

# Machine Learning

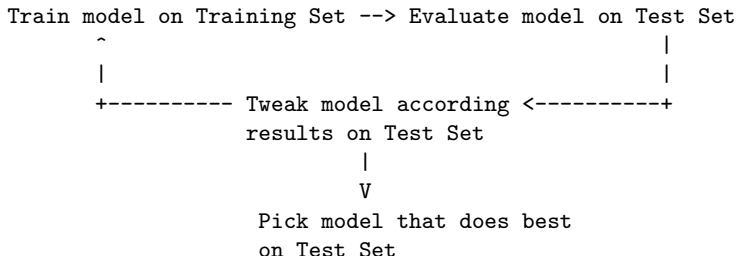
Pawel Wocjan

University of Central Florida

Spring 2019

# Validation set

- ▶ We introduced previously the partitioning a data set into a training set and a test set.
- ▶ This partitioning enabled you to train on one set of examples and then to test the model against a different set of examples.
- ▶ With two partitions, the workflow would look as follows:



# Validation set

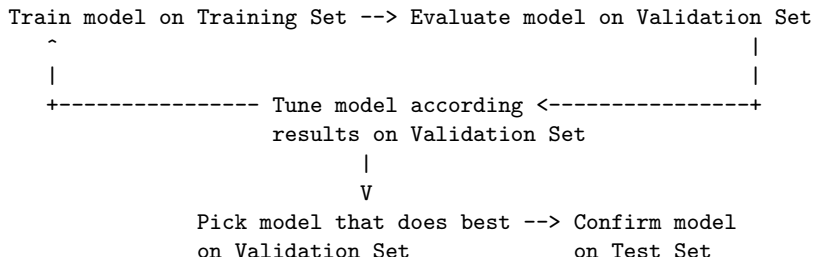
- ▶ Dividing the data set into two sets is a good idea, but it is not enough.
- ▶ You can greatly reduce the chances of overfitting by partitioning the data into three subsets shown below:



- ▶ Use the validation set to evaluate results from the training set.
- ▶ Then, use the test set to double-check your evaluation after the model has “passed” the validation set.

# Validation set

- ▶ With three partitions, the workflow looks as follows:



- ▶ In this improved workflow:
  - ▶ Pick the model that does best on validation set.
  - ▶ Double-check that model against the test set.
- ▶ This is a better workflow because it creates fewer exposures to the data set.

# Validation methods

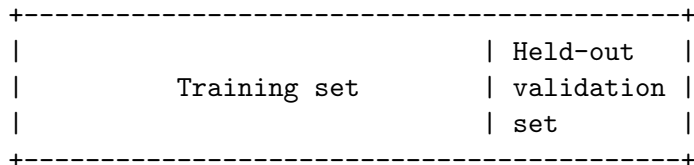
- ▶ Splitting your data into
  - ▶ training set
  - ▶ validation set
  - ▶ test set

may seem straightforward, but there are a few advanced ways to do it.

- ▶ This is especially important when there is little data available.

# Simple hold-out validation

- ▶ Set apart some fraction of your data as your test set.
- ▶ Train on the remaining data, and evaluate at the end on the test set.
- ▶ To prevent information leaks, you shouldn't tune your model based on the test set, and therefore you should also reserve a validation set.
- ▶ Schematically, hold-out validation look as follows:



## Pseudo code for simple hold-out validation

```
# split non-test data into training and validation

training_data = data[num_validation_samples:]
validation_data = data[:num_validation_samples]

model = get_model()
model.train(training_data)

validation_score = model.evaluate(validation_data)

# At this point you can tune your model,
# retrain it, evaluate it, tune it again ...
```

## Pseudo code for simple hold-out validation continued

```
# Once you've tuned your hyperparameters ,  
# it's common to train your final model  
# from scratch on all non-test data available .
```

```
model = get_model()  
model.train(data)  
test_score = model.evaluate(test_data)
```



# Simple hold-out validation

- ▶ This is the simplest evaluation protocol, and it suffers from one flaw: if little data is available, then your validation and test set may contain few samples to be statistically representative of the entire data at hand.
- ▶ This is easy to recognize: if different shuffling rounds of the data before splitting end up yielding very different measures of model performance, then you are having this issue.
- ▶ K-fold validation and iterated K-fold validation are two ways to address this issue.

# K-fold validation

- ▶ With this approach, you split your data into  $K$  partitions of equal size.
- ▶ For each partition  $i$ , train a model on the remaining  $K - 1$  partitions, and evaluate it on partition  $i$ .
- ▶ Your final score is the average of the  $K$  scores obtained.

# K-fold validation

## Three-fold validation

Fold 1	Validation	Training	Training	→	validation score 1
Fold 2	Training	Validation	Training	→	validation score 2
Fold 3	Training	Training	Validation	→	validation score 3

# Iterated K-fold validation

- ▶ This method is for situations in which you have relatively little data available and you need to evaluate your model as precisely as possible.
- ▶ It consists of applying  $K$ -fold validation multiple times, shuffling the data every time before splitting it  $K$  ways.
- ▶ Note that you end up training and evaluating  $P \times K$  models, where  $P$  is the number of iterations you use, which can be expensive.