Machine Learning Engineer

Take-home challenge

Thanks a lot for your interest in the Machine Learning Engineer position in our team! We are very excited to get to know you better and introduce you to our work.

About our team

Our role within the company is to ship products that incorporate Machine Learning so as to automate, streamline and sometimes even outperform manual operations needed throughout the company, from toxicity screening, to gamer support or transactional fraud detection.

We are a team of builders at heart: our motto is to be pragmatic and always go all the way to production. We strive to read the latest research papers, adapt them to our specific problems, build prototypes, and finally craft fully packaged products off of them.

Context of the take-home challenge

Our team is currently responsible for the development of a custom search engine. Aimed at understanding natural language from all gamers, this product will bring automated support to Ubisoft players whenever they face an issue with their game, may they have a technical problem or simply want to know how to beat one tough of a boss. Our end goal is to embed this product within our global game launcher as well as other player-facing apps.

The current prediction pipeline used by the team is partly based on a recent research paper in the field of NLP: sentenceBERT https://arxiv.org/pdf/1908.10084.pdf.

We have built this challenge to give you a taste of the kind of work we do in our team. Actually, it is a simplified, stripped-down version of our product. Don't be worried if your prediction performance is not great, it is expected in such a limited amount of time.

If you feel like there is something unclear, please reach out to us with your questions. There are no bad points for doing so, quite the contrary. We hope you will have fun working on this task, just like we do every day!

The challenge

NB: you do not need extensive knowledge in NLP for this test. No one can be an expert in all the subfields of Machine Learning. However, we expect you to cope on your own by reading the documentation found in the researcher's repository on Github. We strongly advise you to use Google Collab to get access to free GPUs, unless you have one at your disposal.

Based on this repository https://github.com/UKPLab/sentence-transformers and the 20 Newsgroups dataset from sklearn https://scikit-learn.org/stable/datasets/index.html#the-20-newsgroups-text-dataset, you will build a sentence classifier in Python:

1) In a Notebook, create a custom loading class, compatible with Pytorch *DataLoader*, that generates training triplets (anchor, positive example, negative example) from 20 Newsgroups. You might want to take a look at the *SentenceLabelDataset* class https://github.com/UKPLab/sentence-transformers/blob/6fcfdfb30f9dfcc5fb978c97ce02941a7aa6ba63/sentence_transformers/datasets/SentenceLabelDataset.py.
   You should come up with a strategy to generate triplets that will be the most helpful / insightful for the model to train with.

2) Build a training pipeline and fine-tune a **distilbert-base-nli-mean-tokens** model with your custom loading class, using the *TripletLoss* loss function. Since fine-tuning is quite time-consuming, even on a GPU, you can go for a single epoch. Your triplet generation strategy is what matters to us.

3) Do some research online to find an Approximate Nearest Neighbor library. There are many of them and there really is no wrong choice. But we expect you to explain your pick in a few words.

4) Build a basic prediction pipeline:
   a. Vectorize the training set with your fine-tuned sBERT model
   b. Index all these vectors with your ANN library
   c. Build a barebone kNN classifier where new text input gets predicted the same label as that of the closest neighbor from the index
   d. Benchmark this pipeline with the test set
   e. Compare the results with the pretrained sBERT model

5) Create a simple Python REST API that serves this prediction via a "/predict" route. Given some input text, it should return one of the 20 Newsgroups labels as a prediction. We expect your code to be wrapped in a docker container via a Dockerfile.

Please send us back:

- your code for the API in a private Github/Gitlab repo
- a link to your Colab Notebook (or add your jupyter notebook to the repo)
- a pdf version of your notebook (just in case)