Week 5

Model Eval + Refinement

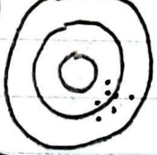
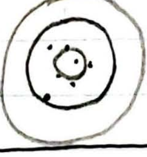
1. • In-sample eval tells how well model will fit the data that's used to train it. However, it can't tell how well the trained model can be used to predict new data.
• Sol: split data
  - 70% → training
  - 30% → testing (if good, send for training)

• There's a function in scikit-learn to do this for us.

```
1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test = train_test_split(x_data,
  y_data, test_size=#, random_state=#)
```

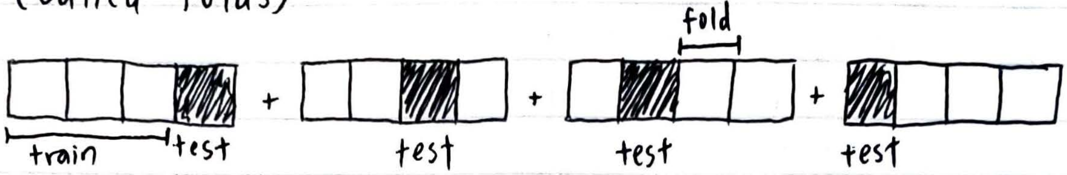
- Inputs (vars), ratio of test data, RNG seed, Outputs (arrays)

• Generalization Performance
  - Gen. error is the measure of how well the model does at predicting unseen data
    ▪ Error from testing data is an approximation of this error
  - There is a balance between using training and testing data:

	↓ Testing data	↑ Testing data
↓ Training data	Use more data lol	 ↓ Accuracy ↑ Precision
↑ Training data	 ↑ Accuracy ↓ Precision	Not enough data to go around

- To solve this, use cross-validation.

- In cross-validation, data is split into k parts (called folds)



- Average results are good approximation

- sklearn has a cross-validation function: output array of scores

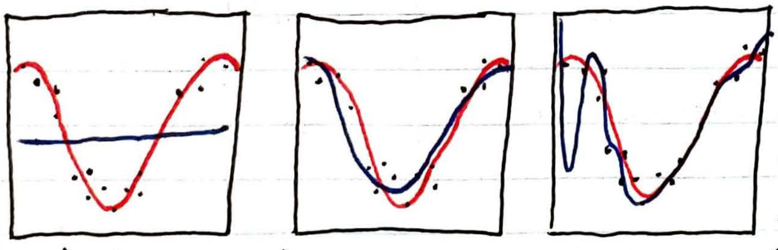
```
1 from sklearn.model_selection import cross_val_score
2 Scores = cross_val_score(lr, x_data, y_data, cv=#)
  Type of model (linreg), # of partitions (folds)
3 np.mean(scores)
```

- For more info, cross_val_predict() can be used to generate the predicted y-val of each test element.

```
1 from sklearn.model_selection import cross_val_predict
2 yhat = cross_val_predict(lr2e, x_data, y_data, cv=#)
```

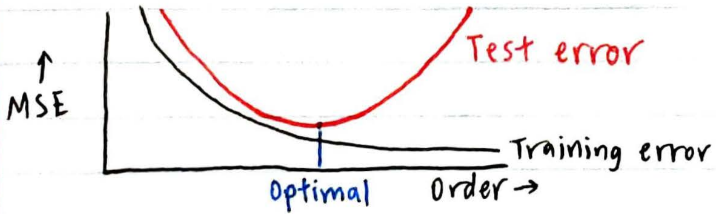
2. How to pick the best polynomial for poly. reg.

- The goal of *model selection* is to det the order that's the best fit of polynomial y(x).
  - Underfitting: #y < desirable (too simple)
  - Overfitting: #y > desirable (too extreme where few data pts)



(fits noise too well)

Notice overfitting is more "accurate" to training pts but can't extrapolate well



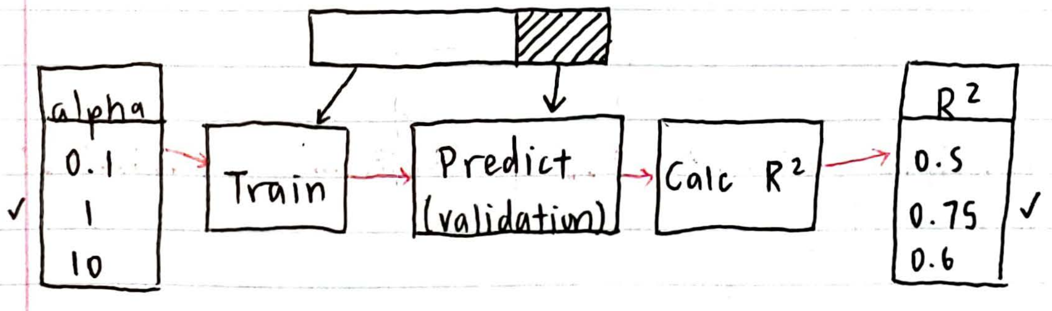
- The order at which R^2 is closest to 1 is the optimal order. This can be done in Python:

```
Rsqu_test = []  # Init empty list
order = [1, 2, 3, 4]  # Set to whatever suits you
for n in order:
    pr = PolynomialFeatures(degree = n)
    x_train_pr = pr.fit_transform(x_train['I.V.'])  # Trans. data
    x_test_pr = pr.fit_transform(x_test['I.V.'])
    lr.fit(x_train_pr, y_train)  # Creates model
    Rsqu_test.append(lr.score(x_test_pr, y_test))  # Calc R²
```

---

3. Ridge Regression (prev. overfitting + manage outliers)
- Ridge reg. uses a param. "alpha" selected before fitting the data
- As $\alpha \to \infty$, the coeff.'s $\to 0$ (if α is too high, it can cause underfitting)
- α is selected w/ cross-fitting:

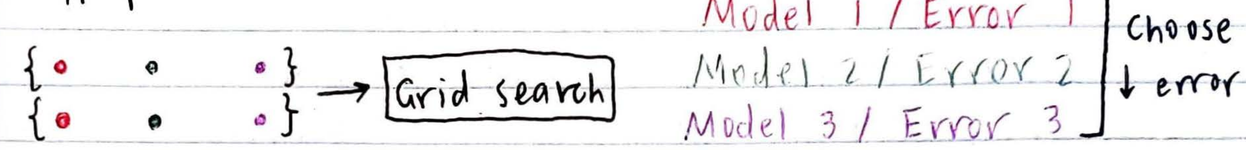
```
from sklearn.linear_model import Ridge
RidgeModel = Ridge(alpha=#)
RidgeModel.fit(x, y)
Yhat = RidgeModel.predict(x)
```

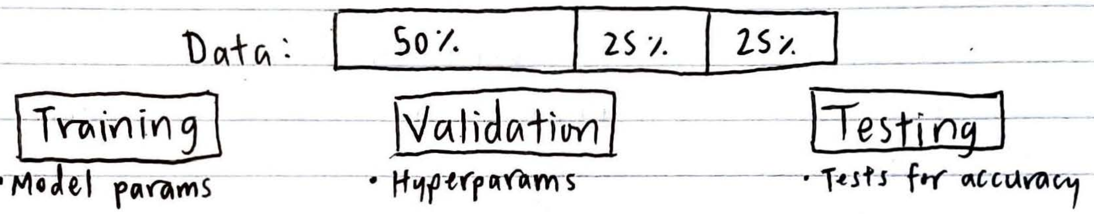


4. Grid search

- Terms such as α (not used in training/ testing) are called *hyperparameters.*
  - Scikit-learn has a way to auto-iterate over these hyperparameters w/ cross-validation (known as grid search)

hyperparameters:





```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
parameters1 = [{'alpha':[#1, #2, ...]}] *
RR = Ridge()
Grid1 = GridSearchCV(RR, parameters1, cv=#)
Grid1.fit(x_data['var1', 'var2', ...], y_data)
Grid1.best_estimator_
scores = Grid1.cv_results_
scores['mean_test_score']
```

For multiple params: ... = [{'alpha': [#1, ...], 'normalize': [T, F]}]