Semantic Parsing of TAU-Related Questions

Peretz Yafin, Alon Kafri, Inbal Joffe

September 15, 2017

Abstract

In this paper, we implemented a question answering engine based on Tel-Aviv University (TAU) related questions. We followed the implementation suggested by [Berant et al.(2013)Berant, Chou, Frostig, and Liang]. Instead of using the Freebase knowledge base, that contains many relations and topics, we created our own database using data we extracted from the TAU website. The goal of this project is to yield an accurate and user-friendly question answering engine, for that specific kind of questions. The tool should be easily used by students and lecturers to seek for relevant data, by phrasing their own questions about the courses or lecturers.

1 Introduction

The paper by Berant presents an implementation of a semantic parser that parses natural language utterances into logical forms. Those expressions can be executed to produce denotations. There are two major differences between our implementation to Berant's

- Berant gives a solution for an unlimited number of domains, that is, the questions can be related to any topic; whereas in our paper the topic is very specific and the relations are predefined by us.
- The number of entities and relations in our database is considerably smaller than in Berant's database, therefore we chose to work with an in-memory cache rather then λ-dcs queries that are executed on the database.

We will describe the database building procedure, the composition step, the learning mechanism, and finally the results we achieved.

2 Database

2.1 Extracting the data

We extracted information from the Tel-Aviv University website. The website contains two kinds of relevant data

- **Alphon** - a .csv file that contains the relevant data about all the lecturers in the university
- **Courses** - a .csv file that contains the relevant data about all the courses of the upcoming year

The data in both structures is in Hebrew. we translated the data using Google Translate in order for the engine to work in English. The translation of names can be misleading since Google Translate tries to fix names into words. Furthermore, some of the contains spelling mistakes, incorrect or empty fields that we could not overcome.

2.2 Database Schema

We used sqlite in order to create our database. The database has multiple tables. The schema of the tables is shown in 1.

The idea behind this schema is that there can be multiple "courses" for each multi-course. That is, each multi-course represents a course number. For example the course *Advanced Methods in Natural Language Processing* has a number *0368-4130-01*. This can be split into the multi-course id which is *0368-4130*. Each of the course groups (if this course has recitations or several groups) is represented by the entity course. Therefore, each multi-course can have many course entities. Each course can have several lecturers and several course occurrences (building, day, time). The *Course Table* has a foreign key to the lecturer table with a *many-to-many* relation. That is, each course can have multiple lecturers. Whereas the other tables has a a foreign key that represents a *one-to-many* relation.

We defined four kinds of predicates:

- **attr** - stand for 'attribute', one to one relation between SQL entity to another entity, for example email.'lecturer' is email of lecturer.

- **rev** - stand for reversed attribute, one to many relation between attribute to SQL entity. for example revBuilding.'building' is all of the occurrences of which their building attribute is 'building'.
- **union** - one to many or many relation between SQL entities. for example courses.'lecturer' is all of the courses which taught by 'lecturer'.
- **func** - predicates which are not simple relation in the database but implemented as a different function for each case, for example <.hour:1800 is all of the hour before 18:00.

2.3 Database Cache

As will described in the learning and composition sections, the training procedure consists of taking question-answer pairs, and for each question, derive all of its $\lambda - dcs$ expressions and execute them on the database to retrieve the answers. Each question may yield a large number of derivations. Combined with the large number of question-answers pairs in the training step, executing all of them on the database can decrease the performance of the training step drastically.

To overcome this issue, we decided to use a database cache which basically represents the database entities and relations in memory. Each $\lambda - dcs$ query is translated to a simple recursive execution on the database cache. Since the number of lecturers is approximately $13,000$ and the number of courses is around $8,000$ the memory consumption of such a cache is redundant. Assuming each entity is 1KB (in reality its much less than that), holding such a cache in memory can take up to 20MB which is not an issue for any computer. Furthermore, we can take the cache a level up by mapping the entities by the expected question; We created the entity types which have few words in its names (courses, lectures, departments...) and mapped them to a tree of words such that every node contains the entities which their name contains some words or its successors contains those words. For example we created a mapping between the departments its multi-courses, a mapping between the office buildings and its lecturers.

This will reduce the complexity of searching the required entities but increase the size of the in memory cache.

3 Approach

We followed the approach suggested in [Berant et al.(2013)Berant, Chou, Frostig, and Liang] with few changes.

3.1 Lexicon Construction

The lexicon represents a mapping between words to predicates, relations and entities. For example we need to transform the words *Introduction To Computer Science* to the actual course entity in our database. Another example of the lexicon usage would be question words; for example we can map the word *Who* to the predicate *Lecturer*. In the actual paper, the number of entities and predicates were huge, so in order to build the lexicon an alignment phase must find the actual mapping between the words and the database entities.

In our implementation, an alignment phase is not necessary since the number of entities and predicates is very small and with few simple rules we can build a proper lexicon that would fit into our goals.

The first part of the lexicon is the **Static Lexicon** which is a set of predefined mapping rules that can be hard-coded. A good example for that is the question words such as:

$$teach \Rightarrow Course, Lecturer$$

$$Who \Rightarrow Lecturer$$

$$Where \Rightarrow Course.Building$$

.

The second part of the lexicon is the **Dynamic Lexicon** which is built in run time to match words to actual entities from the database. Upon the program start, the lexicon is updated with the database entities; For each course a mapping between the course name to the course id is saved and for each lecturer a mapping between each lecturer name to its lecturer id is saved. When parsing a utterance into a query, the words from the utterance will be replaced by the lexicon values if matched.

3.2 Composition Rules

We used a set of composition rules that are used in the derivation process. These composition rules translates the question into a well formed λ-dcs query. As described in Berant's papers, we first define a knowledge base $\mathcal{K}$. The knowledge base is constructed from a set entities $\mathcal{E}$ and a set of properties $\mathcal{P}$. The knowledge base is a set of assertions $(e_1, p, e_2) \in \mathcal{K}$. For example *(JonathanBerant, Teaches, Advanced Methods in Natural Language Processing)*.

The composition rules are built on the knowledge base as follows

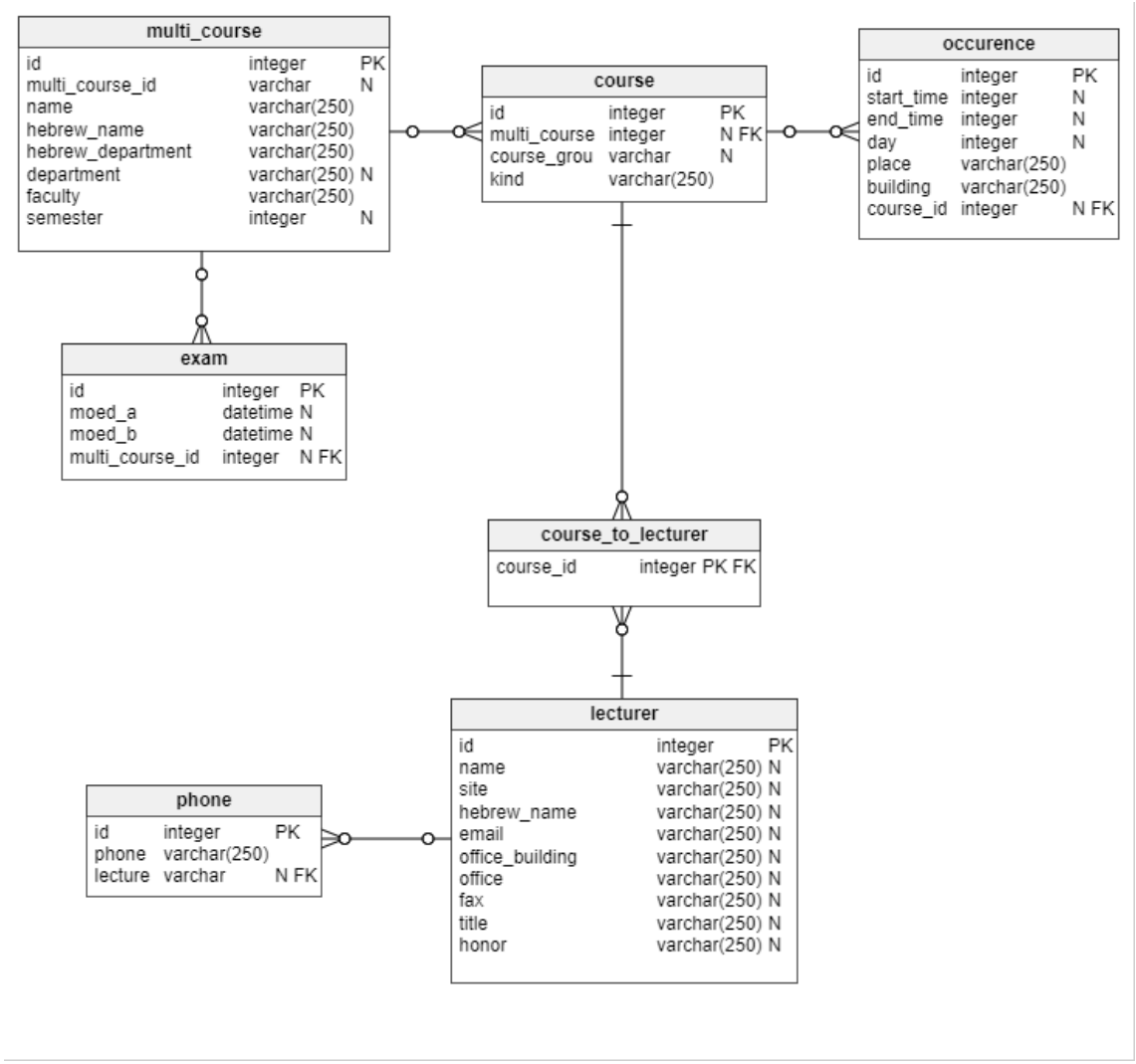

Figure 1: The database schema

- **Unary** - if $e \in \mathcal{E}$, then e is a unary logical form with $[z]_{\mathcal{K}} = \{e\}$
- **Binary** - if $p \in \mathcal{P}$, then p is a binary logical form with $[z]_{\mathcal{K}} = \{(e_1, e_2) : (e_1, p, e_2) \in \mathcal{K}\}$
- **Join** - if b is a binary and u is a unary, then $b.u$ is a unary denoting a join and project: $[b.u]_{\mathcal{K}} = \{(e_1 \in \mathcal{E} : \exists e_2.(e_1, e_2) \in [b]_{\mathcal{K}} \wedge e_2 \in [u]_{\mathcal{K}}\}$
- **Intersection** - if u_1 and u_2 are both unaries, then $u_1 \sqcap u_2$ denotes set intersection $[u_1 \sqcap u_2]_{\mathcal{K}} = [u_1]_{\mathcal{K}} \sqcap [u_2]_{\mathcal{K}}$
- **Aggregation** - if u is a unary, then $count(u)$ denotes the cardinality $[count(u)]_{\mathcal{K}} = |[u]_{\mathcal{K}}|$ In addition, we defined the aggregations max and min:
  $[max(u)]_{\mathcal{K}} = max([u]_{\mathcal{K}})$
  $[min(u)]_{\mathcal{K}} = min([u]_{\mathcal{K}})$
  and two special *latest* and *earliest* aggregations which relates only to unaries from type Occurences:
  $[latest(u)]_{\mathcal{K}} = argmax([start_time.u]_{\mathcal{K}})$
  $[earliest(u)]_{\mathcal{K}} = argmin([start_time.u]_{\mathcal{K}})$
  This allows us to handle complex queries on course times, earliest course on a specific day, etc... For example the question: *What is the earliest course on Sunday?*.

3.3 Composition

For each utterance, we derive all of its derivations using the semantic parser. The derivations are built using both the lexicon mapping between phrases to predicates and by using the composition rules. The derivations follows [Berant et al.(2013)Berant, Chou, Frostig, and Liang]. That is for each utterance we first use the lexicon mapping to replace phrases to single-predicates. Then, we iterate over the spans to construct logical forms such as intersection, join aggregation or bridging. We allow the derivation to skip words but instead use learning features to ignore bad derivations.

3.4 Bridging

The bridging process is designated to cope with implicit predicates. In our case the predicate is almost always implicit, thus we identified a few problematic predicates and added two types of bridging. The first is done during the lexicon stage in two specific cases:

- **Honor** - for example, in the utterance Dr. Moshe, Dr. is mapped by the lexicon to the entity honor:Dr, in this case the join revHonor.honor:Dr will be added.
- **Kind** - the words workshop, recitation, lecture are likely to appear without any explicit word which indicate about honor, in this cases the join revKind.honor:Dr will be added.

In addition, for every expression of type Multi-Course, Course and Occurrence, joins will be added in order to find the relevant predicates. For Multi-Course, a join with all of its courses and the course occurrences will be added, for Course, a join with all of its multi-courses and all of its occurrences will be added and for Occurrence, a join will all of its courses and multi-courses will be added.

3.5 Features

As described above, within the composition step, for each utterance we derive all of its possible derivations. However, some of the derivations will not result the correct answer (this will be handled in the training step by learning only from derivations that yields the expected answer) and some fit better to the model than others. To overcome this issue, we must add some features to our learning model that will eventually yield better derivations for each utterance.

We followed the paper features with some changes.

Rule Features - within the composition step, each utterance is translated to a derivation that consists of several predicates. In order to limit the number of join, intersection and bridging operations we added a feature that counts the number of times each of these operations were done as a part of the derivation. The purpose of this feature is to have well balanced derivations instead of derivations that have one sided preference for some of the operations.

Part of Speech Tag Features - we used two POS tags features as follows:

- **Lexicon** - For every lexicon entity operation, the percent of of the words in the utterance from the enity name is taken, and a feature of this average value is added. in addition, a feature of the percent of words which are written in capital letters is added.
- **Skip-POS** - We added an indicator feature vector that represents skipped POS words. The vector is of the size of all possible POS tags. Whenever the parser skips a word, it marks the bit representing the word POS tag. This will allow the parser to learn that some POS tags can be skipped and some can not.

- **Operation-POS** - We added an indicator feature vector that indicates when a join or bridging operations was done between two POS tags. The vector is of the size of all possible POS tags pairs. Whenever the parser apply a join or an intersection operation, it marks the bit representing the word pair POS tags. This will allow the parse to learn that some POS tags are much likely to be joined or intersected than others.

To detect POS tags we used **nltk** package.

Denotation Features - We added a denotation feature that indicates the number of answers the derived query results. This feature will allow the parser to prefer queries that have fewer answers. In addition we added boolean feature 'query is entity' which indicates if query returns an entity from its very structure; Query is recursively defined entity if it is intersection of entity with other expression, or join which uses attribute predicate. The base case is lexicon operations.

4 Training

4.1 Dataset

One of the main challenges of this project is to give highly accurate answers for any given question. In order to train the model to be accurate we had to have many questions with different phrasing to ensure the model is trained by a variety of question phrases.

We prepared a training set of 1000 question-answers pairs where each question can have multiple answers. For example the answer for the question *What courses does Jonathan Berant teaches?* is *Advanced Course in Natural Language Processing, Advanced Seminar in Nlp* whereas the answer for the question *What seminars does Jonathan Berant teaches?* is *Advanced Seminar in Nlp*. The questions must be on data that can be extracted from the university website such as: lecturer email, lecturer phone number, hours of a course and so on.

We started by manually phrasing 100 questions and gave the set of questions to AMT. Unfortunately, no one responded to us so we added some automatic questions generator that generates multiple questions on different entities. For example the question *What courses does <Lecturer> teaches?* can be asked on several lecturers. The problem is that multiple questions that have the same phrase doesn't add a lot to the learning phase. However, since some answers can be more exact than others, it guide the semantic parser to derive more accurately. With the automatic questions generator we reached 200 question-answers pairs for the training set.

4.2 Model

We followed the implementation of original paper which defines a discriminative log-linear model over derivations $d \in D(x)$. Given a utterance x and its derivation d, we define a feature vector $\phi(x, d)$, and $\theta \in R^b$ is the parameter vector to be learned. For each utterance x we define

$$p_\theta(d|x) = \frac{e^{\phi(x,d)^T\theta}}{\sum_{d' \in D(x)} e^{\phi(x,d')^T\theta}}$$

Given question-answers pairs $\{x_i, y_i\}$ we maximize the log-likelihood of only of the correct answers, formally we define $ans(d) = y_i$ where d is the derivation of x_i. The objective function is therefore

$$O(\theta) = \sum_{i=1}^{n} \sum_{d \in D(x_i): ans(d)=y_i} p_\theta(d|x_i) \quad (1)$$

We used a beam-based bottom-up parses which stores the k-best derivations for each span. We used $k = 500$. Also, we optimize the objective by applying AdaGrad.

5 Results

To train the model we used our custom built question-answers pairs dataset. The total number of question-answer pairs was 200. We divide the dataset to training and development sets where the training set consists of 90% of the dataset and the development set consists of 10% of the dataset. The set are selected randomly from the dataset.

Training the model with 7000 iterations took around 10 hours where at the end of the training, θ is saved to a file in order to allow the command line tool (see the README.txt file in the code) to answer the user's questions.

6 Discussion

The accuracy of the model on the training set was 40% and 27% on the development set. The results are worse than we expected as one of our goals in the project was to achieve a high accuracy question answering model and these numbers does not suffice.

We can see in 2 and 3 the accuracy on the training set vs. the number of iterations and k. We can see that k does not affect the training accuracy but the number of iterations increases

the accuracy. This is reasonable since the training set should be affected mostly by the number of iterations. On the development set we can see in in 4 and 5 that both accuracies vs. the number of iterations and k increases with the increases of k or the number of iterations.

We have several assumptions why we didn't match our goal:

- **Limited size of the dataset** - Our dataset is relatively small to achieve a high accuracy question answering engine. We would expect that the higher the dataset size is, the better the accuracy. Therefore, training on a $3,000$ question-answers pairs dataset would probably increase our accuracy to a reasonable number.
- **Questions quality** - in order to enable the training to learn correctly, the question-answers pairs in the dataset must be well phrased and have exact and accurate answers. Our questions was partially phrased by friends or family which can result inaccurate phrasing or answers.
- **Permissive Lexicon** - Our lexicon maps phrases to logical predicates. However, our lexicon mapping is quite permissive - that is, each phrase can have multiple predicates. This can result a very high number of derivations and therefore inaccurate (or slow) learning rate. To solve this, we could have used more lexicon features. That is, to save information on the lexicon predicate we chose in the derivation. For example the phrase *law* can be translated to a building or a department. But if we will save the number of times this phrase was chosen as a building and the number of times is was chosen as a department, the training would hopefully prefer to translate it as a department since it occurs more as a department.

References

[Berant et al.(2013)Berant, Chou, Frostig, and Liang] Berant, Jonathan, Chou, Andrew, Frostig, Roy, and Liang, Percy. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, pp. 6, 2013.

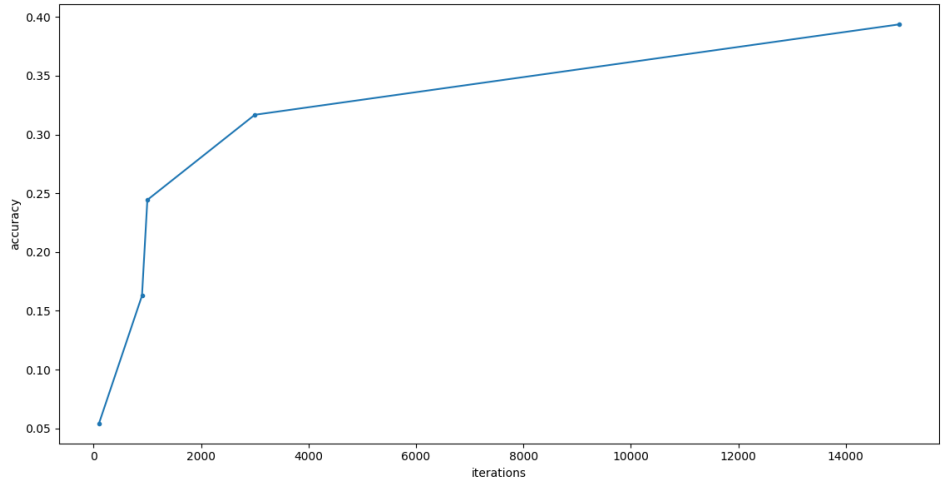

Figure 2: Train Accuracy VS. #Iterations (k=100)

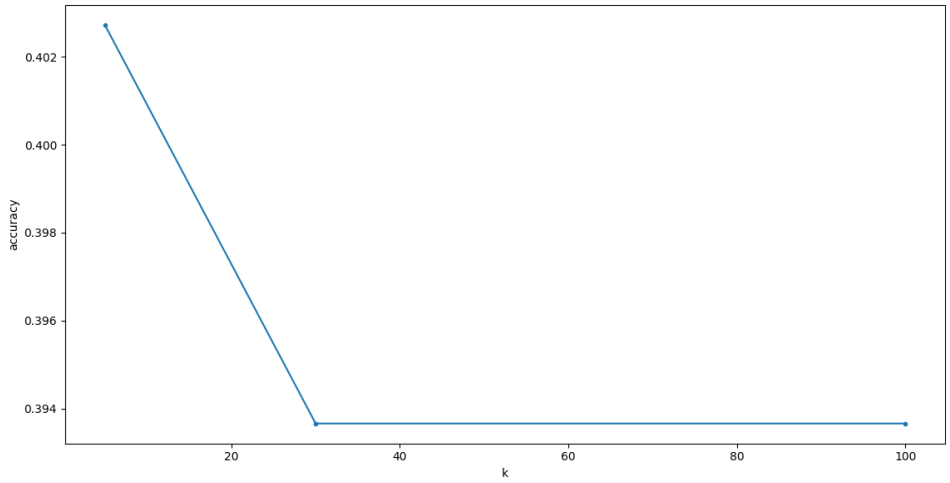

Figure 3: Train Accuracy VS. K (#Iterations=20000)

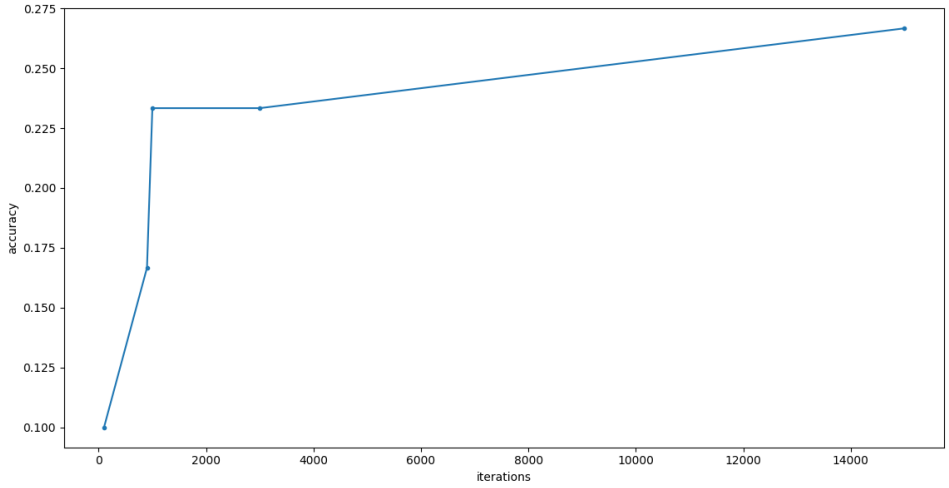

Figure 4: Dev Accuracy VS. #Iterations (k=100)

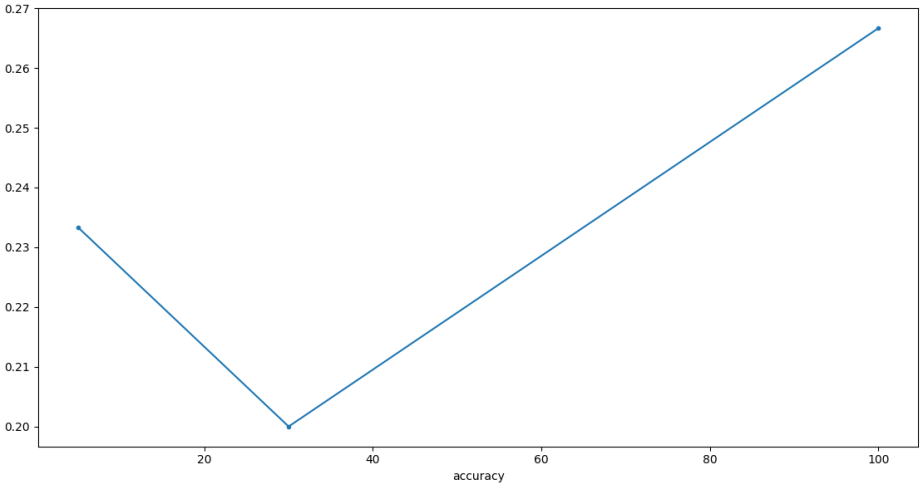

Figure 5: Dev Accuracy VS. K (#Iterations=20000)