accelerated kernel learning

purushottam kar

department of computer science and engineering
indian institute of technology kanpur

november 27, 2012

\(^1\) joint work with harish c. karnick
Learning (7 slides)
learning (7 slides)
  introduction to machine learning
learning (7 slides)
  ▶ introduction to machine learning
  ▶ issues in learning
Learning (7 slides)
- Introduction to machine learning
- Issues in learning

Kernel learning (6 slides)
learning (7 slides)
  ▶ introduction to machine learning
  ▶ issues in learning

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

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

→ 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 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
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

- 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
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

purushottam kar (iit kanpur)
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

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

- clustering
  - given genome data, discover familia, genera and species

- principal 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
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
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
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
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

- **label set**: the property $\mathcal{Y}$ of the objects we are interested in predicting
  - **classification**: discrete label set: $\mathcal{Y} = \{-1, 1\}$ for spam classification
  - **regression**: continuous label set: $\mathcal{Y} \subseteq \mathbb{R}$
  - **ranking, clustering, component analysis**: more structured label sets

- **true pattern**: $f^*: \mathcal{X} \rightarrow \mathcal{Y}$
  - mathematically captures the notion of "correct" labelings
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

- label set: the property $Y$ of the objects we are interested in

- **classification**: discrete label set: $Y = \{\pm 1\}$ for spam classification

- **regression**: continuous label set: $Y \subset \mathbb{R}$

- **ranking, clustering, component analysis**: more structured label sets

- true pattern: $f^*: \mathcal{X} \rightarrow Y$

  mathematically captures the notion of "correct" labellings
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: $\mathcal{Y} = \{-1,1\}$ for spam classification
  - **regression**: continuous label set: $\mathcal{Y} \subseteq \mathbb{R}$
  - **ranking, clustering, component analysis**: more structured label sets

- **true pattern**: $f^* : \mathcal{X} \rightarrow \mathcal{Y}$ mathematically captures the notion of “correct” labelings
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**: $f^* : \mathcal{X} \rightarrow \mathcal{Y}$
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

- training set: \( \{ (x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n)) \} \)

- hypothesis: a pattern \( h: X \rightarrow Y \) 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: \( P[h(x) \neq f^*(x)] \)

- more generally, utilize loss functions: \( \ell: Y \times Y \rightarrow \mathbb{R} \)

- closeness defined as average loss: \( E[\ell(h(x), f^*(x))] \)

- zero-one loss: \( \ell(y_1, y_2) = 1 \) if \( y_1 \neq y_2 \) (for classification)

- quadratic loss: \( \ell(y_1, y_2) = (y_1 - y_2)^2 \) (for regression)
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 \textit{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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis: a pattern \( h : \mathcal{X} \rightarrow \mathcal{Y} \) 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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis: a pattern \( h : \mathcal{X} \rightarrow \mathcal{Y} \) 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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis: a pattern \( h: \mathcal{X} \to \mathcal{Y} \) 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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis: a pattern \( h: \mathcal{X} \rightarrow \mathcal{Y} \) 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: \( \mathbb{P}[h(x) \neq f^*(x)] \)
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: \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis: a pattern \( h: \mathcal{X} \rightarrow \mathcal{Y} \) 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: \( \mathbb{P}[h(x) \neq f^*(x)] \)
  - more generally, utilize loss functions: \( \ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \)
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 : \( \{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n))\} \)
  - hypothesis : a pattern \( h : \mathcal{X} \rightarrow \mathcal{Y} \) 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 : \( \mathbb{P}[h(x) \neq f^*(x)] \)
  - more generally, utilize loss functions : \( \ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \)
    - closeness defined as average loss : \( \mathbb{E}[\ell(h(x), f^*(x))] \)
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: \{ (x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n)) \}
  - hypothesis: a pattern \( h: \mathcal{X} \rightarrow \mathcal{Y} \) 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: \( \mathbb{P} [h(x) \neq f^*(x)] \)
  - more generally, utilize loss functions: \( \ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R} \)
    - closeness defined as average loss: \( \mathbb{E} [\ell(h(x), f^*(x))] \)
    - zero-one loss: \( \ell(y_1, y_2) = 1_{y_1 \neq y_2} \) (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 : \{ (x_1, f^*(x_1)), (x_2, f^*(x_2)), \ldots, (x_n, f^*(x_n)) \}\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 : \( \mathbb{P} [ h(x) \neq f^*(x) ] \)
  ▶ more generally, utilize loss functions : \( \ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R} \)
    ▶ closeness defined as average loss : \( \mathbb{E} [ \ell ( h(x), f^*(x) ) ] \)
    ▶ 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
take the example of spam classification

- how to quantify “similarity”?
  - a bivariate function $K: \mathbb{X} \times \mathbb{X} \to \mathbb{R}$
    - e.g. the dot product in euclidean spaces
      
      \[
      K(x_1, x_2) = \langle x_1, x_2 \rangle := \|x_1\|^2 \|x_2\|^2 \cos(\angle(x_1, x_2))
      \]
    - e.g. number of shared friends on facebook
kernel learning 101

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

assume that emails that look similar have the same label
  essentially saying that the true pattern is *smooth*
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
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”?
- 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 \( K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \)
• 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 $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$
  • e.g. the dot product in euclidean spaces
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 $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$
  e.g. the dot product in euclidean spaces
    $K(x_1, x_2) = \langle x_1, x_2 \rangle := \|x_1\|_2 \|x_2\|_2 \cos(\angle(x_1, x_2))$
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 $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$
- e.g. the dot product in euclidean spaces
  
  $K(x_1, x_2) = \langle x_1, x_2 \rangle := \|x_1\|_2 \|x_2\|_2 \cos(\angle(x_1, x_2))$

- 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

kernel learning uses hypotheses of the form

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

\( \alpha_i \) denotes the usefulness of training email \( x_i \)

for classification one uses sign(\( h(x) \))
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

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

\( \alpha_i \) denotes the usefulness of training email \( x_i \)

for classification one uses \( \text{sign}(h(x)) \)
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

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]
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

  \[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

  - \( \alpha_i \) denotes the usefulness of training email \( x_i \)
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

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

- \( \alpha_i \) denotes the usefulness of training email \( x_i \)
- for classification one uses \( \text{sign}(h(x)) \)
a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(x_1, x_2) = \langle x_1, x_2 \rangle$ (linear kernel)

$$h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle = \langle x, \sum_{i=1}^{n} \alpha_i y_i x_i \rangle = \langle x, w \rangle$$ (linear hypothesis)
a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(x_1, x_2) = \langle x_1, x_2 \rangle$ (linear kernel)

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

- if $\alpha_i$ were absent then $w = \sum_{y_i=1} x_i - \sum_{y_i=-1} x_j$ : weaker model

$\alpha_i$ found by solving an optimization problem: details out of scope
a toy example

- take $\mathcal{X} \subset \mathbb{R}^2$ and $K(x_1, x_2) = \langle x_1, x_2 \rangle$ (linear kernel)

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

- if $\alpha_i$ were absent then $w = \sum_{y_i=1} x_i - \sum_{y_i=-1} x_j$ : weaker model

- $\alpha_i$ found by solving an optimization problem : details out of scope
a toy example

- take \( \mathcal{X} \subset \mathbb{R}^2 \) and \( K(x_1, x_2) = \langle x_1, x_2 \rangle \) (linear kernel)

\[
h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle = \left\langle x, \sum_{i=1}^{n} \alpha_i y_i x_i \right\rangle = \langle x, w \rangle \quad \text{(linear hypothesis)}
\]

- if \( \alpha_i \) were absent then \( w = \sum_{y_i=1} x_i - \sum_{y_i=-1} x_j \) : weaker model

- \( \alpha_i \) found by solving an optimization problem : details out of scope

![figure: linear classifier](image1.png)

![figure: utility of weight variables \( \alpha_i \)](image2.png)
enter mercer kernels

- linear hypothesis are too weak to detect complex patterns in data
linear hypothesis are too weak to detect complex patterns in data
  in practice more complex notions of similarity 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 $\Phi : X \rightarrow H$ to some Hilbert space $H$ such that for all $x_1, x_2 \in X$,
 $K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$

 Mercer kernels give us hypotheses that are linear in the Hilbert space
 $h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \Phi(x), \Phi(x_i) \rangle = \langle \Phi(x), w \rangle$ for some $w \in H$
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
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
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 $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ to some hilbert space $\mathcal{H}$ such that for all $x_1, x_2 \in \mathcal{X}$, $K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$
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 $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ to some hilbert space $\mathcal{H}$ such that for all $x_1, x_2 \in \mathcal{X}$, $K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$

- mercer kernels give us hypotheses that are linear in the hilbert space

$$h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \Phi(x), \Phi(x_i) \rangle = \langle \Phi(x), w \rangle \text{ for some } w \in \mathcal{H}$$
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
  \[
  \Phi(\mathbf{x}) = (p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1) \in \mathbb{R}^6
  \]
a toy example

- consider $\mathcal{X} \subset \mathbb{R}^2$ s.t. $x = (p, q)$ and $K(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^2$

- one can show that the corresponding map is six dimensional

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

- 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

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

- 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. $x = (p, q)$ and $K(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^2$
- one can show that the corresponding map is six dimensional
  \[ \Phi(x) = (p^2, q^2, \sqrt{2}pq, \sqrt{2}p, \sqrt{2}q, 1) \in \mathbb{R}^6 \]
- it is able to implement quadratic hypotheses
  - e.g. $h(x) = p^2 + q^2 - 1$ for $w = (1, 1, 0, 0, 0, -1)$

figure: non linear problem

figure: kernel trick in action
issues in kernel learning

- frequently one requires complex kernels having high dimensional maps

\[ K(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|^2}{\sigma^2}) \]

cannot explicitly compute the map \( \Phi \)

the kernel trick: can compute \( K(x_1, x_2) \) without computing \( \Phi \)

have to use the implicit form

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) : \text{slow} \]

why only mercer kernels?

for algorithmic convenience and a clean theory

can use non-mercer indefinite kernels as well: out of scope
issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel \( K(x_1, x_2) = \exp \left( \frac{\|x_1 - x_2\|^2}{2\sigma^2} \right) \) has an infinite dimensional map
frequently one requires complex kernels having high dimensional maps

- e.g. the gaussian kernel $K(x_1, x_2) = \exp\left(\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$ has an infinite dimensional map
- cannot explicitly compute the map $\Phi$
issues in kernel learning

- Frequently one requires complex kernels having high dimensional maps
  - E.g., the Gaussian kernel $K(x_1, x_2) = \exp\left(\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - Cannot explicitly compute the map $\Phi$
  - The kernel trick: can compute $K(x_1, x_2)$ without computing $\Phi$
issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel \( K(x_1, x_2) = \exp\left(\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right) \) has an infinite dimensional map
  - cannot explicitly compute the map \( \Phi \)
  - the kernel trick: can compute \( K(x_1, x_2) \) without computing \( \Phi \)
  - have to use the implicit form \( h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \): slow
issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(x_1, x_2) = \exp \left( \frac{\|x_1 - x_2\|^2}{2\sigma^2} \right)$ has an infinite dimensional map
  - cannot explicitly compute the map $\Phi$
  - the kernel trick: can compute $K(x_1, x_2)$ without computing $\Phi$
  - have to use the implicit form $h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i)$: slow

- why only mercer kernels?
issues in kernel learning

- frequently one requires complex kernels having high dimensional maps
  - e.g. the gaussian kernel $K(x_1, x_2) = \exp\left(\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$ has an infinite dimensional map
  - cannot explicitly compute the map $\Phi$
  - the kernel trick: can compute $K(x_1, x_2)$ without computing $\Phi$
  - have to use the implicit form $h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i)$: 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 $K(x_1, x_2) = \exp\left(\frac{||x_1 - x_2||^2}{2\sigma^2}\right)$ has an infinite dimensional map
  ▶ cannot explicitly compute the map $\Phi$
  ▶ the kernel trick: can compute $K(x_1, x_2)$ without computing $\Phi$
  ▶ have to use the implicit form $h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i)$: 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

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

requires up to \( n \) (and in practice \( \Omega(n) \)) operations

\[ h(x) = \langle \Phi(x), w \rangle \]

for some \( w \in \mathcal{H} \)

requires a single operation but in a high-dimensional space

\[ Z: X \rightarrow \mathbb{R}^D \]

such that for all \( x_1, x_2 \in X \)

\[ \langle Z(x_1), Z(x_2) \rangle \approx K(x_1, x_2) \]

\[ h(x) = \langle Z(x), w \rangle \]

for some \( w \in \mathbb{R}^D \)

would get power of kernel as well as speed of linear hypothesis
fast kernel learning: the basic idea

▶ two ways of representing mercer kernel hypotheses

▶ \( h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \)

▶ requires up to \( n \) (and in practice \( \Omega(n) \)) operations

▶ \( h(x) = \langle \Phi(x), w \rangle \) for some \( w \in H \)

▶ requires a single operation but in a high dimensional space

▶ can we find an approximate map for the kernel in some low dimensional space?

\( Z: X \rightarrow \mathbb{R}^D \) such that for all \( x_1, x_2 \in X \),

\[ \langle Z(x_1), Z(x_2) \rangle \approx K(x_1, x_2) \]

▶ \( h(x) = \langle Z(x), w \rangle \) for some \( w \in \mathbb{R}^D \)

▶ would get power of kernel as well as speed of linear hypothesis
two ways of representing mercer kernel hypotheses

\[ h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \]

- requires upto \( n \) (and in practice \( \Omega(n) \)) operations
fast kernel learning: the basic idea

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

\[ h(x) = \sum \alpha_i y_i K(x, x_i) \]

\[ h(x) = \langle \Phi(x), w \rangle \] for some \( w \in \mathcal{H} \)
fast kernel learning: the basic idea

- two ways of representing mercer kernel hypotheses
  - \( h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \)
    - requires up to \( n \) (and in practice \( \Omega(n) \)) operations
  - \( h(x) = \langle \Phi(x), w \rangle \) for some \( w \in \mathcal{H} \)
    - requires a single operation but in a high dimensional space
fast kernel learning: the basic idea

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

- can we find an approximate map for the kernel in some low dimensional space?
  - \( Z : \mathcal{X} \rightarrow \mathbb{R}^D \) such that for all \( x_1, x_2 \in \mathcal{X} \), \( \langle Z(x_1), Z(x_2) \rangle \approx K(x_1, x_2) \)
fast kernel learning: the basic idea

- Two ways of representing Mercer kernel hypotheses
  - \( h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) \)
    - Requires up to \( n \) (and in practice \( \Omega(n) \)) operations
  - \( h(x) = \langle \Phi(x), w \rangle \) for some \( w \in H \)
    - Requires a single operation but in a high dimensional space

- Can we find an approximate map for the kernel in some low dimensional space?
  - \( Z: \mathcal{X} \rightarrow \mathbb{R}^D \) such that for all \( x_1, x_2 \in \mathcal{X} \), \( \langle Z(x_1), Z(x_2) \rangle \approx K(x_1, x_2) \)
  - \( h(x) = \langle Z(x), w \rangle \) for some \( w \in \mathbb{R}^D \)
Fast kernel learning: the basic idea

- Two ways of representing Mercer kernel hypotheses
  - $h(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i)$
    - Requires up to $n$ (and in practice $\Omega(n)$) operations
  - $h(x) = \langle \Phi(x), w \rangle$ for some $w \in \mathcal{H}$
    - Requires a single operation but in a high dimensional space

- Can we find an approximate map for the kernel in some low dimensional space?
  - $Z : \mathcal{X} \rightarrow \mathbb{R}^D$ such that for all $x_1, x_2 \in \mathcal{X}$, $\langle Z(x_1), Z(x_2) \rangle \approx K(x_1, x_2)$
  - $h(x) = \langle Z(x), w \rangle$ for some $w \in \mathbb{R}^D$
  - 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?
the underlying math

- why should such approximate maps exist?
  - given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$

- all algorithmic implementations of the jl-lemma require explicit access to $x_i \in \mathcal{H}$
- 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

- why should such approximate maps exist?
  - given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$
    - for all $i, j$, $\langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon$

- need $D = O(\log n \epsilon^2)$ dimensional map

- all algorithmic implementations of the jl-lemma require explicit access to $x_i \in \mathcal{H}$
  - for us, calculating vectors in the hilbert space is prohibitive
  - the number of dimensions depends upon the number of points
  - not satisfactory
why should such approximate maps exist?


- given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$
  - for all $i, j$, $\langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon$
  - need $D = O\left(\frac{\log n}{\epsilon^2}\right)$ dimensional map
why should such approximate maps exist?

  
  given \( n \) points \( x_1, \ldots, x_n \in \mathcal{H} \), there exists a map \( \Psi : \mathcal{H} \rightarrow \mathbb{R}^D \)

  for all \( i, j \), \( \langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon \)

  need \( D = O\left(\frac{\log n}{\epsilon^2}\right) \) dimensional map

problem ??
why should such approximate maps exist?


given \( n \) points \( x_1, \ldots, x_n \in \mathcal{H} \), there exists a map \( \Psi : \mathcal{H} \rightarrow \mathbb{R}^D \)

for all \( i, j \), \( \langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon \)

need \( D = O \left( \frac{\log n}{\epsilon^2} \right) \) dimensional map

problem ??

all algorithmic implementations of the jl-lemma require explicit access to \( x_i \in \mathcal{H} \)
why should such approximate maps exist?


- given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$
  - for all $i, j$, $\langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon$
  - 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 $x_i \in \mathcal{H}$
  - for us, calculating vectors in the hilbert space is prohibitive
the underlying math

- why should such approximate maps exist?
  - given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$
    - for all $i, j$, $\langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon$
    - 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 $x_i \in \mathcal{H}$
    - 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?
  - given $n$ points $x_1, \ldots, x_n \in \mathcal{H}$, there exists a map $\Psi : \mathcal{H} \rightarrow \mathbb{R}^D$
    - for all $i, j$, $\langle \Psi(x_i), \Psi(x_j) \rangle = \langle x_i, x_j \rangle \pm \epsilon$
    - need $D = O \left( \frac{\log n}{\epsilon^2} \right)$ dimensional map

- problem ??
  - all algorithmic implementations of the jl-lemma require explicit access to $x_i \in \mathcal{H}$
    - 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, $K(x_1, x_2) = \int_{\Gamma} \gamma(x_1 - x_2) \, d\mu(\gamma)$
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-Stieltjes transform of some bounded positive measure on the Pontryagin dual group, \( K(x_1, x_2) = \int_{\Gamma} \gamma(x_1 - x_2) \, d\mu(\gamma) \)


Every dot product Mercer kernel arises from an analytic function having a Maclaurin series with non-negative coefficients, \( K(x_1, x_2) = \sum_{i=0}^{\infty} a_n \langle x_1, x_2 \rangle^n \)
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, \(K(x_1, x_2) = \int_\Gamma \gamma(x_1 - x_2) \, d\mu(\gamma)\)

every dot product mercer kernel arises from an analytic function having a maclaurin series with non-negative coefficients, \(K(x_1, x_2) = \sum_{i=0}^{\infty} a_n \langle x_1, x_2 \rangle^n\)

▶ 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 $\mu \geq 0$

$$K(x_1, x_2) = \int_{\Omega} K_\omega(x_1, x_2) d\mu(\omega) = \mathbb{E}_{\omega \sim \mu} [K_\omega(x_1, x_2)]$$

where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a rank-one kernel i.e. for some $\Phi_\omega : \mathcal{X} \rightarrow \mathbb{R}$, for all $x_1, x_2 \in \mathcal{X}$, $K_\omega(x_1, x_2) = \Phi_\omega(x_1) \cdot \Phi_\omega(x_1)$

▶ a random $K_\omega$ 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: 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 $\mu \geq 0$
  \[ K(x_1, x_2) = \int_{\Omega} K_\omega(x_1, x_2) d\mu(\omega) = \mathbb{E}_{\omega \sim \mu} [K_\omega(x_1, x_2)] \]
  where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a rank-one kernel, i.e., for some $\Phi_\omega : \mathcal{X} \to \mathbb{R}$, for all $x_1, x_2 \in \mathcal{X}$, $K_\omega(x_1, x_2) = \Phi_\omega(x_1) \cdot \Phi_\omega(x_1)$.
- A random $K_\omega$ 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 $\mu \geq 0$

$$K(x_1, x_2) = \int_\Omega K_\omega(x_1, x_2) d\mu(\omega) = \mathbb{E}_{\omega \sim \mu} [K_\omega(x_1, x_2)]$$

where for all $\omega \in \Omega$, $K_\omega : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a rank-one kernel i.e. for some $\Phi_\omega : \mathcal{X} \rightarrow \mathbb{R}$, for all $x_1, x_2 \in \mathcal{X}$, $K_\omega(x_1, x_2) = \Phi_\omega(x_1) \cdot \Phi_\omega(x_1)$

- a random $K_\omega$ 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
select $D$ values $\{\omega_1, \omega_2, \ldots, \omega_D\}$ randomly from distribution $\mu$ over $\Omega$
Random features: implementation

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

$$Z(x) = (\Phi_{\omega_1}(x), \Phi_{\omega_2}(x), \ldots, \Phi_{\omega_D}(x)) \in \mathbb{R}^D$$
select $D$ values $\{\omega_1, \omega_2, \ldots, \omega_D\}$ randomly from distribution $\mu$ over $\Omega$

create the map

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

**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 $(1 - \delta)$,

$$\sup_{x_1, x_2 \in \mathcal{X}} |K(x_1, x_2) - \langle Z(x_1), Z(x_2) \rangle| \leq \epsilon$$
random features: properties

- the guarantee is *uniform* unlike the jl-lemma guarantee
random features: properties

- The guarantee is *uniform* unlike the jl-lemma guarantee
- Holds for all (possibly infinite) pairs of points from $X$
random features: properties

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

- hypothesis is of the form $h(x) = \langle Z(x), w \rangle$, for some $w \in \mathbb{R}^D$
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 $h(x) = \langle Z(x), w \rangle$, for some $w \in \mathbb{R}^D$
  - Evaluating a hypothesis takes $O(D)$ time
random features: properties

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

- hypothesis is of the form $h(x) = \langle Z(x), w \rangle$, for some $w \in \mathbb{R}^D$
  - evaluating a hypothesis takes $O(D)$ 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 $h(x) = \langle Z(x), w \rangle$, for some $w \in \mathbb{R}^D$
  - evaluating a hypothesis takes $O(D)$ 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]
random features : in action

- 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
## random features: in action

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

**figure:** speedups for exponential kernel $K(x_1, x_2) = \exp \left( \frac{\langle x_1, x_2 \rangle}{\sigma^2} \right)$
alternative approaches exist that given a set of training points $x_1, \ldots, x_n$, approximate the gram matrix $G = [g_{ij}], g_{ij} = K(x_i, x_j)$
other approaches

- alternative approaches exist that given a set of training points \( x_1, \ldots, x_n \), approximate the gram matrix \( G = [g_{ij}] \), \( g_{ij} = K(x_i, x_j) \)
  - cholesky decomposition: finds a rank \( D \) approximation to \( G \)

- advantages
  - data dependency helps in hard learning instances [yang et al, nips 2010]

- disadvantages
  - slower than random features as the hypothesis takes \( \Omega(D^2) \) time to evaluate in worst case: \( O(D) \) time using random features
  - expensive preprocessing required: increases time taken to learn

purushottam kar (iit kanpur)  accelerated kernel learning  november 27, 2012 26 / 27
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 \( \Omega(D^2) \) time to evaluate in worst case: \( O(D) \) time using random features.
- Expensive preprocessing required: increases time taken to learn.
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 \( \hat{x}_1, \ldots, \hat{x}_D \) as anchor points and creates a \( D \) dimensional map

advantages
other approaches

▶ alternative approaches exist that given a set of training points $x_1, \ldots, x_n$, approximate the gram matrix $G = [g_{ij}]$, $g_{ij} = K(x_i, x_j)$
  ▶ cholesky decomposition : finds a rank $D$ approximation to $G$
  ▶ nyström method : chooses a subsample of training points $\hat{x}_1, \ldots, \hat{x}_D$ as anchor points and creates a $D$ dimensional map

▶ advantages
  ▶ data dependency helps in hard learning instances [yang et al, nips 2010]
alternative approaches exist that given a set of training points $x_1, \ldots, x_n$, approximate the gram matrix $G = [g_{ij}]$, $g_{ij} = K(x_i, x_j)$

- cholesky decomposition: finds a rank $D$ approximation to $G$
- nyström method: chooses a subsample of training points $\hat{x}_1, \ldots, \hat{x}_D$ 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 $x_1, \ldots, x_n$, approximate the gram matrix $G = [g_{ij}]$, $g_{ij} = K(x_i, x_j)$
  - cholesky decomposition: finds a rank $D$ approximation to $G$
  - nyström method: chooses a subsample of training points $\hat{x}_1, \ldots, \hat{x}_D$ 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 $\Omega(D^2)$ 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 $\Omega(D^2)$ time to evaluate in worst case: $\mathcal{O}(D)$ time using random features
  - expensive preprocessing required: increases time taken to learn
what all families admit such random feature constructions?
what all families admit such random feature constructions?
  ▶ there do exist that don't [Balcan et al., Mach. Learn., 65(1): 79–94, 2006]
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
what all families admit such random feature constructions?
  - there do exist that don't [Balcan et al., Mach. Learn., 65(1): 79–94, 2006]

- introduce data awareness in methods
- explore applications in other kernel learning tasks
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]