CS168: The Modern Algorithmic Toolbox
Lecture #6: Regularization

Tim Roughgarden & Gregory Valiant*

April 18, 2018

1 The Context and Intuition behind Regularization

Given a dataset, and some class of models that you are trying to fit (e.g. linear classifiers, linear regressions, neural networks, etc.), *regularization* is a general technique whereby you form some set of preferences among the models/hypotheses in the chosen class, and end up returning a model that takes into account both the extent to which the model fits the data, as well as how "preferred" the model is. To give a concrete example, which we will revisit, suppose we have n datapoints $x_1, \ldots, x_n \in \mathbb{R}^d$, and corresponding real-valued labels $y_1, \ldots, y_n \in \mathbb{R}$ and wish to fit a linear regression. The following is the ℓ_2 *regularized least-squares objective function* corresponding to a model $a \in \mathbb{R}^d$:

$$f(a) = \sum_{i=1}^{n} (\langle a, x_i \rangle - y_i)^2 + \lambda \|a\|_2^2,$$

where $\|z\|_2^2$ denotes the squared Euclidean length of the vector z, and the parameter λ dictates how much the models with small ℓ_2 norm should be prioritized. Note that in the case that λ is set to 0, this objective is simply the usual least-squares objective function, and as λ inceases, models a with small length will be heavily prioritized.

Many of the most common forms of regularization can be viewed as prioritizing different notions of "simple" models. As we discuss below, ℓ_2 regularization—preferring models proportionately to the sum of the squares of the coefficients of the model, as well as sparsity-inducing models that favor sparse sets of coefficients (e.g. learning a linear model where most of the coefficients are 0), are very common and effective forms of regularization. In these notes, we will discuss explicit regularizers where we add some specific regularization term (such as the ℓ_2 term above) to an objective function, capturing both how well the model fits the data as well as how preferred the model is. On the homework, you will explore an

implicit regularizer—where the *algorithm* you use to find a model that fits the data (e.g. stochastic gradient descent) implicitly prefers some hypotheses over others. In general, we are just beginning to understand implicit regularizers. This is currently a very active area of research in the machine learning and theory communities.

Why Regularize? What is wrong with always returning the *empirical risk minimizer* (ERM) discussed in last lecture? [This is the model that fits the data the best—in the case of least-squares regression, it will minimize the squared error over your training set.] If your dataset is extremely large in relation to the expressivity of the model class you are fitting, then the (ERM) will likely be extremely good. The interesting case is where you are flirting with a data-limitation—when you are trying to fit an expressive model class using an amount of data that seems close to, or below the amount of data that you would need for the ERM to *generalize*. In such cases, it is intuitively clear that the ERM will be overfitting to the training data. The idea of regularization is that if you have any additional information, or preferences about the form of the true underlying model, these can be applied as 'regularizers' and might ameliorate this overfitting.

2 Increasing Dimensionality: the Polynomial Embedding and Random Non-Linear Features

At this point, it might seem like the best path is to always ensure that the model class you are considering is sufficiently simple in relation to the size of your dataset, so as to guarantee that the ERM generalizes. From last lecture, in the case of learning a linear separator, this would mean that if you have n datapoints, you would only try to learn a linear model on $< n$ features. Additionally, you might be wondering how useful regularization actually is, if it is only useful in the narrow regime where the number of features is near the threshold of where generalization might cease to hold. (Given all of this data that we have today, surely in most cases we have more than enough data to guarantee generalization for the models we might train, no?)

One viewpoint is that you *always* want to be flirting with this dangerous line between overfitting and generalization. If you are not near this line, then you should consider a larger, more expressive class of models. For example if you want to learn a linear model using a dataset with n datapoints, with each point consisting of d features, if $n \gg d$, then you should add features until the new dimensionality, d' is close to n. The intuition behind this "more-is-better" philosophy stems from the view that in most settings, there *is* a very-complex model underlying the data. Given this view, if you have enough data to learn a very expressive model class, but instead only try to learn a simpler class of models, then you are leaving something on the table. Of course, for some problems, you might know that the true model is likely extremely simple, in which case there is no need to have much data....

Given n datapoints in $\mathbb{R}^d$ for $d \ll n$, one general approach to formulating an expressive/large model class is to *increase* the dimensionality of each datapoint, and then just learn a linear classifier or regression in this larger-dimensional space. This embedding of

the data into a higher dimensional space can be thought of as adding additional "features". Below we outline two common types of such embeddings, the polynomial embedding, and a random non-linear embedding.

2.1 The Polynomial Embedding

One way to increase the dimensionality of a datapoint $x = (x_1, \ldots, x_d)$ is to add a number of "polynomial features". For example, we could add $\binom{d}{2}$ coordinates, corresponding to the $\binom{d}{2}$ products of pairs of coordinates of the original data—this is the quadratic embedding. Specifically, this embedding, $f : \mathbb{R}^d \to \mathbb{R}^{d+d(d+1)/2}$ is defined as

$$f(x) = (x_1, \ldots, x_d, x_1^2, x_1x_2, x_1x_3, \ldots, x_dx_{d-1}, x_d^2).$$

It should be intuitively clear that a linear function in this quadratic space is more expressive than a linear function in the original space. In the case of the quadratic embedding, a linear function corresponds exactly to a quadratic function in the original space. For example, a linear function in this new space would be capable of modeling functions $g(x) = (x_1 + x_2 + x_{17})^2$.

How do you train linear classifiers in these polynomial spaces? One option is to actually map your data into this new space, and then proceed as usual for learning a linear model. Perhaps surprisingly, it is actually possible to do this polynomial embedding (and others) *implicitly* without actually writing out this higher-dimensional embedding of your data. This is sometimes extremely useful—-imagine if your initial data was 1000 dimensional, you would rather not write out the 500,000 dimensional quadratic embedding of the data if you didn't need to. This ability to train on such embeddings implicitly is known as *kernelization* and is covered in the context of Support Vector Machines in most ML classes.

2.2 A Random Projection plus Nonlinearity

A different method for increasing the dimensionality of data is motivated by the Johnson-Lindenstrauss transformation that we saw last week. Consider the mapping $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ defined by choosing a $d' \times d$ matrix M by selecting each index independently from a standard Gaussian [Normal distribution], and then defining $f(x) = Mx$. This is exactly the Johnson-Lindenstrauss dimension reduction transformation, only now we are choosing the new dimensionality, $d' > d$.

This embedding increases the dimension of the data. Nevertheless, if we learn a linear function $h : \mathbb{R}^{d'} \to \mathbb{R}$ in this d'-dimensional space, and apply it by first mapping a point x to Mx then applying h to Mx, the linear functions compose, and we have done nothing more than apply a linear function in the original d-dimensional space!!! [Think about this for a minute—imagine $d = 1$, and $d' = 2$, suppose we map $x \to (3x, 7x)$, and then apply a linear function in the 2-dimensional space—-this is just a linear function of the original space!]

In order for linear functions in this higher-dimensional space to have any additional expressivity, we must perform a non-linear operation. It turns out, almost any non-linear

function, applied to each coordinate of Mx, will suffice, and often performs well in practice. (For example, square each coordinate, take the absolute value, take the square-root, apply a sigmoid function, etc.)

One motivation for this linear projection to a higher-dimensional space, followed by a nonlinear operation, is that this is exactly the computation performed by the bottom layer of a neural network. The linear projection (the matrix M above) is the way the first hidden layer is computed from the inputs, and then an element-wise nonlinear function is applied to each of the resulting internal nodes.

2.3 Comparing Polynomial Embedding vs Random Projection + Nonlinearity

For a given domain, the question of whether to use a polynomial embedding versus the random projection followed by nonlinearity largely comes down to whether you wish to have a basis-dependent embedding or not. If the features of your input space are meaningful, then the polynomial embedding might make most sense. For example, if you are trying to learn physical laws based on data, and your features represent actual quantities, mass, velocity, time, etc, it seems most reasonable to preserve the interpretability of the new features, e.g. a new feature that is the product of mass and velocity. On the other hand, if your features are not all that distinct, or the coordinate basis was chosen arbitrarily, then the rotationally invariant random projection approach might make more sense.

3 Regularization: The Bayesian and Frequentist viewpoints

There are two general ways of formalizing why regularization makes sense: the "Bayesian view", and the "frequentist view".

3.1 The Bayesian View and ℓ_2 Regularization

In the Bayesian view, we assume that the true model underlying the data is drawn from some known prior distribution. Given such a distributional view you can evaluate the *likelihood* of a given model, and may be justified in trying to output the model with maximum likelihood.

To illustrate, suppose $x_1, \ldots, x_n \in \mathbb{R}^d$ are fixed, and we assume that the universe has generated labels $y_1, \ldots, y_n$ by first picking a vector $a^* \in \mathbb{R}^d$ of coefficients by choosing each coordinate independently from a Gaussian of mean 0 and variance σ^2, $N(0, \sigma^2)$, and then assigns labels $y_i = \langle x_i, a^* \rangle + z_i$, where z_i is some noise that has also been chosen independently from a standard Gaussian. (This is referred to as assuming a Gaussian "prior" distribution over the model, a.)

Given $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, the likelihood of a linear model, a, is simply the product $Pr[a] \cdot Pr[data|a]$. Plugging in our distributional assumptions about the likelihood of a and

the probability of the data given a, we obtain the following:

$$Likelihood(a) = \prod_{i=1}^{d} e^{-\frac{a(i)^2}{2\sigma^2}} \prod_{i=1}^{n} e^{-\frac{(\langle a, x_i \rangle - y_i)^2}{2}} = e^{-(1/2)\sum_{i=1}^{n}(\langle a, x_i \rangle - y_i)^2 - \frac{1}{2\sigma^2}\|a\|_2^2}.$$

Hence the maximum likelihood model a is precisely the minimizer of the exponent, which is the ℓ_2 regularized least-squares objective (with $\lambda = 1/\sigma^2$).

Of course, the big caveat with the Bayesian view is that it is often unclear the extent to which such Bayesian priors actually hold. [And, it is worth noting that different priors on the model, a, will give rise to rather different regularizers.]

3.2 The Frequentist View and ℓ_0 Regularization

In contrast to the Bayesian viewpoint where one typically posits a distribution from which the true model is drawn, the "Frequentist" approach to justifying regularization is to argue that if the true model has a specific property, then regularization will allow you to recover a good approximation to the true model. One frequently encountered type of structure that a linear model might possess is sparsity—perhaps the d-dimensional vector of coefficients, a, only has $s \ll d$ nonzero coordinates. One hope is to design a regularizer that will "prefer" sparse models, thereby allowing us to accurately recover sparse models even when the amount of available data is significantly less than what would be required to learn a general (dense) linear model.

Example 3.1 Suppose we have a bunch of data taken from observing some physical experiment—for example data of a ball's trajectory as it is repeatedly thrown. If our features capture natural quantities, e.g. mass, velocity, temperature, acceleration, etc., it is likely that the true underlying model might be sparse.

Example 3.2 Suppose we have a bunch of social, climate, geographic, and economic data for each county in the US, and are trying to learn a model to predict the frequency of certain disease outbreaks. It is possible that the "true" underlying model is dense; nevertheless, there is likely to be a relatively sparse model that is fairly accurate. From the perspective of trying to guide policy decisions, or recommend measures that could be taken to reduce the risk of such outbreaks, it is often more helpful to learn sparse models, than more opaque and uninterpretable dense models. (Of course, there is still the hairy issue of correlation vs. causation, but at least in a sparse model there are fewer factors to disentangle.)

3.3 ℓ_0 regularization

If we know that we are looking for a sparse model, perhaps the most natural regularizer is simply to penalize vectors in proportion to how (non)-sparse they are. This is referred to as an ℓ_0-regularizer (since the ℓ_0 norm of a vector is the number of nonzero entries). The ℓ_0

regularized least squares objective function would be:

$$f(a) = \sum_{i=1}^{n} (\langle a, x_i \rangle - y_i)^2 + \lambda \|a\|_0. \tag{1}$$

From an information theoretic standpoint, this objective function is great! Recall from last lecture that, in d dimensional space, any linear classifier can be approximated by an element of some finite set of $exp(d)$ linear functions. If, however, we know that we are looking for a linear classifier, $a \in \mathbb{R}^d$ in d dimensional that has sparsity level s, that is $\|a\|_0 = s$, then the size of the set of functions that represent such functions will "only" be $O\left(\binom{d}{s} exp(s)\right)$. If $s \ll d$, this is *much* smaller than $exp(d)$. Hence by the union-bound argument of the previous lecture, it will suffice to only have an amount of data that scales as $\log\left(\binom{d}{s} exp(s)\right) = O(s \log d)$, as opposed to scaling linearly with the dimension, d.

The main issue with directly trying to find a sparse model is that the the ℓ_0 norm is highly discontinuous; perturbing a coordinate of a by some infinitesimally small amount can change $\|a\|_0$ by ± 1. For this reason, it is computationally intractable to work directly with the ℓ_0 norm, and usual optimization approaches, such as gradient descent, will generally fail to optimize objective functions such as that of Equation 1.

3.4 ℓ_1 as a computationally tractable proxy for ℓ_0

Both in theory, and in practice, one can often use the ℓ_1 norm as a proxy for the ℓ_0 norm. The ℓ_1 norm of a vector is simply the sum of the absolute values of the coordinates, and hence it is continuous (and linear). The ℓ_1-regularized least-squares objective function, which is the ℓ_1 analog of Equation 1 is:

$$f(a) = \sum_{i=1}^{n} (\langle a, x_i \rangle - y_i)^2 + \lambda \|a\|_1. \tag{2}$$

From a computational standpoint, ℓ_1 regularization does not have the pitfalls of ℓ_0-regularization. Even though the ℓ_1 norm is not differentiable everywhere (because of the absolute values, it is not differentiable at points where any of the coordinates are 0), it is still amenable to gradient-descent, and other optimization approaches. As we will discuss towards the end of the course, it can also be represented as the objective function of a "linear program", and hence can be optimized in polynomial time.

Perhaps surprisingly, from an information theoretic perspective, ℓ_1 regularized regression, and linear classification, enjoys many of the properties of (the computationally intractable) ℓ_0 regularization. The following proposition, which we will return to towards the end of the class, provides one such example:

Proposition 3.3 (e.g. Theorem 1 of [1]) *Given n independent Gaussian vectors $x_1, \ldots, x_n \in \mathbb{R}^d$, and consider labels $y_i = \langle a, x_i \rangle$ for some vector a with $\|a\|_0 = s$. Then the minimizer of the ℓ_1 regularized objective function, Equation 2, will be the vector a, with high probability, provided that $n > c \cdot s \log d$, for some absolute constant c.*

4 Recap of Key Points

1. *Regularization* provides one method for combatting over-fitting in the data-poor regime, by specifying (either implicitly or explicitly) a set of "preferences" over the hypotheses.

2. Regularization is often quite effective, and theoretical explanations for this generally come in two flavors: "Bayesian" arguments in which we make an assumption about the distribution from which the universe picks the model in question, and "frequentist" arguments in which we assume that the true model satisfies some property (e.g. sparsity), and argue that we will recover such a model (possibly with high probability, given some assumptions about the data).

3. Explicit regularization can be accomplished by adding an extra regularization term to, say, a least-squares objective function. Typical types of regularization include ℓ_2 penalties, and ℓ_1 penalties.

4. ℓ_1 regularization should be thought of as a computationally tractable proxy for ℓ_0 regularization, which will favor sparse solutions. When trying to recover a s-sparse model over d dimensional data, the amount of data required to avoid overfitting typically scales as $O(s \log d)$, rather than $O(d)$, as would be the case for fitting linear models without any sparsity or regularization.

References

[1] E. Candes, M. Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 25.2 (2008): 21-30.