𝕘];
    AppendTo[iterationses, iterations];
   )];
  avIterations = Mean[iterationses];
  {runs, Length[𝒟], avIterations}
 ]

(* interactive version *)
(*Experiment3[]:=DoInteractivePLAExperiment[Generate𝒳[]]
  Experiment3[]*)

(* single-run test version *)
```

```
(*Experiment3[]:=DoPLAExperiment[Generate𝒳[]];
{e3g,e3n,e3iterations}=Experiment3[];
StringForm["PLA results\n(N=``):\niterations to convergence=``",
 e3n, e3iterations]*)

(* linear regression to PLA version *)
Experiment3[] := DoLRtoPLASeries[1000];
{e3runs, e3n, e3AveIterations} = Experiment3[];
StringForm[
 "LR to PLA results\n(runs=``, N=``):\naverage iterations to convergence=``",
 e3runs, e3n, e3AveIterations × 1.]
```

```
LR to PLA results
(runs=1000, N=10):
average iterations to convergence=4.005`
```

Nonlinear Transformation

In these experiements we again apply Linear Regression for classification. Using the target function:

$f(x_1,\ x_2) = \text{sign}(x_1{}^2 + x_2{}^2 - 0.6)$

...we generate a training set of N = 1000 points on $\mathcal{X}$ = [-1, 1] × [-1, 1] with a uniform probability of picking each **x** ∈ $\mathcal{X}$ and generate simulated noise by flipping the sign of the output in a randomly selected 10% subset of the generated training set.

Linear Regression (sans transformation)

We carry out Linear Regression without transformation, i.e., with feature vector:

$(1, x_1, x_2)$,

to find the weight **w**. We will determine the classification in-sample error E_{in} by running the experiment 1000 times and taking the average E_{in} to reduce variation in our results.

```
(* clear globals *)
 Clear[NonlinearTarget]
 Clear[Experiment4, DoNonlinearDataSeries]
 Clear[{e4𝒻s, e4𝑔s, e4𝒟, e4avEin, e4avG, e4runs, e4n]
```

```
NonlinearTarget[f_, 𝕏_] := Sign[(#2^2 + #3^2 - 0.6)] & @@@ 𝕏;

DoNonlinearDataSeries[runs_, N_] :=
 Module[{
   𝕘, 𝕘s,
   𝕪, 𝕪s,
   𝕏, 𝕏s,
   f, fs,
   𝒟,
   avEin, avG,
   i, n
  },
  fs = {};
  𝕘s = {};
  𝕪s = {};
  𝕏s = {};
  For[i = 1, i ≤ runs, i++, (
    {𝕘, 𝕏, 𝕪, 𝒟, f} = DoLinearRegressionExperiment[
      Generate𝒳[𝒟Size → N], TargetFunction → NonlinearTarget, Noise → 10];
    AppendTo[fs, f];
    AppendTo[𝕘s, 𝕘];
    AppendTo[𝕪s, 𝕪];
    AppendTo[𝕏s, 𝕏]
   )];
  avEin =
   Mean[MapThread[Function[{x, y, z}, ClassificationEin[x, y, z]], {𝕏s, 𝕘s, 𝕪s}]];
  avG = Mean[𝕘s];
  n = Length[𝒟];
  {fs, 𝕘s, 𝒟, avEin, avG, runs, n}
 ]

Experiment4[] := DoNonlinearDataSeries[1000, 1000];
{e4fs, e4𝕘s, e4𝒟, e4avEin, e4avG, e4runs, e4n} = Experiment4[];
StringForm["Linear regression results\n(runs=``, N=``, Target=Nonlinear,
   Features=None):\n𝕘=`` + ``x1 + ``x2\naverage (class.) Ein=``",
 e4runs, e4n, e4avG[[1]], e4avG[[2]], e4avG[[3]], e4avEin × 1.]
```

```
Linear regression results
(runs=1000, N=1000, Target=Nonlinear, Features=None):
𝕘=0.04742016410294729` + 0.00325062 x1 + -0.00129323 x2
average (class.) Ein=0.507388`
```

Using nonlinear transformation, N = 1000

Now, we transform the N = 1000 training data into the following nonlinear feature vector:

$(1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$

and then find the vector $\tilde{\mathbf{w}}$ that corresponds to the solution of Linear Regression and display our final hypothesis, averaged over a few runs to make sure our answer is stable.

```
(* clear globals *)
 Clear[NonlinearFeature]
 Clear[Experiment5, DoNonlinearDataSeriesWithFeatures]
 Clear[{e5𝒻s, e5𝕘s, e5𝒟, e5avEin, e5avG, e5runs, e5n]
```

```
NonlinearFeature[𝕏_] := {1, #2, #3, #2 #3, #2^2, #3^2} & @@@ 𝕏;

DoNonlinearDataSeriesWithFeatures[runs_, N_, 𝒟_, 𝒻s_] :=
 Module[{
    𝕘, 𝕘s,
    𝕪, 𝕪s,
    𝕏, 𝕏s,
    D, 𝒻,
    avEin, avG,
    i, n
   },
   𝕘s = {};
   𝕪s = {};
   𝕏s = {};
   D = 𝒟;
   For[i = 1, i ≤ runs, i++, (
     {𝕘, 𝕏, 𝕪, D, 𝒻} = DoLinearRegressionExperiment[{D, 𝒻s[[i]]}, TargetFunction →
        NonlinearTarget, Noise → 10, DataFeature → NonlinearFeature];
     AppendTo[𝕘s, 𝕘];
     AppendTo[𝕪s, 𝕪];
     AppendTo[𝕏s, 𝕏]
    )];
   avEin = Mean[MapThread[Function[{x, y, z}, ClassificationEin[
        x, y, z, TargetFunction → NonlinearTarget]], {𝕏s, 𝕘s, 𝕪s}]];
   avG = Mean[𝕘s];
   n = Length[D];
   {𝒻s, 𝕘s, D, avEin, avG, runs, n}
  ]

Experiment5[] := DoNonlinearDataSeriesWithFeatures[1000, 1000, e4𝒟, e4𝒻s];
{e5𝒻s, e5𝕘s, e5𝒟, e5avEin, e5avG, e5runs, e5n} = Experiment5[];
StringForm["Linear regression results\n(runs=``,
   N=``, Target=Nonlinear, Features=Nonlinear):\n𝕘=`` + ``x₁
   + ``x₂ + ``x₁x₂ + ``x₁² + ``x₂²\naverage (class.) E_in=``",
 e5runs, e5n, e5avG[[1]], e5avG[[2]], e5avG[[3]], e5avG[[4]],
 e5avG[[5]], e5avG[[6]], e5avEin × 1.]
```

Linear regression results
(runs=1000, N=1000, Target=Nonlinear, Features=Nonlinear):
$\mathbb{g}$=-0.95734 + -0.00543674 x_1 +
  0.0180005 x_2 + 0.0173879 $x_1 x_2$ + 1.53297 x_1^2 + 1.50226 x_2^2
average (class.) E_{in}=0.119`

E_{out} for nonlinear hypothesis

Estimate the value to the classification out-of-sample error E_{out} of our hypothesis above by generating a new set of 1000 points and adding noise, as before, then averaging over 1000 runs to reduce the variation in results.

```
(* clear globals *)
 Clear[Experiment6, DoNonlinearDataSeriesEoutWithFeatures]
 Clear[{e6𝑓s, e6𝕘s, e6𝒟, e6avEin, e6avG, e6runs, e6n]
```

```
DoNonlinearDataSeriesEoutWithFeatures[runs_, N_, 𝒟_, 𝑓s_, 𝕘s_] :=
 Module[{
    𝕘,
    𝕪, 𝕪s,
    𝕏, 𝕏s,
    D, 𝑓,
    avEin, avG,
    i, n
   },
  𝕪s = {};
  𝕏s = {};
  D = 𝒟;

  For[i = 1, i ≤ runs, i++, (
     {𝕘, 𝕏, 𝕪, D, 𝑓} = DoLinearRegressionExperiment[
       {Generate𝒳[𝒟Size → N][[1]], 𝑓s[[i]]}, TargetFunction → NonlinearTarget,
       Noise → 10, DataFeature → NonlinearFeature];
     AppendTo[𝕪s, 𝕪];
     AppendTo[𝕏s, 𝕏]
    )];
  avEout = Mean[MapThread[Function[{x, y, z}, ClassificationEin[
        x, y, z, TargetFunction → NonlinearTarget]], {𝕏s, 𝕘s, 𝕪s}]];
  avG = Mean[𝕘s];
  n = Length[D];
  {𝑓s, 𝕘s, D, avEout, avG, runs, n}
 ]
```

```
Experiment6[] :=
  DoNonlinearDataSeriesEoutWithFeatures[e5runs, e5n, e5𝒟, e5𝑓s, e5𝕘s];
{e6𝑓s, e6𝕘s, e6𝒟, e6avEin, e6avG, e6runs, e6n} = Experiment6[];
StringForm["Linear regression results\n(runs=``,
   N=``, Target=Nonlinear, Features=Nonlinear):\n𝕘=`` + ``x₁
   + ``x₂ + ``x₁x₂ + ``x₁² + ``x₂²\naverage (class.) Eout=``",
 e6runs, e6n, e6avG[[1]], e6avG[[2]], e6avG[[3]], e6avG[[4]],
 e6avG[[5]], e6avG[[6]], e6avEin × 1.]
```

Linear regression results
(runs=1000, N=1000, Target=Nonlinear, Features=Nonlinear):
$\mathbb{g}$=-0.95734 + -0.00543674 x_1 +
0.0180005 x_2 + 0.0173879 $x_1 x_2$ + 1.53297 x_1^2 + 1.50226 x_2^2
average (class.) E_{out}=0.085`