22_deployment-conclusion

June 25, 2022

CPSC 330
Applied Machine Learning

1 Lecture 22: Deployment and conclusion

UBC 2022 Summer

Instructor: Mehrdad Oveisi

1.1 Imports

[1]:
```
import joblib
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_validate, train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
```

1.2 Lecture outline

- Announcements
- Model deployment (30 min)
- Instructor/TA evaluations + Break (10 min)
- Review / conclusion (20 min)

1.3 Learning objectives

- Describe the goals and challenges of model deployment.

1.4 Announcements

- Last lecture today!
- hw8 is due tonight
- We will take time for formal course evaluations in this lecture.

Final exam

- Our final exam is on **Dec. 19th at noon at PHRM 1101**.
- Cummulative
- Open book. You can refer to the notes.
- Please bring your computer and charger. All of it is going to be on Canvas.
- A combination of: Multiple choice questions, short answer questions, simple coding questions
- No communication/collaboration
- No public Piazza posts

- More details will be posted on Piazza.

1.5 Model deployment (30 min)

Attribution This material adapted from the model deployment tutorial by Tomas Beuzen.

1.5.1 Try out this moment predictor

https://moment-type-predictor.herokuapp.com/

- In this lecture I will show you how to set up/develop this.

What is deployment?

- After we train a model, we want to use it!
- The user likely does not want to install your Python stack, train your model.
- You don't necessarily want to share your dataset.
- So we need to do two things:

1. Save/store your model for later use.
2. Make the saved model conveniently accessible.

We will use Joblib for (1) and Flask & Heroku for (2).

Requirements (I already did these)

- Heroku account. Register here.
- Heroku CLI. Download here.

More python installations:

```
pip install Flask
pip install Flask-WTF
pip install joblib
```

1.6 Demo: Deploying moment classification model

1.6.1 Building a model

Recall the multi-class classification problem using the HappyDB corpus.

[2]:
```
df = pd.read_csv("data/cleaned_hm.csv", index_col=0)
sample_df = df.dropna()
```

```
sample_df.head()
sample_df = sample_df.rename(
    columns={"cleaned_hm": "moment", "ground_truth_category": "target"}
)
sample_df.head()
```

[2]:
```
        wid reflection_period  \
hmid
27676   206              24h
27678    45              24h
27697   498              24h
27705  5732              24h
27715  2272              24h

                                              original_hm  \
hmid
27676  We had a serious talk with some friends of our…
27678                            I meditated last night.
27697  My grandmother start to walk from the bed afte…
27705  I picked my daughter up from the airport and w…
27715        when i received flowers from my best friend

                                                  moment  modified  \
hmid
27676  We had a serious talk with some friends of our…      True
27678                            I meditated last night.      True
27697  My grandmother start to walk from the bed afte…      True
27705  I picked my daughter up from the airport and w…      True
27715        when i received flowers from my best friend      True

       num_sentence     target predicted_category
hmid
27676             2    bonding            bonding
27678             1    leisure            leisure
27697             1  affection          affection
27705             1    bonding          affection
27715             1    bonding            bonding
```

[3]:
```
sample_df["target"].value_counts()
```

[3]:
```
affection           4810
achievement         4276
bonding             1750
enjoy_the_moment    1514
leisure             1306
nature               252
exercise             217
```

```
Name: target, dtype: int64
```

It's a multiclass classification problem!

[4]:
```
train_df, test_df = train_test_split(sample_df, test_size=0.2, random_state=123)
X_train_happy, y_train_happy = train_df["moment"], train_df["target"]
X_test_happy, y_test_happy = test_df["moment"], test_df["target"]
```

[5]:
```
pipe_lr = make_pipeline(
    CountVectorizer(stop_words="english"),
    LogisticRegression(max_iter=2000),
)
```

[6]:
```
pipe_lr.fit(X_train_happy, y_train_happy);
```

[7]:
```
pipe_lr.score(X_train_happy, y_train_happy)
```

[7]:
```
0.9528318584070796
```

[8]:
```
pipe_lr.score(X_test_happy, y_test_happy)
```

[8]:
```
0.8169911504424778
```

1.6.2 Training on the full corpus

- Ideally, you'll try all different models, fine tune the most promising models and deploy the best performing model.

- Sometimes before deploying a model people train it on the full dataset.
  - This is probably a good idea, because more data is better.
  - It's also a little scary, because we can't test this new model.
- Here I'm just deploying the model trained on the training set.

1.6.3 Saving the model

- If we want to deploy a model, we need to save it.
- we are using `joblib` for that.

[ ]:
```
with open("web_api/moment_predictor.joblib", "wb") as f:
    joblib.dump(pipe_lr, f)
with open("web_application/moment_predictor.joblib", "wb") as f:
    joblib.dump(pipe_lr, f)
```

We'll define a function that accepts input data as a dictionary and returns a prediction:

1.6.4 Loading our saved model

Let's write a function to get predictions.

[12]:
```
def return_prediction(model, text):
    prediction = model.predict([text])[0]
    return prediction
```

[13]:
```
model = joblib.load("web_api/moment_predictor.joblib")
text = "I love my students!"
return_prediction(model, text)
```

[13]:
```
'affection'
```

This function appears in the `app.py` that we'll be using shortly.

1.6.5 (Optional) Setting up a directory structure and environment

- We need a specific directory structure to help us easily deploy our machine learning model.
- This is already set up in this repo.

```
lectures
  web_api
      moment_predictor.joblib  # this is the machine learning model we have built locally
      app.py  # the file that defines our flask API
      Procfile  # required by Heroku to help start flask app
      requirements.txt  # file containing required packages

  web_application
      moment_predictor.joblib  # this is the machine learning model we have built locally
      app.py  # the file that defines our flask API
      Procfile  # required by Heroku to help start flask app
      requirements.txt  # file containing required packages
      templates  # this subdirectory contains HTML templates to help us build the web applicat
          style.css  # css template to be used in web application
      static  # this subdirectory contains CSS style sheets
          home.html  # html template to be used in web application
          prediction.html  # html template to be used in web application
```

1.6.6 Model deployment

We have two options for deploying our moment prediction model. We can:

1. Build a web application (app) with a HTML user-interface that interacts directly with our model.
2. Develop a RESTful (REST stands for REpresentational State Transfer) web API that accepts HTTP requests in the form of input data and returns a prediction.

We'll explore both options below.

1.6.7 Building and deploying a web app

	on localhost (my laptop)	on server (the interwebs)
app		
API		

- Flask can create entire web applications.
- We need to link our code to some html and css to create our web application.
- We will use Flask to create a html form, accept data submitted to the form, and return a prediction using the submitted data.
- Again, I won't go into too much detail here, but we can open up `web_application/` and take a quick look.

- We won't go into details here. If you want to learn more about Flask, see:
  - Flask tutorial video series by Corey Schafer
  - Flask docs
  - Flask tutorial by Miguel Grinberg

- Let's try `web_application/app.py` that handles this part.
- We can open it up here in Jupyter and take a look.
- If we run `python app.py` we'll bring it to life.

1.6.8 Web app on local server

1. Go to the terminal.
2. Navigate to the `web_application` directory.
3. Run the following to make the app alive: `python app.py`.
   - If you get an error, you may need to install those extra packages and make sure you have the environment loaded.
4. Now you should be able to access the app at: `http://127.0.0.1:5000/` or `http://localhost:5000/`.

1.6.9 Web app on a real server

- If you want people to use your app/model, you would probably want to put it on a real server and not your laptop so that it's live all the time.

	on localhost (my laptop)	on server (the interwebs)
app		
API		

- We'll use Heroku for this.

1.6.10 Heroku set-up (I already did these):

1. Go to Heroku, log-in, and click "Create new app".
2. Choose a unique name for your app. (I chose cpsc330-test-app.)
3. Create app.

- We will be using the Heroku CLI to deploy our model.
- We'll open up another terminal.

```
heroku login
cd web_application/
git init
heroku git:remote -a moment-type-predictor
git add .
git commit -am "Initial commit"
git push heroku master
```

```
I recommend copying web_application folder somewhere outside the CPSC330 repo and run the above
```

1.6.11 Testing the web application

- I already have done this and our app should be live at this link!

  https://moment-type-predictor.herokuapp.com/

- Try it out!
- This is nice! If you develop a model and you want your friends to try it out without installing anything on their local computers, you can do this.

1.6.12 API on the localhost

- Often you want other people to be able to use your models in their applications.
- We can do this by creating an **API**.
- If you don't know what an API is, that's OK.
  - For our purposes, it's something that exists at a particular address, that can accept information and return information.
  - Sort of like a function but not Python-specific and potentially accessible by anyone on the internet.

	on localhost (my laptop)	on server (the interwebs)
app		
API		

- Go to the terminal.
- Navigate to `web_api` folder in this repo.
- Run the following to make the api alive: `python app.py`

(Note that for more complex applications, you may choose to containerize everything in a Docker container to deploy to Heroku).

1.6.13 Sending a request to the API

- We have a RESTful (REST stands for REpresentational State Transfer) web API that accepts HTTP requests in the form of input data and returns a prediction.
- Now you can send requests to the API and get predictions.

[14]:
```
!curl -d '{"text":"I went for a run!"}' \
      -H "Content-Type: application/json" \
      -X POST http://localhost:5000/predict
```

"exercise"

- `curl` (stands for client URL) is a tool for transferring data using various network protocols.
- Okay, so we have a working API running on localhost, but we don't want to host this service on my laptop!

1.6.14 Deploying the API on a server

- We now want to deploy it on a "real" server so others can send it requests.
- We will use Heroku to deploy our app but you could also use other services such as AWS.

	on localhost (my laptop)	on server (the interwebs)
app		
API		

- Now the same commands:

```
heroku login
cd web_api/
git init
heroku git:remote -a moment-type-predictor-api
git add .
git commit -am "Initial commit"
git push heroku master
```

1.6.15 Using the API on Heroku

This is the process but I had trouble getting this working. I'll try to fix this and post updated notes soon.

[ ]:
```
# !curl -d '{"text":"I love my students!"}' \
#       -H "Content-Type: application/json" \
#       -X POST https://moment-type-predictor-api.herokuapp.com/predict
```

- OK so what this means is that anyone can do this.
- In fact, you all have your laptops - give it a try!
- You can also do the `curl` from a terminal:

```
!curl -d '{"text":"I love my students!"}' \
      -H "Content-Type: application/json" \
      -X POST https://cpsc330-test-app.herokuapp.com/predict
```

That's it for the API approach.

1.6.16 Discussion

- There are many ways to deploy a model; a RESTful API is very common and convenient.
- As you can see, a simple deployment is fairly straightward.
- However, there may be other considerations such as:
  - Privacy/security

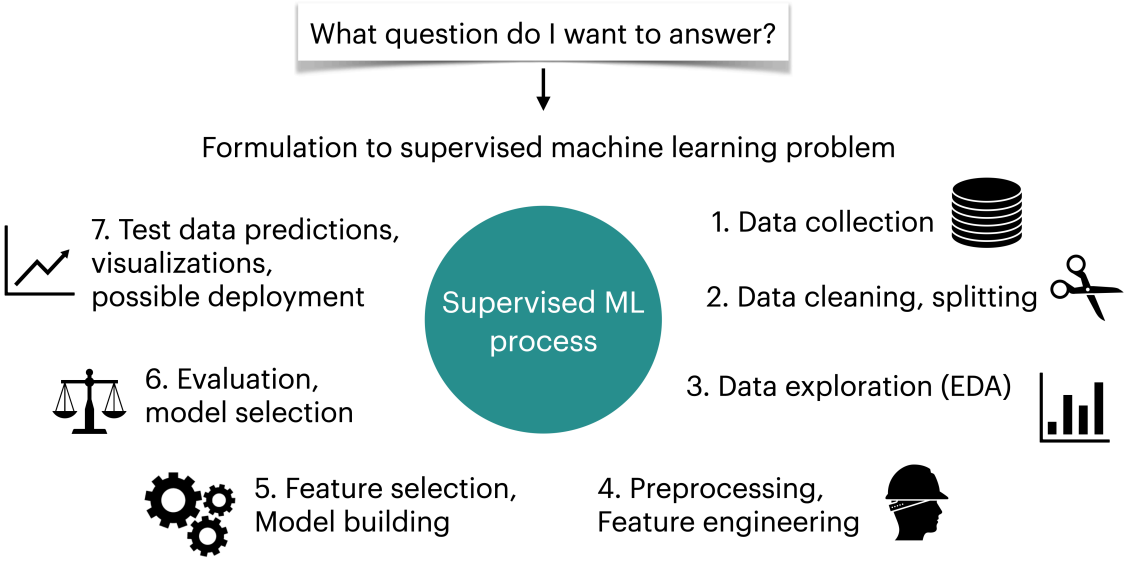
  - Scaling
  - Error handling
  - Real-time / speed
  - Low-resource environments (e.g. edge computing)
  - etc.

1.7 Break (10 min)

- We'll take a longer break today.
- Consider taking this time to fill out the instructor/TA evaluations if you haven't already.
  - Evaluation link: https://canvas.ubc.ca/courses/78046/external_tools/4732

- Here is Mike's post on these evaluations.

1.8 Course review / conclusion (20 min)



1.8.1 Learning objectives

Here are the course learning outcomes:

1. Identify problems that may be addressed with machine learning.
2. Select the appropriate machine learning tool for a problem.
3. Transform data of various types into usable features.
4. Apply standard tools implementing supervised and unsupervised learning techniques.
5. Describe core differences between training, validation, and testing regimes.
6. Effectively communicate the results of a machine learning pipeline.
7. Be realistic about the limitations of individual approaches and machine learning as a whole.
8. Identify and avoid scenarios in which training and testing data are accidentally mixed (the "Golden Rule").
9. Employ good habits for applying ML, such as starting an analysis with a baseline estimator.
10. Create reproducible workflows and pipelines.

- How did we do?
- Hopefully OK, except we skipped the last point (that will likely be its own new course).

1.8.2 What did we cover?

I see the course roughly like this (not in order):

Part 1: Supervised learning on tabular data

- Overfitting, train/validation/test/deployment, cross-validation
- Feature preprocessing, pipelines, imputation, OHE, etc
- The Golden Rule, various ways to accidentally violate it
- Classification metrics: confusion matrix, precision/recall, ROC, AUC
- Regression metrics: MSE, MAPE
- Feature importances, feature selection
- Hyperparameter optimization

- A bunch of models:
  - baselines
  - linear models (ridge, logistic regression)
  - KNNs and RBF SVMs
  - tree-based models (random forest, gradient boosted trees)
  - Ensembles

Part 2: Other data types (non-tabular)

- Clustering: K-Means, DBSCAN
- Recommender systems
- Computer vision with pre-trained deep learning models (high level)
- Language data, text preprocessing, embeddings, topic modeling
- Time series
- Right-censored data / survival analysis

Part 3: Communication and Ethics

- Ethics for ML
- Communicating your results
- ML skepticism

1.8.3 Some key takeaways

Some useful guidelines:

- Do train-test split right away and only once
- Don't look at the test set until the end
- Don't call `fit` on test/validation data
- Use pipelines
- Use baselines

1.8.4 Recipe to approach a supervised learning problem with tabular data

1. Have a long conversation with the stakeholder(s) who will be using your pipeline.
2. Have a long conversation with the person(s) who collected the data.
3. Think about the ethical implications - are you sure you want to do this project? If so, should ethics guide your approach?
4. Random train-test split with fixed random seed; do not touch the test data until Step 16.
5. Exploratory data analysis, outlier detection.
6. Choose a scoring metric –> higher values should make you and your stakeholder happier.
7. Fit a baseline model, e.g., `DummyClassifier` or `DummyRegressor`.
8. Create a preporcessing pipeline. This may involve feature engineering. (This is usually a time-consuming step.)
9. Try a linear model. For example, `LogisticRegression` or `Ridge`; tune hyperparameters with CV.
10. Try other sensible models(s), e.g., `LightGBM`; tune hyperparameters with CV.
11. For each model, look at sub-scores from the folds of cross-validation to get a sense of "error bars" on the scores.
12. Pick a promising model. Best CV score is a reasonable metric, though you may choose to favour simpler models.

13. Look at feature importances.
14. (optional) Perform some more diagnostics like confusion matrix for classification or "predicted vs. true" scatterplots for regression.
15. (optional) Try to calibrate the uncertainty or confidence outputted by your model.
16. Test set evaluation.
17. Question everything again: validity of results, bias/fairness of trained model, etc.
18. Discuss your results with stakeholders.
19. (optional) Retrain on all your data.
20. Deployment and integration.
21. Profit?

```
The order of steps is approximate, and some steps may need to be repeated during prototyping, e
```

1.8.5 What would I do differently?

- Lots of room for improvement.

- Add more interactive components in the lectures.
- Probably more pre-lecture videos.
- Cover outliers material.
- Some material on data collection.
- Allocate two lectures to time series data.

- Improve computer vision lecture material.
- Allocate one more lecture for ethics and communication.

I'm sure you have other suggestions - feel free to drop me an email or drop them in the course evaluations.

330 vs. 340

- I am hoping lots of people will take both courses.
- There is some overlap but not a crazy amount.
- If you want to learn how these methods work under the hood, CPSC 340 will give you a lot of that, such as:
  - Implementing `Ridge.fit()` from scratch
  - Mathematically speaking, what is `C` in `LogisticRegression`?
  - How fast do these algorithms run in terms of the number of rows and columns of your dataset?
  - Etc.
- There are also a bunch of other methods covered.

1.9 Conclusion & farewell

That's all, folks. We made it!


WELL
DONE!