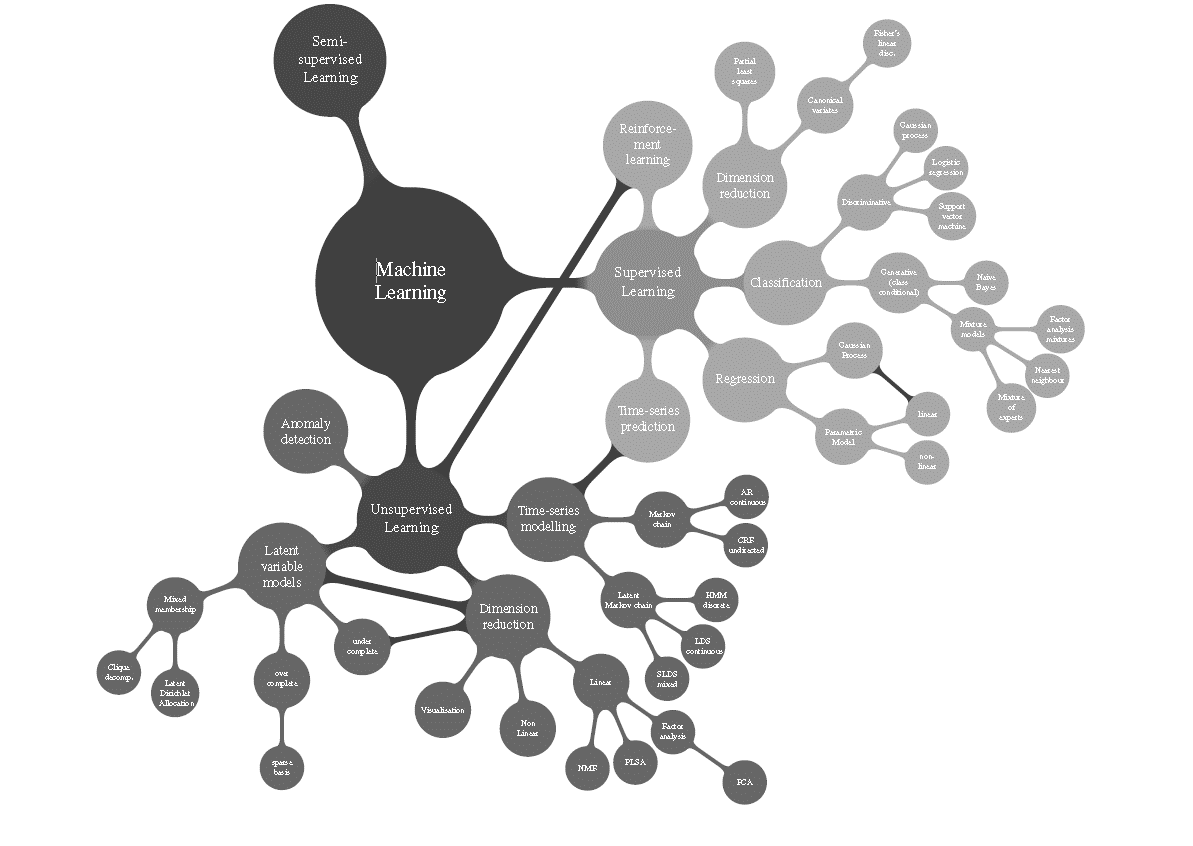
**Support Vector Machines**

12/02/2020

This document has been developed by F. Atiyya Shaw.

Please direct questions/comments to [atiyya@gatech.edu](mailto:atiyya@gatech.edu).



**SVM*, at a glance***

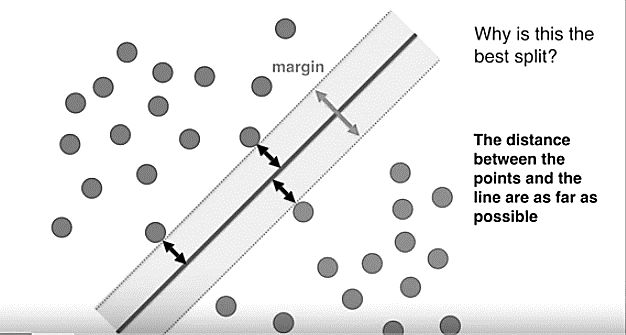
* **Supervised** Learning Algorithm
  + in a nutshell: given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other
  + This is a non-probabilistic binary linear classifier
* Can be adapted to both **classification + regression** approaches
  + i.e. predict both continuous and discrete outcomes
* When data are not labeled, and an unsupervised learning approach is necessary, supervised vector ***clustering*** can be used.
  + i.e. natural clustering of the data to groups, and then map new data to these formed groups

**SVM*, conceptually***

https://www.express.co.uk/news/nature/730827/Dog-cat-incredible-pet-viral-online-baffles-animal-lovers

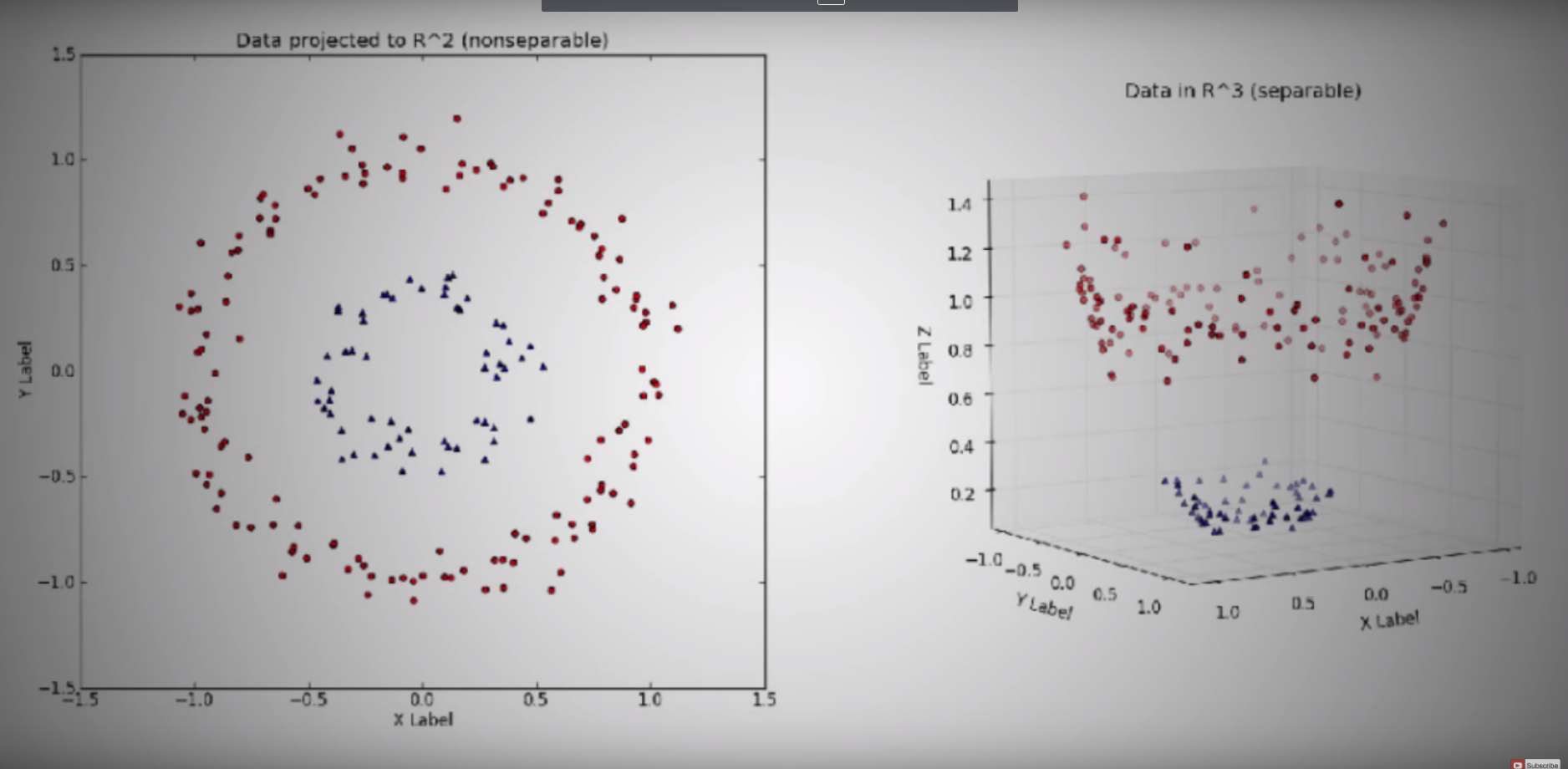
* SVM is a model that best **splits** the data
  + Looking for the widest margin that separates 2 groups
    - Great ML tool for differentiating between extreme cases, i.e. cases that are close to each other, and thus hard to distinguish
  + We want to maximize the margin
    - i.e. SVM is a **constrained optimization** problem
    - what’s the constraint? The data points can’t lie in the margin
      * note, this constraint can be relaxed a bit *if you want*.
    - **Support vectors**
      * Points that the margin pushes up against or points that are close to the opposing class
      * Implies that support vectors are important and training cases are ignorable.

Hyperplane



what happens if we want to keep more constraints in mind? dogs and cats don’t differ just based on their ear geometry and snout length. muffins and cupcakes don’t differ just based on the amount of flour and sugar proportions in each.

* We can transform the data in higher dimensional feature space
  + Downside? This is computationally expensive
  + So, we can use a kernel trick
    - Takes as inputs vectors in the original spaces and return dot products of vectors in the feature space
    - Kernel types that we can use to translate into high dimensional space
      * Linear kernel
      * Polynomial kernel
      * Radial basis function RBF kernel
      * Sigmoid kernel
      * <https://www.youtube.com/watch?v=H_I0pYdzBSk>
      * https://www.youtube.com/watch?v=-Z4aojJ-pdg
    - Using a kernel type is difficult and requires tuning
      * Parameter tuning using k-fold cross validation



*Advantages*:

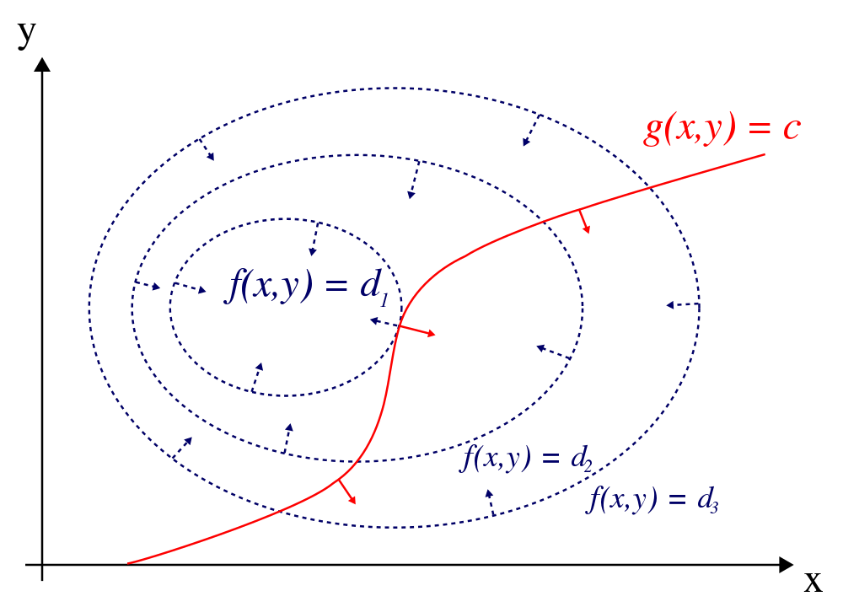
* Effective in high dimensional spaces
* Effective in cases where number of dimensions (cat vs. dog) is greater than number of samples
* Different kernel functions for various decision functions
* We can add kernel functions together to achieve even more complex hyperplanes

*Disadvantages*:

* Poor performance when # features (hair length, snout length, ear geometry, etc) is greater than number of samples
* We can break it if we have data with lots of error
  + Discriminator location depends entirely on the few nearest data points
* Choosing the wrong kernel
  + Kernel selection is trial and error, then by experience
* SVMs do not provide probability estimates
  + Calculated using an expensive K-fold cross validation
  + Large datasets: calculating the kernel is expensive

**SVM*, mathematically***

* Remember how we said SVM is a constrained optimization problem?
  + The way to solve such problems are via **LaGrange Multipliers**
    - A LaGrange Multiplier is a way to find local maxima and minima of a function subject to equality constraints
  + <https://www.youtube.com/watch?v=ax8LxRZCORU>



***SVM, in Oakridge code***

*Classification*

svm = (function() {

fit <- e1071::svm(f, data=work[folds != yk[id,4],])

hit <- mean(predict(fit, work[folds == yk[id,4],]) != work[folds == yk[id,4],yk[id,3]])

return(hit)

}) (),

*Regression*

svm = (function() {

fit <- e1071::svm(f,data=work[folds != yk[id,4],])

mse <- mean((predict(fit, work[folds == yk[id,4],]) - work[folds == yk[id,4],yk[id,3]])^2)

rsq <- cor(predict(fit, work[folds == yk[id,4],]),work[folds == yk[id,4],yk[id,3]])^2

return(list(mse,rsq))

}) (),

***Other resources used in the making of this document***

* <https://www.youtube.com/watch?v=N1vOgolbjSc>
* <https://en.wikipedia.org/wiki/Support_vector_machine>
* <https://en.wikipedia.org/wiki/Lagrange_multiplier>