STATISTICS 452/652: Statistical Learning and Prediction

November 6, 2020

# Lecture 19: Nonlinear Extensions of Classifiers

**(Reading: Section 7.7.2)**

## 1 Goals of lecture

1. We currently have methods that use specific shapes for boundaries (linear and quadratic)
   (a) Logistic regression assumes a linear model for the log-odds of each class
   (b) Discriminant analysis assumes a multivariate normal distribution on explanatory variables in each class
2. We know using wrong shapes or assumptions to model true structures can lead to bias
3. We look at extensions of logistic regression and discriminant analysis to make them more flexible

## 2 Flexible logistic regression by GAM

- We saw that logistic regression directly estimates conditional probability of $Y$ given $X$ for each class,
$$p_k(x) = P(Y = k|X = x),\ k = 1, \ldots, K$$
- Model is a linear model for baseline logit model
$$\log\left(\frac{p_k(X)}{p_K(X)}\right) = \beta_{k0} + \beta_{k1}X_1 + \ldots + \beta_{kp}X_p,\ k = 1, \ldots, K-1 \quad (1)$$
  - Results in probabilities
$$\Pr(Y = k|X) = \frac{\exp[\beta_{k0} + \beta_{k1}X_1 + \ldots + \beta_{kp}X_p]}{1 + \sum_{l=1}^{K-1}\exp[\beta_{l0} + \beta_{l1}X_1 + \ldots + \beta_{lp}X_p]},\ k = 1, \ldots, K-1$$

and

$$\Pr(Y = K|X) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp[\beta_{l0} + \beta_{l1}X_1 + \ldots + \beta_{lp}X_p]}$$

- Produces linear boundaries (can be proved mathematically)
- Estimating parameters results in classifier $\hat{f}(x)$
  - returns class $k$ if $\hat{p}_k(x)$ is the largest class probability.

## 2.1 Generalized Additive Model

- In Lecture 11 we extended linear regression to allow nonlinear shapes in each variable using a GENERALIZED ADDITIVE MODEL (GAM)
  - Linear model $f(X) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$
  - GAM replaced linear terms $\beta_j X_j$ with smoothing splines $f_j(X_j)$:

$$f(X) = \beta_0 + f_1(X_1) + f_2(X_2) + \ldots + f_p(X_p)$$

- The same thing can be done for classification by replacing linear terms in the logit model (1)with smoothing splines:

$$\log\left(\frac{p_k(X)}{p_K(X)}\right) = \beta_{k0} + f_{k1}(X_1) + f_{k2}(X_2) + \ldots + f_{kp}(X_p),\ k = 1, \ldots, K-1 \qquad (2)$$

  - For each logit, there is a different spline in $X_j$
  - It is possible to use different variables in different logits if needed
    - Maybe $X_1$ affects the probability of the first class but not the second.

- Fitting is by a version of the same backfitting algorithm adapted to the new problem
- Otherwise, using the results is the same as with linear logistic model
  - Substitute additive splines for linear model in $\exp[\ldots]/(1 + \sum_{k=1}^{K-1} \exp[\ldots])$
  - Choose class with highest probability
- Creates very flexible decision boundaries
- Does *not* allow interactions among variables in estimating probabilities
  - Potential higher bias vs. more complex machines
- It does have potential to use *lots* more DF than the linear logistic model
  - Each term may have more than 1 DF
  - Potential higher variance vs. linear models

**EXAMPLE: GAM Classification on Wheat Data (L19 GAM Wheat.R)**

The `gam()` function in `mgcv` can fit binomial models to 2-class problems easily. Just change the family argument value to `family=binomial(link=logit)`. Everything else acts the same. You have so specify variables in `s()` for splines and can add linear terms without `s()`. I think it won't fit splines if there are fewer than 8 or 9.

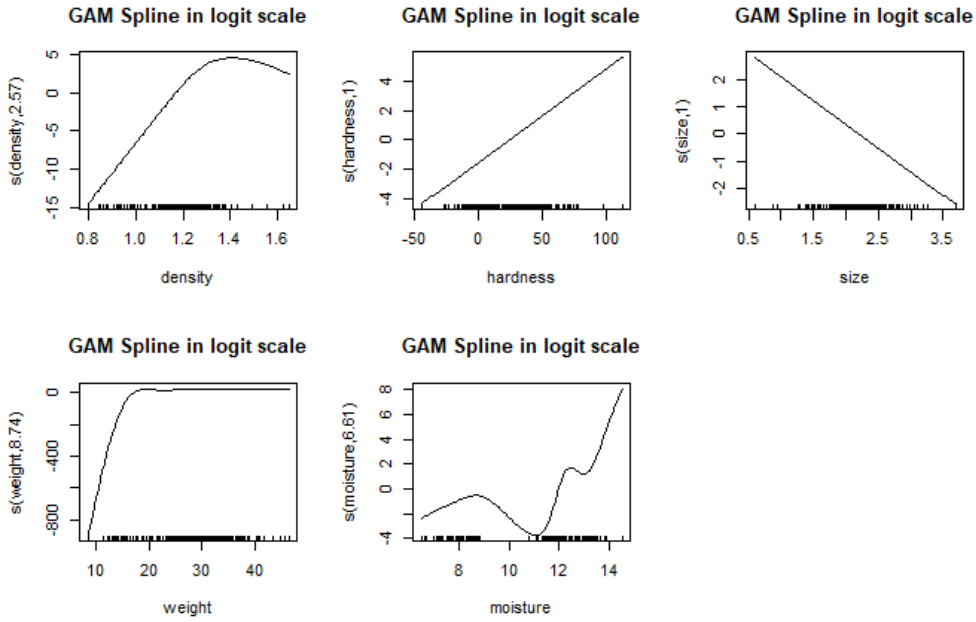
Here, I will demonstrate GAM on a special binary variable I(type=Healthy). First I create the indicators in the training and test sets. I then fit the GAM to all variables, with the binary `classnum` as a linear term and all others as splines. The results are below:

```
> Hbin1 = as.numeric(set1$type=="Healthy")
> Hbin2 = as.numeric(set2$type=="Healthy")
>
> library(mgcv)
> # All variables
> gam2 <- gam(data=set1, Hbin1~s(density) + s(hardness) +
+             s(size) + s(weight) + s(moisture) + classnum,
+             family=binomial(link=logit))
> summary(gam2)

Family: binomial
Link function: logit

Formula:
Hbin1 ~ s(density) + s(hardness) + s(size) + s(weight) + s(moisture) +
    classnum

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -22.5866    14.1264  -1.599    0.110
classnum     -0.5659     0.9193  -0.616    0.538

Approximate significance of smooth terms:
              edf Ref.df Chi.sq  p-value
s(density)  2.566  3.181 31.048 3.49e-06 ***
s(hardness) 1.000  1.000  9.596  0.00195 **
s(size)     1.000  1.000  3.274  0.07041 .
s(weight)   8.744  8.929 18.573  0.02974 *
s(moisture) 6.613  7.628 17.751  0.01894 *
```

Once again `density` is most important, although `weight` seems less useful than it has been in the past. Plots of the splines in each variable are shown in Figure (1).

We can also fit multinomial models using `gam()`. They can be a pain to work with, because

Figure 1: Example of conditional distributions of $X$ for $K = 2$ classes. When each class is equally likely, Bayes rule says to classify according to highest curve. (From ISLR.)




1. The response must be written as numerical $0, 1, \ldots, K-1$ (so for our problem as 0, 1, 2). Class 0 is the baseline.

2. The `formula` must be written as `list(y~ MODEL1, ~MODEL2, ~MODEL3)`, where `y` is replaced by the response variable name, and `MODELk` is replaced with the formula for the $k$th logit model. You can have a different model for each logit.

3. The family is `multinom(K=...)` where the "..." is replaced by the number of logits, $K-1$

The fit to the Wheat data is below:

```
> levels(set1$type)
[1] "Healthy" "Scab"    "Sprout"
> set1$type0 <- as.numeric(set1$type) - 1
> # Healthy will be our baseline class
>
> # Fit full model, all variables in each logit
> gam.m <- gam(data=set1, list(type0
+   ~s(density) + s(hardness) + s(size)
+   + s(weight) + s(moisture) + classnum,
+   ~s(density) + s(hardness) + s(size)
+   + s(weight) + s(moisture) + classnum),
+             family=multinom(K=2))
> summary(gam.m)
```

```
Family: multinom
Link function:

Formula:
type0 ~ s(density) + s(hardness) + s(size) + s(weight) + s(moisture) +
    classnum
~s(density) + s(hardness) + s(size) + s(weight) + s(moisture) +
    classnum

Parametric coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.7984     1.4003  -0.570    0.569
classnum        0.4480     0.8924   0.502    0.616
(Intercept).1   0.4669     1.0362   0.451    0.652
classnum.1      0.3078     0.6868   0.448    0.654

Approximate significance of smooth terms:
                edf Ref.df Chi.sq  p-value
s(density)    3.805  4.729 46.281  < 2e-16 ***
s(hardness)   1.000  1.000  3.049   0.0808 .
s(size)       1.000  1.000  5.216   0.0224 *
s(weight)     1.000  1.000 20.017 8.31e-06 ***
s(moisture)   2.745  3.414  5.277   0.1878
s.1(density)  1.000  1.000 28.035  < 2e-16 ***
s.1(hardness) 1.649  2.068  8.473   0.0160 *
s.1(size)     1.000  1.000  1.062   0.3028
s.1(weight)   1.000  1.000  0.071   0.7897
s.1(moisture) 2.238  2.773  6.806   0.0723 .


> pred.prob.m <- predict(gam.m, newdata=set1, type="response")
> pred.class.m <- apply(pred.prob.m,1,function(x)
                        which(max(x)==x)[1])-1
>
> head(cbind(round(pred.prob.m, digits=3), pred.class.m))
                    pred.class.m
1 0.937 0.040 0.023            0
2 0.898 0.003 0.099            0
4 0.971 0.001 0.029            0
5 0.728 0.091 0.181            0
6 0.922 0.001 0.076            0
7 0.865 0.002 0.133            0
>
> pred.prob.2m <- predict(gam.m, newdata=set2, type="response")
```

```
> pred.class.2m <- apply(pred.prob.2m,1,function(x)
        which(max(x)==x)[1])-1
>
> # Confusion Matrix
> table(set2$type, pred.class.2m,  dnn=c("Observed","Predicted"))
          Predicted
Observed    1  2  3
  Healthy  17  2  6
  Scab      4 14  4
  Sprout    6  6 16
```

We see again that Density is important for distinguishing either `Scab` or `Sprout` from `Healthy`. The estimated probabilities for the first 6 observations show that all are predicted to be Healthy. The confusion matrix is similar to other methods, with a slightly smaller count of mistakes. The training and test error are added to the table below.

| Method | Training | Test | Test "SE" |
|---|---|---|---|
| KNN-Opt(12) | 0.28 | 0.49 | 0.06 |
| Logistic | 0.28 | 0.40 | |
| Logist-LASSO-min | 0.28 | 0.43 | |
| LDA | 0.28 | 0.39 | |
| QDA | 0.26 | 0.43 | |
| GAM | 0.24 | 0.37 | |

# 3 Flexible Discriminant Analysis with Naive Bayes

- Discriminant analysis assumes that the explanatory variables within each class are normally distributed (MVN)
  - Also assumes some overall "prior" probability of each class across the whole population (e.g. sample proportions if sample is random)
- Combines this using "Bayes' rule" to estimate $P(Y = k|X)$ for each $k$
- LDA assumes that variances and correlations of the $K$ MVN distributions are the same, but means may differ
  - Estimate $K + p(p+1)/2$ parameters
  - Linear boundaries
- QDA allows each MVN distribution to have its own mean and variance
  - Estimate *additional* $(K-1)p(p+1)/2$ parameters
    - Higher variance than LDA
  - Quadratic decision boundaries

        * Potentially lower bias than LDA
        * Still specific shape, though.

- Assumption of normality is pretty narrow
    - Nothing is *exactly* normally distributed
    - Categoricals/indicators!

## 3.1 Naive Bayes (not in book)

- We don't *have* to assume a normal distribution for all $X_j$
- There are ways to estimate the "density" (=population histogram) of a population from a sample.
    - Kernel density estimation is very similar to LOESS and other local regression methods
    - Gives a smoothed version of a histogram without using a specific probability distribution
    - Need to specify how much smoothing—a tuning parameter
- Unfortunately, kernel density estimation only works well in one dimension
    - Like splines, not enough data in $p$-dimensions to support reliable estimate
- Naive Bayes classifier makes the assumption that the explanatory variables are all independent of one another, and uses estimated densities in the discriminant analysis calculations
    - Creates boundaries of very flexible shapes
    - But independence assumption is almost certainly absurd!
        * Is it any more absurd that assuming normality on binary explanatories???
- Main difficulty (in my opinion) is in finding the right amount of smoothness in each variable's density estimate
    - Too smooth, and you barely move it from normal (potential bias)
    - Too wiggly, and the boundaries are unnecessarily wiggly (potential variance)
- Some versions of naive Bayes actually assume normality *and* independence!
    - This still creates linear boundaries, but does not adapt them properly to the density shape when there is strong correlation
    - Works well when both assumptions are (nearly) satisfied
    - Another bias-variance tradeoff

- When explanatory variables are actually correlated, it *may* be helpful to preprocess the data by running a principal components analysis before the naive Bayes classifier.
  - PC's $Z_1, Z_2, \ldots, Z_p$ are uncorrelated but contain all of the same information as $X_1, X_2, \ldots, X_p$.
  - Maintaining all $p$ components just causes the classifier to be computed in a rotated version of the original data.
  - This sometimes improves the discrimination of densities across classes, and the resulting class predictions, but not always.

**EXAMPLE: Naive Bayes Classification on Wheat Data (L19 GAM Wheat.R)**

`NaiveBayes()` in the package `klaR` can be run with or without a kernel density estimate (`usekernel=TRUE` or `FALSE`). The `plot()` function for its objects produces plots of the estimated densities across classes for each explanatory variable.

I run this function four different ways, thinking of "density" (kernel or normal) and "preprocess" (none or PC) as two different 2-level tuning parameters. Plots of the normal and kernel density estimates on the original variables are shown in Figure 2.

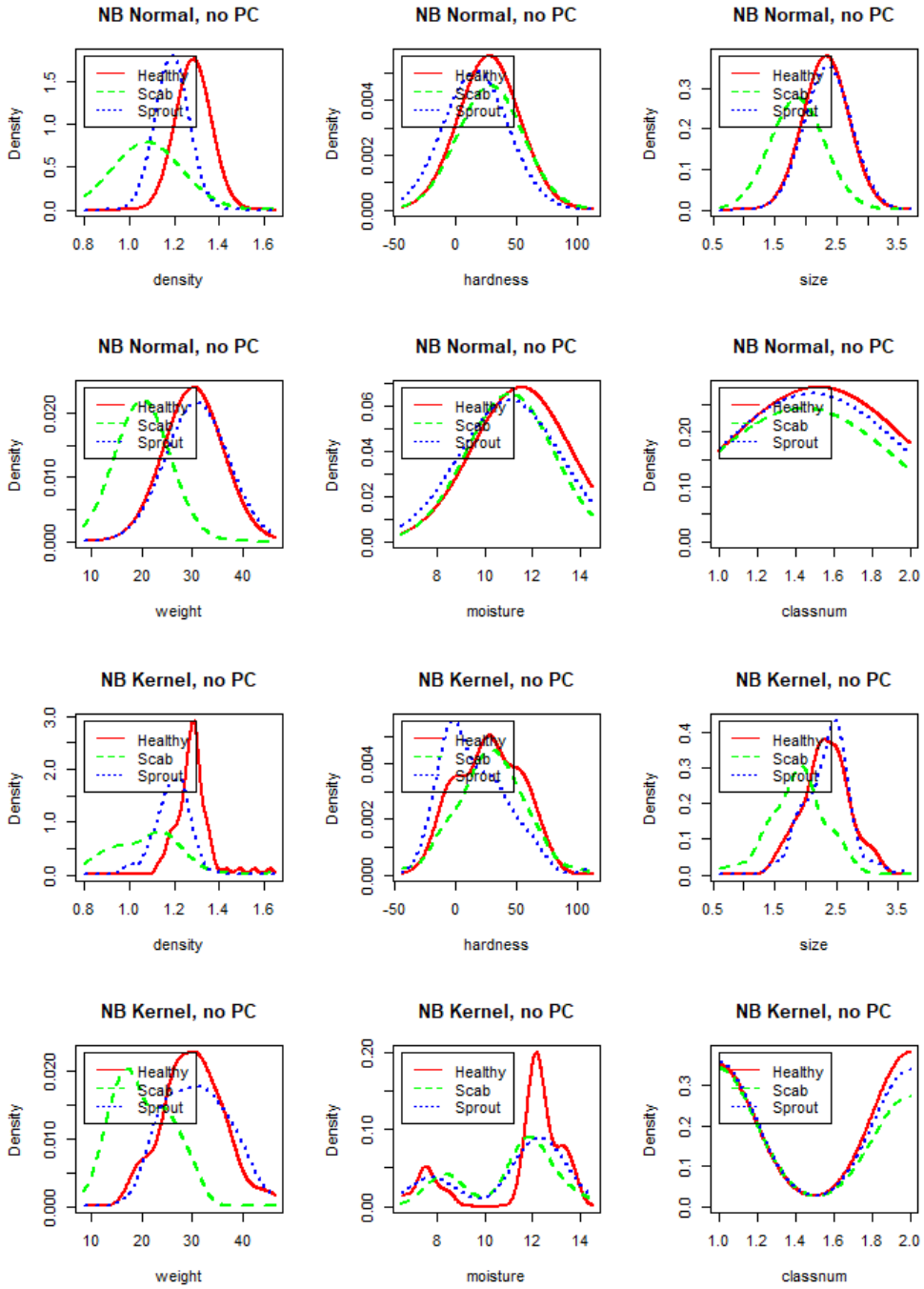
We see that `moisture` and `classnum` (the binary) have particularly non-normal densities as estimates with the kernel method. Other variables' densities seem reasonably mound-shaped for each class. The training and test errors for each of the four methods are in the table below.

| Method | Training | Test | Test "SE" |
|---|---|---|---|
| KNN-Opt(12) | 0.28 | 0.49 | 0.06 |
| Logistic | 0.28 | 0.40 | |
| Logist-LASSO-min | 0.28 | 0.43 | |
| LDA | 0.28 | 0.39 | |
| QDA | 0.26 | 0.43 | |
| GAM | 0.24 | 0.37 | |
| NB-normal | 0.32 | 0.39 | |
| NB-kernel | 0.23 | 0.43 | |
| NB-pc-normal | 0.30 | 0.47 | |
| NB-pc-kernel | 0.23 | 0.50 | |

The biggest thing to notice is that PC preprocessing does not help this time, producing error rated noticeably higher than the best methods. The normal density version performs at least as well as the kernel version. This is likely a bias-variance tradeoff. The two variables for which the normal assumption is violated the worst have estimated densities that are not that different from one another, particularly in terms of shifting the means. Each of the other fours variables seems to shift the means among the three classes more than these two. So failing to estimate their distributions properly costs us little in terms of boas compared to the kernel estimate, but we gain by having a simpler model.

---

Figure 2: Estimated densities for each variable in wheat data from Naive Bayes analysis. Top 6 plots use normal density; bottom 6 use kernel density estimate.



# 4 What to take away from this

1. There are several well-known methods that are non-linear extensions of popular linear classifiers.

2. Each has a role to play in a toolkit, but neither is as completely flexible as other methods we will try later.

# 5 Exercises

## Application

**Return to the Vehicle data used in the previous lecture. Use the same split as before.**

```
set.seed(46685326, kind="Mersenne-Twister")
perm <- sample(x=nrow(vehdata))
set1 <- vehdata[which(perm <= 3*nrow(vehdata)/4), ]
set2 <- vehdata[which(perm > 3*nrow(vehdata)/4), ]
```

I already know that GAM will not fit these data. It uses enough degrees of freedom that it creates a complete separation, and this ruins the logit model.

Instead, we will focus on Naive Bayes. You will fit the same four models that we used in the example: with/without kernel, and with/without PC preprocessing.

1. Run the version with kernel density estimation on the original variables first
   (a) **Present plots of each variable's density separated by classes.**
   (b) Look at the plot:
      i. Do many of the variables look like they have very skewed, multimodal, or otherwise non-normal distributions across the classes? If so, **name any that seem pretty non-normal (but no more than three), and mention a word or two about what non-normal feature(s) each one has**.
      ii. Do any of the variables look like they do a very good job of discriminating among classes, particularly by separating their means? If so, **name any that seem to separate the classes well (but no more than your top 3), and mention which class(es) they seem to separate.**

   In these questions, I mainly want to make sure you are absorbing the information correctly and understanding what to look for. I do not have a specific three variables that are "right" and consider all the rest "wrong". Judgment may vary, but the decisions should be made rationally.

2. Compute training and test error of all four versions of the algorithm.
   (a) **Report them in the order**
      i. No PC, Kernel
      ii. No PC, Normal
      iii. PC, Kernel
      iv. PC, Normal

(b) **Comment on how the test errors compare to each other. Do PC or kernel density estimation seem to help?**

(c) **How does test error compare to other methods?**