Model assessment

How good is my model and how do I properly test it?

Terence Parr
MSDS program
University of San Francisco

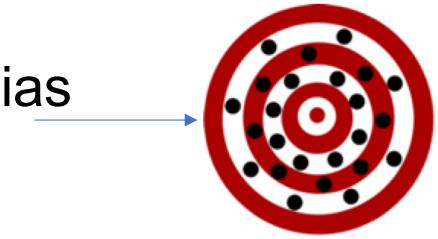
UNIVERSITY OF SAN FRANCISCO

Terminology: Loss function vs metric

- *Loss function*: minimized to train a model (if appropriate)
  E.g., gradient descent uses loss to train regularized linear models
- *Metric*: evaluate accuracy of predictions compared to known results (the business perspective, usually done on validation/test set)
- Both are functions of y and $\hat{y}$, but loss is also possibly func of model parameters (e.g., linear model regularization loss tests parameters)
- Examples:
  - Train: MSE loss & Test: MSE metric
  - Train: MSE loss & Test: MAE metric
  - Train: Gini impurity & Test: misclassification or FP/FN metric
- If metric is applied to validation or test set, informs on *generality* and quality of your model

See also stackoverflow post by Chstiros Tsatsoulis: https://goo.gl/T5AmrT

Accuracy, generality, & training-test sets



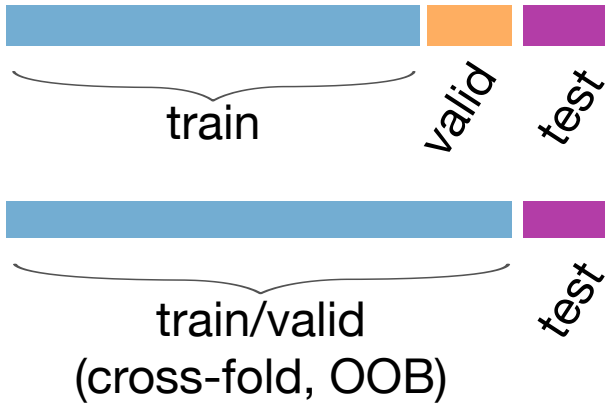
- A model can be inaccurate/noisy but not biased; to me, bias means systematically always under-predicting or always over-predicting
- We measure accuracy (bias oftened used as shorthand) and generality by comparing predictions to known results, usually with the same metric
- But, how can we measure two things with the same metric?
- We compare metrics computed on two data sets pulled from the same distribution (hopefully): training and validation/test sets
- Summary: goal is high accuracy on the test set because it implies generality (and also high accuracy / low bias)
- So, let's look at how to break up our data into different sets

Train, validate, test

- *This might be the most important slide of entire class!*
- We always need 3 data sets with known answers:
  - training (used to train model)
  - validation (as shorthand you'll hear me / others call this test set)
  - testing (put in a vault and **don't peek**!!)
- Validation set: used to evaluate, tune models and features
  - Any changes you make to model tailor it to this specific validation set
- Test set: used exactly **once** after you think you have best model
  - The only true measure of model's *generality*, how it'll perform in production
  - Never use test set to tune model
- Production: recombine all sets back into a big training set again, retrain model but don't change it according to test set metrics

How to extract validation, test sets

- Extract random subsets; perhaps 70%/15%/15%; can shuffle then chop
- Or, grab 15% test set (and hide it away) & use RF OOB or cross-fold on remainder for train/valid
- For RF, we can start with out-of-bag score
- Ensure validation set has same properties as test set (e.g., size, time, …):
  - if 10k samples in test, make 10k sample validation set
  - if test set is latest 2 months, validation must be latest 2 months of remaining data



Mechanics of splitting, testing validation set

- Get X, y from dataframe, then use train_test_split() to split:

```
X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, 0.20)
```

- Train model with training data, test with other set:

```
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
r2 = rf.score(X_test, y_test)
```

Or:

```
y_pred = rf.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
```

Key question: is your data time-sensitive?

- Time series sometimes obvious: temperature, stock prices, sales, inflation, city population, …
- You can try to detrend the data to flatten average y etc…
- Almost all data sets are time sensitive in some way (boo!)
- Some data sets are skewed over time even if no date column; e.g., new users to facebook are different over time
- Try to find things that are less time dependent; e.g., air conditioning sales appear to fluctuate over time but these sales are driven more by associated temperature and humidity than date
- Create lagging windows; e.g., average sales per day over the last week
- Try to convert absolute dates (and date-related variables) to relative variables, such as **age** computed from **salesdate** - **manufacturingdate**

Splitting time-sensitive data sets

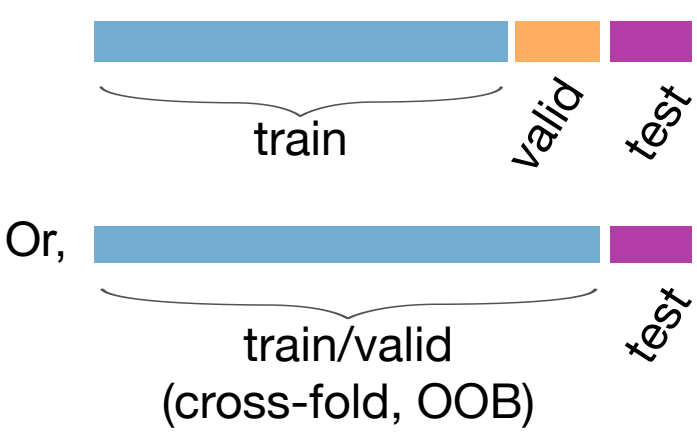
- If your data set is time-sensitive, **do not shuffle**: sort by date then use newest rows as valid/test sets
- This means OOB cannot be used for time-sensitive data sets as it randomly selects test records

```
df = df.sort_values('saledate')
n_train = len(df)-n_valid
df_train = df[:n_train]
df_valid = df[n_train:]
```

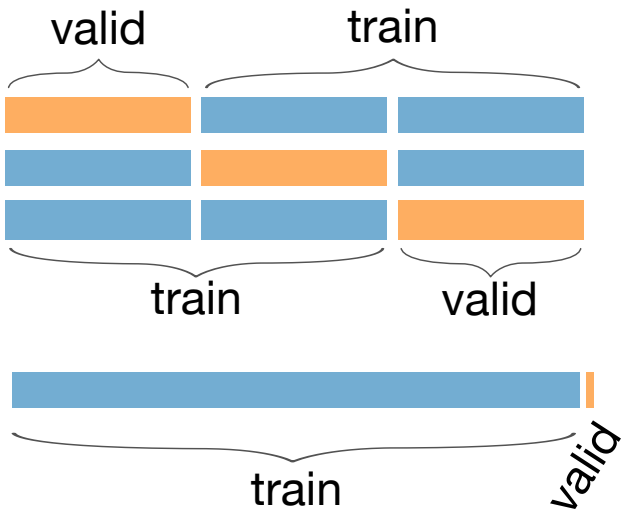
	SalesID	saledate	Hours	UsageBand	Group
Training	1007352	2/19/08	1171	Low	TEX
	2297737	8/15/08	5659	Medium	TEX
	2306055	9/19/08	8999	Medium	TEX
	2275176	10/16/08	4425	Medium	SSL
	2274051	12/16/08	2425	Low	TTT
	2268394	1/31/09	3112		WL
	2288319	2/9/09	5089	Low	BL
	1745722	2/17/09	4489	Low	BL
	2297316	2/27/09	963		WL
	1896854	3/12/09	2851	Medium	BL
	2298201	11/14/09	5075	Low	BL
	2298929	2/15/10	13173	Medium	TTT
Validation	2276590	3/4/10	6758		TEX
	2298928	5/6/10		Medium	WL
	2277691	2/3/11	7191	Low	MG
Testing	2296994	2/10/11	3783		SSL
	2294960	3/18/11	8201	Low	
	2288988	7/27/11	9115	Medium	WL

Testing strategies

Always split out test set



Validation cross-fold or leave-one-out



Metrics interpretation

- Basic idea: for each test record, compute error using y and $\hat{y}$; the metric is then usually the average of these errors
- **Perspective**: Is 99% vs 99.5% accuracy difference 2x (from 100%) or 0.5%? What about 80% vs 90%? 10% diff but also 2x
- As we approach 100%, getting better is tougher and tougher
- Is 90% accuracy or R^2=0.8 good? Maybe. What are lower bounds from a simpler or trivial mode?
- Classifiers must beat *a priori* probabilities
  - If 90% of email are spam, your model must beat 90% accuracy
- Regressors should beat "mean model" and linear model
- Can crappy model outperform a human? Can still be useful

Training loss/metric isn't that useful

- Training score not really useful by itself because, for example, we can get good fit for random data $X \rightarrow y$ with 1000 x 4 data, R^2=.85
- Actually, if training score is low, model is too weak
  - Or, dataset is missing exogenous vars like "we had a sale that day" or "closed on holidays"
- If training score is low, test score will also be low

```
X_train = np.random.random((1000,4))
y_train = np.random.random(1000)
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)
rf.score(X_train, y_train)
```

```
0.847473179728131 4
```

WOW!

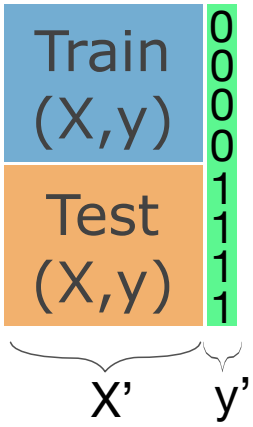
What if training score is good but validation is very bad?

- Overfit model?
- Bad validation set?
  (didn't extract random or time-sorted set or just given bad set?)
- Time-sensitive data set diverging? (try detrending data)
- Not properly applying feature transforms from training to validation set?
- Bug?

Comparing training / validation sets

Awesome but not well-known trick

- Process to see if valid (or test) set is distinguishable from train set
  1. Combine both X and y from train & valid sets into single data set
  2. Create new target column called **istest** (y' in illustration)
  3. Train a model on the combined set (X', y')
  4. Assess (training) metric on this new training set
- If train/valid are drawn from same distribution, should get bad metric since there's no way to distinguish between them
- But if you get a good metric, that implies that train/valid sets are drawn from very different distributions
- Idea: Maybe use repeated trials using random {0,1} for y' as null distribution then see if training score on real 000111 is better than that average score; if so, training and validation sets are different



If train/test are easily distinguishable…

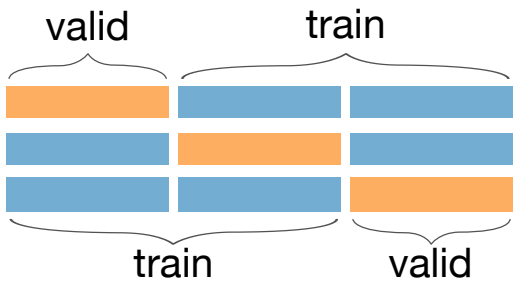
- You might find that ID or date var is different in train vs valid set
- Drop that feature and see how the validation score changes
- Or if y steadily climbs, and all y's are bigger in the validation set, a simple inequality distinguishes training and test set
- Look at the feature importances of original and **istest** models. Features that are important in both are the problem
  - If it's not important in original model, we don't care about it: it's not predictive of target
  - If not important in **istest** model, it's not causing confusion between sets
  - Imagine vehicle age is predictive for training set and to distinguish train/test; what if age=0 for all in test set? It's difference between used (train) and new cars (test)

Leakage: What if your model is too good?

- Be suspicious if you get an excellent metric; yes, you are awesome, but near-perfect scores are often sign of bug or leakage
- Kaggle example: Uni Melbourne (predict grants)
  - Jeremy Howard (who won) told me that one of the data fields leaked information about whether grant was awarded or not
  - An explanatory variable was only filled in but only if grant was awarded
  - A missing value almost perfectly predicted “no grant award”
  - I built basic model and got validation accuracy 87% (red flag)
- Rent price example: price per bedroom is a good feature and I got very high R^2; oops, I just incorporated (leaked) y price into X

https://www.kaggle.com/c/unimelb

Stability of metric values



- Getting a single validation metric is usually not enough because scores can vary from run to run because of outliers and anomalies (even with k-fold)
- Also good to collect and compare a variety of metrics
- Consider score fluctuations in NYC rent data (before cleaning)

	OOB	R^2	MAE	MSE
0	0.582	0.002	1,150.354	2,402,630,918.659
1	0.027	0.050	878.512	2,053,053,487.605
2	-0.309	0.323	408.299	3,852,177.529
3	-0.152	-44.812	527.185	265,146,585.910
4	-0.155	-0.105	404.700	7,275,725.459

Here are results from 5 train/test splits

Noise/outliers can cause mismatch between train/valid sets; k-fold, random subsets will see high variability

See https://github.com/parrt/msds621/blob/master/notebooks/assessment/metrics.ipynb

Summary

- Start machine learning modeling by splitting data into training, validation, and test sets; lock away the test set as only true measure of generality tested after you have your final model
- Good training metric is necessary but not sufficient; we want a good validation/test metric to show generality of model
- Can use cross validation or just single split during development (OOB also with random forests)
- Validation/test sets must have same distribution as training set
- We use loss to train, metrics to validate/test
- Next, we look at measuring regressor and classifier error