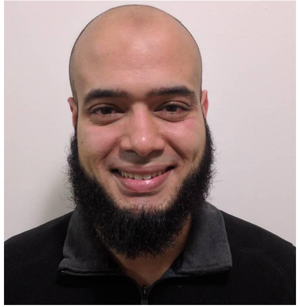
Machine *Learning*

Cross Validation

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



K-Fold Cross Validation

- **K-Fold Cross Validation**
  - **What**: a *data resampling (partitioning)* method for *performance evaluation*
  - **Why**: we may introduce **selection bias** in the way we select train/val/test split.
    - This is especially critical when dealing with small datasets
    - We need a reliable estimate of the model's generalization error
  - **How**: the given data set is split into K **equally** sized folds (subsets)
    - This process is repeated k times, with each fold serving as the validation set once
  - **Cons**: Requires multiple runnings (more time/effort)
- For example: a dataset of 100 examples in a 4-fold setup is divided into 4 folds, each with 25 examples
  - In fold1, the first 25 examples are used for validation and the remaining 75 for training
  - In fold2, the second 25 examples are used for validation
  - Remember to **shuffle** the data first

4-Fold Cross Validation Experiments

- For each split, train a model using the 3 subsets and then validate on the validation set. The **average** performance across all 4 splits is used as the validation performance for the model
- Tip: The test set is NOT used in this process. It is used later after deciding on the final chosen algorithm and its model parameters

					Validation Performance
Split #1	Validation	Training	Training	Training	70%
Split #2	Training	Validation	Training	Training	15%
Split #3	Training	Training	Validation	Training	90%
Split #4	Training	Training	Training	Validation	5%
					45% (**average**)

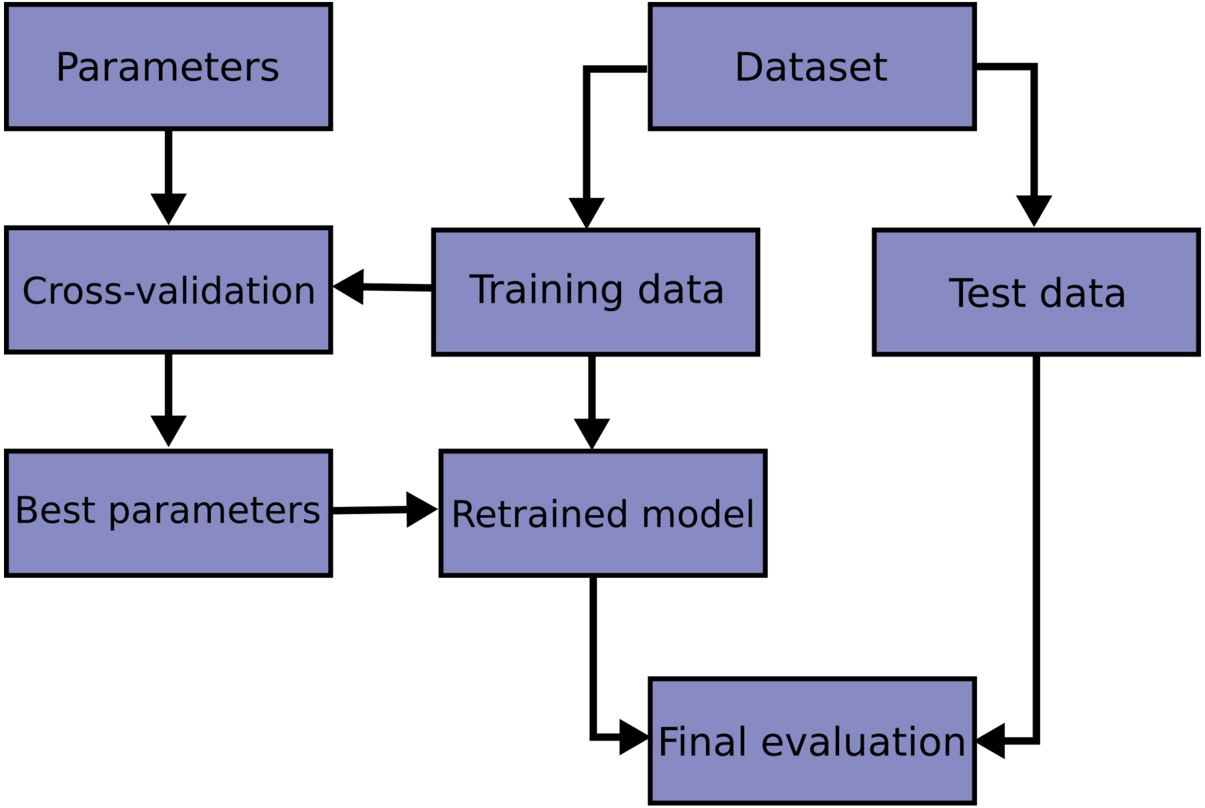
Investigating the performance

- Clearly, the average performance is quite low
- What other information can we get from these numbers?

- There is a **variance** in the performance (90 vs 5)
  - np.std([70, 15, 90, 5]) ~= 36
- This indicates instability in the performance
- Tip: consider the mean and standard deviation

Validation Performance
70%
15%
90%
5%
45% (**average**)

Typical Flow



Img src

Leakage through cross-validation

- It is wrong to compute some statistics or select features based on val/test set
- First, take the test set far from the trainval set
- For each fold
  - Do whatever preprocessing on the train-part. Deal with val as your test set
  - Don’t share preprocessing information between folds using the full trainval set

k-Fold using Sklearn

- sklearn.model_selection.KFold(n_splits=5, *, shuffle=False, random_state=None)

Parameters::

n_splits : *int, default=5*

Number of folds. Must be at least 2.

> *Changed in version 0.22:* `n_splits` default value changed from 3 to 5.

shuffle : *bool, default=False*

Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

random_state : *int, RandomState instance or None, default=None*

When `shuffle` is True, `random_state` affects the ordering of the indices, which controls the randomness of each fold. Otherwise, this parameter has no effect. Pass an int for reproducible output across multiple function calls. See Glossary.

```python
import numpy as np
from sklearn.model_selection import KFold

X = []
t = []
for i in range(1, 10, 1):
    X.append([i, i * 10, i * 20])
    t.append(i * 100)
X = np.array(X)
t = np.array(t)
'''
[[  1  10  20]
 [  2  20  40]
 [  3  30  60]
 [  4  40  80]
 [  5  50 100]
 [  6  60 120]
 [  7  70 140]
 [  8  80 160]
 [  9  90 180]]
'''
```

```python
kf = KFold(n_splits=3, random_state=None, shuffle=False)
#print(kf.get_n_splits(X))    # 3

for train_index, val_index in kf.split(X):
    print(f'Validation index: {val_index} - Training index{train_index}')
    # Extract data
    X_train, X_val = X[train_index], X[val_index]
    t_train, t_val = t[train_index], t[val_index]
'''
Validation index: [0 1 2] - Training index[3 4 5 6 7 8]
Validation index: [3 4 5] - Training index[0 1 2 6 7 8]
Validation index: [6 7 8] - Training index[0 1 2 3 4 5]
'''
```

- **Without replacement**: Each example is used in one part only
- Using 5-folds (20% test) or 10-folds (10% test) are common practice
  - But still think about your dataset size
  - Large K ⇒ a lot of computations (and other issues such as variance)

```python
# The first n_samples % n_splits folds have size n_samples // n_splits + 1,
# other folds have size n_samples // n_splits
kf = KFold(n_splits=4, random_state=None, shuffle=False)

for train_index, val_index in kf.split(X):
    #print(f'Validation index: {val_index} - Training index{train_index}')
    # Extract data
    X_train, X_val = X[train_index], X[val_index]
    t_train, t_val = t[train_index], t[val_index]
'''
Validation index: [0 1 2] - Training index[3 4 5 6 7 8]
Validation index: [3 4] - Training index[0 1 2 5 6 7 8]
Validation index: [5 6] - Training index[0 1 2 3 4 7 8]
Validation index: [7 8] - Training index[0 1 2 3 4 5 6]

We have 9 samples to divide on 4 groups
9/2 = 4
So 4 groups
but 2 * 4 = 9. Then the first group takes the extra sample
'''
```

```python
kf = KFold(n_splits=4, random_state=None, shuffle=True)

for train_index, val_index in kf.split(X):
    print(f'Validation index: {val_index} - Training index{train_index}')
    # Extract data
    X_train, X_val = X[train_index], X[val_index]
    t_train, t_val = t[train_index], t[val_index]

'''
Validation index: [1 3 6] - Training index[0 2 4 5 7 8]
Validation index: [0 2] - Training index[1 3 4 5 6 7 8]
Validation index: [5 8] - Training index[0 1 2 3 4 6 7]
Validation index: [4 7] - Training index[0 1 2 3 5 6 8]

With each run, you get different ordering
you can fix the ordering by passing value to random_state
e.g. random_state = 1
We use this for reproducible results (others can generate the SAME results)
'''
```

SKlearn auto-processing with cross_val_score

- SKlearn allows us to automate the entire process of cross validation
- SKlearn's Pipeline allows multiple transformations before fitting
  - This sequentially applies a list of transforms (fit + transform) and a final estimator (transform)
  - It avoids data leakage
  - Default scoring: accuracy for most classifiers and r2 for regressors
  - We can treat the pipeline like a model
- Once you are done with CV, you again fit your model to your data

```
degree = 3
pipeline = make_pipeline(MinMaxScaler(),
                         PolynomialFeatures(degree),
                         LinearRegression(fit_intercept=True))

kf = KFold(n_splits=4, random_state=35, shuffle=True)
scores = cross_val_score(pipeline, X, t, cv = kf,
                          scoring = 'neg_mean_squared_error')
scores *= -1    # change to mean_squared_error
```

Leave-one-out cross-validation (LOOCV)

- This is a special case of K-fold cross-validation in which the number of folds is the same as the number of observations(K = N)
- Pros:
  - The model is trained with almost all data (N-1) which helps to build a strong model
  - No selection bias or randomness (almost)
- Cons:
  - It is computationally intensive (the model must be fitted N times)
  - If there are a lot of outliers, we will have big variance in the validation performance
- Key use case: when working with very small datasets
- SKlearn: from sklearn.model_selection import LeaveOneOut

Leave-one-group-out Cross-Validation (LOGOCV)

- Sometimes the data represents groups of something. We need the validation fold to be based on 1 single group only (don't leak)
    - Note: we refer here to the **input data groups** NOT to the target data
        - Classification - Target: You can't classify image as a cat if you never saw one!
- Example: We have the profile of different football teams (all players, their history and team's history). We would like to predict something about a team given its data
    - Each team can be a group by itself
    - Or each relevant subset of teams can be a group
- from sklearn.model_selection import LeaveOneGroupOut
    - In addition to (X, Y), you add also group for each sample

```python
import numpy as np
from sklearn.model_selection import LeaveOneGroupOut

X = np.array([101, 102, 103, 301, 302, 305, 400, 501, 502])
y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
groups = np.array([10, 10, 10, 30, 30, 30, 40, 50, 50])

logo = LeaveOneGroupOut()

for train_index, test_index in logo.split(X, y, groups):
    X_train, X_test = X[train_index], X[test_index]
    print(X_test)
'''
[101 102 103]
[301 302 305]
[400]
[501 502]
'''
```

Is random seed a hyperparameter to tune?

- If you run your code with random_state=None, it means that the random seed is not specified
  - This means with every run you will get a **different result**!
- Should you keep trying different random seeds to pick the best results?!
  - Some researchers may do this in order to achieve higher results in their papers!!
- There is a debate with more votes toward **NO**
  - If different seeds has variance in performance, this is an important indicator for **model instability or data issues**
  - It's better to invest time in improving your ML pipeline to achieve better generalization performance
- **Fixing** randomness is important for **reproducibility**

About APIs

- Libraries(Sklearn, Pytorch, etc) tend to provide 2 types of functionalities
  - Low level functionalities (e.g. kfold)
  - High level functionalities (e.g. cross_val_score)
    - It's a single call that's capable of performing multiple tasks
- It is important to familiarize yourself with the available APIs
  - Otherwise, you will reinvent the wheel a LOT
  - Keep in mind that the tools are just a means, not the end goal
  - So don’t burn much time
- Be cautious about high-level APIs
  - Although nice to have something that cuts 10-20 lines of code and their bugs
  - high-level APIs can be inflexible and limit your control over the process

Implementation Tip

- Anytime you see some code snippet from a library, e.g., sklearn, you should have some curiosity about the implementation details
- If you feel this is just matter of implementation, then skip it
- Otherwise, think how to implement or jump in the library code to get some insigths

Relevant Resources

- Cross Validation: Wiki
- Sklearn: Cross-validation: evaluating estimator performance
- Sklearn: scoring functions
- K-fold cross-validation: Article
- Training-validation-test split and cross-validation done right: Article
- Fine Tuning Random Seed
- TransformedTargetRegressor for transforming targets

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."