ModelDeployment-checkpoint

January 1, 2025

1 Model Deployment

- ML techniques are not limited to offline application and analyses
- they have become predictive engine of various web services
  - spam detection, search engines, recommendation systems, etc.
  - online demo CNN for digit recognition: https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html
- the goal of this chapter is to learn how to deploy a trained model and use it to classify new samples and also continuously learn from data in real time

1.1 Working with bigger data

- it's normal to have hundreds of thousands of samples in dataset e.g. in text classification problems
- in the era of big data (terabyes and petabytes), it's not uncommon to have dataset that doesn't fit in the desktop computer memory
- either employ supercomputers or apply **out-of-core learning** with online algorithms
- see https://scikit-learn.org/0.15/modules/scaling_strategies.html

1.1.1 out-of-core learning

- allows us to work with large datasets by fitting the classifier incrementally on smaller batches of a dataset

1.1.2 online algorithms

- algorithms that don't need all the training samples at once but can be trained in batches over time
  - also called incremental algorithms
- these algorithms have `partial_fit` method in sci-kit learn framework
- use **stochastic gradient descent** optimization algorithm that updates the models's weights using one example at a time
- let's use `partial_fit` method of incremental SGDClassifier to train a logistric regression model using small mini-batches of documents

[2]:
```python
import os
import gzip

# check if file exists otherwise download and unzip the zipped imdb dataset
file = os.path.join('data', 'movie_data.csv')
```

```
if not os.path.isfile(file):
    if not os.path.isfile('movie_data.csv.gz'):
        print('Please place a copy of the movie_data.csv.gz'
              'in this directory. You can obtain it by'
              'a) executing the code in the beginning of this'
              'notebook or b) by downloading it from GitHub:'
              'https://github.com/rasbt/python-machine-learning-'
              'book-2nd-edition/blob/master/code/ch08/movie_data.csv.gz')
    else:
        with gzip.open('movie_data.csv.gz', 'rb') as in_f, \
                open(file, 'wb') as out_f:
            out_f.write(in_f.read())
else:
    print(f'File {file} exists!')
```

```
File data/movie_data.csv exists!
```

[3]:

```
import numpy as np
import re
from nltk.corpus import stopwords


# The `stop` is defined as earlier in this chapter
# Added it here for convenience, so that this section
# can be run as standalone without executing prior code
# in the directory
stop = stopwords.words('english')


def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) +\
        ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

# create an iterator function to yield text and label
def stream_docs(path):
    with open(path, 'r', encoding='utf-8') as csv:
        next(csv)  # skip header
        for line in csv:
            text, label = line[:-3], int(line[-2])
            yield text, label
```

[4]:

```
# use next function to get the next document from the iterator
next(stream_docs(path=file))
# should return a tuple of (text, label)
```

[4]:
```
('"In 1974, the teenager Martha Moxley (Maggie Grace) moves to the high-class
area of Belle Haven, Greenwich, Connecticut. On the Mischief Night, eve of
Halloween, she was murdered in the backyard of her house and her murder remained
unsolved. Twenty-two years later, the writer Mark Fuhrman (Christopher Meloni),
who is a former LA detective that has fallen in disgrace for perjury in O.J.
Simpson trial and moved to Idaho, decides to investigate the case with his
partner Stephen Weeks (Andrew Mitchell) with the purpose of writing a book. The
locals squirm and do not welcome them, but with the support of the retired
detective Steve Carroll (Robert Forster) that was in charge of the investigation
in the 70\'s, they discover the criminal and a net of power and money to cover
the murder.<br /><br />""Murder in Greenwich"" is a good TV movie, with the true
story of a murder of a fifteen years old girl that was committed by a wealthy
teenager whose mother was a Kennedy. The powerful and rich family used their
influence to cover the murder for more than twenty years. However, a snoopy
detective and convicted perjurer in disgrace was able to disclose how the
hideous crime was committed. The screenplay shows the investigation of Mark and
the last days of Martha in parallel, but there is a lack of the emotion in the
dramatization. My vote is seven.<br /><br />Title (Brazil): Not Available"',
 1)
```

[5]:
```python
# function takes stream_docs function and return a number of documents␣
 ↪specified by size
def get_minibatch(doc_stream, size):
    docs, y = [], []
    try:
        for _ in range(size):
            text, label = next(doc_stream)
            docs.append(text)
            y.append(label)
    except StopIteration:
        return None, None
    return docs, y
```

1.1.3 HashingVectorizer

- can't use `CountVectorizer` and `TfidfVectorizer` for out-of-core learning
  - they require holding the complete vocabulary and documents in memory
- `HashingVectorizer` is data-independent and makes use of the hashing trick via 32-bit MurmurShash3 algorithm
- difference between `CountVectorizer` and `HashingVectorizer`: https://kavita-ganesan.com/hashingvectorizer-vs-countvectorizer/#.YFF_lbRKhTY

[6]:
```python
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.linear_model import SGDClassifier

# create HashingVectorizer object with 2**21 max slots
vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)
```

[7]:
```
from distutils.version import LooseVersion as Version
from sklearn import __version__ as sklearn_version

clf = SGDClassifier(loss='log', random_state=1)

doc_stream = stream_docs(path=file)
```

[8]:
```
# let's trainithe model in batch; display the status with pyprind library
# takes about 20 seconds
import pyprind
pbar = pyprind.ProgBar(45)

classes = np.array([0, 1])
# use 45 mini batches each with 1000 documents
for _ in range(45):
    X_train, y_train = get_minibatch(doc_stream, size=1000)
    if not X_train:
        break
    X_train = vect.transform(X_train)
    clf.partial_fit(X_train, y_train, classes=classes)
    pbar.update()
```

```
0% [##############################] 100% | ETA: 00:00:00
Total time elapsed: 00:00:18
```

[9]:
```
# let's use the last 5000 samples to test our model
X_test, y_test = get_minibatch(doc_stream, size=5000)
X_test = vect.transform(X_test)
print('Accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Accuracy: 0.868
```

1.1.4 Note:

- eventhough the accuracy is slightly lower compared to offline learning with grid search technique, training time is much faster!
- we can incrementally train the model with more data
  - let's use the 5000 test samples we've not used to train the model yet

[10]:
```
clf = clf.partial_fit(X_test, y_test)
```

[11]:
```
# lets test the model again out of curiosity
print('Accuracy: %.3f' % clf.score(X_test, y_test))
# accuracy went up by about 2%
```

```
Accuracy: 0.884
```

1.2 Serializing fitted scikit-learn estimators

- training a machine learning algorithm can be computationally expensive
- don't want to retrain our model every time we close our Python interpreter and want to make a new prediction or reload our web application
- one option is to use Python's `pickle` module
  - `pickle` can serilaize and deserialize Python object structures to compact bytecode
  - save our classifier in its current state and reload it when we want to classify new, unlabeled examples

[12]:

```python
import pickle
import os
dest = './demos/movieclassifier/pkl_objects'
if not os.path.exists(dest):
    os.makedirs(dest)
# let's serialize the stop-word set
pickle.dump(stop, open(os.path.join(dest, 'stopwords.pkl'), 'wb'), protocol=4)
# let's serialize the trained classifier
pickle.dump(clf,
open(os.path.join(dest, 'classifier.pkl'), 'wb'), protocol=4)
```

[13]:

```python
%%writefile demos/movieclassifier/vectorizer.py
# the above Jupyter notebook magic writes the code in the cell to the provided␣
 ↪file; must be the first line!
from sklearn.feature_extraction.text import HashingVectorizer
import re
import os
import pickle

cur_dir = os.path.dirname(__file__)
stop = pickle.load(open(
                os.path.join(cur_dir,
                'pkl_objects',
                'stopwords.pkl'), 'rb'))

def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',
                           text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) \
                   + ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized

vect = HashingVectorizer(decode_error='ignore',
                         n_features=2**21,
                         preprocessor=None,
                         tokenizer=tokenizer)
```

```
Overwriting demos/movieclassifier/vectorizer.py
```

[14]:

```
# let's deserialize the pickle objects and test them
# change the current working directory to demos/movieclassifier
import os
os.chdir('demos/movieclassifier')
```

[15]:

```
# deserialize the classifer
import pickle
import re
import os
# this is our module generated in above cell
from vectorizer import vect
import numpy as np

clf = pickle.load(open(os.path.join('pkl_objects', 'classifier.pkl'), 'rb'))
```

[16]:

```
def result(label, prob):
    if label == 1:
        if prob >= 90:
            return ':D'
        elif prob >= 70:
            return ':)'
        else:
            return ':|'
    else:
        if prob >= 90:
            return ':`('
        elif prob >= 70:
            return ':('
        else:
            return ':|'
```

[17]:

```
# let's test the classifier with some reviews
label = {0:'negative', 1:'positive'}

example = ["I love this movie. It's amazing."]
X = vect.transform(example)
# predict returns the class label with the largest probability
lbl = clf.predict(X)
# predict_prob method returns the probability estimate for the sample
prob = np.max(clf.predict_proba(X))*100
print('Prediction: %s\nProbability: %.2f%%' %\
      (label[lbl[0]], prob))
print('Result: ', result(lbl, prob))
```

```
Prediction: positive
Probability: 95.55%
Result:  :D
```

[18]:

```python
# let's create a function so we can reuse the code easily...
def predict(review):
    label = {0:'negative', 1:'positive'}
    example = [review]
    X = vect.transform(example)
    # predict returns the class label with the largest probability
    lbl = clf.predict(X)
    # predict_prob method returns the probability estimate for the sample
    prob = np.max(clf.predict_proba(X))*100
    print('Prediction: %s\nProbability: %.2f%%' %\
          (label[lbl[0]], prob))
    print('Result: ', result(lbl, prob))
```

[19]:

```python
predict("The movie was so boring that I slept through it!")
```

```
Prediction: negative
Probability: 94.56%
Result:  :`(
```

[21]:

```python
predict("The movie was okay but I'd not watch it again!")
```

```
Prediction: negative
Probability: 64.22%
Result:  :|
```

[24]:

```python
review = input('Enter your review: ')
```

```
Enter your review: What a fun movie to enjoy with family!
```

[25]:

```python
predict(review)
```

```
Prediction: positive
Probability: 92.18%
Result:  :D
```

1.3 Web application with Flask

- install Flask framework and gunicorn web server
- gunicorn is used by Heroku

```
pip install flask gunicorn
```

- gunicron is recommended web server for deploy Flask app in Heroku
  - see: https://devcenter.heroku.com/articles/python-gunicorn

1.3.1 Hello World App

- follow direction here - https://flask.palletsprojects.com/en/2.1.x/quickstart/

- flask provides development server

```
cd <project folder>
export FLASK_APP=<flaskapp.py>
flask run
```

- don't need to export `FLASK_APP` env variable if the main module is named `app`
- run local web server with gunicorn
  - NOTE: do not use .py extension after

```
cd <project folder>
gunicorn <flaskapp>:app
```

- see complete hello world app here: https://github.com/rambasnet/flask-hello

1.3.2 Deploy Flask App

- see various options here - https://flask.palletsprojects.com/en/2.1.x/deploying/
- let's use Heroku platform to deploy our Flask app
- create Heroku account
- create an app on heroku or create it using Heroku CLI
- download and install Heroku CLI - https://devcenter.heroku.com/articles/heroku-cli
- add heroko to existing git repo or create a new one and add heroku
- follow the instructions found here: https://devcenter.heroku.com/articles/git
- create `requirements.txt` file with Python dependencies for you project

```
cd <projectRepo>
pip list --format=freeze > requirements.txt
```

- create runtime.txt file and add python version that's supported by Heroku (similar to local version)

```
python-3.9.4
```

- create `Procfile` and add the following contents:

```
web: gunicorn hello:app
```

- `IMPORTANT` - note the required space before `gunicorn`
  - app will not launch without it as of April 12, 2021
- tell Heroku to run web server with gunicorn hello.py as the main app file
- deploy the app using heroku CLI
- must add and commit to your repository first before pushing to heroku

```
git init
git branch -m master main # if necessary
git status
```

```
heroku login
heroku create -a "heroku-sub-dmian-app-name"
git add <file...>
git commit -m "..."
git push # push to github if necessary
git push heroku main # push the contents to heroku
heroku open # open the app on a browser
```

- if successfully deployed, visit `<app-name>.herokuapp.com` or run the app from your Heroku dashboard

1.3.3 Demo applications

- `demos/flask_app_1` - a simple app with template
  - install required dependencies using the provided requirement file

```
cd demos/flask_app_1
pip install -r requirements.txt
export FLASK_ENV=developement
flask run
```

- `demos/flask_app_2` - a Flask app with form
  - install required dependencies using the provided requirement file

```
cd demos/flask_app_2
pip install -r requirements.txt
export FLASK_ENV=developement
flask run
```

- `demos/movieclassifier` - ML deployed app
  - install required dependencies using the provided requirement file

```
cd demos/movieclassifier
pip install -r requirements.txt
export FLASK_ENV=developement
flask run
```

- `demos/movieclassifier_with_update` - ML deployed app with model update on the fly
  - install required dependencies using the provided requirement file

```
cd demos/movieclassifier_with_update
pip install -r requirements.txt
export FLASK_ENV=developement
flask run
```