Home assignment 2

Jan Viilma

April 19, 2018

Contents

1 k-nearest neighbours

K-nearest neighbours algorithm iterates over every data point to find the nearest neighbours. The number of these neighbours is assigned as a hyper-parameter. From these neighbours the majority class is assigned to the data point that is being classified. The algorithm doesn't train a model but relies on the data that it already knows.

In my solution, I used matlabs fisheriris dataset for testing purposes. The input of my example set are datapoints in vector form and corresponding labels. The labels can be either numbers or chars.

Solution examples:

- Prediction of point [1 5.1 3.9 4.6] is versicolor»
- Prediction of point [2.5 1.1 9.9 0.6] is virginica»

To actually confirm the accuracy we would have to look from the datase how close these points are to the assigned classes. Since I only had access to a 4 dimensional dataset, graphing it wasn't an option.

2 Gradient descent

For this problem I used matlab's symbolic functions. First, user has to define a function and its symbolic parameters, assign the appropriate hyperparameters for step size and convergence criteria.

The algorithm finds the derivatives of every variable and then in a while loop calculates the gradient from these derivaives at a concrete point. When the norm of this value is less than the convergence criteria the loop finishes.

Example input:

```
e = 10^(-5);
step = 0.1;
syms X Y Z;

g = Gradient;
g = g.setFunction(sin(X)*cos(Y) + Z^2 + 12, [X Y Z]);
g = g.descent(e, step);
```

Example output:

```
f(1.0000  1.0000  1.0000) = 13.4546;   Change [-0.0292   0.0708  -0.2000]
f(0.9708  1.0708  0.8000) = 13.0357;   Change [-0.0271   0.0724  -0.1600]
f(0.9437  1.1432  0.6400) = 12.7454;   Change [-0.0243   0.0737  -0.1280]
.
.
.
f(1.5708  3.1416  0.0000) = 11.0000;   Change [0.0000   0.0000  -0.0000]
f(1.5708  3.1416  0.0000) = 11.0000;   Change [0.0000   0.0000  -0.0000]
f(1.5708  3.1416  0.0000) = 11.0000;   Change [0.0000   0.0000  -0.0000]
```

3 Levenberg-Marqurdt algorithm

Since this the description of the excercise didn't ask to create my own LM algorithm, matlabs nonlinear least-square solver was used with LM algorithm.

The solution of this excercise is made in such a way that the user can create the model thats going to be used with a random number of layers and neurons.

Example of creating a model with 2 layers, each with 2 neurons:

```
model = lm;
model = model.addLayer(2);
model = model.addLayer(2);
```

After that we create the weights matrix for this model:

```
model = model.getWeights();
```

Then we have to fit the model with some training data that the LM algorithm uses:

```
testIn =  [1 2 3 4 5 6];
testOut = [1 4 9 16 25 36];
model = model.fit(testIn, testOut);
```

The algorithm then trains the weights:

```
                                          First-Order                        Norm of
Iteration  Func-count     Residual         optimality      Lambda               step
    0           9              203               154         0.01
    1          18          138.157                12        0.001           0.189205
    2          27          137.846             0.046       0.0001           0.011671
    3          36          137.846            0.0211        1e-05           0.015444
```

We can now use the model to predict a random variable 12 for example:

```
model.use(12)

  58.152613048143756
```

Which doesn't seem like the correct value with our training set but since the training set is very small, its acceptable.

4 Decision tree classifier

This excercise was the most time consuming compared to the other ones and in my opinion shoul be made easier. My implementation uses a set of vectors and corresponding classes for training. For testing I used matlab's ionosphere dataset. For my decision tree the assigned classes has to be a cell array with string values. When building the tree my algorithm iteratively uses gini impurity and information gain for splitting the dataset and assigning the best questions for the decision nodes. When the information gain is zero we can safely assume that we have arrived at a leaf, the final node which assigns a class.

The tree creation part from the outside looks rather simple:

```
dt = DecisionTree;
dt = dt.fit(X, Y);
```

Where X is an array of data points and Y is an array of corresponding classes.

To find the class of a data point we can classify it like this:

```
>> dt.classify(X(1,:), dt.tree)
    'g'    [2]
```

This uses the trained tree to classify the first data point from the training set, which ofcourse classifies it precisely because it was used for training.