3 Stages of Understanding Machine Learning

1) Capable of using packages to implement
2) Comprehend the underlying methodology / theory

} Our goal

3) Construct your own implementation ← NOT the purpose here

Overview from Bank Executives

1) Always compare results with a "base case" (ex: without NLP features) to demonstrate value added

2) True data analytics requires domain knowledge to clean, pre-analyze, choose appropriate methods, + interpret results → cross-disciplinary

3) Big data can give clean + precise results, but can lead to massive failures/losses if used improperly / carelessly

Review

Regression

Goal is to investigate the relationship between dependent variables Y & independent variables X

Can be parametric or nonparametric

Parametric Regression

ex: Linear regression, neural networks, ...

Begin with some parametric family of functions $f(\cdot, \beta)$

Goal is to estimate β from the data

↪ Given paired data (X_i, Y_i), we want β so that $Y_i \approx f(X_i, \beta)$

Generally, we write $Y_i = f(X_i, \beta) + e_i$ for errors e_i

How do we define the best β? minimize the errors

$$\sum_{i=1}^{N} \left(Y_i - f(X_i, \beta)\right)^2$$

$$\sum_{i=1}^{N} \left|Y_i - f(X_i, \beta)\right|$$

$$\max_{i=1,\dots,N} \left|Y_i - f(X_i, \beta)\right|$$

These loss functions differ in how they penalize the large errors on few points

Choice of error function can greatly impact the regression estimated

Linear Regression

Want to predict y using data x

↳ $y = a^T x + b + \boxed{\text{error}}$

Often estimated with least squares error

↳ 1) Mathematically analytical solution exists

2) Computationally easy

3) Tradeoff between minimizing outlier errors + controlling near the "average"

Can easily be generalized to polynomial regression

ex: $y = a_2 x^2 + a_1 x + b + \text{error}$

with modified data set $((x^2, x), y)$ is a linear regression

Be wary of overfitting if using high dimensional models without an underlying reason (for example in physics using laws of motion)

To test for overfitting:

↳ 1) Split data into test + training data

Estimate regression only on the training data

Validate results with the test data

2) Cross-Validation

Repeat train/test split multiple times

Consider the 'typical' errors

If test data performs poorly (in comparison to training data) then we are likely overfitting

---

Can also be used in <u>autoregressive models</u>

↳ Typical in time series analysis (FA 542)

↳ Output desired is "tomorrow's" data

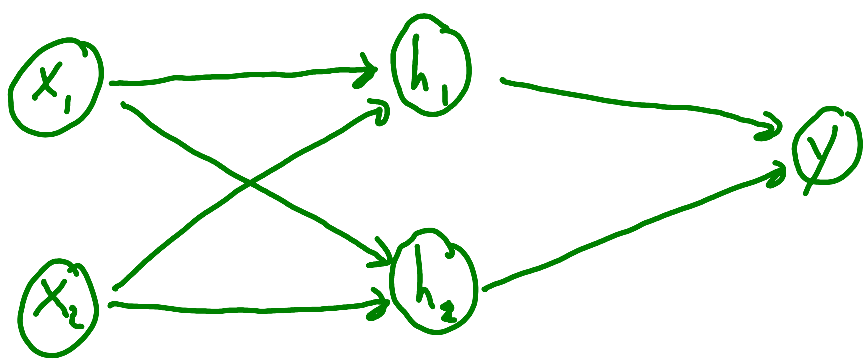
<u>Neural Networks</u>

<u>Feed Forward Neural Networks</u>

Typically (very) nonlinear models with complex structures

Regression model $f(\cdot, \beta)$ that can be decomposed into layers

<u>ex</u>:



$$
\begin{aligned}
y &\approx f(x_1, x_2, \beta) \\
&= f^{(2)}(h_1, h_2, \beta^{(2)}) \\
&= f^{(2)}\left(f_1^{(1)}(x_1, x_2, \beta_1^{(1)}), f_2^{(1)}(x_1, x_2, \beta_2^{(1)}), \beta^{(2)}\right)
\end{aligned}
$$

As before, the choice of error function matters

Also need to decide:

↳ 1) The shape of the network

a) how many hidden layers are there

b) how many nodes in each layer

c) how are these layers connected

2) The form of the regression functions $f_i^{(k)}$

Often choose $f_i^{(k)}(x, \beta) = g_k(W^{ik} x + b^{ik})$ for <u>activation function</u> g_k

$\beta = (W^{ik}, b^{ik})$

<u>ex</u>: a) ReLU: Rectified linear unit. $g(x) = \max(x, 0)$

b) Sigmoid: Logistic function: $g(x) = \dfrac{1}{1 + e^{-x}}$

3) Parameter constraints (<u>ex</u>: $\beta \geq 0$)

Classification: Closely related to the regression problem

Goal is to predict a (discrete) class Y from data X

Can view as a regression with discrete output space

Basics

Classifier f takes input data & provides a "guess" for a class label

ex. $f(x) = \begin{cases} 1 & \text{if } \ell(x) \geq 1/2 \\ 0 & \text{if } \ell(x) < 1/2 \end{cases}$

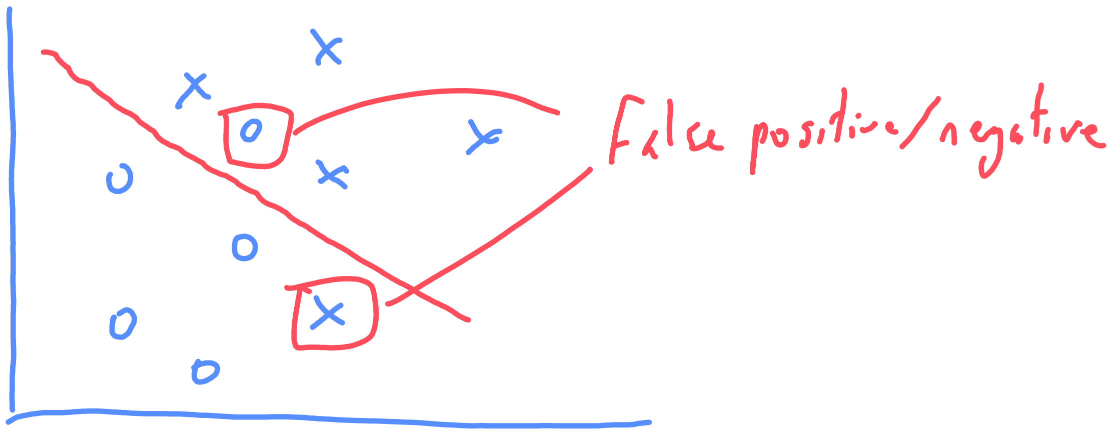
Confusion Matrix for Binary Classification

Predicted class \ Actual Class	1	0
1	TP	FP
0	FN	TN

A large number of accuracy scores are derived from the confusion matrix

Linear Discriminant Function

Idea: Want a linear function $a^T x + b$ so that $f(x) = \begin{cases} 1 & \text{if } a^T x + b \geq 0 \\ 0 & \text{if } a^T x + b < 0 \end{cases}$



Related to the logistic regression

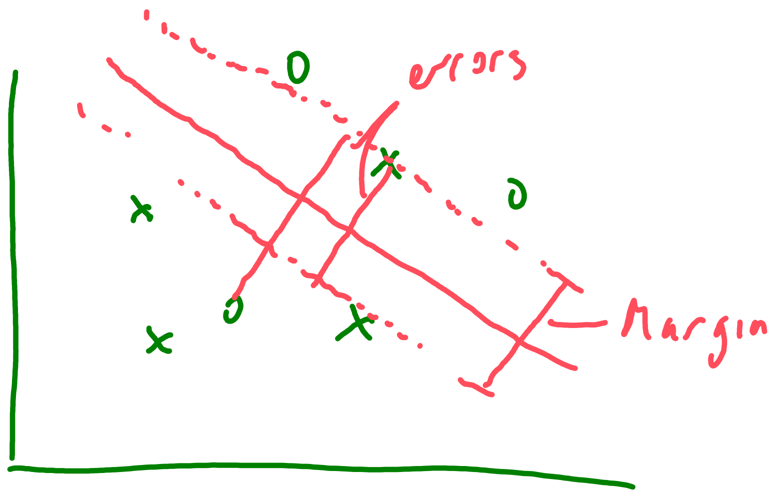
↳ assumes there is a linear relationship between x + the "log-odds" of $y = 1$

$$\hookrightarrow \; l = \log\left(\frac{p}{1-p}\right) = a^T x + b \Rightarrow p = \left(1 + \exp\left(-[a^T x + b]\right)\right)^{-1}$$

Note: $p \geq 1/2 \iff a^T x + b \geq 0$

Support Vector Machine

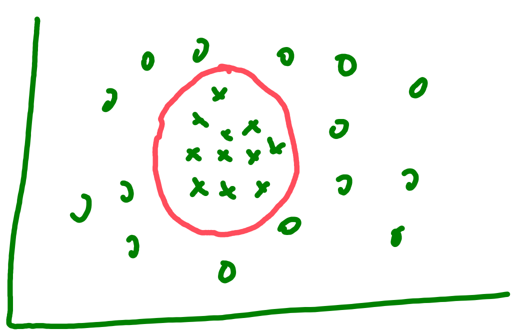
Seeks to create the "widest" gap between the 2 classes while minimizing the error from misclassification (in the training data)



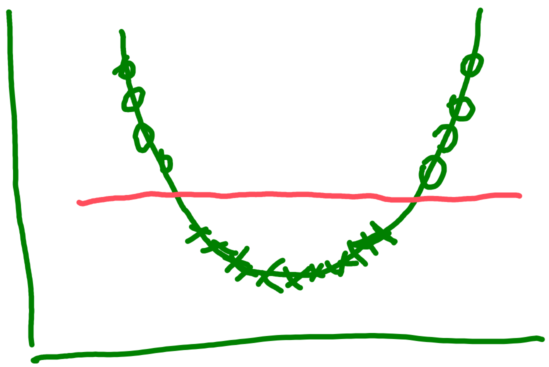
Need to decide how to determine the relative costs of errors vs. the margin

"Plain" SVM finds a linear classifier

What if our data has a nonlinear relationship?



Use the Kernel trick: remap/change the dimensions of the original inputs so that the classes are linearly separable in the remapped space



Decision Tree & Random Forest

Multiple algorithms available to construct decision trees

Idea: Sequentially partition the data until the class is "clear"

Benefit: very easy to interpret

Cost: generally very sensitive to training data + subject to overfitting

Solution: Random Forest

↳ Idea: Create many different trees from (bootstrapped) training data

Choose the class that is most commonly predicted

This is a type of ensemble model (model with the "vote" of many algorithms)

Clustering: Unsupervised Learning

What if we do not have a target output y?

Want to generate the classes from the data itself

Similarity & Distance

To decide if 2 points are "close", we need to define what that means

ex: 1) Euclidean Distance: $d(X,Y) = \sqrt{\sum_{k=1}^{N} (X_k - Y_k)^2}$

2) Manhattan Distance: $d(X,Y) = \sum_{k=1}^{N} |X_k - Y_k|$

can be skewed by data if 1 feature is orders of magnitude larger

3) Cosine Distance: $d(X,Y) = 1 - \dfrac{\sum_{k=1}^{N} X_k Y_k}{||X|| \, ||Y||}$

Ignores the "scale" of the features

Often used for text classification

Nearest Neighbor Clustering

Weight neighbors by how close they are

ex: $\text{score}(c) = \sum_i \frac{1}{d(x, y_i)} * [\text{class}(y_i) = c]$

cluster by largest score

K-Means Clustering

(K = # of clusters)

Closely related to nearest neighbor

↳ 1) Pick K data values at random → called "centroids"

2) Assign clusters by putting data points in the cluster with its nearest centroid

3) Determine new cluster centers by computing the average of the points in that cluster

4) Repeat steps 2 + 3 until convergence

Often repeated many times with different initial centroids

Try with different number of clusters K