Lecture 24: Support vector machines

Reading: Chapter 9

STATS 202: Data mining and analysis

Jonathan Taylor
November 30, 2018
Slide credits: Sergio Bacallado
Support vector machines

- A support vector machine is a support vector classifier applied on an expanded set of predictors, e.g.

\[ \Phi : (X_1, X_2) \rightarrow (X_1, X_2, X_1X_2, X_1^2, X_2^2). \]

- We expand the vector of predictors for each sample \( x_i \) and then perform the algorithm.

- We only need to know the dot products:

\[ \langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(x_i, x_k) \]

for every pair of samples \( (x_i, x_k) \).
The kernel trick

- Often, the dot product:

\[ \langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(x_i, x_k) \]

is a simple function \( f(x_i, x_k) \) of the original vectors. Even if the mapping \( \Phi \) significantly expands the space of predictors.

- Example 1: Polynomial kernel

\[ K(x_i, x_k) = (1 + \langle x_i, x_k \rangle)^2. \]

- With two predictors, this corresponds to the mapping:

\[ \Phi : (X_1, X_2) \rightarrow (\sqrt{2}X_1, \sqrt{2}X_2, \sqrt{2}X_1X_2, X_1^2, X_2^2). \]
The kernel trick

- Often, the dot product:

\[ \langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(x_i, x_k) \]

is a simple function \( f(x_i, x_k) \) of the original vectors. Even if the mapping \( \Phi \) significantly expands the space of predictors.

- **Example 2:** RBF kernel

\[ K(x_i, x_k) = \exp(-\gamma d(x_i, x_k)^2), \]

where \( d \) is the Euclidean distance between \( x_i \) and \( x_k \).
The kernel trick

- Often, the dot product:

\[ \langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(x_i, x_k) \]

is a simple function \( f(x_i, x_k) \) of the original vectors. Even if the mapping \( \Phi \) significantly expands the space of predictors.

- **Example 2**: RBF kernel

\[ K(x_i, x_k) = \exp(-\gamma d(x_i, x_k)^2), \]

where \( d \) is the Euclidean distance between \( x_i \) and \( x_k \).

- In this case, the mapping \( \Phi \) is an expansion into an infinite number of transformations!
The kernel trick

▶ Often, the dot product:

\[ \langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(x_i, x_k) \]

is a simple function \( f(x_i, x_k) \) of the original vectors. Even if the mapping \( \Phi \) significantly expands the space of predictors.

▶ Example 2: RBF kernel

\[ K(x_i, x_k) = \exp(-\gamma d(x_i, x_k)^2), \]

where \( d \) is the Euclidean distance between \( x_i \) and \( x_k \).

▶ In this case, the mapping \( \Phi \) is an expansion into an infinite number of transformations! We can apply the method even if we don’t know what these transformations are.
The kernel trick

- **Fact:** if the matrix $K$ is positive semi-definite, then there exists some mapping $\Phi$ to some feature space, such that $K(x_i, x_k) = \langle \Phi(x_i), \Phi(x_k) \rangle$ for every $\{x_1, \ldots, x_n\}$ in feature space.

- There are lots of known kernels out there.

- Q: If we don’t know which transformations we are using, why would we expect the SVM to work?

  - The kernel $K(x_i, x_k)$ measures the similarity between samples $x_i$ and $x_k$.

  - We can evaluate whether $K$ is a good measure of similarity without understanding the feature expansion $\Phi$. 
Kernels for non-standard data types

- We can define families of kernels (with tuning parameters), which capture similarity between non-standard kinds of data:
  1. Text, strings
  2. Images
  3. Graphs
  4. Histograms

- Sometimes we know the mapping $\Phi$, but there are algorithms that are fast for computing $K(x_i, x_k)$ without doing the expansion explicitly.

- Other times, the expansion $\Phi$ is infinite-dimensional or simply not known.
We can define families of kernels (with tuning parameters), which capture similarity between non-standard kinds of data:

1. Text, strings
2. Images
3. Graphs
4. Histograms

Sometimes we know the mapping $\Phi$, but there are algorithms that are fast for computing $K(x_i, x_k)$ without doing the expansion explicitly.
Kernels for non-standard data types

- We can define families of kernels (with tuning parameters), which capture similarity between non-standard kinds of data:
  1. Text, strings
  2. Images
  3. Graphs
  4. Histograms

- Sometimes we know the mapping $\Phi$, but there are algorithms that are fast for computing $K(x_i, x_k)$ without doing the expansion explicitly.

- Other times, the expansion $\Phi$ is infinite-dimensional or simply not known.
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[ x_1 = ACCTATGCCATA \]
\[ x_2 = AGCTAAGCATAAC \]
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[
\begin{align*}
x_1 &= ACCTATGCCATA \\
x_2 &= AGCTAAGCATA
\end{align*}
\]

▶ **Stringdot kernel:** For each word \( u \) of length \( p \), we define a feature:

\[
\Phi_u(x_i) = \# \text{ of times that } u \text{ appears in } x_i
\]
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[ x_1 = \text{ACCTATGCCATA} \]
\[ x_2 = \text{AGCTAAGCATAC} \]

- **Stringdot kernel:** For each word \( u \) of length \( p \), we define a feature:

\[ \Phi_u(x_i) = \# \text{ of times that } u \text{ appears in } x_i \]

- Naive algorithm would require looping over each sequence, for every subsequence \( u \) of length \( p \).
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[ x_1 = ACCTATGCCATA \]
\[ x_2 = AGCTAAGCATAC \]

▶ **Stringdot kernel**: For each word \( u \) of length \( p \), we define a feature:

\[ \Phi_u(x_i) = \# \text{ of times that } u \text{ appears in } x_i \]

▶ Naive algorithm would require looping over each sequence, for every subsequence \( u \) of length \( p \). This would be \( O(n^2) \) steps, where \( n \) is the length of the sequences.
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[ x_1 = ACCTATGCCATA \]
\[ x_2 = AGCTAAGCATAAC \]

- **Gap weight kernel:** For each word \( u \) of length \( p \), we define a feature:

\[
\Phi_u(x_i) = \sum_{\text{a subsequence of } x_i \text{ containing } u} \lambda^{\text{length}(v)}
\]

with \( 0 < \lambda \leq 1 \).
Example. Kernels for strings

Suppose we want to compare two strings in a finite alphabet:

\[ x_1 = ACCTATGCCATA \]
\[ x_2 = AGCTAAGCATAAC \]

► Gap weight kernel: For each word \( u \) of length \( p \), we define a feature:

\[ \Phi_u(x_i) = \sum_{\text{a subsequence of } x_i \text{ containing } u} \lambda^{\text{length}(v)} \]

with \( 0 < \lambda \leq 1 \).

► The number of features can be huge! However, this can be computed in \( O(Mp \log n) \) steps where \( M \) is the number of matches.
Applying SVMs with more than 2 classes

▶ SVMs don’t generalize nicely to the case of more than 2 classes.
Applying SVMs with more than 2 classes

- SVMs don’t generalize nicely to the case of more than 2 classes.
- Two main approaches:
  
  1. One vs. one: Construct \( \binom{n}{2} \) SVMs comparing every pair of classes. Apply all SVMs to a test observation and classify to the class that wins the most one-on-one challenges.
  
  2. One vs. all: For each class \( k \), construct an SVM \( \beta(k) \) coding class \( k \) as 1 and all other classes as -1. Assign a test observation to the class \( k^* \), such that the distance from \( x_i \) to the hyperplane defined by \( \beta(k^*) \) is largest (the distance is negative if the sample is misclassified).
Applying SVMs with more than 2 classes

- SVMs don’t generalize nicely to the case of more than 2 classes.
- Two main approaches:
  1. **One vs. one:** Construct \( \binom{n}{2} \) SVMs comparing every pair of classes. Apply all SVMs to a test observation and classify to the class that wins the most one-on-one challenges.
Applying SVMs with more than 2 classes

- SVMs don’t generalize nicely to the case of more than 2 classes.

- Two main approaches:
  1. **One vs. one**: Construct \( \binom{n}{2} \) SVMs comparing every pair of classes. Apply all SVMs to a test observation and classify to the class that wins the most one-on-one challenges.
  2. **One vs. all**: For each class \( k \), construct an SVM \( \beta^{(k)} \) coding class \( k \) as 1 and all other classes as \(-1\). Assign a test observation to the class \( k^* \), such that the distance from \( x_i \) to the hyperplane defined by \( \beta^{(k^*)} \) is largest (the distance is negative if the sample is misclassified).
Recall the Lagrange form of the problem.

\[
\min_{\beta_0, w, \epsilon} \quad \frac{1}{2} \|w\|^2 + D \sum_{i=1}^{n} \epsilon_i
\]

subject to

\[y_i(\beta_0 + w \cdot x_i) \geq (1 - \epsilon) \quad \text{for all } i = 1, \ldots, n,\]

\[\epsilon_i \geq 0 \quad \text{for all } i = 1, \ldots, n.\]
Relationship of SVM to logistic regression

- Set $D = 1/\lambda$ and minimize over $\epsilon_i$ explicitly.
Relationship of SVM to logistic regression

- Set $D = 1/\lambda$ and minimize over $\epsilon_i$ explicitly.
- If $1 - y_i(\beta + 0 + w \cdot x_i) \leq 0$ we can take $\hat{\epsilon}_i = 0$. Otherwise, we take

$$
\hat{\epsilon}_i = 1 - y_i(\beta_0 + w \cdot x_i).
$$
Set $D = 1/\lambda$ and minimize over $\epsilon_i$ explicitly.

If $1 - y_i(\beta + 0 + w \cdot x_i) \leq 0$ we can take $\hat{\epsilon}_i = 0$. Otherwise, we take

$$\hat{\epsilon}_i = 1 - y_i(\beta_0 + w \cdot x_i).$$

Or,

$$\hat{\epsilon} = \max(1 - y_i(\beta_0 + w \cdot x_i), 0).$$
Relationship of SVM to logistic regression

- Plugging this into the objective (and replacing $w$ with $\beta$) yields

$$
\min_{\beta} \sum_{i=1}^{n} \max(1 - y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}, 0) + \frac{\lambda}{2} \sum_{j=1}^{p} \|\beta_j\|^2 +
$$
Plugging this into the objective (and replacing $w$ with $\beta$) yields

$$
\min_{\beta} \sum_{i=1}^{n} \max(1 - y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}, 0) + \frac{\lambda}{2} \sum_{j=1}^{p} \|\beta_j\|^2 +
$$

This has a loss that is a function of $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$ and a ridge penalty.
Relationship of SVM to logistic regression

- Plugging this into the objective (and replacing $w$ with $\beta$) yields

$$\min_{\beta} \sum_{i=1}^{n} \max(1 - y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}, 0) + \frac{\lambda}{2} \sum_{j=1}^{p} \|\beta_j\|^2 +$$

- This has a loss that is a function of $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$ and a ridge penalty.

- Loss for logistic regression is also a function of $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$. 

$\lambda \Rightarrow D \Rightarrow C$. 

12 / 1
Relationship of SVM to logistic regression

- Plugging this into the objective (and replacing $w$ with $\beta$) yields

$$\min_\beta \sum_{i=1}^{n} \max(1 - y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}, 0) + \frac{\lambda}{2} \sum_{j=1}^{p} ||\beta_j||^2 +$$

- This has a loss that is a function of $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$ and a ridge penalty.

- Loss for logistic regression is also a function of $y_i(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$.

- Large $\lambda \iff$ small $D \iff$ large $C$. 
Comparing the losses

Flatness of SVM related to insensitivity to outliers...
The kernel trick can be applied beyond SVMs

Kernels and dot products:
The kernel trick can be applied beyond SVMs

Kernels and dot products:

- Associated to $K$ is a dot product. For $x$ in the feature space $\mathbb{R}^p$, define $K_x : \mathbb{R}^p \rightarrow \mathbb{R}$ by

$$K_x(x_0) = K(x, x_0)$$
The kernel trick can be applied beyond SVMs

Kernels and dot products:

- Associated to $K$ is a dot product. For $x$ in the feature space $\mathbb{R}^p$, define $K_x : \mathbb{R}^p \to \mathbb{R}$ by

$$K_x(x_0) = K(x, x_0)$$

- The kernel defines a dot product on linear combinations of the $K_x$’s for different $x$’s:

$$\langle \sum_j c_j K_{x_j}, \sum_i d_i K_{y_i} \rangle_K = \sum_{i,j} c_j d_i K(x_j, y_i)$$

and hence a length

$$\| \sum_j c_j K_{x_j} \|_K^2 = \sum_{i,j} c_i c_j K(x_i, x_j)$$
The kernel trick can be applied beyond SVMs

Kernel regression:

\[
\hat{f}_\lambda = \arg\min_{f} \sum_{i=1}^{n} (Y_i - f(X_i))^2 + \lambda \|f\|_2^2
\]

Remarkably, it is known that

\[
\hat{f} = \sum_{i=1}^{n} \hat{\alpha}_i K(X_i).
\]
The kernel trick can be applied beyond SVMs

Kernel regression:

- For tuning parameter $\lambda$ define

$$\hat{f}_\lambda = \arg\min_f \sum_{i=1}^{n} (Y_i - f(X_i))^2 + \lambda \| f \|_K^2$$
The kernel trick can be applied beyond SVMs

Kernel regression:

- For tuning parameter $\lambda$ define
  \[
  \hat{f}_\lambda = \arg\min_f \sum_{i=1}^{n} (Y_i - f(X_i))^2 + \lambda \|f\|_K^2
  \]

- Remarkably, it is known that
  \[
  \hat{f} = \sum_{i=1}^{n} \hat{\alpha}_i K_{X_i}.
  \]
The kernel trick can be applied beyond SVMs

Kernel regression:
The kernel trick can be applied beyond SVMs

Kernel regression:
- Problem reduces to finding

\[
\hat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{n} \alpha_j K(X_i, X_j))^2 + \lambda \sum_{l,r=1}^{n} \alpha_l \alpha_r K(X_i, X_j)
\]

- Finding \(\hat{\alpha}\) is just like ridge regression!
- Just like smoothing splines, we solved a problem over a big space of functions! Smoothing splines are a special case of the kernel trick...
The kernel trick can be applied beyond SVMs

**Kernel regression:**

- Problem reduces to finding

\[ \hat{\alpha} = \text{argmin}_{\alpha} \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{n} \alpha_j K(X_i, X_j))^2 + \lambda \sum_{l,r=1}^{n} \alpha_l \alpha_r K(X_i, X_j) \]

- Finding \( \hat{\alpha} \) is just like ridge regression!
The kernel trick can be applied beyond SVMs

**Kernel regression:**

- Problem reduces to finding

\[
\hat{\alpha} = \arg\min_\alpha \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{n} \alpha_j K(X_i, X_j))^2 + \lambda \sum_{l,r=1}^{n} \alpha_l \alpha_r K(X_i, X_j)
\]

- Finding \( \hat{\alpha} \) is just like ridge regression!

- Just like smoothing splines, we solved a problem over an big space of functions! Smoothing splines are a special case of the kernel trick...
The kernel trick can be applied beyond SVMs

Kernel PCA:
The kernel trick can be applied beyond SVMs

Kernel PCA:

- Suppose we want to do PCA with an expanded set of predictors, defined by the mapping \( \Phi \).
The kernel trick can be applied beyond SVMs

Kernel PCA:

- Suppose we want to do PCA with an expanded set of predictors, defined by the mapping $\Phi$.
- First principal component is

$$\hat{f}_1 = \arg\max_{f: \|f\|_K \leq 1} \hat{\text{Var}}(f(X)).$$
The kernel trick can be applied beyond SVMs

Kernel PCA:

- Suppose we want to do PCA with an expanded set of predictors, defined by the mapping $\Phi$.
- First principal component is

$$\hat{f}_1 = \arg\max_{f: \|f\|_K \leq 1} \hat{\text{Var}}(f(\mathbf{X})).$$

- Even if $\Phi$ expands the predictors to a very high dimensional space, we can do PCA!
The kernel trick can be applied beyond SVMs

Kernel PCA:

- Suppose we want to do PCA with an expanded set of predictors, defined by the mapping $\Phi$.
- First principal component is

$$
\hat{f}_1 = \arg\max_{f: \|f\|_K \leq 1} \text{Var}(f(X)).
$$

- Even if $\Phi$ expands the predictors to a very high dimensional space, we can do PCA!
- The cost only depends on the number of observations $n$. 
Chapter summary

- Starting with idea of maximum margin classifier, we arrive at the support vector classifier.
- Introduction of kernel yields convenient nonlinear decision boundaries.
- Support vector classifier loss is not unrelated to logistic regression, piecewise linear loss instead of smooth loss.
- Kernel trick can also be used for logistic regression (even PCA).