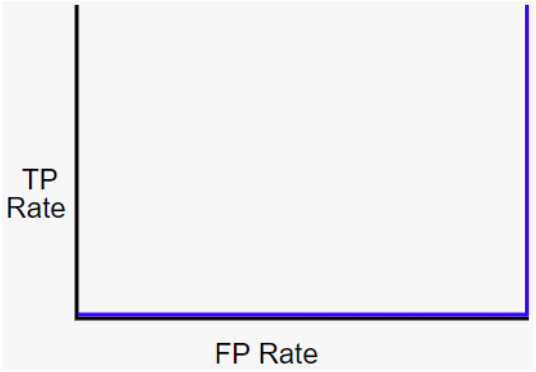
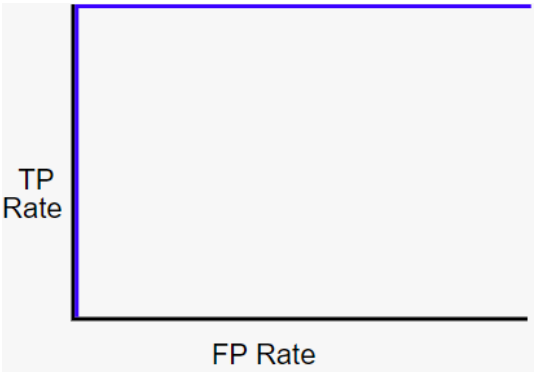
Machine Learning (Lecture 7)

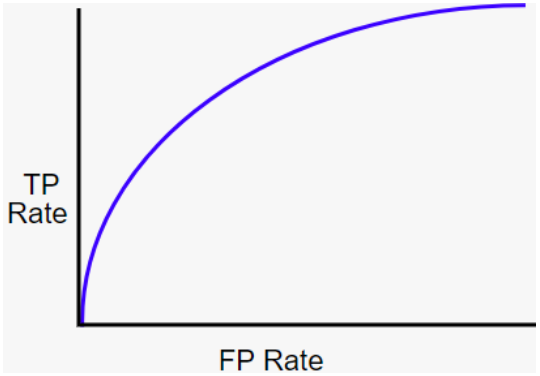
UEM/IEM Summer 2018

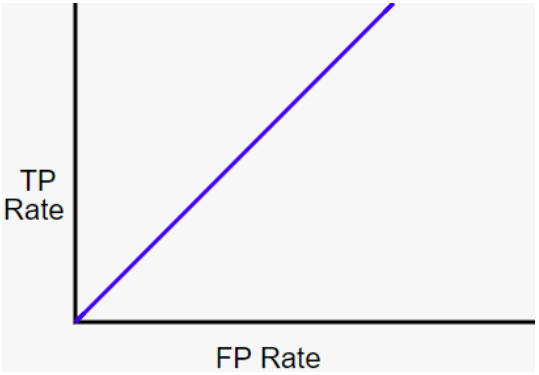
Quiz: ROC and AUC

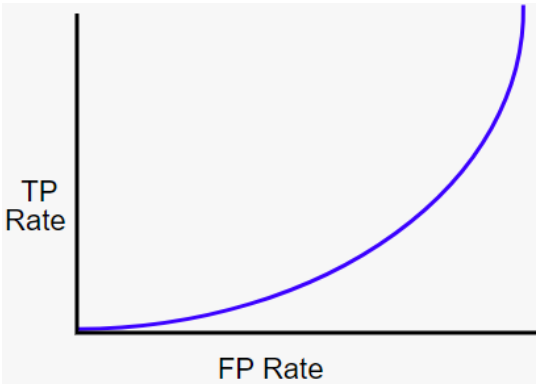
- Which of the following ROC curves produce AUC values greater than 0.5?





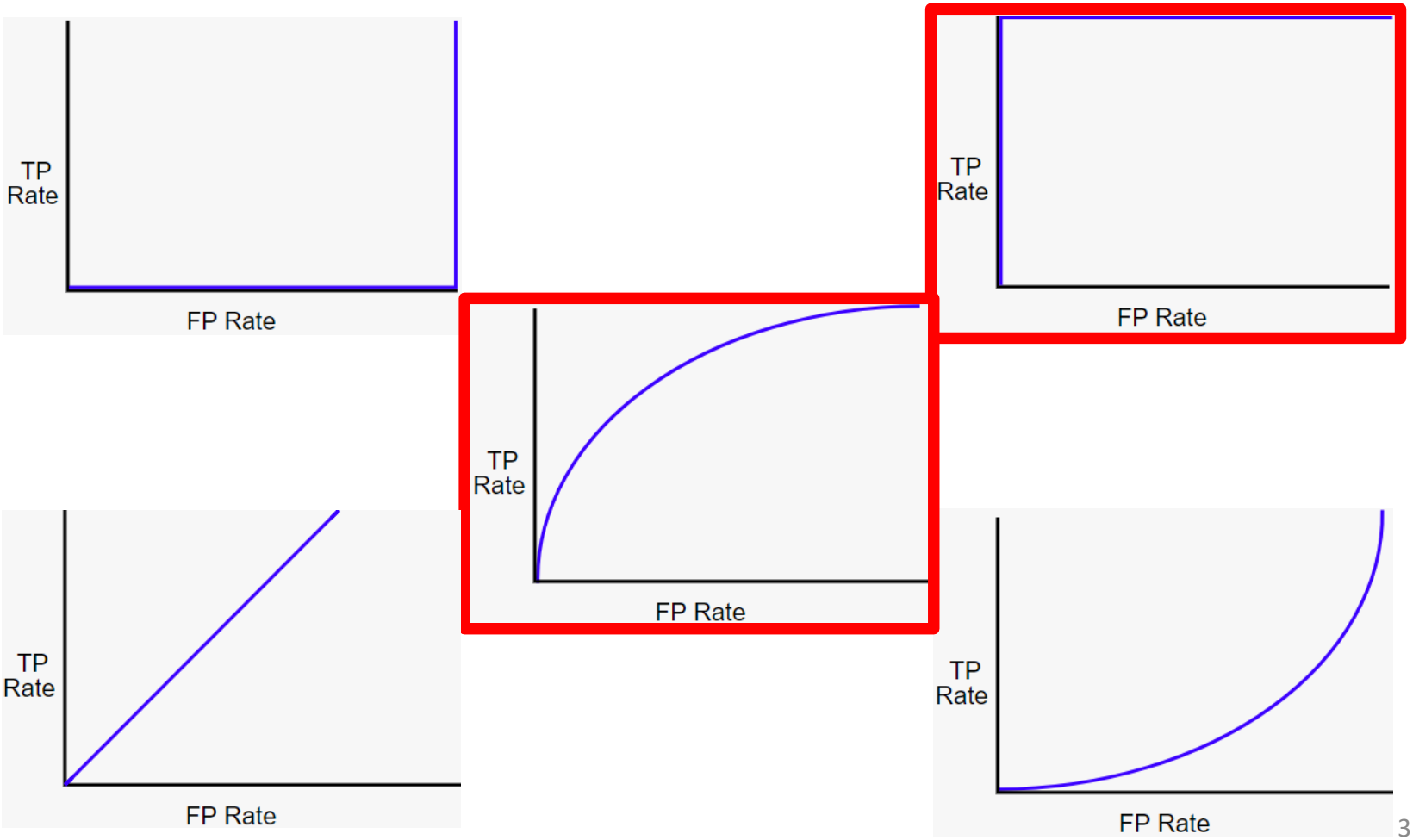






Quiz: ROC and AUC

- Which of the following ROC curves produce AUC values greater than 0.5?



Quiz: AUC and Scaling Predictions

- How would multiplying all of the predictions from a given model by 2.0 (for example, if the model predicts 0.4, we multiply by 2.0 to get a prediction of 0.8) change the model's performance as measured by AUC?

  - It would make AUC terrible, since the prediction values are now way off.
  - It would make AUC better, because the prediction values are all farther apart.
  - No change. AUC only cares about relative prediction scores.

Quiz: AUC and Scaling Predictions

- How would multiplying all of the predictions from a given model by 2.0 (for example, if the model predicts 0.4, we multiply by 2.0 to get a prediction of 0.8) change the model's performance as measured by AUC?

  - It would make AUC terrible, since the prediction values are now way off.
  - It would make AUC better, because the prediction values are all farther apart.
  - No change. AUC only cares about relative prediction scores.

Classification: Prediction Bias

- Logistic regression predictions should be unbiased. That is:

  "average of predictions" should ≈ "average of observations"

- **Prediction bias** is a quantity that measures how far apart those two averages are. That is:

$$\text{prediction bias} = \text{average of predictions} - \text{average of labels in data set}$$

- A significant nonzero prediction bias tells you there is a bug somewhere in your model, as it indicates that the model is wrong about how frequently positive labels occur.

Classification: Prediction Bias

- For example, let's say we know that on average, 1% of all emails are spam.
- If we don't know anything at all about a given email, we should predict that it's 1% likely to be spam.
- Similarly, a good spam model should predict on average that emails are 1% likely to be spam.
- (In other words, if we average the predicted likelihoods of each individual email being spam, the result should be 1%.)
- If instead, the model's average prediction is 20% likelihood of being spam, we can conclude that it exhibits prediction bias.

Classification: Prediction Bias

- Possible root causes of prediction bias are:
  - Incomplete feature set
  - Noisy data set
  - Buggy pipeline
  - Biased training sample
  - Overly strong regularization
- You might be tempted to correct prediction bias by post-processing the learned model—that is, by adding a **calibration layer** that adjusts your model's output to reduce the prediction bias.

Classification: Prediction Bias

- For example, if your model has +3% bias, you could add a calibration layer that lowers the mean prediction by 3%. However, adding a calibration layer is a bad idea for the following reasons:
  - You're fixing the symptom rather than the cause.
  - You've built a more brittle system that you must now keep up to date.
- If possible, avoid calibration layers. Projects that use calibration layers tend to become reliant on them—using calibration layers to fix all their model's sins.
- Ultimately, maintaining the calibration layers can become a nightmare.

Classification: Prediction Bias

- Bucketing and Prediction Bias
  - Logistic regression predicts a value *between* 0 and 1.
  - However, all labeled examples are either exactly 0 (meaning, for example, "not spam") or exactly 1 (meaning, for example, "spam").
  - Therefore, when examining prediction bias, you cannot accurately determine the prediction bias based on only one example; you must examine the prediction bias on a "bucket" of examples.
  - That is, prediction bias for logistic regression only makes sense when grouping enough examples together to be able to compare a predicted value (for example, 0.392) to observed values (for example, 0.394).

Classification: Prediction Bias

- You can form buckets in the following ways:
  - Linearly breaking up the target predictions.
  - Forming quantiles.
- Consider the following calibration plot from a particular model. Each dot represents a bucket of 1,000 values. The axes have the following meanings:
  - The x-axis represents the average of values the model predicted for that bucket.
  - The y-axis represents the actual average of values in the data set for that bucket.
- Both axes are logarithmic scales.

Classification: Prediction Bias

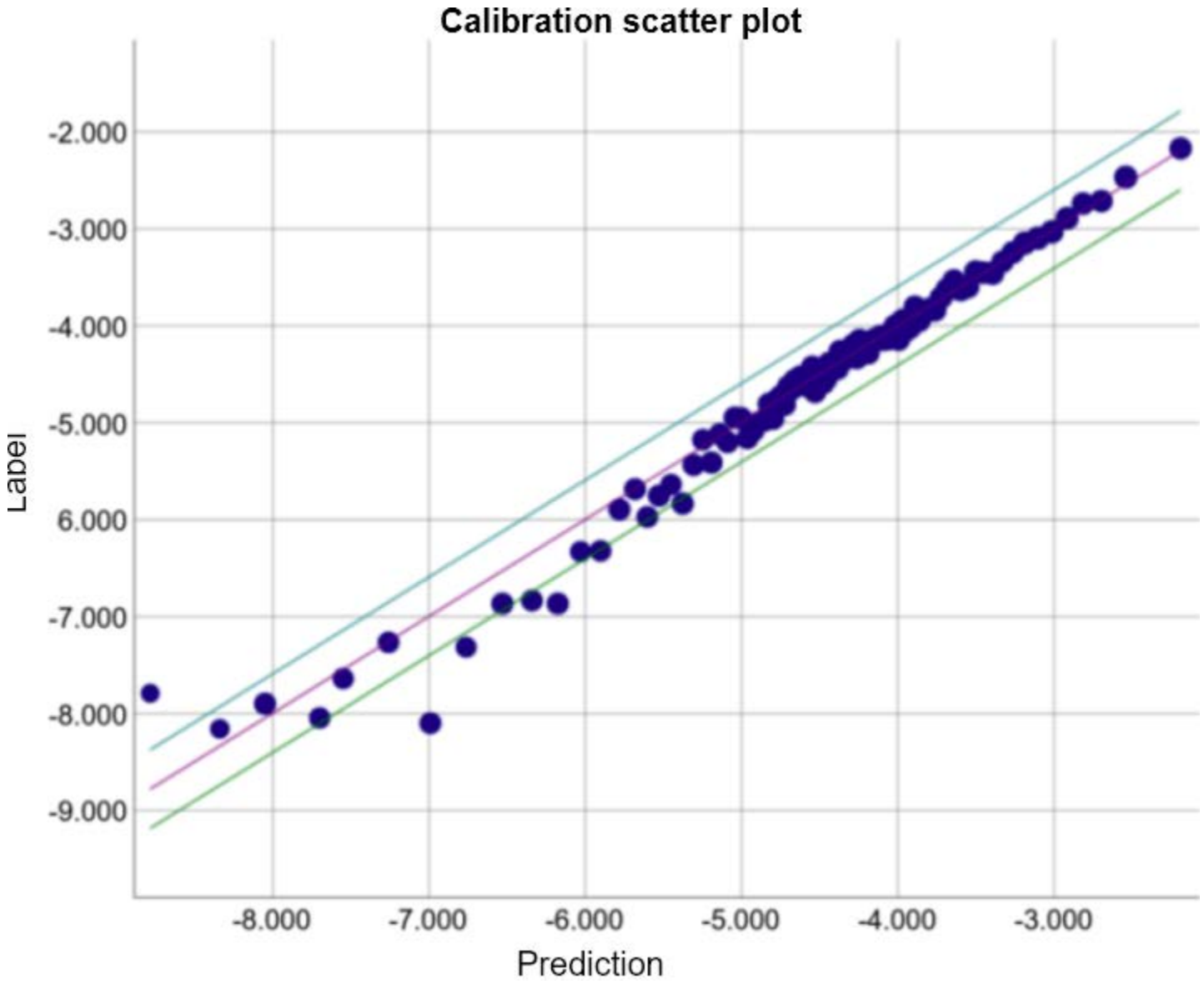


Figure 8. Prediction bias curve (logarithmic scales)

Classification: Prediction Bias

- Why are the predictions so poor for only *part* of the model? Here are a few possibilities:
  - The training set doesn't adequately represent certain subsets of the data space.
  - Some subsets of the data set are noisier than others.
  - The model is overly regularized. (Consider reducing the value of lambda.)

Regularization for Sparsity: L_1 Regularization

- The special requirements for models learned on feature vectors have many dimensions.
- Sparse vectors often contain many dimensions.
- Creating a feature cross results in even more dimensions.
- Given such high-dimensional feature vectors, model size may become huge and require huge amounts of RAM.
- In a high-dimensional sparse vector, it would be nice to encourage weights to drop to exactly 0 where possible.

Regularization for Sparsity: L_1 Regularization

- A weight of exactly 0 essentially removes the corresponding feature from the model.
- Zeroing out features will save RAM and may reduce noise in the model.
- For example, consider a housing data set that covers not just California but the entire globe. Bucketing global latitude at the minute level (60 minutes per degree) gives about 10,000 dimensions in a sparse encoding; global longitude at the minute level gives about 20,000 dimensions.
- A feature cross of these two features would result in roughly 200,000,000 dimensions.

Regularization for Sparsity: L_1 Regularization

- Many of those 200,000,000 dimensions represent areas of such limited residence (for example, the middle of the ocean) that it would be difficult to use that data to generalize effectively.
- It would be silly to pay the RAM cost of storing these unneeded dimensions.
- Therefore, it would be nice to encourage the weights for the meaningless dimensions to drop to exactly 0, which would allow us to avoid paying for the storage cost of these model coefficients at inference time.

Regularization for Sparsity: L_1 Regularization

- We might be able to encode this idea into the optimization problem done at training time, by adding an appropriately chosen regularization term.
- Would L_2 regularization accomplish this task? Unfortunately not. L_2 regularization encourages weights to be small, but doesn't force them to exactly 0.0.

Regularization for Sparsity: L_1 Regularization

- An alternative idea would be to try and create a regularization term that penalizes the count of non-zero coefficient values in a model.
- Increasing this count would only be justified if there was a sufficient gain in the model's ability to fit the data.
- Unfortunately, while this count-based approach is intuitively appealing, it would turn our convex optimization problem into a non-convex optimization problem that's NP-hard. (If you squint, you can see a connection to the knapsack problem.)
- So this idea, known as L_0 regularization isn't something we can use effectively in practice.

Regularization for Sparsity: L_1 Regularization

- However, there is a regularization term called L_1 regularization that serves as an approximation to L_0, but has the advantage of being convex and thus efficient to compute.
- So we can use L_1 regularization to encourage many of the uninformative coefficients in our model to be exactly 0, and thus reap RAM savings at inference time.

Regularization for Sparsity: L_1 Regularization

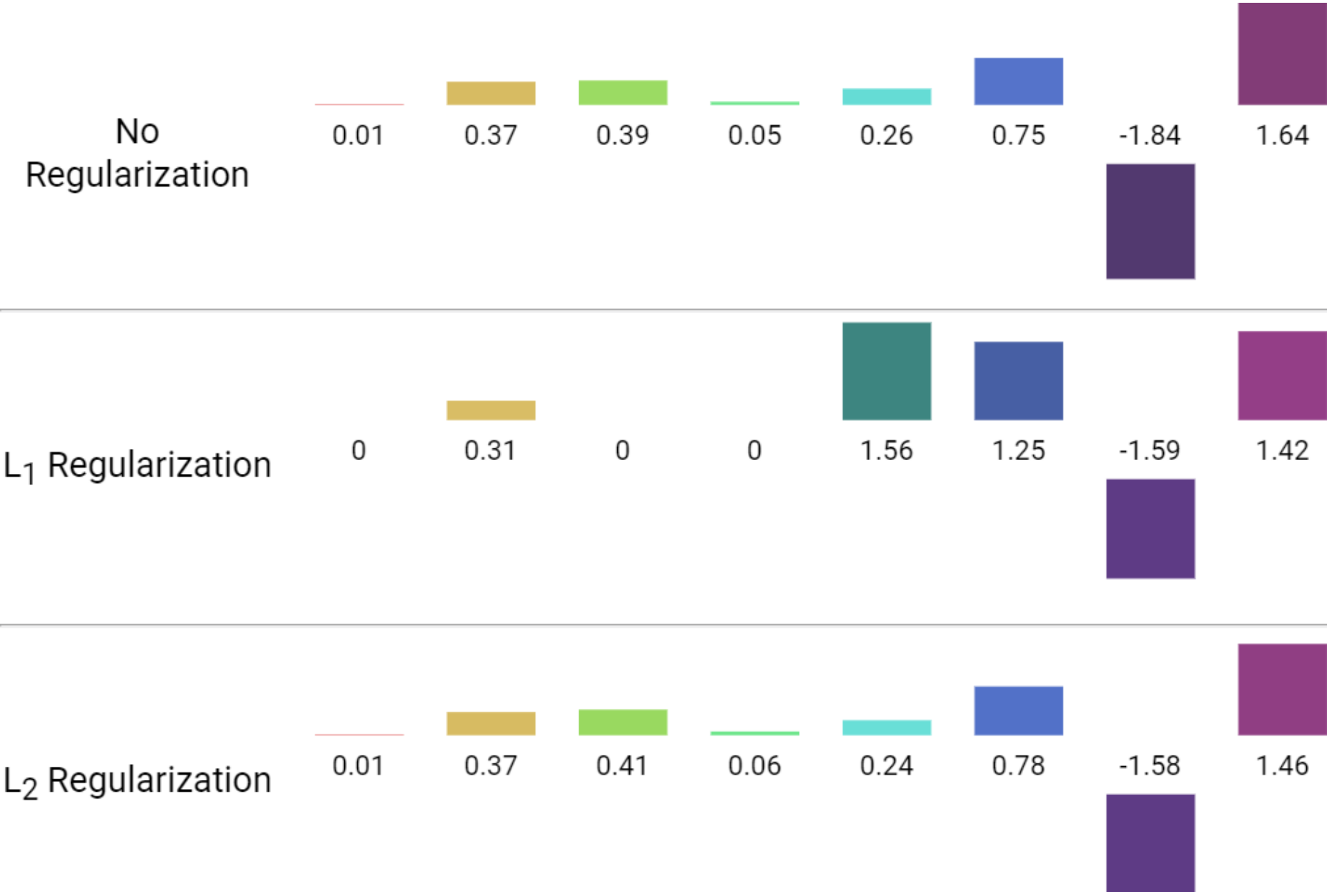
- L_1 vs L_2 regularization.
  - L_2 and L_1 penalize weights differently:
    - L_2 penalizes $weight^2$.
    - L_1 penalizes $|weight|$.
  - Consequently, L_2 and L_1 have different derivatives:
    - The derivative of L_2 is 2 * *weight*.
    - The derivative of L_1 is *k* (a constant, whose value is independent of weight).

Regularization for Sparsity: L_1 Regularization

- You can think of the derivative of L_2 as a force that removes x% of the weight every time. As Zeno knew, even if you remove x percent of a number *billions of times*, the diminished number will still never quite reach zero. (Zeno was less familiar with floating-point precision limitations, which could possibly produce exactly zero.)
- At any rate, L_2 does not normally drive weights to zero.

Regularization for Sparsity: L_1 Regularization

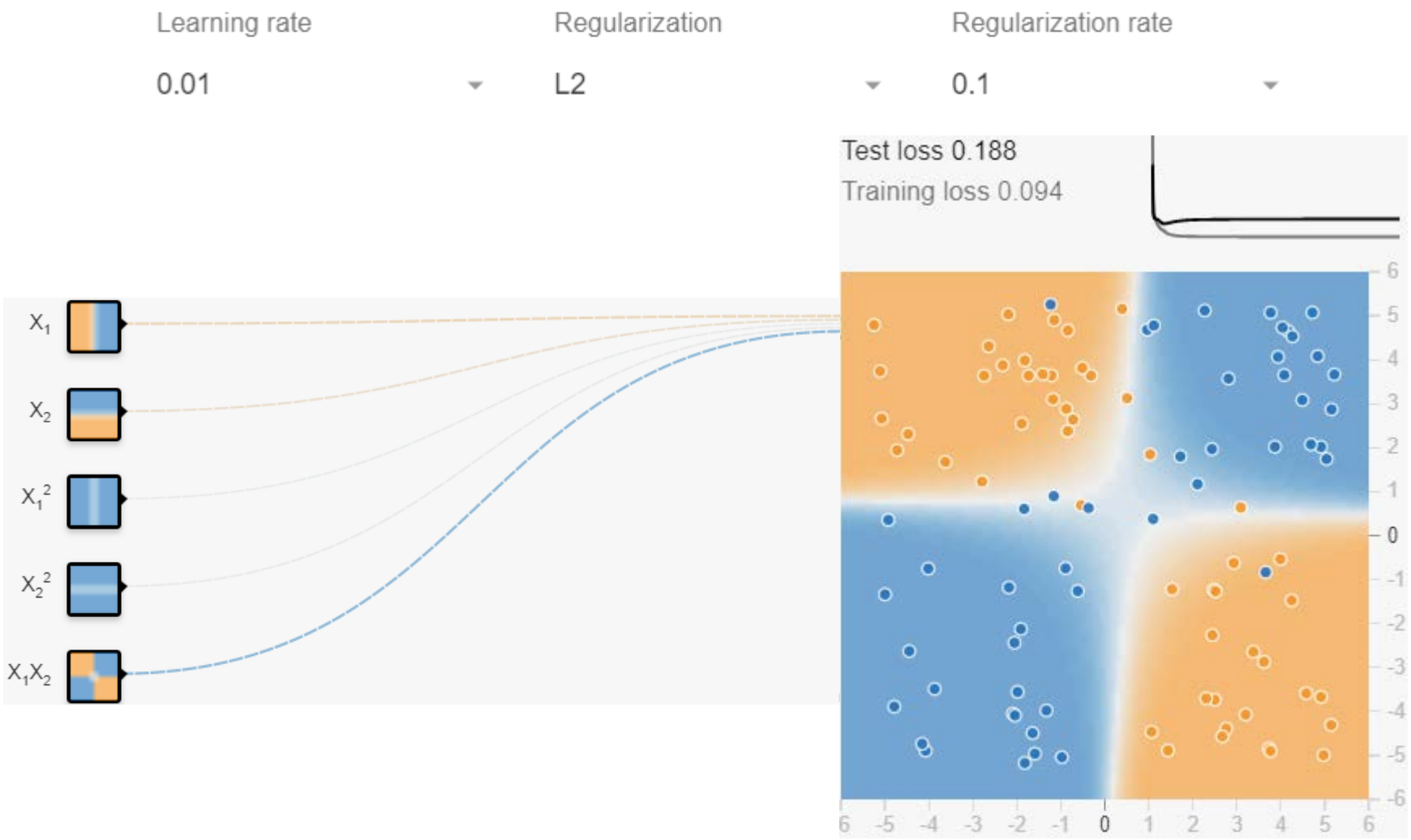
- You can think of the derivative of L_1 as a force that subtracts some constant from the weight every time. However, thanks to absolute values, L_1 has a discontinuity at 0, which causes subtraction results that cross 0 to become zeroed out.
- For example, if subtraction would have forced a weight from +0.1 to -0.2, L_1 will set the weight to exactly 0. Eureka, L_1 zeroed out the weight.
- L_1 regularization—penalizing the absolute value of all the weights—turns out to be quite efficient for wide models.
- Note that this description is true for a one-dimensional model.


No Regularization
0.01
0.37
0.39
0.05
0.26
0.75
-1.84
1.64
L_1 Regularization
0
0.31
0
0
1.56
1.25
-1.59
1.42
L_2 Regularization
0.01
0.37
0.41
0.06
0.24
0.78
-1.58
1.46

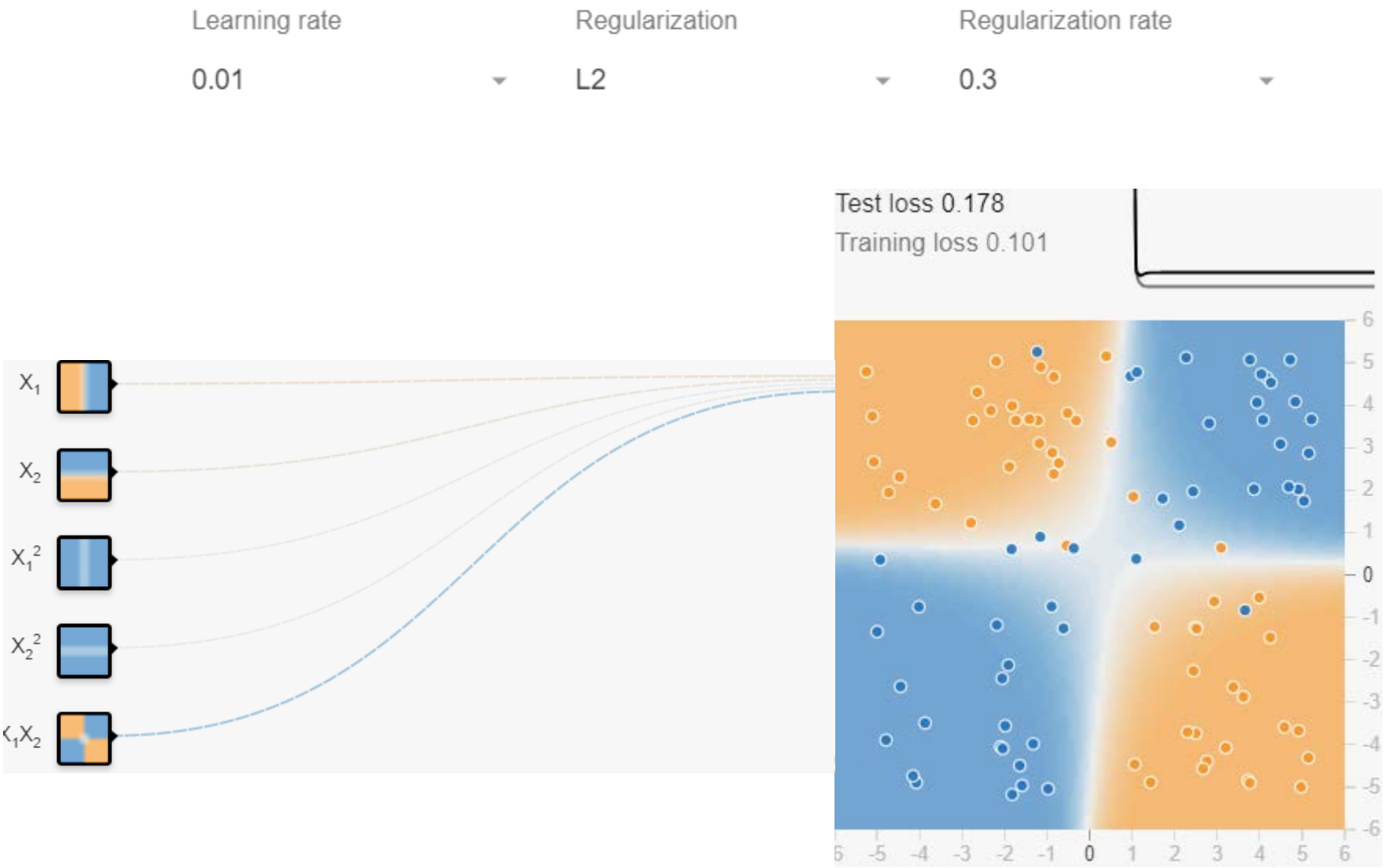
Regularization for Sparsity: L_1 Regularization

- This demonstration contains a small, slightly noisy, training data set. In this kind of setting, overfitting is a real concern. Regularization might help, but which form of regularization?
- Questions:
  - How does switching from L_2 to L_1 regularization influence the delta between test loss and training loss?
  - How does switching from L_2 to L_1 regularization influence the learned weights?
  - How does increasing the L_1 regularization rate (lambda) influence the learned weights?

Regularization for Sparsity: L_1 Regularization

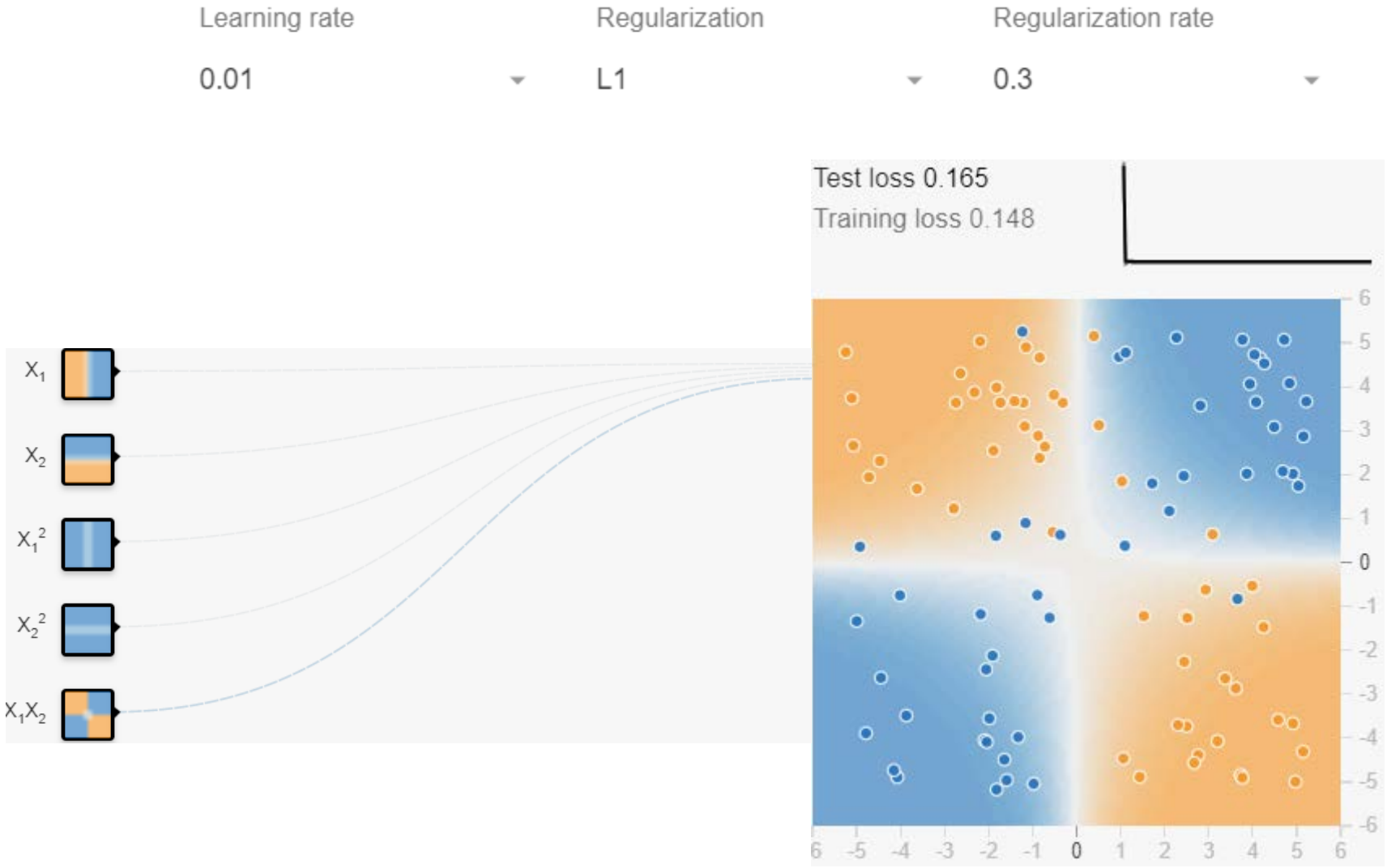


Regularization for Sparsity: L_1 Regularization

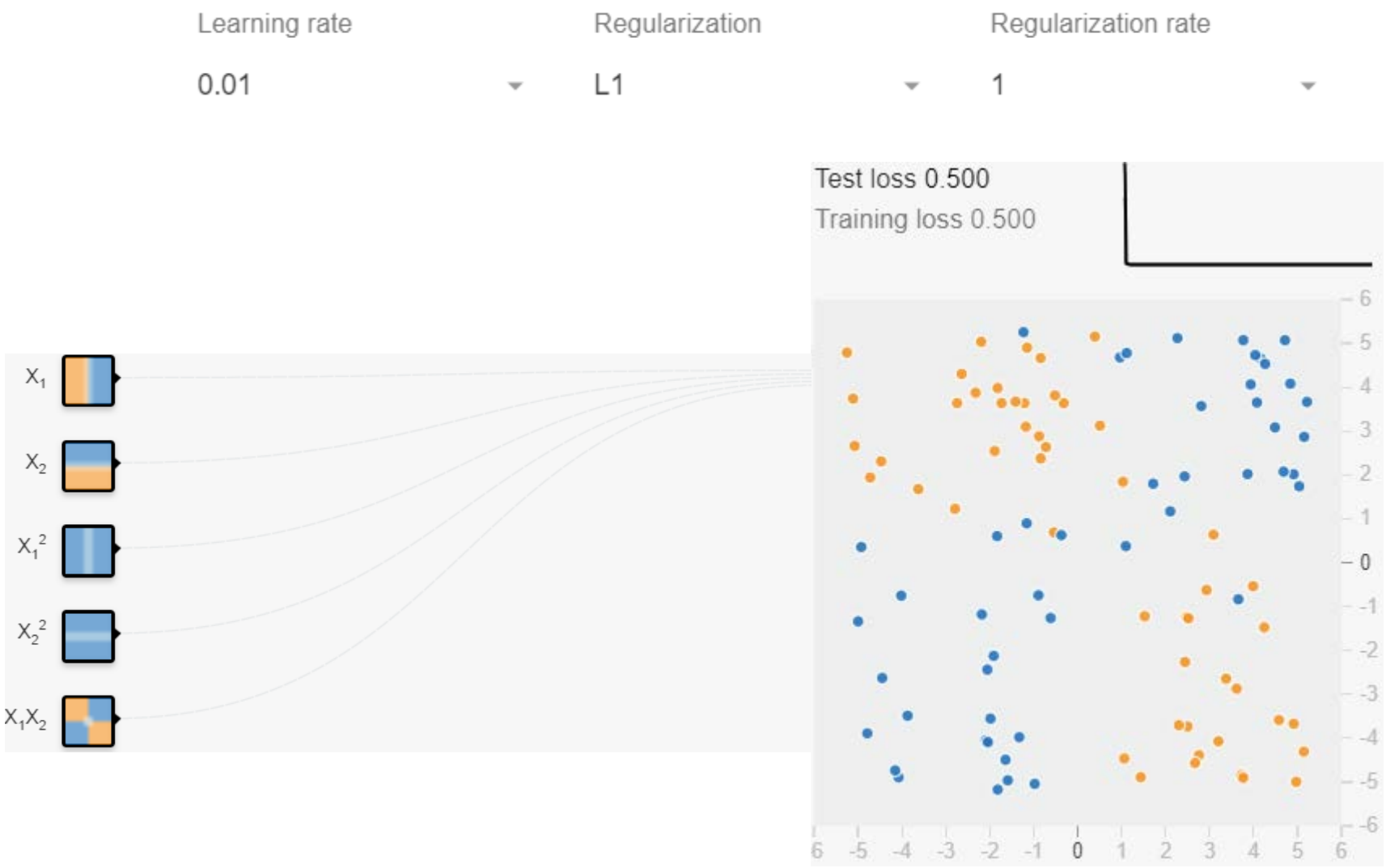


Regularization for Sparsity: L_1 Regularization

Regularization for Sparsity: L_1 Regularization



Regularization for Sparsity: L_1 Regularization



Quiz: L_1 regularization

- Imagine a linear model with 100 input features:
  - 10 are highly informative.
  - 90 are non-informative.

Assume that all features have values between -1 and 1. Which of the following statements are true?

- L1 regularization will encourage most of the non-informative weights to be exactly 0.0.
- L1 regularization will encourage many of the non-informative weights to be nearly (but not exactly) 0.0.
- L1 regularization may cause informative features to get a weight of exactly 0.0.

Quiz: L_1 regularization

- Imagine a linear model with 100 input features:
  - 10 are highly informative.
  - 90 are non-informative.

Assume that all features have values between -1 and 1. Which of the following statements are true?

- L1 regularization will encourage most of the non-informative weights to be exactly 0.0.
- L1 regularization will encourage many of the non-informative weights to be nearly (but not exactly) 0.0.
- L1 regularization may cause informative features to get a weight of exactly 0.0.

Quiz: L_1 vs. L_2 Regularization

- Imagine a linear model with 100 input features, all having values between -1 and 1:
  - 10 are highly informative.
  - 90 are non-informative.

  Which type of regularization will produce the smaller model?

  - L_2 regularization.
  - L_1 regularization.

Quiz: L_1 vs. L_2 Regularization

- Imagine a linear model with 100 input features, all having values between -1 and 1:
  - 10 are highly informative.
  - 90 are non-informative.

  Which type of regularization will produce the smaller model?

  - L_2 regularization.
  - L_1 regularization.

Reference

- This lecture note has been developed based on the machine learning crash course at Google, which is under *Creative Commons Attribution 3.0 License*.