Lecture 06: Kernel methods

Meelis Kull
meelis.kull@ut.ee
Spring 2018
Previously: Lectures 01-05

• Lecture 01 – Introduction to machine learning
• Lecture 02 – Tasks, models, features
• Lecture 03 – Binary classification and related tasks
• Lecture 04 – Linear regression and regularisation
• Lecture 05 – Linear classification
Lectures 06-09

- Distance-based methods (Chapter 8)
- Kernel methods (mostly not in the book)
- Logical models (Chapters 4-6)
- Neural networks (mostly not in the book)
- Deep learning (mostly not in the book)
Lecture 06 – Distance-based and kernel methods

• Distance-based methods
  – Distance-based learning algorithms
  – Why many distance measures?
  – Well-known distance measures
  – How to design a distance measure?

• Kernel methods
  – Non-linearity with linear models
  – Kernel trick
  – Construction of kernels
  – Well-known kernels
Chapter 8:
Distance-based models
Lecture 06 – Distance-based and kernel methods

• Distance-based methods
  – Distance-based learning algorithms
  – Why many distance measures?
  – Well-known distance measures
  – How to design a distance measure?

• Kernel methods
  – Non-linearity with linear models
  – Kernel trick
  – Construction of kernels
  – Well-known kernels
K-nearest neighbours (KNN)
[from Lecture 02]

• **Pro:** One of the simplest learning methods

• **K** - integer, usually an odd number (1, 3, 5, …)

• **Learning the model:**
  – Memorise training data

• **Applying the model on test data:**
  – For each test instance:
    • Find the K closest training instances
    • Predict most frequent label [in classification] or average label [in regression] among those K

• **Cons:** hard to choose a good distance measure, many distance calculations required
KNN modifications

• The K neighbours in KNN contribute equally to choosing the label to predict

• Weighted KNN:
  – Closer neighbours contribute more
  – Typically weights are inverses of distances
Some well-known distance measures
[from Lecture 02]

• Distance of vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$
  
  – Euclidean distance $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

  – Manhattan distance $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

  – Correlation distance $d(x, y) = 1 - r(x, y)$ $r(x, y)$ is Pearson correlation coefficient

• Distance of sequences ACCTTG and TACCTG
  
  – Hamming distance $\frac{\cancel{ACCTTG}}{\cancel{TACCTG}} =$ => $d = 3$

  – Levenshtein distance $\frac{\cancel{ACCTTG}}{T\cancel{ACCC}\cancel{TTG}} =$ => $d = 2$
Distance-based methods

- Distance-based learning algorithms
  - Why many distance measures?
  - Well-known distance measures
  - How to design a distance measure?

Kernel methods

- Non-linearity with linear models
- Kernel trick
- Construction of kernels
- Well-known kernels
Why many distance measures?

• How are machine learning algorithms using the distance measure:
  – Instances that are closer to each other (have more similar feature values) are more likely to have a similar label

• Distance measure specifies the **inductive bias** (which models are more likely)

• We are helping the machine learning algorithm by giving it a relevant inductive bias = domain knowledge
Example: Distances on chess-board

- Example task with 2 features:
  - row and column
- Suppose domain expert says:
  - "If the chess king can get from one instance to another with 1 step then the true label is usually the same"
- We should then use the distance in figure

\[ d = 1 \]
\[ d = 2 \]
\[ d = 3 \]
Example: Distances on chess-board

- Same task, now about rook:

  \[ d = 1 \]
  \[ d = 2 \]
Example: Distances on chess-board

- Now about ‘krook’ (an imaginary piece with constraints both from king and rook)

\[
d = 1 \\
\]

\[
d = 2 \\
\]

\[
d = 3 \\
\]

\[
d = 4 \\
\]

\[
d = 5 \\
\]

\[
d = 6 \\
\]
Distance – any function with 2 arguments?

• No! Learning algorithms can vary slightly but generally they assume that distance measure is a metric:
  – distances between a point and itself are zero: \( \text{Dis}(x, x) = 0 \)
  – all other distances are larger than zero: if \( x \neq y \) then \( \text{Dis}(x, y) > 0 \)
  – distances are symmetric: \( \text{Dis}(y, x) = \text{Dis}(x, y) \)
  – detours can not shorten the distance (triangle inequality): \( \text{Dis}(x, z) \leq \text{Dis}(x, y) + \text{Dis}(y, z) \)
Choosing a distance measure

• How to choose a distance measure for a particular task?
• Think what similarity means in this domain
• Choose a distance function such that:
  – Intuitively similar points have short distance
  – Intuitively dissimilar points have long distance
• One possible strategy:
  – Normalize all features (mean=0, variance=1) or do other transformations before using some well-known distance measure
Distance-based methods

- Distance-based learning algorithms
- Why many distance measures?
  - Well-known distance measures
  - How to design a distance measure?

Kernel methods

- Non-linearity with linear models
- Kernel trick
- Construction of kernels
- Well-known kernels
Minkowski distance

- Minkowski distance is parametrised by $p$:

$$\text{Dis}_p(x, y) = \left( \sum_{j=1}^{d} |x_j - y_j|^p \right)^{1/p} = \|x - y\|_p$$

where $\|z\|_p = \left( \sum_{j=1}^{d} |z_j|^p \right)^{1/p}$ is the $p$-norm
Special cases of Minkowski distance

- $p = 1$ : Manhattan (city-block) distance
  \[ \text{Dis}_1(x, y) = \sum_{j=1}^{d} |x_j - y_j| \]

- $p = 2$ : Euclidean distance
  \[ \text{Dis}_2(x, y) = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2} = \sqrt{(x - y)^T(x - y)} \]

- $p \to \infty$ : Chebyshev distance (max-distance)
  \[ \text{Dis}_\infty(x, y) = \max_j |x_j - y_j| \]

- $0 < p < 1$ : semimetric distances
  (triangle inequality not satisfied)
Circles in Minkowski distances

• Equidistant points from the origin:

  - Chebyshev
  - Euclidean
  - Manhattan
Euclidean distance is a metric
Manhattan distance is a metric
Minkowski distance is not a metric for $0 < p < 1$. 
0-norm distance

• $L_0$-norm (or 0-norm) distance:

\[
\text{Dis}_0(x, y) = \sum_{j=1}^{d} (x_j - y_j)^0 = \sum_{j=1}^{d} I[x_j = y_j]
\]

where $x^0 = \begin{cases} 
0 & \text{if } x = 0 \\
1 & \text{if } x \neq 0
\end{cases}$

• (this is not strictly a Minkowski distance)
Implications on KNN

• 1-NN example:

  ![Euclidean distance](image1)
  ![Manhattan distance](image2)

• All points on pink circle/square are equidistant
Voronoi tessellation

- Partitioning of the instance space into regions based on which instance is its nearest neighbour
Mahalanobis distance

- Multi-dimensional generalization of the idea of measuring how many standard deviations away the points are
- Mahalanobis distance:

\[ \text{Dis}_M(x, y|\Sigma) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)} \]

where \( \Sigma \) is the covariance matrix of features
- If \( \Sigma \) is the identity matrix: Euclidean distance
Mahalanobis equidistant points for different covariance matrices
Lecture 06 – Distance-based and kernel methods

✓ Distance-based methods
  ✓ Distance-based learning algorithms
  ✓ Why many distance measures?
  ✓ Well-known distance measures

  – How to design a distance measure?

• Kernel methods
  – Non-linearity with linear models
  – Kernel trick
  – Construction of kernels
  – Well-known kernels
Designing a distance measure

• How to design a distance measure for a task?

• Think what similarity means in this domain

• Design a distance function such that:
  – Intuitively similar points have short distance
  – Intuitively dissimilar points have long distance

• One possible strategy:
  – Normalize all features (mean=0, variance=1) or do other transformations before using some well-known distance measure
Constructing new features

- Sometimes, knowledge about which instances are similar can be best incorporated by:
  - Constructing new features
  - Using some well-known distance measure
Kernel methods

The following section is mostly not covered in Peter Flach’s machine learning textbook
Distance-based methods

- Distance-based learning algorithms
- Why many distance measures?
- Well-known distance measures
- How to design a distance measure?

Kernel methods

- Non-linearity with linear models
- Kernel trick
- Construction of kernels
- Well-known kernels
KNN learns non-linear models

- KNN results in non-linear models (decision boundary is not a straight line / hyperplane)
- For example, KNN works very nicely on these data:

Figure from: https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
Linear methods and non-linearity

- However, linear methods can also learn this separation! How?

Figure from: https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
Non-linearity with linear models
[from Lecture 02]

• Linear models are linear in the features
• Let us construct new features that are non-linear in the original features
• Linear models on the new features are non-linear in the original features!
• This way we can fit non-linear models using linear model learning algorithms
  – E.g. if the hidden dependency is $y = x^3 + 3x^2 - x + 7$
  – Then if we introduce features $x$, $x^2$, $x^3$ then linear methods can learn this functional dependency
Non-linearity with linear models

- Let us construct new features to make data linearly separable:

\[ \phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)\]

Figure from: https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
Non-linearity with linear models

• Fitting:
  – Transform training data through \( x \leftarrow \phi(x) \)
  – Learn a linear model

• Predicting:
  – Transform test data through \( x \leftarrow \phi(x) \)
  – Apply the linear model

• This can be very slow, if there are a lot of constructed features. Can we make it faster?

• Let us first see how \( \phi(x) \) is used in SVM
SVM on constructed features

• Fitting (dual form soft-margin):
  - \( x_i \mapsto \phi(x_i) \) for \( i = 1, \ldots, n \)
  - \( \alpha_1^*, \ldots, \alpha_n^* = \arg \max_{\alpha_1, \ldots, \alpha_n} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i) \cdot \phi(x_j) + \sum_{i=1}^{n} \alpha_i \)

    subject to \( 0 \leq \alpha_i \leq C \) and \( \sum_{i=1}^{n} \alpha_i y_i = 0 \)

  - \( w^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(x_i) \)
  - \( t^* = w^* \cdot \phi(x_j) - y_j = \sum_{i=1}^{n} \alpha_i^* y_i \phi(x_i) \cdot \phi(x_j) - y_j \)

• Prediction:
  - \( x \mapsto \phi(x) \)
  - \( \hat{y} = \text{sign}(w^* \cdot \phi(x) - t^*) = \text{sign} \left( \sum_{i=1}^{n} \alpha_i^* y_i \phi(x_i) \cdot \phi(x) - t^* \right) \)

  where \( x_j \) is a support vector
Lecture 06 – Distance-based and kernel methods

✓ Distance-based methods
  ✓ Distance-based learning algorithms
  ✓ Why many distance measures?
  ✓ Well-known distance measures
  ✓ How to design a distance measure?

✓ Kernel methods
  ✓ Non-linearity with linear models
  – Kernel trick
  – Construction of kernels
  – Well-known kernels
Everything through dot products

- In SVM fitting and prediction all uses of instances can be made through dot products

- Our example transformation:
  \[ \phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]
  \[ (x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \]

- Dot product:
  \[
  \phi(x) \cdot \phi(x') = \phi(x_1, x_2) \cdot \phi(x'_1, x'_2) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \cdot (x'_1^2, \sqrt{2} x'_1 x'_2, x'_2^2)
  \]
  \[
  = x_1^2 x'_1^2 + \sqrt{2} x_1 x_2 \sqrt{2} x'_1 x'_2 + x_2^2 x'_2^2
  \]
  \[
  = (x_1 x'_1)^2 + 2(x_1 x'_1)(x_2 x'_2) + (x_2 x'_2)^2 = (x_1 x'_1 + x_2 x'_2)^2
  \]
  \[
  = (\mathbf{x} \cdot \mathbf{x'})^2
  \]
Kernel trick

• We can now use the kernel trick!

• Instead of
  – transforming the instances with \( x \mapsto \phi(x) \)
  – then fitting and predicting on the constructed higher-dimensional instances \( \phi(x) \)

• We can now do the kernel trick:
  – Work with original instances \( x \)
  – Use a modified method of calculating dot products \( x \cdot x' \mapsto \kappa(x, x') \)
  – In our example the kernel \( \kappa \) is \( \kappa(x, x') = (x \cdot x')^2 \)
Kernel SVM

• Fitting (soft-margin kernel SVM):
  \[ \alpha_1^*,\ldots,\alpha_n^* = \arg\max_{\alpha_1,\ldots,\alpha_n} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) + \sum_{i=1}^{n} \alpha_i \]
  subject to \( 0 \leq \alpha_i \leq C \) and \( \sum_{i=1}^{n} \alpha_i y_i = 0 \)

  \[ t^* = \sum_{i=1}^{n} \alpha_i^* y_i \kappa(x_i, x_j) - y_j \]
  where \( x_j \) is a support vector of class \( y_j \)

• Prediction:
  \[ \hat{y} = \text{sign} \left( \sum_{i=1}^{n} \alpha_i^* y_i \kappa(x_i, x) - t^* \right) \]
Kernel trick

• We can now learn the separating classifier with kernel SVM without ever explicitly transforming instances into the new space!

Figure from: https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
When does the kernel trick work?

• Can we choose $\kappa(x, x')$ be any function?
• No! We can only choose kernels for which there exists some transformation $\phi$ such that:
  \[
  \kappa(x, x') = \phi(x) \cdot \phi(x')
  \]
  – Otherwise the mathematics of SVM breaks down
• Mercer’s theorem:
  – Such transformation $\phi$ exists if and only if $\kappa(x, x')$ is a positive semi-definite function
Positive semi-definiteness

- A function \( \kappa(x, x') \) is positive semi-definite, if for any \( x_1, \ldots, x_n \) the following matrix is positive semi-definite:

\[
\begin{pmatrix}
\kappa(x_1, x_1) & \kappa(x_1, x_2) & \ldots & \kappa(x_1, x_n) \\
\kappa(x_2, x_1) & \kappa(x_2, x_2) & \ldots & \kappa(x_2, x_n) \\
\vdots & \vdots & \ddots & \vdots \\
\kappa(x_n, x_1) & \kappa(x_n, x_2) & \ldots & \kappa(x_n, x_n)
\end{pmatrix}
\]

- A matrix \( M \) is positive semi-definite, if for any vector \( z \):

\[
z^T M z \geq 0
\]
When does the kernel trick work?

• According to Mercer’s theorem it works whenever $\kappa(x,x')$ satisfies the following:
  
  – For any points $x_1, \ldots, x_n$ and real values $z_1, \ldots, z_n$:
    $$
    \sum_{i=1}^{n} \sum_{j=1}^{n} z_i z_j \kappa(x_i, x_j) \geq 0
    $$

• Checking this criterion might be non-trivial, often new kernels are constructed by combining existing kernels
Lecture 06 – Distance-based and kernel methods

- Distance-based methods
  - Distance-based learning algorithms
  - Why many distance measures?
  - Well-known distance measures
  - How to design a distance measure?

- Kernel methods
  - Non-linearity with linear models
  - Kernel trick
    - Construction of kernels
  - Well-known kernels
Construction of kernels

- If $\kappa_1, \kappa_2$ are kernels, $a \in \mathbb{R}$ and $f : \mathcal{X} \rightarrow \mathbb{R}$ then the following $\kappa(x,x')$ are kernels as well:

\[
\begin{align*}
\kappa(x,x') &= a \kappa_1(x,x') \\
\kappa(x,x') &= \kappa_1(x,x') \kappa_2(x,x') \\
\kappa(x,x') &= \kappa_1(x,x') + \kappa_2(x,x') \\
\kappa(x,x') &= f(x)f(x')
\end{align*}
\]
Why couldn’t we instead just explicitly construct the features?

Often we can, but sometimes:

- The computational gain of not having to explicitly transform instances is needed.
- It is easier to express domain knowledge in the form of a kernel than in the form of constructed features.
Lecture 06 – Distance-based and kernel methods

✓ Distance-based methods
  ✓ Distance-based learning algorithms
  ✓ Why many distance measures?
  ✓ Well-known distance measures
  ✓ How to design a distance measure?

✓ Kernel methods
  ✓ Non-linearity with linear models
  ✓ Kernel trick
  ✓ Construction of kernels
  – Well-known kernels
Some important kernels

- **Linear kernel** (original space unchanged):
  \[ \kappa(x, x') = x \cdot x' \]

- **Polynomial kernel** (for any \( r \in \mathbb{R} \) and \( d \in \mathbb{N} \)):
  \[ \kappa(x, x') = (x \cdot x' + r)^d \]

- **Gaussian kernel**
  (Radial basis function kernel, RBF kernel), for any \( \sigma > 0 \):
  \[ \kappa(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma}\right) \]
Polynomial kernel

- For any \( r \in \mathbb{R} \) and \( d \in \mathbb{N} \):
  \[
  \kappa(x, x') = (x \cdot x' + r)^d
  \]

- For example, take \( r = 1, d = 2 \) with 2 features:
  \[
  \kappa(x, x') = (x_1' + x_2' + 1)^2
  = x_1'^2 + x_2'^2 + 1 \cdot 1 + \sqrt{2}x_1x_2\sqrt{2}x_1'x_2' + \sqrt{2}x_1\sqrt{2}x_1' + \sqrt{2}x_2\sqrt{2}x_2'
  = (x_1^2, x_2^2, 1, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2) \cdot (x_1'^2, x_2'^2, 1, \sqrt{2}x_1'x_2', \sqrt{2}x_1', \sqrt{2}x_2')
  = \phi(x) \cdot \phi(x')
  \]

- Constructs all polynomials of features \( x_1, x_2 \) with degree up to \( d = 2 \)
Gaussian kernel

• Gaussian kernel
(Radial basis function kernel, RBF kernel):

\[
\kappa(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma}\right)
\]

• The constructed feature space is infinite-dimensional !!! E.g. for \( \sigma = 1 \):

\[
\exp\left(-\frac{1}{2} ||x - x'||^2\right) = \sum_{j=0}^{\infty} \frac{(x \cdot x')^j}{j!} \exp\left(-\frac{1}{2} ||x||^2\right) \exp\left(-\frac{1}{2} ||x'||^2\right)
\]

\[
= \sum_{j=0}^{\infty} \sum_{n_1 = j}^{\infty} \cdots \sum_{n_k = j}^{\infty} \exp\left(-\frac{1}{2} ||x||^2\right) \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} \exp\left(-\frac{1}{2} ||x'||^2\right) \frac{x'_1^{n_1} \cdots x'_k^{n_k}}{\sqrt{n_1! \cdots n_k!}}
\]

one component of \( \phi(x) \)
Using Gaussian kernel

Figure from: http://www.bogotobogo.com/python/scikit-learn/scikit_machine_learning_Linearly_Separable_NonLinearly_RBF_Separable_Data_SVM_GUI.php
Kernel methods

• Many learning algorithms have been kernelised:
  – Kernel perceptron
  – Kernel SVM
  – Kernel ridge regression
  – …
Kernels on structured data

• String kernels
  – P-spectrum kernel
  – All-subsequences kernel
  – Gap-weighted subsequences kernel
  – ...

• Graph & tree kernels
  – Co-rooted subtrees kernel
  – All subtrees kernel
  – Random walks kernel
P-spectrum kernel

• The transformation counts the occurrences of all substrings with length \( p \) in the text

\[
\phi(\text{"ababc"}) \rightarrow \{
\text{"ab"} \rightarrow 2, \text{"ba"} \rightarrow 1, \text{"bc"} \rightarrow 1, \text{other} \rightarrow 0\}
\]
All subsequences kernel

• Counts the occurrences of all subsequences (with gaps)
• Each subsequence has higher weight if it is shorter (has less gaps)
• We will not discuss the exact definition in detail
Distance-based methods
- Distance-based learning algorithms
- Why many distance measures?
- Well-known distance measures
- How to design a distance measure?

Kernel methods
- Non-linearity with linear models
- Kernel trick
- Construction of kernels
- Well-known kernels