EECS 189 Project T Note: Robustness and Dimensionality Reduction

Abhinav G., William M., Grace K., Shrey V., Jai B.
Department of Computer Science
University of California, Berkeley
Berkeley, CA 94704
(abhinavg, wmceachen, gracekull21, shreyv, jaibansal) @berkeley.edu

Abstract

WHAT WILL WE COVER: About a page per section

1 Robustness in Data Science

"Garbage In, Garbage Out" is a common adage within the data science community. As the saying suggests, low quality data can produce an inaccurate model. For example, if labeled data was suddenly negated due to an unforeseen technical error, a new model would make predictions exactly opposite to the truth. Therefore, it is critical for data scientists to spend time cleaning their data. This process can take up to 90% of data scientists' time. We'll cover potential sources for data error, and solutions for addressing these errors.

1.1 Type Conversions

A common error in data is issues surrounding types. For example, suppose a SQL table contains the date of an observation measured in UNIX time, the number of seconds since January 1, 1970. If this table is written in SQLite, which does not have a specified time type, then new data could follow a different format. If new data was instead the number of days since 2000, any model assuming it was in UNIX time would make incorrect predictions! In order to address these kinds of errant types, there are several options. One solution is to rely on strict type checking, which guarantees that new data is correctly formatted (albeit potentially erroneous in other ways). However, where strict types are not possible to implement, it is critical to build a data cleaning pipeline that can handle different input formats and convert them to a consistent output. In our date example, if new date data were strings of "MM-DD-YYYY", then a pipeline could use string-to-date functions to ensure that the date feature was properly formatted.
Type conversion is an issue even within primitive types. For example, CSV parsing can lead to numeric values read as strings. Additionally, Boolean casting, in which a Boolean column is converted into a set of binary values, could lead to Boolean values having numeric operations performed on them. In both these cases, translating dirty data without access to the original data generation function is a challenge.

1.2 Debugging

In a commercial environment, technical failure is expensive. For example, if an ad auctioning algorithm crashes, it is critical that the process to identify and address issues within the code is straightforward, reproducible, and reliable. If an on-call engineer needs to fix a problem at 2 AM, they can fix issues more quickly if code is well documented, well tested, and well reviewed. Rather than relying on in-line print statements, a thorough test suite that addresses all edge cases is necessary.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

1.3 Diagnostics

Whenever the data generation functionality and data processing functionality are written independent of one another, data processing is necessary. For example, if the team responsible for "filling the data lake", i.e generating the raw data that is subsequently processed and trained with, changes their approach, data processing teams will need to appropriately address these changes. These data generation changes could be minor, e.g swapping column order, or substantial, e.g a unit change (from Celsius to Fahrenheit, for example). Regardless of the severity, having performance dashboards can help notify data scientists of these changes before newly trained models are sent into production. If model performance drops substantially, it is likely that data generation functionality has changed and data processing has not accounted for these changes accordingly. Thus, creating a dashboard for models acts as a late-stage test suite: if all code works, but models don't, diagnostics dashboards can help.
Diagnostics dashboards are also useful for addressing technical issues. By tracking the causes and solutions to technical failure, e.g a server shutting down, these issues can be addressed quickly. In addition, this process can help highlight potential technical errors: if a certain data generation/processing function is shown to be at the root of the failures, it can be handled appropriately.

2 Motivating Low Rank Approximation

Data generation is often imprecise. When working with physical sensors, their data collection mechanisms may have some allowed degree of noise, for example. When this occurs, it is preferable to work with a lower dimension version of the data. We want our lower dimension approximation to maintain as much of the variance of the original data as possible. If two observations were far apart in the original data, they should be far apart in our transformed data. Because we assume that noise is small compared to our feature values, we can assume that noise in features will account for relatively little of our observation variance. Thus, we can reduce the noise in our data by finding a lower-rank approximation: such a transformation will maintain the variance from the "true" data generation, while eliminating the additional variance from noise.
Industry contexts often involve high feature data. From minute user interactions to their entire history of ad engagement, it is possible to collect a substantial number of features. As an example, suppose any user has 10,000 features relating to specific histories involving ad campaigns, their engagement with a site, etc. In this example, running OLS with many users will be computationally expensive. Recall that linear regression on some observation matrix $X \in \mathbb{R}^{n \times d}$ has run-time O($d^2 * (n + d)$). With over a million users, as medium (or larger) online sites can expect, this computation is untenable. To reduce the run-time, we can either reduce n or d (or both). Because model accuracy improves with the number of observations, we'd prefer to reduce the number of features, d, rather than the observations, n. In the following sections, we'll review approaches to dimensionality reduction.

3 Principal Component Analysis

Principal Component Analysis (PCA) is a very important concept in Machine Learning. Machine Learning models often operate with very high-dimensional data. High dimensional data is data with a large number of features, i.e attributes of an observation. In a classic example, the number of bedrooms in a house is a relevant feature when modeling house values. The dimensionality of the data further increases when we augment our features. For example, we may use polynomial features, in which we apply each feature to the k-th degree: rather than some feature x, we include x, x^2, x^3, ..., x^k . While many features can be beneficial, there are a few important reasons why we may want to work with lower-dimensional data:

- Visualization - as long as we can get the dimensions down to 2 or 3
- Reduce computational load - for faster model performance
- Reduce variance in estimation - regularize the problem

How do we reduce the dimensionality of our data? We need to reduce our feature set, i.e take a subset of our features. Deciding which features to keep is a critical issue with several approaches. One could just keep the features that introduce the most variance to the data, but what if two most variable

features were actually also very correlated to each other? What if we just took a random subset of our features? This is where PCA is introduced in how to make high-dimensional data into precisely picked features that result in a lower-dimensional feature set.

Overview: PCA is an unsupervised dimensionality reduction technique. If PCA is given a matrix of data points, it finds one or more orthogonal directions that capture the largest amount of variance in the data. Intuitively, features with low variance contain "less information" about the data overall, and can be discarded without introducing much error. Why does PCA use an unsupervised approach rather than a supervised one? In contexts where labeling data is more expensive, unsupervised approaches will be more cost-effective. Thus PCA is powerful due to its ability to extract meaningful directions from unlabeled data.

Recall the following fundamentals:

- If we have $x^\top v$ where $v \in \mathbb{R}^d$ is some unit vector then we know that this equals $x^\top v = ||x||||v|| \cos(\theta)$ which is just how much of the projection is on the plane (the ratio of it).

The First Principal component: Let $X \in \mathbb{R}^{n \times d}$ where each row is assumed to be an i.i.d sample from some random vector x. We assume that each row's mean is zero, normalizing the features if necessary. Thus, because our vector data point matrix X is zero mean, the sample variance of the data points projections onto a unit vector v is given by:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^\top v)^2 = \frac{1}{n}\|Xv\|^2 = \frac{1}{n}v^\top X^\top Xv$$

Recall that v is constrained to have unit norm. By the above equation, the first loading vector v_1 as the solution to the constrained optimization problem

$$\text{max}_v v^\top X^\top Xv \text{subject to } v^\top v = 1$$

We observe that by reducing this constrained optimization problem into an unconstrained problem we can derive the optimal value at $\lambda = \lambda_{max}(X^\top X)$ which is achieved when v_1 is a unit eigenvector of $X^\top X$ corresponding to its largest eigenvalue.

4 Canonical Correlation Analysis

As we saw above, PCA provide us with a framework for producing lower-dimensional data without reliance on labels y, making it an unsupervised algorithm. However, there are situations that we can draft where the most relevant directions in x for understanding y are not necessarily the directions with the greatest variances in x. We can think of one example as if the x data was contaminated with a strong, correlated noise signal. In this case, PCA would actually throw away those dimensions with this strong noise variation, the opposite of our desired outcome. Thus we prefer to approach dimensionality reduction in a way that takes advantage of paired, i.e labeled, (x, y) data.

Overview: Let's first assume that we have two vector-valued quantities with many paired samples: X and Y. We also assume that these two X and Y are jointly distributed as random variables. The goal is to understand the underlying relationship between our two random variables. Thus to do this we introduce three new underlying iid standard Gaussian random vectors: Z_J (representing the joint part), Z_X (representing the randomness that is purely in X and not shared by Y), and Z_Y (representing the same as Z_X but for Y). Thus if we have these three random vectors we can see that

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ 0 & C & D \end{bmatrix} \begin{bmatrix} Z_X \\ Z_J \\ Z_Y \end{bmatrix}$$

We use the Pearson correlation coefficient in order to understand the relationship between X and Y as seen in CCA.

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)Var(Y)}}$$

CCA maximizes the expression of $\rho(X_{rv}^T u, Y_{rv}^T v)$. In order to compute the maximizer, we simplify the covariance matrix of X and Y using whitening and decorrelation to get the following solution:

To finally get the associated eigenvectors of interest we have

$$U = W_x D_x U_d, V = W_y D_y V_d$$

where

$$U_d = \begin{bmatrix} I_k \\ 0_{p-k,k} \end{bmatrix} \in \mathbb{R}^{p \times k}$$

$$V_d = \begin{bmatrix} I_k \\ 0_{q-k,k} \end{bmatrix} \in \mathbb{R}^{q \times k}$$

, where $W_x = U_x S^{1/2} U_x^T$, U and S are taken from the SVD of $X^T X$, and W_y is defined similarly using U and S from the SVD of $Y^T Y$. U is chosen to be D_x, and V is chosen to be D_y with $USV^T = X_w^T Y_w$. Finally, $u_d = D_x^T u_w$ and v_d is defined similarly using D_y and v_w.