Machine Learning for Quantum Mechanics in a Nutshell

Part of supplementary material for
M. Rupp, *Int J Quant Chem*, 2015.

\

Please adjust the following path to where you unpacked the reference \

implementation code from the supplementary material.\

```
AppendTo [$Path , FileNameJoin [{"Path ", "to", "library "}]];
(* Parent directory containing QMMLPack directory *)
```

```
Needs ["QMMLPack` "]
```

```
© Matthias Rupp 2006–2015. Please cite
M. Rupp: Machine Learning for Quantum
  Mechanics in a Nutshell , Int. J. Quant. Chem., 2015.
```

\

The following "Code" section \

contains auxiliary code and can simply be executed.\

Code

```
On[Assert ];
```

Auxiliary routines

Plots reference labels versus predictions

```
Clear [ScatterPlot ];

ScatterPlot ::usage  = "ScatterPlot [y,f] plots  true  values  y versus  predicted  values  \

f.";

ScatterPlot [true_ , pred_] := Module [{min , max},
   {min , max} = {Min[true , pred], Max[true , pred]};  (* Cover  all  points  *)
   ListPlot [Transpose [{true , pred }], PlotRange  → {{min , max}, {min , max}},
    Frame → {{True , False }, {True , False }}, FrameStyle  → 12 ,
    FrameLabel  → {{"predicted ", None }, {"reference ", None }},
    Prolog → {Dashed , Black , Line [{{min , min}, {max , max}}]}
   ]
  ];
\
```

Plots performance as a function of the two hyperparameters λ and \

σ\

```
Clear [HyperparameterPlot  ];

HyperparameterPlot  ::usage  =
  "HyperparameterPlot  [hpgrid ,perf ,stat] plots  stat  from  performance  \

statistic  evaluated  at lsgrid  x rsgrid .";

(* Uses  (1-(1-x)^p)^1/p  to  increase  color  range  for
 colors  in [0,1].  Felix  Faber  came  up  with  this  function . *)
HyperparameterPlot  [rsgrid_ , lsgrid_ , perf_ , stat_String  , dots_ : {}] := Module [{
    hpgrid , data , min , max , scalef , contours , opt ,
    colorf , rsticks10  , lsticks10  , epilog , gopts , legend , p1 , p2 ,
    slabels  = {"RootMeanSquareError  " → "RMSE ", "MeanAbsoluteError  " → "MAE ",
      "CorrelationSquared  " → "R²", "OneMinusCorrelationSquared   " → "1-R²"},
    colorbasef  = ColorData ["TemperatureMap  "], colorfexp  = 3
    (* was  2.5  *), precision  = 10. ^-6},

   (* Set  up  data  to  plot  *)
   hpgrid  = Outer [List , lsgrid , rsgrid , 1]; (* Tuples  {σ,λ} arranged  as  a  grid  *)
   data = Flatten [
     MapThread [Append [#1, Round [#2, precision ]] &, {Log[2, hpgrid ], (stat /. perf)}, 2], 1];
   (* List of tuples {{σ,λ},s}, where s is the chosen performance statistic *)

   min = Min[data⟦All , 3⟧];
   max = Max[data⟦All , 3⟧];
   scalef = ((# - min) / (max - min) &);
   (* Range of performance statistic and function scaling to interval [0,1] *)
   contours = Quantile [data⟦All , 3⟧, {0.01 , 0.05 , 0.1 , 0.2 , 0.3 , 0.4 , 0.5 , 0.6}];
   (* Quantiles to use for contour lines *)

   opt = Extract [data , Position [data⟦All , 3⟧, Min[data⟦All , 3⟧], {1}, Heads → False ]];
   (* Position of optimal hyperparameters *)
   opt = First [Sort [opt , (OrderedQ [{Norm [#1], Norm [#2]}] &)]];
   (* If there is an optimal basin , prefer larger values for regularization *)

   (* Set up graph *)
   colorf = Function [x, colorbasef [Power [1 - (1 - scalef [x]) ^ colorfexp , 1 / colorfexp ]]];
   (* Color function *)

   rsticks10 = Log[10 , rsgrid ]; (* Regularizatin strength axes ticks with base 10 *)
   rsticks10 = FindDivisions [{Min[rsticks10 ], Max[rsticks10 ]}, 5];
   (* Find "nice" values *)
   rsticks10 = Map[{Log[2, Power [10 , #]], #} &, rsticks10 ];
   (* Locations with respect to base 2 used by the graph *)

   lsticks10 = Log[10 , lsgrid ]; (* Length scale axis ticks with base 10 *)
   lsticks10 = FindDivisions [{Min[lsticks10 ], Max[lsticks10 ]}, 5];
   (* Find "nice" values *)
   lsticks10 = Map[{Log[2, Power [10 , #]], #} &, lsticks10 ];
   (* Locations with respect to base 2 used by the graph *)

   epilog = {Black , PointSize [Large ], Point [opt⟦{1, 2}⟧], Orange , Point [Log[2, dots ]]};
   (* Epilog plots points at location of
    optimal hyperparameters and any passed ones *)

   (* Create graph *)
   gopts = {
     ColorFunction → colorf , ColorFunctionScaling → False ,
     FrameLabel → {{"log₂(λ)", "log₁₀(λ)"}, {"log₂(σ)", "log₁₀(σ)"}}, FrameStyle → 12 ,
     FrameTicks → {{Automatic , N[rsticks10 ]}, {Automatic , lsticks10 }},
     ImageSize → Medium
    };
```

```
    legend = BarLegend[{colorf, {min, max}}, Round[contours, 0.1],
      LegendLabel → stat /. slabels, LabelStyle → 12, LegendMarkerSize → Automatic];
    p1 = ListDensityPlot[data, InterpolationOrder → 1, gopts, PlotRange → All];
    p2 = ListContourPlot[data, ContourShading → None, Contours → contours,
      ContourStyle → Map[Directive[Thick, Darker[colorf[#]]] &, contours], gopts];

    Legended[Show[{p1, p2}, FilterRules[gopts, Options[Graphics]], Epilog → epilog],
     Placed[legend, After]]
  ];
```

Predicting atomization energies

The dataset

Obtaining the data

\

Download the dataset, set the corresponding path and filename.\

```
filename = FileNameJoin[{"path", "to", "dataset", "dsgdb7ae2.xyz"}];
```

Load the data.

```
raw = Import[filename, "extXYZ"];
an = "VertexTypes" /. raw; (* Element types as string abbreviations *)
xyz = "VertexCoordinates" /. raw; (* Atom positions in Åström *)
ae = ("MolecularProperties" /. raw)⟦2⟧; (* Atomization energies in kcal/mol *)
```

Counting element types

Count number of molecules with given number of non-H atoms

```
(* Bin by number of non-H atoms *)
table4 = Tally[Map[Count[#, Except["H"]] &, an]]
```

```
{{1, 1}, {2, 3}, {3, 12}, {4, 43}, {5, 157}, {6, 935}, {7, 5951}}
```

```
TableForm[Transpose[Append[table4, {Σ, Total[table4⟦All, 2⟧]}]]]
```

```
1    2    3     4     5      6      7       Σ
1    3    12    43    157    935    5951    7102
```

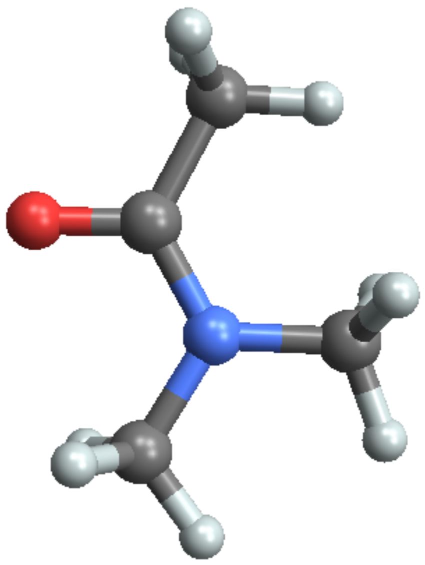
Visualizing molecules

Plot routine expects coordinates in pm, so multiply coordinates by 100.
\

Change index *i* to visualize other molecules. Use the mouse to rotate,
hold the shift \

key to pan, hold the command key to zoom.

```
i = 1046 ; MoleculePlot3D [an⟦i⟧, xyz⟦i⟧ * 100]
```



Creating a training set

```
nmol = Length[an]; (* Number of molecules in dataset *)
indtrain = Flatten[Position[an, _?(Count[#, Except["H"]] ≤ 4 &), {1}, Heads → False]];
(* Indices of molecules with 4 or fewer non-H atoms *)
t = Length[indtrain]; (* Number of such molecules *)

ind = Complement[Range[nmol], indtrain]; (* Indices of remaining molecules *)
ind = Sort[ind, (OrderedQ[{Length[an⟦#1⟧], Length[an⟦#2⟧]}] &)];
(* Sort remaining molecules by number of atoms *)
ind = ind⟦Round[Range[1, nmol - t, (nmol - t)/(1000 - t)]]⟧;
(* Stratified selection of 941 molecules *)
indtrain = Join[indtrain, ind]; (* Add those to the training set *)

indpredict = Complement[Range[nmol], indtrain];
(* All other molecules go into the prediction set *)

Assert[Length[indtrain] == 1000]
Assert[Intersection[indtrain, indpredict] === {}]
```

Verify stratification graphically

```
ListPlot[Map[Length, an⟦indtrain⟧]]
```

Creating a hold-out set

```
ind = Select[indtrain, (Count[an⟦#⟧, Except["H"]] > 4 &)];
(* All molecules in training set with 5 or more non-H atoms *)
ind = Sort[ind, (OrderedQ[{Length[an⟦#1⟧], Length[an⟦#2⟧]}] &)];
(* Sort them by number of atoms *)
ind = ind⟦Round[Range[1, Length[ind], Length[ind]/100]]⟧;
(* Stratified selection of 100 molecules *)

indholdout = ind;
indproper = Complement[indtrain, indholdout];
(* Remaining molecules constitute training set proper *)

Assert[Length[indholdout] == 100]
Assert[Intersection[indproper, indholdout] === {}]
Assert[Union[indproper, indholdout] == Sort[indtrain]]
```

Representation

Computing Coulomb matrices

\

In preparation, convert element types from string abbreviations to atomic \

numbers, and, coordinates from Ångström to atomic units \

(Bohr radii).\

```
(* Convert element types from strings to atomic numbers *)
an = Map[ElementData [#, "AtomicNumber "] &, an, {2}]; (* H→1, C→6, N→7, O→8, S→16 *)
```

```
(* Convert atom positions to atomic units *)
ang2bohr =
  QuantityMagnitude [UnitConvert [Quantity [1, "Ångström "], Quantity [1, "BohrRadius "]]]
 (* 100/52.917720859 *);
xyz = xyz * ang2bohr ;
```

Coulomb matrices.

```
cm = Table[
   If[i == j, 0.5 an⟦k, i⟧^2.4 ,
    an⟦k, i⟧ an⟦k, j⟧ / EuclideanDistance [xyz⟦k, i⟧, xyz⟦k, j⟧]] (* Equ . 26 *)
   , {k, nmol }, {i , Length [an⟦k⟧]}, {j , Length [an⟦k⟧]}];
```

Sort by row norm.

```
t = Table [Norm [cm⟦k, i⟧], {k, nmol }, {i , Length [an⟦k⟧]}];
(* Row norms for each matrix 's rows *)
t = Map[Reverse [Ordering [#]] &, t];
(* Yields index reordering for descending order of row norms *)
cm = MapThread [#1⟦#2 , #2⟧ &, {cm , t}];
(* Simultaneously rearrange rows and columsn of each matrix *)
```

Pad.

```
Max[Map[Length , an]] (* Maximum number of atoms in dataset *)
23
```

```
cm = Map[PadRight [#, {23 , 23}] &, cm];
```

Vectorize

```
cm = Map[LowerTriangularPart , cm];
```

```
Dimensions [cm]
{7102 , 276}
```

Model building

Basic model

Choose some values for hyperparameters.

```
{λ, σ} = {2. ^-22.5 , 2. ^11.5}(* e.g., center of logarithmic grid in Equ. 27 *)
```

$\{1.68587 \times 10^{-7}, 2896.31\}$

Compute kernel matrix K.

```
kk = KernelGaussian [cm⟦indproper ⟧, {σ}];
```

Train kernel ridge regression model.

```
krr = KernelRidgeRegression [kk, ae⟦indproper ⟧, RegularizationStrength → λ];
```

Predict hold-out set.

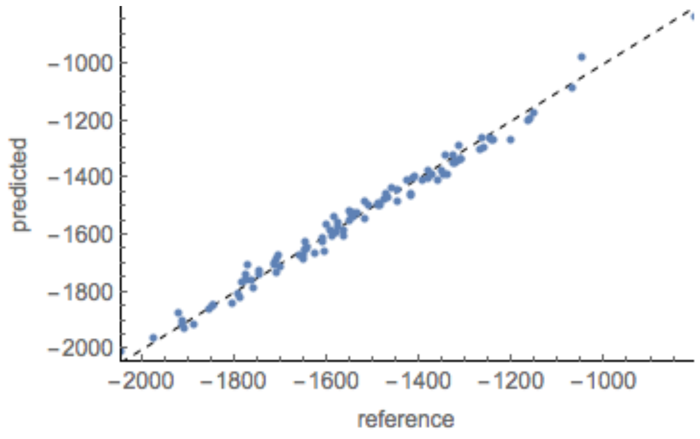
```
ll = KernelGaussian [cm⟦indproper ⟧, cm⟦indholdout ⟧, {σ}];

f = krr[ll];
```

Performance statistics

```
Loss[ae⟦indholdout ⟧, f]
{NumberOfSamples → 100 , RootMeanSquareError → 26.2827 , MeanAbsoluteError → 21.4907 ,
 StandardDeviation → 26.2914 , MedianAbsoluteError → 20.1599 ,
 MaximumAbsoluteError → 74.2684 , Correlation → 0.993541 ,
 CorrelationSquared → 0.987124 , OneMinusCorrelationSquared → 0.0128764 }
```

```
ScatterPlot [ae⟦indholdout ⟧, f]
```



```
Clear[λ, σ]
```

Grid search

Set up grid of hyperparameter values.

```
(* Equ. 27 *)
rsgrid = Power[2, Range[-40, -5, 0.5]];
lsgrid = Power[2, Range[5, 18, 0.5]];
```

\

Define a function that returns performance on hold-out set given \

hyperparameter values.\

```
Clear[estperf];

(* Parameters are regularization strength, length scale, and kernel *)
estperf[λ_, σ_, k_] := Module[{kk, ll, krr, pred, loss},
    kk = k[cm⟦indproper⟧, {σ}];
    ll = k[cm⟦indproper⟧, cm⟦indholdout⟧, {σ}];
    krr = KernelRidgeRegression[kk, ae⟦indproper⟧, RegularizationStrength → λ];
    pred = krr[ll, "Predictions"];
    Loss[ae⟦indholdout⟧, pred]
  ];
```

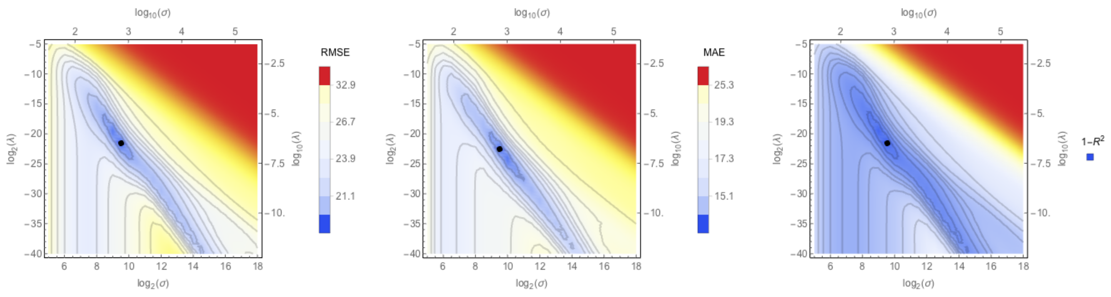
Evaluate the functions on the grid:

```
Dynamic[Log[2, {λ, σ}]]

perfG = Table[estperf[λ, σ, KernelGaussian], {σ, lsgrid}, {λ, rsgrid}];

(*
filename =FileNameJoin[{"path","filename.txt.bz2"}];
Export[filename,ToString[perfG,InputForm],{"BZIP2","String"}];
(* Use this line to store results *)
perfG =ToExpression[Import[filename,{"BZIP2","String"}]];
(* Use this line to retrieve previously stored results *)
*)
```

```
GraphicsRow [Table [HyperparameterPlot  [rsgrid , lsgrid , perfG , stat],
   {stat , {"RootMeanSquareError  ", "MeanAbsoluteError  ", "OneMinusCorrelationSquared   "}}],
  ImageSize  → 18.5 * 72]
```



Results

Optimal hyperparameters for each statistic:

```
Table [t = Position [stat /. perfG , Min[stat /. perfG ], {2}, Heads → False ];
   Join [Log [2, {rsgrid ⟦t⟦1, 2⟧⟧, lsgrid ⟦t⟦1, 1⟧⟧}],
    Log [10 , {rsgrid ⟦t⟦1, 2⟧⟧, lsgrid ⟦t⟦1, 1⟧⟧}], Extract [stat /. perfG , t]],
   {stat , {"RootMeanSquareError  ", "MeanAbsoluteError  ", "OneMinusCorrelationSquared   "}}] //
  TraditionalForm
```

$$\begin{pmatrix} -21.5 & 9.5 & -6.47214 & 2.85978 & 19.161 \\ -22.5 & 9.5 & -6.77317 & 2.85978 & 13.3594 \\ -21.5 & 9.5 & -6.47214 & 2.85978 & 0.00717638 \end{pmatrix}$$

```
estperf [2. ^-22 , 2. ^9.5 , KernelGaussian  ]
```

```
{NumberOfSamples  → 100 , RootMeanSquareError  → 19.2365 , MeanAbsoluteError  → 13.4748 ,
 StandardDeviation  → 19.3156 , MedianAbsoluteError  → 10.1084 ,
 MaximumAbsoluteError  → 75.0852 , Correlation → 0.996369 ,
 CorrelationSquared  → 0.992751 , OneMinusCorrelationSquared  → 0.00724927 }
```

Performance on prediction set

```
{λ, σ} = Power [2. , {-22 , 9.5}];

kk = KernelGaussian  [cm⟦indtrain ⟧, {σ}];
ll = KernelGaussian  [cm⟦indtrain ⟧, cm⟦indpredict ⟧, {σ}];
krr = KernelRidgeRegression  [kk , ae⟦indtrain ⟧, RegularizationStrength  → λ];
Loss [ae⟦indpredict ⟧, krr[ll]]
```

```
{NumberOfSamples  → 6102 , RootMeanSquareError  → 17.6961 , MeanAbsoluteError  → 12.4511 ,
 StandardDeviation  → 17.6912 , MedianAbsoluteError  → 9.5413 ,
 MaximumAbsoluteError  → 203.262 , Correlation → 0.996638 ,
 CorrelationSquared  → 0.993287 , OneMinusCorrelationSquared  → 0.00671322 }
```

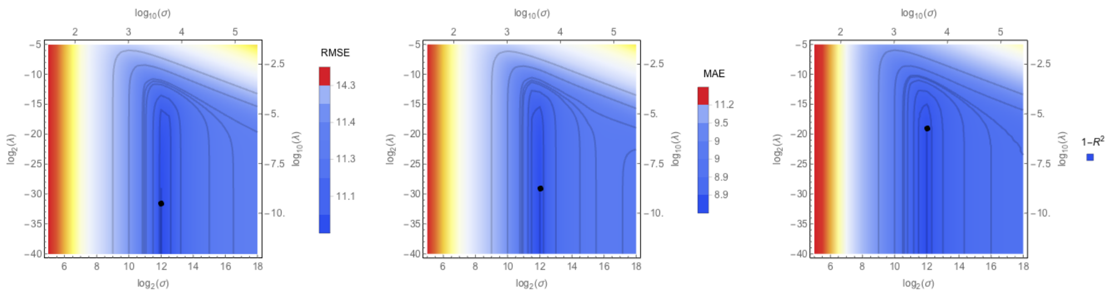
Laplacian kernel

```
Dynamic [Log[2, {λ, σ}]]
```

```
perfL = Table[estperf [λ, σ, KernelLaplacian ], {σ, lsgrid }, {λ, rsgrid }];
```

```
(*
filename =FileNameJoin [{"path","to","file.txt.bz2"}];
Export [filename ,ToString [perfL ,InputForm ],{"BZIP2 ","String "}];
(* Use this line to store results *)
perfL =ToExpression [Import [filename ,{"BZIP2 ","String "}]];
(* Use this line to retrieve previously stored results *)
*)
```

```
GraphicsRow [Table [HyperparameterPlot [rsgrid , lsgrid , perfL , stat],
   {stat , {"RootMeanSquareError ", "MeanAbsoluteError ", "OneMinusCorrelationSquared "}}],
  ImageSize → 18.5 * 72]
```



Optimal hyperparameters for each statistic:

```
Table[t = Position [stat /. perfL , Min[stat /. perfL ], {2}, Heads → False ];
   Join[Log[2, {rsgrid ⟦t⟦1, 2⟧⟧, lsgrid ⟦t⟦1, 1⟧⟧}],
    Log[10, {rsgrid ⟦t⟦1, 2⟧⟧, lsgrid ⟦t⟦1, 1⟧⟧}], Extract [stat /. perfL , t]],
   {stat , {"RootMeanSquareError ", "MeanAbsoluteError ", "OneMinusCorrelationSquared "}}] //
  TraditionalForm
```

$$\begin{pmatrix} -40. & 12. & -12.0412 & 3.61236 & 11.0709 \\ -40. & 12. & -12.0412 & 3.61236 & 8.81136 \\ -40. & 12. & -12.0412 & 3.61236 & 0.00235088 \end{pmatrix}$$

```
estperf [2. ^-40 , 2. ^12 , KernelLaplacian ]
```

```
{NumberOfSamples → 100 , RootMeanSquareError → 11.0709 , MeanAbsoluteError → 8.81136 ,
 StandardDeviation → 11.0002 , MedianAbsoluteError → 6.98926 ,
 MaximumAbsoluteError → 32.9206 , Correlation → 0.998824 ,
 CorrelationSquared → 0.997649 , OneMinusCorrelationSquared → 0.00235088 }
```

Performance on prediction set

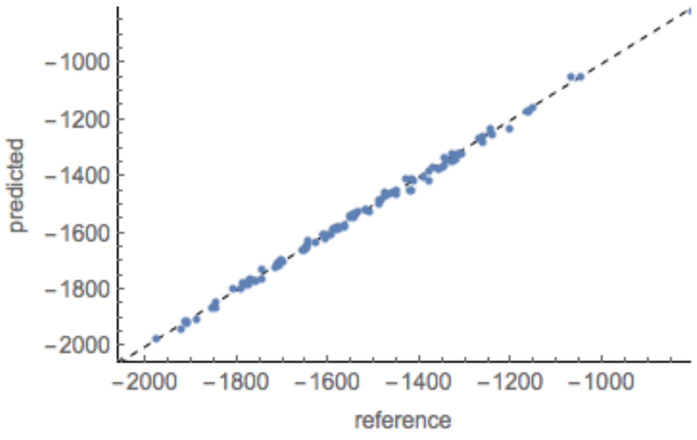
```
{λ, σ} = Power[2., {-40, 12}];

kk = KernelLaplacian [cm⟦indtrain⟧, {σ}];
ll = KernelLaplacian [cm⟦indtrain⟧, cm⟦indpredict⟧, {σ}];
krr = KernelRidgeRegression [kk, ae⟦indtrain⟧, RegularizationStrength → λ];
Loss[ae⟦indpredict⟧, krr[ll]]
```

```
{NumberOfSamples → 6102, RootMeanSquareError → 9.49595, MeanAbsoluteError → 7.08172,
 StandardDeviation → 9.49654, MedianAbsoluteError → 5.60345,
 MaximumAbsoluteError → 80.971, Correlation → 0.999032,
 CorrelationSquared → 0.998066, OneMinusCorrelationSquared → 0.00193449}
```

Scatter plot

```
kk = KernelLaplacian [cm⟦indproper⟧, {σ}];
ll = KernelLaplacian [cm⟦indproper⟧, cm⟦indholdout⟧, {σ}];
krr = KernelRidgeRegression [kk, ae⟦indproper⟧, RegularizationStrength → λ];
ScatterPlot [ae⟦indholdout⟧, krr[ll]]
```


predicted
reference