StatQuest - machine learning

Bias Variance trade off 

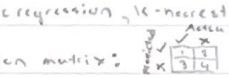
the most important thing isn't how fancy an ML is, but how it performs with Testing Data

Decision Tree

machine learning is all about making predictions and classifications

Cross validation: allows us to compare different machine learning methods and get a sense of how well they work in practice.

Logistic regression, k-nearest neighbor, support vector machines, random forest

confusion matrix: 

1: true positive 3: false negative
2: false positive 4: true negative

Sensitivity = $\frac{\text{true positive}}{\text{true positive + false negative}}$ = % of positives correctly identified

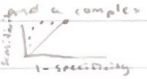
Specificity = $\frac{\text{true negatives}}{\text{true negatives + false positives}}$ = % of negatives correctly identified

Linear regression = least squares

bias: the inability for a machine learning method to capture a true relationship

Variance: difference in fits between datasets ... high variance = overfitting

Sweet spot: between a simple model and a complex model ... regularization, boosting and bagging are methods to find sweet spot

Reciever Operator Characteristic graph: 

... find best threshold for making a decision

Precision = $\frac{\text{true positives}}{\text{true positives + false positives}}$ = proportion of positive results correctly classified

ROC and AUC in R

Surprise = $\log\left(\frac{1}{\text{probability}}\right)$ ... Entropy: expected surprise of an event = $\sum_{\text{specific value } x} P(x=x) \times \log\left(\frac{1}{P(x=x)}\right) = -\sum P(x)\log(P(x))$
probability of observing that value for surprise

Mutual Information: $\sum_{x \in X}\sum_{y \in Y} P(x,y)\log\left[\frac{P(x,y)}{P(x)P(y)}\right]$ ... (perf correlation = 0.5)

↳ tells us how on average, the surprise or change or information we see in one variable is related to the surprise, change, information in another

Sum of squares around the mean = $\sum(\text{data} - \text{mean})^2$ ... variation around the mean = average sum of squares

$R^2 = \frac{\text{Var(mean)} - \text{Var(fit)}}{\text{Var(mean)}} = \frac{\text{SS(mean)} - \text{SS(fit)}}{\text{SS(mean)}}$ (Var of best fit) = % x "explains" y

$F = \frac{\text{\% y is explained by x}}{\text{\% y that isn't explained by x}} = \frac{(\text{SS(mean)} - \text{SS(fit)})/((P_{fit} - P_{mean}))}{\text{SS(fit)}/(n - P_{fit})}$ degrees of freedom

P_{mean} = # of parameters in meaning
P_{fit} = # of parameters in fitting line

t tests, anova

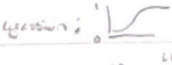
Odds = ratio of $\frac{\text{something happening}}{\text{something not happening}}$ ≠ probabilities = ratio of $\frac{\text{something happening}}{\text{everything that could happen}}$ ... $\text{odds} = \frac{p}{1-p}$

$p < 0.5 \Rightarrow 0 < \text{odds} < 1$ ... $p > 0.5 \Rightarrow 1 < \text{odds} < \infty$ ... log(odds) makes things symmetric and easier for finding statistics

"odds ratio" = "ratio of odds" ... statistically significant? fisher's exact test, chi-squared test, wald test

rule of thumb: anything further than 2 standard deviations from the mean will have a p-value < 0.05, so log(odds ratio) is statistically significant

logistic regression predicts True or False instead of predicting something continuous

logistic regression: 

(transform y axis to $\frac{\log(\text{odds})}{\text{(p value)}}$ or $\log\left(\frac{p}{1-p}\right)$) ... $p = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$

logistic regression $R^2 = \frac{\text{LL(overall probability)} - \text{LL(fit)}}{\text{LL(overall probability)}}$ ... a chi-squared value = 2(LL(fit) - LL(overall probability))

Log Likelihood

Saturated models and Deviance

- understand properties of log and e

Logistic regression in R

Deviance Residuals

Regularization: Ridge Regression (a larger λ makes you less sensitive to the x axis variable), Lasso Regression

Elastic-Net Regression: groups and shrinks parameters associated with the correlated variables and leaves them in equation or removes them.

Principle Component Analysis: PC1, Eigenvector = $\frac{SS(\text{distances for PC1})}{n-1}$, singular value: $\sqrt{SS(\text{distances for PC1})}$

dimension reduction — PC2: perpendicular to PC1, Eigenvalue: $\frac{SS(\text{distances for PC2})}{n-1}$ ... variation for PC: $\frac{SS(\text{distances for PC})}{n-1}$

Linear Discriminant Analysis: like PCA, but focuses on maximizing separability among known categories; $\frac{(\bar{x}_1-\bar{x}_2)^2}{s_1^2+s_2^2}$ = distance between two means / scatter

MDS and PCoA → distances among samples; PCA → correlations among samples → eigen decomposition → coordinates for a graph; % variation each axis accounts for; loading scores (to determine variable w/ greatest effect)

t-SNE

Hierarchical clustering, K-means clustering, DBSCAN, K-nearest neighbor

Naive Bayes: discrete probabilities are also called likelihoods ... super cool, like this, super good for spam detection! (high bias + low variance)

Gaussian Naive Bayes: also super cool! for continuous variables

Decision and Classification Trees: super cool! can use gene impurity or information gain

feature selection and missing data: lots of ways to guess what missing data might be

Regression Trees: super cool! sum of square residuals

Cost complexity Pruning: 10 fold cross validation?

One-Hot encoding, Label encoding, Target encoding, Bayesian Mean Encoding, K-fold target encoding

(1 hour) Classification Trees in Python

Random Forest: random, super cool ... no matter what data are, if you can use it to make a tree, you can make a similarity matrix and then draw a heat map or an MDS plot to show how samples are related to each other.

Gradient Descent: Determine loss function, (sum of squared residuals). derive loss function, determine learning rate and multiply by derivative equation output of initial guess. new guess = initial guess − stepsize. (loss function must be differentiable for gradient descent to be possible)

Stochastic gradient descent: uses a randomly selected subset of the data (randomly one sample or a minibatch) at every step rather than the full dataset.

○ Adaboost, Gradient Boost, XgBoost

○ XGBoost in Python

Cosine similarity: $= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$ ... Support Vector Machines: maximal margin classifier ... Polynomial kernel: $(a \times b + r)^d$

... radial kernel: $e^{-\gamma(a-b)^2}$ ... support vector classifier