Comparing metrics and plots in DVC

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

datacamp

Configuring DVC YAML file

- Configure DVC YAML file to track metrics across experiments
- Change from `outs`

```
stages:
  preprocess:
    ...
  train:
    ...
    outs:
    - metrics.json
    - confusion_matrix.png
```

- To `metrics`

```
stages:
  preprocess:
    ...
  train:
    ...
    outs:
    - confusion_matrix.png
    metrics:
      - metrics.json:
          cache: false
```

Querying and comparing DVC metrics

```
-> dvc metrics show
Path           accuracy    f1_score    precision    recall
metrics.json   0.947       0.8656      0.988        0.7702
```

Change a hyperparameter and rerun `dvc repro`

```
-> dvc metrics diff
Path           Metric     HEAD    workspace    Change
metrics.json   accuracy   0.947   0.9995       0.0525
metrics.json   f1_score   0.8656  0.9989       0.1333
metrics.json   precision  0.988   0.9993       0.0113
metrics.json   recall     0.7702  0.9986       0.2284
```

[1] https://dvc.org/doc/command-reference/metrics

Setting up DVC Github Action

- Add `setup-dvc` GitHub Action
- Replace running Python scripts with DVC pipeline

```
steps:
  ...
  - name: Setup DVC
    uses: iterative/setup-dvc@v1
  - name: Run DVC pipeline
    run: dvc repro
```

Setting up DVC Github Action

```
- name: Write CML report
  env:
    REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  run: |
    # Print metrics of current branch
    dvc metrics show --md >> report.md
    # Compare metrics with main branch
    git fetch --prune
    dvc metrics diff --md main >> report.md
    # Create CML report
    cml comment create report.md
```

Pipeline in action

github-actions bot commented 1 minute ago

Metrics

Path	accuracy	f1_score	precision	recall
metrics.json	0.956	0.95359	0.98261	0.92623

Metrics comparison

Path	Metric	main	workspace	Change
metrics.json	accuracy	0.916	0.956	0.04
metrics.json	f1_score	0.91286	0.95359	0.04072
metrics.json	precision	0.92437	0.98261	0.05824
metrics.json	recall	0.90164	0.92623	0.02459

Plot types in DVC

- `scatter` - scatter plot
- `linear` - interactive linear plot
- `simple` - non-interactive customizable linear plot
- `smooth` - linear plot with smoothing
- `confusion` - confusion matrix
- `confusion_normalized` - confusion matrix with values normalized to <0, 1> range
- `bar_horizontal` - horizontal bar plot
- `bar_horizontal_sorted` - horizontal bar plot sorted by bar size

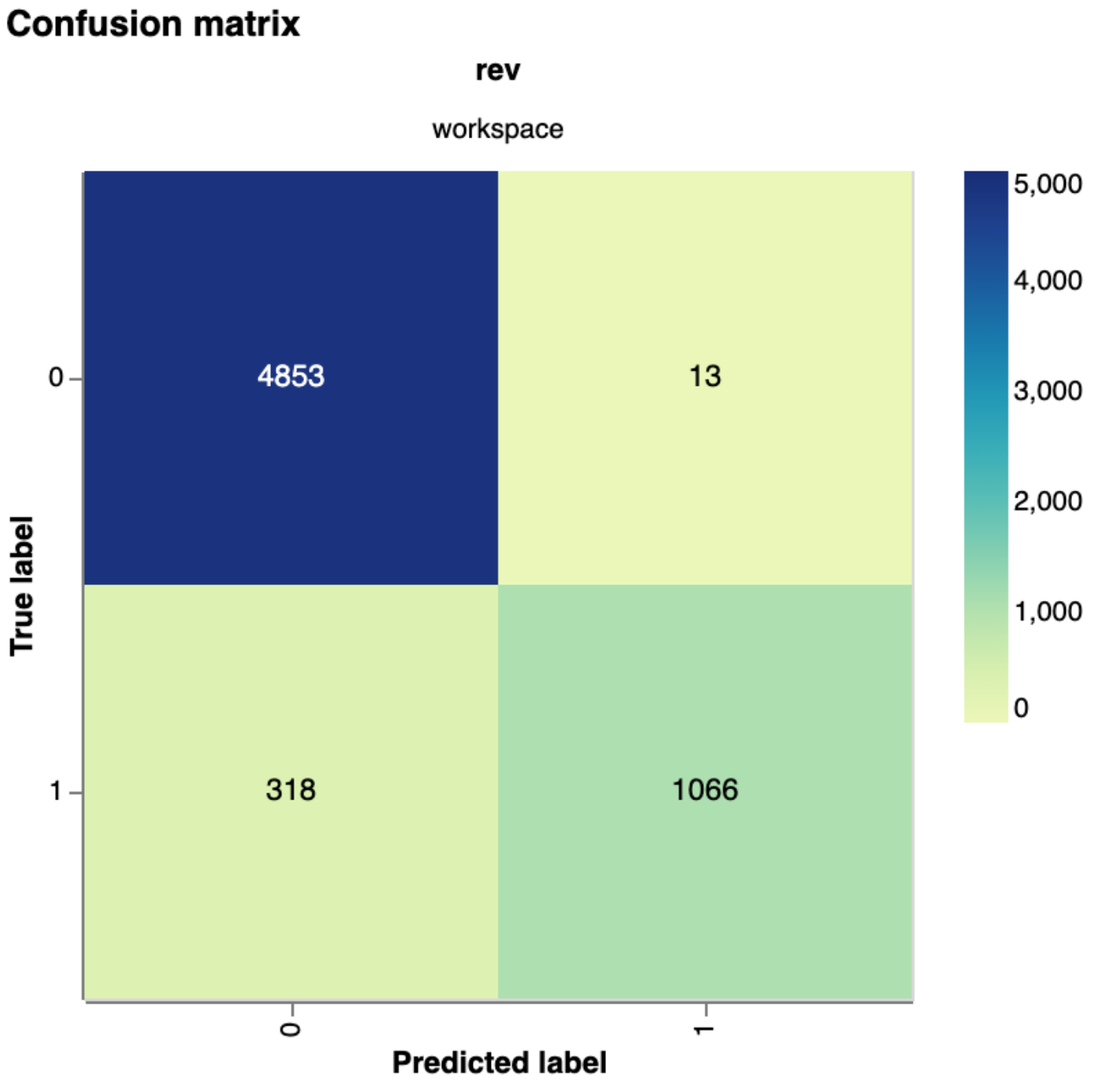
[1] https://dvc.org/doc/user-guide/experiment-management/visualizing-plots#plot-templates-data-series-only

Configuring DVC YAML for plots

```
stages:
  train:
    ...
    plots:
    - predictions.csv: # Name of file containing predictions
        template: confusion # Style of plot
        x: predicted_label # X-axis column name in csv file
        y: true_label # Y-axis column name in csv file
        x_label: 'Predicted label'
        y_label: 'True label'
        title: Confusion matrix
        cache: false # Save in Git
```

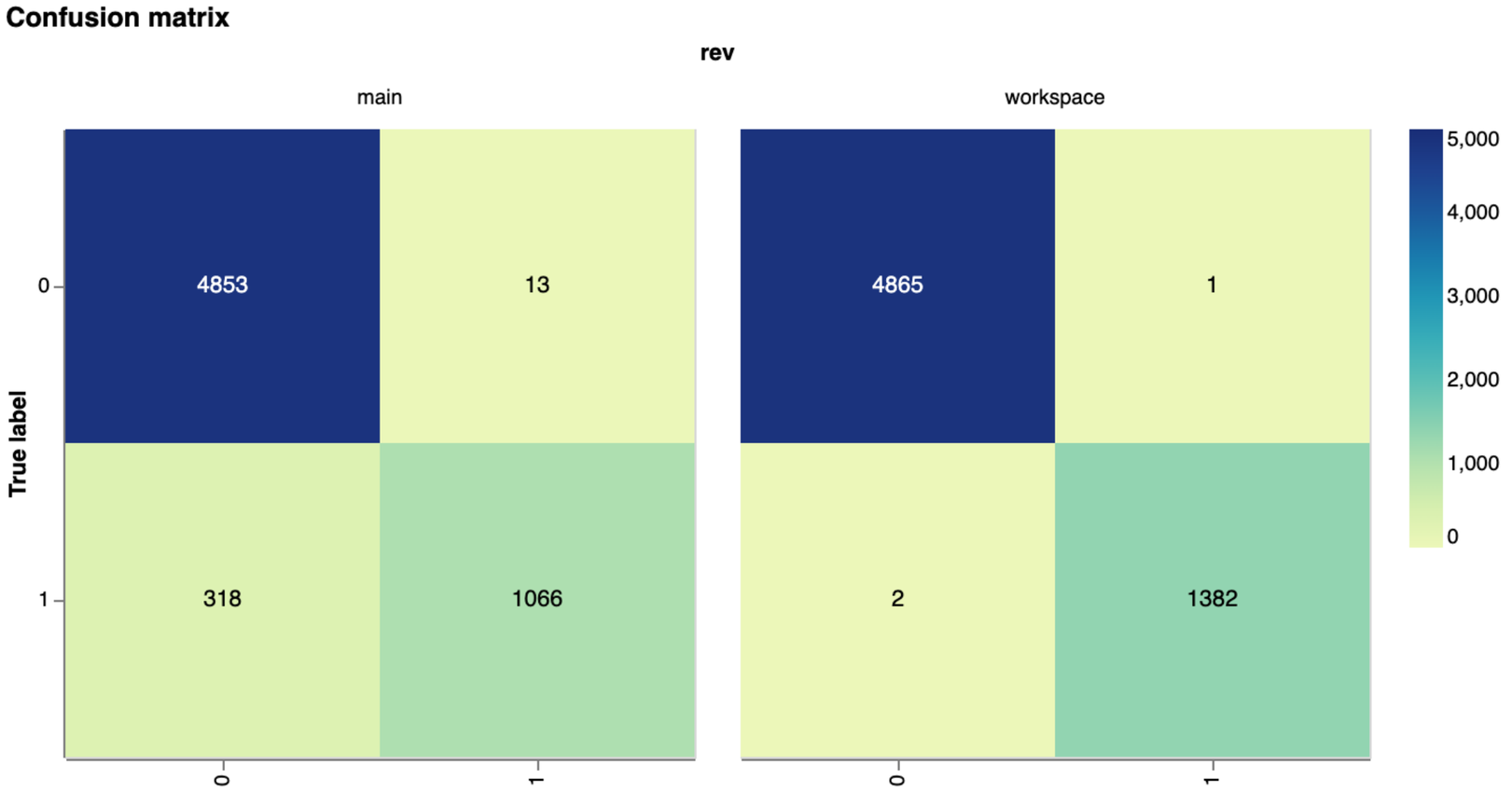
Plotting Confusion Matrix

```
-> dvc plots show predictions.csv
file:///path/to/index.html
```



Comparing Confusion Matrix

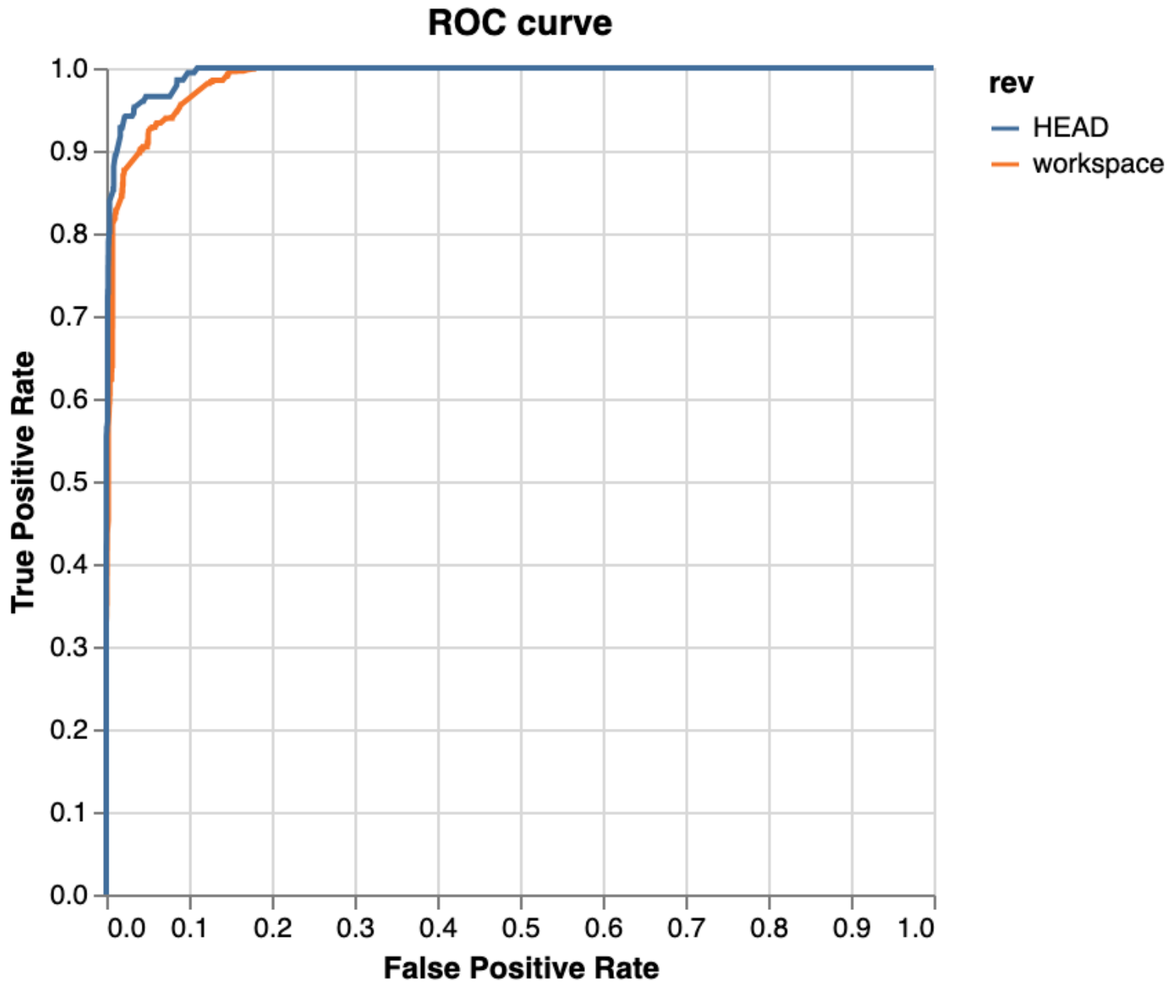
```
-> dvc plots diff --target predictions.csv main
file:///path/to/index.html
```



Comparing ROC Curves

```
# Changes in Python
y_proba = model.predict_proba(X_test)
fpr, tpr, _ = roc_curve(y_test,
                        y_proba[:, 1])
```

```
# Changes in dvc.yaml
plots:
- roc_curve.csv:
    template: simple
    x: fpr
    y: tpr
    x_label: 'False Positive Rate'
    y_label: 'True Positive Rate'
    title: ROC curve
    cache: false
```



Let's practice!

CI/CD FOR MACHINE LEARNING

Hyperparameter Tuning with DVC

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

datacamp

Hyperparameter tuning workflow

- Hyperparameter tuning
  - Input: Parameter sweep ranges
  - Output: Best parameters
- Training
  - Input: Best parameters
  - Output: Metrics and plots (already covered)
- Loose coupling for independent training
  - Hyperparameter tuning sufficient but not necessary
- Both jobs are dataset dependent

```
# Contents of hp configuration
{
    "n_estimators": [2, 4, 5],
    "max_depth": [10, 20, 50],
    "random_state": [1993]
}
```

```
# Contents of best parameters
{
    "n_estimators": 5,
    "max_depth": 20,
    "random_state": 1993
}
```

Training code changes

Changes in Python Hyperparameter tuning script

```
# Load hyperparameters from the JSON file
with open("rfc_best_params.json", "r") as params_file:
  rfc_params = json.load(params_file)

# Define and train model
model = RandomForestClassifier(**rfc_params)
model.fit(X_train, y_train)
```

Hyperparameter Tuning with GridSearch

```
# Define the model and hyperparameter search space
model = RandomForestClassifier()
param_grid = json.load(open("hp_config.json", "r"))

# Perform GridSearch with five fold CV
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
with open("rfc_best_params.json", "w") as outfile:
        json.dump(best_params, outfile)
```

DVC YAML changes

Hyperparameter Tuning

```
stages:
  preprocess: ...
  train: ...
  hp_tune:
    cmd: python hp_tuning.py
    deps:
    - processed_dataset/weather.csv
    - hp_config.json
    - hp_tuning.py
    outs: # Not tracking best parameters
      - hp_tuning_results.md:
          cache: false
```

Training

```
stages:
  preprocess: ...
  hp_tune: ...
  train:
    cmd: python train.py
    deps:
    - processed_dataset/weather.csv
    - rfc_best_params.json # Best parameters
    - train.py
    metrics:
      - metrics.json:
          cache: false
```

Triggering individual stages

- Stages can be triggered independently `dvc repro <stage_name>`
- Force run hyperparameter tuning stage `dvc repro -f hp_tune`
  - Ensures best parameter file will update
- Training can be run with `dvc repro train`
- Both stages trigger preprocessing step as dependency

Hyperparameter Run Output

mean_test_score	std_test_score	max_depth	n_estimators	random_state
0.999733	0.000413118	20	5	1993
0.999307	0.000574418	50	5	1993
0.99888	0.000617378	10	5	1993
0.997813	0.00117333	10	4	1993

Changes in Python hyperparameter tuning script

```
# Save the results of hyperparameter tuning
cv_results = pd.DataFrame(grid_search.cv_results_)
markdown_table = cv_results.to_markdown(index=False)
with open("hp_tuning_results.md", "w") as markdown_file:
  markdown_file.write(markdown_table)
```

Summary

- Hyperparameter tuning route
  - Branch name `hp_tune/<some-string>`
  - Make changes to search configuration
  - Manually open a PR
    - Force runs DVC pipeline `dvc repro -f hp_tune`
    - Uses `cml pr create` to create a new training PR with best parameters
  - Force push a commit to training PR to kick off model training job
- Manual route
  - Branch name `train/<some-string>`
  - Edit best parameters file and commit changes
  - Manually open a PR to kick off model training job

Let's practice!

CI/CD FOR MACHINE LEARNING

GitHub Actions workflow for Hyperparameter Tuning

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Instructor

datacamp

Branching workflow

- Separate feature branches for training and hyperparameter tuning
  - Intended job should trigger
  - Other job should not trigger
  - Implemented using `if` condition
- Hyperparameter tuning
  - Print statistics table for analysis
  - Automatically open a new PR with parameter changes
- Training
  - Read new parameter file in training PR

Setting conditionals

Hyperparameter Tuning

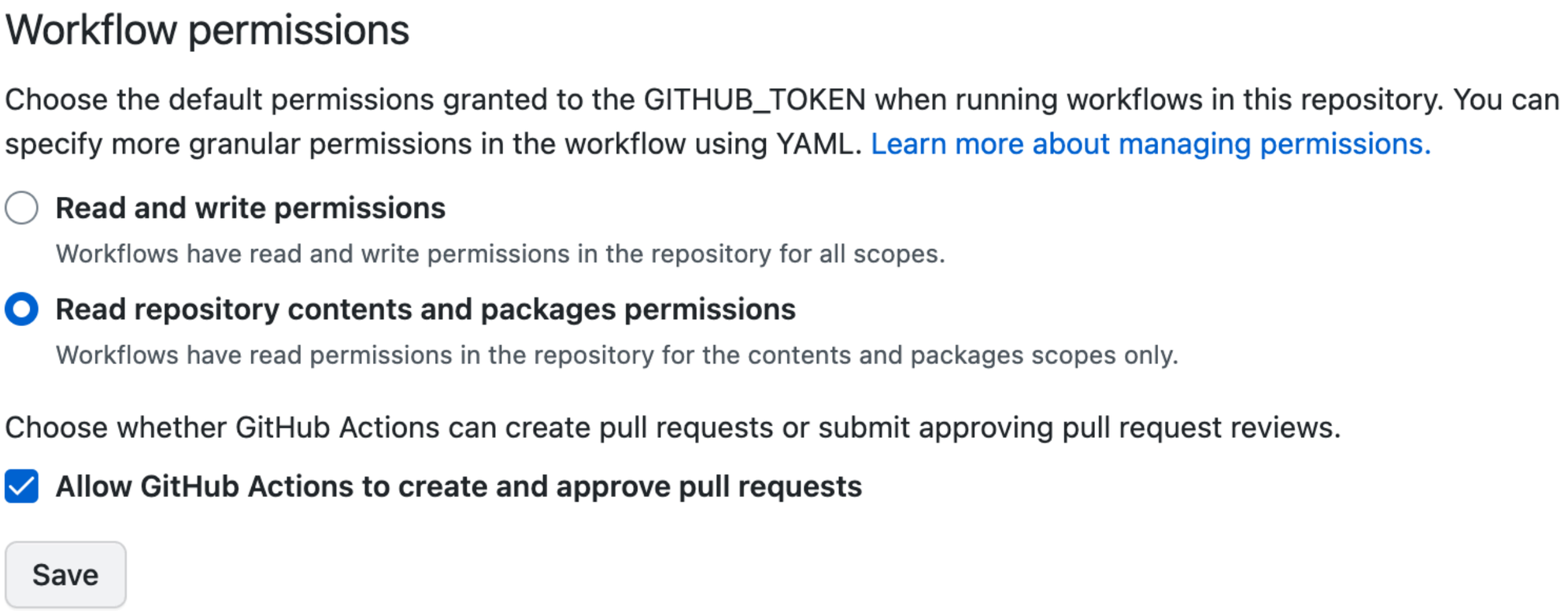
```
jobs:
  hp_tune_and_publish_report:
    # Run when branch name starts with hp_tune/
    if: startsWith(github.head_ref, 'hp_tune/')
    steps:
      ...
      - name: |
          DVC pipeline for hyperparameter tuning
        run: dvc repro -f hp_tune
```

Training

```
jobs:
  train_and_publish_report:
    # Run when branch name starts with train/
    if: startsWith(github.head_ref, 'train/')
    steps:
      ...
      - name: Run DVC pipeline for training
        run: dvc repro train
```

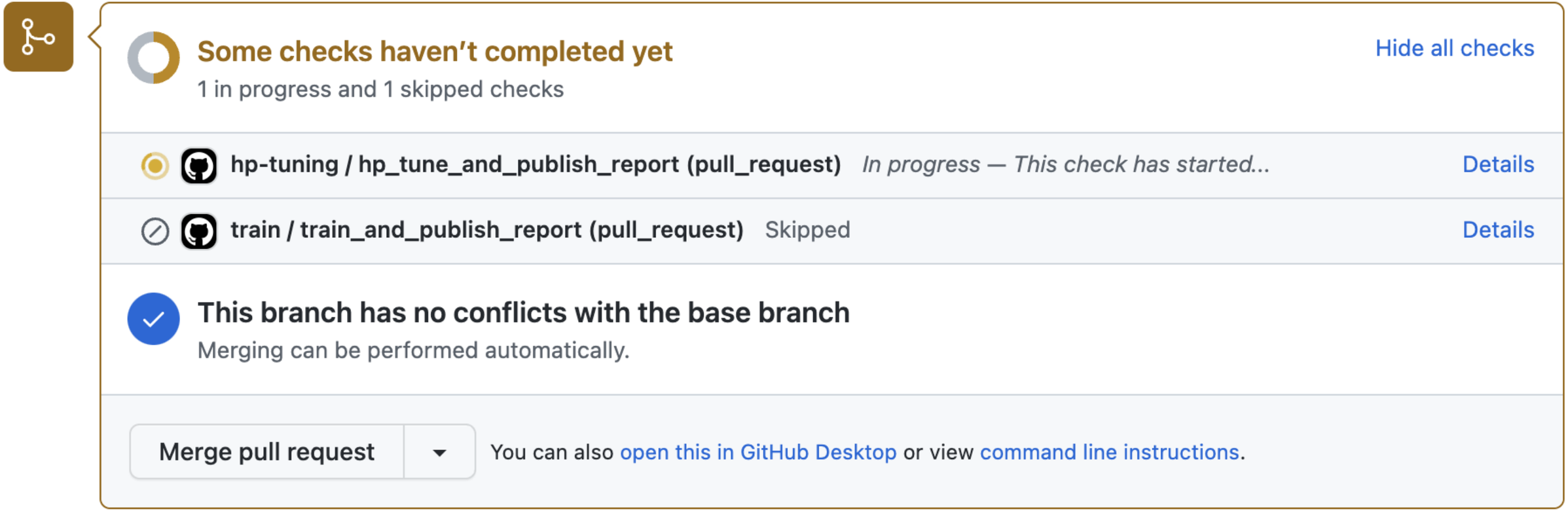
Setup workflow permissions

Repository Settings > Actions > General



Hyperparameter tuning job kickoff

- Make sure to prefix branch name with `hp_tune/`



Hyperparameter tuning job metrics

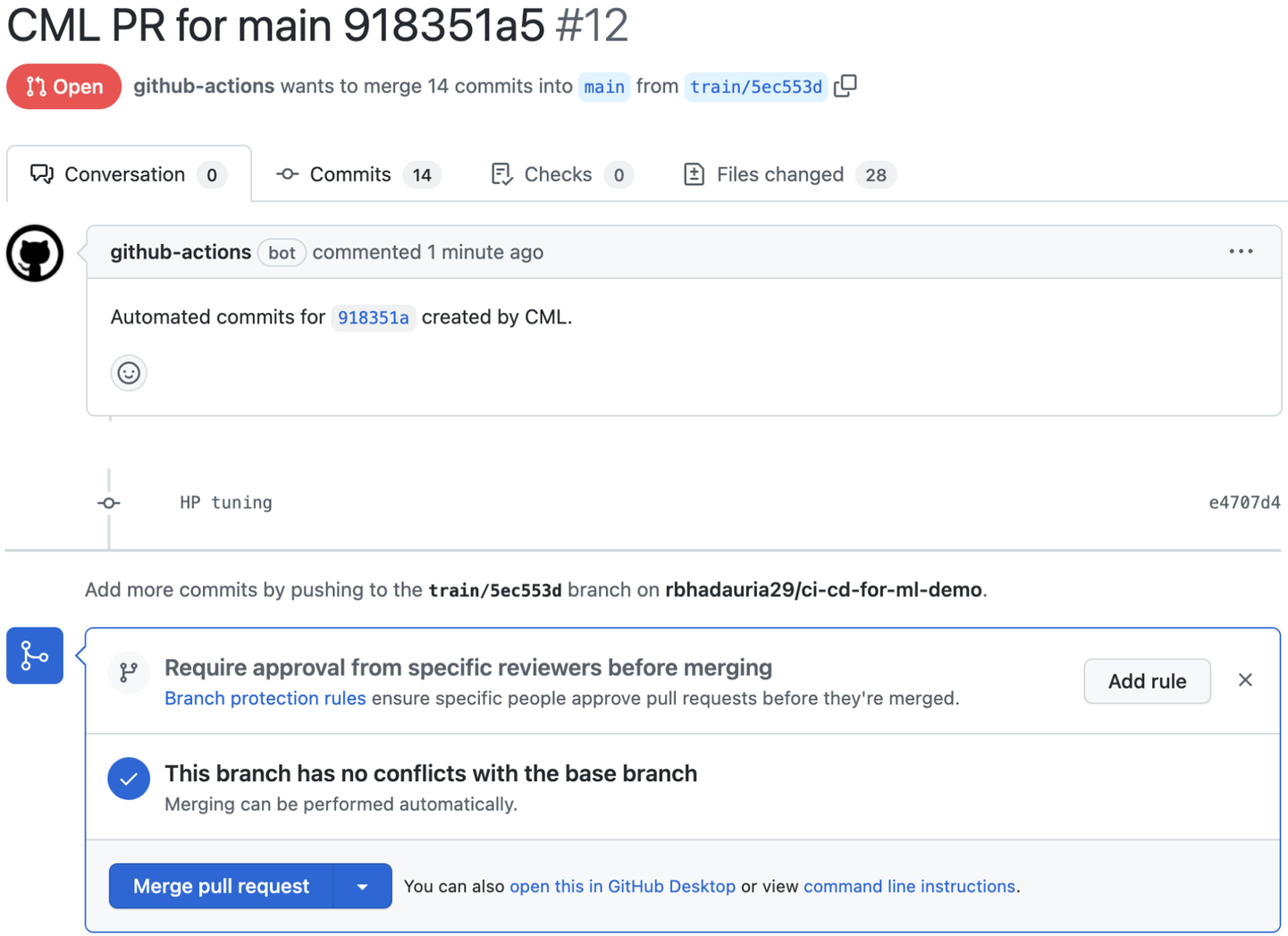
github-actions bot commented now

rank_test_score	mean_test_score	std_test_score	max_depth	n_estimators	random_state
1	0.999733	0.000413118	20	5	1993
2	0.999307	0.000574418	50	5	1993
3	0.99888	0.000617378	10	5	1993
4	0.997813	0.00117333	10	4	1993
5	0.997173	0.0011997	20	4	1993
6	0.996107	0.00184444	50	4	1993
7	0.982613	0.00441863	10	2	1993
8	0.974187	0.00522122	20	2	1993
9	0.972907	0.00835412	50	2	1993

Creating a training PR from hyperparameter run

```
steps:
  - name: Create training branch
    env:
      REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
    run: |
      # Branch name begins with train/
      export BRANCH_NAME=train/$(git rev-parse --short "${{ github.sha }}")
      # Create PR for training
      cml pr create \
        --user-email hp-bot@cicd.ai \
        --user-name HPBot \
        --message "Hyperparameter tuning" \
        --branch $BRANCH_NAME \
        --target-branch main \
          rfc_best_params.json
```

New training branch PR



New training branch PR

HP tuning

train/5ec553d (#12)

HPBot committed 13 minutes ago

2 rfc_best_params.json

```
...   ...   @@ -1 +1 @@
  1         - {"max_depth": 15, "n_estimators": 1, "random_state": 1993}
        1   + {"max_depth": 20, "n_estimators": 5, "random_state": 1993}
```

Starting training run manually

- `GITHUB_TOKEN` cannot trigger workflows on self created PRs
  - Prevention from recursive runs
- Workarounds
  - Use a Personal access token with proper permissions

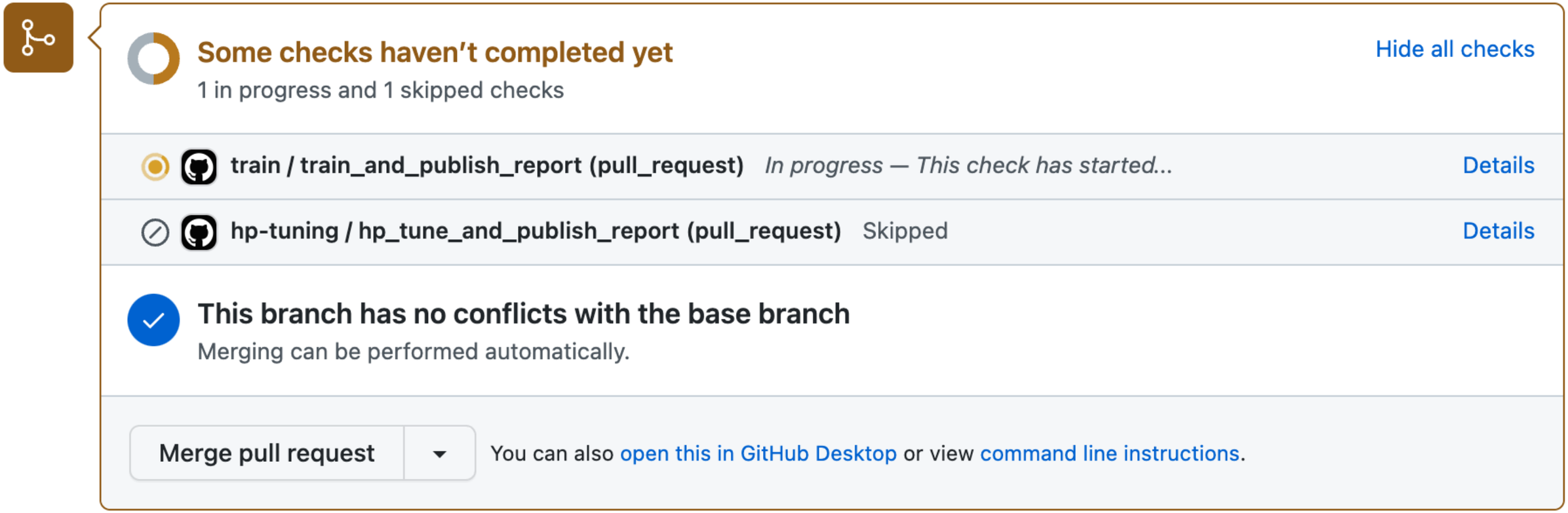
```
steps:
  - env:
      GITHUB_TOKEN: ${{ secrets.MY_TOKEN }}
```

  - Run training job right after hyperparameter tuning in GHA pipeline
  - Force push the code to trigger a run (forces inspection)

```
-> git checkout train/1f34fs
-> git commit --amend --no-edit && git push -f
```

[1] https://docs.github.com/en/actions/using-workflows/triggering-a-workflow#triggering-a-workflow-from-a-workflow

Training job kickoff



Let's practice!

CI/CD FOR MACHINE LEARNING

Congratulations!

CI/CD FOR MACHINE LEARNING

Ravi Bhadauria
Machine Learning Engineer

datacamp

YAML Syntax

- Indentation
- Mappings `a: 1`
- Arrays
  - Flow `[1, 2]`
  - Block

```
- 1
- 2
```

- Multi-line strings
  - Style indicators (`|` , `>`)
  - Chomping indicators (`-` , `+`)

GitHub Actions

- Workflow (pipeline)
- Events (`on`)
- Jobs (`jobs`)
- Runners (`runs-on`)
- Steps (`steps`)
- Contexts
- Secrets and environment variables (`GITHUB_TOKEN`)
- Actions (`checkout` , `setup-python`)
  - CML: `cml comment create` , `cml pr create`

Versioning data and building reproducible pipelines

- `dvc init`
- DVC remotes (including local)
- `dvc push` , `dvc pull`
- `dvc repro <target>`
- `dvc metrics show/diff`
- `dvc plots show/diff`

- DVC YAML (`dvc.yaml`)
  - Steps or targets
  - Commands (`cmd`)
  - Dependencies (`deps`)
  - Outputs (`outs`)
  - Metrics (`metrics`)
  - Plots (`plots`)

Datacamp resources

- Courses
  - Developing Machine Learning Models for Production
  - MLOps Deployment and Life Cycling
  - Fully Automated MLOps
  - Introduction to DevOps
- Blogs
  - Version Control For Data Science

Further reading

CI/CD and branching patterns

DevOps vs MLOps

Data Version Control

Model registry and life cycle management

Thank you!

CI/CD FOR MACHINE LEARNING