

Improving prediction: Understanding the Loss function

In performing Error Analysis

- we *identified* **test** examples where our model failed to generalize correctly
- but we didn't propose a **solution** to improve generalization

That is the topic of this module.

When we perform Error Analysis

- we focus on the Performance Metric (e.g., Accuracy)
- on a per-example basis
- for **out of sample** examples

But we can't *directly* influence the Performance on an out of sample example.

Instead, we will perform a *Loss Analysis*

- we focus on the Loss
- on a per-example basis
- for **in sample** examples

This is the analog of Error Analysis, but performed on Training rather than Test examples.

Reminder

- the Loss Function and Performance metrics are **not necessarily** the same
- the Loss Function is computed **in-sample**
- the Performance Metric is computed **out of sample**

The hope is

- that improving in-sample Loss
- will *indirectly* lead to a better out of sample Performance Metric

To illustrate, recall how Logistic Regression creates a prediction

- the model computes a score/logit for an example

$$\hat{\mathbf{p}}^{(i)} = \sigma(\Theta^T \cdot \mathbf{x}^{(i)})$$

- the score/logit is converted into a prediction by comparison with a threshold

$$\hat{\mathbf{y}}^{(i)} = \begin{cases} \text{Negative} & \text{if } \hat{\mathbf{p}}^{(i)} < 0.5 \\ \text{Positive} & \text{if } \hat{\mathbf{p}}^{(i)} \geq 0.5 \end{cases}$$

The Loss is Binary Cross Entropy evaluated on the probabilities $\hat{\mathbf{p}}^{(i)}$

As we observed [before \(Classification Loss Function.ipynb#Classification:-Loss-function\)](#).

- a small change in $\hat{\mathbf{p}}^{(i)}$
- does **not necessarily** change prediction $\mathbf{y}^{(i)}$
- unless $\hat{\mathbf{p}}^{(i)}$ crosses the threshold

Thus

- the Loss varies continuously with changes in parameters Θ
- but the Performance **may not**

Loss Analysis

- examines the per-example Loss using in-sample examples
- in order to finding common attributes of problematic (mis-predicted) in-sample examples

Once we *diagnose* the problem with Loss

- we can explore remedies
 - feature engineering
 - pre-processing
- with the goal of causing the optimizer to change Θ
- in order to push $\hat{\mathbf{p}}^{(i)}$ in the right direction

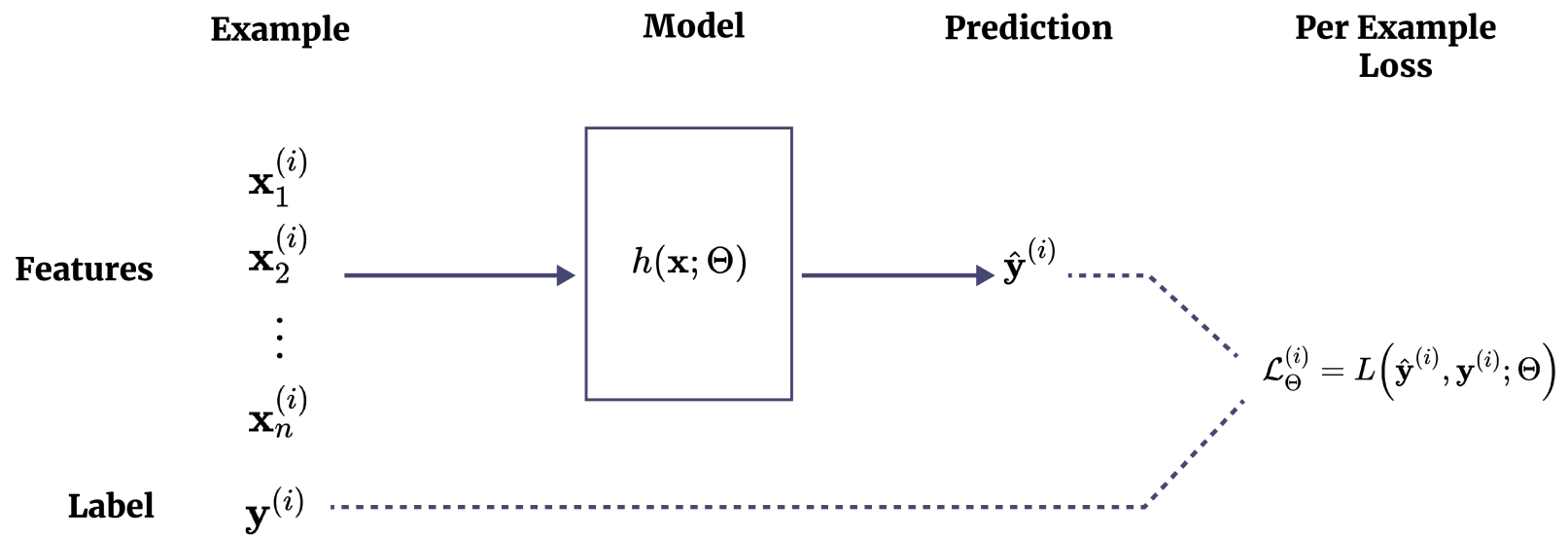
Recall the basics of minimizing Loss Functions

- Predictions $h(\mathbf{x}; \Theta)$ are a function of both inputs and parameters Θ
- A given Θ induces a per-example loss $\mathcal{L}_{\Theta}^{(i)}$
- Average Loss is the average of the per-examples losses $\mathcal{L}_{\Theta}^{(i)}, i = 1, \dots, m$
- We seek the optimal Θ^* :

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \mathcal{L}_{\Theta}$$

In pictures:

Training Example



Training Example

$\mathbf{x}^{(1)}$	$\mathbf{y}^{(1)}$	$\hat{\mathbf{y}}^{(1)}$	$\mathcal{L}_{\Theta}^{(1)}$
$\mathbf{x}^{(2)}$	$\mathbf{y}^{(2)}$	$\hat{\mathbf{y}}^{(2)}$	$\mathcal{L}_{\Theta}^{(2)}$
	\vdots		
$\mathbf{x}^{(i)}$	$\mathbf{y}^{(i)}$	$\hat{\mathbf{y}}^{(i)}$	$\mathcal{L}_{\Theta}^{(i)}$
	\vdots		
$\mathbf{x}^{(m)}$	$\mathbf{y}^{(m)}$	$\hat{\mathbf{y}}^{(m)}$	$\mathcal{L}_{\Theta}^{(m)}$
			<u><u>\mathcal{L}_{Θ}</u></u>
			Total Loss

•

Conditional loss

In Error Analysis we partition test examples into groups with some common property, such as

- Commonality of result: TP, FN, TN, FP
- Commonality of features in order to compute a *conditional* out of sample metric.

In Loss Analysis we partition training examples into groups to in order to compute a *conditional* in sample metric.

The following picture uses colors to identify which group a training example belongs to:

Loss analysis: conditional loss

$$\mathbf{x}^{(1)} \quad \mathbf{y}^{(1)} \quad \hat{\mathbf{y}}^{(1)} \quad \mathcal{L}_{\Theta}^{(1)}$$

$$\mathbf{x}^{(2)} \quad \mathbf{y}^{(2)} \quad \hat{\mathbf{y}}^{(2)} \quad \mathcal{L}_{\Theta}^{(2)}$$

\vdots

$$\mathbf{x}^{(i)} \quad \mathbf{y}^{(i)} \quad \hat{\mathbf{y}}^{(i)} \quad \mathcal{L}_{\Theta}^{(i)}$$

\vdots

$$\mathbf{x}^{(m)} \quad \mathbf{y}^{(m)} \quad \hat{\mathbf{y}}^{(m)} \quad \mathcal{L}_{\Theta}^{(m)}$$

\equiv

$$\mathcal{L}_{\Theta}$$

Total Loss

$$\mathcal{L}_{\Theta}$$

Conditional Loss

$$\mathcal{L}_{\Theta}$$

Conditional Loss

The real advantage of performing Conditional analysis in sample

- In sample examples (training/validation) can be re-used, unlike Test examples
- Added features based on in sample analysis is likely to affect the Loss
 - Unknown whether it will affect Performance Metric (when it is different than Loss, e.g., Accuracy)

What can we do to reduce loss ?

Understanding the per example loss can help you "push" the optimizer toward find a "better" Θ .

We will outline some simple strategies via examples that identify a problem and propose a solution.

Increase number of "problem" training example

For MNIST digit classification

- We hypothesize a commonality that causes images of the digit 8 to be misclassified
 - 8's that are slanted in the "opposite" direction of normal
 - We will refer to this as the *problematic* class

One reason our classifier may fail on this sub-class of 8's

- There are many fewer of them than the more prevalent images of 8's

Mathematically, the Average Loss is equally weighted

$$\mathcal{L}_{\Theta} = \frac{1}{m} \sum_{i=1}^m \mathcal{L}_{\Theta}^{(i)}$$

but the cumulative weight of the problematic class (mis-shaped 8's) is very small.

So even if all examples in the problematic class were mis-classified

- The impact on Average Loss may be sufficiently small.
- That Θ doesn't get updated in the direction that will improve these examples
 - Especially if we end optimization before absolute convergence occurs, as is common

One strategy for pushing the model to better fit the problematic examples is

- Increase their cumulative weight in the Loss
- By increasing their number !

The strategy known as *Data Augmentation* adds examples to the Training examples

- Here we try to find/synthesize more instances of the problematic type

We can augment examples by repeating them (as above).

- re-sampling the Training data
- covered in the module on Imbalanced Data

This is a simple method that works well for most data types.

For some types of data (e.g., Image), other means of augmentation are available.

- Create a new training example
- By perturbing the features of an existing training example
- In such a way as to preserve the label

For instance, give a training example we can

- Add a small quantity of noise to the feature vector
- Perform data-type specific transformations
 - Images: [shift, rotate, transpose](#)
[\(DataAugmentation/Data_augmentation.ipynb#Original-image\)](#)

Influential points

We have described the case where the issue is mis-classification of an important but small sub-class.

- Which results in a small cumulative contribution to the Loss

Sometimes the problem is a small sub-class with an *out-sized* contribution to the Loss

- Having a few problem examples
- Whose contribution to Loss is so large
- That it pushes Θ in the wrong direction for the more numerous non-problem examples

That is: $\mathcal{L}_{\Theta}^{(i)}$ is so large (for some example i) that

- Θ is changed to reduce $\mathcal{L}_{\Theta}^{(i)}$
- Resulting in an increase in $\mathcal{L}_{\Theta}^{(i')}$ for each non-problematic example i'

The phenomenon we just described is sometimes called *Influential Points*.

These have been particularly well-studied in the context of Linear Regression.

We will use Linear Regression as an illustration.

Loosely speaking, an example is **influential** if

- the parameter estimate Θ changes greatly depending on whether the example is included/excluded

Feature values on the extreme ends of the range have greater potential for being influential.

This is one argument for constraining the range of the feature (MinMax, Standardization).

Here's an interactive tool to get a feel for influential points in Linear Regression.

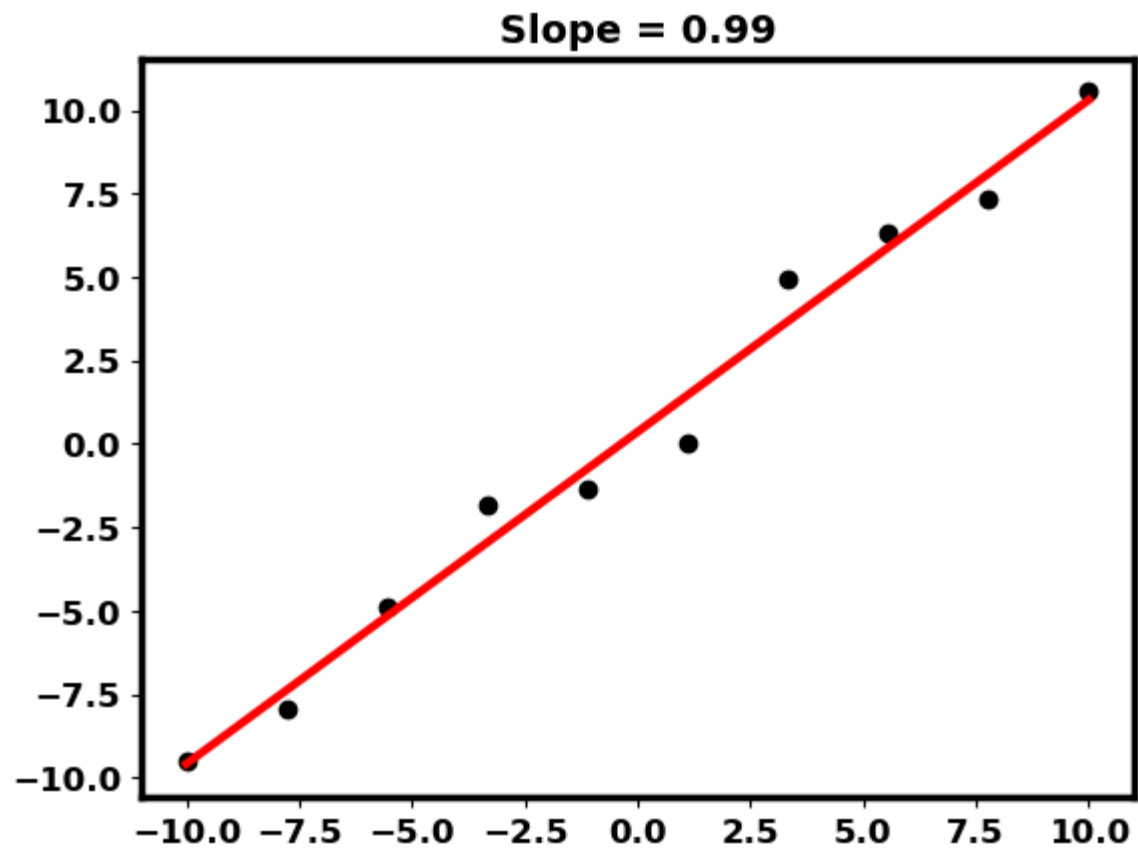
It allows you to change the value of a single data point and see the effect on the fitted line.

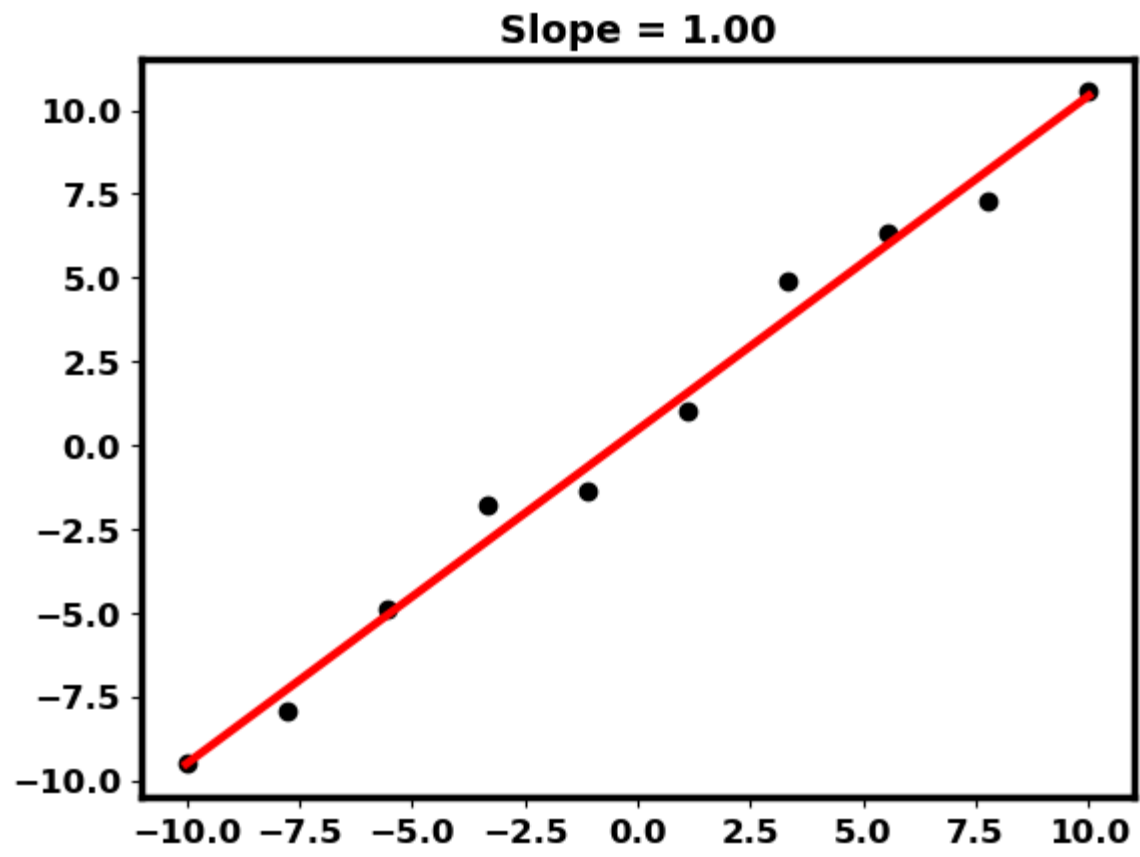
Observe how the slope changes (displayed in the title)

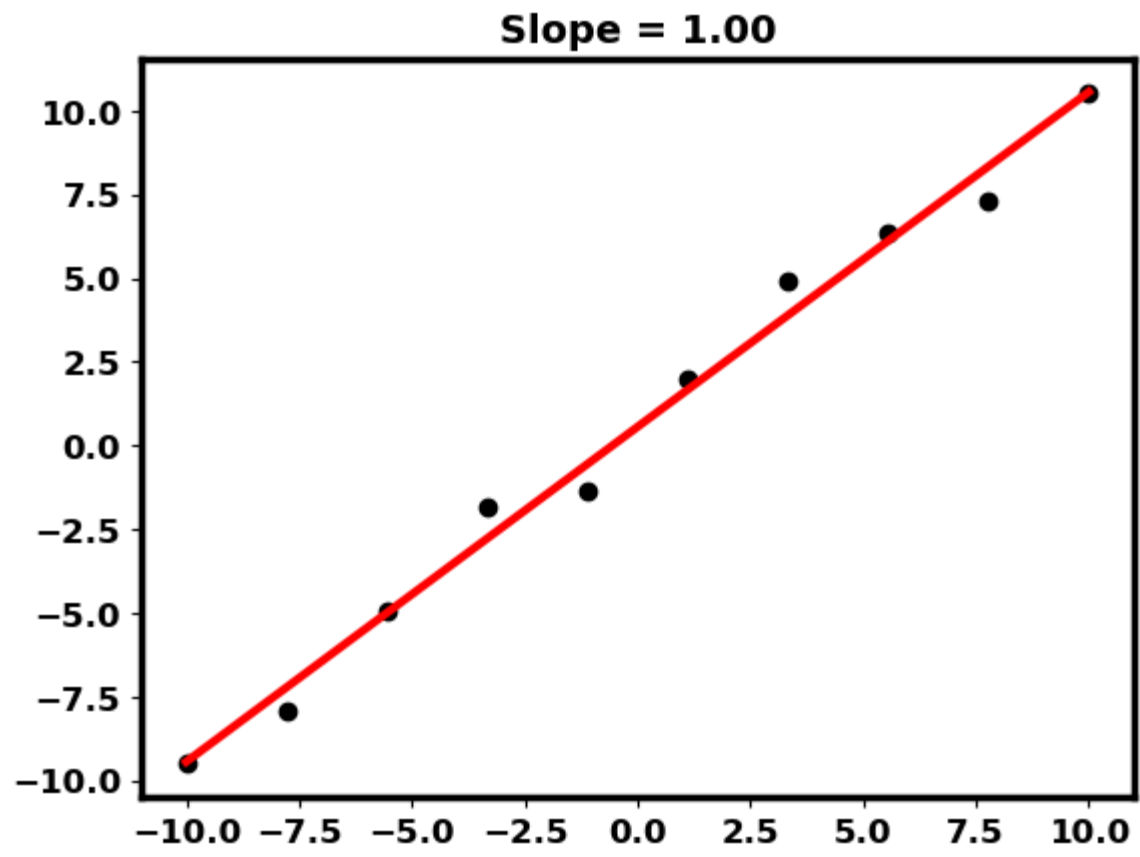
- 10 labeled examples $\{[\mathbf{x}^{(i)}, \mathbf{y}^{(i)}] \mid 0 \leq i < 10\}$
- The top slider chooses the index $i \in \{0 \dots 9\}$ of one data point to change
- The bottom slider is the new value $\mathbf{y}^{(i)}$ for the point at the chosen index i

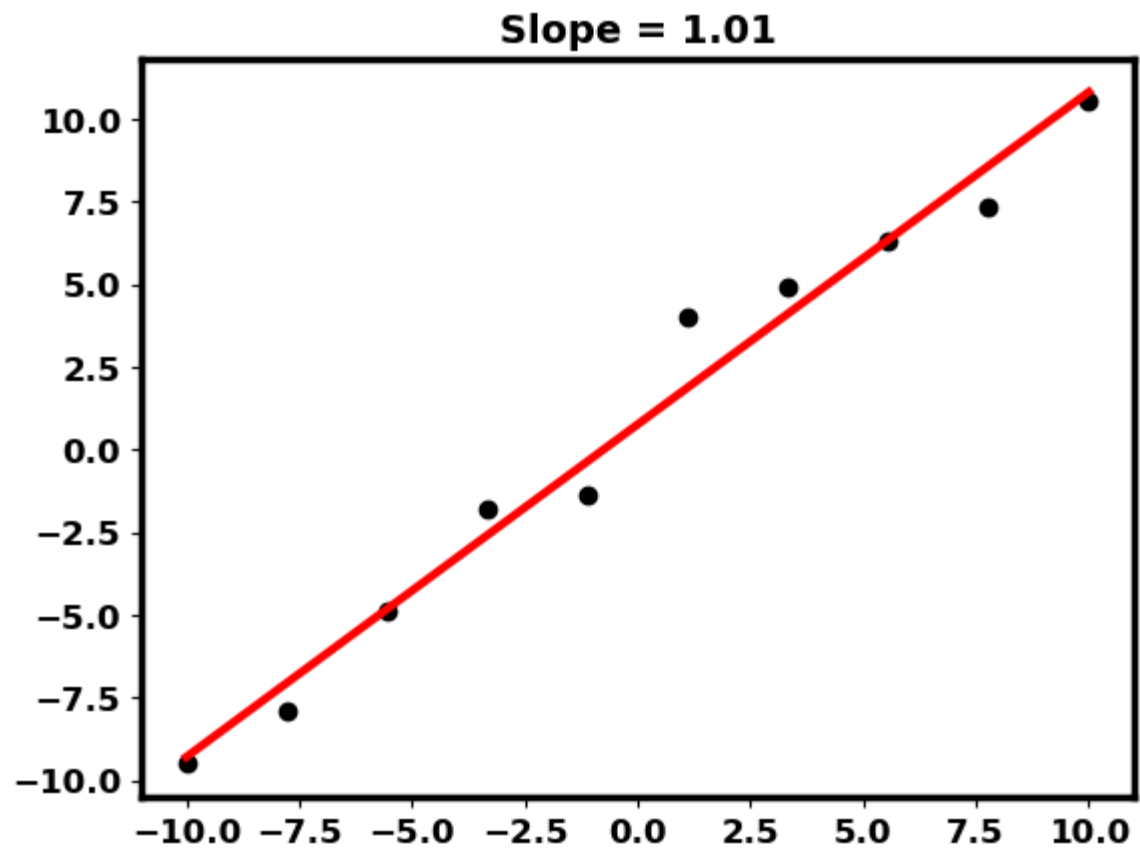
```
In [4]: # Generate some points  
        (x_ip,y_ip) = iph.gen_data(10)  
  
        # Fit a line to the points; get a function to update the fit and the plot  
        fit_update = iph.plot_init()
```

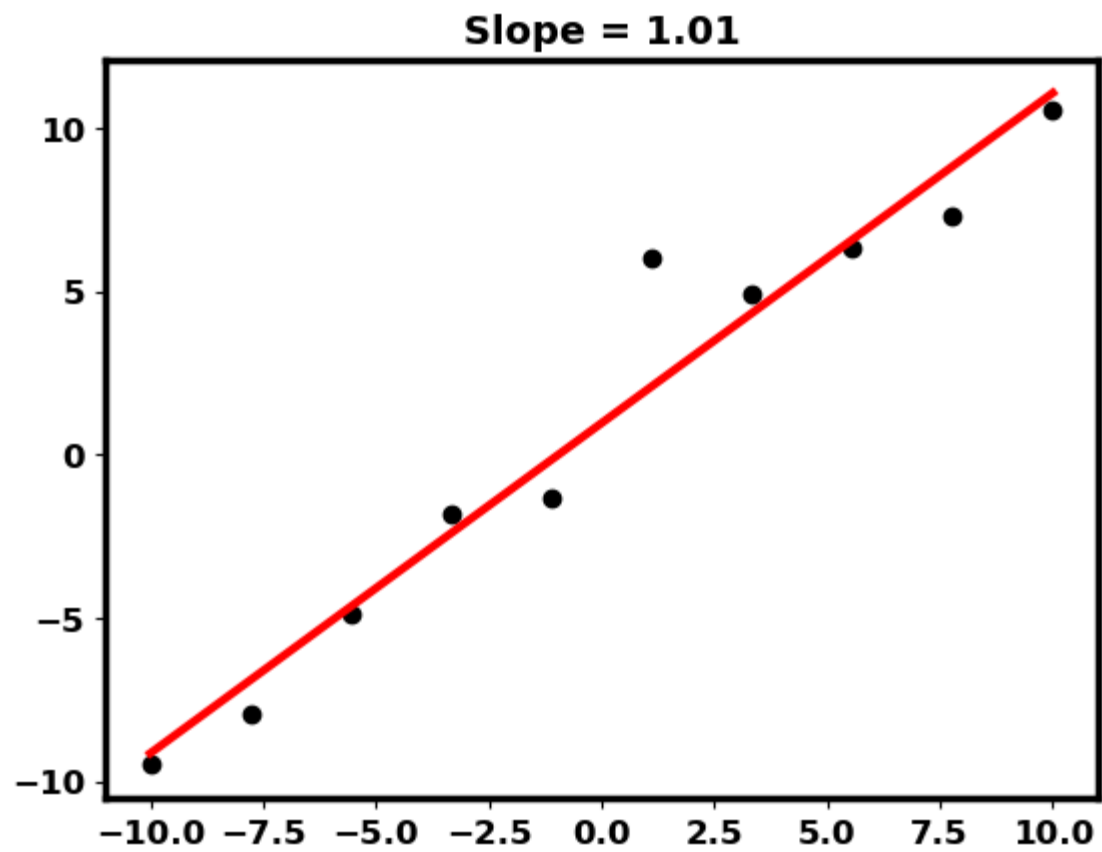
```
In [5]: iph.plot_interact(fit_update)
```

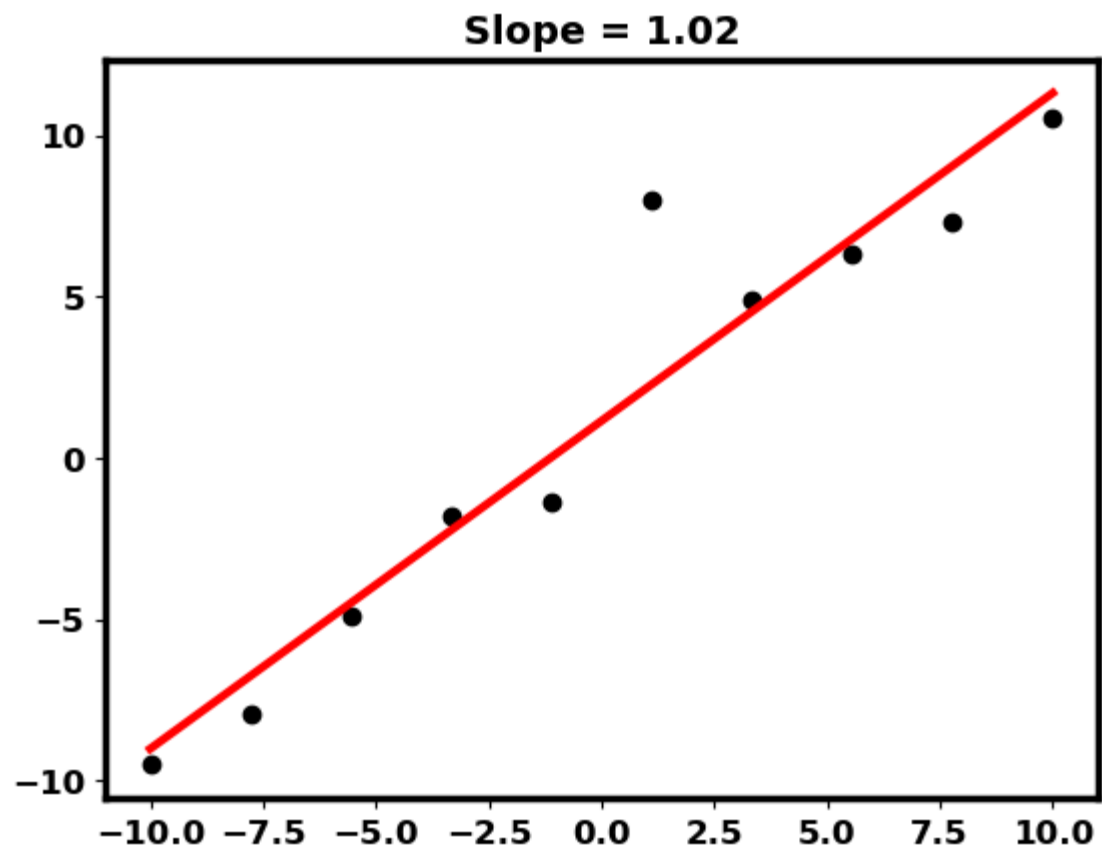


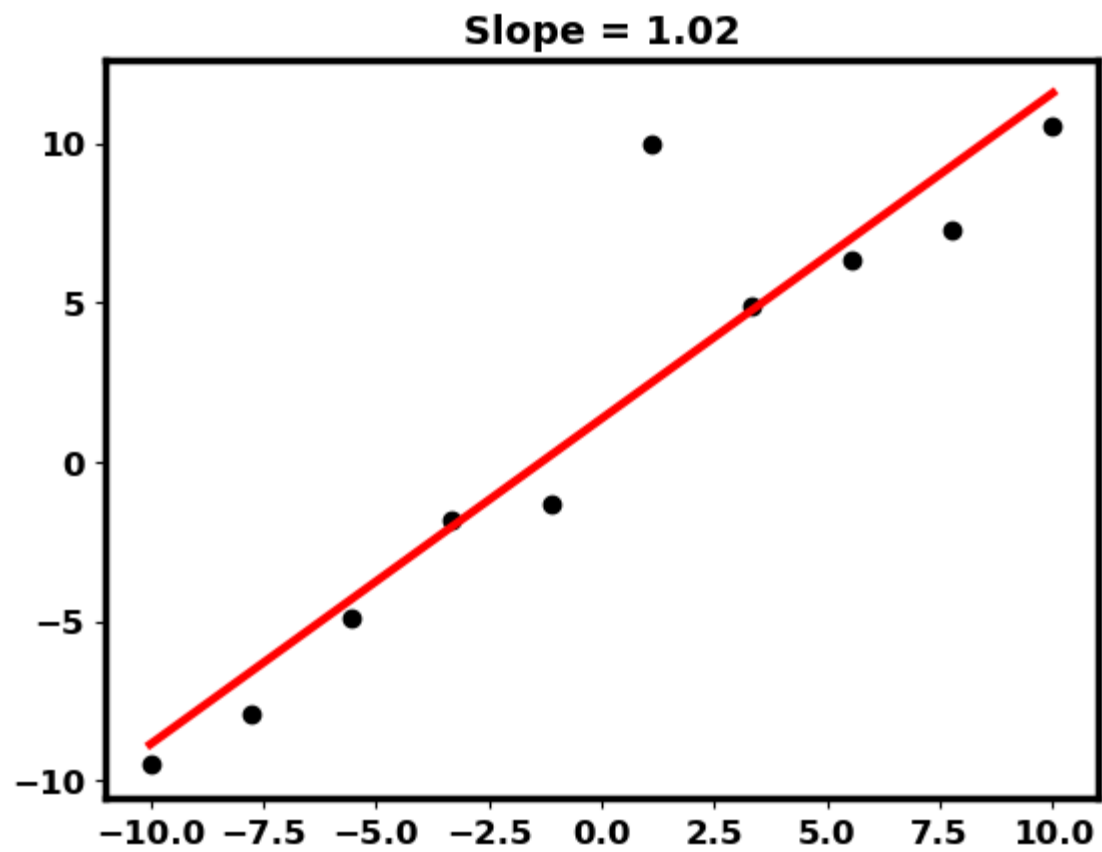


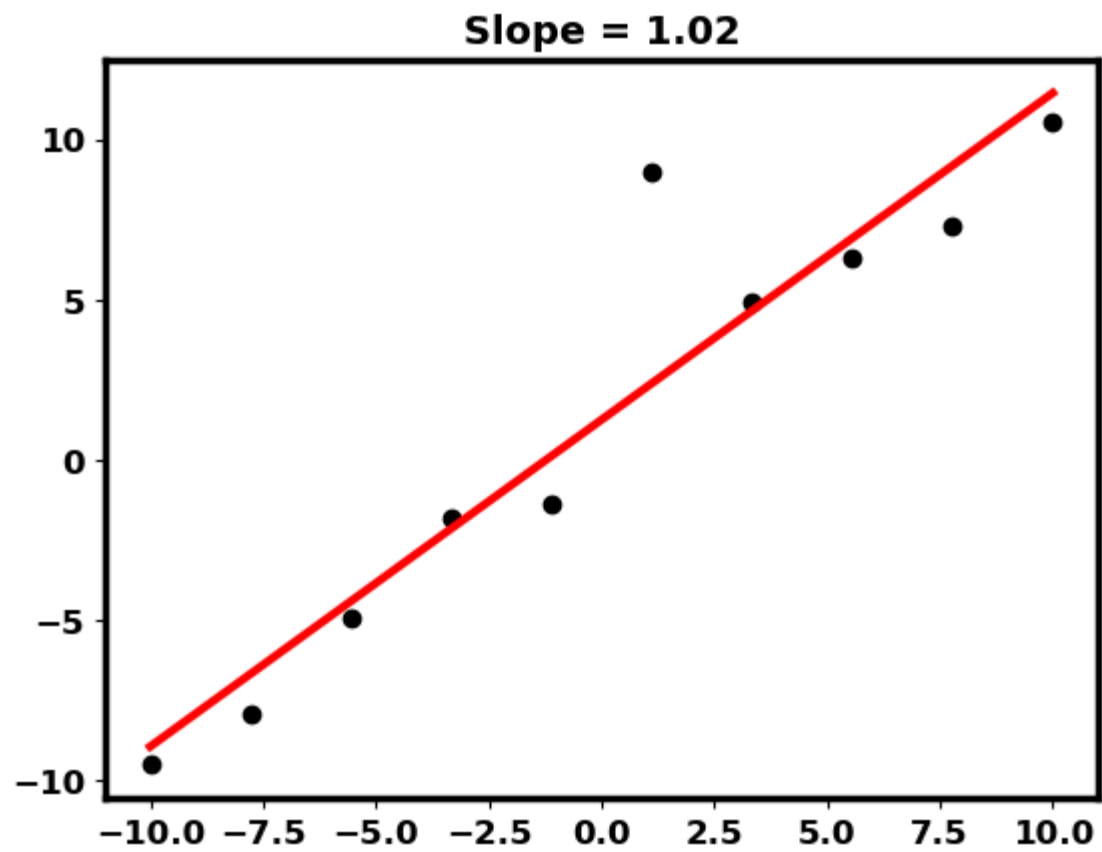


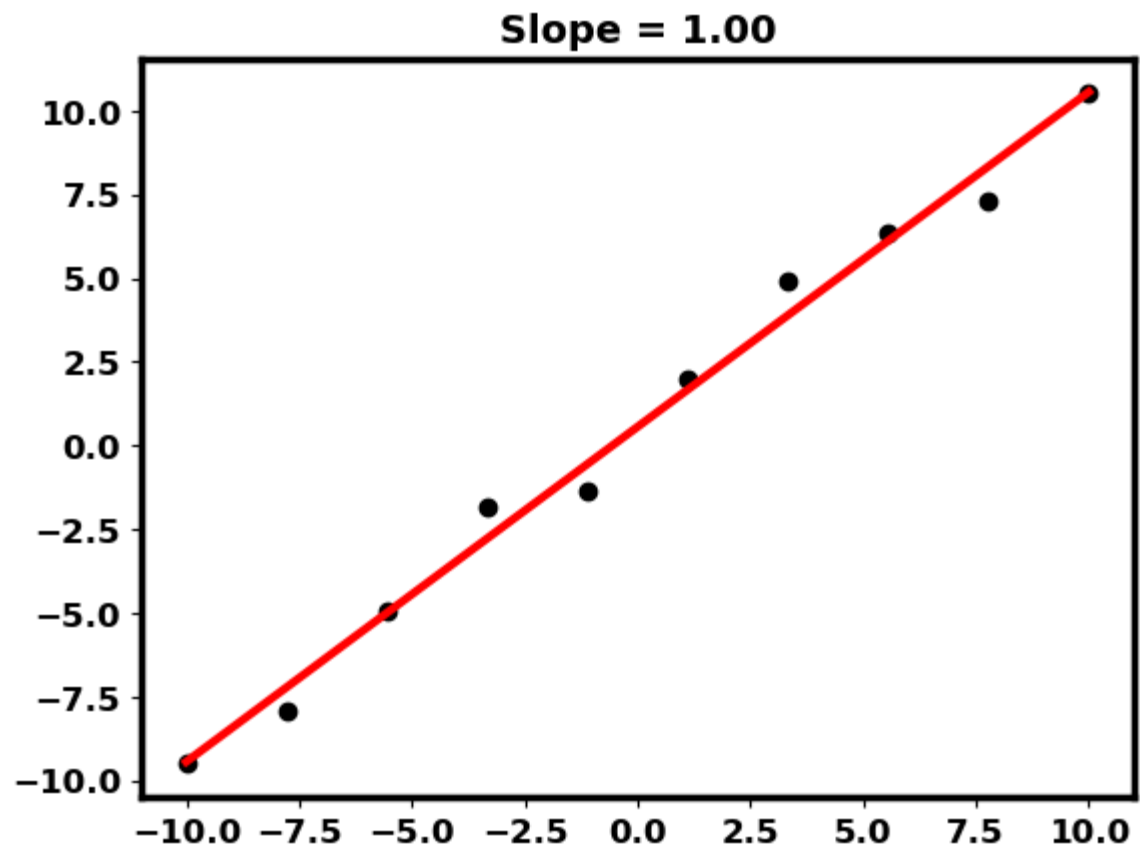


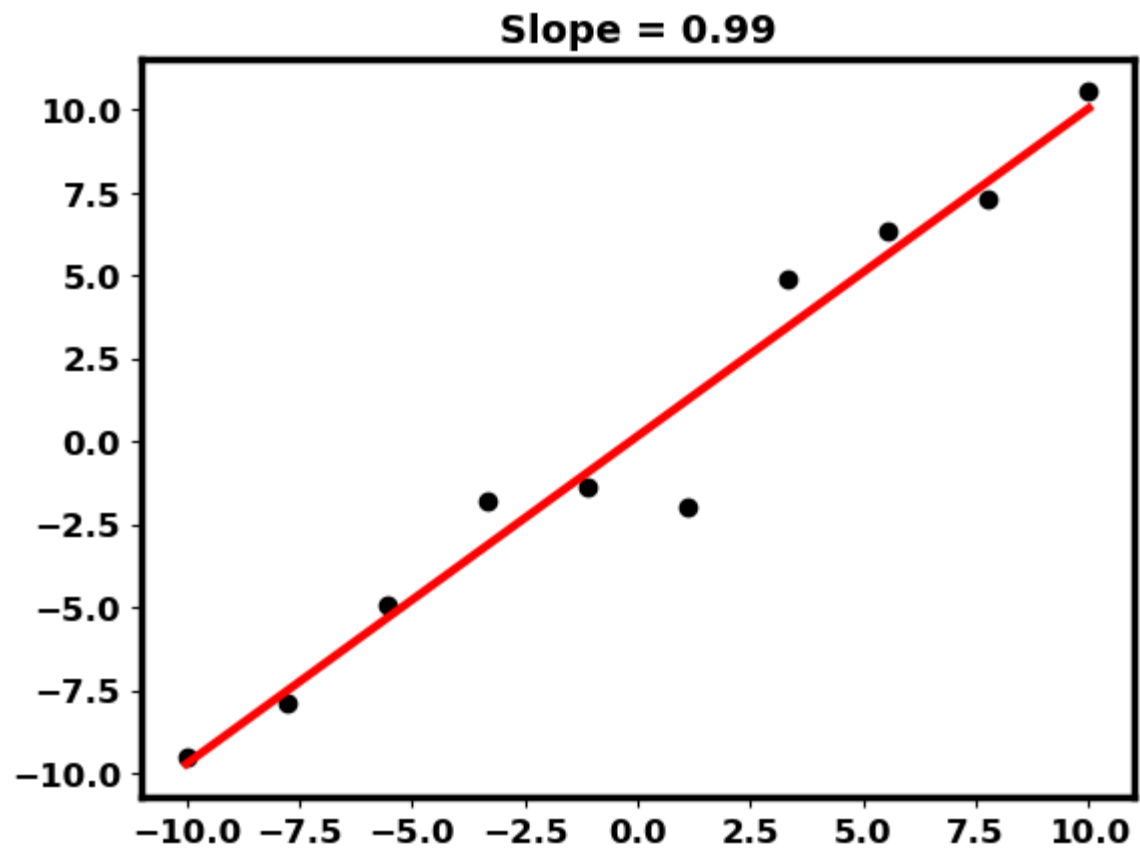


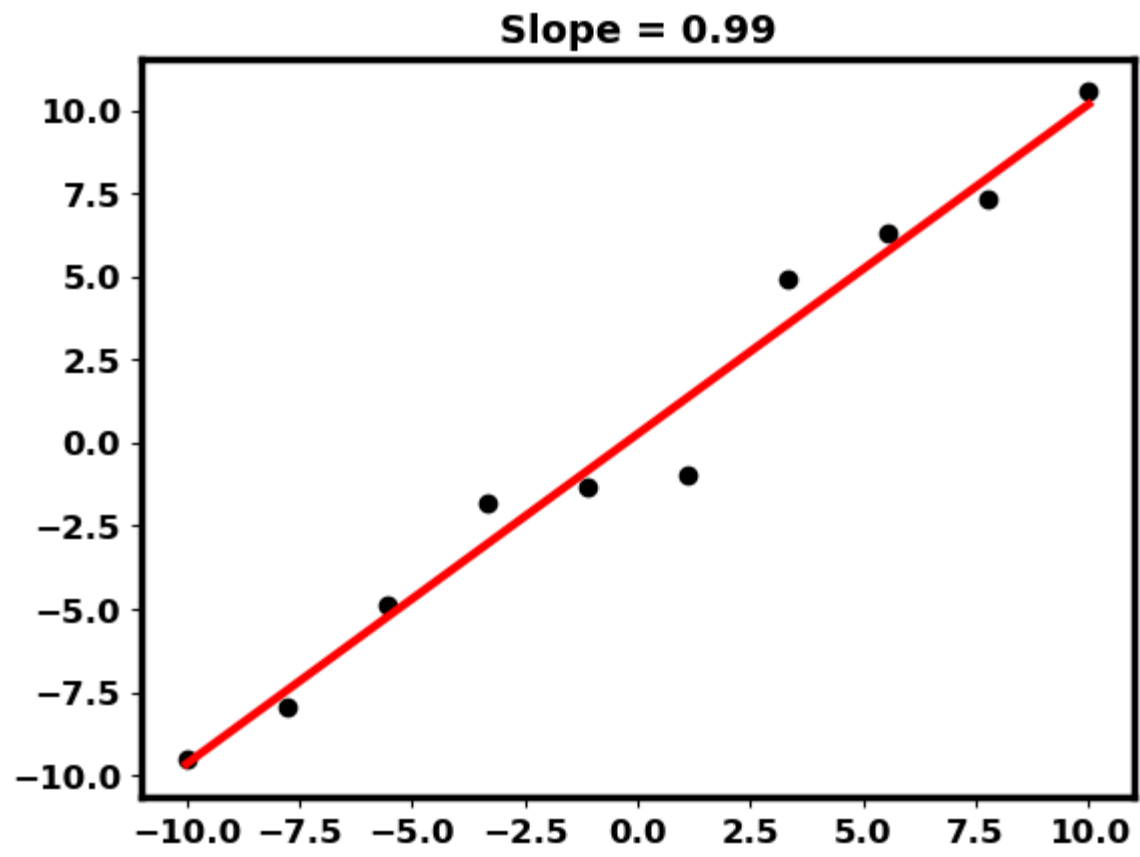


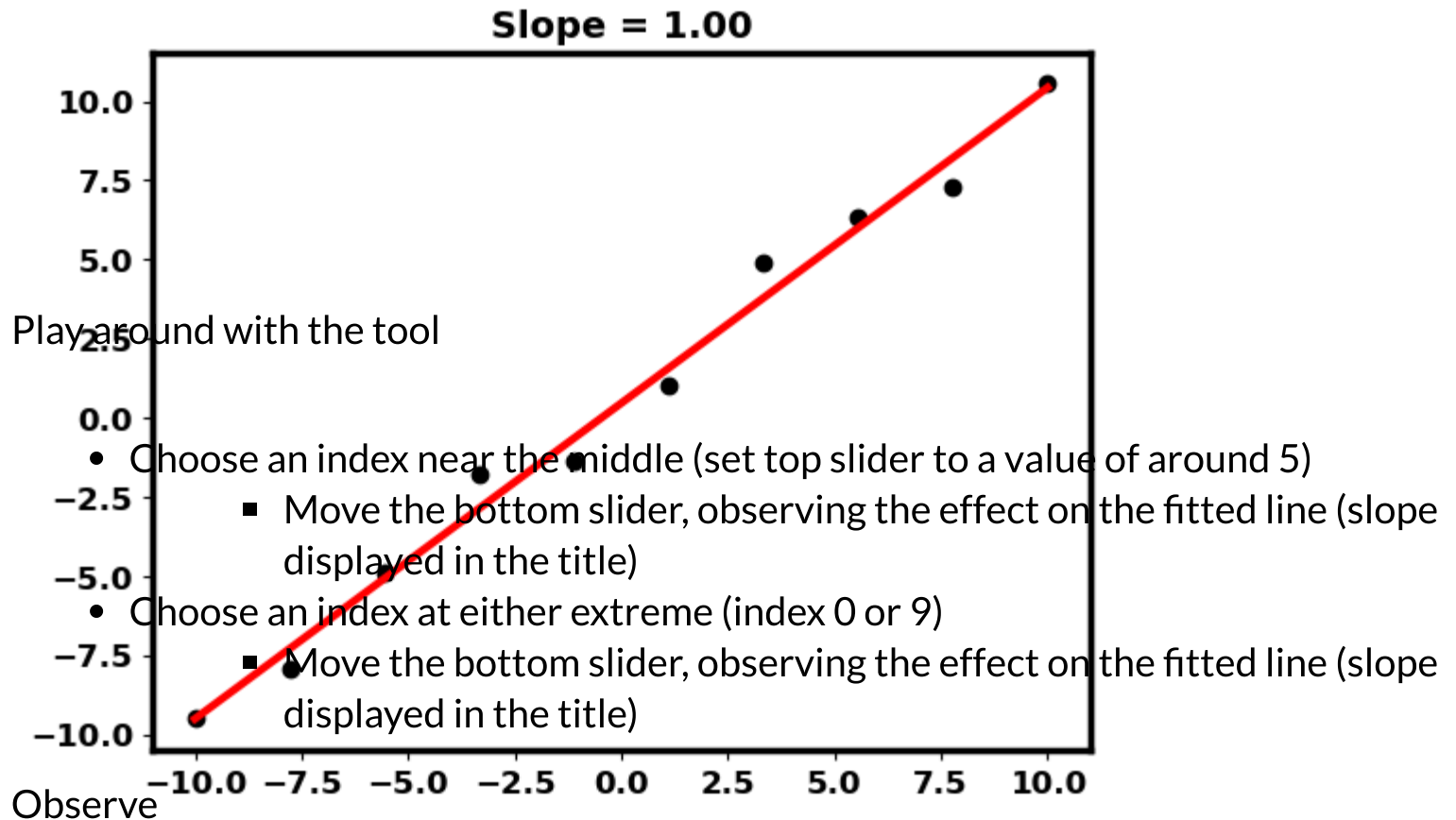




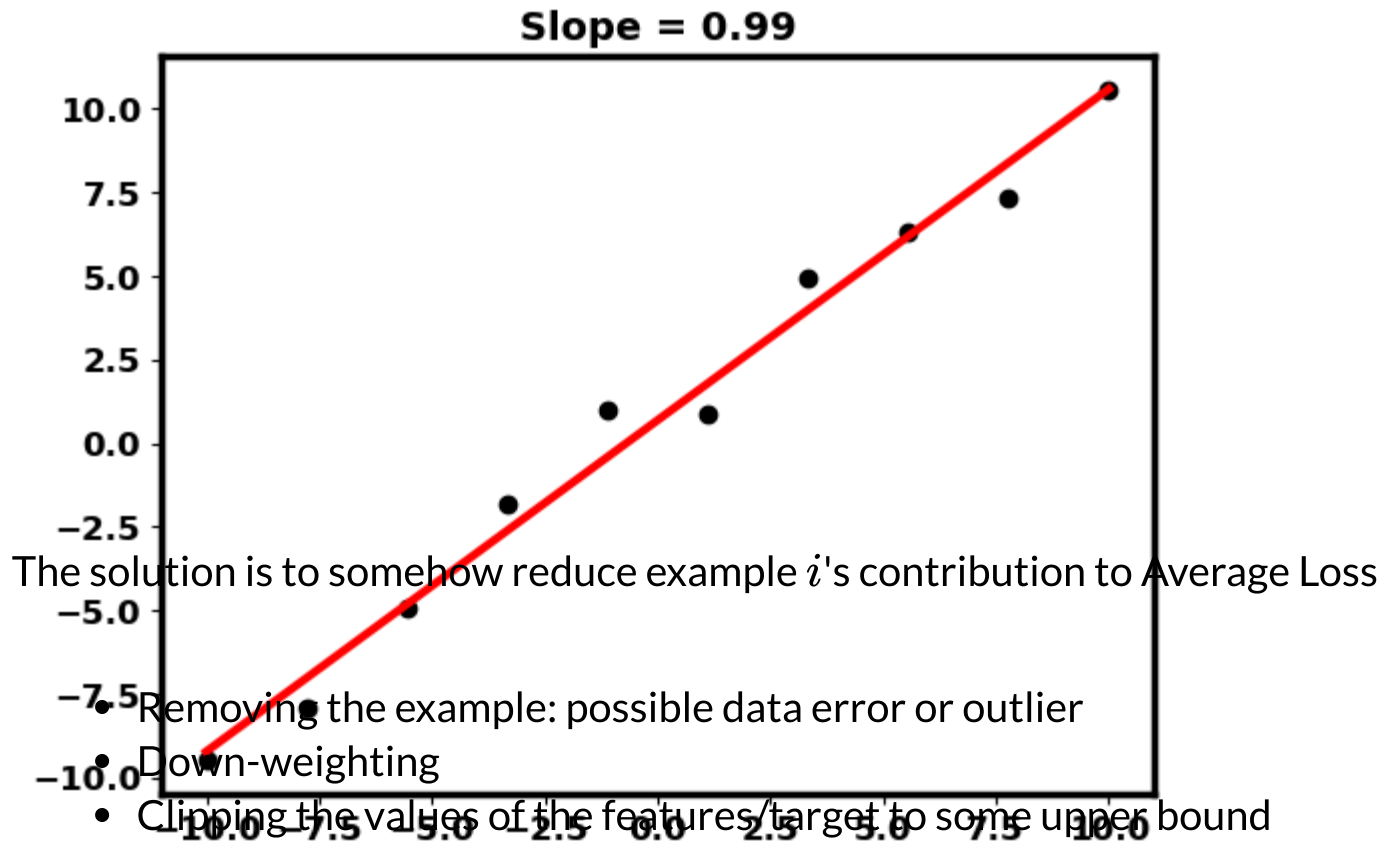


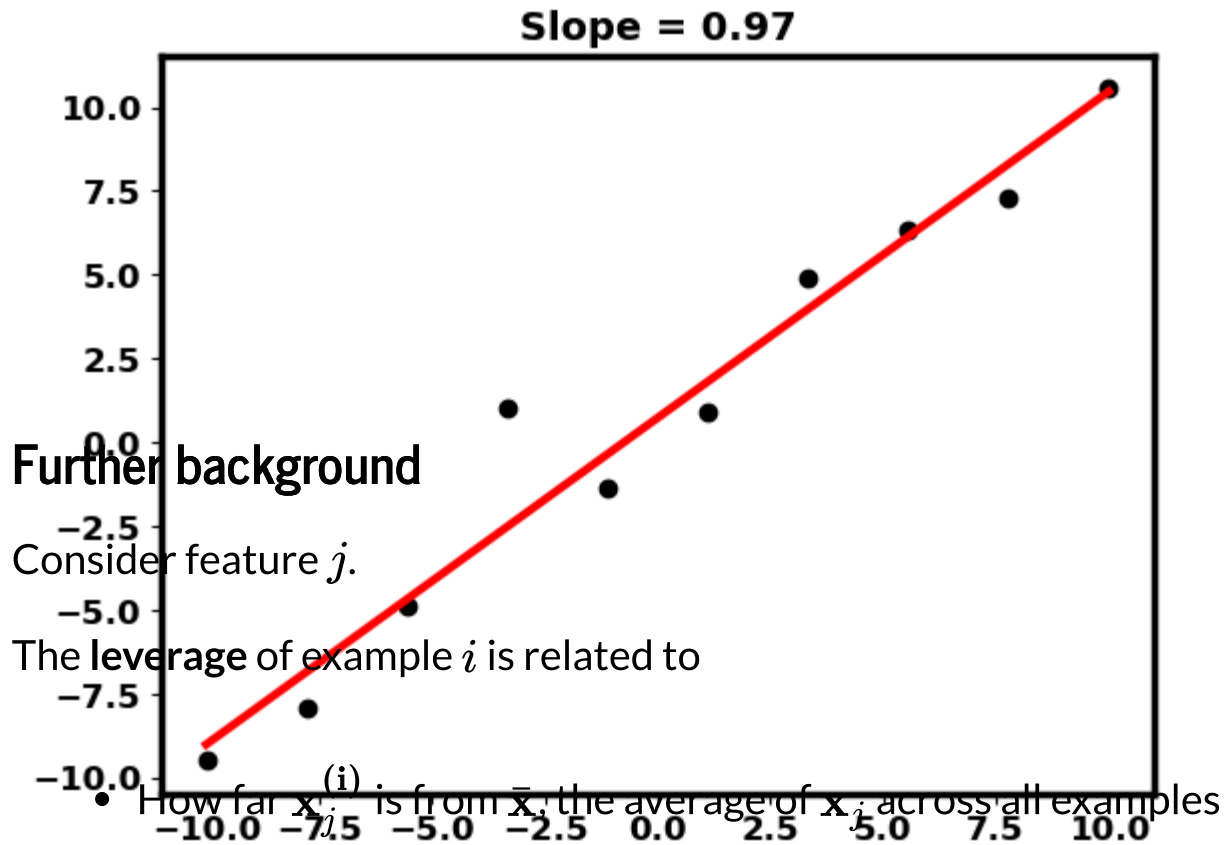




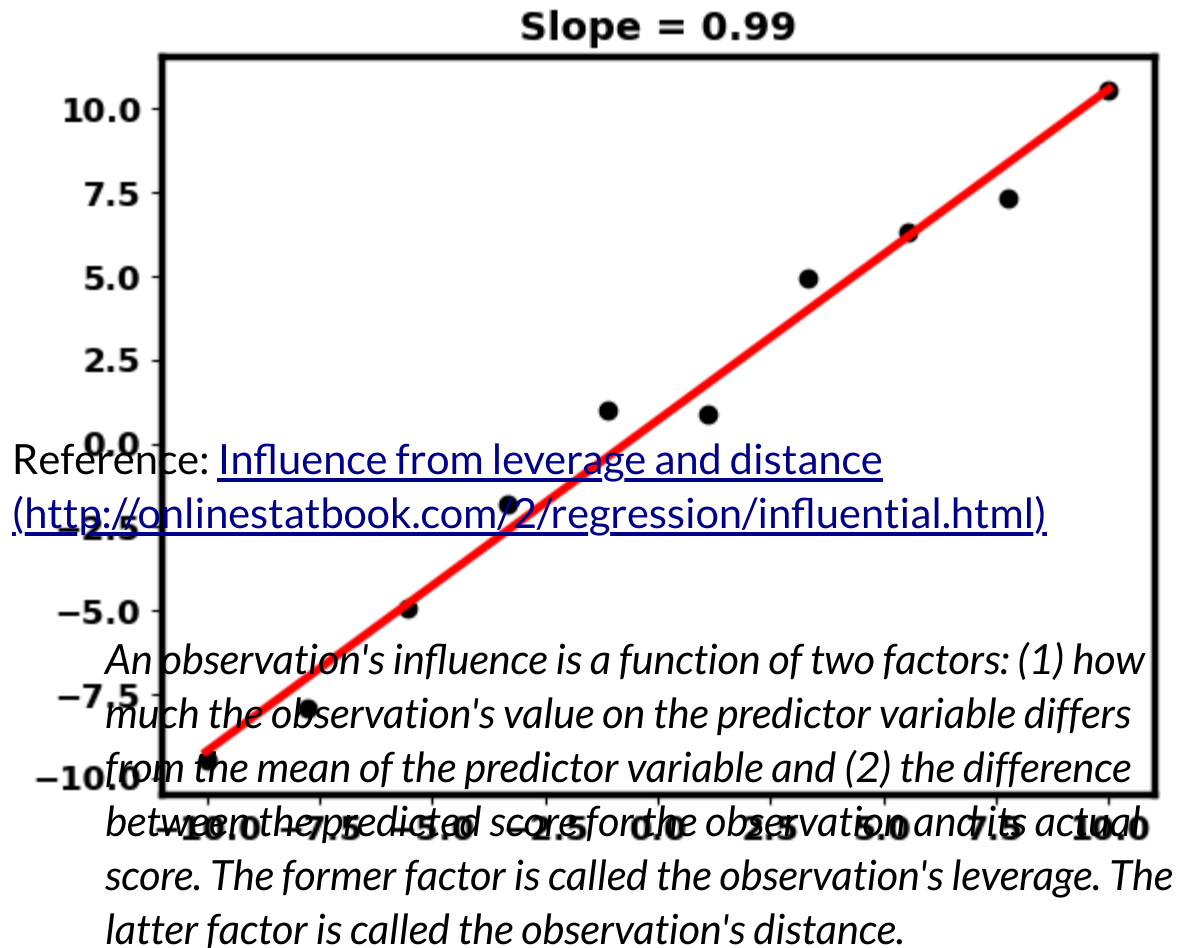


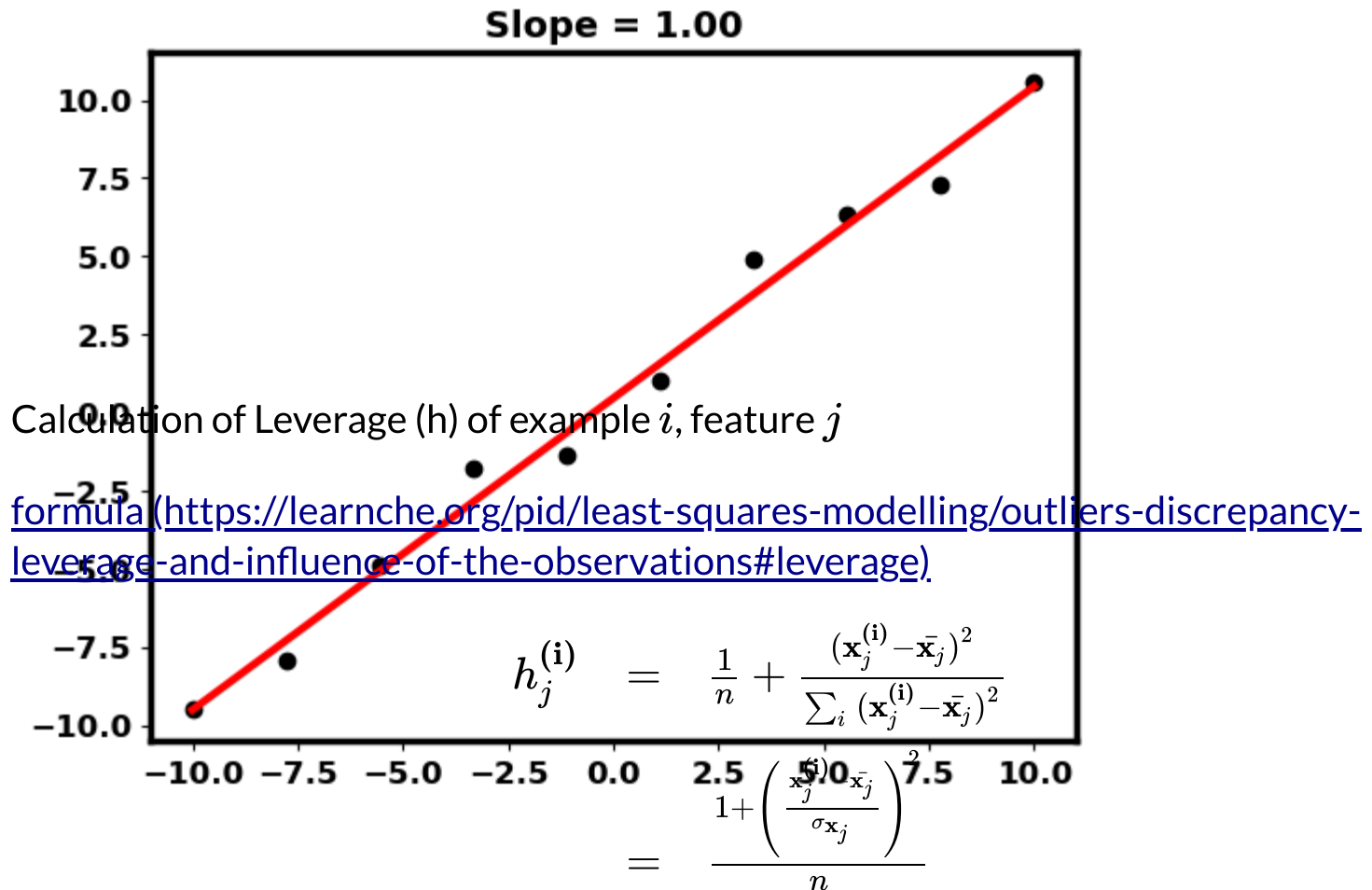
- Changing $y^{(i)}$ for i near the middle of the range has little effect on the fit
- Changing $y^{(i)}$ for i near either end (0 or 9) has a large effect on the fit





It is not always the case, but high leverage sometimes makes the point influential





You can see that the leverage of $\mathbf{x}_j^{(i)}$ depends on the (standardized) distance of $\mathbf{x}_j^{(i)}$ from the mean (over all i) of \mathbf{x}_i .

In [6]: `print("Done")`

Done

