Discussion Questions:

You should always standardize your predictors for knn as a first step – why?
$Distance metric used to make predictions

How do you handle categorical predictors using knn?
$No good way for knn to handle

For 1-nearest neighbor, what would a plot of versus x look for the gas mileage example with only Displacement as a predictor?
**$**g(x;$\theta$): Step function

How can you tell which predictors have the largest effect on the response in the K-nearest neighbors model?
$Hardest to interpret

What are the parameters of the fitted model (KNN)?
$No parameters. ALE Plot to interpret model

The Gaussian kernel weights decay more smoothly, but the other two "compact support" kernels have computational advantages. Why?
**$**Because they only contain values within a certain window. Think of |x-x_i|$\geq \lambda$. Normal doesn't have that.

How would increasing λ change the predictor plotted three slides earlier?
$It would make the window larger; $|\frac{x-x_i}{\lambda}| \geq \lambda$

Regarding the omnipresent bias/variance tradeoff, would increasing l
- *$Increase or **decrease** the variance of the $predictor?*
- ***Increase** or decrease the bias of the predictor?*

What is the relationship between a local linear regression model and a linear regression model? Is a local linear regression model "almost" linear?
$Not similar at all not even close

Where is a local linear regression predictor most likely to be biased?
$Boundary

With no interactions, what are the implications of the GAM model structure Y(x) = a + f1(x1)+ f2(x2)+ . . . + fk(xk) + e, in terms of the type of predictive relationships it can capture?
$It can't capture interaction terms

How would you handle categorical predictors in a GAM model?
$You can't unless its ordinal categorical

Neural networks with a linear output activation function and one hidden layer are very similar to PPR, but with the neural network logistic Hj(⍰) replaced by the completely nonparametric PPR fj(⍰) Yes

How can you interpret the PPR response surface and component functions?
$Higher β or v_1 means the function is more important. Most important coef in each v means it has the largest coef. Or just use ALE Plots

What is the computational expense associated with stacking? Which part of the stacking algorithm is the most computationally expensive?
$It's "horrible" for the same reason n-fold cv is bad (n*n) models

Would the computational expense be better or worse if you used K-fold, instead of n-fold CV

It could be somewhat better or it could still be awful. Then you're fitting nxk models

How does stacking relate to model selection using CV?
$Can give the ones better at predicting y more weight. View it as a linear combination of betas times g(x). Viewed as not worth the effort unless you have tons of time to fit all the models. Also we have better methods such as boosting and random forests using way less computational power.

Smaller λ generally gives better predictive performance but worse computational expense, since it requires a larger M
$Higher M leads to overfitting

The variable importance measure is simply the sum of the variable importance measures for each of the M individual trees
$For interpreting the model and the effects of the predictors

- Individual marginal plots are very useful but cannot show any interactions (and can be misleading if interactions are strong)
- Pairwise marginal plots show interactions between pairs of variables
- Printing out the first few trees using print(pretty.gbm.tree(gbm1,i)) (for i = 1, 2, . . .) and attempting to interpret these may also help, but this is usually difficult

$Random Forest:

- M: num of random subset of features for each tree
- Ntree: number of trees that were going to use
- Nodesize: controls size, number of leafnodes of each individual tree
- OOB MSE almost = CV MSE

Random forests can overfit if the individual trees are chosen too large, but as the # trees increases they do not increasingly overfit, unlike boosted trees (why?), so you do not have to worry about choosing ntrees too large
$Because you're just smoothing out the trees, since its just the average of all trees.

$Mean of squared residuals = OOB MSE
$% Var explained = OOD R squared
$%IncMSE (Permutation Based Var Importance)
$%IncMSE is the percentage increase in MSE if you randomly jumble the column youre looking at and try to predict the dependent variable. Randomly permuting the column breaks the dependence.

Random forests have virtually all the same many advantages of boosted trees. They inherit almost all the advantages of trees, but, like boosted trees, usually excellent predictive power. Unlike trees, they lose interpretability. But this can be improved with the build-in partial dependence (aka marginal) plots and the variable importance measures. Overall, random forests may give a little smoother model than boosted trees, whereas boosted trees may be a little better at capturing more complex nonlinearities.

Time Series
Trend: T_t: a deterministic linear, quadratic, exponential, etc. pattern over time
Predict: Simple extrapolation (linear, quadratic, etc.)
Seasonal: S_t: up/down variations with a regular period (yearly, weekly, daily, etc.)
Prediction: July=Average of past few Julys, March same
Cyclical: C_t: random longer-term up/down variations with an irregular period (often tied to the economy as a whole, war, natural disasters, etc.)
Prediction: Average of the past few observations (best prediction of the future value is just the current level
Random: R_t: the completely random, totally unpredictable part that is left over
Prediction: Don't even try

MA:
####MA filtering/prediction for chem.csv
```
chem<-read.csv("chem.csv",header=F)
y<-ts(chem[[1]], frequency=1)
m=20;k=20;n=length(y)  #m = MA window length, k = prediction horizon
plot(y,type="b",xlim=c(0,n+k))
MAchem<-filter(y, filter=rep(1/m,m), method = "convolution", sides = 1) (sides=1: non-centered moving average, 2:centered moving average)
yhat=c(NA, MAchem, rep(MAchem[n],k-1))  #One-step-ahead forecasts. The output of MAchem is L_t. The leading NA in yhat gives yhat_t = L_(t-1)
```
$$L_t = \frac{y_t + y_{t-1} + \Lambda + y_{t-m+1}}{m}$$

EWMA:
####Manual EWMA filtering/prediction for chem.csv
```
y<-ts(chem[[1]], frequency=1)
alpha=0.2;k=20;n=length(y)  #alpha = EWMA parameter, k = prediction horizon
plot(y,type="b",xlim=c(0,n+k))
EWMAchem<-filter(alpha*y, filter=1-alpha, method = "recursive", sides = 1, init=y[1])
yhat=c(NA,EWMAchem,rep(EWMAchem[n],k-1))
lines(yhat,col="red")
```

EWMA (generally better than MA)

- The EWMA estimate of the current level is defined as:

$$L_t = \alpha\, y_t + (1-\alpha) L_{t-1}$$

current estimate | current observation | previous estimate

i.e., L_t = weighted average of y_t and forecast (L_{t-1}) of current level

you pick an $0 < \alpha \leq 1$ and a starting value L_0, and then calculate L_t recursively for $t = 1, 2, 3, \ldots$

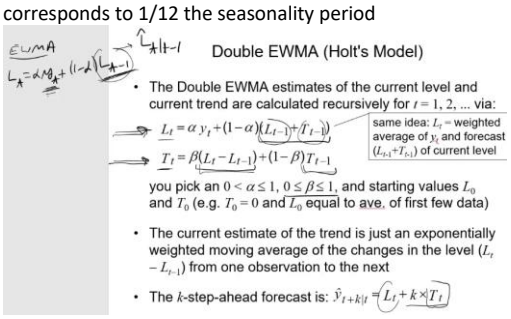
- An equivalent form of the EWMA is:

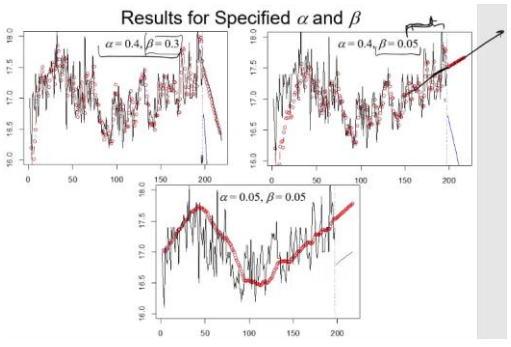
$$L_t = \alpha\left[y_t + (1-\alpha) y_{t-1} + (1-\alpha)^2 y_{t-2} + (1-\alpha)^3 y_{t-3} + \Lambda\right]$$

which shows that the EWMA really is an "exponentially weighted moving average" of the past observations.

- The k-step-ahead forecast is: $\hat{y}_{t+k|t} = L_t$

Holt Winters:
```
y<-ts(chem[[1]], frequency=1)
k=20;n=length(y)  #k = prediction horizon
EWMAchem<-HoltWinters(y, seasonal = "additive", beta = FALSE, gamma = FALSE)
EWMAchemPred<-predict(EWMAchem, n.ahead=k, prediction.interval = TRUE, level = 0.95)
plot(EWMAchem,EWMAchemPred,type="b")
y<-ts(trade[[1]], deltat=1/12)  #sampling interval corresponds to 1/12 the seasonality period
```





Full Holt Winters:

Holt-Winters Additive Method

- The **Holt-Winters Additive** method is just a slight modification of the Holt method:

$L_t = \alpha(y_t - S_{t-s}) + (1-\alpha)(L_{t-1} + T_{t-1})$ same except $y_t \to y_t - S_{t-s}$

$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$ same

$S_t = \gamma(y_t - L_t) + (1-\gamma)S_{t-s}$ new but same logic

select or fit an $0 < \alpha \le 1$, $0 \le \beta \le 1$, $0 \le \gamma \le 1$, and starting values L_0, T_0, S_0, S_{-1}, S_{-2}, … (e.g. $T_0 = S_0 = S_{-1} = S_{-2} = \ldots = 0$) and L_0 equal to ave. of first few data)

- Note that when estimating the level L_t, we subtract the estimated seasonality from y_t. When estimating the seasonality S_t, we subtract the estimated level from y_t.
- The k-step-ahead forecast is: $\hat{y}_{t+k|t} = L_t + k \times T_t + S_{t+k-s}$

Holt-Winters Multiplicative Method

- The **Holt-Winters Multiplicative** method is a slight modification of the Holt-Winters Additive method:

$L_t = \alpha(y_t / S_{t-s}) + (1-\alpha)(L_{t-1} + T_{t-1})$ same except $y_t - S_{t-s} \to y_t / S_{t-s}$

$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$ same

$S_t = \gamma(y_t / L_t) + (1-\gamma)S_{t-s}$ same except $y_t - L_t \to y_t / L_t$

select or fit an $0 < \alpha \le 1$, $0 \le \beta \le 1$, $0 \le \gamma \le 1$, and starting values L_0, T_0, S_0, S_{-1}, S_{-2}, … (e.g. $T_0 = 0$, $S_0 = S_{-1} = S_{-2} = \ldots = 1$ and L_0 equal to ave. of first few data)

- Note that when estimating the level L_t, we *divide* y_t by the estimated seasonality. When estimating the seasonality S_t, we *divide* y_t by the estimated level.
- The k-step-ahead forecast is: $\hat{y}_{t+k|t} = (L_t + k \times T_t) \times S_{t+k-s}$

Decomposition:

Fits, Forecasts, and Residuals for Decomposition Models

- After fitting a decomposition model, the fits, forecasts (which requires a parametric trend to extrapolate), and residuals are

$$\hat{y}_{n+k|n} = \begin{cases} T_{n+k} + S_{n+k}: & \text{additive} \\ T_{n+k} \times S_{n+k}: & \text{multiplicative} \end{cases} \quad \text{(forecasts)}$$

$$\hat{y}_{t|t-1} = \begin{cases} T_t + S_t: & \text{additive} \\ T_t \times S_t: & \text{multiplica tive} \end{cases} \quad \text{(fits are the same)}$$

$$e_t = y_t - \hat{y}_{t|t-1}$$

- Beware forecasting with decomposition methods!!
  - parametric trends often fit poorest at the ends of the data
  - the seasonalities and trend may change over time
  - Holt-Winters and ARIMA are usually better for forecasting

Usually the fit is worse at the ends of the data

In the decomposition model results, why are there no trend and residuals estimated for the first six months and last six months?
$Because it's using a centered moving average filter

Extras:

$$R^2 = 1 - \frac{var(ytest - fit)}{var(ytest)} = 1 - \frac{MSEAve}{var(y)}$$

$1 - R^2_{cv} = \frac{SSE_{cv}}{SST} = \frac{MSE_{cv}}{MST}$ given large n

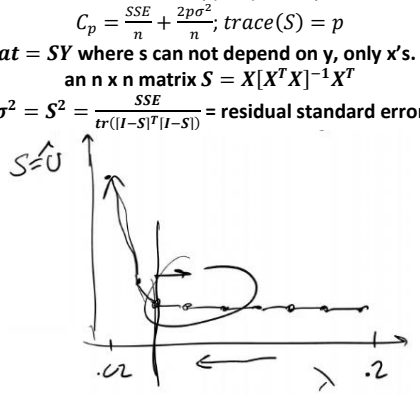
$$MSE = \frac{sum(y - yhat)^2}{n}$$

$$SSE = sum((y - yhat)^2)$$

$$C_p = \frac{SSE}{n} + \frac{2p\sigma^2}{n}; trace(S) = p$$

$\boldsymbol{Yhat = SY}$ where s can not depend on y, only x's. S is an n x n matrix $\boldsymbol{S = X[X^TX]^{-1}X^T}$

$\boldsymbol{\sigma^2 = S^2 = \frac{SSE}{tr([I-S]^T[I-S])}}$ = residual standard error



$$\boldsymbol{R^2_{cv} = 1 - gbm1\$cv.\frac{error[best.iter]}{var(CRT1\$Strength)}}$$

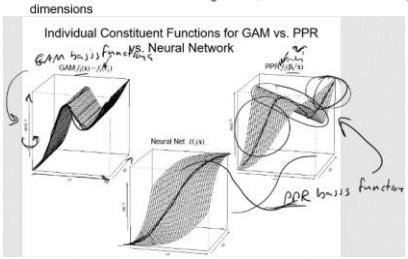
Summary(gbm1,n.trees=best.iter)=feature importance

Pros and Cons of Nearest Neighbors

- Pros:
  - The most flexible of all – can represent any nonlinear relationship (as long as you have sufficiently large n).
  - Easy to use. No model fitting required
- Cons:
  - There is no real fitted model (nor even any indication of which predictors are most important), so not suitable for interpretation or explanatory purposes.
  - Because there is no fitted model, you need to retain all the training data to predict.
  - With large k (the number of predictors), you need very large n, because neighbors get further away in higher dimensions.
  - For most supervised learning methods, large n increases the computational expense for training, but not for new case prediction. Large n is more problematic for nearest neighbors, because the "training" occurs for every new case prediction
  - With very large n, we need computational tricks (e.g. tree-based methods) to efficiently search for nearest neighbors.
  - Not well suited for categorical predictors

Summary of Local Methods (K-NN or Kernel)

- No assumed structure and completely nonparametric, so very flexible and avoids specifying the "wrong" model structure (pro), but poorer bias/variance tradeoff (con)
- Computational/Memory: No training step (pro), but must fit a new local model for each case to predict (con) and retain all of the original "training" data in memory (con)
- Comp. expense of K-fold CV independent of K (pro) and can be fast using the same K-NN computational tricks
- No good way to handle categorical predictors, especially for kernel methods
- Not well suited for high dimensions
  - Points are inherently sparser in high dimensions
  - Originally intended as a scatterplot smoother for visualization [which accounts for the ss in loess(), which allows at most 4 predictors]
  - Additive nonparametric models [ppr() and gam() in mgcv package in R] borrow ideas from local kernel regression, but are better suited for high dimensions



When Bagging Might Be Useful

- For predictors that are linear ($\hat{y}$ a linear function of training $\mathbf{y}$), bagging has no effect on the predictor
- The types of predictors that have potential to be most improved by bagging are ones for which:
  - fitting is unstable (i.e., a small change in the data gives a very different fitted model) and/or
  - the structure is such that the sum of multiple predictors no longer has the same structure
- Trees are the most important type of predictor that can be improved by bagging
  - Random forests are bagging with trees, but with a few whistles and bells added (boosted trees are related to random forests, but not quite bagging)
  - Bagging is not commonly used with predictors other than trees, so we will not cover bagging in its generality here

Advantages of Boosted Trees

- Trees have relatively poor predictive power, but have many nice properties for data mining problems:
  - handles missing data
  - robust to outliers in predictor space
  - insensitive to monotone transformation of predictors
  - computationally scalable (to large n)
  - handles mixed and categorical predictors with many categories
  - automatically discards irrelevant predictors
  - interpretable (for moderate sized trees)
  - convenient variable importance measure
- Boosting trees can substantially improve predictive power, while retaining almost all the advantages (except interpretability, but this can be improved with individual and pairwise marginal plots)

Discussion Points and Questions

- For classification, gbm() calculates the CV deviance, not the misclassification rate. How can we interpret this to assess how good the overall predictive power is?

the best CV deviance was 0.592

$$\text{deviance} = -2\times\text{ave}\{\log f(y_i)\}$$
$$= -2\times\text{ave}\{\log[p_i^{y_i}(1-p_i)^{(1-y_i)}]\}$$
$$= -2\times\text{ave}\{y_i\log(p_i) + (1-y_i)\log(1-p_i)\}$$
$$= -2\times\text{ave}\{\log(\alpha_i)\}$$

where $\alpha_i = \begin{cases} p_i: & y_i = 1 \\ 1-p_i: & y_i = 0 \end{cases}$ = predicted probability for correct class of y_i

thus, $\text{ave}\{\log(\alpha_i)\} = -\text{deviance}/2$, so that (very approximately)

$$\text{ave}\{\alpha_i\} \approx \exp\{-\text{deviance}/2\} = \exp\{-0.592/2\} = 0.74$$

The Permutation-Based VIM in the randomForest Package

- As the accuracy measure, it uses MSE for regression and misclass rate for classification.
- For computational reasons:
  - It uses OOB predictions, instead of CV predictions
  - The accuracy difference is calculated for each individual tree in the random forest, and then this is averaged across all *ntree* trees
  - The average difference is then normalized by the standard deviation of the differences
- In general, when the predictor variables are highly correlated, the permutation-based VIM is subject to the same "extrapolation" problem as partial dependence plots (why?)

Three Common Kernel Weighting Functions

Gaussian kernel: $K_\lambda(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\left[\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\lambda}\right]^2\right)$

Epanechnikov quadratic kernel:

$$K_\lambda(\mathbf{x}, \mathbf{x}_i) = \begin{cases} \frac{3}{4}\left(1 - \left[\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\lambda}\right]^2\right): & \|\mathbf{x} - \mathbf{x}_i\| \le \lambda \\ 0: & \text{otherwise} \end{cases}$$

Tri-cube kernel: $K_\lambda(\mathbf{x}, \mathbf{x}_i) = \begin{cases} \left(1 - \left[\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\lambda}\right]^3\right)^3: & \|\mathbf{x} - \mathbf{x}_i\| \le \lambda \\ 0: & \text{otherwise} \end{cases}$

Three Common Measures of Forecasting Accuracy

Define the *one*-step-ahead forecasting error (aka residual) for $t = 1, 2, \ldots, n$: $e_t = y_t - \hat{y}_{t|t-1}$

- **Mean Square Deviation**: $MSD = \frac{\sum_{\text{all forecasts}} e_t^2}{\#\ forecasts}$
- **Mean Absolute Deviation**: $MAD = \frac{\sum_{\text{all forecasts}} |e_t|}{\#\ forecasts}$
- **Mean Absolute Percentage Error**: $MAPE = \frac{\sum_{\text{all forecasts}} |e_t / y_t|}{\#\ forecasts}$

Sqrt(MSEAve) = CV estimate of the prediction error

```
> test<-c(59, 0, 10, 0, 3, 0, 4, 300)
> xbar<-apply(as.matrix(IHD[,2:9]),2,mean)
> std<-apply(as.matrix(IHD[,2:9]),2,sd)
> test<-(test-xbar)/std #must standardize
> test<-matrix(test,1,8)
> K<-8
> out<-ann(as.matrix(IHD1[,2:9]),test,K,verbose=F)
> ind<-matrix(out$knnIndexDist[,1:K],nrow(test),K1)
> yhat<-mean(log10(IHD1$cost[ind]))
> ind #indices of 8 nearest neighbors
 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 278 717 778 517 37 659 114 456
> yhat
[1] 3.379489
```

PPR: Most significant terms have low p-value

```
> xbar<-apply(as.matrix(IHD[,2:9]),2,mean)
> std<-apply(as.matrix(IHD[,2:9]),2,sd)
> test<-(test-xbar)/std #must standardize

> best.iter <- gbm.perf(gbm1,method="cv");best.iter
> SSECV<-gbm1$cv.error[best.iter]*nrow(IHD1);SSECV

MISCLASS[j,]=sum(y != yhat1)/length(y)

> k=24;n=length(y) #k = prediction horizon
> Decair<-decompose(y, type = "additive")
(multiplicative for multiplicative model)
> plot(Decair,type="b")
> y_hat<-Decair$trend+Decair$seasonal
> plot(y,type="b")
> lines(y_hat,col="red")
```