Week 4 : Feature Engineering & Model Validation

Ramzi Saouma

June 21, 2020

In the previous weeks, we looked at a considerable number of algorithms across different categories of machine learning. This week we will focus on the work that we do as data scientists prior and after the model estimation. That is, first engineering the features before they go into the algorithm and the eventual model validation and evaluation.

Outline: Learning Objectives

1. Feature Engineering
2. Cross Validation
3. Grid Search
4. Model Evaluation and Improvement

Feature Engineering

So far we have learned some of most the basic ideas of machine learning, While we have not yet worked with real data, we have always assumed that the features are stored in data table ready to be used. In reality, data rarely comes organized, clean and ready to be consumed. In fact, one important step in data analysis is the art and science of feature engineering. In other words how to turn the information you collect about your problem and turning it into table of data that you can be used to build your feature matrix. In today's session we will present some technique such as: designing features for representing categorical data, features for representing text, and features for representing images. Moreover, we will discuss derived features for increasing model complexity and dealing with missing data. Some books call this process vectorization [1], as it involves converting arbitrary data into well-behaved vectors.

[1] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

Categorical Features

A very common type of non-numerical data is categorical data. For example, say you are exploring some housing data that comes with prices, but along with numerical features like "price" and number of

"rooms," you also have "neighborhood" information. For example, your data might look something like this:

```
In[1]: data = [
           {'price': 850000, 'rooms': 4, 'neighborhood': 'Queen Anne'},
           {'price': 700000, 'rooms': 3, 'neighborhood': 'Fremont'},
           {'price': 650000, 'rooms': 3, 'neighborhood': 'Wallingford'},
           {'price': 600000, 'rooms': 2, 'neighborhood': 'Fremont'}
       ]
```

Figure 1: Dictionary of data in python. Source- Muller2017

Our first intuition might suggest to transform the non-numerical data into a numerical by mapping each ""neighborhood" into an integer such as

- 'Queen Anne' -> 1
- 'Fermont' -> 2
- 'Wallingford'->3

```
In[2]: {'Queen Anne': 1, 'Fremont': 2, 'Wallingford': 3};
```

Unfortunately this is not best approach when using python SciKit: 'the package's models make the fundamental assumption that numerical features reflect algebraic quantities. Thus such a mapping would imply, for example, that Queen Anne < Fremont < Wallingford, or even that Wallingford - Queen Anne = Fremont, which does not make much sense.' [2] Instead we use a famous technique known as one-hot encoding[3], which effectively creates extra columns indicating the presence or absence of a category with a value of 1 or 0, respectively. When your data comes as a list of dictionaries, Scikit-Learn's DictVectorizer will do this for you. The neighborhood column will be replaced by a number of columns each referring to one neighborhood with values 0 or 1.

[2] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

[3] One-hot-encoding is a representation of categorical variables as binary vectors.This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

Text Features

Another case where feature engineering is needed is to convert text to a set of representative numerical values. For example, most automatic mining of social media data relies on some form of encoding the text as numbers. One of the simplest methods of encoding data is by word counts: you take each snippet of text, count the occurrences of each word within it, and put the results in a table. For example, consider the following set of three phrases:

```
In[6]: sample = ['problem of evil',
                 'evil queen',
                 'horizon problem']
```

In order to victorize this data, we create a column representing each as below.

```
Out[8]:    evil  horizon  of  problem  queen
        0     1        0   1        1      0
        1     1        0   0        0      1
        2     0        1   0        1      0
```

There are some issues with this approach, however: the raw word counts lead to features that put too much weight on words that appear very frequently, and this can be suboptimal in some classification algorithms. One approach to fix this is known as term frequency–inverse document frequency (TF–IDF), which weights the word counts by a measure of how often they appear in the documents.

```
Out[9]:        evil   horizon        of   problem     queen
        0  0.517856  0.000000  0.680919  0.517856  0.000000
        1  0.605349  0.000000  0.000000  0.000000  0.795961
        2  0.000000  0.795961  0.000000  0.605349  0.000000
```

Image Features

It is increasingly becoming a common need to encode images for machine learning analysis. The simplest approach is using the pixel values themselves of each image. But depending on the application, such approaches may not be optimal. Unless we have students in class who opt for term projects that require imaging data we would not dwell on such features during this course.

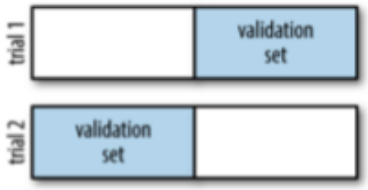
Model Validation

Holdout Sets

One common and simple way to evaluate our model is by using the holdout set technique. That is we hold back some subset of the data from the training of the model. Then once the parameters of the model are estimated to apply it to the holdout set and evaluate performance. In Python, SciKit-learn has a build functionality to do that.

Cross-Validation

One drawback of using a holdout set for model validation is that we have giving up information from a large portion of our data to the model training. I This is not optimal, and can cause problems—especially if the initial set of training data is small. One way to address this is to use cross-validation—that is, to do a sequence of fits where each subset of the data is used both as a training set and as a validation set. Visually, it might look something like in the figure fig:



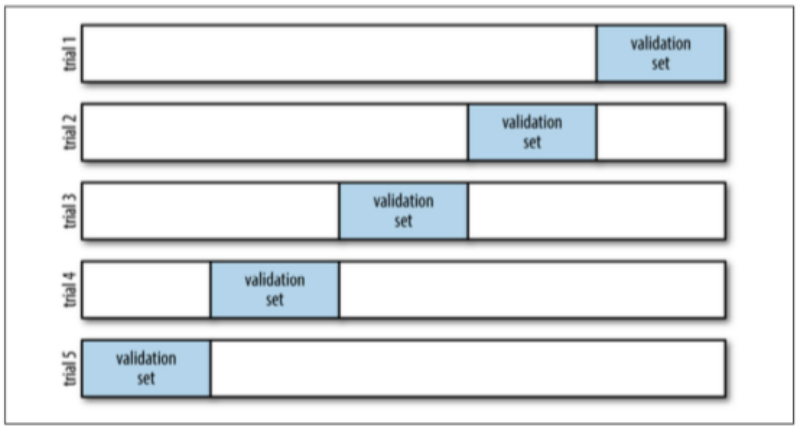


Figure 2: Visualising a five-fold cross validation

Naturally we could take this one step further and instead of creating two subsets of data, we could opt for 5 or more cross-validations

(see figure 2).

Model Evaluation

Now that we've seen the basics of validation and cross-validation, we will go into a little more depth regarding model selection and selection of parameters. Making those evaluations and decision is what defines the success or the failure of a machine learning practitioner.

If the model and its estimators are underperforming, what are the steps to follow? There are several possible answers:

- Opt for more complicated/more flexible model
- Opt for less complicated/less flexible model
- Increase the training samples size
- Hunt for more data to add features

As we have already discussed in class there is always a trade-off in that answer. We know that complicated model could give worse results, and adding more training samples may not reflect any improvement in performance. In more technical terms the question of what is the best fit comes down to finding an optimal point in the trade-off between bias and variance.(see figure 3)

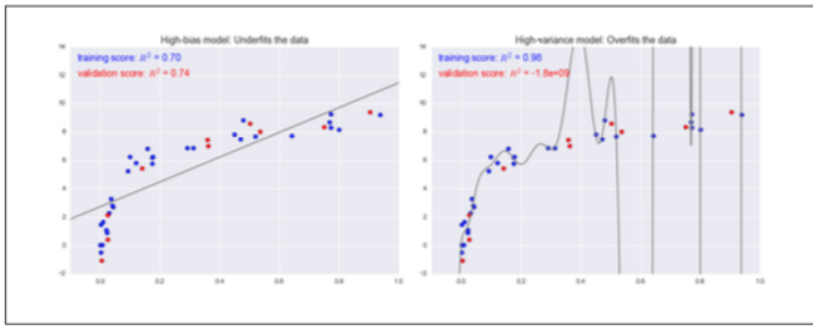


Figure 3: High bias versus high variance models

On left hand side we fit a straight-line through the data. Fairly straightforward, simple and not flexible. Because the data appersto be intrinsically more complicated than a straight line, the straight-line model will never be able to describe this dataset well. "Such a model is said to underfit the data, that is, it does not have enough model flexibility to suitably account for all the features in the data. Another way of saying this is that the model has high bias. The model on the right attempts to fit a high-order polynomial through the data. Here the model fit has enough flexibility to nearly perfectly account for the

fine features in the data, but even though it very accurately describes the training data, its precise form seems to be more reflective of the particular noise properties of the data rather than the intrinsic properties of whatever process generated that data. Such a model is said to overfit the data; that is, it has so much model flexibility that the model ends up accounting for random errors as well as the underlying data distribution. Another way of saying this is that the model has high variance." [4]

[4] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

Now if you observe the red dots on the figure 3, which represent data from the holdout or validation set, we can notice the following when we compare the R^2:

- The linear model with high bias, the performance on the validation set is in line to the performance on the training set.
- The polynomial model with high-variance, the performance of the on the validation set is far worse than the performance on the training set.

If have the freedom to tune bias against variance in our model we expect a relationship that can best be described in figure 4

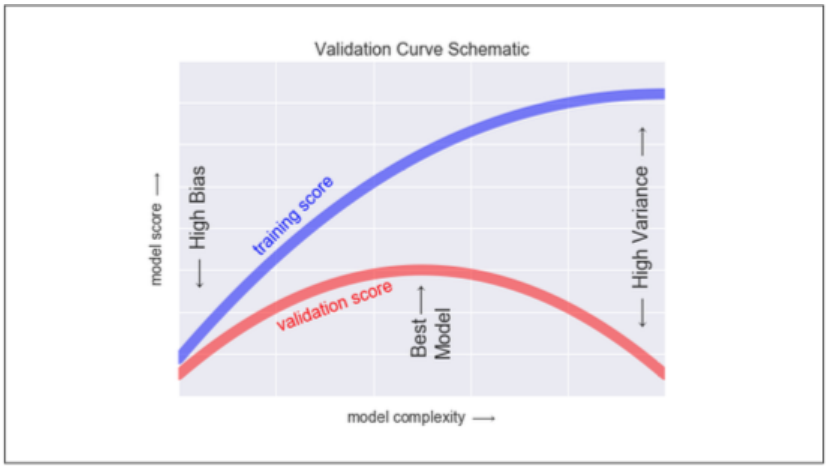


Figure 4: Optimizing the Bias-Variance trade-off

Grid Search

The preceding discussion is meant to give you some intuition into the trade-off between bias and variance, and its dependence on model complexity and training set size. In practice, models generally have more than one knob to turn, and thus plots of validation and learning curves change from lines to multidimensional surfaces. In these cases, such visualizations are difficult and we would rather simply

find the particular model that maximizes the validation score. Scikit-Learn provides automated tools to do this in the grid search module.

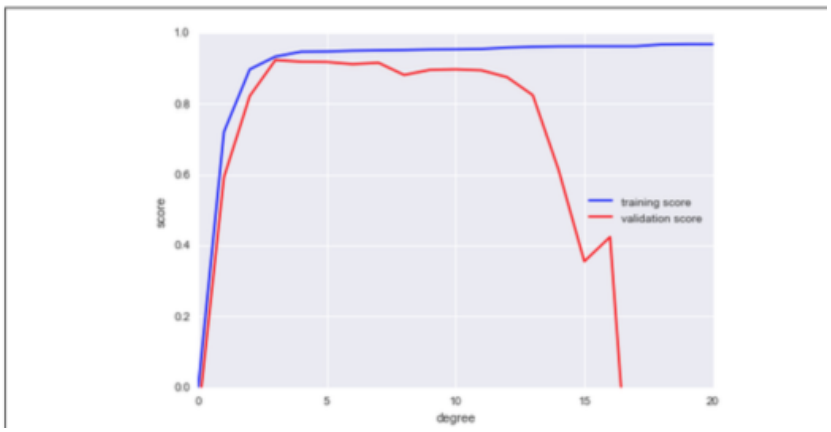


Figure 5: Validation curve help visualize the optimal choice of complexity by varying the complexity parameters such as the degree of a polynomial regression. The results here are consistent with the schematic graph we showed in 4

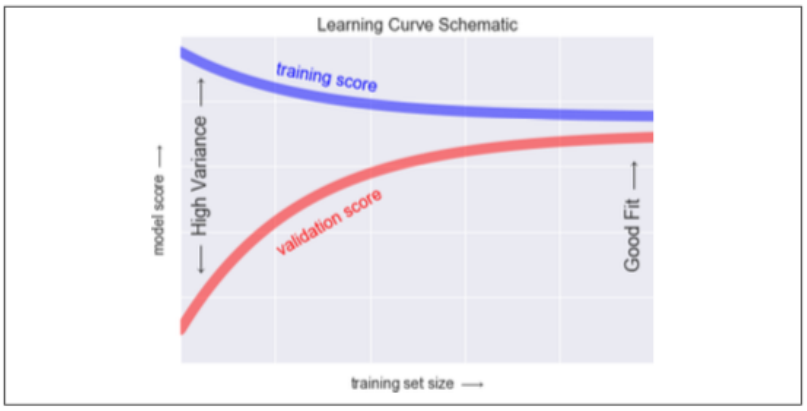


Figure 6: Learning curve help observe the improvement in the model as we increase the data size

References

Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5.