Summarizing the first week, a more advanced example, exercise set 4

Data Analysis and Machine Learning

Jul 3, 2020

Regression examples, from linear regression, via decision trees and various forests to neural networks

The main aim of this project is to study some specific regression problems, starting with the regression algorithms studied in homework set 3 (exercise 2 in particular). We will include decision trees, random forests and eventually boosting methods and neural network with **tensorflow** (feel free however to write your own code).

The case we encounter here is the so-called Ising model for our training data and we will focus on supervised training. We will follow closely the recent article of Mehta et al, arXiv 1803.08823. This article stands out as an excellent review on machine learning (ML) algorithms. The added benefit is that each figure and model presented in this article is accompanied by its jupyter notebook. This means that we can start using these and compare with our own results.

You can also look up the Regression slides for a discussion of the Ising model (scroll down to the end).

Alternatively, you can replace the Ising thorughout the exercises with the nuclear binding energies. The choice is yours. Or if you have other data sets suitable for regression, feel free to use those.

What follows here is however a discussion of the Ising model. The nuclear binding energies were discussed during the lectures.

With the abovementioned configurations we will determine, using first various regression methods, the value of the coupling constant for the energy of the one-dimensional Ising model. We will mainly use **scikit-learn** or **tensorflow** or other Python packages such as **keras** or other.

Feel free to use the notebooks to benchmark your code.

Part a): Producing the data for the one-dimensional Ising model. The model we will employ in our studies is the so-called Ising model. Together with models like the Potts model and similar so-called lattice models, the Ising model has been widely studied in mathematics (in statistics in particular), physics, life science, chemistry and even in the social sciences in order to model social behavior. It is a simple binary value system where the variables of the model (spins often in physics) can take two values only, for example ±1 or 0 and 1. The system exhibits a phase transition in two or higher dimensions and the first person to find the analytical expressions for various expectation values was the Norwegian chemist Lars Onsager (Nobel prize in chemistry) after a tour de force mathematics exercise.

In our discussions here we will stay with a physicist's approach and call the variables for spin. You could replace this with any other type of binary variables, ranging from a two political parties to blue and red spheres. In its simplest form we define the energy of the system as

$$E = -J \sum_{<kl>}^{N} s_k s_l,$$

with $s_k = \pm 1$, N is the total number of spins, J is a coupling constant expressing the strength of the interaction between neighboring spins.

The symbol $< kl >$ indicates that we sum over nearest neighbors only. Notice that for $J > 0$ it is energetically favorable for neighboring spins to be aligned. This feature leads to, at low enough temperatures, a cooperative phenomenon called spontaneous magnetization. That is, through interactions between nearest neighbors, a given magnetic moment can influence the alignment of spins that are separated from the given spin by a macroscopic distance. These long range correlations between spins are associated with a long-range order in which the lattice has a net magnetization in the absence of a magnetic field.

We start by considering the one-dimensional Ising model with nearest neighbor interactions. This model does not exhibit any phase transition.

Consider the 1D Ising model with nearest-neighbor interactions

$$E[\hat{s}] = -J \sum_{j=1}^{N} s_j s_{j+1},$$

on a chain of length N with so-called periodic boundary conditions and $S_j = \pm 1$ Ising spin variables. In one dimension, this model has no phase transition at finite temperature.

In the Python code below we generate, with a coupling coefficient set to $J = 1$, a large number of spin configurations say 10000 as shown in the code below. It means that our data will be a set of $i = 1 \dots n$ points of the form $\{(E[\boldsymbol{s}^i], \boldsymbol{s}^i)\}$. Our task is to find the value of J from the data set using linear regression.

Here is the Python code you need to generate the training data, see also the notebook of Mehta et al.

We can now recast the problem as a linear regression model using our codes from homework set 3 (exercise 2 in particular). The way we are going to build our model mimicks the way we could think of finding say the gravitional constant for the graviational force between two planets. In the absence of any prior knowledge, one sensible choice is the all-to-all Ising model

$$E_{\mathrm{model}}[\boldsymbol{s}^i] = -\sum_{j=1}^{N}\sum_{k=1}^{N} J_{j,k} s_j^i s_k^i.$$

Here i represents a particular spin configuration (one of the possible n configurations we generated with the code above).

This model is uniquely defined by the non-local coupling strengths J_{jk} which we want to learn. The model is linear in $\mathbf{J}$ which makes it possible to use linear regression.

To apply linear regression, we recast this model in the form

$$E_{\mathrm{model}}^i \equiv \mathbf{X}^i \cdot \mathbf{J},$$

where the vectors $\mathbf{X}^i$ represent all two-body interactions $\{s_j^i s_k^i\}_{j,k=1}^N$, and the index i runs over the samples in the data set. To make the analogy complete, we can also represent the dot product by a single index $p = \{j,k\}$, i.e. $\mathbf{X}^i \cdot \mathbf{J} = X_p^i J_p$. Note that the regression model does not include the minus sign, so we expect to learn negative J's.

Part b): Estimating the coupling constant of the one-dimensional Ising model using linear regression. We start with the one-dimensional Ising model and use the data we have generated with $J = 1$ in the previous point.

Use linear regression, Lasso and Ridge regression as done in homework 3. Make an analysis of the guessed coupling constant as function of the hyperparameters λ. You can compare your results with those of Mehta et al.. Make sure it is the 1D data which is used.

Discuss the methods and how they perform in computing the coupling constant J and include a bias-variance analysis using the bootstrap resampling method. Discuss also the mean squared error and the $R2$ score as measures to assess your model.

Give a critical analysis of your results.

You can replace the Ising model data with the nuclear binding energy data.

Part c): Random forests and boosting. Repeat the above analysis but using random forests and boosting (XGboost or normal gradient boosting). You can use the functions in **scikit-learn** for random forests and gradient boosting. For **XGBoost** you need to install it separately. You can still use other functionality in **scikit-Learn**.

Part d): Regression analysis of the one-dimensional Ising model using neural networks. Your aim now is to use either **scikit-learn** or **tensorflow** in order to set up a neural network to find the optimal weights and biases.

Train your network and compare the results with those from your linear regression code and random forests/boosting methods.

You can test your results against a similar code using **scikit-learn** or **tensorflow/keras**.

A useful reference on the back progagation algorithm is Nielsen's book. It is an excellent read.

Finally, give a critical analysis of your results with pros and cons of the various methods.

Checklist for handling data, Machine Learning cheat sheet

This is a short cheat sheet for doing machine learning experiments. It'll discuss data exploration the modelling pipeline and model validation in short with links to external resources for reference

Data Exploration. When modelling with machine learning it's easy to just present your data to a model, while this has the odd chance of working more likely this will give you a very weak model.

The first step of the machine learning process is then to look at your data, and your chosen representation of the data.

Visualization. Depending on your data, if it's sequential, real-continious or image based your data should be visualized to get a reference for the distribution of data. In particular for representations of continious or discreet variables visualizing ditributions can be hugely helpful to determining the style of normalization to apply. In short: look at your data.

Normalizing. After inspecting your data distribution(s) you should consider what standardization techniques to apply. In particular you need to think about if you need the covariance matrix of your data to be unchanged under the normalization. Mean centering doesn't change your covariance matrix, standardization of the variance does.

Also consider if your features are on the same scale. Your model might be in trouble if one feature is on the order 10^0 and a second is on the order 10^3.

Prototyping. It's likely that the first version of your chosen model, or even that the model type or representation might provide unsuitable for the problem at hand. A fundamental part of the process then is exploring representations, and model types that might be suitable. The `keras` API and `scikit-learn` off-the-shelf algorithms are fantastic for this purpose.

Remember to keep your model assumptions in mind The following links are useful https://sebastianraschka.com/Articles/2014_about_feature_scaling.html and https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab0

Model Validation. For a regression problem, supervised classification or unsupervised classification with a set of ground truth labels it is absolutely necessary to split your data in disjoint sets for training and testing prior to any processing. The processing variables should be established from the training set and applied to test and training. This is to estimate if your normalization is sane for "unseen" data.

Hyperparameter tuning. For almost all interesting problems the computational cost of running a gridsearch to find the optimal configuration for the hyperparameters is not feasible, both because it is expensive, but the loss-surface might be glassy excepting one very narrow spike your grid doesn't hit. There are many sophisticated tools for the job, but for most cases the tool for the job has shown to be simple random search. Run N experiments with random hyperparameter configurations and pick the best performing on the validation set.

Performance estimation. For estimating your models performance the tool depends on your application. For regression the R^2 coefficient (explained variance) is commonly used, and for linear regression models you usually want an estimate of coefficient significance (`scipy statsmodels` is a python package good for this). For estimating significance and impact you should have a clear image of the degree of multi-colinearity in your data (does your design matrix have full rank?). For non-linear models of 1d vectors (not sequence or image models) estimating feature importance (the impact of removing one feature on model performance). For image data and convolutional nets visualizing the max activation of the filters is a good way to estimate what your model is doing.

For classification there usually is a tradeoff between number of true and false positives. The f1-score (or depending on your model you might favor precision over recall or vice versa which would need an adjusted f-score) is a good single number-measure. Plotting the ROC (reciever-operator characteristic) curve and estimating it's area under the curve (analogous to accuracy). These curves are plotted per-class basis, and then you can average over them to produce an aggregate performance.

You should also use some statsitical measure for model validation. *k*-fold cross validation is recommended for most cases.

In essence: what the goal of all this is to be able to estimate the generalization performance. The training set is used to estimate the optimal parameters and hyperparameters as well as the distribution of performance under these. For each hyperparameter search you use the test set to make a final estimation of your generalization performance, going back and adjusting parameters based on test performance strongly reduces your certainty of generalization and is strongly discouraged.

Scikit-learn has many off-the shelf measures for model performance and validation, with examples. The following blogs on this may be of interest `https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8` and https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html#sphx-glr-auto-examples-model-selection-plot-roc-crossval-py

Background literature

1. The text of Michael Nielsen is highly recommended, see Nielsen's book. It is an excellent read.

2. The textbook of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer, chapters 3 and 7 are the most relevant ones for the analysis here.

3. Mehta et al, arXiv 1803.08823, *A high-bias, low-variance introduction to Machine Learning for physicists*, ArXiv:1803.08823.

If you wish to read more about the Ising model and statistical physics here are three suggestions.

1. M. Plischke and B. Bergersen, *Equilibrium Statistical Physics*, World Scientific, see chapters 5 and 6.

2. D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge, see chapters 2,3 and 4.

3. M. E. J. Newman and T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford, see chapters 3 and 4.