University of Essex Online

MSc Artificial Intelligence

Transcript

Development Individual Project:

Presentation

Submission date: 21.10.2024

Contents

Transcript

Slide 1: Title Slide

Welcome to my presentation for the individual project assessment. I would like to give you an overview of my agent and what I actually did.

Slide 2: Agenda

Let's start with the agenda. First, I'm going to give you an introduction to this project. Then we're talking about the requirements followed by the architecture and design I chose for this agent. We're going to look at the techniques and libraries with rationales. Then we're going to have a demo evidence or let's say an execution of the whole thing. That's going to be a video in my case. Then we're going to look at the testing unit and functional testing that we did. Last but not least, we're going to have a look at the results, metrics and discussion.Slide

3: Introduction

*N*ow, let me start with a short introduction to the project. The idea is basically the implementation of an agent that runs within the context of the digital forensics domain. On the right side, you can see the end-to-end process that we defined in our group project during week six. The blue arrow indicates that the main process is this line there from a scheduling trigger to the end of the process. The bottom part or the one that goes below is just for quarantining files if there is a security threat. Let's quickly look at the process there. We are getting files from a share. We are extracting all relevant file information and analyze them. We decide whether or not there is a security threat. If not, we're going to extract the contents of the file, analyze PII, personal identifiable information and set the DC, which stands for data classification. Then we're going to record it in an audit block and create a report. Last but not least, we're going to log all steps and results to this audit block and then end the process. Now in my scenario, I built the port that is about extracting file contents, analyzing PII, and setting the data classification. The goals of this part or of my agent that I built is to, first of all, detect and redact PII. I'm using Presidio and Spassi for this according to Microsoft, Honiball and Montani. I'm going to estimate the data classification using SkiKit Learn. SkiKit Learn, yes, sorry. The text classifier for data classification according to Petre Guza. And then last but not least, I'm going to return the files keeping their formatting. Idea is I'm going to grab a file. I'm going to extract information of it. I'm going to take a look at the data to understand if there is PII and redact it. I'm also going to look at the data to understand the data classification it has according to a model that I'm going to build myself. And then last but not least, I'm going to output this information so that the user understands if there is PII and he also sees that it's redacted. And then it also going to show the user the data classification.

Slide 4: Requirements

Now let's take a look at the requirements. We have some business requirements. I already mentioned them. We want to see data classification for each file. We want to detect personal identifiable information on documents and redact them. We're going to want to have different input file formats. We're going to want to preserve file formatting for defined files. So it's the idea that for example a PowerPoint with a certain layout is also outputted in the same layout even if we have to redact certain informations. And then last but not least, we want to add a capability for policies so that we can give the agent additional information and additional rules to base his actions on. And we're going to have some technical requirements and design. We're going to use the language Python for this agent. The architecture should be modular. That means we want to be able to exchange things if we need to. It should not be built as one big monolith. Then we want to run this code locally and offline. And then last but not least, we want to extract text and then apply feature-based methods. This aligns with classic text processing assumptions which is about converting documents to analyzable text then applying feature-based methods. And that's according to southern McGill. Note for the scope of this project, I excluded OCR. So if there is text on images or any other form, it may not be taken care of by the agent. And last but not least, I wanted to point out the disclaimer that all I built is running in a POC, so proof of concept context. There would be many things to look at if you wanted to use the agent in real-world scenario, especially when it comes to accuracy, security, and scalability.

Slide 5: Architecture – Design

Now let's take a look at the architecture design. This is a very high-level design of the agent. We start on the left side, the yellow box, we have the file storage. Then we have the agent itself, green, which gets files from this file storage. The agent then calls PII detection, which actually detects if PII is in this file or in the text of the file. Same goes for the redact PII, which simply means that if there is PII, the agent wants to redact it and mark it black so that it's not readable anymore. And then it also wants to do data classification detection according to the model that we're going to build. And that's the point six. This DC detection is using the model that we built for data classification.

Slide 6: Architecture – Sequence Diagram

Now let's look at the architecture sequence diagram. Now when it comes to the architecture sequence diagram, it's pretty straightforward. It may look complicated on the first look, but basically we have the data input and the bot which work together in terms of files coming from the data input to the bot. Then we have the bot calling the input-output-utils, the load-text part, so it gets the text out of those files and this function returns this text to the bot. Then we have the detect-PII part which is called by the pot and in return it receives those PII spans. Then the bot will call the mark-PII-redact function to actually redact this PII that the PII-detect found. Afterwards, the agent will ask the DC predict label function to actually predict the data classification. And then last but not least, it will use the input-output-utils to save, actually, then redact the PPTX, so the PowerPoint, and then redact the PDF so that those files will be stored again as PDF or PowerPoint files. If we wouldn't have this function in the input-output area, then we would basically receive txt files as outputs instead of PowerPoint or PDF files or whatever we input there. Here it's important to understand that whatever file format we have, we need to add some functions to this part, so currently it's set up for PowerPoint, PDF, and txt files. And then last but not least, the data output is being generated.

Slide 7: Techniques & Libraries (with rationale)

Now, let's talk about the techniques and libraries with Rationals. So first of all, the techniques, we are using PII detection with Presidio and Spussy. Why are we doing this? Because that's a strong baseline without training, and it also works locally offline, so we don't have to call any online service or use APIs. Then Reduction with Same Length Block Replacement in text, that's just the black boxes that we place on the places where we have PII, so that's just visibly clear. You will see this in the demonstration later. Then we have Document Classification, so the data classification. We use TF IDF 1, 2 grams, and Logistic Regression here. Why do we do that? Because it's fast, it's explainable baseline for short text, and it's just also a machine learning part that we wanted to introduce as part of this assessment. Then we have Policies, so if PII is found in the document, we want to uplift wherever the data classification has been set by our model. We want to uplift it to highly confidential. That's a deterministic safety net, and it's also just to introduce policies to the agent which we could build on in case we wanted to extend this agent. Then we have Input, Output, and Formats. The text extraction for Analyzing Output in the same format, that means we want to keep the format that we had in the first place also in our output and uniform the Analyzes path. Last but not least, from the techniques, we're going to do unit testings and functional testings in combination that helps us to be fast and to still have high quality testing. On the right side, we have the different libraries. We are using Microsoft Presidio, that's an Analyzer Anonymizer, because it's very mature, it's available, it's a trained model. Then we are using Spacey. Here is in the Encore Web SM, because it's lightweight, quick to install, very straightforward, and it's good for a POC. Now, in reality, we may use some other models here, but that's great for what we are doing. Then Skizzyk Learn, Sikid Learn, I'm not sure how to pronounce this. There, it's also about just a strong classical baseline, fast training, inference, reproducibility. That's with the data classification model that we built, and it is really straightforward, and we took a look at this library in the Machine Learning class already. Then we have JobLib, because it persists vectorize a classifier, so it's where we store the data model, the machine learning model that we built. Then we have the Python docX and Python PowerPoint TX, or PPTX. That's just to handle docX files and PowerPoint files less, but at least PyMu PDF. That's the same as the one before, but for PDF files in this case.

Slide 8: Demo Evidence (Execution)

Welcome to my demonstration of the agent that I built. I thought it makes most sense to start with the readme file. You can find some basic instructions here. Most importantly, the file structure, which is required to run the code properly. You can see it here on the left side on my end, where I have the different files and folders. Then we have some general requirements. I built that using Python 3.12, but you can also run it with Python 3.10 or higher. You'll need to install some local packages. I've created this requirements file for that, which will help you to simply install it. With this command down here, there are no external APIs or connections required to fool agents run locally. You can simply follow those steps. Step 0 is optional but basically installing all the dependencies requirements. You have to download a certain spacey model that I am using. Then we have to train the document classification DC model. That's also optional, to be honest, because I will provide the final model. Then we need to place the test files data into the input folder here. You have all those files. The output folder is currently empty. That will change once we run the agent. Here with Python bot.py, what is the agent, the orchestration, the orchestrator part of it.

We run it and then we check the data output folder to see the final files, including their data classification as a label in the name. All the PII is redacted and blacked out in the final files. Here is simply the node that I will provide the model trainer.py as well, which will help you to build the model. You can find it here as the data classification classifier and vectorizer joblib files. Back quickly to the model trainer, because it's a good example for how the files are structured. I added a comment here that explains what this file does. Then we have some comments on the important parts. Here, for example, you can see where the training data is stored. That would also be very important in case you want to rerun the model trainer. Inside here, a CSV file I used around 150 plus entries of public, internal and highly confidential reflect data. All of them have a class. It is supervised learning that we used here. A simple classifier and the agent is using this model to do the DC classification. Let's take a closer look at the bot.py, because that's basically our agent that orchestrates the whole environment. The agent itself is again described here. You can see it basically does everything from reading the data, extracting the text up to PII detection and reduction. Then we have the data classification part. We also have the policies that are applied. Then last but not least, it creates the output files that include the data classification and the redacted black boxes for PII. Just to give you an example of how the code is structured, we have the detect PII for example here. That's done in this scenario here, the second step inside this main function where it calls the detect PII. In this case, that's in our PII file, which we can find over here. Then you can see this function down here. Again, it's described what it does. Also for this file, as you saw before, we always have a comment that explains the file more or less, and then different comments down in the code that explain the different functions that we are using. That's how the whole thing is structured. If I go to mark PII, I will see the same IAUtils as well. That's also a very important file. That's the input output IAUtils. All the files are important. The agent will not run if some of them are missing. Here it's getting a little bit more interesting, just a type of form lant here, correcting it quickly. That's where we do certain things in data manipulation. Data classification here is pretty straightforward because it basically uses the model that we created and stored. You can see here again, it loads those job lib files that you saw earlier in the model folder down here. The configuration is also very interesting, and that's where I want to be at before starting the agent. Down here we have the policies part that has been briefly explained in the requirements slide. Here it's all about whether or not something should be put to highly confidential in case that there is PII. What the agent does is identifying PII on one side and then it's also identifying the data classification according to the model that we trained. Now if we say, hey, no matter what the model outputs, it is always highly confidential. If there is PII, we would simply set this flag to true. Then once we run the agent, it would overwrite whatever the data classification model returned if there is PII. It simply set it to highly confidential. I think that's also a very interesting case to look at for the demonstration, and that's what we're going to do. I'm here in the folder, same as here in Visual Studio Code. We have data, output folder currently empty. The input folder has those nine files inside. Now an interesting file is this PDF here, the public one, which should basically out turn public. But there is some PII inside. We have this Moxmooster mon in here and down here. So we expect from the agent to mark this as PII. We expect from the agent according to the model to classify this as public. Because of the contents that it has with this public agenda. Those are things that trigger what I added to the machine learning model. Now, as mentioned before, we have this policy and currently it's set to true. That means if PII is detected, which is the case for our public document, it should set it to highly confidential. If we set this to false, it should set it to public. So what we're going to test. So let's see what happens if I simply run the bot. Sorry, close this. So pi bot.pi. Now it's going through all the steps here. It takes the data up to writing it back into the folder. It just did it. We can see this here, the results for each of the files. But we're going to look at this in the folder. So let me quickly copy this again so that we can go to input and output. Here again, the input folder, the output folder looks a bit different as you can see here in this comparison. It actually has the data classification in Bracets at the end. Also, if I open something like this PDF that is highly confidential, I can see that things are now blacked out here in comparison. The input file and here the output file where you can actually see that the name, the email and the phone number have successfully been detected as PII. And the file has been marked as highly confidential, which is according to what we want to get. Now for the easiness, I added always the expected data classification into the name of the file. And then we have to brace it so we can easily compare whether or not the agent did what the agent should be doing. And we can see here that there is one difference. We have this PDF file that is actually marked as highly confidential but should be public. And here just as a reminder, I will open both. That's the one that I actually wanted to take a look at because we said it's public. It should be public in our machine learning model that we trained because we trained it accordingly. But there is PII inside and now the policy that we set to true flagged this as highly confidential and overwrote the data classification from our machine learning model. Now I want to cross check that by going back to the configuration setting this flag to false, which means now it should not overwrite it and then simply run the agent again. What is happening here now is that the agent should be creating a new file public with embrace it's public and it just did it. Because now what it does is it's still doing the PII part but the data classification is not set because of the PII but rather because it actually got this return from our trained model where it said this data classification must be public. So here again we see that the agent worked. Just to give another example for here word file and see how that looks like. We can see again that it did some things in here. Let's see how the input file for highly confidential looked like. It was like that so we can see that what it did is it crossed the country and the I-Band Banking number because it identified those as PII. Now of course PII Switzerland that could be a discussion but for this proof of concept that's totally fine. Let's take a look at some last thing which is the PowerPoint presentation. Let's take just the public one and also the public as a result because here there is something important and that's the formatting of the files. We don't want it to change when the agent does work on it so we can see here on the input file how it looked like and the output file pretty much looks the same. So for us that means that this was successful and we did not change the formatting of the files. Now that's it from the presentation. You can rebuild everything using the readme file and the files I provided. I hope that that makes sense and if it's unclear feel free to reach out to me.

Slide 9: Testing (Unit & Functional)

Then after the demonstration, we have the testing unit and functional. Here it's important to understand that we did some unit testing, so we did PII detection testing, redacting testing, data classification, model testing, loads, and save file function testing. Now, how did we do that? We did that by simply using those functions isolated from everything else, so that we could actually test this specific unit, and the results were pretty good in terms of we had some failures, but using the unit tests, we were able to quickly resolve those and then finally move to the functional testing where we did a lot of test cases. I won't go through all of the test cases because of time concerns, but I want to point out the test cases seven and eight, because it's the same input data, but another expected result. The reason for this is that one time we wanted the policy to be active and one time not, and that's also the case that we had in our demonstration.

Slide 10: Results, Metrics & Discussion

Now, what are the results? The results basically were that the PII detection worked as intended. The data classification detection worked as intended as well. The policy option worked as well as intended, and now we also have machine learning metrics for our machine learning model that you can see on the right side, so it was pretty good. I would say not perfect, but a reliable data classification detection. The results overall were very satisfying, and I'm happy about those. Discussion and outlook-wise, I would say it would be a great idea to include large language model functions for further analysis that could be used, for example, to analyze the files before or the responses before they are stored in the output folder, or maybe even increase the accuracy of the data classification returns or the PII detection. That also includes various models to ensure PII detection and DC estimations are even better. We could include more file types. As mentioned, I'm using PDF and PowerPoint files as well as text files. Being sophisticated UI would also be a great point so that actually someone could steer the configuration of the tool, for example, the policy set to false and true currently. I'm doing that in the backend, in the code, of course, for users that may not be the greatest thing to do. Now, you can see here on the right side just as another, let's say, proof of the agents' results that it actually did black out PII and on the button it did the DC estimations for the different files. That's great. I'm overall very happy with those results.

Slide 11: Thank you

With that, I'm already finished. I thank you very much for listening to my presentation. I hope you also liked the demonstration in case you have any issues or questions when you want to run the code yourself, feel free to reach out to me. Thank you for listening.

References

Microsoft (n.d.) Presidio documentation: Analyzer and Anonymizer. Available at: https://microsoft.github.io/presidio/ (Accessed: 5 October 2025).

Honnibal, M., Montani, I., Van Landeghem, S. and Boyd, A., 2017. spaCy: Industrial-strength NLP [online]

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, pp.2825-2830.

Russell, S.J. and Norvig, P., 2021. Artificial Intelligence: A Modern Approach, Global Edition 4e.

Salton, G., 1983. Modern information retrieval. (No Title).

Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.