Worflow description and considerations - ACAPS technical interview

Nicolas Taba

2022-12-19

Which programming language and platform do you choose for a prototype of the model? Why?

In order to share the result of a prototype or proof-of-concept (POC) machine learning (ML) model, I have chosen to use a Jupyter notebook and the programming language Python.

Python was chosen for its extensive use in ML applications and available documentation and resources. Python is also the programming language that I am the most familiar with.

I have then chosen to use a Jupyter notebook to share the code, model as this platform allows for rapid prototyping, plotting and reporting in a single file with an ease to share in different format to different collaborators to show early results. Jupyter notebooks also present the added advantage that when the prototype is approved, it should be relatively easy to port the code into proper development format by choosing the appropriate design paradigm, implementing routines as functions in .py files.

Jupyter notebooks are not the best way to share code in the case of continuous integration and continuous deployment development in practice as it can be difficult to track changes made to a Jupyter notebook through Git and track down the change that breaks a workflow.

How did you decide on the architecture of the model?

I first established baselines that have proven to be competitive in literature (logistic regression and random forest classifier). I eliminated support vector machines because they are usually inefficient from a computation time perspective. I then set out to perform cross-validation using countvectorizer and tfidf-vectorizer as word vectorization choices. Once a good baseline was found, I tried 2 other models using embeddings. I implemented embeddings with logistic regression and embeddings in a network using Bidirectional-LSTMs (using an architecture that is used for text classification according to the keras documentation). In all models, care was taken to include regularization parameters in order to improve generalization and reduce bias.

I then chose the model that performed best along all classes by evaluating the precision, recall and f1-score. The best model found was logisitc regression using tf-idf-vectorizer

Which metric, apart from the ROC curve, did you choose to evaluate performance, and why?

The precision, recall and f1 score were used to evaluate the performance of different models. Precision and recall give us an indication of how well the model is able to correctly identify a document to be in a given class. The higher the values, the better performing (the less assignment errors) the model is doing. The f1-score is then an aggregate (weighted average) of this two values and is often used in NLP for model evaluation. It is a measure that is more useful because the f1-score measures better the “assignment error-rate” than accuracy does, especially with uneven class distribution.

What is the performance of your model? Please provide the value of ROC and your selected metric here, in aprticular for the performance when tagging the topic "Statistics"?

The performance metrics, ROC and AUC for the "Statistics" can be found on the Readme page of the repository

What are the main limitations that you have encountered in Part 1?

The uneven distribution of the classes leads us to have to perform undersampling to avoid class distribution bias when fitting our model. This leads us to reduce the dataset that is used for training and is detrimental to training in general. We find that this affected the f1-score by 1 percentage point so it did not become a large issue in our case. There is also a lot of overlap between the classes statistics and computer science classes that causes the model to fail (from the confusion matrix). This means that the model is limited if there are many words that appear in documents of different classes. We will then be forced to use a different model that relies on a different premise than bag-of-words.

Would the workflow you have designed be reliable also as the number of users will grow? Please motivate your answer

As stated in the fist question, using a Jupyter notebook can work if we have a manageable amount of data that can be processed by a developer/data engineer in case some edge case is not handled properly. This will not work as the number of users increase.

The general workflow is sound: load the data, remove uncesssary information, normalize the data, train the model, evaluate performance, return results. The platform used is not designed to handle requests and to ingest increasing amounts of data. This should ideally be automated.

If this model was deployed as a web-app using a scalable API such as FastAPI that can handle several requests at the same time, the service should be able to respond to the user fast enough, especially if the application is deployed using a containerized environment like Docker. Scaling API to user request is the first hurdle.

When the users submit files in large quantities and size of file, it is necessary to use a service to avoid overhead costs for hosting the data and performing training or responding to API requests. For storage Amazon S3 can be used which offers easy access to cloud resources and can store for many use case such as data lakes (especially if the file types can be varied). Then for batch analytics work that are scheduled, we could use Amazon Batch to perform all batch jobs to then store data from the S3 storage to a DynamoDB database that can then be leveraged through Sagemaker to perform retraining and evaluation. The clear advantage of these services is that they scale natively.

Beyond this there is a need to use other tools or pipelines to streamline the data processing. As the number of user increases, there is an increase in the API calls to the database, but there is also an increase in the need to retrain the model on new data to avoid model drift. We also need to keep in mind that we need to change the window of data that is used for training to avoid drift (older data is forgotten) during retraining. We also have to keep in mind that the data that is submitted might not be properly annotated for the database for training. If there is an inability to get annotated data, we could use non-supervised clustering techniques to create new classes.

Given the scenario outlined, what deployment strategies do you suggest? Please detail examples of workflows/platforms/pipelines that you think could be relevant.

There are several aspects to take into account for this question: user requests for classification of document, handling of users uploading data and (re)training of the model.

- User request for classification: Through a containerized application (Docker) that sends requests through FastAPI (scalable) for evaluation of our model that is implemented in AWS SageMaker or computed on Amazon EC2. The use of Docker ensures easy redeployment of the service.
- Handling of user uploading data: Data is uploaded to a datalake on Amazon S3, a amazon Batch handles a scheduled batch job to process the data that can then be sent to DynamoDB for storage. A specialized script using third party tools could handle the different file types from the S3 datalake.
- (Re)training of the model: Using Batch and Sagemaker, we can leverage the data presented in DynamoDB to perform model training and evaluation. Compute resources can be leverage on Amazon EC2.Since the data is read from DynamoDB, some of the metadata can be leveraged such as manual annotations and can be triggered through scripts to run weekly for example.

How do you expect that the increase in data will impact the performance of your model? How could you verify that and what re-training strategies could you develop?

The increase in data is expected to cause a drift in the classifier model and reduce its performance over time. New words are used in different topics, new classes become relevant or new file types include more or different data that is not usually handled by the implemented scripts. The impact of increasing data can be evaluated at training time using the evaluation metrics presented before when the model performance falls below a threshold. The drift in data should show a reduced performance of our model on new data as time goes on. In order to retrain, we could re-train our model only on new data using a sliding window and test on previously annotated data. In this retraining scheme, we could also try other model candidates on the same data in parallel depending on resources available. Through A/B testing we can also decide if a new model is to be deployed in the place of the previous one.

Finally for hyperparameter optimization, bayesian optimization can be used where previous hyperparameters from previous models are used as "first guess" or prior for the next iteration of hyperparameter optimization to reduce computation time (as long as the assumption of the drift being continuous is true).

What will be your strategy to handle different file types? Do you anticipate developing a standalone solution to the problem, or use available services? Argument your point

The handling of different file types is notoriously complicated. Third party libraries exist to handle most file formats except for pdfs (from my personal experience) that can sometimes be difficult to extract data from. I would implement different scripts to handle the different file types based extracted from the archive submitted. The best solution would be to use ready resources that are available such as amazon S3 with its data lake implementation. The user uploading the archive would have to fill in information about the archive uploaded and perhaps the files themselves. An ETL pipeline could then be implemented via AWS Batch to finally store the cleaned data into DynamoDB.