Challenge Project Answers

Antonella Schiavoni

PART 1

What additional information can you extract from the event titles?

We could extract additional information such as:

- The geographical location of the event (country, city, state)
- The name of stadium or theater (or if it's a virtual event)
- The date of the event
- Any price in the title
- Whether the event is part of a world tour
- The kind of event (Football, Music, Theatre, etc)

What additional business value do you think this information can provide?

- We could do customer segmentation to show events to customer that are located in the same or close range to the event.
- We could also disregard passed events based on the date, as they are no longer relevant for the ticket business.
- If we had the information about which events our users are interested in, we could also send them notifications that certain event will be available in the web prior to the release date.
- Doing fraud detection: there cannot be two events happening at the same time the same day with different artist unless it's a festival.

Can you differentiate between your confidence levels for when different artists are extracted?

With the current implementation for this fast prototype, it's not possible. However, it would be extremely useful as we could filter results using a threshold and hide those results that have low confidence. By using a custom NER model built in-house, we

could specify that confidence is returned in the predictions. For Spacy, there's currently an open issue so this confidence is added.

PART 2

What is the best way to make your predictions available for the rest of the product to consume?

By creating an API and exposing an endpoint that returns the predictions.

Will your solution differ in case the predictions are needed in real-time vs offline batch jobs?

Yes, It will. In the scenario in which offline batch job is needed, it will be necessary to use a tool to accumulate the raw data before we make an ETL job to make it available for future analysis. Normally these ETL jobs run once a day. Some tools that can be consider when we need to apply offline batch jobs are:

- Spark
- MapReduce
- Google BigQuery
- Amazonn Redshift

In the scenario a real time solution is needed, we will need to integrate the service with a tool that allows us to process and analyse the results in real time so that fast business actions can be taken (for example, if we are trying to detect fraudulent events). Some tools that could be used in real-time are:

- Spark Streaming
- Confluent KSQL
- Amazon Kinesis

- Google Cloud Dataflow.

What are the different deployment options you can think of?

We have to consider two different deployment options. First of all, the deployment of the backend service that loads the model and expose an API for other services to consume predictions. For this, pipeline of continuos delivery will upload the image of the service to a container registry every time the main branch is updated and then that image will be automatically deployed to production. A way to do this is by combining CircleCI + Spinnaker, but there are tons of tools for CI/CD.

Secondly, we have to consider the deployment of the machine learning model. For the sake of a fast prototyping for this challenge, I haven't built a model artifact but if this solution was to be deployed in production, the best alternative is to build a model artifact and plan a special deployment pipeline for this model. For example, we could design a retraining pipeline so each time we gather new event titles data, we do a manual annotation process, store the annotated data in AWS S3 service (or similar), using airflow we could trigger the retraining pipeline (with some cadence that will depend on a business analysis in which we compute the frequency with which new entities appear in the data). Once the training pipeline finishes, the new model is stored in a model registry, again this could be AWS S3 service (or any other similar cloud alternative). When the service detects there's a new model version in S3, we just have to update the model the service is using by using boto3 library.

In a productive architecture we would also have a layer to perform A/B testing, so we could use different versions from the model registry to redirect the traffic based on the configuration.

PART 3

How would you approach other teams to put it in production, and make sure they consume your predictions?

There will be a lot of talks needed with different teams (but it depends on the amount of teams, and the responsibility of each). To start with, which will be the team that consumes these predictions? probably a team that handles the web application of

ticket swap as the goal is to extract the entities to detect the events customers are looking for. So the first talk would be with them. The goal would be to agree on the API contract. The results of these predictions should be stored somewhere. For example, we could have Kafka connected with the ML service, all the predictions the ML service produces are written there. Then, with an ETL process, we could extract these results and make a dump of them into a database. For these there will be a necessary sync with the Data Engineering team so they can set up the pipelines. As regards the Data Science team, they will be able to access the predictions and compute business metrics after the ETL pipeline from Data Engineering is run. However, it will be also necessary to meet with DS as there might be other additional information, apart from the predictions, that they might find interest in. Finally, if there's a platform team, we will need to sync with them too, so we can agree on how to integrate the ML service to their deployment pipeline.

How will you convince the different business stakeholders about your solution?

I'll present a document explaining the pros, cons, and the goals of this solution, how it improves the current existing one (if it exist) in terms of money saved, time reduction, and different performance enhancements. Additionally, I will show business metrics improvements that support the argument.

What additional documentation will you need to communicate with the corresponding technical teams?

A technical document in which we show:

- The integration with other teams
- The goals and limitations of the ML solution
- The api contract they will have to follow if they want to make use of the prediction service. We could use API documentation like swagger.
- The schemas with which the prediction data will be stored.
- The monitor and logging system that will be used, together with the relevant metrics for the service and the business metrics that will be tracked.
- The CI pipeline for the code and component validation and tests.

- Which will be the continuous delivery pipeline (and components) that will be used, including the re-deployment process.
- Where and how is the retraining pipeline defined, including which tool will be used.
- An entire system diagram that shows how the ML service will interact with other teams.

Will you need to collect additional data for your future iterations on your solution?

Yes, It's the key to make the model recognise new (or variation of the existing) entities. By collecting additional data periodically, we can tag this data to find new entities and then retrain the model so it learns to detect them.

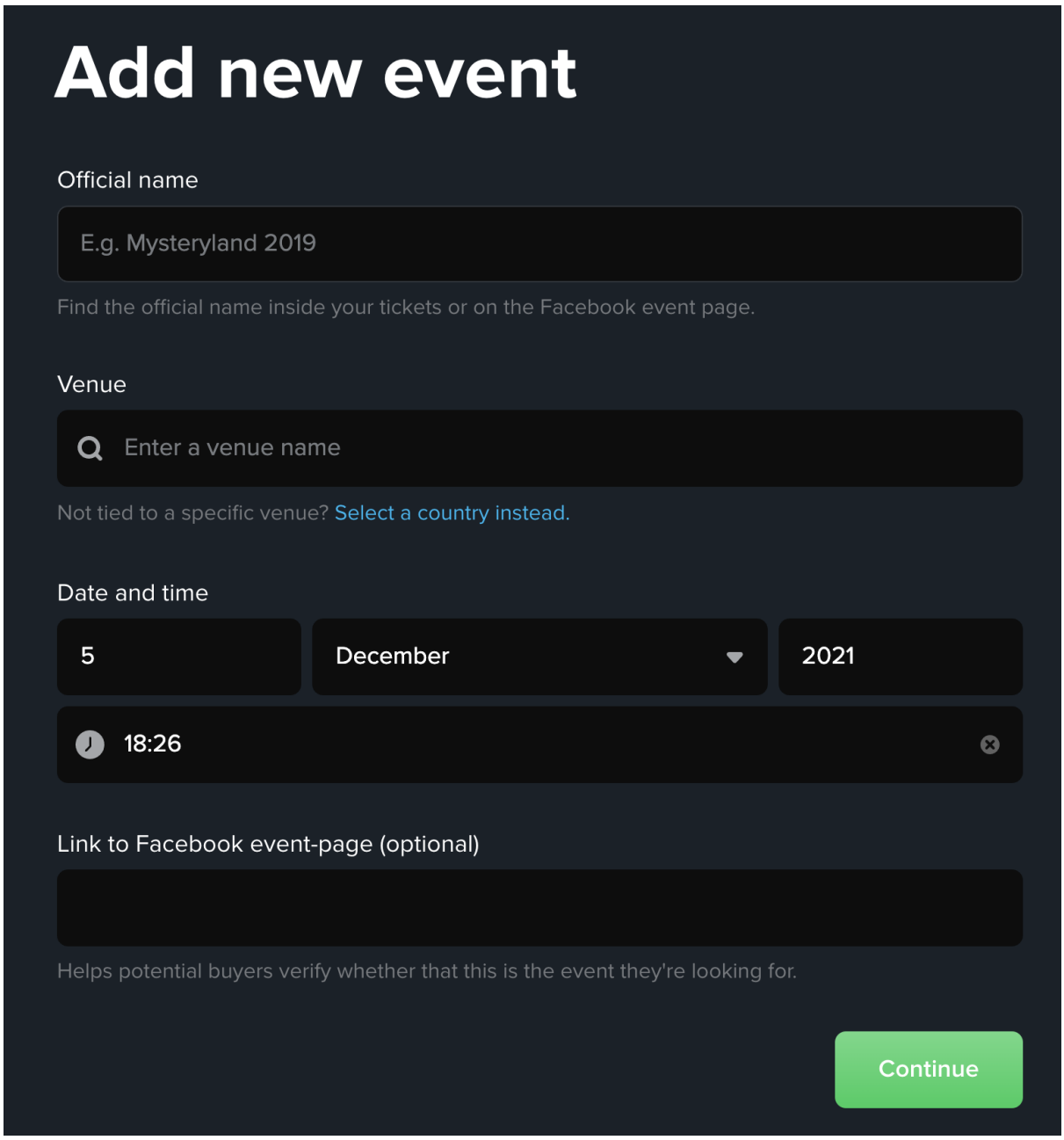
Next Steps

- Enhance the performance of the SynonymEntityRecognizer by manually adding a more robust list of synonyms.
- Sample data regularly from production, using search data, and manually tag the search. Then, fine tune the space NER model to increase the accuracy of the entity recognition.
- Add metrics: currently, I don't really know how well the entities are recognized, and if not which are the errors. This can be done by creating an evaluation dataset which can be used as ground truth, then infer the entities of the titles form this dataset, and compute the metrics such as precision to have an intuition of the model performance.
- In production, add a service monitor using datadog (to name an example) and create different dashboards to measure and monitor server latency, server successful request, server error, server resources (such as cpu, memory and cpu throttle)

- Create swagger api documentation.
- Improve preprocessing: right now the preprocessing step is only splitting sentences, but can be improved if we look deeper in the data and find some more valuable insights of the patterns in the title. For example, there are a lot of duplicated or partial matching event titles in the dataset that could be deduplicated before using the information in the server.

  In the artist.txt file, for example, Fish (Record Label), Future, James Taylor to name a few appear twice. In the event_titles.txt, Scott Bradlee's Postmodern Jukebox, Richard Groenendijk - Om Alles and The Australian Pink Floyd Show - All That You Feel World Tour to name a few, are also duplicated.
- Implement RegexEntityRecognizer object that will load a list of regex used to extract information from the title. Most of the titles in the dataset follow a pattern. Some of them are listed below:
  - "**w/**" , "**&", "and**", "**+**" "**/**" are used to state that two artist will appear together in the same event
  - "**-**" is used to separate different kind of information. Depending the context, what comes next to the dash it might be
    - a city/country
    - a football club's name
    - an artist
    - a date
    - the place where the event takes place
  - "@" is used to indicate where the event takes place
- To increase the accuracy of the entity recognition in the football matches, we can create a regex and add it to the RegexEntityRecognizer to detect football club's name. Most of the matches titles contain "**vs", "-" or "v"** to indicate which team plays against which so this could be easily detected with a regex.
  - Another useful information that can be extracted from football match titles is the location of the match, as normally the team that plays at home plays first.
- Improve server tests. Right now there are no end to end test for the service, but it can be done with a some work that includes a refactor of the service code.

- Another idea is to have a more structured input form in which we splicitly ask for the artist when an event is being created. Right now the input form looks like this:



However, if we can have an extra field for the artist(s) name, event type and the location, that would help us tag the event title, and also will help in the retraining NER model pipeline as we could learn to identify the entities requiring less work that manually tagging.