Classifier complexity and SVM's

David M.J. Tax
Information and Communication Theory group,
Electrical engineering, Mathematics and Computer Science
Delft University of Technology
D.M.J.Tax@tudelft.nl

Contents
• Complexity and learning curve
• Bias-variance dilemma
• Adapting the complexity
• Regularization (QDC, Fisher)
• Measuring complexity: VC-dimension
• Support vector classifiers
  • Class overlap
  • Kernel trick
• Examples
• Conclusions

Complexity?

The complexity of a classifier indicates the ability to fit to any data distribution

Which complexity to choose?

• A simple classifier fits to only a few specific data distributions
• A complex classifier fits to almost all data distributions
• So, we should always use a complex classifier! (?)

The learning curve

error
true error $\varepsilon$
asymptotic error
apparent error $\varepsilon_A$
number of training objects

apparent error is too optimistic

The learning curve

error
true error $\varepsilon$
asymptotic error
apparent error $\varepsilon_A$
number of training objects
**The learning curve**

- More complex classifier
- Lower training error
- Higher test error
- Lower asymptotic error

**Learning and Feature curve**

- Complex classifiers are good when you have sufficient number of training objects
- When a small number of training objects is available, you overtrain
- Use a simple classifier when you don’t have many training examples

Choose the complexity according to the available training set size

**Peaking phenomenon**

- The minimum error for a given number of samples is obtained for a specific complexity

**Bias-variance dilemma**

- Assume, our function $f$ tries to model the relation between $x$ and $y$ given dataset $\mathcal{X} = \{x_i, y_i\}, i = 1...N$
- The mean-squared error can be decomposed:
  
  $\varepsilon = E_X \left[ (f_X(x) - y(x))^2 \right]$
  
  $= (E_X [f_X(x) - y(x)])^2 + E_X \left[ (f_X(x) - E_X[y(x)])^2 \right]$ 

(squared) bias variance
**Bias-variance dilemma**

- error = squared bias + variance
- This tradeoff is very general
- More flexible models have lower bias, but higher variance
- Simple models have high bias, but low variance

**Bias-variance in real life**

- Generate 10 times a dataset, plot classifier
- Still 40 objects are generated (a lot!)

**Adapting the complexity**

- Start with a simple classifier and increase the complexity till it fits
- In practice it is not simple to increase complexity
- Start with a complex classifier, add a regularizer term to diminish the chance of an over-complex solution

\[
\text{minimize} \quad \varepsilon_A + \lambda \Omega(w)
\]

**Regularization**

- For normal based classifiers, avoid that the covariance matrix adapts to outliers or a few examples
  \[\Sigma_{\lambda} = \Sigma + \lambda I\]
- For general classification/regression problems, avoid that weights become too large (also called ridge regression):
  \[\varepsilon_{\lambda} = \frac{1}{N} \sum_i (y_i - w^T x_i)^2 + \lambda \|w\|^2\]

**Quadratic classifier**

\[
R(x) = \begin{cases} 
(x - \mu_B)^T G_B^{-1} (x - \mu_B) \\
- (x - \mu_A)^T G_A^{-1} (x - \mu_A)
\end{cases}
\]

- When insufficient data is available (nr of obj's is smaller than dimensionality), the inverse covariance matrices are not defined: the classifier is not defined
- Regularization solves it:
  \[G_B = G_B + \lambda I\]
**Quadratic classifier**

- QDC crashes
- Regularized QDC

**Fisher classifier**

\[ R(x) = (\mu_B - \mu_A)^T \tilde{G}^{-1} x + C \]

- The same phenomenon appears, but because the covariance matrix is averaged over two classes, the peak appears for small sample size.
- Instead of regularization the pseudo-inverse of the covariance matrix is used.

**Learning & feature curve**

```
Concordia dataset, 4000 obj in 1024D
```

**Measuring complexity**

- By changing the regularization parameter, the complexity is changed.
- Is there a direct way to measure complexity?
- Yes: the VC-dimension (Vapnik-Chervonenkis dimension) of a classifier.

**VC-dimension**

- \( h \): the VC-dimension of a classifier:
  
  The largest number of vectors that can be separated in all the \( 2^h \) possible ways.
VC-dim for linear classifier

- For linear classifier: \( h = p + 1 \)

Use of VC-dimension

- Unfortunately, only for a very few classifiers the VC-dimension is known
- Fortunately, when you know \( h \) of a classifier, you can bound the true error of the classifier

Bounding the true error

With probability at least \( 1 - \eta \) the inequality holds:

\[
\varepsilon \leq \varepsilon_A + \frac{\mathcal{E}(N)}{2} \left( 1 + \sqrt{1 + \frac{2 \varepsilon_A}{\mathcal{E}(N)}} \right)
\]

where

\[
\mathcal{E}(N) = 4h(\ln(2N/h) + 1) - \ln(\eta/4)
\]

V. Vapnik, Statistical learning theory, 1998

- When \( h \) is small, the true error is close to the apparent error

Is this bound practical?

- The given bound (and others) is very loose
- The worst case scenario is assumed: objects can be randomly labeled
- In practice, features are chosen such that objects from one class are nicely clustered and can be separated from other classes

Compactness hypothesis

Representations of real world objects are close. There is no ground for any generalization on representations that do not obey this demand.

A.G. Arkadev and E.M. Braverman, Computers and Pattern Recognition, 1966

Changing \( h \) for linear cl.

- The linear classifier had \( h = p + 1 \)
- By putting some constraints on this linear classifier, the VC dimension can be reduced
- Assume a linearly separable dataset, constrain the weights such that the output of the classifier is always larger than one:

\[
w^T x_i + b \geq +1, \quad \text{for } y_i = +1
\]

\[
w^T x_i + b \leq -1, \quad \text{for } y_i = -1
\]
Constraining the weights

The constraints force the weights to have a minimum value: 'canonical hyperplane'

\[ w^T x + b \leq -1 \]

\[ w^T x + b \geq +1 \]

\[ w^T x + b = 0 \]

Finding a good classifier

• To find a classifier with small VC-dimension:
  1. minimize the dimensionality \( p \)
  2. minimize the radius \( R \)
  3. maximize the margin \( \rho \)

• Using simple algebra

\[ \min \| w \|^2 = \frac{1}{\rho^2} \]

• So:

\[ w^T x_i + b \geq +1, \text{ for } y_i = +1 \]

\[ w^T x_i + b \leq -1, \text{ for } y_i = -1 \]

Optimizing the classifier

It appears you can rewrite the constraints inside the optimization using Lagrange Multipliers:

\[ \min_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j \]

\[ \sum_{i=1}^{N} \alpha_i y_i = 0 \]

\[ w = \sum_{i=1}^{N} \alpha_i y_i x_i \]

Support vectors

• The classifier becomes:

\[ w = \sum_{i=1}^{N} \alpha_i y_i x_i \]

• The solution is phrased in terms of objects, not features.

• Only a few weights become non-zero

• The objects with non-zero weight are called the support vectors
Support vector classifier

\[ w^T x + b = +1 \]

\[ w^T x + b = -1 \]

\[ w^T x + b = 0 \]

Error estimation

By leave-one-out you can obtain a bound on the error:

\[ E[R(\alpha)] \leq \frac{\text{#support vectors}}{N} \]

High dimensional feature spaces

The classifier is determined by objects, not features: the classifier tends to perform very well in high dimensional feature spaces.

Limitations of the SVM

1. The data should be separable
2. The decision boundary is linear

Weakening the constraints

Introduce a 'slack' variable \( \xi \) for each object to weaken the constraints

SVM with slacks

- The optimization changes into:
  \[
  \min ||w||^2 + C \sum_{i=1}^{N} \xi_i \\
  w^T x_i + b \geq +1 - \xi_i, \text{ for } y_i = +1 \\
  w^T x_i + b \leq -1 + \xi_i, \text{ for } y_i = -1 \\
  \xi_i \geq 0 \forall i
  \]
Tradeoff parameter $C$

$$\min ||w||^2 + C \sum_{i=1}^{N} \xi_i$$

- Notice that the tradeoff parameter $C$ has to be defined beforehand.
- It weighs the contributions between the training error and the structural error.
- Its values are often optimized using cross-validation.

Influence of $C$

Erroneous objects have still large influence when $C$ is not optimized carefully.

Trick: transform your data

- Magic: all operations are on inner products between objects.
- Assume I have a magic transformation of my data that makes it linearly separable...

Example transformation

- Original data: $x = (x_1, x_2)$
- Mapped data: $\Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

Polynomial kernel

- When we have two vectors
  $$\Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$
  $$\Phi(y) = (y_1^2, y_2^2, \sqrt{2}y_1y_2)$$
  $$\Phi(x)^T \Phi(y) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
  $$= (x_1, x_2)(y_1, y_2)^2$$
  $$= (x^T y)^2$$
  it becomes very cheap to compute the inner product.

Transform your data

$$\min \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} y_i y_j \alpha_i \alpha_j \Phi(x_i)^T \Phi(x_j)$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \alpha_i \geq 0 \quad \forall i$$

$$f(z) = \sum_{i=1}^{N} \alpha_i y_i \Phi(x_i)^T \Phi(z) + b$$

- If we have to introduce the magic $\Phi(x)$
  we can as well introduce the magic kernel function: $K(x, y) = \Phi(x)^T \Phi(y)$

Polynomial kernel
The ‘kernel-trick’

- The idea to replace all inner products by a single function, the kernel function \( K \), is called the kernel trick
- It implicitly maps the data to a (most often) high dimensional feature space
- The practical computational complexity does not change (except for computing the kernel function)

Kernel functions

- Polynomial kernel
  \[ K(x, y) = (x^T y + 1)^d \]
- RBF kernel
  \[ K(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right) \]

Results

USPS handwritten digits dataset

Cortes & Vapnik, Machine Learning 1995

Advanced kernel functions

- People construct special kernel functions for applications:
  - Tangent distance kernels to incorporate rotation/scaling insensitivity (handwritten digits recognition)
  - String matching kernels to classify DNA/protein sequences
  - Fisher kernels to incorporate knowledge on class densities
  - Hausdorff kernels to compare image blobs
  - ...

High dimensional feature spaces

- SVM appears to work well in high dimensional feature spaces
- The class overlap is often not large
- Minimizing \( h \) minimizes the risk of overfitting
- The classifier is determined by the support objects and not directly by the features
- No density is estimated

Advantages of SVM

- The SVM generalizes remarkably well, in particular in high-dimensional feature spaces with (relatively) low sample sizes
- Given a kernel and a \( C \), there is one unique solution
- The kernel trick allows for a varying complexity of the classifier
- The kernel trick allows for especially engineered representations for problems
More advantages (?)

- No strict data model is assumed (when you can assume it, use it!)
- The foundation of the SVM is pretty solid (when no slack variables are used)
- An error estimate is available using just the training data (but it is a pretty loose estimate, and crossvalidation is still required to optimize $K$ and $C$)

Disadvantages of SVM

- The quadratic optimization can be computationally expensive (although more and more specialized optimizers appear)
- The kernel and $C$ have to be optimized
- The SVM tends to have problems with highly overlapping classes

Conclusions

- Always tune the complexity of your classifier to your data (#training objects, dimensionality, class overlap, class shape...)
- ‘Avoid solving a more general intermediate problem, when you have limited data’ can be implemented by the SVM
- The SVM can be used in almost all applications (by adjusting $K$ and $C$ appropriately)