Deep-er Kernels

John Shawe-Taylor

Department of Computer Science
University College London

LSOLDM Meeting, September 2013
Joint work as referenced plus Dimitrios Athanasakis, Delmiro Fernandez-Reyes
Deep learning has (re-)emerged as having important research and commercial value

Deep belief networks and related approaches have led this charge

Kernels are sometimes referred to as ‘shallow’

Aim of this talk is to:

- Discuss what we mean by deep learning
- Describe a number of ways in which kernel learning has been made ‘deeper’
Deep learning has (re-)emerged as having important research and commercial value.

Deep belief networks and related approaches have led this charge.

Kernels are sometimes referred to as ‘shallow’.

Aim of this talk is to:

- Discuss what we mean by deep learning
- Describe a number of ways in which kernel learning has been made ‘deeper’
Deep learning has (re-)emerged as having important research and commercial value.

Deep belief networks and related approaches have led this charge.

Kernels are sometimes referred to as ‘shallow’.

Aim of this talk is to:
- Discuss what we mean by deep learning
- Describe a number of ways in which kernel learning has been made ‘deeper’
Deep learning has (re-)emerged as having important research and commercial value

Deep belief networks and related approaches have led this charge

Kernels are sometimes referred to as ‘shallow’

Aim of this talk is to:

Discuss what we mean by deep learning
Describe a number of ways in which kernel learning has been made ‘deeper’
Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems.

However, they actually implement linear functions in the kernel defined feature space:

\[ \mathbf{x} \xrightarrow{\text{fixed}} \phi(\mathbf{x}) \xrightarrow{\text{learned}} \langle \mathbf{w}, \phi(\mathbf{x}) \rangle \]

so that the learning (of \( \mathbf{w} \)) only occurs in one ‘layer’.

This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions.

- Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed.
- Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel.
Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems. However, they actually implement linear functions in the kernel defined feature space:

\[ \mathbf{x} \overset{\text{fixed}}{\mapsto} \phi(\mathbf{x}) \overset{\text{learned}}{\mapsto} \langle \mathbf{w}, \phi(\mathbf{x}) \rangle \]

so that the learning (of \(\mathbf{w}\)) only occurs in one ‘layer’. This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions. Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed. Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel.
Why Shallow Learning?

- Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems.
- However, they actually implement linear functions in the kernel defined feature space:

\[ x \xrightarrow{\text{fixed}} \phi(x) \xrightarrow{\text{learned}} \langle w, \phi(x) \rangle \]

so that the learning (of \( w \)) only occurs in one ‘layer’.
- This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions.
  - Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed.
  - Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel.
In practice we typically do perform some learning of the kernel:

- fix some hyper-parameters via some heuristic (e.g. width $\sigma$ of a Gaussian kernel)
- use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)

In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:

- standard generalisation bounds no longer apply if we choose the feature space based on the training data
- even test set bounds will be invalidated if we include the testing data in the representation learning phase

Often more sophisticated representations encode ‘deep’ prior knowledge, but are ‘learned’ by trial and error

- for example the histograms of patch cluster presence used in an object detection system
What happens in practice?

In practice we typically do perform some learning of the kernel:
- fix some hyper-parameters via some heuristic (e.g. width $\sigma$ of a Gaussian kernel)
- use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)

In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:
- standard generalisation bounds no longer apply if we choose the feature space based on the training data
- even test set bounds will be invalidated if we include the testing data in the representation learning phase

Often more sophisticated representations encode ‘deep’ prior knowledge, but are ‘learned’ by trial and error
- for example the histograms of patch cluster presence used in an object detection system
What happens in practice?

- In practice we typically do perform some learning of the kernel:
  - fix some hyper-parameters via some heuristic (e.g. width $\sigma$ of a Gaussian kernel)
  - use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)

- In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:
  - standard generalisation bounds no longer apply if we choose the feature space based on the training data
  - even test set bounds will be invalidated if we include the testing data in the representation learning phase

- Often more sophisticated representations encode ‘deep’ prior knowledge, but are ‘learned’ by trial and error
  - for example the histograms of patch cluster presence used in an object detection system
Aim of this talk

- Present a number of promising directions that tick (some of) the following boxes:
  - Learn a (kernel) representation possibly tuned to the main learning task
  - Provide any analysis of the resulting system that supports its design and bounds its performance
  - Provide empirical evidence that supports the approach on real world data
- The different contributions may appear disjointed but I hope a convincing and coherent story will emerge:
  - Deep-er learning of kernels is alive and kicking!
Aim of this talk

- Present a number of promising directions that tick (some of) the following boxes:
  - Learn a (kernel) representation possibly tuned to the main learning task
  - Provide any analysis of the resulting system that supports its design and bounds its performance
  - Provide empirical evidence that supports the approach on real world data
- The different contributions may appear disjointed but I hope a convincing and coherent story will emerge:
  - deep-er learning of kernels is alive and kicking!
Aim of this talk

- Present a number of promising directions that tick (some of) the following boxes:
  - Learn a (kernel) representation possibly tuned to the main learning task
  - Provide any analysis of the resulting system that supports its design and bounds its performance
  - Provide empirical evidence that supports the approach on real world data
- The different contributions may appear disjointed but I hope a convincing and coherent story will emerge:
  - Deep-er learning of kernels is alive and kicking!
Present a number of promising directions that tick (some of) the following boxes:

- Learn a (kernel) representation possibly tuned to the main learning task
- Provide any analysis of the resulting system that supports its design and bounds its performance
- Provide empirical evidence that supports the approach on real world data

the different contributions may appear disjointed but I hope a convincing and coherent story will emerge:

- deep-er learning of kernels is alive and kicking!
Aim of this talk

- Present a number of promising directions that tick (some of) the following boxes:
  - Learn a (kernel) representation possibly tuned to the main learning task
  - Provide any analysis of the resulting system that supports its design and bounds its performance
  - Provide empirical evidence that supports the approach on real world data
- the different contributions may appear disjointed but I hope a convincing and coherent story will emerge:
  - deep-er learning of kernels is alive and kicking!
Aim of this talk

Present a number of promising directions that tick (some of) the following boxes:

- Learn a (kernel) representation possibly tuned to the main learning task
- Provide any analysis of the resulting system that supports its design and bounds its performance
- Provide empirical evidence that supports the approach on real world data

the different contributions may appear disjointed but I hope a convincing and coherent story will emerge:

- deep-er learning of kernels is alive and kicking!
Matching pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates.
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace.
- Allows different criteria for selection to be implemented in one framework, e.g., sparse PCA, classification, regression, canonical correlation analysis, etc. and all come with bounds.

Matching pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates.
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace.
- Allows different criteria for selection to be implemented in one framework, e.g., sparse PCA, classification, regression, canonical correlation analysis, etc., and all come with bounds.

Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates.

Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace.

Allows different criteria for selection to be implemented in one framework, e.g., sparse PCA, classification, regression, canonical correlation analysis, etc., and all come with bounds.

Figure: Bound plot for sparse KCCA using 1-dimension.
If we consider learning a representation as pre-processing stage, it is natural to consider modelling the data with a probabilistic model.

There are then two main methods of defining kernels from probabilistic models:

- Averaging over a model class - i.e. each model gives one feature:
  \[ \kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m) \]
  also known as the marginalisation kernel.

- Fisher kernels for cases where the model is determined by a real parameter vector.

Give example of Fisher kernel.
If we consider learning a representation as pre-processing stage, it is natural to consider modelling the data with a probabilistic model.

There are then two main methods of defining kernels from probabilistic models:

- Averaging over a model class - i.e. each model gives one feature:
  \[ \kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m) \]
  also known as the marginalisation kernel.

- Fisher kernels for cases where the model is determined by a real parameter vector.

Give example of Fisher kernel.
If we consider learning a representation as pre-processing stage, it is natural to consider modelling the data with a probabilistic model.

There are then two main methods of defining kernels from probabilistic models:

- Averaging over a model class - i.e. each model gives one feature:
  \[ \kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m) \]
  also known as the marginalisation kernel.
- Fisher kernels for cases where the model is determined by a real parameter vector
- Give example of Fisher kernel
Fisher kernels

- We assume the model is parametrised according to some parameters: consider the simple example of a 1-dim Gaussian distribution parametrised by $\mu$ and $\sigma$:

$$M = \left\{ P(x|\theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{(x - \mu)^2}{2\sigma^2} \right) : \theta = (\mu, \sigma) \in \mathbb{R}^2 \right\}.$$

- The Fisher score vector is the derivative of the log likelihood of an input $x$ wrt the parameters:

$$\log L_{(\mu, \sigma)}(x) = -\frac{(x - \mu)^2}{2\sigma^2} - \frac{1}{2} \log (2\pi\sigma).$$
We assume the model is parametrised according to some parameters: consider the simple example of a 1-dim Gaussian distribution parametrised by $\mu$ and $\sigma$:

$$M = \left\{ P(x|\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(x - \mu)^2}{2\sigma^2} \right) : \theta = (\mu, \sigma) \in \mathbb{R}^2 \right\}.$$ 

The Fisher score vector is the derivative of the log likelihood of an input $x$ wrt the parameters:

$$\log \mathcal{L}_{(\mu, \sigma)}(x) = -\frac{(x - \mu)^2}{2\sigma^2} - \frac{1}{2} \log (2\pi\sigma).$$
Hence the score vector is given by:

\[
g(\theta^0, x) = \left( \frac{x - \mu_0}{\sigma_0^2}, \frac{(x - \mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0} \right).
\]

Taking \( \mu_0 = 0 \) and \( \sigma_0 = 1 \) the feature embedding is given by:
Fisher kernels

Hence the score vector is given by:

\[ g(\theta^0, x) = \left( \frac{x - \mu_0}{\sigma_0^2}, \frac{(x - \mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0} \right). \]

Taking \( \mu_0 = 0 \) and \( \sigma_0 = 1 \) the feature embedding is given by:
Fisher kernels

Shawe-Taylor

Deep-er Kernels
We can consider a Markov model of generating text conditioned on the previous $n$-characters.

Taking the uniform distribution model gives the class of string kernels - but these can now be learned based on a corpus.

can extend to probabilistic Finite State Automata learned from the corpus.

results competitive with tfidf BoWs on Reuters, with some improvements in average precision.

We can consider a Markov model of generating text conditioned on the previous $n$-characters.

Taking the uniform distribution model gives the class of string kernels - but these can now be learned based on a corpus.

can extend to probabilistic Finite State Automata learned from the corpus.

results competitive with tfidf BoWs on Reuters, with some improvements in average precision.

We can consider a Markov model of generating text conditioned on the previous $n$-characters.

Taking the uniform distribution model gives the class of string kernels - but these can now be learned based on a corpus.

can extend to probabilistic Finite State Automata learned from the corpus.

results competitive with tfidf BoWs on Reuters, with some improvements in average precision.

String kernels as Fisher kernels

- We can consider a Markov model of generating text conditioned on the previous $n$-characters.
- Taking the uniform distribution model gives the class of string kernels - but these can now be learned based on a corpus.
- Can extend to probabilistic Finite State Automata learned from the corpus.
- Results competitive with tfidf BoWs on Reuters, with some improvements in average precision.

MKL puts a 1-norm constraint on a linear combination of kernels:

\[
\kappa(x, x') = \sum_{t=1}^{N} z_t \kappa_t(x, x') : z_t \geq 0, \sum_{t=1}^{N} z_t = 1
\]

and trains an SVM while optimizing \(z_t\) – a convex problem.

Also obtain corresponding bound (using convex hull bound for Rademacher complexity):

\[
P(y \neq \text{sgn}(g(x))) \\
\leq \frac{1}{m \gamma} \sum_{i=1}^{m} \xi_i + \frac{1}{\gamma} \hat{R}_m \left( \bigcup_{t=1}^{N} \mathcal{F}_t \right) + 3 \sqrt{\frac{\ln(2/\delta)}{2m}}
\]

where \(\mathcal{F}_t = \{x \rightarrow \langle w, \phi_t(x) \rangle : \|w\| \leq 1\}\).
Multiple kernel learning

MKL puts a 1-norm constraint on a linear combination of kernels:

\[
\kappa(x, x') = \sum_{t=1}^{N} z_t \kappa_t(x, x') : z_t \geq 0, \sum_{t=1}^{N} z_t = 1
\]

and trains an SVM while optimizing \( z_t \) – a convex problem.

obtain corresponding bound (using convex hull bound for Rademacher complexity):

\[
P(y \neq \text{sgn}(g(x))) \leq \frac{1}{m\gamma} \sum_{i=1}^{m} \xi_i + \frac{1}{\gamma} \hat{R}_m \left( \bigcup_{t=1}^{N} \mathcal{F}_t \right) + 3 \sqrt{\frac{\ln(2/\delta)}{2m}}
\]

where \( \mathcal{F}_t = \{ x \rightarrow \langle w, \phi_t(x) \rangle : \|w\| \leq 1 \} \).
The Rademacher complexity provides a way of measuring the complexity of a function class $\mathcal{F}$ by testing how well on average it can align with random noise:

$$\hat{R}_m(\mathcal{F}) = \mathbb{E}_\sigma \left[ \sup_{f \in \mathcal{F}} \frac{2}{m} \sum_{i=1}^m \sigma_i f(x_i) \right].$$

is known as the Rademacher complexity of the function class $\mathcal{F}$. 

Rademacher complexity
Bounding MKL

Need a bound on

\[ \hat{R}_m \left( \mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t \right) \]

McDiarmid gives with probability \( 1 - \delta_0 \) of a random selection of \( \sigma^* \):

\[
\hat{R}_m(\mathcal{F}) \leq \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i f(x_i) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}
\]

and

\[
\frac{2}{m} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i f(x_i) \leq \hat{R}_m(\mathcal{F}_t) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}
\]

with probability \( 1 - \delta_t \)
Bounding MKL

- Need a bound on
  \[ \hat{R}_m \left( \mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t \right) \]

- McDiarmid gives with probability $1 - \delta_0$ of a random selection of $\sigma^*$:

  \[ \hat{R}_m(\mathcal{F}) \leq \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(x_i) + 4 \sqrt{\frac{\ln(1/\delta)}{2m}} \]

  and

  \[ \frac{2}{m} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(x_i) \leq \hat{R}_m(\mathcal{F}_t) + 4 \sqrt{\frac{\ln(1/\delta)}{2m}} \]

  with probability $1 - \delta_t$
Hence taking $\delta_t = \delta / 2(N + 1)$ for $t = 0, \ldots, N$

$$
\hat{R}_m \left( \mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t \right) \\
\leq \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(x_i) + 4 \sqrt{\ln(2(N + 1)/\delta) \over 2m} \\
\leq \frac{2}{m} \max_{1 \leq t \leq N} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(x_i) + 4 \sqrt{\ln(2(N + 1)/\delta) \over 2m} \\
\leq \frac{2}{m} \max_{1 \leq t \leq N} \hat{R}_m(\mathcal{F}_t) + 8 \sqrt{\ln(2(N + 1)/\delta) \over 2m}
$$

with probability $1 - \delta / 2$. 
This gives an overall bound on the generalisation of MKL of

$$P( y \neq \text{sgn}(g(x))) \leq \frac{1}{m\gamma} \sum_{i=1}^{m} \xi_i + \frac{2}{\gamma m} \max_{1 \leq t \leq N} \sqrt{\text{tr}(K_t)} +$$

$$8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} + 3\sqrt{\frac{\ln(4/\delta)}{2m}}$$

where $K_t$ is the $t$-th kernel matrix.

Bound gives only a logarithmic (additive) dependence on the number of kernels.

Bounding MKL

- This gives an overall bound on the generalisation of MKL of

\[
P(y \neq \text{sgn}(g(x))) \leq \frac{1}{m\gamma} \sum_{i=1}^{m} \xi_i + \frac{2}{\gamma m} \max_{1 \leq t \leq N} \sqrt{\text{tr}(K_t)} + 
\]
\[
8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} + 3\sqrt{\frac{\ln(4/\delta)}{2m}}
\]

where \(K_t\) is the \(t\)-th kernel matrix.

- Bound gives only a logarithmic (additive) dependence on the number of kernels.

Vedaldi et al. have applied to the PASCAL Visual Objects Challenge (VOC 2007) data and
- improvements over the winners of the challenge in 17 out of the 20 categories
- in more than half of the categories the increase in average precision was over 25%
- have also scaled effectively to millions of kernels

Replacing the 2-norm regularisation of the SVM with a 1-norm gives a linear programme: can solve its dual using an iterative method:

1. initialise $u_i = 1/m$, $i = 1, \ldots, m$, $\beta = \infty$, $J = \emptyset$
2. choose $j^\star$ that maximises $f(j) = \sum_{i=1}^{m} u_i y_i H_{ij}$
3. if $f(j^\star) \leq \beta$ solve primal restricted to $J$ and exit
4. $J = J \cup \{j^\star\}$
5. Solve dual restricted to set $J$ to give $u_i, \beta$
6. Go to 2

- Note that $u_i$ is a distribution on the examples
- Each $j$ added acts like an additional weak learner
- $f(j)$ is simply the weighted classification accuracy
- Hence gives ‘boosting’ algorithm - with previous weights updated satisfying error bound
- Guaranteed convergence and soft stopping criteria
Column generation gives efficient MKL if we can pick the best weak learner in each $\mathcal{F}_t$ efficiently:

$$
\sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} u_i y_i f(x_i) = \sup_{w : \|w\| \leq 1} \sum_{i=1}^{m} u_i y_i \langle w, \phi_t(x_i) \rangle \\
= \sup_{w : \|w\| \leq 1} \left\langle w, \sum_{i=1}^{m} u_i y_i \phi_t(x_i) \right\rangle \\
= \left\| \sum_{i=1}^{m} u_i y_i \phi_t(x_i) \right\| \\
= \sqrt{u' Y K_t Y u} =: N_t
$$

easily computable from the kernel matrices (note that $u$ is sparse after first iteration and can also be chosen sparse at the start).
The optimal weak learner from $\mathcal{F}_t$ is realised by the weight vector that achieves the supremum

$$w = \frac{\sum_{i=1}^{m} u_i y_i \phi_t(x_i)}{\left\| \sum_{i=1}^{m} u_i y_i \phi_t(x_i) \right\|}$$

which has dual representation:

$$\alpha_i = \frac{1}{N_t} u_i y_i$$

Hence, can use the linear programming boosting approach to implement multiple kernel learning.

More generally can view the $u$ vector as a signal to refine other representations.
The optimal weak learner from $\mathcal{F}_t$ is realised by the weight vector that achieves the supremum

$$w = \frac{\sum_{i=1}^{m} u_i y_i \phi_t(x_i)}{\| \sum_{i=1}^{m} u_i y_i \phi_t(x_i) \|}$$

which has dual representation:

$$\alpha_i = \frac{1}{N_t} u_i y_i$$

Hence, can use the linear programming boosting approach to implement multiple kernel learning.

More generally can view the $u$ vector as a signal to refine other representations.
The optimal weak learner from $\mathcal{F}_t$ is realised by the weight vector that achieves the supremum

$$w = \frac{\sum_{i=1}^{m} u_i y_i \phi_t(x_i)}{\|\sum_{i=1}^{m} u_i y_i \phi_t(x_i)\|}$$

which has dual representation:

$$\alpha_i = \frac{1}{N_t} \frac{1}{u_i y_i}$$

Hence, can use the linear programming boosting approach to implement multiple kernel learning.

More generally can view the $u$ vector as a signal to refine other representations.
As an example consider Fisher kernels over a parametrised probabilistic model.

Signal $u$ can be used to optimise the kernel by adjusting the parameters of the model.

Using HMMs for modelling time series data this approach was applied to forecasting foreign exchange rates.

Some encouraging results:
Learning Fisher kernels

As an example consider Fisher kernels over a parametrised probabilistic model

Signal $u$ can be used to optimise the kernel by adjusting the parameters of the model

Using HMMs for modelling time series data this approach was applied to forecasting foreign exchange rates.

Some encouraging results
Learning Fisher kernels

- As an example consider Fisher kernels over a parametrised probabilistic model
- Signal $u$ can be used to optimise the kernel by adjusting the parameters of the model
- Using HMMs for modelling time series data this approach was applied to forecasting foreign exchange rates.

Some encouraging results

Learning Fisher kernels

- As an example consider Fisher kernels over a parametrised probabilistic model.
- Signal $u$ can be used to optimise the kernel by adjusting the parameters of the model.
- Using HMMs for modelling time series data this approach was applied to forecasting foreign exchange rates.
- Some encouraging results:
Non-linear Feature Selection

There is an interesting result that relates kernel target alignment to maximal covariance with the output

\[
\sqrt{E(x,y) \sim P, (x',y') \sim P [yy' \kappa(x, x')]} = \\
= \sup_{w: ||w|| \leq 1} E(x,y) \sim P [y \langle w, \phi(x) \rangle]
\]

Suggests defining the contribution of a feature as

\[
c_i = E_{S \sim S_i} [E(x,y) \sim P, (x',y') \sim P [yy' \kappa_S(x, x')]] - E_{S' \sim S \setminus i} [E(x,y) \sim P, (x',y') \sim P [yy' \kappa_{S'}(x, x')]],
\]

where \( S_i \) and \( S \setminus i \) are distributions over fixed size sets of features.
Non-linear Feature Selection

- There is an interesting result that relates kernel target alignment to maximal covariance with the output

\[
\sqrt{E_{(x,y) \sim P,(x',y') \sim P}[yy' \kappa(x, x')]} = \\
= \sup_{w: \|w\| \leq 1} E_{(x,y) \sim P}[y \langle w, \phi(x) \rangle]
\]

- Suggests defining the contribution of a feature as

\[
c_i = E_{S \sim S_i} [E_{(x,y) \sim P,(x',y') \sim P}[yy' \kappa_S(x, x')]] - E_{S' \sim S \setminus i} [E_{(x,y) \sim P,(x',y') \sim P}[yy' \kappa_{S'}(x, x')]],
\]

where \(S_i\) and \(S \setminus i\) are distributions over fixed size sets of features.
Consider 200-dimensional function that is XOR of the first two features. Take Gaussian kernel - gives results after successive cullings:
Some properties:

- Irrelevant features make negative contribution.
- Chances of relevant feature being in bottom quarter of the ranked contributions on a sufficiently large random sample is arbitrarily small.
- Hence, can cull 25% of bottom ranked features without risking losing good features.
- Possibility of locking in features that appear in top 25% consistently.
Some properties:

- Irrelevant features make negative contribution
- Chances of relevant feature being in bottom quarter of the ranked contributions on a sufficiently large random sample is arbitrarily small
- Hence, can cull 25% of bottom ranked features without risking losing good features
- Possibility of locking in features that appear in top 25% consistently
Some properties:

- Irrelevant features make negative contribution
- Chances of relevant feature being in bottom quarter of the ranked contributions on a sufficiently large random sample is arbitrarily small
- Hence, can cull 25% of bottom ranked features without risking losing good features
- Possibility of locking in features that appear in top 25% consistently
Some properties:

- Irrelevant features make negative contribution
- Chances of relevant feature being in bottom quarter of the ranked contributions on a sufficiently large random sample is arbitrarily small
- Hence, can cull 25% of bottom ranked features without risking losing good features
- Possibility of locking in features that appear in top 25% consistently
Some properties:

- Irrelevant features make negative contribution
- Chances of relevant feature being in bottom quarter of the ranked contributions on a sufficiently large random sample is arbitrarily small
- Hence, can cull 25% of bottom ranked features without risking losing good features
- Possibility of locking in features that appear in top 25% consistently
## Results

### On artificial data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Weston</td>
<td>randSel</td>
<td>97.7 ± 2.0</td>
<td>3.0 ± 0.0</td>
<td>91.8 ± 23.1</td>
<td>72.0 ± 16.6</td>
</tr>
<tr>
<td></td>
<td>BaHsic</td>
<td>97.3 ± 3.1</td>
<td>5.0 ± 0.0</td>
<td>91.5 ± 19.4</td>
<td>70.7 ± 14.9</td>
</tr>
<tr>
<td></td>
<td>FoHsic</td>
<td>97.1 ± 3.1</td>
<td>6.0 ± 0.0</td>
<td>95.9 ± 12.0</td>
<td>74.7 ± 17.7</td>
</tr>
<tr>
<td></td>
<td>Corr. Coeff.</td>
<td>92.4 ± 7.8</td>
<td>4.0 ± 0.0</td>
<td>96.1 ± 15.1</td>
<td>76.0 ± 15.5</td>
</tr>
<tr>
<td></td>
<td>Stab. Sel.</td>
<td>97.3 ± 3.1</td>
<td>2.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>40.0 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>RFE</td>
<td>95.3 ± 3.9</td>
<td>5.0 ± 0.0</td>
<td>66.9 ± 33.7</td>
<td>56.0 ± 13.5</td>
</tr>
<tr>
<td>Non-Linear Weston</td>
<td>randSel</td>
<td>99.0 ± 1.4</td>
<td>5.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>89.3 ± 12.8</td>
</tr>
<tr>
<td></td>
<td>BaHsic</td>
<td>99.8 ± 0.9</td>
<td>4.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>80.0 ± 7.6</td>
</tr>
<tr>
<td></td>
<td>FoHsic</td>
<td>99.8 ± 0.9</td>
<td>4.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>82.7 ± 7.0</td>
</tr>
<tr>
<td></td>
<td>Corr. Coeff.</td>
<td>56.2 ± 6.8</td>
<td>21.0 ± 0.0</td>
<td>1.7 ± 2.5</td>
<td>18.7 ± 31.6</td>
</tr>
<tr>
<td></td>
<td>Stab. Sel.</td>
<td>50.0 ± 7.1</td>
<td>2.0 ± 0.0</td>
<td>0.0 ± 0.0</td>
<td>0.0 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>RFE</td>
<td>98.9 ± 2.7</td>
<td>5.0 ± 0.0</td>
<td>97.8 ± 5.9</td>
<td>100.0 ± 0.0</td>
</tr>
<tr>
<td>XOR</td>
<td>randSel</td>
<td>95.7 ± 3.3</td>
<td>2.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>BaHsic</td>
<td>95.7 ± 3.3</td>
<td>2.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
<td>100.0 ± 0.0</td>
</tr>
<tr>
<td></td>
<td>FoHsic</td>
<td>52.0 ± 6.5</td>
<td>53.0 ± 0.0</td>
<td>9.4 ± 25.3</td>
<td>36.7 ± 44.2</td>
</tr>
<tr>
<td></td>
<td>Corr. Coeff.</td>
<td>58.1 ± 14.9</td>
<td>8.0 ± 0.0</td>
<td>10.4 ± 10.3</td>
<td>50.0 ± 42.3</td>
</tr>
<tr>
<td></td>
<td>Stab. Sel.</td>
<td>49.3 ± 11.1</td>
<td>2.0 ± 0.0</td>
<td>13.3 ± 22.9</td>
<td>13.3 ± 22.9</td>
</tr>
<tr>
<td></td>
<td>RFE</td>
<td>91.8 ± 12.1</td>
<td>2.0 ± 0.0</td>
<td>96.7 ± 12.9</td>
<td>96.7 ± 12.9</td>
</tr>
</tbody>
</table>
# Results

## On real world omic and microarray data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Features</th>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB</td>
<td>randSel</td>
<td>82.9 ± 8.4</td>
<td>64.6 ± 70.3</td>
<td>TB</td>
<td>randSel</td>
<td>82.0 ± 8.6</td>
<td>42.0 ± 47.7</td>
</tr>
<tr>
<td>Task 1</td>
<td>BaHsic</td>
<td>81.7 ± 9.0</td>
<td>74.7 ± 101.3</td>
<td>Task 2</td>
<td>BaHsic</td>
<td>81.1 ± 8.9</td>
<td>33.1 ± 40.6</td>
</tr>
<tr>
<td></td>
<td>FoHsic</td>
<td>81.3 ± 9.4</td>
<td>68.0 ± 66.5</td>
<td></td>
<td>FoHsic</td>
<td>80.6 ± 10.8</td>
<td>31.1 ± 35.3</td>
</tr>
<tr>
<td></td>
<td>Corr. Coeff.</td>
<td>82.4 ± 8.8</td>
<td>123.6 ± 85.8</td>
<td></td>
<td>Corr. Coeff.</td>
<td>82.7 ± 9.4</td>
<td>73.4 ± 55.5</td>
</tr>
<tr>
<td></td>
<td>Stab. Sel.</td>
<td>82.9 ± 7.3</td>
<td>121.7 ± 56.4</td>
<td></td>
<td>Stab. Sel.</td>
<td>80.7 ± 8.4</td>
<td>137.3 ± 154.7</td>
</tr>
<tr>
<td></td>
<td>RFE</td>
<td>81.9 ± 8.0</td>
<td>236.2 ± 160.2</td>
<td></td>
<td>RFE</td>
<td>80.2 ± 9.1</td>
<td>82.4 ± 139.9</td>
</tr>
<tr>
<td>TB</td>
<td>randSel</td>
<td>86.0 ± 8.1</td>
<td>45.3 ± 33.6</td>
<td>TB</td>
<td>randSel</td>
<td>87.6 ± 4.9</td>
<td>58.5 ± 93.8</td>
</tr>
<tr>
<td>Task 3</td>
<td>BaHsic</td>
<td>85.6 ± 9.5</td>
<td>53.3 ± 39.5</td>
<td>Micro</td>
<td>BaHsic</td>
<td>86.1 ± 6.4</td>
<td>61.2 ± 94.7</td>
</tr>
<tr>
<td></td>
<td>FoHsic</td>
<td>85.6 ± 8.8</td>
<td>53.6 ± 44.7</td>
<td>Array</td>
<td>FoHsic</td>
<td>85.2 ± 7.9</td>
<td>52.5 ± 92.9</td>
</tr>
<tr>
<td></td>
<td>Corr. Coeff.</td>
<td>85.4 ± 8.8</td>
<td>132.9 ± 89.7</td>
<td></td>
<td>Corr. Coeff.</td>
<td>84.1 ± 6.6</td>
<td>143.5 ± 114.2</td>
</tr>
<tr>
<td></td>
<td>Stab. Sel.</td>
<td>84.1 ± 9.6</td>
<td>60.0 ± 47.9</td>
<td></td>
<td>Stab. Sel.</td>
<td>87.1 ± 5.9</td>
<td>161.8 ± 136.0</td>
</tr>
<tr>
<td></td>
<td>RFE</td>
<td>83.9 ± 9.2</td>
<td>43.5 ± 71.6</td>
<td></td>
<td>RFE</td>
<td>85.7 ± 6.8</td>
<td>158.0 ± 137.6</td>
</tr>
</tbody>
</table>
Results

Have applied to Deep learning challenge (Black Box Learning Challenge 2013)

- Initial sparse filtering step (Jiquan et al., 2011) – just one preprocessing layer
- performed the culling steps described above
- used the LPBoost MKL method to combine the corresponding kernels created
- Method was third in the final ranking (scored 0.685 vs winning score of 0.702)
Results

Have applied to Deep learning challenge (Black Box Learning Challenge 2013)

- Initial sparse filtering step (Jiquan et al., 2011) – just one preprocessing layer
  - performed the culling steps described above
  - used the LPBoost MKL method to combine the corresponding kernels created

Method was third in the final ranking (scored 0.685 vs winning score of 0.702)
Have applied to Deep learning challenge (Black Box Learning Challenge 2013)

- Initial sparse filtering step (Jiquan et al., 2011) – just one preprocessing layer
- performed the culling steps described above
- used the LPBoost MKL method to combine the corresponding kernels created
- Method was third in the final ranking (scored 0.685 vs winning score of 0.702)
Have applied to Deep learning challenge (Black Box Learning Challenge 2013)

- Initial sparse filtering step (Jiquan et al., 2011) – just one preprocessing layer
- performed the culling steps described above
- used the LPBoost MKL method to combine the corresponding kernels created
- Method was third in the final ranking (scored 0.685 vs winning score of 0.702)
Results

Have applied to Deep learning challenge (Black Box Learning Challenge 2013)

- Initial sparse filtering step (Jiquan et al., 2011) – just one preprocessing layer
- performed the culling steps described above
- used the LPBoost MKL method to combine the corresponding kernels created
- Method was third in the final ranking (scored 0.685 vs winning score of 0.702)
Learning deep representations is important for analysis of real data

- Many kernel practitioners are using deep learning but typically in a relatively ad-hoc manner
- Attempts to use more principled methods have been rewarded with considerable success
- There is already a range of theoretical results relating to deep-er learning kernel methods that place the approaches on a firm-er footing
Learning deep representations is important for analysis of real data.

Many kernel practitioners are using deep learning but typically in a relatively ad-hoc manner.

Attempts to use more principled methods have been rewarded with considerable success.

There is already a range of theoretical results relating to deep-er learning kernel methods that place the approaches on a firm-er footing.
Summary and Conclusions

- Learning deep representations is important for analysis of real data
- Many kernel practitioners are using deep learning but typically in a relatively ad-hoc manner
- Attempts to use more principled methods have been rewarded with considerable success
- There is already a range of theoretical results relating to deep-er learning kernel methods that place the approaches on a firm-er footing
Summary and Conclusions

- Learning deep representations is important for analysis of real data
- Many kernel practitioners are using deep learning but typically in a relatively ad-hoc manner
- Attempts to use more principled methods have been rewarded with considerable success
- There is already a range of theoretical results relating to deep-er learning kernel methods that place the approaches on a firm-er footing