Composable ML

Boris Arnoux

2016

Two lessons from the Netflix prize

- The winners used models that combined different ML approaches.
- The winning algorithm was never used in production.

Context

Relevant problems to this presentation:

- Supervised regression or classification
- Simple structures in feature space (not cats)

Good examples: online recommendations, Ad tech, some finance, some physical models.

Linear methods rock

- Simple, fast, transparent.
- Online learning available
- Scalable, CPU & memory efficient.
- Sparse

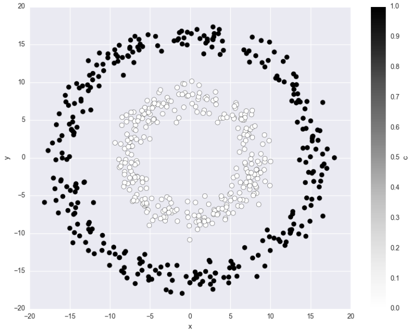
Such practical, operational and computational benefits are very important at scale.

Non linearity

Not all patterns are linearly separable.

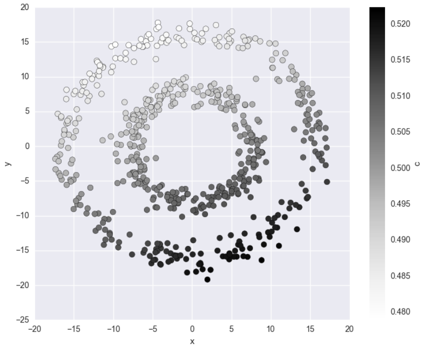
Non linearity

Not all patterns are linearly separable.



Non linearity

Not all patterns are linearly separable.



Kernels

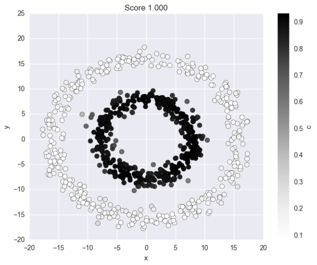
Theoretical justification for the use of kernels in linear models:

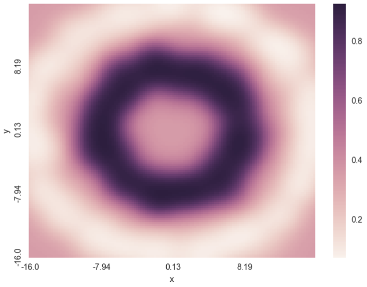
$$L(X.\mu, y) \equiv L(XX^T.v, y)$$

Since μ is learned against X, μ can be spanned by X.
Most famous algorithm: kernel support vector classification.

What a kernel machine sees (1/2)

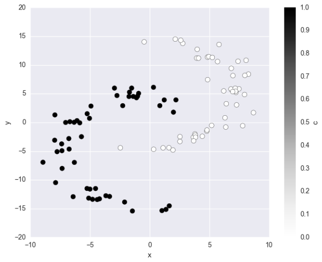
L2 Logistic regression, RBF kernel, nonlinear dataset:

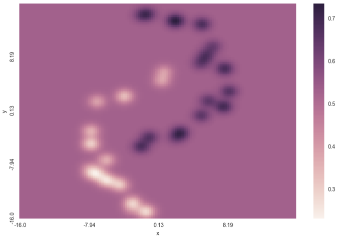




What a kernel machine sees (2/2)

L2 Logistic regression, RBF kernel, arcs dataset:





Here γ (as in $K(x_1, x_2) = exp(-\gamma||x1 - x2||^2)$) is chosen to highlight how the new feature space is built. For each sample, the RBF kernel constructs a local indicator variable in the original feature space. Each sample can become such a feature.

Cogs in the kernel machine

How a typical kernel decides how to classify a new sample x_{new}:

- x_{new} is compared to the training $X = (x_1, x_2, \ldots, x_n)$ using $K(.,.)$.
- The sample-to-sample distance $K(.,.)$ makes use of the original feature space.
- The kernel based features values $k = K(x_i, x_{new})$ are used for computing $y_{new} = v.k$.

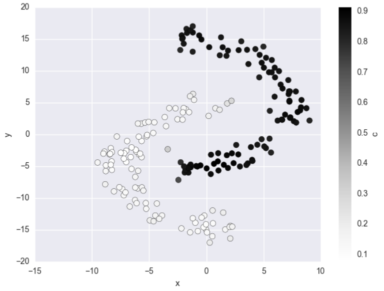
Weaknesses of kernels

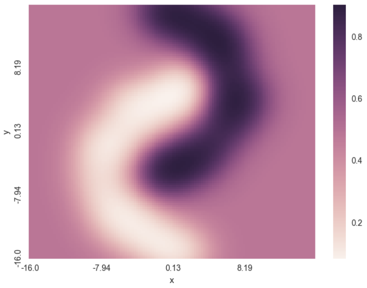
This leads to two points:

1. Often $K(.,x)$ uses all the features in the original space, which as dimensionality grows, eventually scrambles relevant dimensions with the less relevant dimensions.
2. Scalability issues (often $O(N^3)$ complexity)

Kernels with admixture of irrelevant features

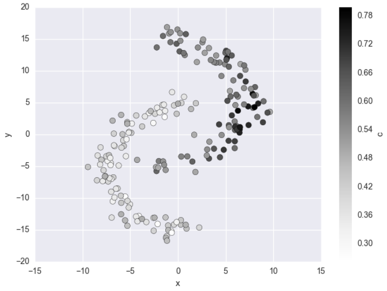
When all features (x, y) are relevant to the pattern:

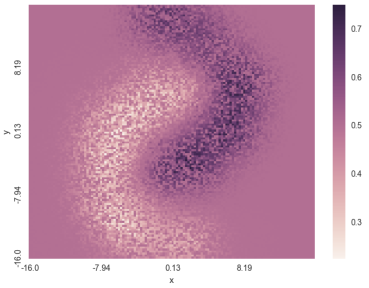




Kernels with admixture of irrelevant features

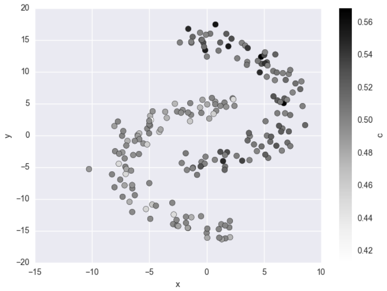
When two irrelevant features are added (of similar variance):

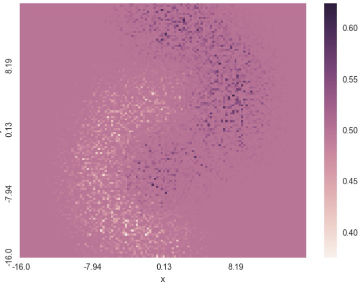




Kernels with admixture of irrelevant features

When four irrelevant features are added (of similar variance):





The pure-kernel approach breaks down.

Kernel machines weaknesses

Mitigating kernel weaknesses (1):

1. High number of features: use only a subset of original features, where distances make sense.

But how to bring in new information if we can't use new features?

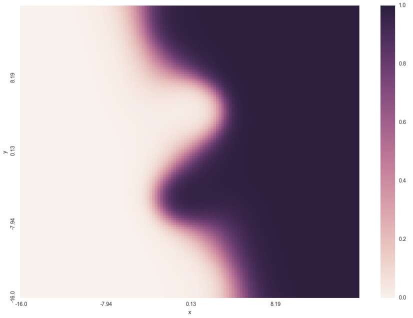
Explicit feature maps

Explicit kernel transform step-by-step:

- Pick features from the original feature space which makes sense to include in kernel calculations.
- Choosing "Interesting" samples, to promote as features $f_x(.) = K(., x)$ (lots of ways to be smart here!)
- Augment (rather than replace) the original feature space with these features.

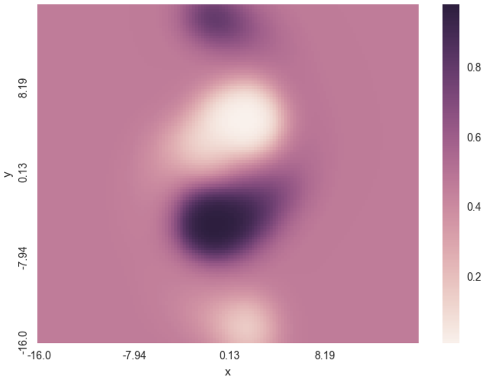
Explicit feature maps

L2 Logistic regression, ”arcs” dataset, decision function for RBF kernel feature transform:



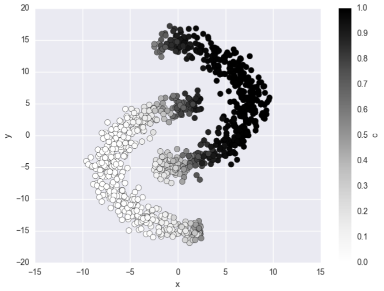
Explicit feature maps

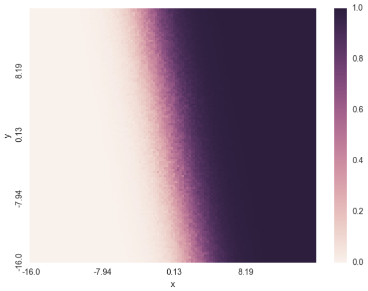
Decision function when the weights of the original features are erased after training (arcs dataset, RBF feature transform, L2 Logistic regression):



Kernels with admixture of irrelevant features

Original features augmented with an explicit mapping of kernel features, four irrelevant features added:





In this case, the kernel features break down too, but the model returns to a linear treatment.

Kernel machines weaknesses

Mitigating kernel weaknesses (2):

1. Scalability issues: use kernel approximation.

Addressing scalability issues

Nystroem sampling:

- Approximates any kernel.
- Based on sampling & interpolation.

Addressing scalability issues

For linear classification & regression purposes, it is (mostly) equivalent to picking random samples as features as opposed to promoting all the samples.

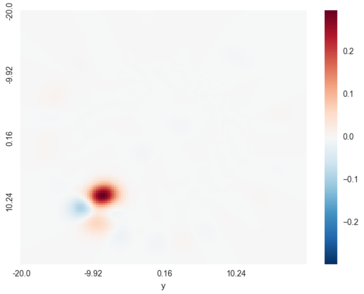
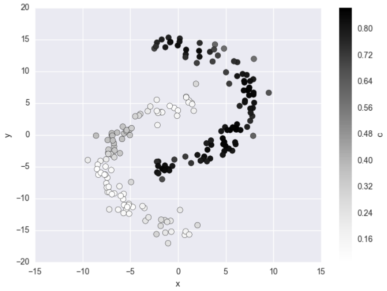
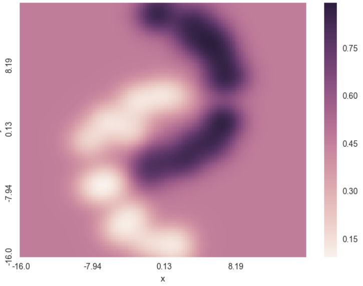


Figure: A Nystroem dimension.

Addressing scalability issues

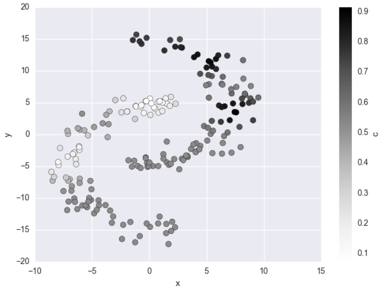
Nystroem sampling in action, L2 Logistic regression, 50 kernel dimensions & 100 samples:

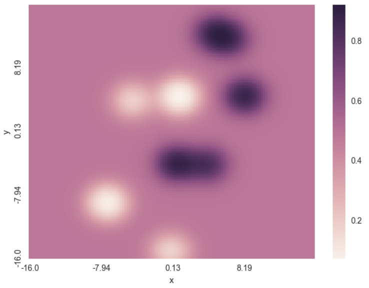




Addressing scalability issues

Nystroem sampling breaking down, L2 Logistic regression, 10 kernel dimensions & 100 samples:

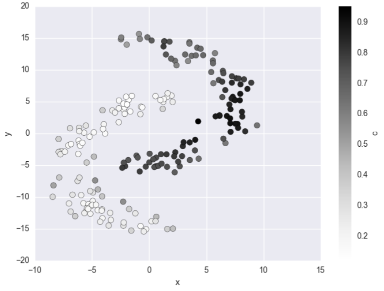


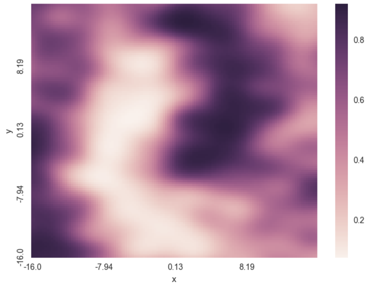


Important note: vanilla Nystroem sampling is unsupervised and does not attempt to find the best samples to pick.

Addressing scalability issues

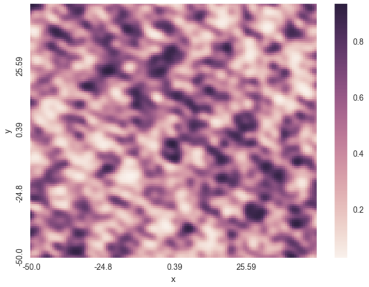
Alternative, Fourrier RBF kernel approximation





Addressing scalability issues

Fourrier RBF kernel approximation, zooming out:



Addressing scalability issues

You can make your own kernel specific approximation, promote samples based on:

- Feature space coverage.
- Where it helps the loss function.

Optionally go through a step of Nystroem or Cholesky for "normalization".

Kernels wrap up

Kernels allow non linear learning, and explicit mappings allows:

- Properly taking into account new features.
- Gives a lot of engineering latitude in limiting the dimension of the new feature space by sampling.

Non linearity II

Special cases of non-linearity merit special treatment:

- per-feature internal structure.
- particular features combinations.

Non linearity II

Special cases of non-linearity merit special treatment:

- per-feature internal structure.
- particular features combinations.

Feature engineering helps with specialized feature transforms:

- Split a feature in binary bins (indicator variables) or linear steps (within-bin barycentric coordinates).
- Bins extend to feature pairs (products of feature bin indicators).

Non linearity II

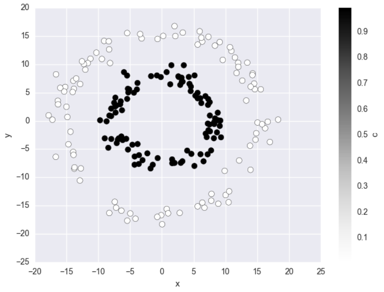
How to find these relevant combinations systematically, and optimally?

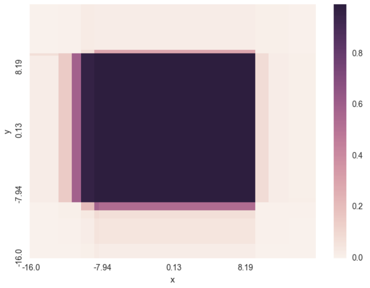
Non linearity II

How to find these relevant combinations systematically, and optimally? We need a supervised feature transform.

Boosting feature transform

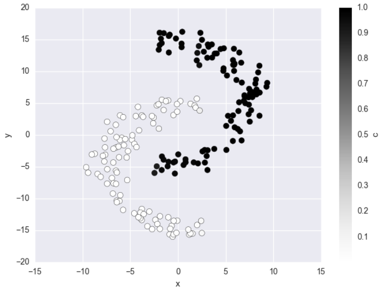
Original features augmented with boosting features, L2 Logistic regression:

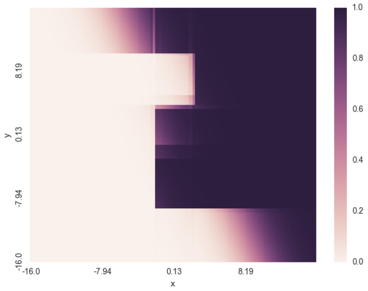




Boosting feature transform

Original features augmented with boosting features, L2 Logistic regression:





Boosting transforms basics

What is a boosting transform:

- Train a gradient boosting model.
- Each leaf of each tree is an indicator variable.
- Augment the initial feature space with the leaf indicators as features.

They are the workhorse of CTR prediction at Facebook (see ADKDD 2014)

Boosting feature transform analogies:

- one-level tree (decision stump): analogous to feature bin indicator.
- multiple branching levels: analogous to feature tuples.
- minimum samples per leaf: analogous to quantile binning.

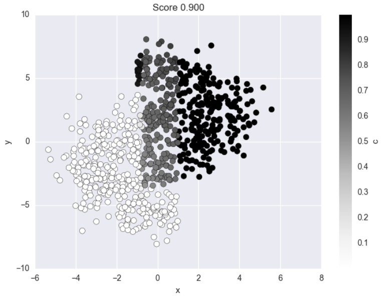
Boosting transforms pros & cons

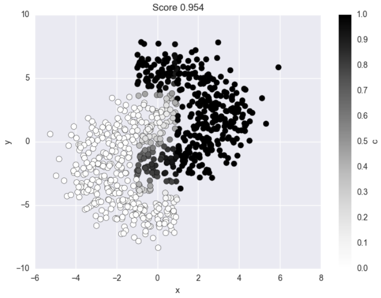
Boosting transform are great, they are

- Sparse.
- Supervised: mostly no need to worry about irrelevant features.
- A bit rough around the edges...

Boosting & Mixed classes

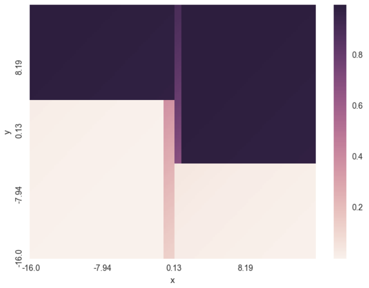
L2 Logistic regression, boosting transform alone vs boosting plus RBF features

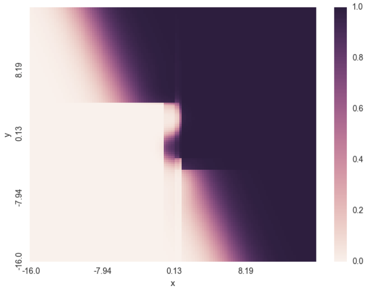




Boosting & Mixed classes

L2 Logistic regression, boosting transform alone vs boosting plus RBF features





Conclusion

An approach based on composable feature transforms, with:

- Linear learning core (with all the benefits)
- Feature transforms create a white box, supervised map of the feature space.
- Feature transforms operate correctly side by side, with other transforms and with linear features.

It contrasts with the "pick the right black box" approach.
Code on github `https://github.com/borithefirst/epfl_pres/blob/master/kernel_lin.py`