STA414/2104: Week 9

Validating Predictive Models

Alex Stringer

March 13th, 2018

Recap

We have spent the first 8 weeks of the course talking about

- Theory of probabilistic modelling
- How to build probabilistic models, *mathematically*

We haven't yet talked about how to implement models, in practice, on real data.

“Real Data”

It’s not possible to give you a satisfying overview of how to do this, because

- It’s really **hard**. At work I spent 90% of my time, easily, just cleaning, validating the integrity of, and *finding* data. The only way to learn it is by doing it.
- It’s really **boring**. Plain and simple. It can be very satisfying to get right, especially if you are working with data that you are actually interested in. But preprocessing in general is boring and tedious, in my experience.

"Real Data"

- It's really **broad**. We can talk about loading one kind of image data, but what if you have other structures or formats? What about other types, like text, sound, video? There are entire books and series of courses that talk just about, for example, feature engineering for text data.

It is, though, very important, and often when you read about things like kaggle competitions, the way they are won is through really detailed, time-consuming, and problem-specific feature engineering and preprocessing:
http://blog.kaggle.com/category/winners-interviews/

Validating Models

The other nagging practical question when applying all this theory to data is: once I have fit a model, what do I *do*?

You're not just going to try one model, you're going to try many. How to compare, and choose a final model?

Validating Models

There are two kinds of comparison between models: **nested** and **unnested**.

- **Nested** models are models where one can be viewed as a special case of the other. *Feature selection* is an example of nested model selection: model A has d features, and model B has those d features plus one (or more) extra features; you need to choose whether model A or B is a better fit to your data, which corresponds to choosing whether the features present in B but not A are predictive or not
- **Unnested** model selection refers to any other type of comparison: models from different classes (neural net vs regression, say); models from the same class with different tuning/hyperparameters; models with non-nested subsets of features

Validating Models

There are two kinds of validation that can be performed: assessing how well the model fits the *training data* (given complexity), and assessing out-of-sample predictive performance, i.e. how the model is likely to perform on new samples

- To assess the training set fit, we use some penalized version of the loss function that accounts for complexity, e.g. AIC or BIC. We can use training set fit to choose between models of the same class, i.e. feature selection and choosing hyperparameters
- To assess out-of-sample predictive performance, we typically use metrics that know nothing about the model, and only take in y and $\hat{y}$- for example the ROC/AUC, KS, or decile plots

Validating Models

. . . and finally, there are two kinds of response variables we can have, **continuous** and **categorical**.

- For validating continuous predictions, we typically look at some kind of a distance metric between the target and fitted vectors, and/or correlation between targets and predictions
- For validating categorical predictions, we can look at
  - The error rate; how many right/wrong predictions we make
  - Metrics which assess the strength of the prediction by comparing the targets to the soft classifications

Nested Models

First, let's look at comparing nested models. We will talk about feature selection, but this discussion extends to any models which can be described as nested.

Suppose we have a feature space $S = \{j : x_j \text{ is a feature }, j = 1 \dots p\}$, and we wish to evaluate two subsets of features, $B \subset A \subseteq S$.

We build models M_A and M_B. How to decide?

Nested Models

There are some important theoretical models we might uncover.

- A **saturated** model contains one parameter for each observation. For example, for Gaussian data, a saturated regression model sets $\hat{y}_i = y_i$, i.e. draws a line through the points. For binomial data (n_i, y_i), a saturated model sets $\hat{y}_i = y_i/n_i$
- A **null** model is the opposite; it has only one parameter, the intercept. A null model sets $\hat{y}_i = \bar{y}$

Nested Models

There are some important theoretical models we might uncover.

- The **true** model refers to the actual underlying structural relationship between features and targets that generated our observed data- so, the right class of models, and the right features. Note that if we actually got the true model, we wouldn't recover our training set perfectly (that's what the *saturated* model does)- this is the best *model* we can hope for, as if we find this, then all the errors we make are due to random noise
- A **correct** model is a model that is nested in the true model.

Nested Models

We note a few immediate properties

- Neither the saturated nor the null models are really models
- The saturated model has 0 loss, for any *reasonable loss function*, so by definition it is the best we can do on the training data
- The null model should have the highest loss of any *reasonable model*
    - "Reasonable" means "not a case that you construct to deliberatly violate this assertions", like $\hat{y}_i = 0$ or something
- We probably won't ever be able to find the true model, since it's not likely that we have all the features required

Nested Models

- Correct models have zero bias* but higher variance than the true model; incorrect models have additional bias but may have lower variance
- Technically, the null model is correct, so clearly **correct** isn't enough on its own

Our goal is then: find a **correct** model that fits the data as well as the **saturated** model, and fits better than the **null** model.

It won't always be possible to satisfy this for a given dataset; we will use this principle to guide our development of a model selection procedure.

*Note: I mean zero *additional* bias due to having wrong features present. The modelling procedure can still have bias by design, e.g. Ridge Regression.

Comparing Nested Models

It is immediately clear that comparing only the value of the loss function between two models will always result in the saturated model being picked.

We need to penalize complexity in an intellegent way.

Consider the **deviance**, which is the difference in loss from the saturated model:

$$D = 2(\tilde{\ell} - \hat{\ell})$$

where $\tilde{\ell}$ is the loss under the saturated model, and $\hat{\ell}$ is the loss under the candidate model.

How can we interpret this quantity?

Deviance

Note that under our defintion, $\tilde{\ell} = 0$, so the deviance is just twice the negative loss.

The change in deviance when adding a new predictor can be written

$$\Delta D = D_d - D_{d+1} = 2(\tilde{\ell} - \hat{\ell}_d) - 2(\tilde{\ell} - \hat{\ell}_{d+1}) = 2(\hat{\ell}_{d+1} - \hat{\ell}_d)$$

Deviance

Often, e.g. in regression, the loss function is just minus the log likelihood, so ΔD is recognized as being the standard *likelihood ratio statistic*.

Under regularity conditions, then, $\Delta D \to \chi^2_{p-d}$ as $n \to \infty$, where the candidate model has d parameters, *if the two models fit the data equally well*.

This is used, classically, to compare values of the change in deviance to χ^2_{p-d} critical values to assess whether adding a feature appreciably improves the fit of the model.

Is that a good idea?

Deviance

No.

There are a few reasons for this, but here is the biggest (and least obvious): in practice, you don't just randomly pick a predictor out of the available ones and include it in the model; you would include each predictor separately, and see which one decreases D the most.

You would then compare *this* decrease in deviance to the apprpriate χ^2 critical value.

. . . except the maximum of a sample of χ^2 random variables doesn't have a χ^2 distribution, so in practice this procedure adds many useless predictors.

Better Criteria

In practice, it is better to consider penalized loss criteria of the form

$$IC = c(n, d) - 2\hat{\ell}$$

where IC stands for Information Criterion, n is the number of observations, d is the number of parameters in the model, $\hat{\ell}$ is the maximized log-likelihood, and $c(n, p)$ is a penalization criteria that we need to choose.

This formulation of the problem is due to Davison (2003), *Statistical Models*.

Comparing Nested Models

There are two common choices for $c(n, d)$:

- $c(n, d) = 2d$ gives the **AIC**
- $c(n, d) = d \log n$ gives the **BIC**

Which one to use?

The BIC is Consistent

The **BIC** has the property that if the true model is among the candidates, then $P(\text{BIC chooses true model}) \to 1$ as $n \to \infty$ (if the loss function is minus the log-likelihood).

Denote by BIC the BIC for the true model, and by BIC_+ the BIC for a model with one more parameter. Then

$$P(\text{choose bigger, incorrect model}) = P(BIC_+ < BIC)$$
$$= P(2(\hat{\ell}_+ - \hat{\ell}) > c(n, d+1) - c(n, d))$$
$$\to \begin{cases} 0.16 \text{ for AIC} \\ 0 \text{ for BIC} \end{cases}$$

because $2(\hat{\ell}_+ - \hat{\ell}) \to \chi^2_1$, and $c(n, d+1) - c(n, d) = 2$ for AIC and $\log n$ for BIC.

But in finite samples. . .

However, in finite samples, the BIC is very conservative, so the AIC still might be preferable.

In practice, it is ideal to repeat your model selection procedure using both metrics, and investigate any disagreement manually.

How to do it

One way to actually use these things would be the following procedure, called **forward stepwise selection**

- Start with the null model M_0
- At iteration t,
  - Add each predictor into the model M_t separately and calculate the AIC (BIC) for each
  - Set M_{t+1} to be the model with the lowest AIC (BIC)

This is not guaranteed to converge, or to select the best model, but it is very useful in practice for getting an idea of which features are the strongest, and what the simplest model is that fits the training data well.

You would stop adding features when the AIC (BIC) stopped decreasing quickly, or started increasing.

Out of Sample Performance

We see that model selection is extremely subjective, and is as much art as it is science.

We now discuss evaluating a model's out of sample predictive performance.

Out of Sample Performance

Recall in lecture 1, we said that we need to split the data 3 times:

- A **training** set, which is used to fit the model
- A **validation** set, which is used to assess model performance
- A **test** set, which is used to... assess model performance

Out of Sample Performance

The difference between the **validation** and **test** sets is that the validation set is used *during* the model-building procedure, to compare models.

The **test** set is kept in a vault, and only brought out at the very end. It is used to *report* performance.

If you were reporting to your boss, or publishing a paper, you would quote statsitics from the **test** set.

You would not use the **test** set for any other purpose- doing so would cause you to report overly optimistic out of sample performance.

The **validation** performance, though, is an unbiased estimate of the **test** performance.

Continuous Targets

In the case of continuous targets, it is fairly simple to evaluate a model's test-set performance; you use the same metrics (squared error, correlation) as you would have on the training set.

We will focus our discussion on evaluating *binary classification models*, because

- This is a very common use case and
- It is actually more difficult than it sounds

Binary Classification Models

Why is it more difficult than it sounds?

Define the **confusion matrix**:

	True 1	True 0
Pred 1	TP	FP
Pred 0	FN	TN

which compares the targets to the predicted values. For the stats-oriented folks, this is just a 2×2 contingency table.

Confusion Matrix

The acronyms are

- TP: True Positive
- TN: True Negative
- FP: False Positive
- FN: False Negative

and we have

$$N = TP + TN + FP + FN$$

$$\sum_{n=1}^{N} t_n = TP + FN$$

$$\sum_{n=1}^{N} (1 - t_n) = TN + FP$$

Confusion Matrix

The *classification error* is then

$$Err = \frac{FP + FN}{N}$$

The *accuracy* is 1 minus the error.

Confusion Matrix

The *true positive rate* is

$$TPR = \frac{TP}{TP + FN}$$

The *false positive rate* is

$$FPR = \frac{FP}{FP + TN}$$

TPR estimates $P(\text{classifier predicts positive}|\text{true positive})$, and FPR estimates $P(\text{classifier predicts positive}|\text{true negative})$

Important: $TPR \neq 1 - FPR$

Confusion Matrix

Let's do an actual example. Suppose we are building a model to predict whether it will be rainy tomorrow or not. We get a year's worth of weather data for our region and code up $t_n = 0$ for sunny and $t_n = 1$ for rainy, for the n^{th} day. We get the confusion matrix

	True 1	True 0
Pred 1	200	30
Pred 0	80	55

We can tell from this table that

$$TP = 200$$
$$FP = 30$$
$$TN = 55$$
$$FN = 80$$

Confusion Matrix

We can tell from this table that

$$TPR = \frac{TP}{TP + FN} = \frac{200}{280} = 71\%$$
$$FPR = \frac{FP}{TN + FP} = \frac{30}{85} = 35\%$$

From the table, we can make the following statements:

- It rained on $200 + 80 = 280$ days in the observed data (maybe this study was done in Vancouver)
- We predicted 200 of these rainy days correctly, and 80 incorrectly; our model captured $200/280 = 71\%$ of the rainy days
- We predicted 30 out of 85 sunny days to be rainy, or $30/85 = 35\%$

Confusion Matrix

Our model had a misclassification rate of $(80 + 30)/365 = 42\%$, so we are pretty confident it's not a good model. Why bother with those other statistics?

Consider now a similar binary classification problem: we want to predict default in the next 12 months for the customers in our credit card portfolio at the bank. We model $N = 10,000$ customers, and observe the following confusion matrix:

	True 1	True 0
Pred 1	3	150
Pred 0	7	$9,840$

The misclassifcation rate is only $157/10,000 = 1.57\%$. The model is awesome!

Unbalanced Data

The model is not awesome. The default rate in this portfolio is only $10/10,000 = 0.1\%$.

If we just predicted $\hat{t}_n = 0$ for every customer, we'd get the following contingency table:

	True 1	True 0
Pred 1	0	0
Pred 0	10	$9,990$

The model is predicting $1.57/0.1 = 15.7$ times worse than what we would get if we just predicted $\hat{t}_n = 0$ for every observation, *in terms of misclassification rate*.

Unbalanced Data

The TPR and FPR for these tables tell a different story. For model 1,

$$TPR = \frac{3}{3+7} = 30\%$$
$$FPR = \frac{150}{150 + 9,840} = 1.5\%$$

where for model 2,

$$TPR = \frac{0}{10} = 0\%$$
$$FPR = \frac{0}{9,990} = 0\%$$

Unbalanced Data

This is why TPR and FPR are important- they allow for the tuning of the model to produce the right *balance* of misclassifications.

In this example, since the cost of the lost business (150 good customers predicted as bad) is probably less than the cost of the true predicted defaults, the model might not be that bad. In fact for business purposes, you'd probably have actual estimates of each dollar cost, which you would use directly.

Confusion matrices have another major use in model validation.

Soft Classification

Most of the models we learn about don't directly give hard 0/1 classifications- they give predicted probabilities, called **soft** classifications.

Before, we translated these into hard classifications by assigning

$$\hat{t}_{hard} = \begin{cases} 1 \text{ if } \hat{t} > 0.5 \\ 0 \text{ if } \hat{t} < 0.5 \end{cases}$$

There is no real reason why using 0.5 would be optimal in general, though it is intuitive.

We turn to the question: for a given classification model, is it possible to pick a cutoff that gives us good hard classifications?

Thresholds

The key observation is that each choice of the threshold, $0 < h < 1$, generates a contingency table.

And therefore, gives distinct Err, TPR, FPR.

We can choose the threshold to give the balance that we want.

We can also use this idea to validate the model itself, by asking "is it even possible to pick a good h?"

The KS Statistic

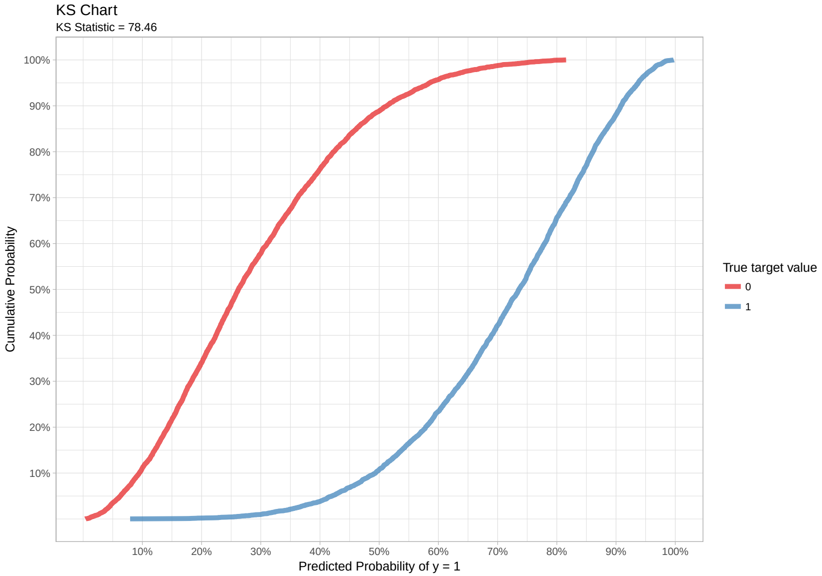
The first way we can use this idea is to look at the **KS** statistic.

KS stands for Kolmogorov-Smirnov; this statistic is the same metric that they used in their development for a statistical test for equality between two probability distributions.

In ML, we use this idea to compare the cumulative distribution of $\hat{t}$, the soft classifications, for the true positives $t = 1$ and true negatives $t = 0$, as a function of h.

The maximum distance between these curves is the KS Statistic.

The KS Chart


KS Chart
KS Statistic = 78.46
Cumulative Probability
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
Predicted Probability of y = 1
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
True target value
0
1

The KS Statistic

The KS Statistic is a measure of how well the classifier *discriminates* between the two classes.

If all of the predicted probabilities are very close together, *regardless of their magnitude*, then these two cumulative distributions will be close together, and KS will be small.

If the predicted probabilities are very high for the $t = 1$ group and very low for the $t = 0$ group, then the curves will be well separated and KS will be high.

KS is invariant to strictly monotone transformations of the predicted probabilities, which is convenient.

ROC Curve

How does the KS statistic relate to the notion of picking a cutoff for assigning hard classifications?

For models where a good cutoff (in terms of separation of the true 1's and 0's) exists, the KS will be high.

We can look at another commonly used statistic for assessing the power of the model to discriminate between classes.

ROC Curve

Another commonly used method to assess the power of a model to discriminate between classes is the ROC (Receiver Operating Characteristic) Curve.

Core concept: Each value of predicted probability, $h \in (0, 1)$, that we use as a cutoff generates a distinct 2×2 confusion matrix.

Sometimes the subset of $\mathbb{R}^{2\times 2}$ containing all possible contingency tables for all possible cutoffs is called the *ROC Space*.

ROC Curve

Suppose, again, we have a collection of soft classifications $\hat{t}_n \in (0, 1)$.

We choose a cutoff h such that we classify

$$\hat{t}_{n,hard} = \begin{cases} 1 \text{ if } \hat{t}_n > h \\ 0 \text{ else} \end{cases}$$

This gives us a contingency table.

This, in turn, gives us a distinct TPR_h and FPR_h.

ROC Curve

The *parametric curve* $(x(h), y(h)) = (FPR_h, TPR_h)$ is called the **ROC Curve**.

A model with a wider ROC curve will allow us to choose a threshold that gives a high TPR with a low FPR, which is desirable.

The area under the ROC curve, or **AUC**, is a measure of the ability of the model to rank order observations by their probability of $t = 1$.

In fact, $AUC = P(\hat{t}_1 > \hat{t}_2 | t_1 = 1, t_2 = 0)$, i.e. it is the probability of the model correctly rank-ordering a true 1 vs a true 0.

Plotting the ROC Curve

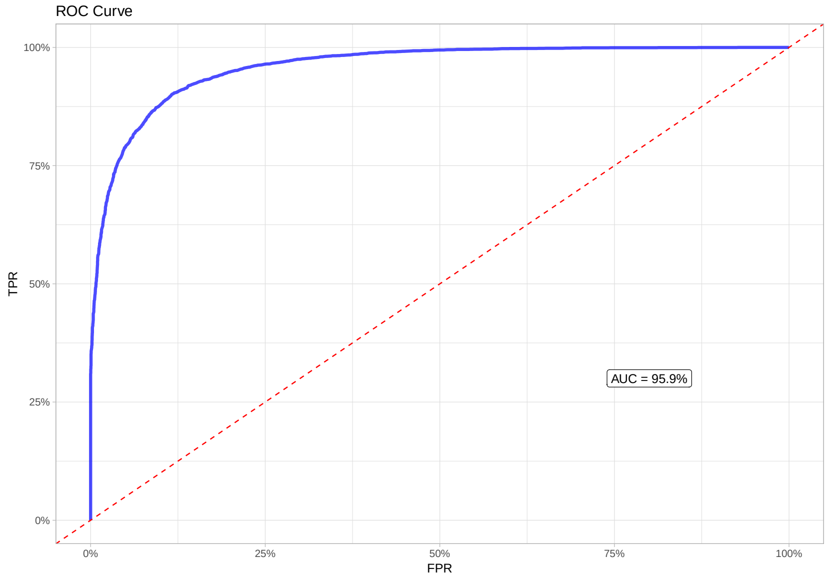
To actually plot the ROC curve for a particular application:

- Take all the distinct predicted probabilities that your model yields
- Compute the TPR and FPR that result from using each as a cutoff
- Plot the curve (FPR,TPR)

Compute the AUC using numerical integration (e.g. trapizoid rule):

$$A\hat{U}C = \sum_i \frac{TPR(h_i) + TPR(h_{i-1})}{2} \times (FPR(h_i) - FPR(h_{i-1}))$$

ROC Curve


ROC Curve
TPR
FPR
AUC = 95.9%