Machine Learning

Lecture 7

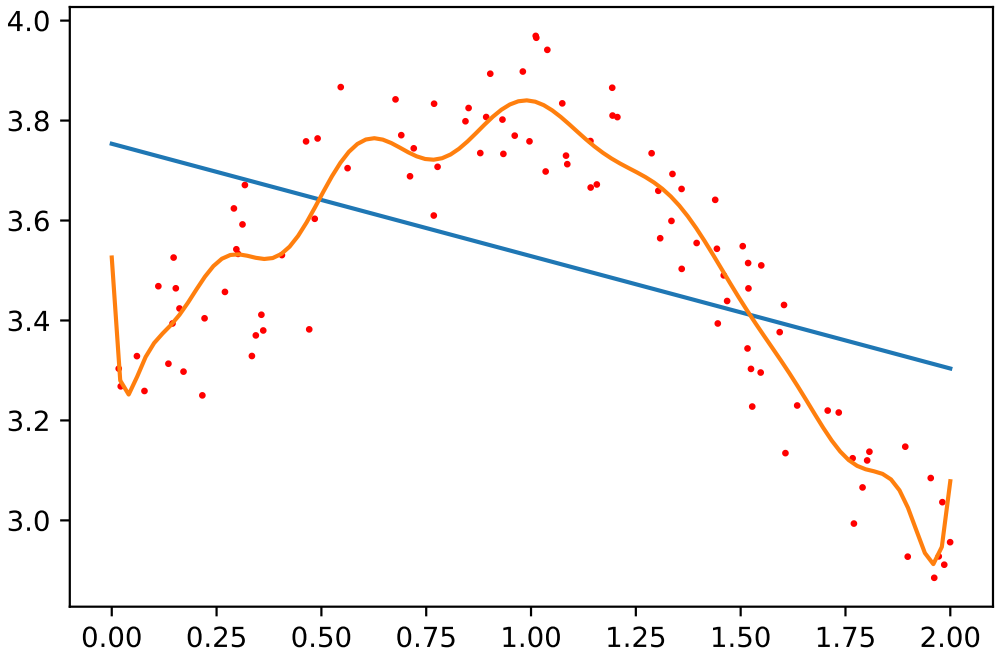
Some feature engineering and Cross validation

Justin Pearson[1]

2020

[1] http://user.it.uu.se/~justin/Teaching/MachineLearning/index.html

Over Fitting vs Bias



The model for the blue line is under fitting the data, or the model is biased towards solutions that will not explain the data. The other model is over-fitting the data. It is trying to model the irregularities in the data.

Epic Python fail

This week I spent hours debugging my demo code, and wondering why it was not adding noise. I had written something like

```
X = np.random.uniform(0,2,number_of_samples)
y = f(X)   + np.random.normal(0,0.1)
```

Instead of

```
X = np.random.uniform(0,2,number_of_samples)
y = f(X) +   np.random.normal(0,0.1,len(X))
```

The idea was to add some random noise to each sample. If you forget the `len(X)` then you add the same random noise to each sample.

Training and Validation Data

What is the goal of machine learning?

- To predict future values of unknown data.

If you are doing statistics, then you could start making assumptions about your data and start proving theorems.
Machine learning is a often a bit different, you cannot always make sensible assumptions about the distribution of your data.

Training and Validation Data

- Ideally we would like to train our algorithm on all the available data and then evaluate the performance of the model on the future unknown data.
- Since we cannot really do this we have to fake it, by splitting our data into two parts: training and test data.

The function

```
sklearn.model_selection.train_test_split
```

is maybe one most important functions that you will use.

Training and Validation Data

- There are lots of reasons to split, but it avoids over-fitting. It avoids learning how to exactly predict how well you learned your training set.
- When you report how well your learning algorithm does, you should report the score on validation set and not the training set.
- You can compare several learning algorithms and compare their validation errors.
- Statistically it is all about reducing variance.

Training and Validation

- You might use different error metrics for the training and validation set. With logistic regression you would train the model by minimising

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} -y \log(\sigma(h_\theta(x))) - (1-y) \log(1 - \sigma(h_\theta(x)))$$

  But you might evaluate the model using accuracy, precision, recall or the F-score from the confusion matrix.

Terminology Warning

In a few slides we will split the data into three parts

- Training, Validation and Test data.

When you split the data into two parts sometimes people write

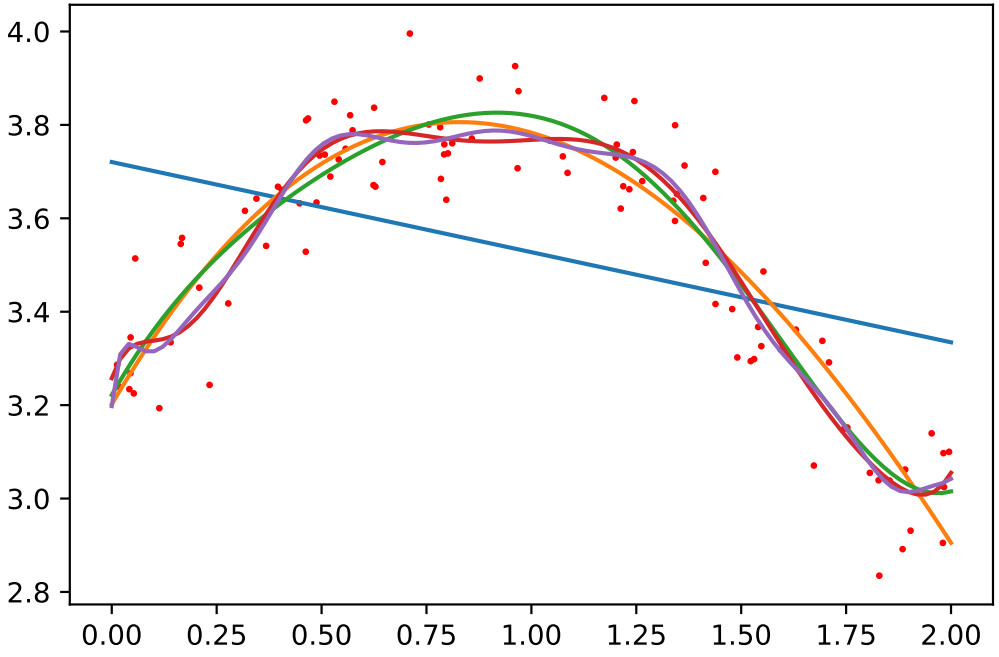
- Training and Test data

and sometimes

- Training and Validation data.

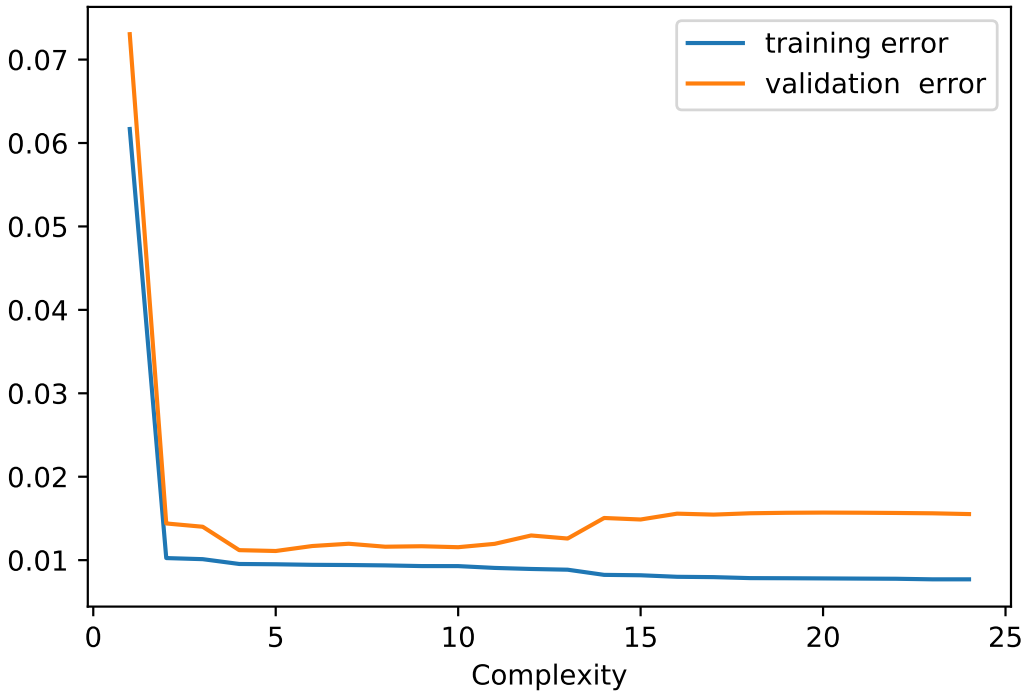
Overfitting vs Bias again

If you have a series of models that get more and more complex, then how do you know when you are over fitting?



Overfitting vs Bias

Assuming that you have split the model into training and validation sets then you can look at training and validations errors as your models get more complicated.



Overfitting vs Bias

- If the test set error and the training set error is very high then you are probably under-fitting.
- When the training error gets smaller and smaller, but your test set error starts increasing you are probably over-fitting.

Overfitting vs Bias

There are lots of problems with this approach including:

- It is not always easy to judge the complexity of your model on a neat straight line.
- What if you picked the wrong division of your data into training and test sets?

Two Goals

Model Selection: estimating the performance of different models in order to chose the best one.

Model assessment: Having chosen a final model, estimate its prediction error on new data.

If we are doing model selection then there is a problem that we might overfit on the validation set.

Train — Validation — Test

If we have enough data then we can split out data into three parts:

Training This is what we use to train our different algorithms. Typical split 50%.

Validation This is what we use to choose our model. We pick the model with the best validation score. Typical split 25%.

Test This is the data that you keep back until you have picked a model. You use this to predict how well you model will do on real data. Typical split 25%.

This avoids overfitting in the model selection. If you are comparing models then you use the validation set to pick the best model, but report the error score on the test set to give an indication on how well the model will generalise.

k-fold cross validation

What if we don't have enough data to split into three parts. Then we can use k-fold validation.

- Split your data randomly into k equal size parts.
- For each part, hold one back as a test set and train on the $k-1$ remaining parts, evaluate on the part you held back.
- Report the average evaluation.

k-fold cross validation

If $k = 5$, then you have 5 parts $T_1, \ldots, T_5$ you would run 5 training runs

- Train on T_1, T_2, T_3, T_4 evaluate on T_5.
- Train on T_1, T_2, T_3, T_5 evaluate on T_4.
- Train on T_1, T_2, T_4, T_5 evaluate on T_3.
- Train on T_1, T_3, T_4, T_5 evaluate on T_2.
- Train on T_2, T_3, T_4, T_5 evaluate on T_1.

Good values of k are 5 or 10. Obviously the larger k is the more time it takes to run the experiments.

A Fold[2]

Sheep near a dry stone sheepfold, one of the oldest types of livestock enclosure

[2] https://commons.wikimedia.org/wiki/File:Sheep_Fold.jpg

k-fold cross validation

What do you do after k-ford cross validation.

- Cross validation only returns a value that is a prediction of how well the model will do on more data.
- Assuming that you sample of the data is randomly drawn (not biased) then there are good statistical reasons why the k-fold valuation is a good idea.
- There are ways of combining an ensemble of models that come from the different folds, such as voting.
- Often we only want one model.

k-fold cross validation

- k-fold cross validation without ensemble methods only tells you which model is better. It does not give you a trained model.
- Once you have decided which model or set of parameters to use, you then train a new model over the whole data set and use that for prediction.
- For example you could test if SVMs and Logistic regression on the same data-set and use k-fold cross validation to decide which model would perform best. Once you know this, you can then retrain on the whole data-set and use this model in production.

Hyper-Parameters and Models

The practical problem for machine learning is how do you pick the right machine learning algorithm or model.
Remember that different model can represent different hypotheses.

- If you hypotheses space is too simple then you have bias or under-fitting.
- If your hypotheses spaces contains hypotheses that can represent complicated decisions then there is a danger that you can over-fit.

Non-linear search spaces and other learning parameters

- With for example k-means clustering, the final result you get also depends on the random initial starting points that you pick.
- There might be other learning parameters that affect how well you converge on a solution.
- The architecture of your neural network is very important.

Regularisation

Regularisation is an attempt to stop learning too complex hypotheses. With linear regression and non-logistic regression we modified the cost function J

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

or

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} -y \log(\sigma(h_\theta(x))) - (1-y) \log(1 - \sigma(h_\theta(x))) + \lambda \sum_{i=1}^{n} \theta_i^2$$

Increasing λ forces the optimisation to consider models with small weights.

More features and Kernels

- Support Vector machines without kernels, linear regression and logistic regression can only learn linear hypotheses.
- Embedding your problem via a kernel function into a higher dimensional space to make the problem more linear is one way of making something learnable. For SVMs you have a lot of choice of different kernels and paramters.
- For linear and logistic regression you can try to invent non-linear features.

Hyper parameters and Models

The terminology is a bit unclear but

Hyper-parameters These are parameters to the learning algorithm that do not depend on the data. They are often continuous values such as the regularisation parameter, but not always. Sometimes people refer to the choice of kernel as a hyper parameter.

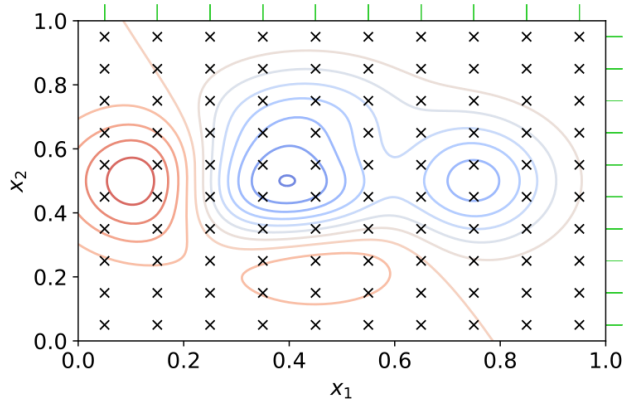
- In a Bayesian framework it is possible to reason about the value of hyper parameters, but it can get quite complicated.
- The main problem with hyper parameters is that it is hard to use the data to optimise the values of the hyper-parameters.

Estimating Hyper parameters

- We can obviously use cross-validation or splits of or data. If your parameters are continuous then it might not be clear which values you are going to pick.
- If the values are continuous then you might have to try too many experiments.

Estimating Hyper parameters — Grid Search[3]

Very simple idea choose some step size and divide you continuous parameters into a grid. Go through all the combinations and return the parameters that minimise the training error.



With a split into training and validation sets you can find close to optimal values for your hyper-parameters. Of course you will need to combine this with cross-validation to get something meaningful if you are comparing different models.

[3] https://de.wikipedia.org/wiki/Datei:Hyperparameter_Optimization_using_Grid_Search.svg

Some feature engineering — One-Hot Encoding

Remember Categorical data is data that can take on a number of discrete values. For example if your data contains the type of car somebody drives:

- Audi
- Volvo
- Saab

You could pick some coding where you just assign a natural number to the each type of car

- Audi $= 0$
- Volvo $= 1$
- Saab $= 2$

One-Hot Encoding

- But what is special about the values 0,1 and 2? If you where trying to do some sort of regression then learning a weight that made sense.
- If x_c is your variable for your car type, then what sense does

$$h_\theta(x_c, \ldots) = \theta_c x_c + \ldots$$

even make?

One-Hot Encoding

Instead we use binary 0/1 variables to represent the categorical variables. In our example we would have three variables

- x_a equals 1 if the car is an Audi and 0 otherwise
- x_v equals 1 if the car is an Volvo and 0 otherwise
- x_s equals 1 if the car is an Saab and 0 otherwise

Note that Scikit Learn has functions to do this automatically for you.

One-Hot Encoding

With our binary variables our models are easier to learn

$$h_\theta(x_a, x_v, x_s, \ldots) = \theta_a x_a + \theta_v x_v + \theta x_s$$

Boosting for feature selection of linear models

Boosting is a general framework, and it can also be combined with cross-validation in a technique called bagging (bootstrap aggregating).
The idea is very simple, learn you model one feature at time, at each stage pick the next feature that gives you optimal performance.
You order the features in order of importance and this gives you models that are easier to interpret for humans.

Boosting for feature selection of linear models

Don't forget to scale your data, so that all dimensions have roughly the same range.
For a linear model you are trying to learn some linear hypothesis

$$h_\theta = \theta_0 + \theta_1 x_1 + \ldots + \theta_n$$

Boosting Stage 0

At stage 0 just learn the model

$$h_\theta^0 = \theta_0$$

This will be a terrible model.

Boosting stage i

You have learnt a model

$$h^i_\theta = \theta_0 + \theta_1 x_{j_1} + \theta_2 x_{i_2} + \cdots + \theta_i x_{j_i}$$

Note that the order $x_{j_1}, x_{i_2}, \ldots x_{j_i}$ is not necessarily $x_1, \ldots, x_i$.
Try all the remain unused variables $x_{j_{i+1}}, \ldots, x_n$ and learn all the $n - i$ models. Pick the variable and θ_{i+1} value that gives the lowest error.
Repeat while the cost goes down.

Boosting for Linear Models

Advantages

- You order the variables in terms of importance.
- There is the possibility to stop early when the model does not improve. This is a way of selecting a subset of the features.
- Later we will see other techniques such as principle component analysis (PCA) that allows you to pick subsets of the features that are important.