Machine Learning Notes

Contents

Error rates

Reminder on errors. Consider (again) a test used to predict whether or not someone has a disease.

		disease	
		Present	Absent
test	positive	TP	FP
	negative	FN	TN

- Sensitivity is the probability to test positive in the prescence of the disease
  - P(positive test| disease) = $TP/(TP + FN)$
- Specificity is the probability to test negative in the absence of the disease
  - P(negative test| no disease) = $TN/(TN + FP)$
- Positive predictive value is the probability of having the disease given a positive test
  - P(disease|positive test) = $TP/(TP + FP)$
- Negative predictive value is the probability of not having the disease given a negative test
  - P(no disease|negative test) = $TN/(TN + FN)$
- Accuracy is the probability of the test giving the correct result
  - P = $(TN + TP)/(TP + FP + TN + FN)$

In sample errors

Even after a model is trained on a given dataset, it probably won't *perfectly* predict the outcome for each observation (it shouldn't, as this would most likely result in overfitting). In sample errors are the errors that occur when re-running a model on the sample used to train it.

Out of sample errors

Out of sample errors are the errors that occur when the model is applied to new data, i.e. data that the model was not trained on. These are generally higher than the in sample error.

Overfitting

There is noise in our training data. We *want* to fit to the underlying mechanic/model that gives rise to our data. This is not known, and has to be estimated. However, our sample data contains noise. We want to fit to the model, not to the noise. We might optimise our model such that the in sample (training) errors are small, but this "overfits" our model to the noise present in the training sample. When used in practice, this could cause our model to perform poorly on real world data (which consists of a different set of observations). A good model should capture the signal (features) of a dataset, but not be too tightly tuned to the noise. Overfitting is when the model is too heavily influenced by the training noise, and can result in large out of sample errors when the model is applied to a test/validation dataset.

Receiver Operating Characteristic curves

When we are trying to predict the classification of something, generally our model will output a continuous probability, and we will classify as true if $p \geq 0.5$ or false otherwise. How do we choose the cut value/threshold on our model's probability/likelihood?

An ROC curve plots the probability of a true positive (sensitivity) vs the probability of a false positive (1 - specificity). At low values of our threshold, then we are very likely to accept anything: we have a high probability of correctly identifying true positives, but also a high probability of accepting false positives. Our ROC curves thus approach (1.0,1.0). Conversely, at high thresholds we do not accept anything, so our ROC curve approaches (0,0). As we increase our threshold, our model should accept more true positives than false positives, and so the curve should lie above the line p(TP) = p(FP). The performance of our model can be described by the area under the ROC curve, larger areas correspond to better performance (we're getting more true positives per false positive). Note that a performance of 0.5 is the same as just random guessing (coin flipping). A value of 0.8 is generally considered "good".

Cross Validation

Don't want to use the test data when training our model. We should use the test data only once, at the end. Would like to get an idea of how our model would perform on the test data (or the real world) though. In cross validation, we take our train data, and split *this* into train and test subsets. We train our model on the training set, then evaluate it on the test set. Then we repeat, resampling the original training set to obtain a different "train" and "test" set. After multiple repetitions, we average the performance of our model to get an idea of how it will perform on the actual test set (or in TRW). This is a useful method to:

- pick variables/features
- determine the type of prediction function to use
- determine the parameters of our prediction function
- compare different predictors

There are different ways of doing this resampling. We can randomly resample (pretty basic). In K fold resampling, we divide our data into k parts. Each part is used as the test set exactly one time, and is part of the training set the other (k-1) times. "leave one out" resampling is essentially k fold resampling, with k equal to the number of observations.

bootstrapping 0.632

Reduces bias from bootstrapping somehow? math prolly

Caret

Caret is a wrapper around a bunch of different machine learning packages. It allows many different machine learning packages to be utilised through the same interface. Some documentation on caret is here

```
library(caret)
library(ISLR)
data(Wage)

summary(Wage)
```

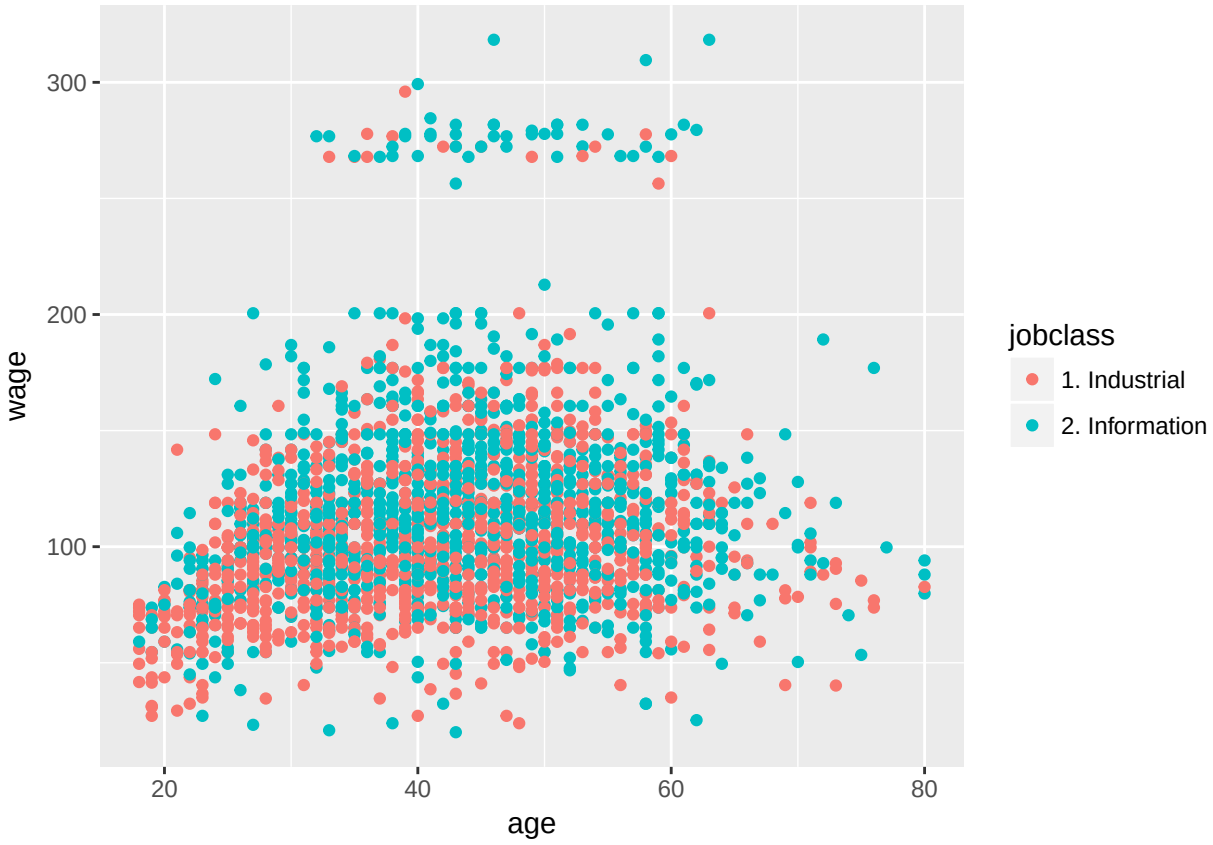
```
##       year           age               sex                    maritl
##  Min.   :2003   Min.   :18.00   1. Male  :3000   1. Never Married: 648
##  1st Qu.:2004   1st Qu.:33.75   2. Female:   0   2. Married      :2074
##  Median :2006   Median :42.00                    3. Widowed      :  19
##  Mean   :2006   Mean   :42.41                    4. Divorced     : 204
##  3rd Qu.:2008   3rd Qu.:51.00                    5. Separated    :  55
##  Max.   :2009   Max.   :80.00
##
##        race                   education                     region
##  1. White:2480   1. < HS Grad      :268   2. Middle Atlantic   :3000
##  2. Black: 293   2. HS Grad        :971   1. New England       :   0
##  3. Asian: 190   3. Some College   :650   3. East North Central:   0
##  4. Other:  37   4. College Grad   :685   4. West North Central:   0
##                  5. Advanced Degree:426   5. South Atlantic    :   0
##                                           6. East South Central:   0
##                                           (Other)              :   0
##            jobclass               health      health_ins      logwage
##  1. Industrial :1544   1. <=Good     : 858   1. Yes:2083   Min.   :3.000
##  2. Information:1456   2. >=Very Good:2142   2. No : 917   1st Qu.:4.447
##                                                            Median :4.653
##                                                            Mean   :4.654
##                                                            3rd Qu.:4.857
##                                                            Max.   :5.763
##
##       wage
##  Min.   : 20.09
##  1st Qu.: 85.38
##  Median :104.92
##  Mean   :111.70
##  3rd Qu.:128.68
##  Max.   :318.34
##
```

split the data. Want to predict `Wage$wage`

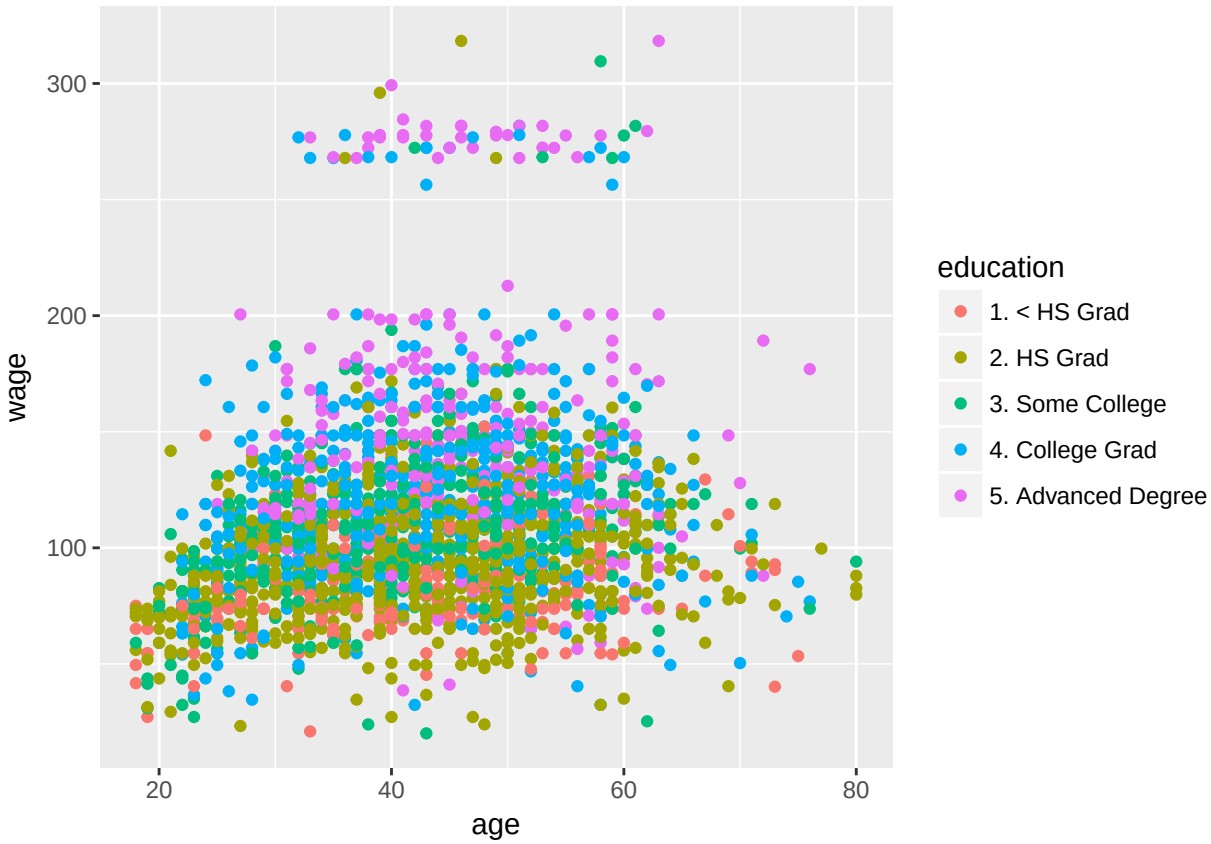
```
set.seed(5074491)
inTrain<- createDataPartition(y=Wage$wage,p=0.80,list=FALSE)
wageTrain<-Wage[inTrain,]
wageTest<-Wage[-inTrain,]
```

some plots

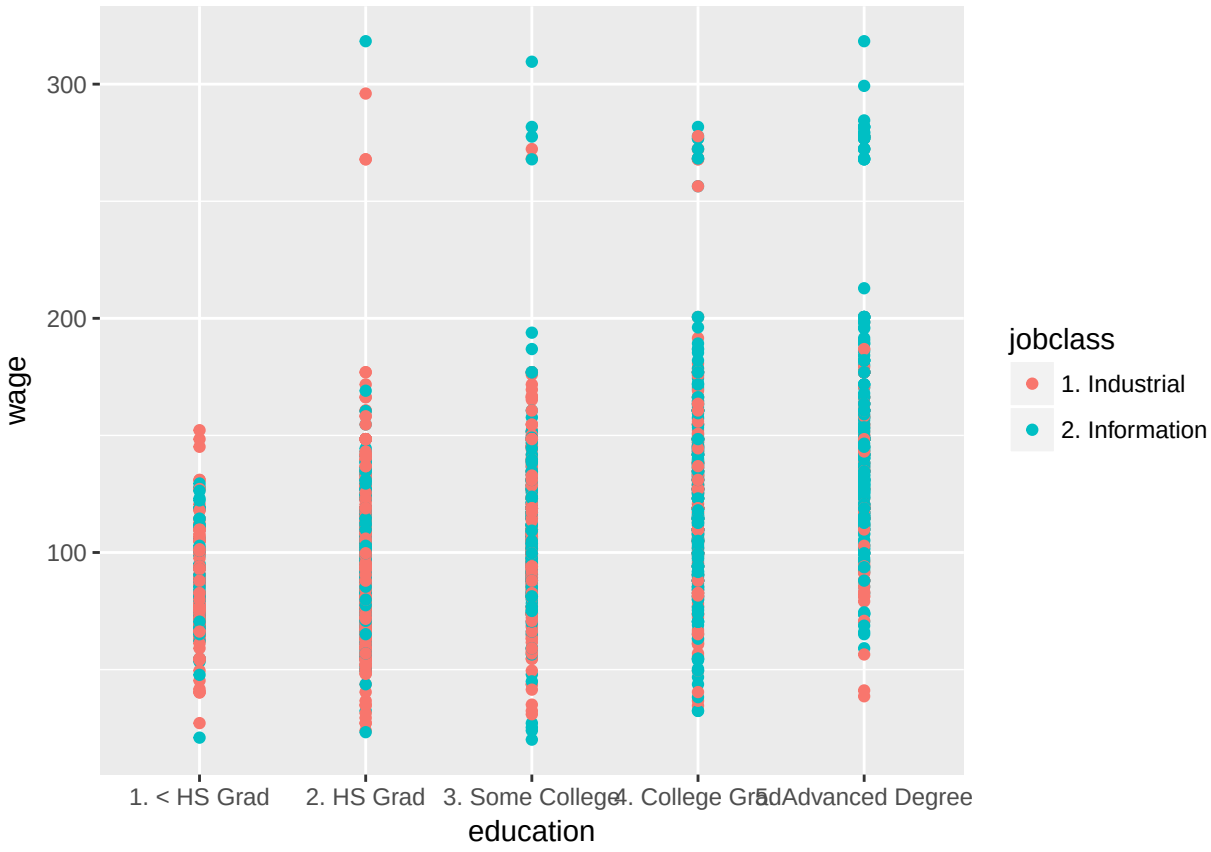
```
g<-ggplot(data=wageTrain,aes(y=wage,x=age,col=jobclass)) + geom_point()
print(g)
```




```
g<-ggplot(data=wageTrain,aes(y=wage,x=age,col=education)) + geom_point()
print(g)
```




```
g<-ggplot(data=wageTrain,aes(y=wage,x=education,col=jobclass)) + geom_point()
print(g)
```


300
200
100
wage
1. < HS Grad
2. HS Grad
3. Some College
4. College Grad
5. Advanced Degree
education
jobclass
1. Industrial
2. Information