Regularization

INFO-4604, Applied Machine Learning
University of Colorado Boulder

September 19, 2017

Prof. Michael Paul

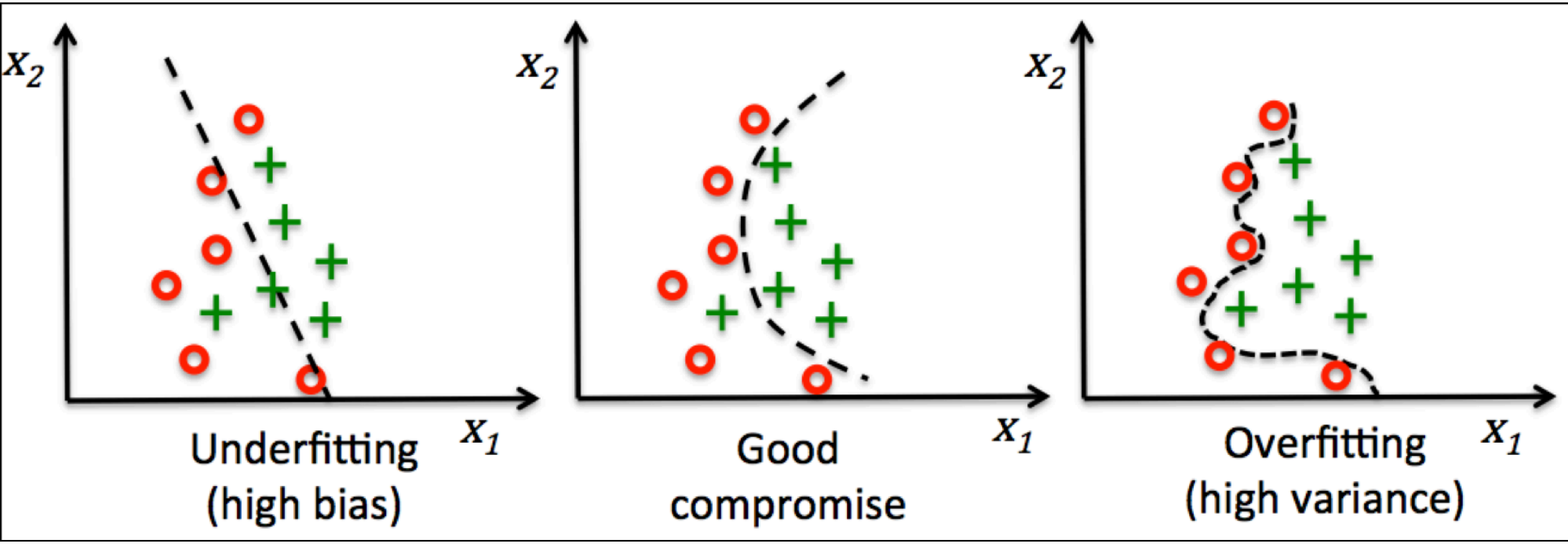
Generalization

Prediction functions that work on the training data might not work on other data

Minimizing the training error is a reasonable thing to do, but it's possible to minimize it "too well"

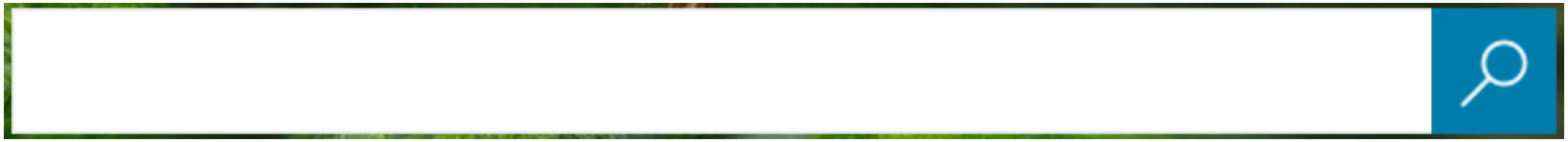
- If your function matches the training data well but is not learning general rules that will work for new data, this is called **overfitting**

Generalization



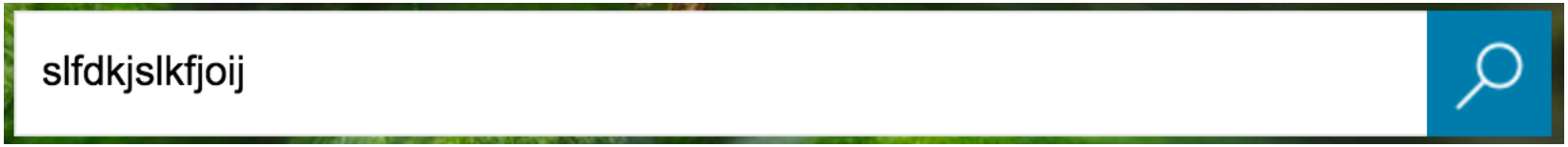
Overfitting: Logistic Regression

Suppose you are a search engine and you build a classifier to infer whether a user is over the age of 65 based on what they've searched.



Overfitting: Logistic Regression

One person in your dataset searched the following typo:

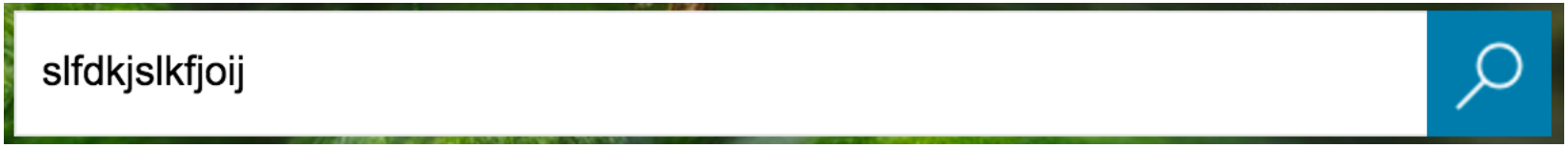


This person was over age 65.

Optimizing the logistic regression loss function, we would learn that anyone who searches *slfdkjslkfjoij* is over 65 with probability 1.

Overfitting: Logistic Regression

One person in your dataset searched the following typo:

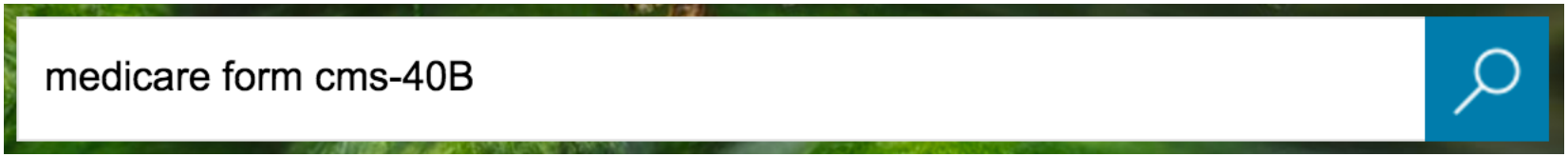


Hard to conclude much from 1 example.

Don’t really want to classify all people who make this typo in the future this way.

Overfitting: Logistic Regression

Ten people searched for the following form:

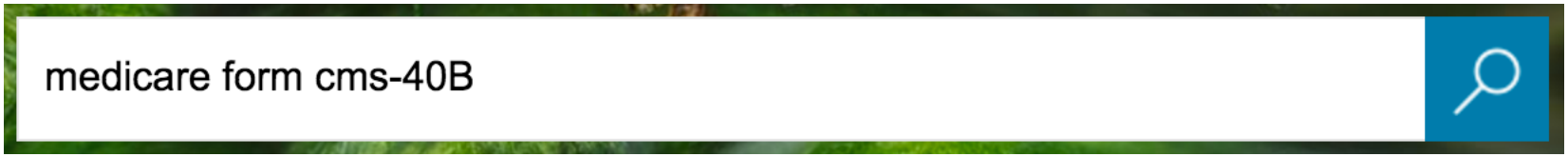


All ten people were over age 65.

Optimizing the logistic regression loss function, we would learn that anyone who searches this query is over 65 with probability 1.

Overfitting: Logistic Regression

Ten people searched for the following form:



This query is probably good evidence that someone is older than (or near) 65.

Still: what if someone searched this who otherwise had hundreds of queries that suggested they were younger? They would still be classified >65 with probability 1. The probability 1 overrides other features in logistic regression.

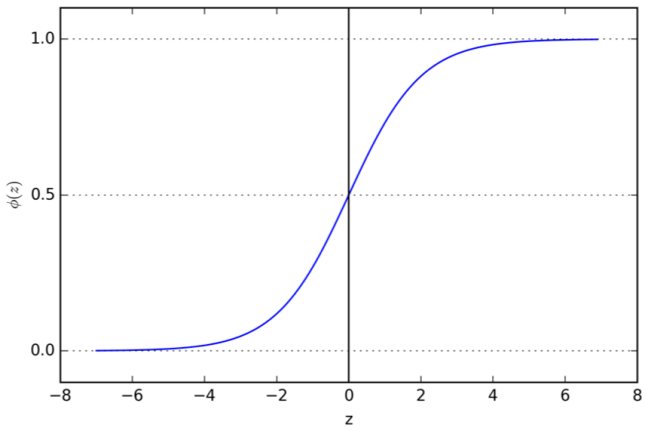
Overfitting: Logistic Regression

There is also a computational problem when trying to make something have probability 1.

- Risk of overflowing if weights get too large.

Recall the logistic function:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



z would have to be ∞ (or -∞) in order to make ϕ(z) equal to 1 (or 0)

Regularization

Regularization refers to the act of modifying a learning algorithm to favor “simpler” prediction rules to avoid overfitting.

Most commonly, regularization refers to modifying the loss function to **penalize** certain values of the weights you are learning.

- Specifically, penalize weights that are *large*.

Regularization

How do we define whether weights are *large*?

$$d(\mathbf{w}, \mathbf{0}) = \sqrt{\sum_{i=1}^{k} (w_i)^2} = \|\mathbf{w}\|$$

This is called the **L2 norm** of **w**

- A norm is a measure of a vector's length
- Also called the Euclidean norm

Regularization

New goal for minimization:

$$L(\mathbf{w}; \mathbf{X}) + \lambda \|\mathbf{w}\|^2$$

This is whatever loss function we are using (for a dataset $\mathbf{X}$)

Regularization

New goal for minimization:

$$L(\mathbf{w}; \mathbf{X}) + \underbrace{\lambda \|\mathbf{w}\|^2}$$

By minimizing this, we prefer solutions where **w** is closer to **0**.

Regularization

New goal for minimization:

$$L(\mathbf{w}; \mathbf{X}) + \lambda \|\mathbf{w}\|^2$$

Why squared? It eliminates the square root; easier to work with mathematically.

By minimizing this, we prefer solutions where **w** is closer to **0**.

Regularization

New goal for minimization:

$$L(\mathbf{w}; \mathbf{X}) + \underbrace{\lambda \|\mathbf{w}\|^2}$$

Why squared? It eliminates the square root; easier to work with mathematically.

By minimizing this, we prefer solutions where **w** is closer to **0**.

λ is a hyperparameter that adjusts the tradeoff between having low training loss and having low weights.

Regularization

Regularization helps the computational problem because gradient descent won't try to make some feature weights grow larger and larger…

At some point, the penalty of having too large $l\mathbf{w}l^2$ will outweigh whatever gain you would make in your loss function.

- In logistic regression, probably no practical difference whether your classifier predicts probability .99 or .9999 for a label, but weights would need to be much larger to reach .9999.

Regularization

This also helps with generalization because it won't give large weight to features unless there is sufficient evidence that they are useful

- The usefulness of a feature toward improving the loss has to outweigh the cost of having large feature weights

Regularization

More generally:

$$L(\mathbf{w}; \mathbf{X}) + \lambda R(\mathbf{w})$$

This is called the **regularization term** or **regularizer** or **penalty**

- The squared L2 norm is one kind of penalty, but there are others

λ is called the regularization **strength**

L2 Regularization

When the regularizer is the squared L2 norm $\|\mathbf{w}\|^2$, this is called L2 regularization.

- This is the most common type of regularization
- When used with linear regression, this is called *Ridge regression*
- Logistic regression implementations usually use L2 regularization by default
  - L2 regularization can be added to other algorithms like perceptron (or any gradient descent algorithm)

L2 Regularization

The function $R(\mathbf{w}) = \|\mathbf{w}\|^2$ is convex, so if it is added to a convex loss function, the combined function will still be convex.

L2 Regularization

How to choose λ?

- You'll play around with it in the homework, and we'll also return to this later in the semester when we discuss hyperparameter optimization.

Other common names for λ:

- *alpha* in sklearn
- *C* in many algorithms
  - Usually C actually refers to the inverse regularization strength, 1/λ
  - Figure out which one your implementation is using (whether this will increase or decrease regularization)

L1 Regularization

Another common regularizer is the L1 norm:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{k} |w_j|$$

- Convex but not differential when $w_j = 0$
  - But 0 is a valid subgradient for gradient descent
- When used with linear regression, this is called *Lasso*
- Often results in many weights being exactly 0 (while L2 just makes them small but nonzero)

L2+L1 Regularization

L2 and L1 regularization can be combined:

$$R(\mathbf{w}) = \lambda_2 \|\mathbf{w}\|^2 + \lambda_1 \|\mathbf{w}\|_1$$

- Also called *ElasticNet*
- Can work better than either type alone
- Can adjust hyperparameters to control which of the two penalties is more important

Feature Normalization

The scale of the feature values matters when using regularization.

- If one feature has values between [0, 1] and another between [0, 10000], the learned weights might be on very different scales – but whatever weights are "naturally" larger are going to get penalized more by the regularizer.

Feature **normalization** or **standardization** refers to converting the values to a standard range.

- We'll come back to this later in the semester.

Bias vs Variance

We learned about **inductive bias** at the start of the semester.

What exactly is **bias**?

Bias vs Variance

Remember: the goal of machine learning is to learn a function that can correctly predict all data it might hypothetically encounter in the world

- We don't have access to all possible data, so we approximate this by doing well on the *training data*
- The training data is a *sample* of the true data

Bias vs Variance

When you estimate a parameter from a sample, the estimate is **biased** if the expected value of the parameter is different from the true value.

The *expected value* of the parameter is the theoretical average value of all the different parameters you would get from different samples.

Example: random sampling (e.g. in a poll) is *unbiased* because if you repeated the sampling over and over, on average your answer would be correct (even though each individual sample might give a wrong answer).

Bias vs Variance

Regularization adds a bias because it systematically pushes your estimates in a certain direction (weights close to 0)

If the true weight for a feature should actually be large, you will consistently make a mistake by underestimating it, so on average your estimate will be wrong (therefore biased).

Bias vs Variance

The **variance** of an estimate refers to how much the estimate will vary from sample to sample.

If you consistently get the same parameter estimate regardless of what training sample you use, this parameter has low variance.

Bias vs Variance

Bias and variance both contribute to the error of your classifier.

- Variance is error due to *randomness* in how your training data was selected.
- Bias is error due to something *systematic*, not random.

Bias vs Variance

High bias

- Will learn similar functions even if given different training examples
- Prone to underfitting

High variance

- The learned function depends a lot on the specific data used to train
- Prone to overfitting

- Some amount of bias is needed to avoid overfitting.
- Too much bias is bad, but too much variance is usually worse.

Summary

Regularization is really important!

It can make a big difference for getting good performance. You usually will want to tune the regularization strength when you build a classifier.