CS 189 Introduction to Machine Learning
Fall 2017 Note 2

1 Feature Engineering

We've seen that the least-squares optimization problem

$$\min_{\vec{x}} \|A\vec{x}-\vec{b}\|_2^2$$

represents the "best-fit" *linear* model, by projecting $\vec{b}$ onto the subspace spanned by the columns of A. However, sometimes we would like to fit not just linear models, but also *nonlinear* models such as ellipses and cubic functions. We can still do this under the framework of least-squares, by augmenting (in addition to the raw features) new arbitrary features to the data. Note that the resulting models are still linear w.r.t the augmented features, but they are nonlinear w.r.t the original, raw features.

1.1 Example: Fitting Ellipses

Let's use least-squares to fit a model for data points that come from an ellipse.

1. **Applications and Data**: We have n data points (x_i, y_i), which may be noisy (could be off the actual orbit). Our goal is to determine how the x_i's determine the y_i's.
2. **Model**: Model ellipse in the form $a_0 + a_1x^2 + a_2y^2 + a_3xy + a_4x + a_5y = 1$. There are 6 unknown coefficients (or 'weights').
3. **Optimization Problem**: We formulate the problem with least-squares as before:

   $$\min_{\vec{a}} \|A\vec{a}-\vec{b}\|_2^2$$

   Since every single data point has the same prediction value of 1, we can represent $\vec{b} \in \mathbb{R}^n$ as all 1's. Each coefficient a_i can be interpreted as the weight of the i'th feature. In expanded form, the optimization problem can be formulated as the following:

   $$\min_{a_0,a_1,a_2,a_3,a_4,a_5} \| \begin{bmatrix} 1 & x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 \\ 1 & x_2^2 & y_2^2 & x_2y_2 & x_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^2 & y_n^2 & x_ny_n & x_n & y_n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_5 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \|^2$$

   Specifically:

   - Each column represents a feature

- Each row represents a data point
- A **feature** is a building block of a model
- Assuming the columns are linearly independent, we can fit the data with OLS
- Augmenting more features in addition to the raw features makes the columns less and less linearly independent, making OLS more and more infeasible

1.2 Model Notation

A model has the following form:

$$y = \sum_{i=1}^{P} \alpha_i \Phi_i(\vec{x})$$

- y is the prediction from the model (note that y is a linear combination of the features)
- Φ_i represents the i'th feature - which depends on the raw features of the data point x
- α_i is the fixed coefficient/weight corresponding to the i'th feature

1.3 Polynomial Features

There are many different kinds of features, and an important class is **polynomial features**. Polynomial features can be interpreted as polynomials of the raw features.

Remember that polynomials are merely linear combination of monomial basis terms. Monomials can be classified in two ways, by their degree and dimension:

degree $\rightarrow$	0	1	2	3	$\cdots$
$\downarrow$ **dimension**					
univariate	1	x	x^2	x^3	$\cdots$
bivariate	1	x_1, x_2	x_1^2, x_2^2, x_1x_2	$x_1^3, x_2^3, x_1^2x_2, x_1x_2^2$	$\cdots$

Why are polynomial features so important? Note that under **Taylor' Theorem**, any sufficiently smooth function can be approximated arbitrarily closely by a high enough degree polynomial. This is true for both univariate polynomials and multivariate polynomials, giving them the title of **universal approximators**. In this sense, polynomial features are building blocks for many kinds of functions.

One downside of polynomials is that as their degree increases, they increase exponentially in the number of terms. That is, for a polynomial of degree at most d in l dimensional space,

$$\text{\# of terms} = O(l^d)$$

The justification is as follows: there are d "slots" (corresponding to the degree), and l "choices" for each slot (corresponding to which of the l original "raw" features will be used).

Due to their exponential number of terms, polynomial features suffer from the curse of dimensionality. Note however that we can bypass this issue by employing kernels, allowing us to even deal with infinite-diemsional polynomials! More on this later in the course.

2 Training vs. True Error

One question when using polynomial features is how do we choose the right degree for the polynomial? In order to answer this question we would idealy measure the prediction error of our model for different choices of the degree and choose the degree that minimizes the error.

Here we must make a distinction between training error and true error. **Training error** is the prediction error when the model is evaluated on the training data, in other words the same data that was used to train the model is being used to evaluate it. **True error** is the prediction error when the model is evaluated on unseen data that comes from the true underlying model.

For different choices of the degree, let's observe what happens to the training error versus the true error. We observe that as degree increases, both training error and true error decrease initially due to using a more complex model. Training error continues to always decrease as the degree increases. This is true, because as the degree increases, the polynomial becomes strictly more expressive, fitting the training data more accurately. However, the true error eventually increases as the degree increases. This indicates the presence of **overfitting**: the model is too complex, and it is fitting the noise too much, leading to a low training error but high true error. Overfitting highlights that getting a better fit for your training data does not mean that you've learned the correct model.

3 Ridge Regression

3.1 Motivation

While OLS can be used for solving linear least squares problems, it falls short for two reasons: numerical instabilty and overfitting.

Numerical Instability: Numerical instability arises when the features of the data are close to collinear (leading to linearly dependent feature columns), causing the feature matrix A to have singular values that are either 0 or very close to 0. Why are small singular values bad? Let us illustrate this via the singular value decomposition (SVD) of A:

$$A = U\Sigma V^T$$

Now let's try to expand the $(A^TA)^{-1}$ term in OLS using the SVD of A:

$$(A^TA)^{-1} = (V\Sigma U^T U\Sigma V^T)^{-1} = (V\Sigma(I)\Sigma V^T)^{-1} = (V\Sigma^2 V^T)^{-1} = (V^T)^{-1}(\Sigma^2)^{-1}V^{-1} = V\Sigma^{-2}V^T$$

This means that $(A^TA)^{-1}$ will have singular values that are the squared inverse of the singular values of A, leading to either extremely large or infinite singular values (when the singular value of A is 0). Such excessively large singular values can be very problematic for numerical stability purposes.

Overfitting: Overfitting arises when the features of the A matrix are too complex and prevent OLS from producing a model that generalizes to unseen data. We want to be able to still keep these complex features, but somehow penalize them in our objective function so that we do not overfit.

3.2 Solution

There is a very simple solution to both of these issues: penalize the entries of $\vec{x}$ from becoming too large. We can do this by adding a penalty term constraining the norm of $\vec{x}$. For a fixed, small scalar λ, we now have:

$$\min_{\vec{x}}(\|A\vec{x}-\vec{b}\|^2+\lambda^2\|\vec{x}\|^2)$$

Note that the λ in our objective function is a **hyperparameter** that measures the sensivity to the values in $\vec{x}$. Just like the degree in polynomial features, λ is a value that we must choose arbitrarily. Let's expand the terms of the objective function:

$$\begin{aligned}\|A\vec{x}-\vec{b}\|^2+\lambda^2\|\vec{x}\|^2 &= (A\vec{x}-\vec{b})^T(A\vec{x}-\vec{b})+\lambda^2\vec{x}^T\vec{x}\\ &= \vec{x}^TA^TA\vec{x}-\vec{b}^TA\vec{x}-\vec{x}^TA^T\vec{b}+\vec{b}^T\vec{b}+\lambda^2\vec{x}^T\vec{x}\end{aligned}$$

Finally take the gradient of the objective and find the value of $\vec{x}$ that achieves 0 for the gradient:

$$\begin{aligned}0 &= 2\vec{x}(A^TA+\lambda^2I)-2\vec{b}^TA\\ \vec{x} &= \vec{b}^TA(A^TA+\lambda^2I)^{-1}\\ \vec{x} &= (A^TA+\lambda^2I)^{-1}A^T\vec{b}\end{aligned}$$

The technique we just described is known as **ridge regression**, aka **Tikhonov regularization**. Note that now, the SVD of $(A^TA+\lambda^2I)^{-1}$ becomes:

$$(A^TA+\lambda^2I)^{-1}=(V\Sigma^2V^T+\lambda^2I)^{-1}=(V\Sigma^2V^T+V\lambda^2IV^T)^{-1}=(V(\Sigma^2+\lambda^2I)V^T)^{-1}$$

Now with our slight tweak, the singular values have become $1/(\sigma^2+\lambda^2)$, meaning that even if $\sigma=0$, the singular values are guaranteed to be at least $1/\lambda^2$, solving our numerical instability issues. Furthermore, we have partially solved the overfitting issue. By penalizing the norm of $\vec{x}$, we encourage the weights corresponding to relevant features that capture the main structure of the true model, and penalized the weights corresponding to complex features that only serve to fine tune the model and fit noise in the data.