Managing Machine Learning Experiments

Dr. Rutu Mulkar

About me:

BS, MS, PhD in Computer Science

Ph.D. thesis in Natural Language Processing

Founder of AI company - Ticary Solutions - sold to Sigmoidal in 2019

Contributor to IBM Watson that defeated humans in Jeopardy!

Previously worked at:

Pricewaterhouse Coopers, IBM, Moz and a Healthcare company

Outline: Managing Machine Learning Experiments

- Typical ML pipeline
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- Open Source Software
- Paid Software
- Conclusion

Outline

- **Typical ML pipeline**
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- Open Source Software
- Paid Software
- Conclusion

Consider this scenario

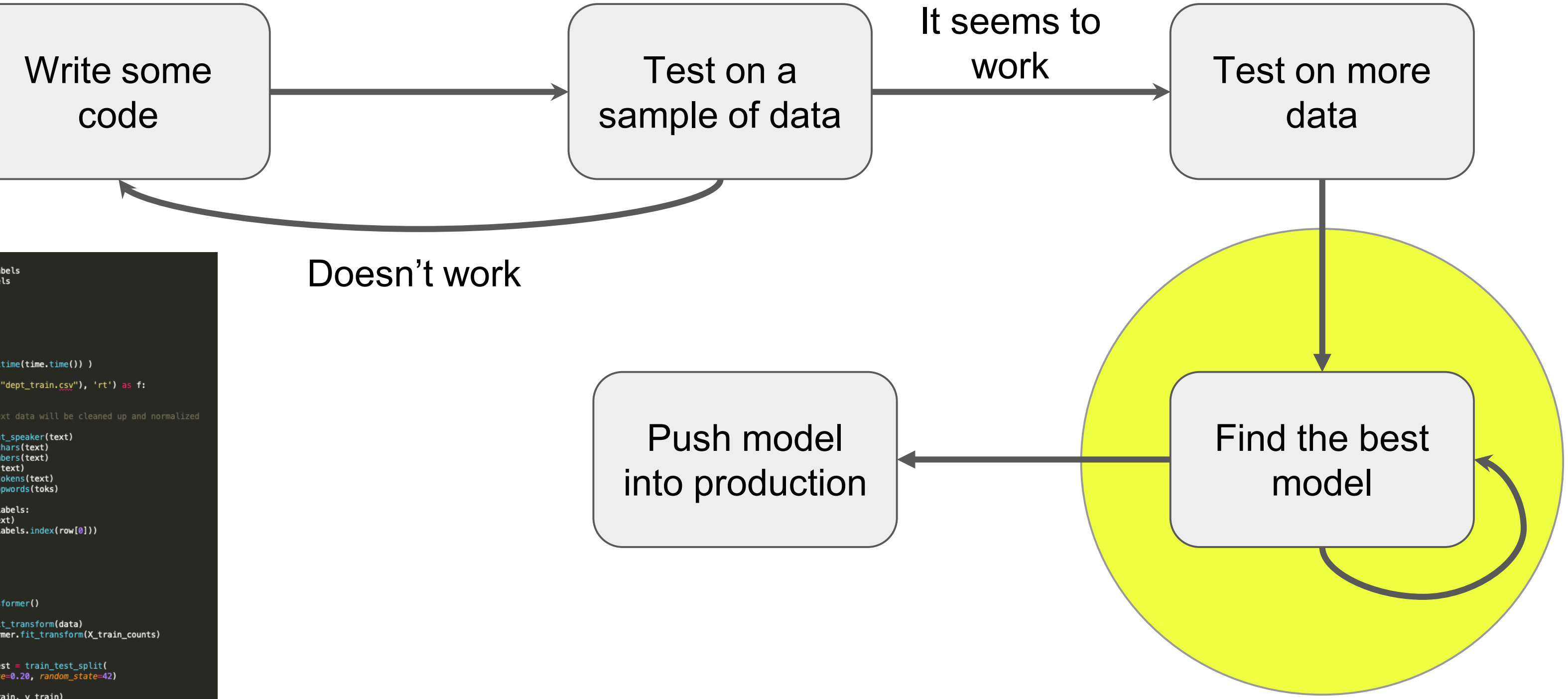
You are a data scientist

You are provided with 10,000 samples of conversational data

You are asked to build a classifier to classify a question/sentence by intent

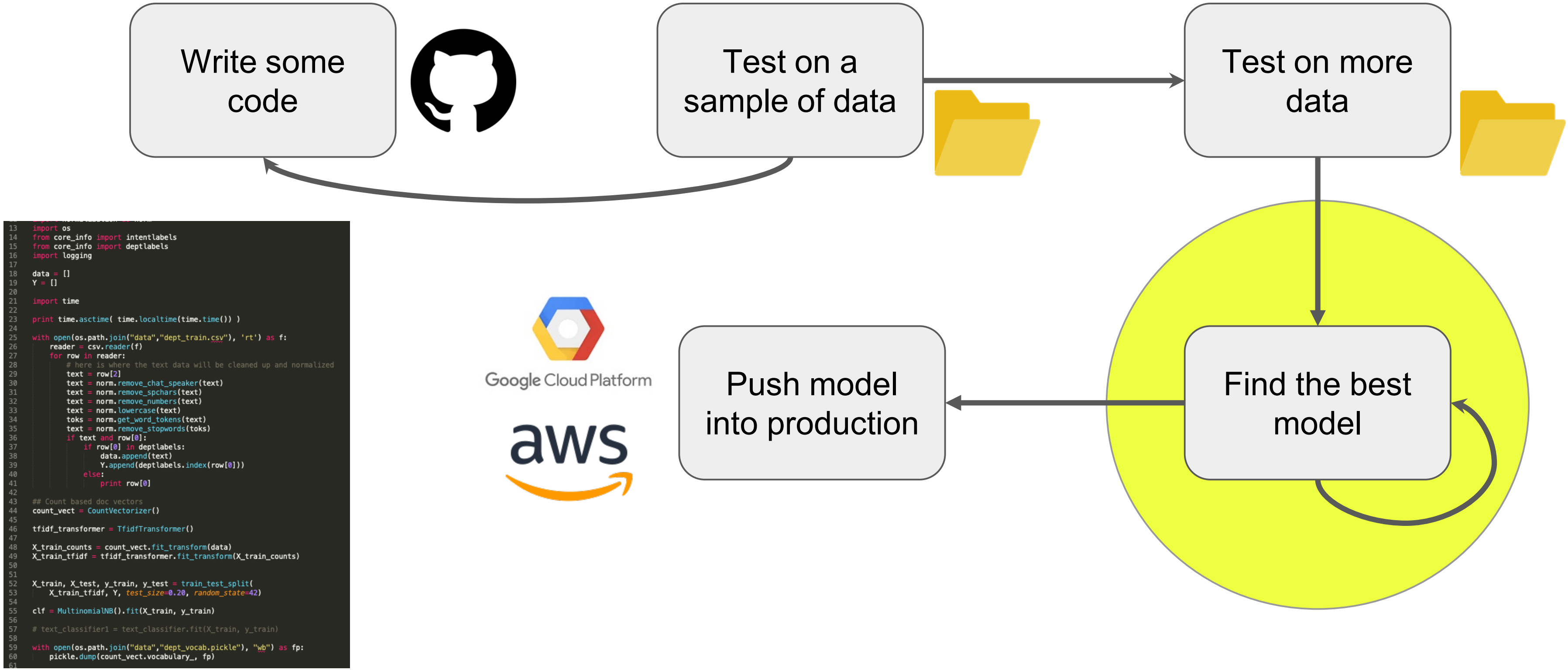
- What time is it?
- Can you tell me the time?
- Set an alarm for 8:00PM
- Can you remind me at 8:00PM?

A typical ML workflow



```python
import os
from core_info import intentlabels
from core_info import deptlabels
import logging

data = []
Y = []

import time

print time.asctime( time.localtime(time.time()) )

with open(os.path.join("data","dept_train.csv"), 'rt') as f:
    reader = csv.reader(f)
    for row in reader:
        # here is where the text data will be cleaned up and normalized
        text = row[2]
        text = norm.remove_chat_speaker(text)
        text = norm.remove_spchars(text)
        text = norm.remove_numbers(text)
        text = norm.lowercase(text)
        toks = norm.get_word_tokens(text)
        text = norm.remove_stopwords(toks)
        if text and row[0]:
            if row[0] in deptlabels:
                data.append(text)
                Y.append(deptlabels.index(row[0]))
            else:
                print row[0]

## Count based doc vectors
count_vect = CountVectorizer()

tfidf_transformer = TfidfTransformer()

X_train_counts = count_vect.fit_transform(data)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)


X_train, X_test, y_train, y_test = train_test_split(
    X_train_tfidf, Y, test_size=0.20, random_state=42)

clf = MultinomialNB().fit(X_train, y_train)

# text_classifier1 = text_classifier.fit(X_train, y_train)

with open(os.path.join("data","dept_vocab.pickle"), "wb") as fp:
    pickle.dump(count_vect.vocabulary_, fp)
```

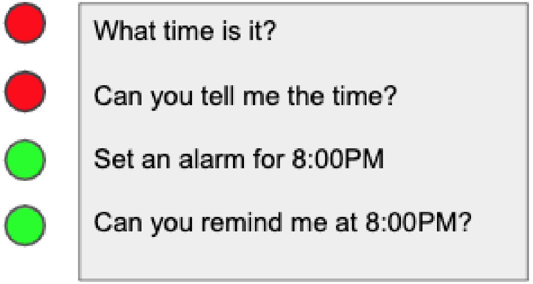
A typical ML workflow




```python
import os
from core_info import intentlabels
from core_info import deptlabels
import logging

data = []
Y = []

import time

print time.asctime( time.localtime(time.time()) )

with open(os.path.join("data","dept_train.csv"), 'rt') as f:
    reader = csv.reader(f)
    for row in reader:
        # here is where the text data will be cleaned up and normalized
        text = row[2]
        text = norm.remove_chat_speaker(text)
        text = norm.remove_spchars(text)
        text = norm.remove_numbers(text)
        text = norm.lowercase(text)
        toks = norm.get_word_tokens(text)
        text = norm.remove_stopwords(toks)
        if text and row[0]:
            if row[0] in deptlabels:
                data.append(text)
                Y.append(deptlabels.index(row[0]))
            else:
                print row[0]

## Count based doc vectors
count_vect = CountVectorizer()

tfidf_transformer = TfidfTransformer()

X_train_counts = count_vect.fit_transform(data)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)


X_train, X_test, y_train, y_test = train_test_split(
    X_train_tfidf, Y, test_size=0.20, random_state=42)

clf = MultinomialNB().fit(X_train, y_train)

# text_classifier1 = text_classifier.fit(X_train, y_train)

with open(os.path.join("data","dept_vocab.pickle"), "wb") as fp:
    pickle.dump(count_vect.vocabulary_, fp)
```

How to find the best model

- Add new features to your code
  - Adding keywords like “time” indicate the red class
  - Adding keywords like - “reminder”, “alarm” signify the green class
  - Maybe use Word2Vec to get keywords similar to “time” and “alarm” (link)
  - Use an n-gram word vectorizer
- Change the Algorithm
  - Naive Bayes (baseline)
  - Random Forests
  - Recurrent Neural Networks
- Change hyperparameters
  - Change the # of words, and # of chars
  - Change the values or other values in built in functions
- Pipeline changes
  - [remove sp chars, word chunking]
  - [word chunking, remove sp chars]



We might record each code change in git

but

how can we record each training and testing run?

Outline

- Typical ML pipeline
- **How we are managing ML projects right now**
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- Opensource Software
- Paid Software
- Conclusion

Documenting ML Experiments

Experiment ID

A	B	C	D	E	F	G
Experiment ID	**Accuracy**	**Precision 0.5**	**Recall 0.5**	**F1-0.5**	**Additional Details**	**Notes**
0_jan_feb_output	0.436416185	0.7008778836	0.2598066296	0.3790895308	https://drive.google.com/	Baseline scores - score to beat
1_tfidfoutput1	0.4454479769	0.4488333081	0.1092543365	0.1757322018	https://drive.google.com/	2000 tfidf char grams of size 1-2
2_tfidfoutput2	0.4738078035	0.574528196	0.1238582369	0.2037841689	https://drive.google.com/	tfidf char grams 3-4, max features 5000
3_preproc_output	0.4501445087	0.6401523854	0.08468742423	0.1495857593	https://drive.google.com/	Hashing vectorizer with preprocessing
4_tfidf	0.4875361272	0.732219383	0.09915287236	0.1746549865		2500 char grams 3-4 grams, 2500 Word grams 1-2 grams, TFIDF vectorizer
5_tfidf_5000	0.4985549133	0.7526197727	0.1025996849	0.1805818398		5000 char grams 3-4 grams, 5000 Word grams 1-2 grams, TFIDF vectorizer
6_tfidf_10000	0.5028901734	0.7642833247	0.1065390523	0.1870094826		10000 char grams 3-4 grams, 10000 Word grams 1-2 grams, TFIDF vectorizer
6_tfidf_10000_2	0.5036127168	0.7564419212	0.1029915207	0.1812987487		normalization error fixed from previous experiment
7_tfidf_svm	0.4367774566	0.5886689759	0.135238904	0.2199477291		SVM classifier, aggressive preprocessing
8_tfidf_15000_LR	0.492232659	0.7439588317	0.09816293454	0.1734409085		TFIDF, 15000 features
9_tfidf_10000_LG	0.4878973988	0.7624622412	0.09266619797	0.1652488065		char grams from 3-6 chars, preprocessing
10_tfidf_NB	0.2881141619	0.4590017825	0.004422918115	0.008761411707		Naive Bayes model

Tracking ML Experiments

How to reproduce experiments?

- Make copies of each run that you run
- Commit each code change to git and tag it

Other things:

- Trained model
- Outputs
- Environment Variables

Tracking ML Experiments: Problems

- Separation of code, outputs and metrics
- No side by side comparison of code
- No side by side comparison of metrics
- Missing storage of
    - Trained models
    - Artifacts
    - Outputs
    - Environment variables

We are usually not very deliberate and disciplined about storing each of these artifacts

I am not the only one with this issue

Some verbatim comments from others in the community

Industry

1. “I just don’t understand how they got that result”
2. “I thought I used the same parameters but I’m getting different results”
3. “I can’t remember which version of the code I used to generate Figure 6”
4. “The new employee wants to reuse that analysis I published three years ago but he can’t reproduce the figures”
5. “It worked yesterday”

Academia

1. “The Materials and Methods section doesn’t explain how the results were normalized”
2. “The description in the Results section is different from what’s in the Supplementary Material”
3. “I’m sure I’m using the same parameters as the original authors, but I just can’t get it to work”

Reproducing code is hard!!

- What code was run?
    - Which script?
        - name, location, version
        - options, parameters
        - dependencies (name, location, version)
    - What were the input data?
    - What were the outputs?
- Machine name(s), other identifiers (e.g. IP addresses)
- Processor architecture
- Available memory
- Operating system
    - why was it run?
    - what was the outcome?
    - which project was it part of
- Data, logs, stdout/stderr
    - who launched the computation?
    - when was it launched/when did it run? (queueing systems)
    - where did it run?

This talk

- Typical ML pipeline
- How we are managing ML projects right now
- **How ML workflow is different from Software Engineering**
- Tools to manage your ML experiments better
- Opensource Software
- Paid Software
- Conclusion

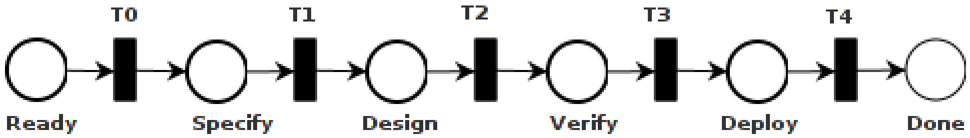


Image Credit: Lean Software Engineering

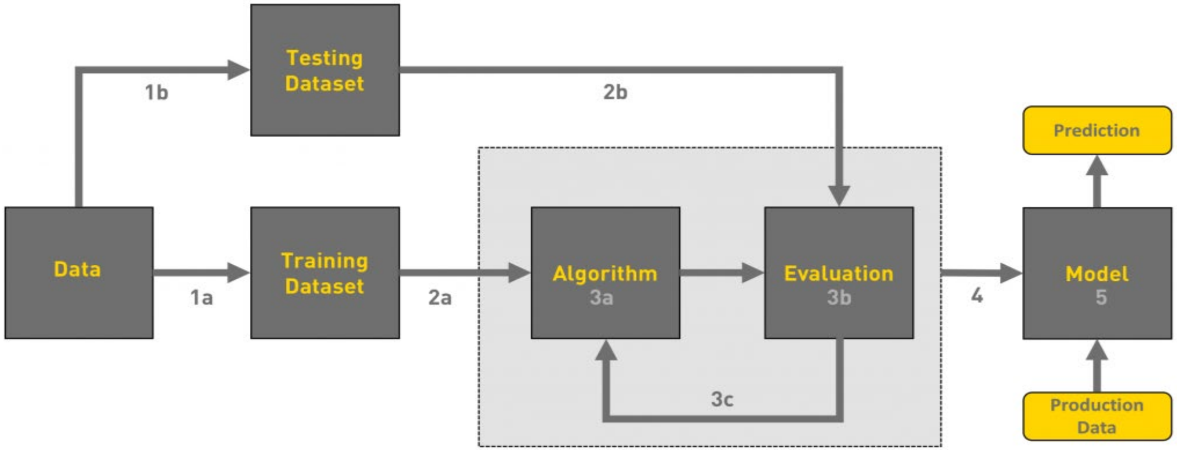


Image Credit: Towards Data Science

Outline

- Typical ML pipeline
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- **Tools to manage your ML experiments better**
- Opensource Software
- Paid Software
- Conclusion

Write Robust and Reproducible Code

1. Make your code modular
2. Write tests for your code
3. Work in virtual environments
4. Design your code to be easily understood by others (where “others” can also include “yourself in-six-months-time”)

Make the code modular

Rule of thumb: If you find yourself repeating/duplicating any code, then make it into its own function.

E.g. file opening and reading in a json or a csv

```
def train():
    filename = "train.json"
    file = open(filename, "r")
        data = json.load(file)
    train_data(data)


def test():
    filename = "test.json"
    file = open(filename, "r")
        data = json.load(file)
    test_data(data)
```

```
def get_data_from_json(filename):
    file = open(filename, "r")
        data = json.load(file)
    return(data)

def train():
    data = get_data_from_file("train.json")
    train_data(data)


def test():
    data = get_data_from_file("test.json")
    test_data(data)
```

Automated Testing

- For all the tests you were performing manually before, write scripts
- Gives you confidence that your code is doing what you think it is doing
- Frees you to make wide-ranging changes to the code (for the purposes of optimization or making the code more robust, for example) without worrying that you will break something
- If you do break something, your tests will tell you immediately and you can undo the change.

Automated testing

Initial time investment

- if you already perform manual, informal testing, this time will be paid back the first or second time you run the automated suite of tests.
- Even if you did no testing at all previously, the loss of fear of changing code will lead to more rapid progress.

Write tests

> nosetests

> nosetests --with-coverage --cover-package=<package name>

```
(env) rutus-mbp:air_cli rutumulkar$ nosetests --with-coverage --cover-package=air-cli
.
Coverage.py warning: Module air-cli was never imported. (module-not-imported)
Name                     Stmts   Miss  Cover
--------------------------------------------
__init__.py                  2      0   100%
experiment_manager.py       33      9    73%
libems.py                  310    216    30%
templates.py                 3      0   100%
tests/test_air.py           11      0   100%
--------------------------------------------
TOTAL                      359    225    37%
----------------------------------------------------------------------
```

Work with a virtual environment

Advantages - reproducibility

Python3

Have a requirements.txt file to know the library dependencies

```
python3 -m venv env

. ./env/bin/activate

> pip install requirements.txt

> deactivate
```

requirements.txt

```
pip freeze

# will tell you all the python modules on your system

# most of which you don't need

Pip freeze | grep <name>

# get a specific package
```

Sample requirements.txt

```
GitPython==2.1.11
sklearn==0.0
scikit-learn==0.21.3
nose==1.3.7
coverage==4.5.4
```

Ensure your code can be easily understood by others

- Write comments explaining anything slightly complex, or why a particular choice was made
- Write clear documentation

Outline

- Typical ML pipeline
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- **Open Source Software**
- Paid Software
- Conclusion

Experiment Management Systems

Guild	Command Line	guild run train.py	https://guild.ai/
Sumatra	Command Line	smt run --executable=python --main=main.py	https://pythonhosted.org/Sumatra/
StudioML	Command Line	studio run train_mnist_keras.py	https://github.com/studioml/studio
Datmo	Immersive	import datmo	https://github.com/datmo/datmo
Modelchimp	Immersive	**from** modelchimp **import** Tracker	https://modelchimp.com/
MLFlow	Immersive	**import** mlflow **import** mlflow.sklearn	https://mlflow.org/
Sacred	Immersive	from sacred import Experiment	https://github.com/IDSIA/sacred

Command Line

- Independent of Programming Language
- High freedom to code like you already do - without changes in your workflow
- Need to learn new command line tools

Immersive

- Highly tied to the programming language
- Need to edit your workflow a little bit to fit to the platform/software
- Need to learn an API

This talk

- Typical ML pipeline
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- Open Source Software
- **Paid Software**
- Conclusion

Paid Tools

Comet.ml	https://www.comet.ml/	Immersive	Startups, Individuals
Neptune.ml	https://neptune.ml/	Immersive	Startups, Individuals
Weights and Biases	https://www.wandb.com/	Immersive	Startups, Individuals
Determined AI	https://determined.ai/	Immersive + Command Line	Enterprises
Dot Science	https://www.dotscience.com/	Immersive	Startups, Individuals (Focus on Jupyter Notebooks)

Outline

- Typical ML pipeline
- How we are managing ML projects right now
- How ML workflow is different from Software Engineering
- Tools to manage your ML experiments better
- Open Source Software
- Paid Software
- **Conclusion**

Conclusion

ML process doesn't have predefined guidelines

Best to practice Software Engineering design principles to write robust code

- Write modular code
- Write test cases
- Use Virtual environments (always)

When you are ready, start using open source tools for managing machine learning experiments and models.

Once you are done using open source tools - start looking into paid software to manage ML experiments

Thank you!
Questions?

@rutumulkar