Data Visualization: Final Project

Sushmita V Gopalan

6/2/2018

1. Introduction

My final project for Data Visualization is the first step towards my endeavour to use data visualization to illustrate concepts in machine learning. Visualizations have always been an incredibly useful pedagogical tool. Often, describing ideas in machine learning through visualization can be tricky because of two things - the iterative nature of most algorithms and the problem of high-dimensional datasets. To address each of these issues, I use principal component analysis to bring the dimensionality of my dataset down to 2 so that I can map it onto a 2-D graph and I use animated gifs to illustrate what happens at each iteration of a classification algorithm.

I illustrate two things in this project-

- Class Imbalance Problems in Machine Learning and some ways to address them. (See `ClassImbalance.Rmd`)
- The Adaptive Boosting or AdaBoost algorithm. (See `AdaBoost.Rmd`)

2. Methods

2.1 Principal Component Analysis

For both the Class Imbalance lesson and the AdaBoost lesson, I use sample datasets to illustrate the problem in 2-D space. In order to do this, I use Principal Component Analysis (PCA) to reduce dimensionality of my datasets to 2. This is done in Python using `scikitlearn`. PCA uses orthogonal transformation to convert a set of variables into a set of linearly uncorrelated values called principal components. The first two principal components form my features throughout this analysis.

2.2 Class Imbalance

A common problem encountered in classification problems is that of class imbalance. In a binary classification context, this means that one class occurs far more than the other. Majority class examples are depicted in blue while minority class examples are depicted in black. In this sample dataset, 10% of observations belong to the minority class and the remaining 90% belongs to the majority class.

The sparseness of observations from the minority class makes it difficult for the classifier to learn to identify it accurately. Further, the metrics of model evaluation we use in balanced datasets do not work quite as well in these contexts. Consider a classifier that naively predicts 'Blue' for all examples in the dataset - since 90% of the data is indeed blue, it would achieve an accuracy of 90%. However, if we're interested in identifying the black observations, this would be useless as it would fail to identify 100% of them.

2.2.1 Random Undersampling

Random undersampling a non-heuristic resampling method where we train our classifier with a random subset of examples from the majority class and all of the data from the minority class, such that the two are equal in number. This method is also not robust to cross-validation because of the fact that not all small subsets of

a large dataset follow the same probability distribution. Results vary significantly depending on which subset of the majority class examples are considered for training. To illustrate this, I use a little button that allows users to refresh multiple times and see different subsets to understand the way in which this method is very dependent on the seed that is set.

2.2.2 Random Oversampling

This is a non-heuristic resampling method, where we randomly sample from the minority class with replacement, until we have as many examples as there are in the majority class. An obvious drawback of this approach is that by simply replicating observations we are not actually training the classifier with more information. It is equivalent to up-weighting each observation from the minority class and down-weighting each observation from the majority class. This also places undue emphasis on outliers because the classifier repeatedly encounters examples that are outliers and learns that they are typical instances of the minority class, leading to poor prediction accuracy on held-out data. This approach makes the classifier overfit to a great degree, it learns to recognize the examples it has encountered very well, but fails to generalize to out-of-sample examples. Since the new examples are identical to the existing ones, the graph with oversampled data looks identical to one on the left with the original data. A button labeled 'Jitter' allows users to add a little random noise to the observations so they can see the different observations more clearly.

2.2.3 Synthetic Minority Oversampling

Chawla et al. (2002) came up with the Synthetic Minority Oversampling Technique (SMOTE) where we don't merely replicate examples from the minority class, but generate new, synthetic examples. This is done by interpolating between several minority class examples that lie together. Intuitively, this allows the classifier to build larger decision regions that contain nearby instances from the minority class and helps avoid overfitting to the very few examples available from the minority class. All the new synthetic samples lie within the same convex feature space as the original minority class samples. To illustrate this, I used the `ggpubr` library to draw a convex hull around the borders of the minority class examples.

2.2.4 Removing Tomek Links

In order to help the classifier better distinguish between the two classes, Tomek suggests that borderline examples and those suffering from class-label noise be removed. Consider two observations x and y, each having a different class label. Let $\delta(x, y)$ be the Euclidian distance between x and y. Then, a pair (x, y) is considered a Tomek link if and only if no other example z exists such that $\delta(x, z) < \delta(x, y)$ or $\delta(y, z) < \delta(y, x)$. In other words, two observations form a Tomek link if they have different class labels and are each other's nearest neighbour. Observations that belong to Tomek links are thus considered either borderline or noisy. The idea here is that the presence of borderline or noisy examples occludes the classifier's ability to learn what an example from the minority class looks like. By removing the majority class example from each Tomek link, we create clearer decision boundaries. These are highlighted with manually added labels.

2.3 Adaptive Boosting

The term AdaBoost comes from the phrase **Ada**ptive **Boost**ing. This refers to a family of algorithms that create a strong classifier with a weighted combination of weak classifiers. A weak classifier is any decision rule that performs better than random guessing. At each iteration, examples misclassified by the particular weak classifier are upweighted or boosted before training the next weak classifier. The intuition behind this is to aggregate simple rules into a complex function and boost incorrect classifications to gain more information.

For illustrative purposes, I use a dataset hosted by the Python package `scikitlearn`. It consists of 8x8 images of digits from 0 through 9. We subset out digits `1` and `8` to illustrate a binary classification problem.

Each digit has 180 observations. So, we have a well-balanced dataset of 360 observations. Henceforth, `1` will be indicated by blue points and `8` by black points. PCA is used to reduce the 64 dimensions to two principal components.

I made several modifications to an implementation of the adaboost algorithm found here to extract the weights vectors I needed to illustrate what was happening at each iteration. Training data and weights vectors from each weak learner were used to extract predictions at each iteration. Classifier weights were extracted to make predictions on the test set at each iteration. These were saved as text files which are then read by the R script for plotting. The python script for this is `adaboost.py`. Note that the weak learners used for this implementation are decision stumps.

3. Discussion

My 'lessons' are aimed at a semi-technical audience - individuals who have some background and understand the terms and rhetoric employed by the machine learning literature but not necessarily the mathematical background to grasp the intuition behind these concepts from equations alone.

In the AdaBoost lesson, I leaned heavily into the 'boosting' metaphor by literally increasing the size of the misclassified examples to illustrate the fact that their weights were being boosted. Functionalities offered by `ggplot2` were useful in depicting 4 different things in a 2-D graph. For each point, its position in the x-y plane indicates the values of the principal components, colour indicates *true* classification and size indicates the correctness of our prediction. These visual components, I believe, help gain an understanding of the algorithm.

What has the audience learned from your work?

I trust that a viewer looking at my lessons will gain a good intuitive understanding of class imbalance issues in machine learning and how to address them, and how an AdaBoost algorithm works. I tested out iterations of my apps on friends without significant machine learning backgrounds to see if they understand what it was saying and incorporated their feedback, along with the very useful feedback I received in class.

How is the visualization truthful, functional, beautiful, insightful, and/or enlightening?

Truthful

My visualizations are truthful in that they are a faithful depiction of the data they display. Generating data for the class imbalance lesson was done through the package `imbalanced-learn`. I modified a from-scratch implementation of AdaBoost and I'm confident it is a correct and statistically sound implementation of the algorithm.

Functional

They are functional in that they convey the message I intend to convey through them. I use colour, size, labels, titles and backgrounds that are imbued with meaning to clarify the concepts I attempt to describe. Since my axis labels are not very meaningful - they are just principal components from toy datasets, I drop them wherever they don't seem necessary. The storyboard layout of my apps helps streamline the information disbursal by allowing the user to click through at their own pace, and move back and forth as required to help their understanding.

Beautiful

My visualizations are focused on effectively conveying concepts and use colour, fonts and layouts accordingly. A lot of the colour palette is base `ggplot2` - I think this helps because not only is it clean and simple, audiences are also accustomed to these colours and do not waste time adjusting to new colour schemes. I tried to keep the text pared down and minimal in each frame so as to not overwhelm the user with cluttered information.

Insightful

I think all illustrations used in my lessons are clear and convey the concept they intend to depict. For example, the convex hull in the SMOTE illustration emphasizes the fact that new features are still created within the same feature space. Using size to indicate the correctness of a prediction drives home the idea of upweighting misclassified examples.

Enlightening

I think the coolest part of my apps is the simultaneous gifs that show the linear progression of the adaboost algorithm and how it fares on both the training and test sets. This is definitely something I haven't seen before and I think it is enlightening in that it really helps you understanding what this algorithm improves over iterations.