Continuous Integration and Tree Implementation, part 2

PU Machine learning mit Java, Weka und UIMA

Nils Reiter,
nils.reiter@uni-koeln.de

December 15, 2020
Winter term 2020/21

Section 1

Looking Back

Unit Tests

- Automatic verification of code
- Define expected results, verify that code delivers
  - Alternatively: Fix current results, verify that future code changes don't break it
- JUnit
  - Standard library for unit testing in Java
  - Uses
    - Java annotations `@Test`
    - Static import `import static`
    - `assertX()` methods
  - Integration into maven
- Requires a good code structure
  - I.e., small units that actually can be tested

Exercise 6

- Issues
  - Construct test data
  - What to test
    - Some regular results – but be specific
    - Borderline cases
- If you skipped the data part, you haven't made your job!

Section 2

Continuous Integration

Introduction

- Automatically run the tests whenever someone changes the code
- Notify responsible person (and/or co-workers)

Introduction

- Automatically run the tests whenever someone changes the code
- Notify responsible person (and/or co-workers)
- Idea: Whenever we *push* to the remote, we want to run the tests
- Different platforms
  - GitLab CI: For your self-hosted server
  - Jenkins: Your own CI server
  - Travis CI: Cloud-based CI, integrates with GitHub
  - GitHub Actions: Most recent addition, CI on GitHub

GitHub Actions

- Controlled by files in `.github/workflows/`
- Specification includes
  - Type of (virtual) machine the code should be run on
  - Prerequisites
  - Actions in the form of links to repos
    - Optionally with parameters
- File format: YAML

YAML Ain't Markup Language (YAML)

- Human readable 'language' to encode arbitrary data
- Indentation used to denote structure
- Typical data structures: Arrays, Dictionaries

Example

```yaml
name: value

# this is a comment

list:
  - element 1
  - element 2

dict:
  key1: value1
  key2:
    embedded dict: value2
```

linter.yml

```yaml
name: Super-Linter

# Run this workflow every time a new commit pushed to your repository
on: push

jobs:
  # Set the job key. The key is displayed as the job name
  # when a job name is not provided
  super-lint:
    # Name the Job
    name: Lint code base
    # Set the type of machine to run on
    runs-on: ubuntu-latest

    steps:
      # Checks out a copy of your repository on the ubuntu-latest machine
      - name: Checkout code
        uses: actions/checkout@v2

      # Runs the Super-Linter action
      - name: Run Super-Linter
        uses: github/super-linter@v3
        env:
          DEFAULT_BRANCH: main
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Small Exercise

- Go into your branch
- Add a file called `.github/workflows/checkout.yml`
- Insert the following (⚠ copying from the slides likely breaks white spaces):

```
1  name: Checkout
2  on: push
3  jobs:
4    checkout:
5      name: demo
6      runs-on: ubuntu-latest
7      steps:
8        - name: Checkout code
9          uses: actions/checkout@v2
10         with:
11           submodules: false
```

- Commit and push

Section 3

Decision Tree Implementation

Decision Tree Implementation

- Next steps:
  - Talk about implementation strategy
  - Implement leaf detection
  - Implement entropy
  - Implement information gain

How a Tree makes a prediction

How a Tree makes a prediction

- Tree is a recursive thing
- We only need to care about local structure

How a Tree makes a prediction

- Tree is a recursive thing
- We only need to care about local structure
- If a tree is a leaf: Return predicted class
- If not: Check a certain attribute value, call predict function of the corresponding sub tree

How a Tree makes a prediction

- Tree is a recursive thing
- We only need to care about local structure
- If a tree is a leaf: Return predicted class
- If not: Check a certain attribute value, call predict function of the corresponding sub tree

```
1  public class Tree {
2    Tree[] children = null;
3    int attributeIndex = -1;
4    double prediction = Double.NaN;
5
6    public double predict(Instance instance) {
7      return 0.0;
8    }
9
10   public boolean isLeaf() {
11     return false;
12   }
13 }
```

Let's implement `double predict(Instance instance)`

Let's implement `double predict(Instance instance)`

```
1 public double predict(Instance instance) {
2   if (this.isLeaf()) {
3     return prediction;
4   } else {
5     int subtree = (int) instance.value(attributeIndex);
6     return children[subtree].predict(instance);
7   }
8 }
```

Section 4

Next Exercise

Exercise 07