Predicting measurement of roasted coffee color (agtrons) based on data retrieved from roasting curve (.alog file) using simple neural network

*Fundamentals Of Computer Programming - Project by **Oskar Forreiter***

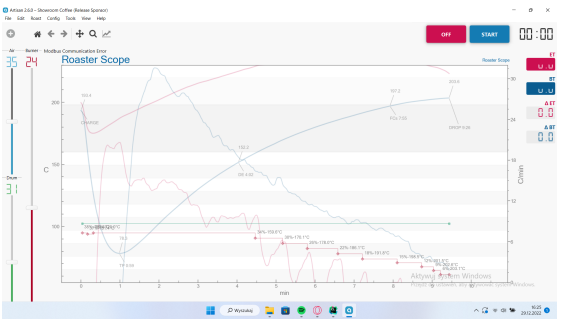
1. Color:

According to a study by Morten Munchow (CoffeeMind Aps, Denmark), **roast color** of coffee is one of the **most important variables** when thinking about the impact of the coffee roasting process on **coffee flavor**. This is the reason why so many coffee roasters are mainly focused on this parameter. Color is predominantly measured in **Agtrons Scale** using **colorimeter**. Colorimeter calculates a value that for coffee ranges between **130-50 Ag**. Where lower value means darker color and vice versa. When Roasting for **espresso** (medium roast) modern roasteries usually aim between **75-100 Ag**, for **filter** (light roast) usually between **90-120 Ag**.

LightTells Colorimeter measuring light roast (105.2)

2. Roasting Software:

Roasting Process is monitored through software that has access to reading multiple **temperature probes**, and operating on **burner** output power, **drum** rotation(rpm), and **airflow**. It allows for efficient, easy to read and precise control of the coffee roasting process. Most popular softwares are **Artisan** and **Cropster**, former being open-source free to use, latter being subscription based but also more advanced and robust.



Artisan Scope roasting software view

3. Data:

In this project we will focus on roasting data that comes from Artisan software, because this is one that I have access to (data directly from my coffee roastery: https://www.fussyfellas.com/en/fellas-2/).

Artisan's data files are **.alog** type, and the way it's being stored makes it easy to work with, usually certain value is preceded by the attribute that specifies what that value means, see below:

```
'beans': 'gwatemala washed filtr', 'wei
0.0, 'l'], 'density': [0.0, 'g', 1, 'l'
'], 'roastertype': '', 'roastersize': 0
'', 'operator': '', 'organization': '',
, 'lowFC': False, 'lightCut': False, 'd
alse, 'uneven': False, 'tipping': False
 'whole_color': 0, 'ground_color': 106,
```

snippet from .alog file opened in text mode

The data that I will retrieve from this file for learning neural network will contain:
Training set:

1) **"ground_color"** - the value of roast color measured with colorimeter

2) **“DROP_time”** - time of whole roasting process measured in seconds
3) **“DROP_BT”** - temperature of coffee beans at the moment of dropping them from the roaster
4) **“AUC”** - area under curve of beans temperature over time of roasting
5) **“finishphasetime”** - time from ‘first crack’ (moment when coffee beans are being exploded by the inside pressure caused by accumulation of gasses during roasting process, usually around 200C) to the end of the process
6) **“beans”** - imputed by hand into each .alog file information about the coffee in format: “origin, processing method, roast degree”

Test set - exactly the same parameters but **“ground_color”** will be only used to check how precise the predictions made by neural network are.

Choice of these data is based on the fact that I already had them clean and labeled. There are probably some better attributes we could use and that would make the output more precise but we are adapting to the data set that we have.

4. Code:

As the whole code is **well commented** I will just explain the flow of the program without going too much in details:

First I declare a `struct features` that will store data extracted from each .alog file

```
struct features{
   double color;
   double end_time;
   double end_temp;
   double auc;
   double dev_time;
   double origin;
   double processing;
   double roastType;
};
```

Then I create two arrays of features:

```
struct features inputData[100];
struct features testData[100];
```

One for training set (inputData), and one for testing set (testData)

Now we need to fill them with data and this is done by the function

```
int collectData(char in_dir[64],
struct features *data);
```

where `char in_dir[64]` is a string containing name of the directory, and `struct features *data` is a pointer to the struct of data where we want to extract our data from file

Aim of this function is to extract all the data from the given directory, and structure it so that we can feed this data to the algorithm, and at the end also returns numbers of files in the directory.

So we call it like that:

```
numTrainingSets =
collectData("input-data",
inputData);
numTestSets =
collectData("test-data",
testData);
```

After collecting data we proceed to the algorithm by calling this function:

```
double     neuralNetwork(double
lr,double epochs);
```

lr (`0.0039`) is learning rate, and epochs (`10000`) are the numbers of iterations. This function returns a double value that is

a precision after calculating the predictions and error.
We call it like that:

```
double precision =
neuralNetwork(0.0039,10000);
```

Whole Neural Network algorithm is based on one described in this article:
https://towardsdatascience.com/simple-neural-network-implementation-in-c-663f51447547
I've made a few changes such as extending the algorithm to work with **more features**, implemented **min-max normalization**, and made it calculate **predictions** of the test set and then also calculate **precision** of these predictions which is then returned as double by this function.

```
return model_precision;
```

5. Results:

For hyperparameters values:
lr = 0.0039 and epochs = 10000
we get a precision around **7.53 Ag**

```
precision = 7.534789
Process finished with exit code 0
```

as the average of agtrons in the test set is around 97 Ag, from simple calculations we get:
1 - 7.53/97.0 = 0.9224 = **92.24%**
This is the precision of the algorithm. (it's a rough estimate, but for this project is enough)

Some articles cite "***anything greater than 70% is a good precision, while above 90% is really good***", if we take it as our reference point the precision of our neural network is really good, unless…

There are few problems:

a) In a coffee roastery difference of 7.53 Ag is huge. Most roasteries aim for **maximum error** of their roasts around **0.0-2.0 Ag**. Which makes it inapplicable for real life use.

b) predictions made by our algorithm are too uniform which probably means that our data is **noisy** or that features that we picked for model of our network were **weakly correlated** with color change.

```
Actual 91.000000 Ag, Predicted 88.025726 Ag
Actual 91.000000 Ag, Predicted 88.025726 Ag
Actual 92.000000 Ag, Predicted 88.025726 Ag
Actual 92.000000 Ag, Predicted 88.025726 Ag
Actual 93.000000 Ag, Predicted 88.025726 Ag
Actual 93.000000 Ag, Predicted 88.025726 Ag
Actual 93.000000 Ag, Predicted 88.025726 Ag
Actual 93.000000 Ag, Predicted 88.025726 Ag
Actual 106.000000 Ag, Predicted 107.834667 Ag
Actual 109.000000 Ag, Predicted 88.025726 Ag
Actual 85.000000 Ag, Predicted 88.025726 Ag
Actual 91.000000 Ag, Predicted 88.025726 Ag
```

Comparison sample of actual color and color predicted by our algorithm

c) Data set is probably too small

6. Conclusions:

Although algorithm precision is high, final results has 3 main underlying problems:
a) algorithm is not useful for real life application
b) we selected wrong or not enough features for the algorithm to perform well - if we would add features like beans density, beans moisture, and screen size (bean size) we would probably get better results.
c) the data set is too small to give more precise output.

Final note:

While making this program my aim was to better understand mechanisms underlying the coffee roasting process and how its color reacts to some changes. I've also wanted to hone some higher level algorithmic skills to make more use of the time spend on this project, and programming it in C language allows for it due to its lack of AI libraries, so everything included in the program had to be first understood by me then formulated into a flow chart, and only then implemented in code. Even though this algorithm happened to be not so useful in reality, the knowledge I've gained from multiple hours spent during christmas break on understanding, formulating, coding, recoding, cleaning, and throwing my laptop out of the window (that's a joke) is invaluable. Hope that everything included in the project is clearly explained, if not I'm happy to answer any questions, and most importantly I'm happy to hear any constructive criticism regarding this project.