accelerated kernel learning[1]

purushottam kar

department of computer science and engineering
indian institute of technology kanpur

november 27, 2012

[1] joint work with harish c. karnick

menu del dia

- learning (7 slides)

menu del dia

- learning (7 slides)
  - introduction to machine learning

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)
  - introduction to kernel learning

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)
  - introduction to kernel learning
  - issues in kernel learning

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)
  - introduction to kernel learning
  - issues in kernel learning
- accelerated kernel learning (11 slides)

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)
  - introduction to kernel learning
  - issues in kernel learning
- accelerated kernel learning (11 slides)
  - random features

menu del dia

- learning (7 slides)
  - introduction to machine learning
  - issues in learning
- kernel learning (6 slides)
  - introduction to kernel learning
  - issues in kernel learning
- accelerated kernel learning (11 slides)
  - random features
  - other methods

learning 101

- why machine learning ?

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user
  - predict stock market prices

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user
  - predict stock market prices
  - predict new friends for a facebook user

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user
  - predict stock market prices
  - predict new friends for a facebook user
- how does one do machine learning ?

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user
  - predict stock market prices
  - predict new friends for a facebook user
- how does one do machine learning ?
  - discover patterns in data

learning 101

- why machine learning ?
  - automate tasks that are difficult for humans
- where is machine learning used ?
  - point out spam mails for a gmail user
  - predict stock market prices
  - predict new friends for a facebook user
- how does one do machine learning ?
  - discover patterns in data
  - what sort of patterns ?

ml task 1 : classification

ml task 1 : classification

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects
- observe a gmail user as he tags his mails as spam or useful

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects
- observe a gmail user as he tags his mails as spam or useful
  - can we figure out a pattern ?

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects
- observe a gmail user as he tags his mails as spam or useful
  - can we figure out a pattern ?
  - can we automatically detect spam mails for him ?

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects
- observe a gmail user as he tags his mails as spam or useful
  - can we figure out a pattern ?
  - can we automatically detect spam mails for him ?
  - can we use his patterns to tag his girlfriend's emails ?

ml task 1 : classification

- goal : find a way to assign the "correct" label to a set of objects
- observe a gmail user as he tags his mails as spam or useful
  - can we figure out a pattern ?
  - can we automatically detect spam mails for him ?
  - can we use his patterns to tag his girlfriend's emails ?

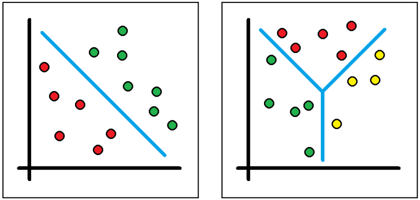

figure: linear classification

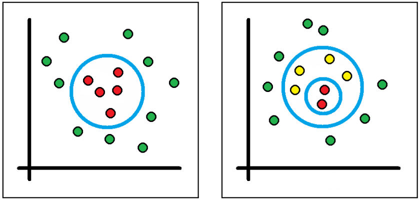

figure: non-linear classification

ml task 2 : regression

ml task 2 : regression

ml task 2 : regression

- goal : more like generalized curve fitting

ml task 2 : regression

- goal : more like generalized curve fitting
- observe variables such as company performance, past trends etc and the stock prices of a given company

ml task 2 : regression

- goal : more like generalized curve fitting
- observe variables such as company performance, past trends etc and the stock prices of a given company
  - can we predict today's stock prices for the company ?

ml task 2 : regression

- goal : more like generalized curve fitting
- observe variables such as company performance, past trends etc and the stock prices of a given company
  - can we predict today's stock prices for the company ?
- no "labels" here

ml task 2 : regression

- goal : more like generalized curve fitting
- observe variables such as company performance, past trends etc and the stock prices of a given company
    - can we predict today's stock prices for the company ?
- no "labels" here
    - non-discrete pattern

ml task 2 : regression

- goal : more like generalized curve fitting
- observe variables such as company performance, past trends etc and the stock prices of a given company
  - can we predict today's stock prices for the company ?
- no "labels" here
  - non-discrete pattern

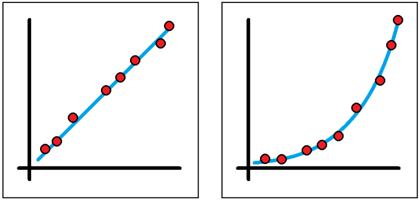

figure: real valued regression

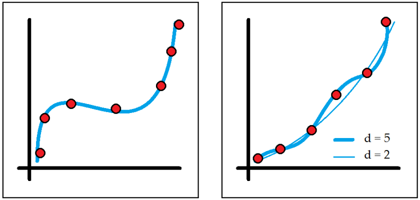

figure: dangers of overfitting

other ml tasks

other ml tasks

other ml tasks

- ranking

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering
  - given genome data, discover familia, genera and species

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering
  - given genome data, discover familia, genera and species
- component analysis

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering
  - given genome data, discover familia, genera and species
- component analysis
  - find principal or independent components in data

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering
  - given genome data, discover familia, genera and species
- component analysis
  - find principal or independent components in data
  - useful in signal processing, dimensionality reduction

other ml tasks

- ranking
  - find the top 10 facebook users with whom I am likely to make friends
- clustering
  - given genome data, discover familia, genera and species
- component analysis
  - find principal or independent components in data
  - useful in signal processing, dimensionality reduction

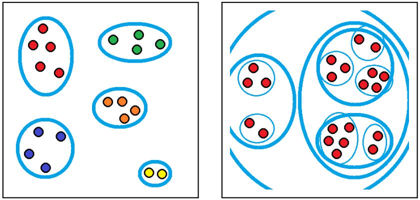

figure: clustering problems

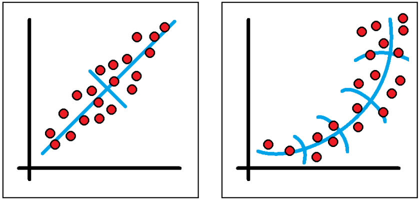

figure: principal component analysis

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
    - emails, stocks, facebook users, living organisms, analog signals
    - set may be discrete/continuous, finite/infinite
    - may have a variety of structure (topological/geometric)

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting
  - classification : discrete label set : $\mathcal{Y} = \{\pm 1\}$ for spam classification

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting
  - classification : discrete label set : $\mathcal{Y} = \{\pm 1\}$ for spam classification
  - regression : continuous label set : $\mathcal{Y} \subset \mathbb{R}$

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting
  - classification : discrete label set : $\mathcal{Y} = \{\pm 1\}$ for spam classification
  - regression : continuous label set : $\mathcal{Y} \subset \mathbb{R}$
  - ranking, clustering, component analysis : more structured label sets

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting
  - classification : discrete label set : $\mathcal{Y} = \{\pm 1\}$ for spam classification
  - regression : continuous label set : $\mathcal{Y} \subset \mathbb{R}$
  - ranking, clustering, component analysis : more structured label sets
- true pattern : $f^* : \mathcal{X} \longrightarrow \mathcal{Y}$

a mathematical abstraction

- domain : a set $\mathcal{X}$ of objects we are interested in
  - emails, stocks, facebook users, living organisms, analog signals
  - set may be discrete/continuous, finite/infinite
  - may have a variety of structure (topological/geometric)
- label set : the property $\mathcal{Y}$ of the objects we are interested in predicting
  - classification : discrete label set : $\mathcal{Y} = \{\pm 1\}$ for spam classification
  - regression : continuous label set : $\mathcal{Y} \subset \mathbb{R}$
  - ranking, clustering, component analysis : more structured label sets
- true pattern : $f^* : \mathcal{X} \longrightarrow \mathcal{Y}$
  - mathematically captures the notion of "correct" labellings

- supervised learning

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern
- formalizing closeness of hypothesis to true pattern

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern
- formalizing closeness of hypothesis to true pattern
  - how often do we give out a wrong answer : $\mathbb{P}[h(\mathbf{x}) \neq f^*(\mathbf{x})]$

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern
- formalizing closeness of hypothesis to true pattern
  - how often do we give out a wrong answer : $\mathbb{P}[h(\mathbf{x}) \neq f^*(\mathbf{x})]$
  - more generally, utilize loss functions : $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern
- formalizing closeness of hypothesis to true pattern
  - how often do we give out a wrong answer : $\mathbb{P}[h(\mathbf{x}) \neq f^*(\mathbf{x})]$
  - more generally, utilize loss functions : $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$
    - closeness defined as average loss : $\mathbb{E}[\![\ell(h(\mathbf{x}), f^*(\mathbf{x}))]\!]$

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today

- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\left\{\left(\mathbf{x}_1, f^*\left(\mathbf{x}_1\right)\right), \left(\mathbf{x}_2, f^*\left(\mathbf{x}_2\right)\right), \ldots, \left(\mathbf{x}_n, f^*\left(\mathbf{x}_n\right)\right)\right\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern

- formalizing closeness of hypothesis to true pattern
  - how often do we give out a wrong answer : $\mathbb{P}\left[h(\mathbf{x}) \neq f^*(\mathbf{x})\right]$
  - more generally, utilize loss functions : $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$
    - closeness defined as average loss : $\mathbb{E}\left[\!\left[\ell\left(h(\mathbf{x}), f^*(\mathbf{x})\right)\right]\!\right]$
    - zero-one loss : $\ell(y_1, y_2) = \mathbb{1}_{y_1 \neq y_2}$ (for classification)

the learning process

- supervised learning
  - includes tasks such as classification, regression, ranking
  - shall not discuss *unsupervised*, *semi-supervised* learning today
- learn from the teacher
  - we are given access to lots of domain elements with their true labels
  - training set : $\{(\mathbf{x}_1, f^*(\mathbf{x}_1)), (\mathbf{x}_2, f^*(\mathbf{x}_2)), \ldots, (\mathbf{x}_n, f^*(\mathbf{x}_n))\}$
  - hypothesis : a pattern $h : \mathcal{X} \longrightarrow \mathcal{Y}$ we infer using training data
  - goal : learn a hypothesis that is *close* to the true pattern
- formalizing closeness of hypothesis to true pattern
  - how often do we give out a wrong answer : $\mathbb{P}[h(\mathbf{x}) \neq f^*(\mathbf{x})]$
  - more generally, utilize loss functions : $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$
    - closeness defined as average loss : $\mathbb{E}[\![\ell(h(\mathbf{x}), f^*(\mathbf{x}))]\!]$
    - zero-one loss : $\ell(y_1, y_2) = \mathbb{1}_{y_1 \neq y_2}$ (for classification)
    - quadratic loss : $\ell(y_1, y_2) = (y_1 - y_2)^2$ (for regression)

issues in the learning process

- how to learn a hypothesis from a training set

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?
- how many training points should i choose ?

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?
- how many training points should i choose ?
- how do i output my hypothesis to the end user ?

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?
- how many training points should i choose ?
- how do i output my hypothesis to the end user ?
- ...

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?
- how many training points should i choose ?
- how do i output my hypothesis to the end user ?
- ...
- shall only address the first and the last issue in this talk

issues in the learning process

- how to learn a hypothesis from a training set
- how do i select my training set ?
- how many training points should i choose ?
- how do i output my hypothesis to the end user ?
- ...
- shall only address the first and the last issue in this talk
- shall find the nearest carpet for rest of the issues

kernel learning 101

- take the example of spam classification

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before
- how to quantify "similarity" ?

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before
- how to quantify "similarity" ?
  - a bivariate function $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before
- how to quantify "similarity" ?
  - a bivariate function $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$
  - e.g. the dot product in euclidean spaces

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before
- how to quantify "similarity" ?
  - a bivariate function $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$
  - e.g. the dot product in euclidean spaces
    - $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle := \left\| \mathbf{x}_1 \right\|_2 \left\| \mathbf{x}_2 \right\|_2 \cos(\angle(\mathbf{x}_1, \mathbf{x}_2))$

kernel learning 101

- take the example of spam classification
- assume that emails that look similar have the same label
  - essentially saying that the true pattern is *smooth*
  - can infer the label of a new email using labels of emails seen before
- how to quantify "similarity" ?
  - a bivariate function $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$
  - e.g. the dot product in euclidean spaces
    - $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle := \|\mathbf{x}_1\|_2 \|\mathbf{x}_2\|_2 \cos(\angle(\mathbf{x}_1, \mathbf{x}_2))$
  - e.g. number of shared friends on facebook

learning using similarities

- a new email can be given the label of the most similar email in the training set

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise

- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence
  - some training emails are more useful than others

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence
  - some training emails are more useful than others
  - more resilient to noise but still can be slow

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence
  - some training emails are more useful than others
  - more resilient to noise but still can be slow
- kernel learning uses hypotheses of the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence
  - some training emails are more useful than others
  - more resilient to noise but still can be slow
- kernel learning uses hypotheses of the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

  - α_i denotes the usefulness of training email $\mathbf{x}_i$

learning using similarities

- a new email can be given the label of the most similar email in the training set
  - not a good idea : would be slow and prone to noise
- take all training emails and ask them to vote
  - training emails that are similar to new email have more influence
  - some training emails are more useful than others
  - more resilient to noise but still can be slow
- kernel learning uses hypotheses of the form

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

  - α_i denotes the usefulness of training email $\mathbf{x}_i$
  - for classification one uses $\mathrm{sign}(h(x))$

a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ (linear kernel)

  $$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle = \left\langle \mathbf{x}, \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right\rangle = \langle \mathbf{x}, \mathbf{w} \rangle \text{ (linear hypothesis)}$$

a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ (linear kernel)

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle = \left\langle \mathbf{x}, \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right\rangle = \langle \mathbf{x}, \mathbf{w} \rangle \text{ (linear hypothesis)}$$

- if α_i were absent then $w = \sum_{y_i=1} \mathbf{x}_i - \sum_{y_i=-1} \mathbf{x}_j$: weaker model

a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ (linear kernel)
  $$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle = \left\langle \mathbf{x}, \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right\rangle = \langle \mathbf{x}, \mathbf{w} \rangle \text{ (linear hypothesis)}$$
- if α_i were absent then $w = \sum_{y_i=1} \mathbf{x}_i - \sum_{y_i=-1} \mathbf{x}_j$: weaker model
- α_i found by solving an optimization problem : details out of scope

a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ (linear kernel)

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle = \left\langle \mathbf{x}, \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right\rangle = \langle \mathbf{x}, \mathbf{w} \rangle \text{ (linear hypothesis)}$$

- if α_i were absent then $w = \sum_{y_i=1} \mathbf{x}_i - \sum_{y_j=-1} \mathbf{x}_j$: weaker model
- α_i found by solving an optimization problem : details out of scope

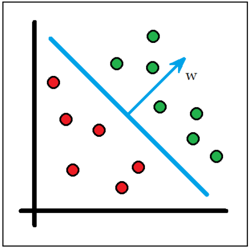

figure: linear classifier

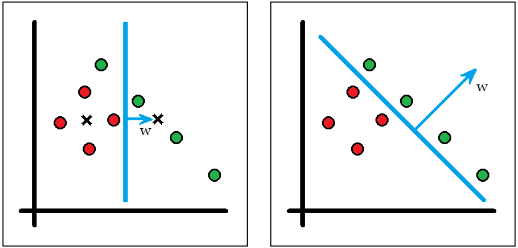

figure: utility of weight variables α_i

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used
  - most often, mercer kernels are used

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used
  - most often, mercer kernels are used
- mercer kernels satisfy the conditions of the mercer's theorem

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used
  - most often, mercer kernels are used
- mercer kernels satisfy the conditions of the mercer's theorem
  - loosely speaking, they correspond to measures of similarity that are actually inner products in some hilbert space

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used
  - most often, mercer kernels are used
- mercer kernels satisfy the conditions of the mercer's theorem
  - loosely speaking, they correspond to measures of similarity that are actually inner products in some hilbert space
  - more formally, a similarity function K is a mercer kernel if there exists a map $\Phi : \mathcal{X} \longrightarrow \mathcal{H}$ to some hilbert space $\mathcal{H}$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle$

enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
  - in practice more complex notions of similarity are used
  - most often, mercer kernels are used
- mercer kernels satisfy the conditions of the mercer's theorem
  - loosely speaking, they correspond to measures of similarity that are actually inner products in some hilbert space
  - more formally, a similarity function K is a mercer kernel if there exists a map $\Phi : \mathcal{X} \longrightarrow \mathcal{H}$ to some hilbert space $\mathcal{H}$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle$
- mercer kernels give us hypotheses that are linear in the hilbert space

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle \text{ for some } \mathbf{w} \in \mathcal{H}$$

a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $\mathbf{x} = (p, q)$ and $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^2$

a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $\mathbf{x} = (p, q)$ and $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^2$
- one can show that the corresponding map is six dimensional

$$\Phi(\mathbf{x}) = \left(p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1\right) \in \mathbb{R}^6$$

a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $\mathbf{x} = (p, q)$ and $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^2$
- one can show that the corresponding map is six dimensional

$$\Phi(\mathbf{x}) = \left(p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1\right) \in \mathbb{R}^6$$

- it is able to implement quadratic hypotheses

a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $\mathbf{x} = (p, q)$ and $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^2$
- one can show that the corresponding map is six dimensional

$$\Phi(\mathbf{x}) = \left(p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1\right) \in \mathbb{R}^6$$

- it is able to implement quadratic hypotheses
  - e.g. $h(\mathbf{x}) = p^2 + q^2 - 1$ for $\mathbf{w} = (1, 1, 0, 0, 0, -1)$

a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $\mathbf{x} = (p, q)$ and $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^2$
- one can show that the corresponding map is six dimensional

$$\Phi(\mathbf{x}) = \left(p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1\right) \in \mathbb{R}^6$$

- it is able to implement quadratic hypotheses
  - e.g. $h(\mathbf{x}) = p^2 + q^2 - 1$ for $\mathbf{w} = (1, 1, 0, 0, 0, -1)$

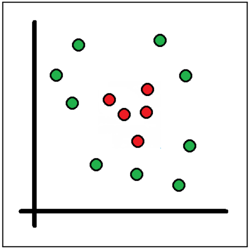

figure: non linear problem

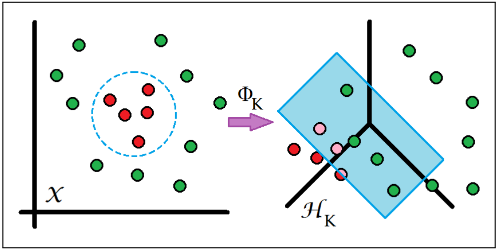

figure: kernel trick in action

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ
  - the kernel trick : can compute $K(\mathbf{x}_1, \mathbf{x}_2)$ without computing Φ

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ
  - the kernel trick : can compute $K(\mathbf{x}_1, \mathbf{x}_2)$ without computing Φ
  - have to use the implicit form $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$: slow

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ
  - the kernel trick : can compute $K(\mathbf{x}_1, \mathbf{x}_2)$ without computing Φ
  - have to use the implicit form $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$: slow
- why only mercer kernels ?

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ
  - the kernel trick : can compute $K(\mathbf{x}_1, \mathbf{x}_2)$ without computing Φ
  - have to use the implicit form $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$: slow
- why only mercer kernels ?
  - for algorithmic convenience and a clean theory

issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map Φ
  - the kernel trick : can compute $K(\mathbf{x}_1, \mathbf{x}_2)$ without computing Φ
  - have to use the implicit form $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$: slow
- why only mercer kernels ?
  - for algorithmic convenience and a clean theory
  - can use non-mercer *indefinite* kernels as well : out of scope

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$
    - requires a single operation but in a high dimensional space

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$
    - requires a single operation but in a high dimensional space
- can we find an approximate map for the kernel in some low dimensional space ?

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$
    - requires a single operation but in a high dimensional space
- can we find an approximate map for the kernel in some low dimensional space ?
  - $Z : \mathcal{X} \longrightarrow \mathbb{R}^D$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $\langle Z(\mathbf{x}_1), Z(\mathbf{x}_2) \rangle \approx K(\mathbf{x}_1, \mathbf{x}_2)$

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$
    - requires a single operation but in a high dimensional space
- can we find an approximate map for the kernel in some low dimensional space ?
  - $Z : \mathcal{X} \longrightarrow \mathbb{R}^D$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $\langle Z(\mathbf{x}_1), Z(\mathbf{x}_2) \rangle \approx K(\mathbf{x}_1, \mathbf{x}_2)$
  - $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathbb{R}^D$

fast kernel learning : the basic idea

- two ways of representing mercer kernel hypotheses
  - $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$
    - requires upto n (and in practice $\Omega(n)$) operations
  - $h(\mathbf{x}) = \langle \Phi(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathcal{H}$
    - requires a single operation but in a high dimensional space
- can we find an approximate map for the kernel in some low dimensional space ?
  - $Z : \mathcal{X} \longrightarrow \mathbb{R}^D$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $\langle Z(\mathbf{x}_1), Z(\mathbf{x}_2) \rangle \approx K(\mathbf{x}_1, \mathbf{x}_2)$
  - $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$ for some $\mathbf{w} \in \mathbb{R}^D$
  - would get power of kernel as well as speed of linear hypothesis

the underlying math

- why should such approximate maps exist ?

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
- problem ??

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
- problem ??
  - all algorithmic implementations of the jl-lemma require explicit access to $\mathbf{x}_i \in \mathcal{H}$

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
- problem ??
  - all algorithmic implementations of the jl-lemma require explicit access to $\mathbf{x}_i \in \mathcal{H}$
    - for us, calculating vectors in the hilbert space is prohibitive

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
- problem ??
  - all algorithmic implementations of the jl-lemma require explicit access to $\mathbf{x}_i \in \mathcal{H}$
    - for us, calculating vectors in the hilbert space is prohibitive
  - the number of dimensions depends upon the number of points

the underlying math

- why should such approximate maps exist ?
- johnson-lindenstrauss flattening lemma [cont. math., 26:189–206, 1984.]
  - given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \longrightarrow \mathbb{R}^D$
    - for all i, j, $\langle \Psi(\mathbf{x}_i), \Psi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \pm \epsilon$
    - need $D = \mathcal{O}\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
- problem ??
  - all algorithmic implementations of the jl-lemma require explicit access to $\mathbf{x}_i \in \mathcal{H}$
    - for us, calculating vectors in the hilbert space is prohibitive
  - the number of dimensions depends upon the number of points
    - not satisfactory

the underlying math

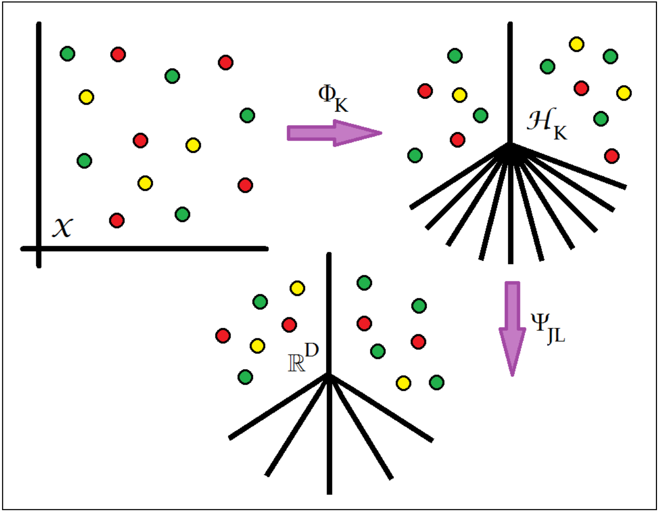

figure: dimensionality reduction via jl transform

structure theorems

- characterization of certain kernel families

structure theorems

- characterization of certain kernel families

bochner's theorem [rudin, fourier analysis on groups, 1962]

every translation invariant mercer kernel on a locally compact abelian group is the fourier-steiltjes transform of some bounded positive measure on the pontryagin dual group, $K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Gamma \gamma(\mathbf{x}_1 - \mathbf{x}_2)\, d\mu(\gamma)$

structure theorems

- characterization of certain kernel families

bochner's theorem [rudin, fourier analysis on groups, 1962]

every translation invariant mercer kernel on a locally compact abelian group is the fourier-steiltjes transform of some bounded positive measure on the pontryagin dual group, $K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Gamma \gamma\left(\mathbf{x}_1 - \mathbf{x}_2\right) d\mu(\gamma)$

schoenberg's theorem [duke math. journ., 9(1):96–108, 1942]

every dot product mercer kernel arises from an analytic function having a maclaurin series with non-negative coefficients, $K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=0}^{\infty} a_n \left\langle \mathbf{x}_1, \mathbf{x}_2 \right\rangle^n$

- characterization of certain kernel families

bochner's theorem [rudin, fourier analysis on groups, 1962]

every translation invariant mercer kernel on a locally compact abelian group is the fourier-steiltjes transform of some bounded positive measure on the pontryagin dual group, $K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Gamma \gamma\,(\mathbf{x}_1 - \mathbf{x}_2)\, d\mu(\gamma)$

schoenberg's theorem [duke math. journ., 9(1):96–108, 1942]

every dot product mercer kernel arises from an analytic function having a maclaurin series with non-negative coefficients, $K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=0}^{\infty} a_n \left\langle \mathbf{x}_1, \mathbf{x}_2 \right\rangle^n$

- allows us to develop fast routines for radial basis, homogeneous and dot product kernels

random features : the basic idea

- a kernel whose map is one-dimensional is called a rank-one kernel

random features : the basic idea

- a kernel whose map is one-dimensional is called a rank-one kernel
- one can interpret structure theorems as telling us that every kernel is a positive combination of rank-one kernels, i.e. for $\mu \geq 0$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Omega K_\omega(\mathbf{x}_1, \mathbf{x}_2) d\mu(\omega) = \underset{\omega \sim \mu}{\mathbb{E}} [\![K_\omega(\mathbf{x}_1, \mathbf{x}_2)]\!]$$

where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a rank-one kernel i.e. for some $\Phi_\omega : \mathcal{X} \longrightarrow \mathbb{R}$, for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $K_\omega(\mathbf{x}_1, \mathbf{x}_2) = \Phi_\omega(\mathbf{x}_1) \cdot \Phi_\omega(\mathbf{x}_1)$

random features : the basic idea

- a kernel whose map is one-dimensional is called a rank-one kernel
- one can interpret structure theorems as telling us that every kernel is a positive combination of rank-one kernels, i.e. for $\mu \geq 0$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Omega K_\omega(\mathbf{x}_1, \mathbf{x}_2) d\mu(\omega) = \mathop{\mathbb{E}}_{\omega \sim \mu} [\![K_\omega(\mathbf{x}_1, \mathbf{x}_2)]\!]$$

  where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a rank-one kernel i.e. for some $\Phi_\omega : \mathcal{X} \longrightarrow \mathbb{R}$, for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $K_\omega(\mathbf{x}_1, \mathbf{x}_2) = \Phi_\omega(\mathbf{x}_1) \cdot \Phi_\omega(\mathbf{x}_1)$
- a random K_ω gives us an unbiased estimate of K on all pairs of points

random features : the basic idea

- a kernel whose map is one-dimensional is called a rank-one kernel
- one can interpret structure theorems as telling us that every kernel is a positive combination of rank-one kernels, i.e. for $\mu \geq 0$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \int_\Omega K_\omega(\mathbf{x}_1, \mathbf{x}_2) d\mu(\omega) = \mathop{\mathbb{E}}_{\omega \sim \mu} [\![K_\omega(\mathbf{x}_1, \mathbf{x}_2)]\!]$$

  where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a rank-one kernel i.e. for some $\Phi_\omega : \mathcal{X} \longrightarrow \mathbb{R}$, for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $K_\omega(\mathbf{x}_1, \mathbf{x}_2) = \Phi_\omega(\mathbf{x}_1) \cdot \Phi_\omega(\mathbf{x}_1)$
- a random K_ω gives us an unbiased estimate of K on all pairs of points
  - once we have an unbiased estimate for a quantity, independent repetitions can help reduce variance

random features : implementation

- select D values $\{\omega_1, \omega_2, \ldots, \omega_D\}$ randomly from distribution μ over Ω

random features : implementation

- select D values $\{\omega_1, \omega_2, \ldots, \omega_D\}$ randomly from distribution μ over Ω
- create the map

$$Z(\mathbf{x}) = (\Phi_{\omega_1}(\mathbf{x}), \Phi_{\omega_2}(\mathbf{x}), \ldots, \Phi_{\omega_D}(\mathbf{x})) \in \mathbb{R}^D$$

random features : implementation

- select D values $\{\omega_1, \omega_2, \ldots, \omega_D\}$ randomly from distribution μ over Ω
- create the map

$$Z(\mathbf{x}) = (\Phi_{\omega_1}(\mathbf{x}), \Phi_{\omega_2}(\mathbf{x}), \ldots, \Phi_{\omega_D}(\mathbf{x})) \in \mathbb{R}^D$$

theorem (approximation guarantee for random features)

for a compact domain $\mathcal{X} \subset \mathbb{R}^d$, for any $\epsilon, \delta > 0$, take $D = \mathcal{O}\left(\frac{d}{\epsilon^2} \log \frac{1}{\epsilon\delta}\right)$ and construct a D-dimensional map, then with probability $(1 - \delta)$,

$$\sup_{\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}} |K(\mathbf{x}_1, \mathbf{x}_2) - \langle Z(\mathbf{x}_1), Z(\mathbf{x}_2) \rangle| \leq \epsilon$$

- the guarantee is *uniform* unlike the jl-lemma guarantee

- the guarantee is *uniform* unlike the jl-lemma guarantee
  - holds for all (possibly infinite) pairs of points from $\mathcal{X}$

random features : properties

- the guarantee is *uniform* unlike the jl-lemma guarantee
  - holds for all (possibly infinite) pairs of points from $\mathcal{X}$
- hypothesis is of the form $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$, for some $\mathbf{w} \in \mathbb{R}^D$

random features : properties

- the guarantee is *uniform* unlike the jl-lemma guarantee
  - holds for all (possibly infinite) pairs of points from $\mathcal{X}$
- hypothesis is of the form $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$, for some $\mathbf{w} \in \mathbb{R}^D$
  - evaluating a hypothesis takes $\mathcal{O}(D)$ time

random features : properties

- the guarantee is *uniform* unlike the jl-lemma guarantee
  - holds for all (possibly infinite) pairs of points from $\mathcal{X}$
- hypothesis is of the form $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$, for some $\mathbf{w} \in \mathbb{R}^D$
  - evaluating a hypothesis takes $\mathcal{O}(D)$ time
- procedure gives approximation to the kernel function directly

random features : properties

- the guarantee is *uniform* unlike the jl-lemma guarantee
  - holds for all (possibly infinite) pairs of points from $\mathcal{X}$
- hypothesis is of the form $h(\mathbf{x}) = \langle Z(\mathbf{x}), \mathbf{w} \rangle$, for some $\mathbf{w} \in \mathbb{R}^D$
  - evaluating a hypothesis takes $\mathcal{O}(D)$ time
- procedure gives approximation to the kernel function directly
  - same random features can be used for different tasks : classification, regression etc

random features : properties

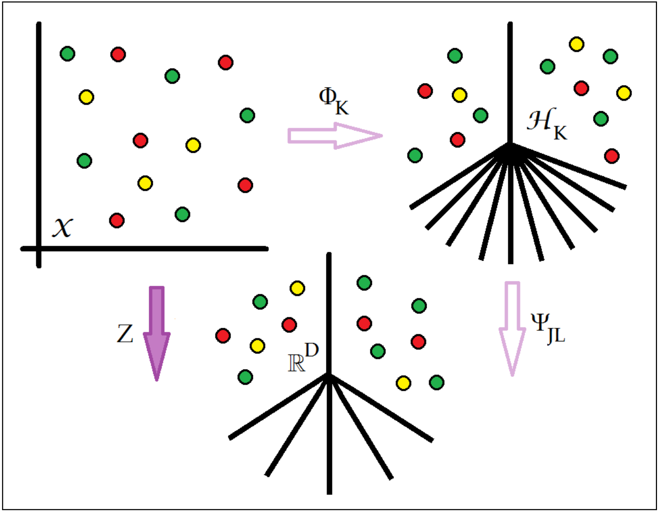

figure: random features providing dimensionality reduction

random features : in action

- several constructions for various families

random features : in action

- several constructions for various families
  - translation invariant kernels [rahimi, recht, nips 2007]

- several constructions for various families
  - translation invariant kernels [rahimi, recht, nips 2007]
  - homogeneous kernels [vedaldi, zisserman, cvpr 2010]

random features : in action

- several constructions for various families
    - translation invariant kernels [rahimi, recht, nips 2007]
    - homogeneous kernels [vedaldi, zisserman, cvpr 2010]
    - dot product kernels [k., karnick, aistats 2012]

random features : in action

- several constructions for various families
  - translation invariant kernels [rahimi, recht, nips 2007]
  - homogeneous kernels [vedaldi, zisserman, cvpr 2010]
  - dot product kernels [k., karnick, aistats 2012]

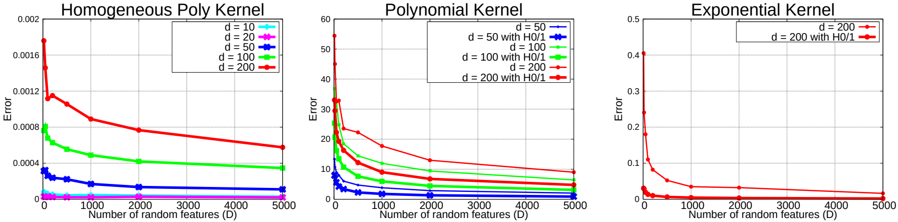

figure: approximation error in reconstructing kernel values

dataset	**K** + libsvm	**RF** + liblinear	**H0/1** + liblinear
`nursery` N = 13000 $d = 8$	acc = 99.8% trn = 10.8s tst = 1.7s	acc = 99.6% trn = 2.52s (**4.3**×) tst = 0.6s (**2.8**×) D = 500	acc = 97.96% trn = 0.4s (**27**×) tst = 0.18s (**9.4**×) D = 100
`cod-rna` N = 60000 $d = 8$	acc = 95.2% trn = 91.5s tst = 17.1s	acc = 94.9% trn = 11.5s (**8**×) tst = 2.8s (**6.1**×) D = 500	acc = 93.8% trn= 0.67s (**136**×) tst = 1.4s (**12**×) D = 50
`adult` N = 49000 $d = 123$	acc = 83.7% trn = 263.3s tst = 33.4s	acc = 82.9% trn = 39.8s (**6.6**×) tst = 14.3s (**2.3**×) D = 500	acc = 84.8% trn = 7.18s (**37**×) tst = 9.4s (**3.6**×) D = 100
`covertype` N=581000 $d = 54$	acc = 80.6% trn = 194.1s tst = 695.8s	acc = 76.2% trn = 21.4s (**9**×) tst = 207s (**3.6**×) D = 1000	acc = 75.5% trn = 3.7s (**52**×) tst = 80.4s (**8.7**×) D = 100

figure: speedups for exponential kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{\langle \mathbf{x}_1, \mathbf{x}_2 \rangle}{\sigma^2}\right)$

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map
- advantages

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map
- advantages
  - data dependency helps in hard learning instances [yang et al, nips 2010]

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map
- advantages
  - data dependency helps in hard learning instances [yang et al, nips 2010]
- disadvantages

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map
- advantages
  - data dependency helps in hard learning instances [yang et al, nips 2010]
- disadvantages
  - slower than random features as the hypothesis takes $\Omega\left(D^2\right)$ time to evaluate in worst case : $\mathcal{O}(D)$ time using random features

other approaches

- alternative approaches exist that given a set of training points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - cholesky decomposition : finds a rank D approximation to G
  - nyström method : chooses a subsample of training points $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_D$ as anchor points and creates a D dimensional map
- advantages
  - data dependency helps in hard learning instances [yang et al, nips 2010]
- disadvantages
  - slower than random features as the hypothesis takes $\Omega\left(D^2\right)$ time to evaluate in worst case : $\mathcal{O}\left(D\right)$ time using random features
  - expensive preprocessing required : increases time taken to learn

conclusion

- what all families admit such random feature constructions ?

conclusion

- what all families admit such random feature constructions ?
  - there do exist that dont [balcan et al., mach. learn., 65(1): 79–94, 2006]

conclusion

- what all families admit such random feature constructions ?
  - there do exist that dont [balcan et al., mach. learn., 65(1): 79–94, 2006]
- introduce data awareness in methods

conclusion

- what all families admit such random feature constructions ?
  - there do exist that dont [balcan et al., mach. learn., 65(1): 79–94, 2006]
- introduce data awareness in methods
- explore applications in other kernel learning tasks

conclusion

- what all families admit such random feature constructions ?
  - there do exist that dont [balcan et al., mach. learn., 65(1): 79–94, 2006]
- introduce data awareness in methods
- explore applications in other kernel learning tasks
  - some work in clustering [chitta et al., icdm 2012]