Project T Final

Topic: 9. Training/Testing, Cross-Validation, Bias-Variance
Notes

Sean Betancourt, Erik Fisher, and Jenny Wang

December 2020

Topics seen so far

Week 1: Basic Tools, Data Preprocessing

1. NumPy for Linear Algebra
2. Pandas Tutorial for Data Wrangling
3. SQL
4. Best Practices for Writing/Documenting code
5. Matplotlib and Seaborn for Visualization

Week 2: Linear models, Cross-Validation,
and Scikit Learn for...

6. Linear Regression
7. Perceptrons
8. SVM
9. **Training/Testing, Cross-Validation, Bias-Variance**
10. Implementing Stochastic Gradient Descent (Mini-Batches)

1 Training/Testing

A model is useless if we cannot evaluate how well it performs. The main goal of these notes is to teach you how to evaluate the models you build. These processes are critical to tackling any engineering problem and will be applicable whether you are tuning SVMs from this week, neural networks later in the course, or top-of-the-line hebabibibababaloo networks which will be invented in 2102 (mark my words).

2 Getting to Know Your Dataset

The first step in the engineering process is always to get to know the problem and the dataset. Things you should ask yourself include:

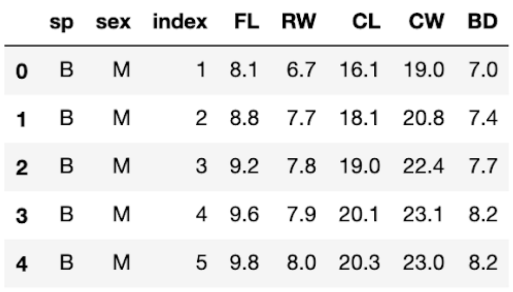
- Are the features/columns intuitively relevant to the problem?
- Do relationships seem to exist among the features?
- Is there too much noise in the data?

Machine learning models are not magical machines. At the current state of technology, they require tuning and regularization to work with high accuracy. One way to ensure quality during the training and testing process is to sanity check aspects of the dataset to make sure we're doing something that seems truly possible.

2.1 Crab Dataset

Let's take a look at an example and put your Pandas knowledge from last week to good use! We will be pulling data from Kaggle's Crab body metrics public dataset. This section will take you through the thought process of how to get to know your dataset.

```
import pandas as pd
crab_df = pd.read_csv('data.csv')
crab_df.head()
```



	sp	sex	index	FL	RW	CL	CW	BD
0	B	M	1	8.1	6.7	16.1	19.0	7.0
1	B	M	2	8.8	7.7	18.1	20.8	7.4
2	B	M	3	9.2	7.8	19.0	22.4	7.7
3	B	M	4	9.6	7.9	20.1	23.1	8.2
4	B	M	5	9.8	8.0	20.3	23.0	8.2

Figure 1: The first few rows of the Crab body metrics dataset. From the documentation, **sp** is species, **sex** is what it says, **index** is a number for the crab, **FL** is the frontal lobe size (mm), **RW** is the rear width (mm), **CL** is the carapace length (mm), **CW** is the carapace length (mm), **CW** is the carapasce width (mm), and **BD** is the body depth (mm).

The columns look like a plethora of relevant information! Can we predict the frontal lobe size of a crab given other body metrics? Let's dig a bit deeper.

```
ax1 = crab_df.plot.scatter(x='FL', y='CL', marker='o')
ax1.set_xlabel("Frontal Lobe Size (mm)")
ax1.set_ylabel("Carapace Length (mm)")

ax2 = crab_df.plot.scatter(x='RW', y='CW', marker='o')
ax2.set_xlabel("Rear Width (mm)")
ax2.set_ylabel("Carapace Width (mm)")
```

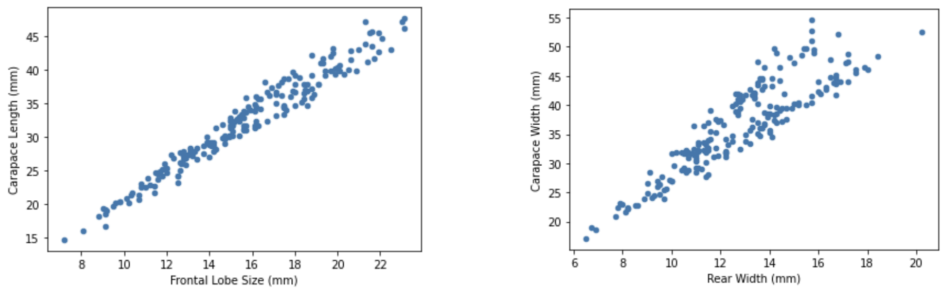


Figure 2: Does one variable strongly predict the other? (trick question!)

At first glance, there seem to be strong correlations between these random features we plotted, which makes sense because a crab with a larger width should have a longer length, etc. However, the right hand plot in Figure 2 shows two lines branching off from each other. Why is that? If we dig and separate our data a bit more, we can see that one line represents male crabs and another line represents female crabs. A cleaned dataframe is in Figure 3; it does not include male crabs.

```
crab_f_df = crab_df[crab_df['sex'] == 'F']
# Then replot the crab dataframe
```

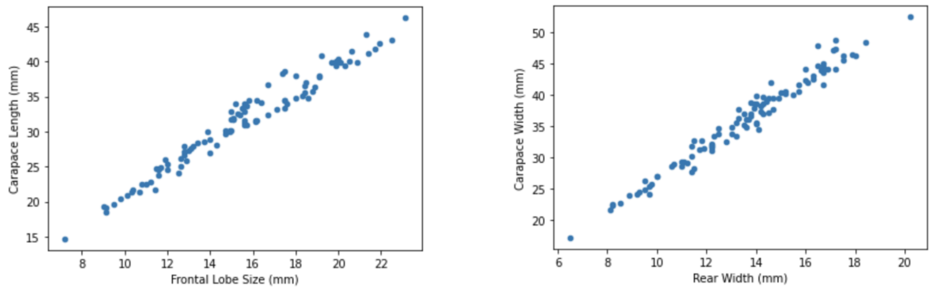


Figure 3: Cleaned out the dataframe! Now there is less "noise" for what we will be predicting

Great! So now we know there is a way to clean out the "noise" from our dataframe by conditioning on the column for sex. A linear model can probably do well in prediction, so let's whip it up and give it a shot!

3 Cross Validation

Previously, you have seen that we need to set aside a portion of our data for testing specifically to see how well we are training our model. This is to make sure that our model is learning the overall pattern of the data and not some specific occurrences that only happened in one portion of the data. The important idea is that the test set cannot be used to tune our model–we should not modify our model based on test accuracy or else test accuracy would not be representative of the model's generalization strength anymore. This is often called "data incest."

Thus, we introduce the notion of the validation set. Let's first review some some ways we can measure a model's success

3.1 Measures of Success (for regression problems)

Let's say we have a Support Vector Regression (SVR; a variant of SVMs from the previous lecture) model initialized with default parameters. How do you know the model does well for this problem?

Here are some common metrics:

- Correlation (but this metric does not work for nonlinear models)
- Mean Squared Error (MSE), which averages the squared prediction error. MSE = $\frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y_j})^2$
- Root Mean Squared Log Error (RMSLE), which is the square root of the average log prediction error. RMSLE = $\sqrt{\frac{1}{N}\sum_{j=1}^{N}(log(y_j) - log(\hat{y_j}))^2}$
- and more...

MSE is used commonly in practice, and is a great measure of empirical risk.

3.2 Creating a Validation Set

You have probably seen the train-test split of datasets when trying out various models with sklearn. Now, let's do a train-validation-test split of a dataset in order to accomodate the procedure to obtain validation accuracy. This allows us to tune the structure of the model, itself, in addition to the tuning of the weights using the train set. This can follow the same pattern as the train-test split:

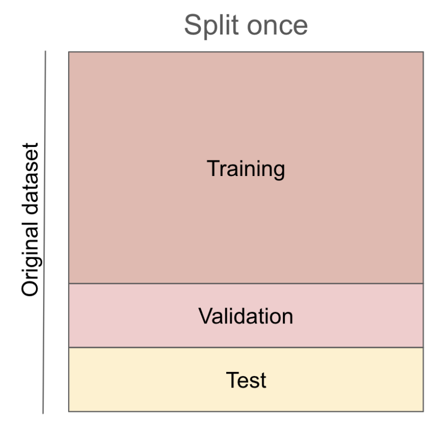


Figure 4: Splitting the dataset into three parts: the train set, validation set, and test set

One pro to this strategy is that we are now able to tune the model structure. One con is that the train set is smaller! Luckily, k-fold cross validation provides a way to estimate validation error while essentially using the whole training set throughout training. The procedure is as follows:

- Split the training set into K ”folds” (groups)
- For each of the K splits:
    1. Train with the training data
    2. Compute accuracy with the validation set
- Overall validation accuracy is the average of all K accuracies

This can be visualized in Figure 5.

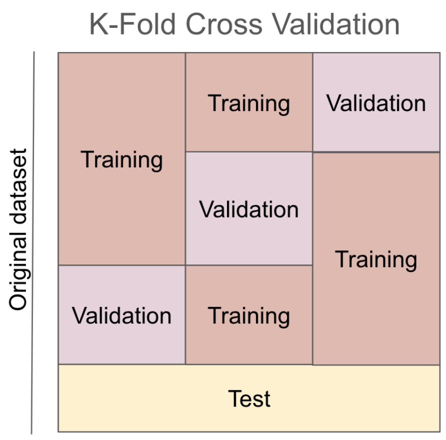


Figure 5: Splitting the dataset into the train set, validation set, and test set with k-fold cross validation

4 Bias and Variance

There is a common phenomenon in machine learning called the Bias Variance tradeoff. In your machine learning travels, you might often see something like that in Figure 6.

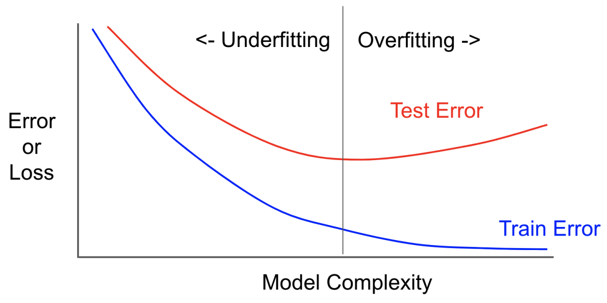


Figure 6: The Bias-variance tradeoff, exhibited as underfitting, overfitting, and a difference in test error and train error.

We see that a less complex model has high train and test error, a moderately complex model has a low train error and a low test error, and a very complex model has a low test error and a high test error. Of course the "complexity" of the model is relative to the problem it tries to solve (a more complex model is required to distinguish between a dog and a cat than one required to distinguish between a circle and a square), but this pattern holds true in many cases.

Why does this happen? A model that is too small does not have enough parameters to represent the relationships and thus relies on assumptions and smoothing that may not be hold true in all cases–this is a model with high **bias**. A model that is too big often has so many parameters that some of them are not necessary to solve the problem; these parameters often latch onto noise in the training set and cause weird behaviors on the test set, where those patterns do not exist–this is a model with high **variance**. The graph in Figure 6 shows that decreasing bias (moving to the right) often increases variance and vice versa. This is the bias-variance tradeoff.

4.1 Bias and Variance in MSE

So far, we've talked about bias and variance in loose terms. Let's try to structure the argument in mathematical notation. This next section is based off of EECS189 Note 5.

Let's define some variables.

- $f(x)$ = the true underlying label
- Z = some zero-mean random variable
- Y = a noisy measurement of the label = $f(x) + Z$
- D = a random variable denoting the training set = $\{(x_i, y_i)\}_{i=1}^n$
- $h(x|D)$ = the model's label prediction for a data point, given that it was trained on training set D. This is a random variable because D is a random variable

Remember the following facts:

- A *random variable* has an unknown value that varies by a probabilistic distribution
- $E[A]$ denotes the expectation of random variable A

Remember that mean squared error (MSE) = $E[(h(x; D) - Y)^2]$. Let's do some expression massaging:

$$
\begin{aligned}
\varepsilon(\mathbf{x}; h) &= \mathbb{E}[(h(\mathbf{x}; \mathcal{D}) - Y)^2] = \mathbb{E}[h(\mathbf{x}; \mathcal{D})^2] + \mathbb{E}[Y^2] - 2\mathbb{E}[h(\mathbf{x}; \mathcal{D}) \cdot Y] \\
&= \Big(\mathrm{Var}(h(\mathbf{x}; \mathcal{D})) + \mathbb{E}[h(\mathbf{x}; \mathcal{D})]^2\Big) + \Big(\mathrm{Var}(Y) + \mathbb{E}[Y]^2\Big) - 2\mathbb{E}[h(\mathbf{x}; \mathcal{D})] \cdot \mathbb{E}[Y] \\
&= \Big(\mathbb{E}[h(\mathbf{x}; \mathcal{D})]^2 - 2\mathbb{E}[h(\mathbf{x}; \mathcal{D})] \cdot \mathbb{E}[Y] + \mathbb{E}[Y]^2\Big) + \mathrm{Var}(h(\mathbf{x}; \mathcal{D})) + \mathrm{Var}(Y) \\
&= \Big(\mathbb{E}[h(\mathbf{x}; \mathcal{D})] - \mathbb{E}[Y]\Big)^2 + \mathrm{Var}(h(\mathbf{x}; \mathcal{D})) + \mathrm{Var}(Y) \\
&= \underbrace{\Big(\mathbb{E}[h(\mathbf{x}; \mathcal{D})] - f(\mathbf{x})\Big)^2}_{bias^2 \text{ of method}} + \underbrace{\mathrm{Var}(h(\mathbf{x}; \mathcal{D}))}_{\text{variance of method}} + \underbrace{\mathrm{Var}(Z)}_{\text{irreducible error}}
\end{aligned}
$$

Nice! Certain terms in here can be interpreted as the bias and variance of a model, when evaluated against MSE. The underlined variance term can be interpreted as the "variance of the model's predition for x given various training datasets" because D is the random variable in the equation. The underlined bias term can be interpreted as the "squared error between what is expected from model predictions of x and the true labels". Bias and various of for other error metrics are broken down in similar ways, but we will leave that exercise up to you.

Further readings can be found in EECS189 Note 5!

5 Training and Testing Workflow

For your convenience, here is the full training and testing workflow recommended by this note:

1. Visualize and understand the data
2. Split dataset into the train and test sets
3. Formulate a model (ex: OLS)
4. Either
    - Section off part of the train set as a validation set, if there are lots of samples
    - Use cross validation to section off certain parts of the train for validation at a time, if data is limited
5. Calculate validation accuracy or loss. Perhaps also compute the bias and variance of the model given various training datasets to help you understand how to improve the model.
6. Repeat 2-5 until the model does well enough on the validation set
7. Calculate test accuracy or loss