Mathematical Modelling I
Final Course Project

Kristoffer Vinell

2018-02-14

1 Background

Machine Learning has gained a lot of traction across various fields and is able to solve a diverse range of problems. In the following, we are going to look at an application within *fraud detection*, basically by correctly classifying user behaviour as fraudulent or not. We are going to employ a *logistic regression* model to a large data set comprised of credit card transactions. It is up to you to implement the necessary algorithms to make it happen, such as *gradient descent* as well as selecting appropriate features to include in your model.

We are going to use Python for this project, which is the most widely used programming language for Machine Learning applications in the industry today. There are loads of packages that can be used to facilitate the process of modelling that we are going to put to good use during the project.

2 Problem Statement

Out of the data set with hundreds of thousands of rows and dozens of features, your job is to train a logistic regression model to classify credit card transactions as either fraudulent or not. The data set contains historic, labeled data points that already have been marked as either fraudulent (1) or not (0).

To train your model, we are going to split the data set into a *training data set* to be used for finding optimal model parameters as well as a *test data set* to be used for validating the accuracy of your model.

You have the data set as well as a computational environment based on the Python programming language at your disposal. The latter is called a *Jupyter Notebook* that allows you to run Python code in an interactive

manner in any browser without having to set up a Python environment on your own computer.

In the following subsections, we will go through various steps needed to complete the project. You will be asked to implement certain parts of the algorithm in Python as well as answer a few questions along the way.

2.1 Setting up your modelling environment

You should be able to find a link to the interactive notebook in the README file of the following repository:

`https://github.com/kvinell/math-modelling-1-project`

Please note that you should continuously backup your work in case of your browser crashing or similar. You can do so by going to File $\Longrightarrow$ Download as, and choosing some appropriate file format of your liking. Whenever you close the browser window containing your notebook, or the network connection is lost, your work will be lost and a new notebook instance will be started the next time you click the link above.

2.2 Loading and exploring the data

Once you've set up your environment, the first step is to familiarise yourself with the data set. This is known as *data exploration* and is the first step in any Machine Learning project.

1. Open the Jupyter Notebook called `data-exploration.ipynb`.
2. Run the first two cells to load necessary libraries and the entire data set.
3. The following two cells let's you inspect the first few rows and look at the structure of the dataset.

You should now be able to answer a few basic questions regarding the data:

- How many data points (rows) are in the data set?
- How many features? (Remember that there is a class/label column in the data set)
- Are the features numerical or categorical?
- What range of values do the features take on?

Having looked a bit closer at the features in the data set, it is time to take a closer look at the *target variable*, `Class`, the last column. By running the following cells you should be able to get the ratio of the number of fraudulent examples to the number of non-fraudulent ones.

- Is there any *data skew* in the target variable? I.e. does one class (fraud/non-fraud) account for the majority of the examples in the data set?
- What impact do you think this will have on the accuracy of your model?

As a last stage, this notebook extracts all features for display. This could come in handy when you later do the *feature engineering*, i.e. select which features to include in your model. Given that feature engineering is an iterative process, it is important to quickly be able to try out various setups.

2.3 Defining the algorithm

When you've familiarised yourself a bit more with the data set, it is time to actually implement logistic regression. Firstly, open up notebook `train-and-categorize.ipynb`. As you can see, many cells have an overall structure in place but are missing a few actual implementations. Your task is to fill out the gaps to turn this notebook into a fully-implemented logistic regression model. At the end, you should be able to run each cell, one-by-one and top-down, to produce a classification. My recommendation is that you go through the entire notebook before you start, so that you know how your implementations will be used during training and model validation. In the theory, we've mostly worked with scalars, but in this setup we will generalise the computations to hold for matrices and vectors as well.

The first cell includes a few libraries that need to be imported, but in the subsequent cells it is up to you to do the following:

1. Implement the *sigmoid* function:
    - Use the theory material for the definition, but remember that the function should accept a **vector** as argument to the function
    - Hint: use `np.exp` function from the *Numpy* library
2. Choose an appropriate value for the `threshold` variable in the `predict` function.
3. Implement the *cost* function:
    - Note that the y parameter is the actual label of the data point, and $\hat{y}$ is the predicted value

- Hint: use `np.log` function provided by Numpy library, and `mean` function

4. Implement the *gradient* function:

   - Use the theory material for the definition, but remember that the function should accept a feature matrix, a vector of predictions and a vector of the true labels
   - Hint: use transpose function (e.g. `x.T` on a matrix `x`) and `np.dot` from Numpy library

5. Implement the *fit* function:

   - Note that we are adding a column vector of ones to cater for the intercept in the model
   - The fit function will iterate a number of times to find optimal weights (parameters) to your model
   - The value you need to compute is `z` (which is what we then pass to the sigmoid function)
   - Hint: use `np.dot` function from Numpy when computing `z`

As a final stage, there is a function provided for you that can be used to compute the accuracy of your model, `compute_accuracy`.

2.4 Selecting features

You are now ready to start applying the data to your model. Firstly, let's load the data set (CSV file), and extract the features as well as the labels (`Class` column).

Your task is now to select the features that gives you the best accuracy of your model, i.e. the features that gives the best categorisation of fraud vs. non-fraud in the data set. This is the part known as *feature engineering*, where usually a large bulk of the modelling effort is spent for any kind of modelling within Machine Learning.

You are encouraged to try out a variety of features to see how they influence accuracy of the model. There are basically two options that you can use in order to select features from the feature set, either by dropping features you don't want (using `all_features.drop` function) or by selecting the ones that you want to incorporate (using `all_features.loc`). Examples of how these are used can be found in the notebook.

2.5 Training the model

When you have decided on a set of features to include in your model, it is time to train your model on a training data set to find optimal parameters (weights). The training set is taken from the larger data set. Usually about 80% of the data set is used as training set. The rest is dedicated for the test set, which is used to compute the accuracy of your model. The notebook has already provided some code to divide the data set into a training set and a test set.

At this point, you should apply the *fit* function to your training set (features and labels) to find optimal parameters. As you can see, there are two additional arguments to the *fit* function, namely

learning rate, which is the scalar in gradient descent

number of iterations, which determines how many iterations of gradient descent we should run

Experiment with different values for the learning rate and number of iterations. The *fit* function will print out the current value of the *cost* function so that you can follow at what pace the algorithm converges. A few questions to ask yourself:

- How does the learning rate affect convergence?
- How many iterations are needed for convergence for a particular choice of learning rate?

When you've found reasonable values for these parameters, try to modify your choice of features and see how that impacts convergence.

2.6 Computing accuracy of the model

To understand how well your model performs, and especially your choice of features, it is time to compute the accuracy of the model. Firstly, you need to compute the predictions on your test set (given by `predict` function available to you) by using the optimal weights as returned by the *fit* function. Then, you may utilise the `compute_accuracy` that prints the accuracy on the screen. These are enclosed in the last two cells of the notebook.

You've now been able to do one iteration of your model. You can now go back and change parameters such as learning rate and do another few rounds of feature engineering to improve the accuracy of your model.

3 Submitting your work

When you're happy with your model, you can save your notebook as an `ipynb` file and send it to me at `kristoffer.vinell@gmail.com`.