Applied Machine Learning - Extra Slides

Max Kuhn (RStudio)

Extra Feature Engineering Slides

Principal Component Analysis

More Recipe Steps

The package has a rich set of steps that can be used including transformations, filters, variable creation and removal, dimension reduction procedures, imputation, and others.

For example, in the previous bivariate data problem, the Box-Cox transformation was conducted using:

```
bivariate_rec <- recipe(Class ~ ., data = bivariate_data_train) %>%
  step_BoxCox(all_predictors())

bivariate_rec <- prep(bivariate_rec, training = bivariate_data_train, verbose = FALSE)

inverse_test_data <- bake(bivariate_rec, newdata = bivariate_data_test)
```

Correlated Predictors

In these data there are potential clusters of *highly correlated variables*:

- proxies for size: `Lot_Area`, `Gr_Liv_Area`, `First_Flr_SF`, `Bsmt_Unf_SF`, `Full_Bath` etc.
- quality fields: `Overall_Qual`, `Garage_Qual`, `Kitchen_Qual`, `Exter_Qual`, etc.

It would be nice if we could combine/amalgamate the variables in these clusters into a single variable that represents them.

Another way of putting this is that we would like to create artificial features of the data that account for a certain amount of variation in the data.

There are a few different methods that can accomplish this; we will focus on principal component analysis (PCA) as a solution.

PCA Signal Extraction

Principal component analysis (PCA) is a multivariate statistical technique that can be used to create artificial new variables from an existing set.

The new variables are created to account for the most variation in the data. In this case, "variation" means correlation.

Conceptually, PCA determines which variables account for the most correlation in the data and creates a new variable that is a linear combination of all the predictors.

- This is called the *first principal component* (aka `PC1`)
- This linear combination emphasizes the variables that are the most correlated.

The information that PC #1 represents is then *removed from the data*.

The second PCA component is the linear combination that accounts for the most left-over correlation in the data (and so on).

PCA Signal Extraction

To recap:

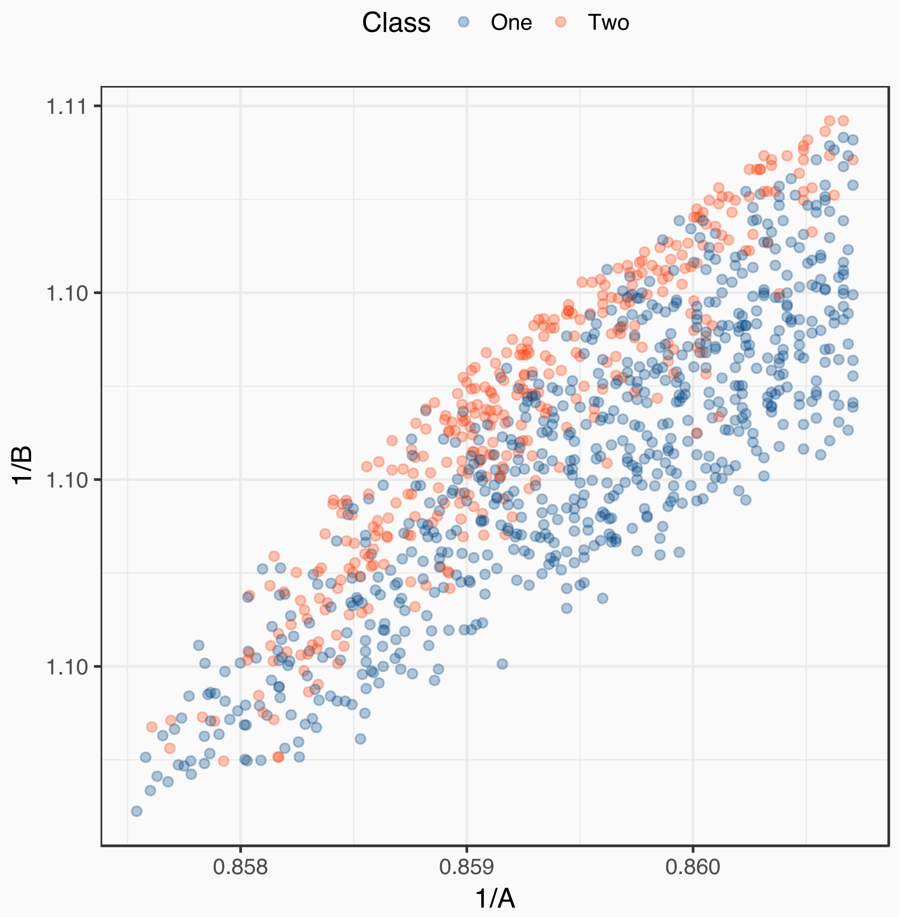
- The components account for as much as the variation in the original data as possible.
- Each component is uncorrelated with the others.
- The new variables are *linear combinations* of all of the input variables and are effectively unitless.

For our purposes, we would use PCA on the *predictors* to

- Reduce the number of variables exposed to the model (but this is not feature selection).
- Combat excessive correlations between the predictors (aka multicollinearity).

In this way, the procedure is often called *signal extraction* but this is poorly names since there is no guarantee that the new variables will have an association with the outcome.

Back to the Bivariate Example - Transformed Data



Back to the Bivariate Example - Recipes

We can build on our transformed data recipe and add normalization:

```r
bivariate_pca <-
  recipe(Class ~ PredictorA + PredictorB, data = bivariate_data_train) %>%
  step_BoxCox(all_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_pca(all_predictors()) %>%
  prep(training = bivariate_data_test, verbose = FALSE)

pca_test <- bake(bivariate_pca, newdata = bivariate_data_test)

# Put components axes on the same range
pca_rng <- extendrange(c(pca_test$PC1, pca_test$PC2))

ggplot(pca_test, aes(x = PC1, y = PC2, color = Class)) +
  geom_point(alpha = .2, cex = 1.5) +
  theme(legend.position = "top") +
  scale_colour_calc() +
  xlim(pca_rng) + ylim(pca_rng) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```

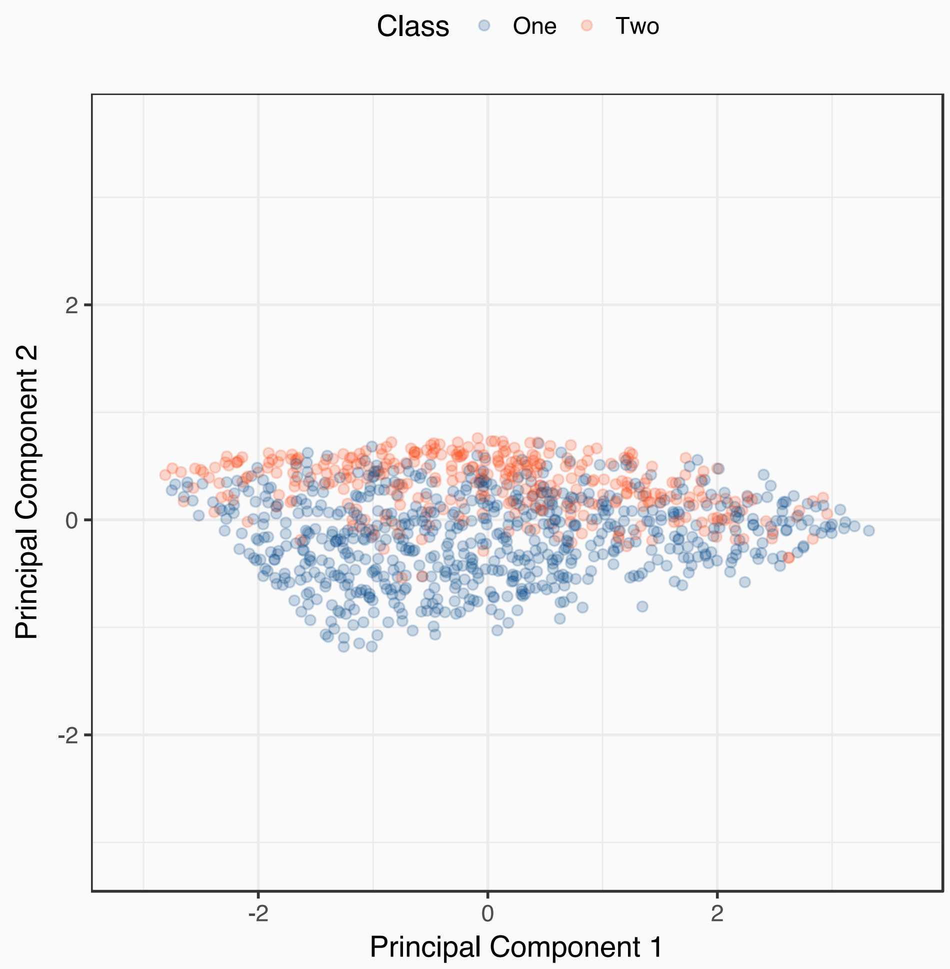
Back to the Bivariate Example

Recall that even after the Box-Cox transformation was applied to our previous example, there was still a high degree of correlation between the predictors.

After the transformation, the predictors were centered and scaled, then PCA was conducted. The plot on the right shows the results.

Since these two predictors are highly correlated, the first component captures 91.2% of the variation in the original data. However...

...recall that PCA has not guarantee that the components are associated with the outcome. In this example, the *least important* component has the association with the outcome.



Graphical Checks for Predictors

Graphical Checks for Predictors

We have 10 variations of the same model. We can look at their results to try to understand if there are any irrelevant predictors.

We can get the `lm` summary table using `broom`'s `tidy` function on each object and binding the results together:

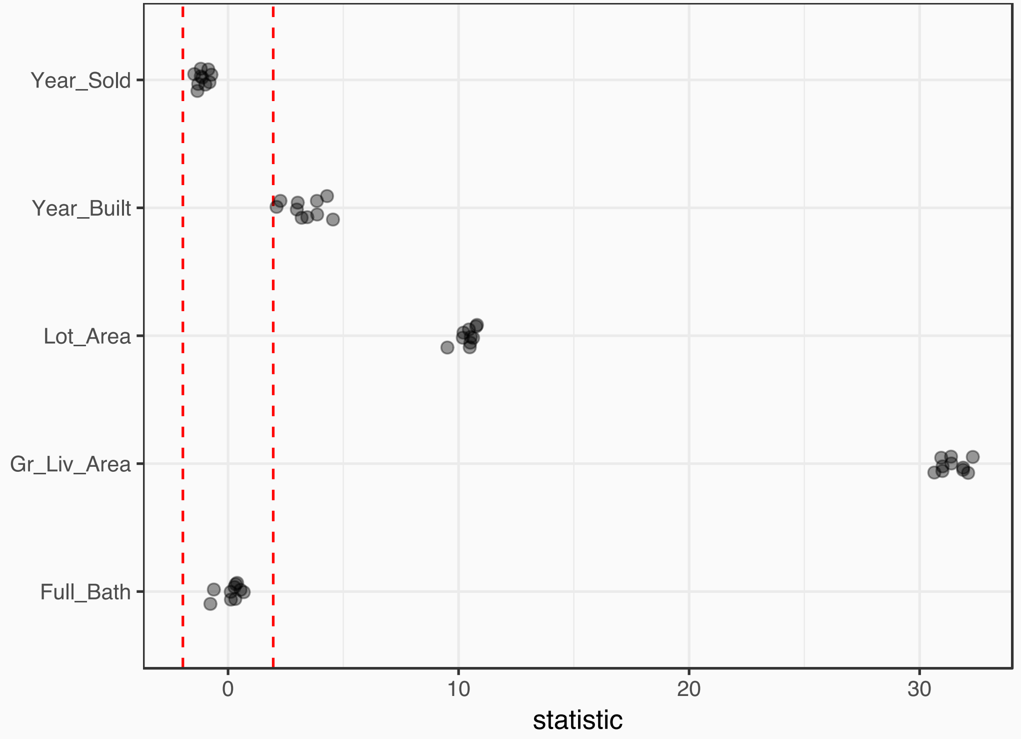
```
coef_summary <- map(cv_splits$fits, tidy) %>% bind_rows
head(coef_summary)
```

```
##          term estimate std.error statistic   p.value
## 1 (Intercept)  3.13271  2.866604     1.093  2.75e-01
## 2  Year_Built  0.00150  0.000349     4.294  1.84e-05
## 3 Gr_Liv_Area  0.25974  0.008146    31.886 7.18e-180
## 4   Full_Bath -0.00304  0.004930    -0.617  5.37e-01
## 5   Year_Sold -0.00163  0.001379    -1.183  2.37e-01
## 6    Lot_Area  0.02292  0.002194    10.449  6.68e-25
```

The `statistic` column corresponds to a 1 degree of freedom *t*-test of the parameter. The residual `df` are fairly high (about 1948) so we can compare these to quantiles of a standard normal distribution.

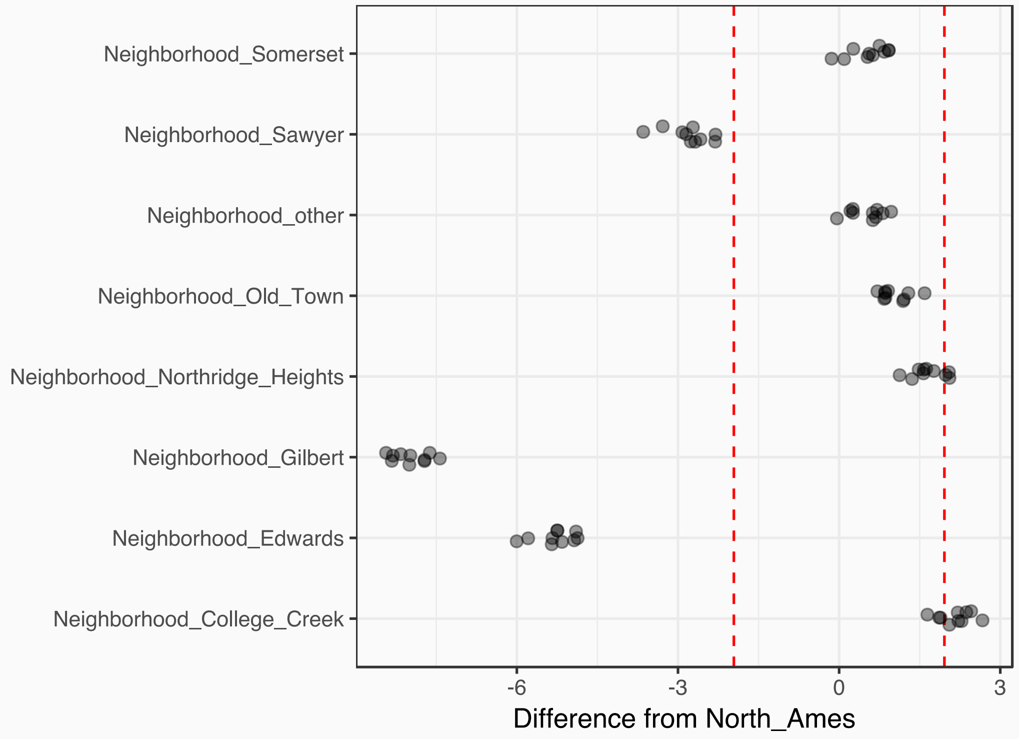
Some Continuous Predictors

```r
num_pred <- c("Year_Built", "Gr_Liv_Area",
              "Full_Bath", "Year_Sold",
              "Lot_Area")
z_lim <- qnorm(c(0.025, 0.975))
coef_summary %>%
  filter(term %in% num_pred) %>%
  ggplot(aes(x = term, y = statistic)) +
  geom_hline(
    yintercept = z_lim,
    lty = 2,
    col = "red"
  ) +
  geom_jitter(
    width = .1,
    alpha = .4,
    cex = 2
  ) +
  coord_flip() +
  xlab("")
```



Neighborhood

```r
coef_summary %>%
  filter(grepl("^Neighborhood", term)) %>%
  ggplot(aes(x = term, y = statistic)) +
  geom_hline(
    yintercept = z_lim,
    lty = 2,
    col = "red"
  ) +
  geom_jitter(
    width = .1,
    alpha = .4,
    cex = 2
  ) +
  coord_flip() +
  xlab("") +
  ylab(
    paste("Difference from",
          levels(ames_test$Neighborhood)[1])
  )
```



Extra Regression Analysis Slides

Measuring Performance for Numeric Outcomes

Hands-On: Illustrative Example

`yardstick` contains the test set results in a data frame called `solubility_test`:

```r
library(yardstick)
data(solubility_test)
str(solubility_test, nchar.max = 70)
```

```
## 'data.frame':    316 obs. of  2 variables:
##  $ solubility: num  0.93 0.85 0.81 0.74 0.61 0.58 0.57 0.56 0.52 0.45 ...
##  $ prediction: num  0.368 -0.15 -0.505 0.54 -0.479 ...
```

Take 5 minutes and graphically assess the data set.

How well do you think that the model worked?

Measuring Performance

The manner in which we measure the model's efficacy can make a big difference in how the model is tuned and viewed. There are a few different options for summarizing performance, including:

Root mean squared error (RMSE) the model residuals are computed and squared. They are then averaged and the square-root is used to put the value back into the original units.

```r
rmse(solubility_test, truth = solubility, estimate = prediction)
```

```
## [1] 0.722
```

RMSE is a good metric since it is in the original units and a direct measure of model *accuracy*.

Measuring Performance - R^2

The coefficient of determination (aka R^2) is commonly used to evaluate models. In the case of linear regression, R^2 is the proportion of variation in the data that can be explained by the model.

Computationally, the most direct method of estimation is to compute the correlation between the observed and predicted values (i.e. R) and square it.

```
rsq(solubility_test, truth = solubility, estimate = prediction)
```

```
## [1] 0.879
```

R^2 can be problematic for a few reasons

- It is a measure of correlation, not accuracy.
- It can be artificially inflated when the variation in the outcome is large.

Measuring Performance - R^2

There are more traditional estimates but behave poorly when the model is ineffective.

```r
library(dplyr)
set.seed(3545)
solubility_test <- solubility_test %>% mutate(noise = sample(prediction))

rsq(solubility_test, truth = solubility, estimate = noise)
```

```
## [1] 0.00449
```

```r
rsq_trad(solubility_test, truth = solubility, estimate = noise)
```

```
## [1] -0.792
```

Both RMSE and R^2 can be sensitive to outliers since they involved squared terms. This may be good or bad, depending on the problem.

Measuring Performance - Other Approaches

- In the case where new data points are being ranked (i.e. the most lucrative customer), using *Spearman's rank correlation* would be a good choice.

- Robust measures, such as the mean absolute error (MAE) can also be used.

- When the outcome is highly skewed, the average *fold-difference* between the observed and predicted values might be helpful.

- Try to avoid any discretization procedures such as the percentage of samples that are more than 2-fold wrong.

- The best metrics are those that are the most relevant. These may be complex equations that involve other estimates, such as
    - customer lifetime value
    - expected return on an investment

Diagnosing the Model

How should we graphically evaluate the model without using the test set?

The assessment sets from resampling can be used to compute and plot the residuals. *Partial residual plots* can be created where the residuals are plotted against different (potential) predictors to diagnose predictors that should be included in the model.

In the case of repeated cross-validation (and others), the sample training set sample might be included in multiple assessment sets. In this case, the predictions can be averaged and these means can be used in visualizations.