Support Vector Machines for Classification

Applications with Random Point Clouds and Image Sets

Ryan Honea

Austin Peay State University

December 8th, 2017

Introduction

The Classification Problem

Defining the Classification Problem:

- Consider a set of elements A that contains two subsets of elements defined as A^- and A^+.
- Let $x_1, ..., x_n \in A^+$ and let $y_1, ..., y_n \in A^-$
- Now let $z \in A$, but it is unknown whether or not $z \in A^+$ or $z \in A^-$.
- The objective of a classification problem is to define some rule that could determine the subset in which z lies.
- This problem can be generalized to have a set of elements A with n subsets and determining which subset A^n that z is an element of.
- Typically, an n-subset decision problem requires $n-1$ decision boundaries.

Decision Rules and Boundaries

We can define a general decision rule as such

$$z \in \begin{cases} A^-, & \text{if } P(A^-|z) > P(A^+|z) \\ A^+, & \text{otherwise} \end{cases}$$

but perhaps a more specific version utilizing Baye's Formula

$$z \in \begin{cases} A^-, & \text{if } P(z|A^-)P(A^-) > P(z|A^+)P(A^+) \\ A^+, & \text{otherwise} \end{cases}$$

These decision rules are a generalization of Bayes' Decision Rules. In it's most simple form that assumes independence and randomness of elements, an algorithm called Naive Bayes determines what subset z belongs to.

Define the Support Vector Machine

- Again, consider a set of elements A with subsets A^- and A^+ with elements $\vec{x_-} \in A^-$ and $\vec{x_+} \in A^+$.
- The objective in using Support Vector Machines is to create a hyperplane ω that intersects the set of elements A in such a way that the distance d from the closest point of A^- and A^+ to ω is maximized.
- That is, we seek to find ω that maximizes d
- We then define a new term

$$y_i = \begin{cases} +1, & \text{if } x_i \in A^+ \\ -1, & \text{o/w} \end{cases}$$

Maximizing Distance

- This creates the following decision rule, where $\vec{v}$ is a vector that is perpendicular to ω

$$\text{If } v_i * x_i \text{ is beyond the decision boundary, } x_i \in A^+$$

- This idea defines two constraints

$$\vec{v} \cdot \vec{x_+} + b \geq +1$$

$$\vec{v} \cdot \vec{x_-} + b \leq -1$$

- This constraint is simplified by y_i to be

$$y_i(v_i \cdot x_i + b) \geq 1$$

Quadratic Programming

Because $\vec{v}$ is unknown, we seek to find $\vec{v}$ such that d is maximized. For sake of time, this becomes a quadratic programming problem that seeks to solve

$$v(\alpha) = \sum_i^n \alpha_i - \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$

$$\sum_i^n \alpha_i y_i = 0, \qquad \alpha_i \geq 0$$

There are a number of ways to maximize $\vec{v}$ through quadratic programming, and those are all a key part of the algorithm.

Training, Testing Datasets

Steps to Machine Learning

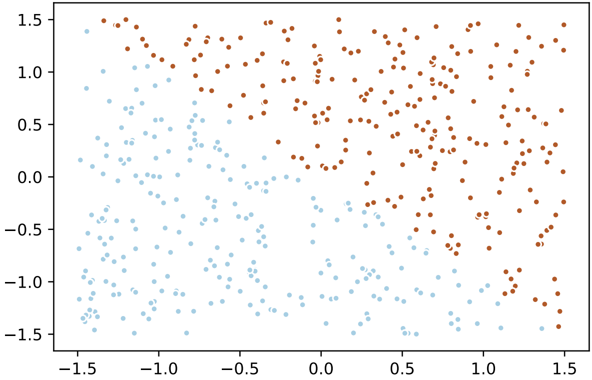
1. Define a training subset and testing subset of data
2. Define decision boundary based on training subset
3. Predict on the testing subset and calculate accuracy/error
4. If low accuracy, attempt other methods and repeat steps 1-3

The Linear Case

Solving the Linear Case

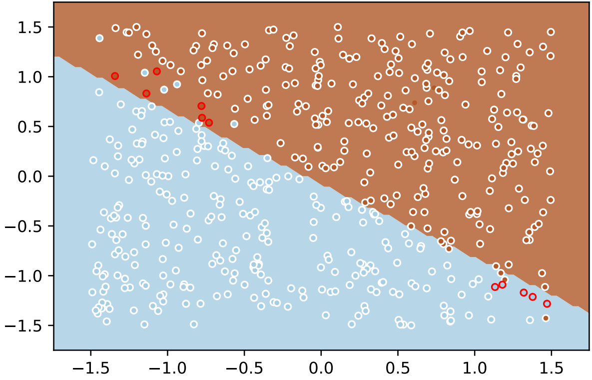
Objectives

- In the linear case, we try to draw a straight line through a series of points to define the hyperplane
- We use the algorithm to define the plane
- Example points:



Results

- In this case the testing subset is 50% of the data and the training subset is the other 50% of the data
- Errors in testing are represented by red edges



Comparison to KNN, LDA, QDA

Average Accuracy over one-hundred runs of linear case with 500 training samples and 500 testing samples:

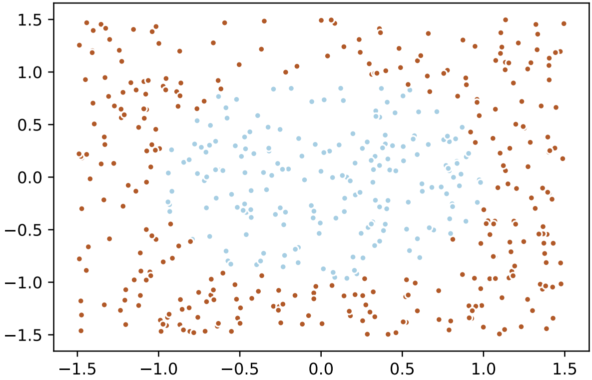
- Linear SVM-Average: 0.98006012024
- Linear LDA-Average: 0.97995991984
- Linear QDA-Average: 0.979944583044
- Linear KNN-Average: 0.979820465674

The Polynomial Case

Solving the Polynomial Case

Objectives

- What's about the case where we can't just draw a straight line?
- How do we draw a line through this set of samples?



Kernel Trick

Instead of maximizing

$$v(\alpha) = \sum_{i}^{n} \alpha_i - \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$

we maximize

$$v(\alpha) = \sum_{i}^{n} \alpha_i - \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j)$$

where $K(\vec{x}_i, \vec{x}_j)$ is a function that maps two samples to some distance.

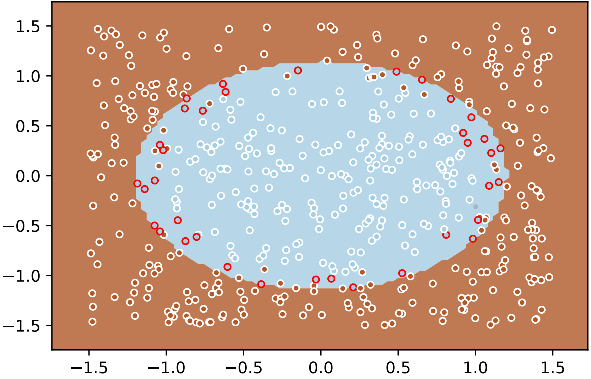
Mercer's Condition and Example Kernels

Mercer's Condition states that $K(\vec{x}_i, \vec{x}_j)$ must provide some measure of distance. This provides us with the means to develop a number of kernels.

- Linear Kernel: $\vec{x}_i^T \vec{x}_j + b$ where b is some bias.
- Polynomial Kernel: $(\vec{x}_i^T \vec{x}_j + b)^d$ where b is some bias and d is chosen dimension of polynomial
- Radial Basis Kernel: $\exp\left[\dfrac{||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}\right]$ where σ is some smoothing factor.
- Sigmoid Kernel: $\tanh(\gamma * \vec{x}_i^T \vec{x}_j + b)$ where γ is chosen to maximize accuracy under a logistic model.

Results

Choosing the polynomial kernel with dimension 2, we obtain the below results:



Comparison to KNN, LDA, QDA

Average Accuracy over one-hundred runs of linear case with 500 training samples and 500 testing samples:

- Polynomial SVM-Average: 0.837870741483
- Polynomial LDA-Average: 0.83778668448
- Polynomial QDA-Average: 0.837660013905
- Polynomial KNN-Average: 0.837768320145

The Image Case (MNist Database)

MNist Database – Handwriting Digits

Now that we've seen this on point clouds, we consider a set of images, specifically in this case handwriting.

- Utilizing the MNist Dataset of hand-writing samples, I compare numbers labeled 5 and 1 to each other in an attempt to classify them from each other.
- Example of this database's images below:

Results

Using a polynomial-2 kernel, the following results were observed:

- With a training set of 12163 images of 1s or 5s and a testing set of 2027 images of 1s or 5s
- Accuracy is approximately 100%.
- Computational time on a single core running at 2.6GHz was roughly 2 hours.
- Would show images, but most are in ascii format and don't show well.
- The algorithm can conclusively differentiate between the number 5 and the number 1.

Conclusion

Strengths/Weaknesses of Support Vector Machines

Strengths

- Popularized the "kernel trick" as a method for improving already used classification systems
- Computationally efficient with $O(nd^2)$ where n is number of samples and d is number of dimensions.
- Minimizing number of points needed for algorithm following quadratic maximization results in extremely fast prediction time
- Able to form complex boundaries by use kernel trick

Weaknesses

- Primary weakness lies in weakness of kernel trick. For tasks such as facial recognition, multiple kernels are required which reduce accuracy compared to neural nets or clustering techniques.
- Sets with large dimensions reduce the computational efficiency which lends to using our machine learning methods
- Easily falls into over-fitting problems for difficult point clouds

Future Work

Items for future development

- One vs. All method for multiple classification (i.e. classifying on numbers 0-9 as opposed to 1 and 5)
- Increased number of kernels for different tasks (such as facial classification or map identification)
- Parallel Computing versions in order to handle larger datasets

References

The mnist database.
THE MNIST DATABASE.

Corinna Cortes and Vladimir Vapnik.
Support-vector networks.
Machine Learning, 20(3):273–297, Sep 1995.

Yann Lecun, Leon Bottou, Yoshua Bengio, and Pattrick Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, Nov 1998.

MIT.
Learning: Support vector machines, Jan 2014.