CS/CNS/EE 253: Advanced Topics in Machine Learning
Topic: Active Learning Review and Kernel Methods **Lecturer:** Andreas Krause
Scribe: Jonathan Krause **Date:** Feb. 17, 2010

12.1 Active Learning: A Review

When learning, it may be the case that getting the true labels of data points is expensive, and so we employ active learning in order to reduce the number of label queries we have to preform. This came with its own set of challenges:

- *Active Learning Bias*: Unless we are careful, we might actually do worse than passive learning. We saw this in the case of uncertainty sampling, in which case there are distributions of points that result in requiring orders of magnitude more label queries than necessary. To fix this issue, pool-based active learning can be used, in which case we pick our label queries in such a way that the labels on unqueried points are implied by the labels we have. One drawback of pool-based active learning is that it depends on the hypothesis space having nice structure.
- *Determining which labels to query*: Here we introduced the concept of the *version space*, the set of all hypotheses consistent with the labels given so far. As our primary goal is to determine a good hypothesis while minimizing the number of label queries performed, we can instead opt to reduce the version space as quickly as possible, where the concept of "reducing" the version space depends on the concept of the "size" of the version space.

  How, then, does one go about shrinking the version space as quickly as possible? If possible, a (generalized) binary search is optimal, as this reduces the size of the version space by half with each query. However, this might not be possible, and depends on the structure of the hypothesis space. An alternative method is the greedy algorithm, in which case at each step we query the point that will eliminate the largest number of candidate hypotheses. Although the greedy approach is not, in general, optimal, it is competitive with the optimal querying scheme.
- *Problems for which shrinking the version space is effective*: We have previously discussed the concept of the *splitting index*, which requires certain structure in the hypothesis space, but guarantees that active learning can help. For example, homogeneous linear separators have a constant splitting index, and thus active learning will help. The splitting index is somewhat analogous to the VC dimension, but here we are looking at label complexity as opposed to hypothesis complexity.

Several interesting topics which we have not discussed are:

- How does active learning change when there is noise in the data set? This introduces the concept of *agnostic active learning*.

- Beyond pool-based active learning: active learning can always help, but pool-based active learning is not always the solution. For example, *activized learning* reduces active learning to passive learning.

12.2 Kernel Methods

In many cases, we do not want to limit our hypothesis space just to ensure having a lower VC dimension, and thus better generalization. For example, it would be nice if one could somehow work with hypothesis classes with an infinite VC dimension. To do so, we introduce *kernel methods.*

12.2.1 Support Vector Machines

In support vector machines we are presented with the following problem:

$$\min \mathbf{w}^T\mathbf{w}$$
$$y_i\mathbf{w}^T\mathbf{x_i} \geq 1 \quad \forall i$$

This is known as the *primal problem* for support vector machines, and it is a convex optimization problem with constraints. Now we shall transform it into an unconstrained optimization problem, its *dual problem.* Noting that minimizing $\mathbf{w}^T\mathbf{w}$ is equivalent to minimizing $\frac{1}{2}\mathbf{w}^T\mathbf{w}$, we can introduce Lagrange multipliers α_i. Our new objective function is:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_i \alpha_i(y_i\mathbf{w}^T\mathbf{x}_i - 1)$$

where the new objective is

$$\min_{\mathbf{w}} \max_{\boldsymbol{\alpha}} L(\mathbf{w}, \boldsymbol{\alpha})$$

Theorem 12.2.1 (KKT) *Suppose we have the optimization problem* $(\star)$

$$\min f(\mathbf{x})$$
$$c_i(\mathbf{x}) \leq 0 \quad \forall i$$

Where f, c_i *are convex and differentiable. Define*

$$L(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_i \alpha_i c_i(\mathbf{x})$$

Then $\bar{\mathbf{x}}$ *is an optimal solution to* $(\star)$ *iff* $\exists \bar{\boldsymbol{\alpha}} \geq 0$ *(in all components) such that*

1. $\frac{\partial}{\partial \mathbf{x}} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\alpha}}) = \frac{\partial}{\partial \mathbf{x}} f(\bar{\mathbf{x}}) + \sum_i \bar{\alpha}_i \frac{\partial}{\partial \mathbf{x}} c_i(\mathbf{x}) = 0$
2. $\frac{\partial}{\partial \alpha_i} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\alpha}}) = c_i(\bar{\mathbf{x}}) \leq 0$
3. $\sum_i \bar{\alpha}_i c_i(\bar{\mathbf{x}}) = 0$ *(complementary slackness)*

These are known as the KKT *(Karush-Kuhn-Tucker) conditions for differentiable convex programs.*

Now apply the KKT theorem to the SVM optimization problem to get the dual problem:

1. $\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, \boldsymbol{\alpha}) = 0 \rightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
2. $y_i \mathbf{w}^T \mathbf{x}_i - 1 \geq 0$
3. $\sum_i \alpha_i (y_i \mathbf{w}^T \mathbf{x}_i - 1) = 0$

From these conditions, we can see that

1. $\mathbf{w}$ can be represented as a linear combination of data points.
2. All data points are at least a normalized distance of 1 from the separating hyperplane.
3. As $\alpha_i \geq 0$, either $\alpha_i = 0$ or $y_i \mathbf{w}^T \mathbf{x}_i = 1$ for all i.

In other words, the points for which $\alpha_i > 0$ are "supporting" the hyperplane, and are thus known as support vectors. So the set of support vectors can be written as

$$S = \{\mathbf{x}_i : y_i \mathbf{w}^T \mathbf{x}_i = 1\}$$

Now substitute $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ into the Lagrangian to get a simplified objective function:

$$\begin{aligned} L(\boldsymbol{\alpha}) &= \frac{1}{2}(\sum_i \alpha_i y_i \mathbf{x}_i)(\sum_j \alpha_j y_j \mathbf{x}_j) - \sum_i \alpha_i (y_i (\sum_j \alpha_j y_j \mathbf{x}_j)^T \mathbf{x}_i - 1) \\ &= \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j - \sum_{i,j} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j + \sum_i \alpha_i \\ &= \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j {\mathbf{x}_i}^T \mathbf{x}_j \end{aligned}$$

Now we can solve for $\mathbf{w}$ if we have $\boldsymbol{\alpha}$, and only need to solve for $\boldsymbol{\alpha}^* = \text{argmax}_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha})$ such that $\alpha_i \geq 0 \quad \forall i$. More importantly, the objective function now only depends on inner products ${\mathbf{x}_i}^T \mathbf{x}_j$, which is extremely useful for nonlinear classification.

12.2.2 The Kernel Trick

For an example of why nonlinear transformations can be necessary, consider the following scenario:

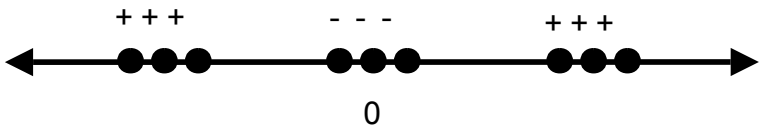

Figure 12.2.1: The original data without a nonlinear transformation.

In order for this data to be linearly separable, a nonlinear transformation to a higher-dimensional space is needed. We use $\varphi(x) = [x, x^2]$ in figure 12.2.2:

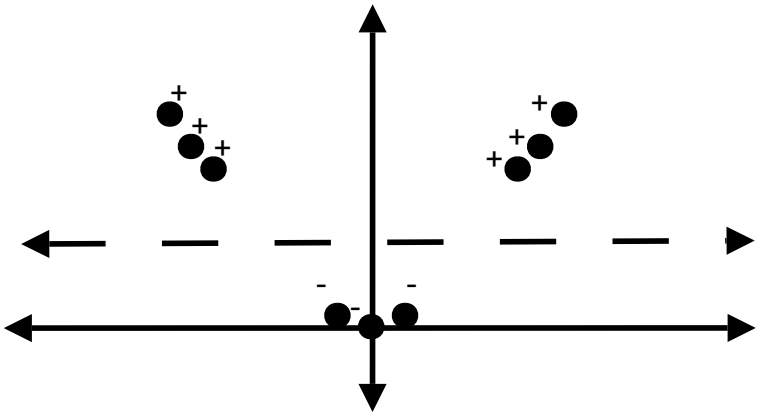

Figure 12.2.2: Data after nonlinear transformation. A separating hyperplane is now possible.

The data is now linearly separable due to the nonlinear transformation used. However, an explicit transformation will not be as useful when the dimension of $\varphi(\mathbf{x})$ becomes very large. For example, if $\varphi(\mathbf{x})$ consists of all monomials of $\mathbf{x} \in \mathbb{R}^N$ of degree d, then $\varphi(\mathbf{x})$ is $\binom{d+N+1}{d}$-dimensional, which is much too large for practical purposes. The goal now is do this embedding into a higher dimensional space implicitly.

Suppose $\varphi(\mathbf{x}) = [{x_1}^2, {x_2}^2, x_1x_2, x_2x_1]$. Then $\varphi(\mathbf{x})^T\varphi(\mathbf{x}') = {x_1}^2{x_1'}^2 + {x_2}^2{x_2'}^2 + 2x_1x_1'x_2x_2' = (\mathbf{x}^T\mathbf{x}')^2$. Therefore, to get the benefit of using monomials of degree 2, we need only replace the dot product $\mathbf{x}^T\mathbf{x}'$ with $(\mathbf{x}^T\mathbf{x}')^2$. In general, if $\varphi(\mathbf{x})$ is all ordered monomials of degree d, then $\varphi(\mathbf{x})^T\varphi(\mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^d$. Now we can "implicitly" work in a higher-dimensional space rather than performing the nonlinear transformation explicitly, merely by using a different dot product. This is called the *kernel trick*, and $\varphi(\mathbf{x})^T\varphi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$ is known as a *kernel function*. It is worth noting that this kernel trick works for other types of algorithms besides support vector machines, and typically involves rewriting an objective function and manipulating terms until everything relies on a dot product, where the kernel trick can be used.

Somewhat surprisingly, we can even do an implicit transformation into infinite-dimensional feature spaces if we use the right kernel function. For example, the kernel function

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2h^2})$$

is known as a *Gaussian* kernel function (also known as a squared exponential or radial basis function), and it corresponds to an inner product in an infinite-dimensional feature space.

Going the other direction, one might like to know what types of kernel functions correspond to inner products in higher-dimensional spaces, which the current analysis will now turn to.

12.2.3 Kernel Functions

Theorem 12.2.2 *Given some input space X, in order for a kernel function $k : X \to X$ to correspond to an inner product, it must satisfy:*

1. $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}) \quad \forall \mathbf{x}, \mathbf{x}' \in X$ *(Symmetry)*
2. *For $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\} \subset X$, the matrix*

$$K = \begin{pmatrix} k(\mathbf{x_1}, \mathbf{x_1}) & \ldots & k(\mathbf{x_1}, \mathbf{x_m}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x_m}, \mathbf{x_1}) & \ldots & k(\mathbf{x_m}, \mathbf{x_m}) \end{pmatrix}$$

known as the Gram matrix *or the* kernel matrix*, is positive semidefinite.*

This second condition can be hard to show, and is equivalent to each of the following conditions:

- $\forall \boldsymbol{\alpha} \in \mathbb{R}^m, \boldsymbol{\alpha}^T K \boldsymbol{\alpha} \geq 0$
- All eigenvalues of K are non-negative.

We also have the following property of kernel functions:

Suppose k_1, k_2 are kernel functions, $\alpha, \beta \geq 0$. Then $k(\mathbf{x}, \mathbf{x}') = \alpha k_1(\mathbf{x}, \mathbf{x}') + \beta k_2(\mathbf{x}, \mathbf{x}')$ is also a kernel function.

In particular, if $\alpha = \beta = 1$, we have that $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$ is a kernel function.