Good representations, distance, metric learning and supervised dimensionality reduction

2014-03-20

How to represent features for machine learning is an important business. For example, deep learning is all about finding good representations. What exactly they are depends on a task at hand. We investigate how to use available labels to obtain good representations.

Motivation

The paper that inspired us a while ago was Nonparametric Guidance of Autoencoder Representations using Label Information by Snoek, Adams and LaRochelle. It's about autoencoders, but contains a greater idea:

> Discriminative algorithms often work best with highly-informative features; remarkably, such features can often be learned without the labels. (…) However, pure unsupervised learning (…) can find representations that may or may not be useful for the ultimate discriminative task. (…)
>
> In this work, we are interested in the discovery of latent features which can be later used as alternate representations of data for discriminative tasks. That is, we wish to find ways to extract statistical structure that will make it as easy as possible for a classifier or regressor to produce accurate labels.

You might say, well, train a feed-forward neural network and use the hidden layer activations as features. That’s a valid approach, but:

> The objective now is to learn (…) a hidden representation that is (…) immediately good for discrimination under the simplified choice of model, e.g., logistic regression. This is undesirable because it potentially prevents us from discovering informative representations for the more sophisticated nonlinear classifiers that we might wish to use later.

In practice this may or may not be a problem - Paul Mineiro thinks that features good for linear classifiers are good for non-linear ones too:

> engineer features that are good for your linear model, and then when you run out of steam, try to add a few hidden units.

By the way, it turns out that Paul has a paper on the subject matter:

> Representing examples in a way that is compatible with the underlying classifier can greatly enhance the performance of a learning system. In this paper we investigate scalable techniques for inducing discriminative features by taking advantage of simple second order structure in the data.

In any case, we’d like to extract features that are relevant for a supervised learning task at hand. But to do so, we won’t be looking at neural networks nor second moments. We’ll look at distance instead.

Distance

Many machine learning methods rely on some measure of distance between points, usually Euclidian distance. These methods are notably nearest neighbours and kernel methods.

Euclidian distance depends on a scale of each feature. If one column has a big spread compared to others, it will obviously dominate them when computing distance. Which leads us to scaling.

Scaling

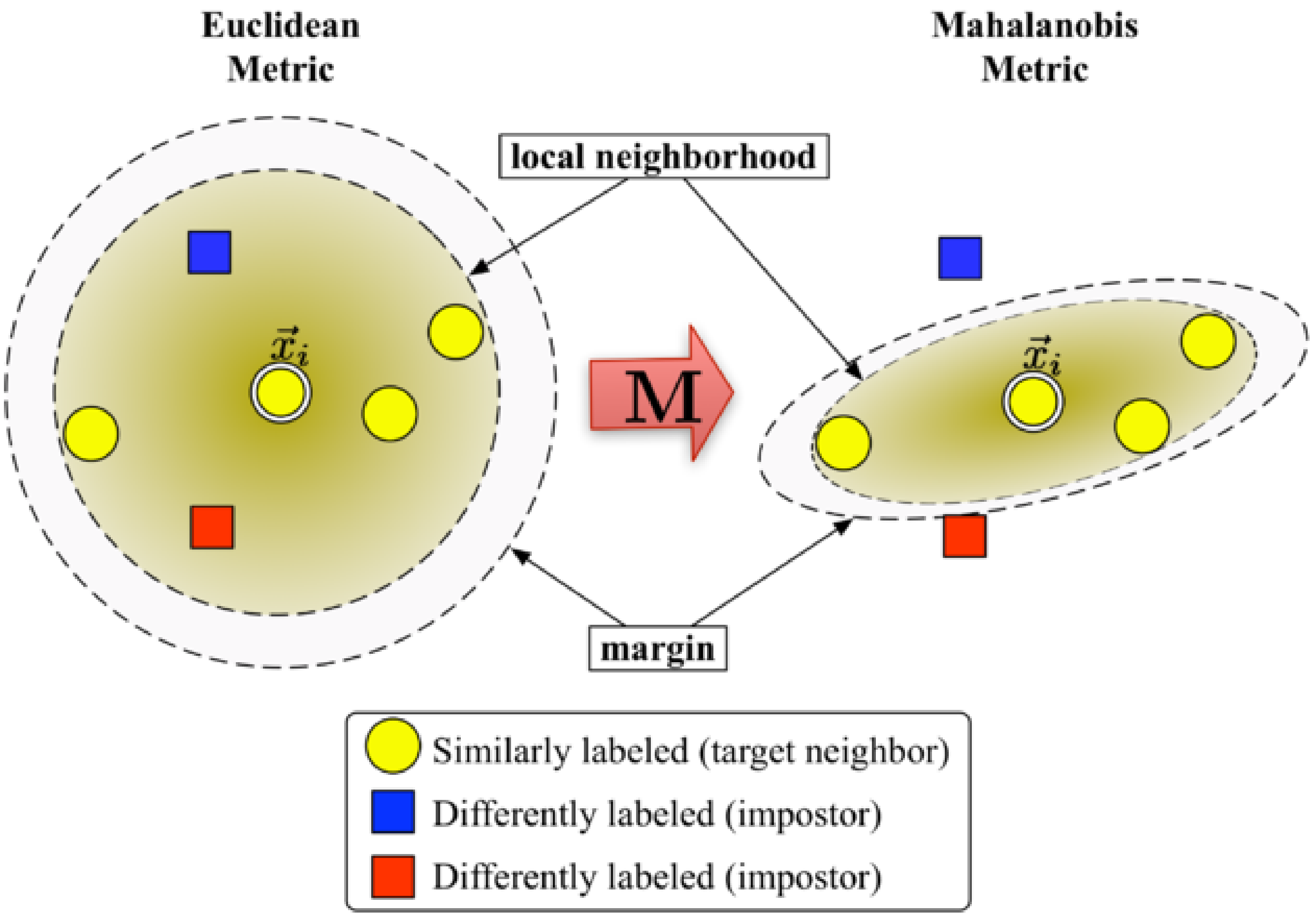
Standardizing data is a very common pre-processing method. It’s common because many algorithms, specifically those using gradient descent, perform best with the features centered around zero and on the same scale (at least approximately).

Standardizing consists of centering (or shifting, or subtracting the mean) and scaling.

When scaling, we multiply the design matrix X by a diagonal matrix. In special case when that diagonal matrix is an identity matrix, we get back X:

```
X * I = X
```

The entries on the main diagonal are the factors by which to scale each column. Here’s an image for a 2D case, from the Wikipedia article on LMNN (Large Margin Nearest Neighbours, see the Software section below):


Euclidean Metric
Mahalanobis Metric
local neighborhood
$\vec{x}_i$
M
$\vec{x}_i$
margin
Similarly labeled (target neighbor)
Differently labeled (impostor)
Differently labeled (impostor)

While it’s supposed to show LMNN, it shows the importance of scaling - by blowing up the vertical axis we can get yellow elements relatively close together and the other elements out of the neighbourhood. This illustrates the fact that there may be better options than scaling each axis to a unit standard deviation, at least when using methods concerned with distance.

Also, what if we extended the idea of scaling by adding some non-diagonal nonzero entries to the transformation matrix?

Metric learning

For the purpose of this article we’ll define metric learning as learning the matrix M with which we can linearly transform the design matrix as a whole. If M is square, we can transform a distance between two points. The idea is to warp it so that the points with the same (or similiar - in regression) labels get closer together and those with different labels get farther apart.

We can warp a Euclidian distance by sticking M in the middle of the inner product:

```
x1 - x2 = [1xD]
[1xD] * [DxD] * [Dx1] = [1x1]
```

`Dx1` is just the `x1 - x2` transposed.

We can also multiply the original matrix of features by M. The resulting matrix will have the same number of rows. The number of columns stays the same if M is square, changes otherwise.

N is the number of points, D - dimensionality. Matrix M is either `DxD` or `Dx?`. If it is square, dimensionality of the transformed design matrix stays the same:

```
[NxD] * [DxD] = [NxD]
```

If M is not square, we achieve either supervised dimensionality reduction or get an overcomplete representation:

```
[NxD] * [Dx?] = [Nx?]
```

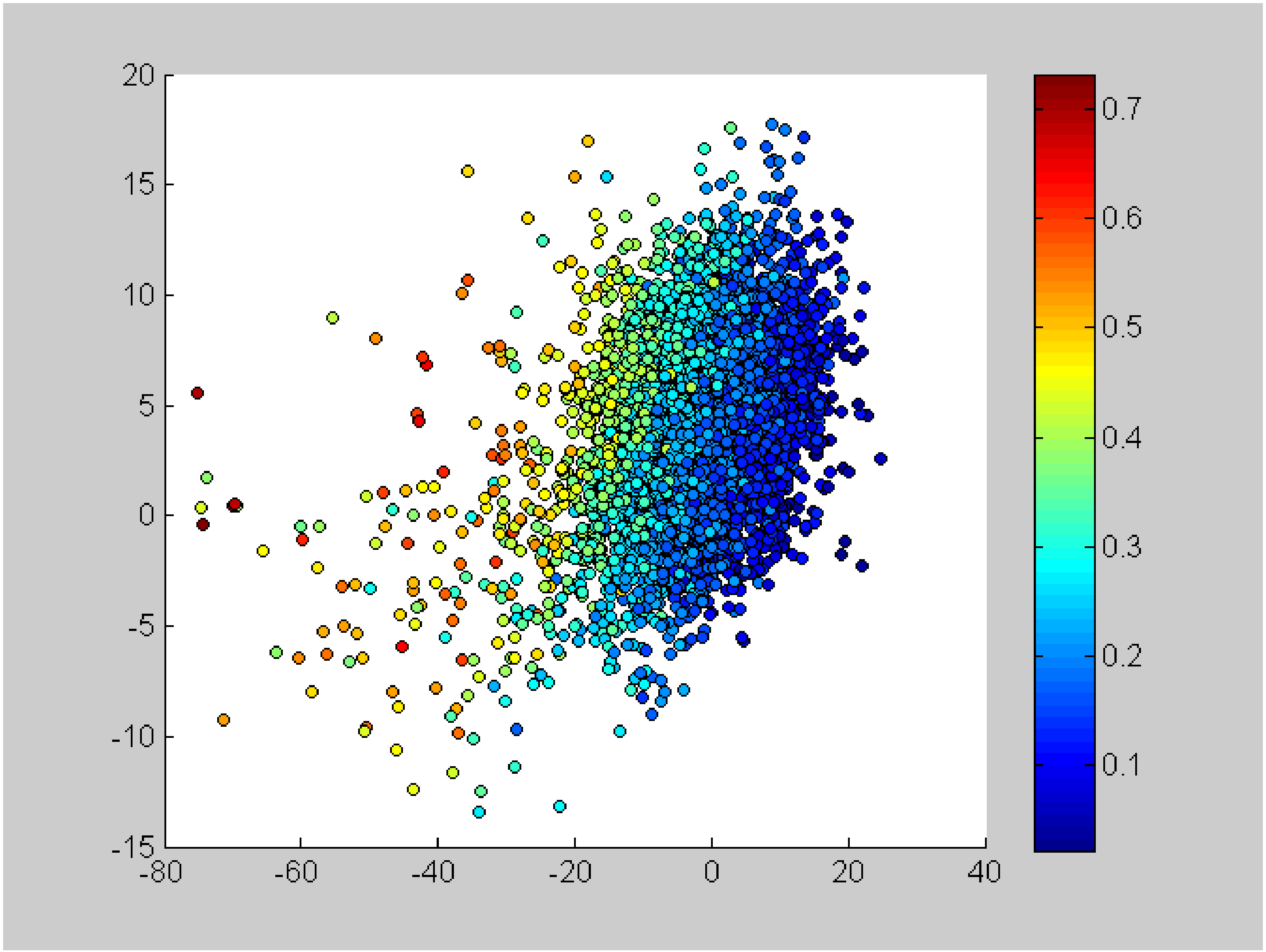
Software

Most available metric learning software concerns classification. Popular methods include Large Margin Nearest Neighbours, and (LMNN) and Information Theoretic Metric Learning (ITML). ITML seems to be faster, judging from results in various papers. There are many different methods which we haven't tried. One that we tried is Neighbourhood Component Analysis (NCA) by Hinton et al., which is awfully slow, even when implemented in C.

That's a general problem with those methods: they are not very scalable. If you have a big dataset, however, you could select a smaller representative subset of points (for example by clustering) to learn a transformation matrix, then apply the transformation to all the points.

Metric Learning for Kernel Regression

Kilian Weinberger is a prominent researcher in metric learning. While he's known for LMNN, he has another algorithm, MLKR, for regression. It features a demo with an animation showing how two-dimensional features evolve during training.



MLKR demo after twenty iterations

We apply MLKR to the *kin8nm* dataset to see how it fares against other methods we tried. Random forest on the raw data scores roughly 0.14 RMSE, and 0.09 after transformation. It’s a substantial improvement.

That’s all very good and well, but maybe old PCA would give the same result? We checked and got 0.127 on the test set when fitting PCA on the training examples only. If you fit on all available points, the random forest score is slightly worse than the one from raw data.

The code is available at GitHub.

Posted by Zygmunt Z. 2014-03-20 code, software

Tweet G+1 1

« PyBrain - a simple neural networks library in Python If you use R, you may want RStudio »

Comments

2 Comments FastML - machine learning made easy 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

jason • 2 years ago
What about sparsity? Doesn't sparsity (in sparse auto encoders) and sparse SVD etc promote discriminatory features..Do you have any experience/comments on that?
Reply • Share ›

ZygmuntZ Mod jason • 2 years ago
In my opinion overcomplete sparse representations a la Stanford work by disentangling factors of variation - "spreading out" the features.
Reply • Share ›

ALSO ON **FASTML - MACHINE LEARNING MADE EASY**

Real-time interactive movie recommendation
12 comments • 10 months ago
Rick de Villiers — A new real-time recommender has been developed. It's called TasteMonster: www.tastemonster.com. Real-

Coming out - FastML
2 comments • a day ago
zankbennett — April Fools! Nicely done :)

We were wrong - an apology to the readers
4 comments • a year ago
AB — April Fools?

Loading data in Torch is a mess
15 comments • a year ago
Atcold — tbl = {1, 2, 3, 4} -- defines a table tns = torch.Tensor(t) -- gives you the tensor from the table Above you don't have tensors, because

Recent Posts

- Coming out
- Bayesian machine learning
- What next?
- What is better: gradient-boosted trees, or a random forest?
- Numerai - like Kaggle, but with a clean dataset, top ten in the money, and recurring payouts
- What you wanted to know about TensorFlow
- Predicting sales: Pandas vs SQL

Twitter

Follow @fastml for notifications about new posts.

- Status updating...

Follow @fastml 3,230 followers

Also check out @fastml_extra for things related to machine learning and data science in general.

GitHub

Most articles come with some code. We push it to Github.

https://github.com/zygmuntz

MovieMood

MovieMood is our fast interactive movie recommender, introduced in this article. It enables rapid movie discovery. Check out the September beta.

http://moviemood.co