Contents

Overfitting & Regularization

What is Overfitting?

- the model accurately remembers all training data, including noise and unrelated features
- failed to learn any meaningful pattern
- Reason
  - training data too small
  - model too complex
- Consequence
  - Such a model often performs badly on new test or real data that have not been seen before.
- how to avoid overfitting
  - cross-validation
  - regularization

Bias vs Variance

- bias and variance are 2 different sources of error in an estimator
- *bias-variance tradeoff*
  - to be able to "solve" a problem, we want our model to have enough degrees of freedom to resolve the underlying complexity of the data we are working with, but we also want it to have not too much degrees of freedom to avoid high variance and be more robust.
- bias

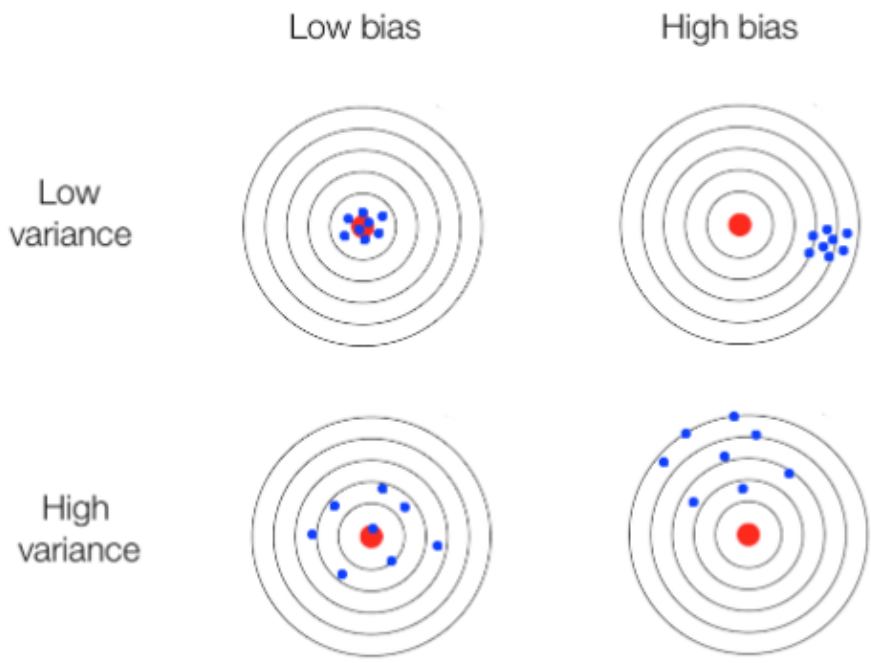

Figure 1: img

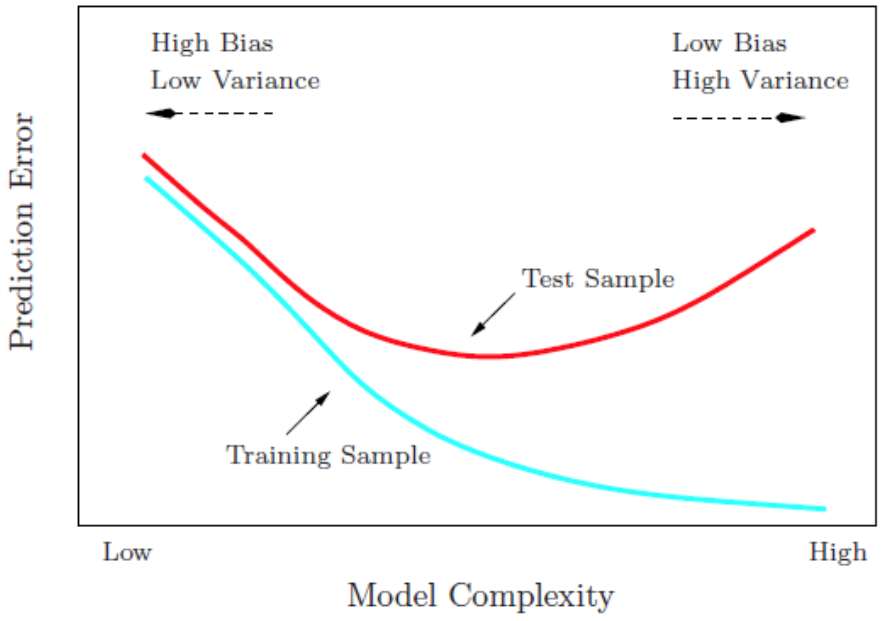

Figure 2: img

  - expected deviation from the true value of the parameter/function
  - $bias(\hat{\theta}) = E(\hat{\theta}) - \theta$
- variance
  - how much the estimates will vary as we independently re-sample the dataset from the underlying data generating process
  - standard error $sd = \sqrt{var(\hat{\theta})}$
  - too much degree of freedom?
- From the bias-variance trade-off, can you derive the bound on training set size?

Name	model fit	consequence	reason
bias	underfitting	bad accuracy	erroneous/simplistic assumption
variance	overfitting	don't generalize	too complicated, learned noise

What is Regularization?

- A theoretical justification for regularization is that it attempts to impose Occam's razor on the solution (as depicted in the figure above, where the green function, the simpler one, may be preferred).
- From a Bayesian point of view, many regularization techniques correspond to imposing certain prior distributions on model parameters.
- Ian Goodfellow: Any modification we make to the learning algorithm that is intended to reduce the generalization error, but not its training error
- Goal
  - discourages learning a more complex or flexible model, so as to avoid the risk of overfitting
  - ignores the background noise (data points that don't represent the true properties of your data but just random chance)
  - reduces variance of a model without substantially increase its bias
  - improves generalizability of a learned model
- λ

Weight Penalty

- assumption: a model with small weights is simpler than a model with large weights
- Lasso (L1)
  - Least Absolute Deviations: sum of absolute values of weights
  - more binary/sparse: a dumber model (simpler pattern)
    - sparse means the majority of components are zeros
    - L1 tends to drive some weights to exactly 0 while allowing some to be big
    - built-in feature selection
  - corresponds to setting a Laplacean prior on the terms
  - computationally inefficient on non-sparse cases
- Ridge (L2)
  - sum of squared weights
  - spread error among all the terms
  - tends to drive all the weights to a smaller value
  - corresponds to a Gaussian prior
  - computationally efficient due to having analytical solutions
  - need to standardize the variables
- L1 vs L2
  - the shape formed by all points whose L1 norm equals to the same constant `c` has many corners/tips/spikes that happen to be sparse (lays o one of the axises of the coordinate)
  - and if you grow this shape to touch the solution we find for our problem (a surface or a cross-section), the probability that the touch point of the 2 shapes is at one of the corners is very high
  - L_p norm is more sharp when $0 < p < 1$ but it's computationally challenging
  - example: consider there are 2 parameters β_1 and β_2, the lasso regression is expressed by $|\beta_1| + |\beta_2| \leq s$ and the ridge regression is $\beta_1^2 + \beta_2^2 \leq s$
  - the green areas are the constraint (feasible regions for lasso regression and ridge regression respectively)
  - the red ellipses are the contours for the least squares error function or RSS (residual sum of squares) in terms of parameters β_1 and β_2.
    - points on the same ellipse share the same RSS value
    - The value increases as the red ellipses out expand.
  - without constraint/regularization, the error function is minimized at $\hat{\beta}$ (MLE) -> the unconstrained least squares estimate
  - when s is really big, the green regions can contain the center of the ellipse, making the same prediction of $\hat{\beta}$
  - Heuristically, for each method, we are looking for the intersection of the red ellipses and the blue region as the objective is to minimize the error function while maintaining the feasibility.
  - Lasso constraint has corners at each of the axes so the intersection

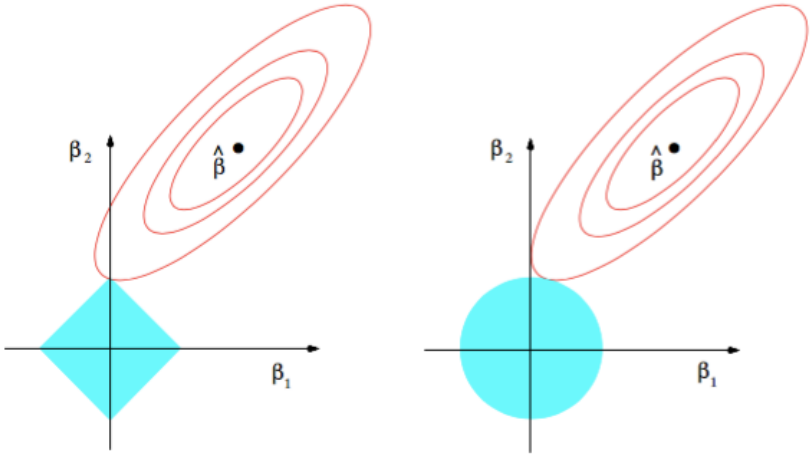

FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Figure 3:

often happen at an axis, and one of the coefficient will equal 0.
  - But the intersection with Ridge will not generally occur on an axis (non-zero).
  - L1 constraint is more likely to produce an intersection that has one component of the solution is zero (i.e., the sparse model) due to the geometric properties of ellipses, disks, and diamonds. It is simply because diamonds have corners (of which one component is zero) that are easier to intersect with the ellipses that extending diagonally.
- Elastic-net
  - a mix of both L1 and L2 regularizations.
  - A penalty is applied to the sum of the absolute values and to the sum of the squared values.

Dataset Augmentation

- creating synthetic data for training
- expand the dataset but reflect real world variations
- images
  - translating the picture a few pixels, rotation, scaling
  - add random negative examples (unrelated pictures)

Early Stopping

- stops training once performance on validation gets worse
- tuning epochs/steps

Figure 4:

Dropout

- removing some random neurons while training
  - with all their incoming and outgoing connections
  - hidden or input layer
- why dropout works
  - neurons become more insensitive to the weights of other nodes
  - less co-adaptive
  - more robust
  - analogy: other people take over your work when you are on vacation
  - or can be viewed as an ensemble technique that averages multiple models
  - training a collection of 2^n thinned networks with parameter sharing
  - bagging
- hyperparameter: p as the probability of keeping a unit
  - typical value $p \geq 0.5$

    ```
    hidden = tf.nn.dropout(hidden, 0.5, seed=SEED
    ```
- and adding them back during backpropagagtion?
- How will you implement dropout during forward and backward pass?
- all neurons are used for prediction

Dense-Sparse-Dense Training

1. Perform initial regular training, but with the main purpose of seeing which weights are important, not learning the final weight values.
2. Drop the connections where the weights are under a particular threshold. Retrain the sparse network to learn the weights of the important connections.
3. Make the network dense again and retrain it using small learning rate, a step which adds back capacity.

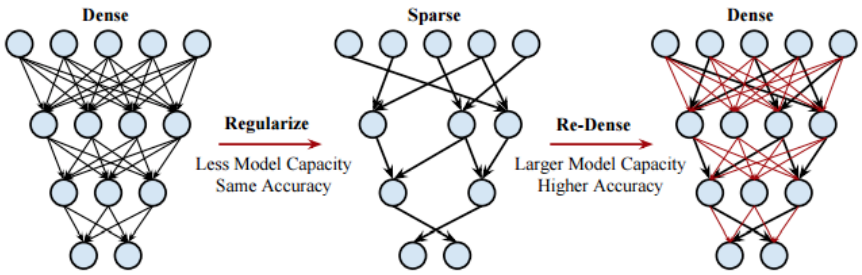

Figure 5:

Pruning

Loss Function

- loss function: individual training example
- cost function: sum for all examples
- objective function:

L1 Loss	L2 Loss
robust	not very robust
stable solution	unstable solution
always 1 solution	possibly multiple solutions

- robust: resistant to outliers in data
- stable
  - L2: for any small adjustment of a data point, the regression line will always move only slightly; that is, the regression parameters are continuous functions of the data.
  - L1: for a small horizontal adjustment of a datum, the regression line may jump a large amount. The method has continuous solutions for some data configurations; however, by moving a datum a small amount, one could "jump past" a configuration which has multiple solutions that span a region. After passing this region of solutions, the least absolute deviations line has a slope that may differ greatly from that of the previous line.
- Mean Squared Error (MSE)
  - linear regression
- cross-entropy loss
  - measures the divergence between 2 probability distribution
  - `H(P, Q) = -sum(P(x)logQ(x)))`
  - `P` is the distribution of the true labels
  - `Q` is the probability distribution of the predictions
  - binary classification: `-ylog(p) + (1-y)log(1-p)`
  - multi-class classification: average cross entropy across all examples

L1-loss vs. L2-loss and L1-regularization vs. L2-regularization

- use L1-norm or L2-norm either as a loss function or a regularization term
- L1 loss: Least Absolute Deviations (LAD), Least Absolute Errors (LAE)
- L2 loss: Least Square Errors (LSE)
- L1 regularization: sum of absolute weights
- L2 regularization: sum of squared weights

$$S = \sum_{i=1}^{n} |y_i - f(x_i)|.$$

Figure 6:

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Figure 7:

Norm

Definition

- total size or length of all vectors in a vector space or matrices
- used as a distance measure or loss function or regularization term
- $\|x\|_p = \sqrt[p]{\sum_i \|x_i\|^p}$ where $p \in \mathbb{R}$
  - x is a vector or a matrix
  - p-th-root of a summation of all elements to the p-th power
  - there is a norm for every single *real number*
  - not just integers
  - very different math properties -> different applications

Types

- L0 norm
  - $\|x\|_0 = \sqrt[0]{\sum_i \|x_i\|^0}$
  - not actually a norm, a cardinality function (a measure of the "number of elements of the set")
  - zeroth-power, zeroth-root
  - use $\|x\|_0 = \#(i|x_i \neq 0)$ instead
    - number of non-zero elements in a vector
  - L0 optimization problem
    - $min\|x\|_0$ subject to $Ax = b$
    - try to find the sparsest solution of the under-determined linear system -> lowest L0 norm
    - Compressive Sensing scheme
    - NP-hard
    - sometimes relax to become L1 or L2 optimization
- L1 norm == Manhattan norm
  - on one vector: $\|x\|_1 = \sum_i |x_i|$

  - on the difference between 2 vectors:

$$SAD(x_1, x_2) = \|x_1 - x_2\|_1 = \sum_i |x_{1i} - x_{2i}|$$

    * Sum of Absolute Difference (SAD)
  - Mean Absolute Error (MAE)

$$MAE(x_1, x_2) = \frac{1}{n}\|x_1 - x_2\|_1 = \frac{1}{n}\sum_i |x_{1i} - x_{2i}|$$

    where n is the size of x
  - Optimization
    * $min\|x\|_1$ subject to $Ax = b$
    * not a smooth function -> hard to solve
    * convex optimization
- L2 norm == Euclidean norm == Euclidean distance
  - on one vector:

$$\|x\|_2 = \sqrt{\sum_i |x_i^2|}$$

  - on vector difference:

$$SSD(x_1, x_2) = \|x_1 - x_2\|_2^2 = \sum_i |(x_{1i} - x_{2i})^2|$$

    * Sum of Squared Difference (SSD)
  - Mean Squared Error (MSE)

$$MSE(x_1, x_2) = \frac{1}{n}\|x_1 - x_2\|_2^2 = \frac{1}{n}\sum_i |(x_{1i} - x_{2i})^2|$$

    * similarity between 2 signals
    * quality
    * correlation between 2 signals
  - Optimization
    * $min\|x\|_2$ subject to $Ax = b$
    * using a Lagrange multiplier trick, the optimal solution for x is: $A^T(AA^T)^{-1}b = A^+b$
      · Moore-Penrose Pseudoinverse
    * Least Square problem
    * easy to compute
    * but curve is too smooth to find a single best solution
- L-infinity norm
  - $\|x\|_\infty = \sqrt[\infty]{\sum_i |x_i^\infty|}$
  - say x_j is the highest entry in vector x, $x_j^\infty \gg x_i^\infty \forall i \neq j$
  - then $\sum_i |x_i^\infty| = X_j^\infty$
  - then $\|x\|_\infty = \sqrt[\infty]{x_j^\infty} = |x_j|$
  - so $\|x\|_\infty = max(|x_i|)$
  - l_∞ norm is the maximum entries magnitude of that vector