Contents

1 Searching Python projects with neural embeddings

1.1 Introduction

As of 2021, we observe several years of increased adoption of neural network methods in NLP. There is evidence that language models trained on massive datasets capture various aspects of these datasets. As an evidence for that task one can point to transformer-based neural network models which are able to generate Python code. Even though the models aren't explicitly trained to generate code, the model is able to generate programs, because massive datasets used for training language models contain code.

In the following we propose a dataset and methods for searching Python machine learning repositories using natural language descriptions.

1. Use case

   Github repositories are a standard way to share projects for software engineering and data science projects. Software engineers, and to a greater extent data scientists often encounter a situation where for a

given novel problem they need to find a solution, possibly for a slightly different problem. This is especially important for a problem that might require machine learning, because this domain is far less standardized than traditional software engineering, and some decompositions might lead to suproblems which might be solved by methods that are less explored than others.

For an example take the task of developing system for reading information from billboards. Whereas some Optical Character Recognition methods are well explored, they are heavily biased towards regular text. If our solution has to read billboards that contain curved text, then we will have to use methods that are far less standardized than reading documents. In this problem, we know that these methods might underperform on curved text. Because of this we might try to search for curved text detection.

"Curved text detection" which the developer might guess as a search query is a phrase that actually occurs in some research papers. In many real world problems we will not have this luxury.

2. Searching on github

   Github (as of 2021) provides a search interface that can be used to search repository names, its README files, and topics . This functionality is good when we have a good idea for a query, but it suffers from usual problems of bag-of-words model of information retrieval.

   Our work can be described as improving Github's topic functionality. These labels (which are optionally specified by repository contributors) are very useful for information retrieval. Nonetheless, searching using topics has two problems:

   - because they are optional, many projects have few or none topics specified
   - topic names are not standardized, so synonymous topics are hard to find

1.1.1 Related work

Using NLP and more generally machine learning for analyzing code has long tradition. Mining Software Repositories (MSR) is an example conference on this topic, existing from 2004.

Papers submitted to MSR typically focus on several tasks:

- software error detection and correction
- mining software QA sites
- mining metadata (commit history et c) of sotware projects
- NLP in software engineering

Another research track relevant to our work is using neural networks for feature extraction from code. These features are then used for code classification, generation or search.

In the following we list topics related to our problem and summarize important papers. The two approaches most relevant for our work are Language Models for code search and Import2Vec.

1. Programming language-specific feature extraction
   (a) Code2Vec
   (b) CodeSearchNet Husain et al 2019 propose a dataset for semantic search of code snippets. They use pretrained CodeBert model (Feng et al, 2020) which is then trained for matching of function comments with code. This can be considered as microscopic version of our problem, as we propose to match **repositories** with descriptions.
   (c) Import2Vec Theeten et al, 2019 explore using Word2Vec model for extracting features of modules. Import2Vec is an unsupervised model which tries to classify whether two modules cooccur in some context. The approach proposed in this paper is most relevant to our work, as it is the only approach that provides features for higher-level software objects. Whereas most work focuses on extracting features for code snippets or functions, Import2Vec provides features that can be directly used to featurize repositories.
   (d) Crossminer
2. Zero-shot Learning Because matching repositories with tasks requires comparing data from possibly different modalities (repository features vs features extracted from text), we use zero-shot learning models for matching.

   Zero-Shot Learning – A Comprehensive Evaluation of the Good, the Bad and the Ugly

Zero-shot learning is an alternative to standard supervised learning where we assume that classes at test time might be unknown. This is possible because in ZSL classes are assumed to be structured. In standard supervised learning classes are independent, whereas ZSL classes are given by their attributes, for example for image classification one might use word embeddings of class names.

This approach enables a type of nearest neighbor search across two different types embeddings.

1.2 Contributions

We propose a dataset for information retrieval on Github repositories using their natural language descriptions given by PapersWithCode tasks.

Several methods are proposed for feature extraction from code are proposed and evaluated.

These approaches roughly fall into two categories:

- use structural properties of repositories (leveraging graph node embeddings)
- aggregate lower-level features extracted from code snippets

1.3 Dataset and problem description

In the following we will work on addressing the drawbacks of standard bag-of-words Github search by leveraging structure of code repositories and NLP techniques for their natural language descriptions.

To this end we propose semantic search system on top of features extracted from repository code.

1.3.1 General ideas

1. Repository feature aggregation

   Due to explosion of interest in transformer-based models in NLP there exist several methods enabling search beyond bag-of-words model. This kind of functionality is very useful for cases where given specific query we want to find text fragment related to query, like in question answering models. Unfortunately this cannot be used as-is for repository search, because repositories contain multitude of such fragments. This means that for repository search we would need to find method that aggregates features from parts into more global features.

(a) Using hierarchical structure

Because repository code consists of files, which in turn consist of classes and functions defined in them, a natural question is:

RQ1: can we leverage this hierarchy to obtain features?

The following work evaluates following approaches to use hierarchical structure of repositories:

- use definition dependencies (file contains functions, functions contain other subfunctions called in them etc) to create a graph that can be used to extract features
- summarize repositories using clustering methods

For the first approach we propose using embeddings into hyperbolic space, which is a continuous analog of trees.

(b) Hyperbolic geometry for exploiting hierarchical graph structure

Embeddings of graphs, and more general, finite metric spaces into Euclidean spaces is a well-studied topic. An example fact about such embeddings is Johnson-Lindenstrauss Lemma.

A known fact about such embeddings is that they have large distortion for some classes of graphs. For example, there doesn't exist an isometric embedding of ternary tree into Euclidean space.

In contrast to this, for each tree there exists an embedding into 2-dimensional hyperbolic space with low distortion, as proved in Sarkar 2011.

This fact justifies calling hyperbolic spaces continuous analogs of trees.

i. Hyperbolic graph embeddings Embeddings into hyperbolic spaces were first proposed in Poincaré Embeddings for Learning Hierarchical Representations.

Because of numerical issues with Poincare model (in this model hyperbolic space is represented by unit disk, where distances between points near the unit circle are unbounded) there have been proposals for improving this model using hyperboloid (Lorentz) model.

One of these is Hyperbolic Multidimensional Scaling proposed in Representation Tradeoffs for Hyperbolic Embeddings.

Comparing these methods is challenging, especially for larger networks, as some of them use distance matrices.

RQ2: are embeddings into hyperbolic spaces feasible for big graphs?

Analyzing code requires to either use model that can handle both code and natural language, or matching embeddings across two different domains.

2. Matching repository and query features

   We evaluate retrieval of arXiv paper's Python repository given its PapersWithCode **task**.

   An example of gold standard query-results:

   query: 'Anomaly Detection'

   titles of 10 matched papers:

```
Learning Representations of Ultrahigh-dimensional Data for Random Distance-based O
The Semantic Knowledge Graph: A compact, auto-generated model for real-time traver
How to find a unicorn: a novel model-free, unsupervised anomaly detection method f
A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivar
ToyADMOS: A Dataset of Miniature-Machine Operating Sounds for Anomalous Sound Dete
High-Dimensional Multivariate Forecasting with Low-Rank Gaussian Copula Processes
Extended Isolation Forest
Learning Deep Features for One-Class Classification
Constrained Concealment Attacks against Reconstruction-based Anomaly Detectors in
Deep Semi-Supervised Anomaly Detection
```

   There are 1625 **tasks** that are used for queries. PapersWithCode groups them into areas:

area	number of tasks
adversarial	9
audio	28
computer-code	37
computer-vision	500
graphs	49
knowledge-base	22
medical	181
methodology	138
miscellaneous	125
music	16
natural-language-processing	341
playing-games	38
reasoning	15
robots	26
speech	51
time-series	49
total tasks	1625

(a) Train-test split

Test queries were selected as 20% from each area (PapersWithCode tasks are grouped into areas like computer-vision, natural-language etc).

1.4 Models

1.5 Results

1.5.1 Topline

Topline was established by matching queries (tasks) with paper's name.

The matching was scored by using cosine similarity embeddings of queries and paper titles obtained using three models:

- GLoVe
- sentence-bert
- SPECTER: Document-level Representation Learning using Citation-informed Transformers

Results are calculated using standard information retrieval metrics.

Accuracy here means that in k retrieved results at least one is a correct result.

	Accuracy@1	Accuracy@3	Accuracy@5	Accuracy@10
sentence-bert	0.561	0.707	0.768	0.829
glove	0.659	0.768	0.829	0.89
specter	0.683	0.768	0.854	0.902

	Precision@1	Precision@3	Precision@5	Precision@10
sentence-bert	0.561	0.451	0.371	0.257
glove	0.659	0.545	0.471	0.354
specter	0.683	0.512	0.432	0.317

	MRR@10
sentence-bert	0.65
glove	0.732
specter	0.751

	NDCG@10
sentence-bert	0.365
glove	0.473
specter	0.445

	MAP@10
sentence-bert	0.258
glove	0.359
specter	0.319

1. Search system description Our system can be decomposed into two separate parts:

   - feature extraction from repositories
   - matching repository features with query features

1.5.2 Dataset

We use source files obtained from Github BigQuery dataset.

1. Python Dependency graph

   We define a directed graph (V, E) where

   V_{repo} - repository edges

   V_{file} - files from repositorires

   V_{df} - functions defined in repositories

   V_f - functions called in functions defined in repositories

   $V = \{ROOT\} \cup V_{repo} \cup V_{file} \cup V_{df} \cup V_f$

   Vertices from V form dependency hierarchy, where next level contains elements dependent on previous level elements. This gives a natural definition of (directed) edges: $(v, w) \in E$ if w is defined/contained in v, where v is from one level higher (root-repo, repo-file, file-function, function-called function)

1.5.3 Selected queries

1. Results

   10 results were retrieved for queries in two groups:

   - metric learning/similarity learning/distance learning
   - word embeddings/semantic similarity/paraphrase detection

Query	0
metric learning	metric-learn: Metric Learning Algorithms in Python
similarity learning	Semi-supervised Sequence Learning
distance learning	DeepSDF: Learning Continuous Signed Distance Functions for Shape Re
word embeddings	Dict2vec : Learning Word Embeddings using Lexical Dictionaries
semantic similarity	Learning Semantic Textual Similarity from Conversations
paraphrase detection	Encoding Structure-Texture Relation with P-Net for Anomaly Detection

   (a) Evaluation

   We evaluate results for three queries:

   - metric learning

- word embeddings
- scene text

"accuracy" and MRR was dropped because for all the queries first retrieved result is correct, so $Accuracy@k = 1$

(b) Query: "metric learning"

	Precision@1	Precision@3	Precision@5	Precision@10
sentence-bert	1	0.333	0.6	0.7
glove	1	1	1	0.8
specter	1	0.667	0.8	0.6

	NDCG@10
sentence-bert	0.678
glove	0.87
specter	0.676

	MAP@10
sentence-bert	0.476
glove	0.8
specter	0.486

(c) Query: "word embeddings"

	Precision@1	Precision@3	Precision@5	Precision@10
sentence-bert	1	1	0.6	0.5
glove	1	1	1	1
specter	1	1	0.8	0.8

	NDCG@10
sentence-bert	0.611
glove	1
specter	0.836

	MAP@10
sentence-bert	0.417
glove	1
specter	0.704

(d) Query: "scene text"

	Precision@1	Precision@3	Precision@5	Precision@10
sentence-bert	1	0.333	0.2	0.2
glove	1	1	1	0.8
specter	1	0.667	0.6	0.4

	NDCG@10
sentence-bert	0.29
glove	0.857
specter	0.494

	MAP@10
sentence-bert	0.125
glove	0.758
specter	0.293

1.6 Code analysis

1.6.1 Code prototypes

Because each repository contains multiple Python files, we first try to extract typical code fragments using process similar to text summarization.

Descriptive lines were selected as class/function declarations from Python files.

From these lines we select prototypical lines by clustering their CodeBERT/FastText representations, and then choosing the lines that are closest to centroids.

1. huggingface transformers

	codebert	fasttext
0	class BaseModel(object):	class LayerNormalization(nn.Module):
1	class MultiHeadAttention(nn.Module):	class Translator(object):
2	class Transformer:	def __init__(self, d_{model}, d_{ff}, dropout=0.1):
3	def __init__(self, opt, use_{cuda}):	def __init__(self, hp):
4	def eval(self, xs, ys):	def $build_{vocab}$(examples, max_{size}, min_{freq},
5	def forward(self, q, k, v, $attn_{mask}$=None):	def $calc_{numbatches}$($total_{num}$, $batch_{size}$):
6	def $get_{tokenembeddings}$($vocab_{size}$, num_{units}...	def forward(self, q, k, v, $attn_{mask}$):
7	def $layer_{norm}$(inputs, epsilon=1e-8, scope='la...	def inference(self, dec_{output}):
8	def $load_{hparams}$(parser, path):	def $sort_{key}$(ex):
9	def $update_{lr}$(self):	def train(self, xs, ys):

2. mmdetection

	codebert	fasttext
0	class FCOSHead(nn.Module):	class ATSS(SingleStageDetector):
1	class MaskRCNN(TwoStageDetector):	class AnchorHead(nn.Module):
2	def __init__(self, dataset, times):	def __init__(
3	def $\_pad_{img}$(self, results):	def __init__(self):
4	def $check_{normstate}$(modules, $train_{state}$):	def __init__(self, normalizer=4.0):
5	def $collect_{resultscpu}$($result_{part}$, size, tmp...	def $add_{extrarepr}$(m):
6	def $ga_{shapetarget}$($approx_{list}$,	def $build_{assigner}$(cfg, **kwargs):
7	def get_{bboxes}(self, det_{bboxes}, $grid_{pred}$, im...	def forward(self, x):
8	def $show_{result}$(self,	def $get_{clsresults}$($det_{results}$, annotations, ...
9	def $test_{bitmapmaskinit}$():	def $init_{weights}$(self, pretrained=None):

3. Recommenders-movielens

	codebert	fasttext
0	class LibffmConverter(object):	class MetricsLogger:
1	def $\_\text{init}\_\text{(self)}$:	class _TrainLogHook(tf.train.SessionRunH
2	def _convert(field, feature, $\text{field}_{\text{index}}$, fiel...	def $\_\text{hash}\_\text{(self)}$:
3	def $\text{cached}_{\text{wrapper}}$(*args, **kwargs):	def $\_\text{init}\_\_\text{(}$
4	def end(self, session):	def $\_\text{init}\_\text{(self, filepath=None)}$:
5	def mae(	def auc(
6	def $\text{merge}_{\text{rankingtruepred}}$(	def $\text{cached}_{\text{wrapper}}$(*args, **kwargs):
7	def $\text{precision}_{\text{atk}}$(	def $\text{fit}_{\text{transform}}$(self, df, $\text{col}_{\text{rating}}$=DEFAULT.
8	def $\text{python}_{\text{randomsplit}}$(data, ratio=0.75, seed...	def $\text{from}_{\text{pandashash}}$(val):
9	def $\text{to}_{\text{pandashash}}$(val):	def $\text{python}_{\text{randomsplit}}$(data, ratio=0.75, seed

1.6.2 Baseline results

Code prototypes were queried by task name

1.6.3 Embedding dependency graph

Node embedding methods:

- ProNE on this graph.
- Poincare Embeddings

NEEDS REVISION

Retrieving most similar nodes using this method gets nodes with similar level of abstraction - repositories are similar to repositories et c.

	apache/incubator-mxnet	google-research/recsim
0	apache/incubator-mxnet	google-research/recsim
1	uber/pyro	google-research/task_adaptation
2	sony/nnabla	google-research/understanding-transfer-learni
3	tensorflow/models	google-research/tiny-differentiable-simulator
4	huggingface/pytorch-transformers	google-research/tensorflow-coder
5	tensorflow/tensorflow	google-research/tapas
6	huggingface/transformers	google-research/seed_rl
7	tensorflow/probability	google-research/torchsde
8	natanielruiz/android-yolo	google-research/valan
9	google-research/google-research	google/TensorNetwork

```
,DistilBertForMaskedLM,__init__,fit,tensorflow,test,train
0,image1,devlpl.f:file,fit,image1,test,train
1,hand_guide,__init__,estimate,hand_guide,predict,evaluate
2,hand_dapg_demos:file,ast_quality_perf_test,objective,hand_dapg_demos:file,infer,run_t
3,hamp,factored:file,test_prediction,hamp,eval,train_model
4,hamming_window,time_rhoxy:file,test,hamming_window,process,eval
5,hamming_distance,test_get_charge,run,hamming_distance,get_data,run_experiment
6,hamming_dist,demogame_renderer:file,process,hamming_dist,plot,train_epoch
7,hamiltonian_sparse,pyunit_constant_response_rf:file,learn,hamiltonian_sparse,go,learn
8,hamiltonian_graph_network:file,test_utils_tf:file,_evaluate,hamiltonian_graph_network
9,hamiltonian:file,02_op_attributes:file,test_model,hamiltonian:file,validate,train_net
```

1.6.4 Zero shot learning approach results

1. Data preparation We retrieve three types of data for repositories:

   - repository names
   - Python modules imported in repository
   - Python modules and functions defined or called repository (graph descendants)

   Embeddings of repositories are retrieved by averaging embeddings of above data.

   Used embeddings:

   - import2vec
   - ProNe
   - Poincare embeddings

   The following table shows results for top-k classification of repository task classification on test set. Note that test set uses different tasks that training set.

	Accuracy@1	Accuracy@3	Accuracy@5
import2vec	0.183	0.22	0.265
poincare	0.131	0.175	0.205
poincare (both repository and task)	0.104	0.142	0.168
prone	0.104	0.147	0.179
prone (both repository and task)	0.075	0.116	0.136