

Project Documentation Explain-AI

Aleksej Strassheim, Konstantin Strassheim

Contents

Contents	1
1. Targets	3
2. Development Hardware	3
2.1.1. Developing Machine	3
2.1.2. CI/CD and Git	4
2.1.3. Cloud Provider	4
3. Dataset	4
3.1. Description	4
3.2. Problems	4
4. ML Models	4
4.1. Lemmatized TFIDF Neural Network	4
4.1.1. Description	4
4.1.2. Implementation	5
4.1.3. History	5
4.1.4. Problems	5
4.1.5. Results	5
4.2. LSTM	5
4.2.1. Description	5
4.2.2. Implementation	5
4.2.3. History	6
4.2.4. Problems	6
4.2.5. Result6	
4.3. BERT/Distilbert (Hugging-Face)	6
4.3.1. Description	6
4.3.2. Implementation	6
4.3.3. History	7
4.3.4. Problems	7

4.3.5.	Result8	
4.4.	Support Vector Classifier	8
4.4.1.	Description	8
4.4.2.	Implementation	8
4.4.3.	Problems	8
4.4.4.	Result8	
4.5.	Common Problems	8
4.5.1.	Implementing ML for a Webapp	8
4.5.2.	Saving Hash code of a trained ML	9
4.5.3.	Reloading model	9
4.5.4.	TensorFlow/Keras Framework	9
4.5.5.	Determinism	9
4.6.	Conclusion	10
5.	Preparation	10
5.1.	Prerequisites	10
5.2.	Tools	10
6.	GitHub Authentication Codes	10
7.	Environment Settings	12
8.	Compiling and Launch (Production)	13
9.	Debugging	13
10. ..	Data and Database Architecture	15
10.1.	JSON Item	15
10.2.	File DB	15
10.3.	Azure File Storage	15
11. ...	Code Architecture	15
11.1.	Explain Engines	15
11.2.	Machine Learning	15
11.2.1.	Abstract	15
11.2.2.	ML Models	16
11.2.3.	Training	16
11.2.4.	Data Preprocessing	16
11.2.5.	LIME Extensions	17
11.2.6.	Fixed Files	17
11.3.	Backend (Django)	17
11.3.1.	Root Files	17
11.3.2.	API Folder	17
11.3.3.	API Sockets Folder	17
11.4.	Frontend (React)	18
11.4.1.	Folder src	18
11.4.2.	Folder build and public	18
12. ...	Testing	18
12.1.	Major Challenge	18

12.1.1.	Smaller set of data to test if training runs.	18
12.1.2.	Hash code	19
12.2.	Other Challenges	19
12.2.1.	TensorFlow Session	19
12.2.2.	Pretrained Bert Models determinism	19
12.3.	Run Tests	19
13. ..	Application	19
13.1.	Home Screen – Item List	20
13.2.	Home Tasks	20
13.2.1.	Delete Prompt	20
13.3.	Show/Edit Form – View Mode	21
13.3.1.	Detail view	22
13.4.	Edit Mode – Edit	22
14. ..	Progressive Web App	24
14.1.	Android	24
14.2.	IOS	24

1. Targets

The major target of this project is to make Machine Learning prediction decisions visible to a user via an web app / PWA. Another target is to compare runtimes of different explain algorithms together.

We decided to use a progressive webapp because of the following reasons.

1. Optimal support for presentation by lots of JavaScript libraries and dynamic text + responsive design frameworks
2. Running ML requires highly scaled and performant resources. The scalability is the most issue here because the ML models have to be trained for a specific set of Memory and the Processing Unit (GPU or CPU) which cannot be provided by every client.
3. A progressive web app that runs inside a cloud provider is a common up to date solution which makes it easy for the user to run the app on many devices and also to install and update the app.

2. Development Hardware

2.1.1. Developing Machine

We got basically 4 possibilities for development of this project.

One offer was to use Google Collab to run the training on a GPU. We denied this solution because we need a scalable training support for our deployment and a more agile way to run a ML several times to fit the optimum settings for the deployment hardware. Even with a Pro subscription you have no guarantee to get the advanced GPU resources for the calculations, disregarding the scalability of the final ML Model. Therefore this solution was useless for us.

The second possibility was to use dedicated GPU VM's from cloud providers. This solution is first very expensive, but also the provided GPUs don't reach nearly the performance of a RTX 3090 card. Also, the agility is not given because we would have to copy around the trained model files through the internet. Therefore it was also denied by us.

The third possibility was to get access on the TU Darmstadt GPU cluster. This solution would provide us a lot of computation power but not enough flexibility, because we would have to copy around the saved models to test and check them into git to run them on CPU's again to make them work on a webserver. The training would not reach the agility that we need for this project.

The last possibility was to upgrade our both home pc's with RTX 3090 GPU's and the MB RAM to 64GB, to have identical computation power. Disregarding the highly speculative prices of the GPU during the start of our project we decided to privately invest in our hardware also for future project with ML. Using Windows System Linux (WSL) with CUDA support gave us enough agility to rerun model trainings until we found the optimal ML settings.

After especially the ending month with a lot of issues with the ML training and customizations, we realized that we would not barely reach the current result without choosing this solution. Therefore the choice for custom hardware was perfect.

2.1.2. CI/CD and Git

Like in previous projects like our B.Sc. Praktika, we decided to use the free version of Microsoft's Azure-Devops cloud solution, because it provides everything you need for a project (Board, Git Repo, Pipelines) out from stock for free, without the need to modify anything.

2.1.3. Cloud Provider

Due to its easy pipeline integration into Azure-Devops we choose to deploy the test site to a Microsoft Azure Web application. Another reason for this choice was the possibility to up- and down-scale the application hardware in runtime. A feature that was mandatory for us to find possible hardware settings.

3. Dataset

3.1. Description

For the basement of our Explainable AI project, we had given the Kaggle dataset "Fake News" <https://www.kaggle.com/c/fake-news/data>. The major columns we used for training were "title

3.2. Problems

The dataset was pre-spitted into test, validation, and train data. This we had to merge back to provide a custom split by a custom split size. Another issue was that the labels are in a separated file. All these issues we could fix in a one-liner pandas command.

The full-text values had no identical newline values or formats.

At all the max accuracy to reach was around 90 percent, which was approached by all the models. Therefore, we assume that this is the training limit for the dataset.

4. ML Models

4.1. Lemmatized TFIDF Neural Network

4.1.1. Description

This model starts with lemmatized preprocessing and then transfers every string into a term-frequency-inverse-document frequency vector. Very important to mention here is, that the order of the words is not regarded, but just the count of their appearance. TFIDF in comparison with simple Count-Vectorizing also limits the effect of binding words like (a, the, not) through dividing with the inverse document appearance of those words. In short, it values words that are unique in a document higher than words that appear in all documents.

4.1.2. Implementation

For the lemma preprocessing we use spacy with "en_core_web_sm" library. For the TFIDF Tokenizer we use SKLearn TFIDF Vectorizer. Finally, on that we build a TensorFlow Neural Network with RELU Layers that end with a Softmax Layer for classification.

4.1.3. History

This was the first model that we implemented and that we used as base to build the software architecture for the ML Training and the Webapp. Until the Midterm presentation we had just this model present because we focused on the architecture the presentation and the unit tests.

At the beginning we trained it with the titles of the dataset (short text) to find the optimal settings. In the end of this project, we expanded it to the full content text column of the dataset where the first problems appeared.

4.1.4. Problems

The first problems appeared with changing the training column to full text. The spacy preprocessing takes about 13 Minutes to transform the complete training data. For finding the optimal ML settings it can be skipped by applying already lemmatized columns and Identity-Preprocessing to the training class during the setup. Afterwards it has to be set back for the final training.

Another problem with taking full text is a memory overload on several steps, which does not always show up in an error but just getting the process killed. It took very much effort to debug every step with every possible setting to maximize the word length while minimize the training time.

At the first the max text length was only possible with 100, but after figuring out how to limit the max features of the TFIDF Transformation this limitation increased rapidly to a max length of about 10000 with max features of 20000.

At the end we found some possible settings which also work for the CPU. The unit tests which always validate a main branch pull request on the CPU in a completely new environment were really helpful here and caught a lot of deployment errors before appearing in the final application.

4.1.5. Results

The results of this Model are deterministic and reach around 88 percent of accuracy which is enough. The max text length is set to 10000 and the max feature size is 20000.

4.2. LSTM

4.2.1. Description

Since we use the TensorFlow word vector transformation it is not required to preprocess anything here. The preprocessing was required for TFIDF to get the word counts at the word vector transformation any word is directly mapped to a word vector.

A long-short-term-memory layer is a recurrent neural network with forget/delete memory functionality. Before BERT attention model, it was the up-to-date way of NLP processing with neural network.

4.2.2. Implementation

We have chosen TensorFlow because it has a massive performance opportunity against Spacy. We can say that spacy is almost unusable for this application because it takes a lot of time for transforming the

whole Training set (>1h), and also required the bigger “en_core_web_md” or “en_core_web_lg” packages which slow down the process even more.

After the transformation we use an LSTM, followed by a simple RELU Layer with Dropout, and finished by a Softmax activation which provides the classification.

4.2.3. History

The first implementation with title columns (short text) was very simple and worked well, because the developed architecture and the unit tests did their work well. We had to use around 60 epochs on a batch size of 3072 which worked fast enough to reach the training accuracy maximum of 90.6 percent. Because of the deterministic implementation this could be figured out very easily by training 100 epochs and see at which epoch the validation accuracy goes down and the model starts overfitting (training accuracy goes up while validation accuracy goes down).

The problems came with implementing the full text columns. We had to adjust the max features with the max text length comparing to the batch size to reach an optimal result.

4.2.4. Problems

Like on the other models we had to figure out the memory overflows which resulted in just killing the process. Later we had to increase the max text length for the transformation to 10000, with significantly reducing the batch size to 128. This in first resulted in a very long training (on 60 epochs), but we also figured out that the overfitting starts with full text after epoch 6. This was very helpful to provide the final result.

4.2.5. Result

At the end we have a usable and powerful classifier at the end with an accuracy of about 89 percent accuracy.

4.3. BERT/Distilbert (Hugging-Face)

4.3.1. Description

Since BERT is the current standard for NLP text classification, which is also used by the PEASEC department, it was a requirement to use it in our explanation framework.

BERT is a pretrained attention model. The training of the basic model takes several days, but once this is done the abstract training result can be downloaded and trained further on a specific dataset with a custom number of classes.

Distillation is process of mapping a big neural network on a smaller neural network with similar results but higher performance. Any further information can be taken out from following paper:

<https://arxiv.org/abs/1503.02531> .

Hugging Face (<https://huggingface.co/>) is a famous AI community that provides several pretrained models and a python framework to load them into TensorFlow as well as PyTorch.

4.3.2. Implementation

For our project we use the “distilbert-base-cased” from Hugging-Face, which we downloaded completely to load it from local files.

We could not use the AbstractKerasMLClassifier class for this, because of the different structure of the ML load, a custom specific tokenizer with non-numpy datasets and the disability to save the model as a JSON stream which was required for the hash code generation.

Another important change is that we had to limit the unit tests to just test the predictions of the model and skip all the determinism tests. The reason for that is that it is impossible to set a pretrained model to train deterministic when it is already pretrained. TensorFlow simply does not react on the determinism setting.

At the end we came on a max text length of 500 and an epoch value of 1.

4.3.3. History

In the first implementation we tried to use the TensorFlow hub pretrained models (https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1) which simply don't work on our machines. We took about 5 days to figure out why the tokenizer is not working, but there is not usable description of this issue, so we had to search for another solution.

To avoid any issues, we tried to orientate on a working Kaggle solution (<https://www.kaggle.com/tanulsingh077/deep-learning-for-nlp-zero-to-transformers-bert>) on the next step which took about 4 days to get a working implementation based on AbstractKerasMLClassifier practically we provided a ready training implementation, but in the end we get an issue from the unit tests at the pull request level. The reason was simply that the custom tokenizer implementation does not work at all on a CPU, and there for the whole solution was useless for our reason.

On the final step we used the basic ML lib from the Kaggle solution which leaded us to the Hugging-Face community <https://huggingface.co/distilbert-base-cased>. We realized that there is a ready tokenizer specially for the models provided with the files which leaded us to the final solution. We had to kick out the AbstractKerasMLClassifier for that because the validation splitting, and preprocessing was different from the sequential Keras models. We had also to cut out most of the unit tests like determinism or tokenizer testing because it did not make sense for this pretrained model.

We also tried to train it on bert-base-cased but due to requirement of more than double of the performance comparing to distilbert-base-cased with similar results, we decided to kick the plain BERT model out for this project.

After training it successfully on titles we switched to full text where we discovered the practical max length border of text length of 500 for our development hardware. Like you can read in this paper (<https://proceedings.neurips.cc/paper/2020/file/96671501524948bc3937b4b30d0e57b9-Paper.pdf>) BERT has a quadratic growth for required memory related to the text length which makes it impossible to fix for us.

4.3.4. Problems

The first problems came with the implementation references which can be seen in History.

Another problem was the incapability of TensorFlow to switch to deterministic mode for pretrained models. The model has always another training path on every run and can have varying results (from 80 percent to 90 percent accuracy). To figure out the

The training setup was also very difficult because we had to find an optimizer that worked. By using the sparse_categorical_crossentropy setting via string like in the other models the model seemed not to have access to the logits like on the other models that we used. The result was that it trained backwards (accuracy went down on more training). After using the class SparseCategoricalCrossEntropy with setting logits=true it went much better and trained forward (At least on the smaller title columns).

After setting the full text classifiers, we discovered that the training goes up rapidly on the first epoch but again backward on the following epochs, which lead us to the final settings.

We tested out the optimum settings by reducing the batch size with increasing the word length, but we came up just to 1500 or word length which is barely a sparse of the capacity of the other models (10000). The issue came not only from the GPU training Hardware but also from the unit testing server which has a memory of 16GB like the webserver. We therefor set the final text length to 500 to make the unit tests working

4.3.5. Result

At all we finished the model with 1 epoch and a max text length of 500. The model works but is only usable for smaller texts in the dataset.

4.4. Support Vector Classifier

4.4.1. Description

To provide a completely different non-neural-network solution for comparison we included a Support Vector Classifier into the project. Since we have only 2 classes to predict this solution should work well in this solution.

A support vector machine is a distinction separator between (basically) two classes which calculates the max margin between the “Lagrange Multipliers” of the dataset. Due to the feature of providing custom kernels in dual mode, this kind of machine learning algorithm is still very powerful disregarding its age which goes down to the 1970th.

For the transformation we used here the Lemmatized TFIDF Transformation like in the first solution.

4.4.2. Implementation

Thanks to the SKLearn Library, it was very easy to implement the model in very few lines. But since the explain models required a softmax value for prediction we had to build a workaround for the prediction result. Due to its incapability to work with batches the validation data was also useless. We mapped the classes [0] or [1] to the arrays [1,0] and [0,1] which basically worked. The problems came later with the runtime.

4.4.3. Problems

Due to its incapability to work with batches we had to train it first on the full dataset, which takes around 5 hours to complete which is followed by 1 hour testing. By using the smaller validation set for training we reached a training + testing time of 1 hour. We realized that for these extreme large dimensions of around 20000 we have a lot of support vectors, and the prediction time increases a lot. Running LIME or Anchors on this model became so extremely slow, that it did not even finish after 1 day. Another problem was the extreme file size of the saved model of around 2GB.

4.4.4. Result

Due to its complete performance incapability to run on any explain engine we kicked this model out in the final implementation.

4.5. Common Problems

4.5.1. Implementing ML for a Webapp

As mentioned already at the hardware choice section, one of our major issues in this project was to provide scalable and runnable ML Model's on a normal Cloud Webserver. Our solution for this issue was to develop, a specific architecture and specific unit tests for the ML Models. These unit tests were run by the CI/CD on every pull request to the main branch, by the free default hosted CI machine of

Azure-Devops to ensure that the ML Models were definitely running their predictions on a reduced hardware set.

4.5.2. Saving Hash code of a trained ML

We aimed to ensure that the models are trained like the current set up in the train*.py file. To ensure this we had to save a hash code based on the train data as well as the whole ML class attributes. Since a TensorFlow model is not pickable due to its session attributes we had to disapply the model into a json stream and saved weights, but also to ensure that all TensorFlow operations are run inside specific methods. The Spacy tokenized confronted us with similar issues by being not deterministic on every load which resulted in pickling the loaded spacy NLP into separate files. At all this solution worked to basic trained models.

4.5.3. Reloading model

The problem which was followed by the hash code's solution was to reload the model in a performant way. It did not take very long to load the model from the json file + weight and compile it (0.3 s) but it was enough to significantly stretch the runtime of the lime sampling. We choose to provide a saving attribute into the prediction to avoid reloading it on prediction time. It worked because hash code generation is not required any more after the training and testing.

4.5.4. TensorFlow/Keras Framework

The whole framework seems to be a code merge from TF1 which is not barely object oriented. Several settings (Like logging levels or determinism) are only possible to set up through global variables without the ability to effectively change the value in runtime. This probably is not issue in Jupyter Notebooks but for a running Object-Oriented code like a website a big obstacle. If this is not already enough, the developers now convert several settings into the framework version after version, which results into that some settings are not working at all.

The serializing incompatibility due to session, and the missing feature to delete the session forced us into a workaround solution in the classes to save the pretrained model with tokenizer into a black box solution.

Anther issue due to the version switch and merge with Keras is that several libraries are provided multiple times for different reasons in different namespaces with being incompatible to each other.

All these issues are mostly found when working with the framework which makes it a big obstacle for proper solution development.

4.5.5. Determinism

The reason why we need determinism in our solution, is to find optimal settings for the ML training which does not work at all when the model is trained in another way. The core problem of this was that NVIDIAS CUDA Library CUDNN did only supported their major reduce-sum function in a non-deterministic way, which is practically a Race-Condition. The provided a deterministic equivalent now but this is not always used by all the libraries.

Many libraries in python are also not deterministic. Some of them like TensorFlow require several settings at multiple layers to be deterministic, but don't work for everything. For the pretrained hugging face model which use a different layer architecture of TensorFlow this setting does not work which always result in a different ML Training.

Since also other libraries like Spacy seem to have a determinism issue the idea of having deterministic model trainings becomes futuristic by now. Probably it will get fixed the next years by the major frameworks, but until then it is very hard to implement.

4.6. Conclusion

We have provided workarounds for all the issues, except of the hugging face model determinism, which cost us a lot of time and overran the expected project time of 13 CP per person. Most of the time in this project was spent to make the ML Models run, which were just the basement for the explain engines but required to work in a normal and scalable behavior. As a suggestion for a similar future project, we would use rather PyTorch that TensorFlow because of its Object-Oriented design, it's clear settings and ability to serialize the objects. We also hope that all the other libraries will become more deterministic in future and not only target for a Jupyter like line-down implementation but for real solutions.

But after all we are also very proud to be confronted with that because it gives us more confidence in estimating solution development with ML and to design better architectures.

In the following chapters we will provide a technical manual on how the architecture works and how to install and run the solution.

5. Preparation

5.1. Prerequisites

The project was developed on Windows Subsystem Linux with Ubuntu 20.04 as Distribution. The following are the tools that must be installed using apt install command.

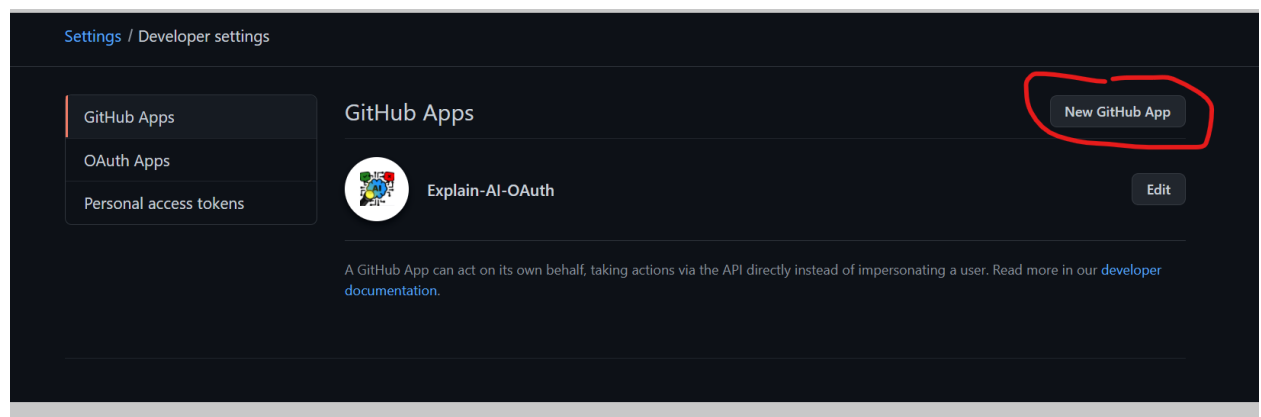
- Python 3.8 with PIP and VENV
- Node.JS 10.19
- NPM (Node Package Manager) 6.14.4
- Cuda 11.4 (optional but ML Training is very slow without)

5.2. Tools

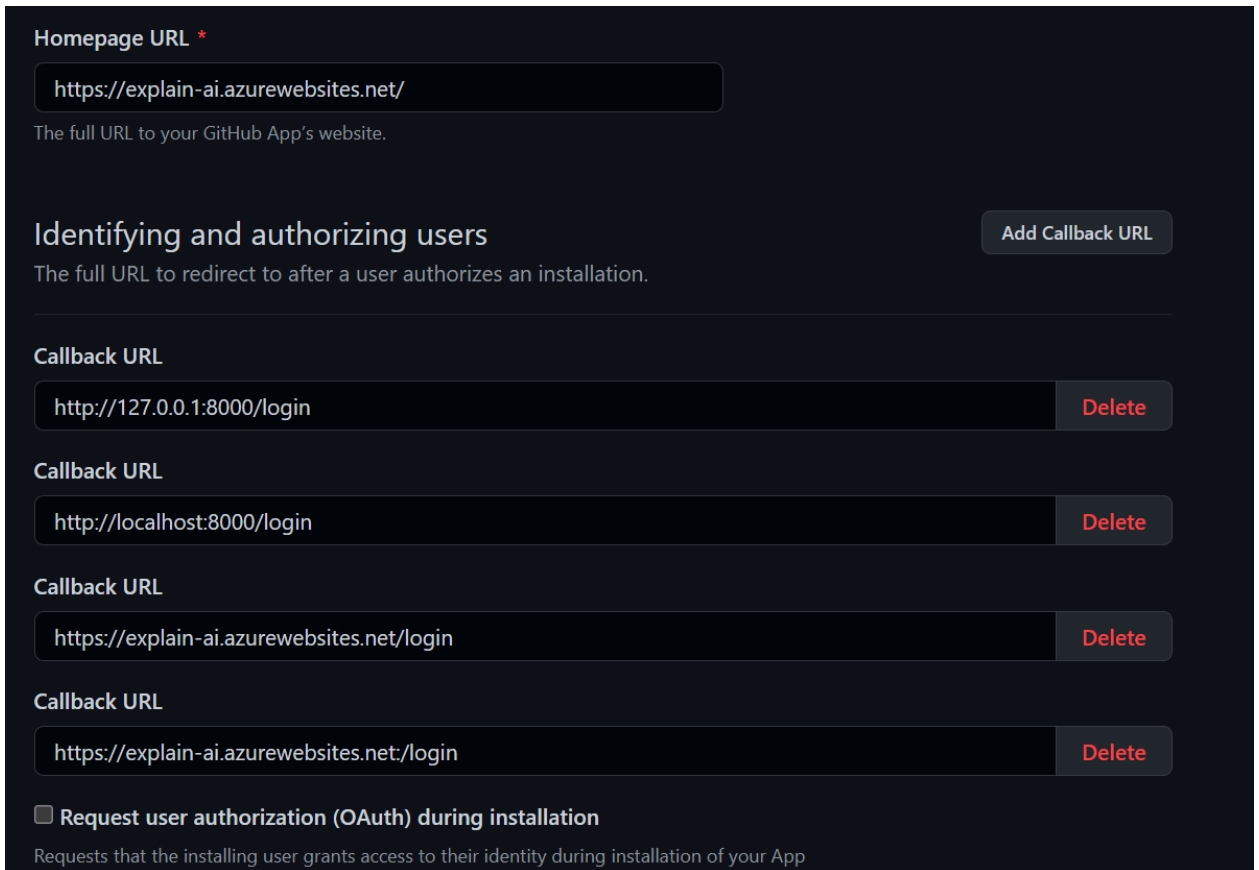
Using and IDE is not required but for fluent coding experience and debugging we recommend using Visual Studio Code with Python Extension.

6. GitHub Authentication Codes

The authentication is fixed into the application and uses GitHub OAuth to authenticate. To use it you must get a client id and a secret from your GitHub account. To do this logon to your GitHub account go to Settings/Developer settings -> GitHub Apps then Click on New GitHub App.



You must provide a homepage URL of your deployment and some possible callback URLs. See in the following pictures our URL settings that work.



Homepage URL *

`https://explain-ai.azurewebsites.net/`

The full URL to your GitHub App's website.

Identifying and authorizing users Add Callback URL

The full URL to redirect to after a user authorizes an installation.

Callback URL

`http://127.0.0.1:8000/login` Delete

Callback URL

`http://localhost:8000/login` Delete

Callback URL

`https://explain-ai.azurewebsites.net/login` Delete

Callback URL

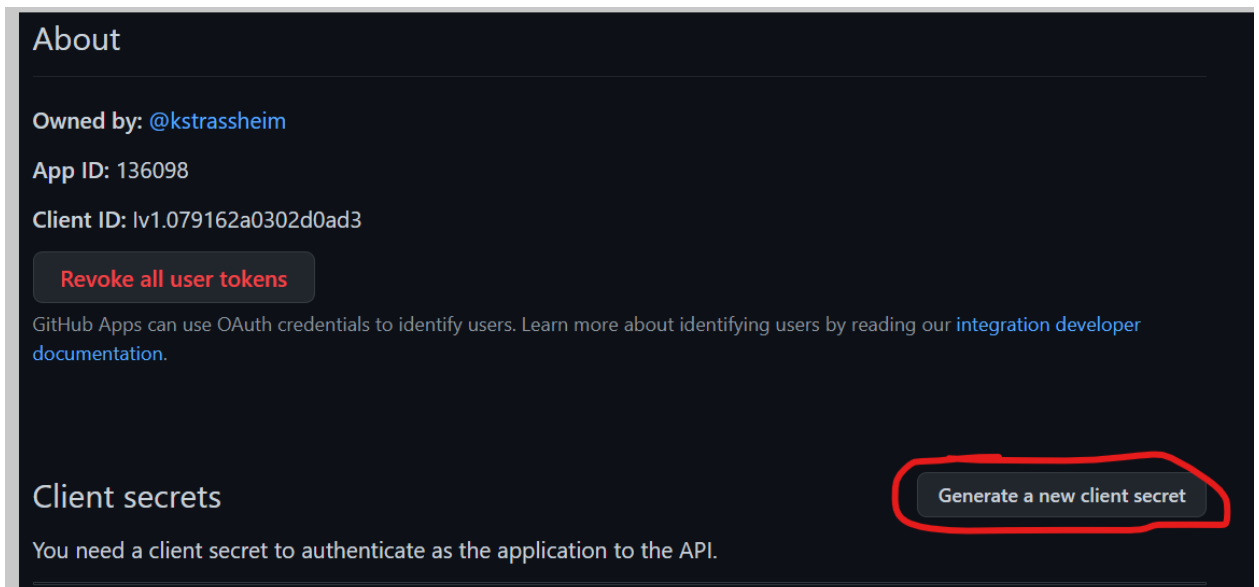
`https://explain-ai.azurewebsites.net/login` Delete

☐ **Request user authorization (OAuth) during installation**

Requests that the installing user grants access to their identity during installation of your App

You don't have to request access to any GitHub resources. It's just about the login here.

Once you finished you will see the client id. To generate the secret, you must click on the button for it



About

Owned by: [@kstrassheim](#)

App ID: 136098

Client ID: lv1.079162a0302d0ad3

Revoke all user tokens

GitHub Apps can use OAuth credentials to identify users. Learn more about identifying users by reading our [integration developer documentation](#).

Client secrets Generate a new client secret


You need a client secret to authenticate as the application to the API.

The secret will appear. Make sure to note it otherwise you do not have any chance to receive it back but must generate a new one.


Client secrets

Generate a new client secret

Make sure to copy your new client secret now. You won't be able to see it again.



Client secret

✓ 869dccab2189b942350bd39f068a7fada8b8173d 

Added now by kstrassheim

Never used

You cannot delete the only client secret. Generate a new client secret first.

Delete

You must enter the ClientID as WEBAPP_AUTH_CLIENTID and the Secret as WEBAPP_AUTH_SECRET into your environmental settings. On the

7. Environment Settings

You can provide a „env“ File to your project folder which will be ignored by git that contains all required settings to launch the program.

This is only for the local development. For production, please check your provider/cloud settings how to set them.

#Log Level Settings

FORCE_CPU=False

TF_LOG_LEVEL=3

VERBOSE=True

RETURN_STD_OUT=False

Unit Testing

UNIT_TEST_TF_LOG_LEVEL=3

UNIT_TEST_FORCE_CPU=False

UNIT_TEST_DATA_FILTER_SCALE=0.01

UNIT_TEST_VERBOSE=False

Preprocessing Settings

PREPROCESSING_TARGET_FILE_PATH=datasets/data.csv

PREPROCESSING_RAW_SOURCE_TRAIN_PATH=datasets/raw/train.csv

PREPROCESSING_RAW_SOURCE_TEST_LABEL_PATH=datasets/raw/submit.csv

PREPROCESSING_RAW_SOURCE_ID_FIELD_NAME=id

```
PREPROCESSING_LEMMATIZE_FIELD_NAMES=title,text
```

```
PREPROCESSING_LEMMATIZE_SUFFIX_NAME=_lemma
```

```
WEBAPP_DEBUG=True
```

```
WEBAPP_FRONTEND_DEBUG=True
```

```
WEBAPP_ALLOWED_HOSTS=*
```

```
WEBAPP_SECRET_KEY='SOME HASH KEY'
```

```
WEBAPP_AUTH_CLIENTID=SOME CLIENT ID'
```

```
WEBAPP_AUTH_SECRET='SOME SECRET'
```

```
WEBAPP_AUTH_AUTHORIZED_USERS='kstrassheim,astrassheim,markusbayer109'
```

```
#OPTIONAL IF Set then local file storage will be deactivated
```

```
WEBAPP_AZURE_STORAGE_CONNECTION_STRING='CONN STRING FROM AZURE  
SERVICE'
```

```
WEBAPP_AZURE_STORAGE_CONTAINER_NAME='development'
```

8. Compiling and Launch (Production)

To compile the project, you must run the following steps. We assume that the project was just checked out via git and the user navigated into the project folder

1. Create Python Virtual Environment
 - a. Run *python3 -m venv ./venv*
 - b. Run *source ./venv/bin/activate* to launch the virtual environment
2. Install pip packages
 - a. Run *pip3 install -r requirements.txt*
3. Install NPM Packages
 - a. Run *npm i*
4. Build NPM Packages (Production)
 - a. Run *npm run-script build*
5. Collect Static Files for Django Web App
 - a. Run *python3 webapp.py collectstatic --noinput*
6. Start gunicorn server for hosting the web app
 - a. Run *gunicorn -w 1 -k uvicorn.workers.UvicornWorker --bind=0.0.0.0 --timeout 600 --env DJANGO_SETTINGS_MODULE=webapp_settings webapp_asgi*

9. Debugging

Do steps 1 -3 of Chapter 3 to install the app and the packages. Set Environment Settings WEBAPP_DEBUG and WEBAPP_FRONTEND_DEBUG to True.

1. Start webpack dev server for frontend
 - a. Run *npm start*
 2. Start Django
 - a. Run *python webapp.py runserver*
-

If you use VSCode you can put the following launch.json into .vscode/launch.json

#VSCode debug launch.json

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [

    {
      "name": "Launch Chrome",
      "request": "launch",
      "type": "pwa-chrome",
      "url": "http://127.0.0.1:8000",
      "webRoot": "${workspaceFolder}/webapp"
    },
    {
      "name": "Python: Django",
      "type": "python",
      "request": "launch",
      "program": "${workspaceFolder}/webapp.py",
      "args": [
        "runserver",
        "127.0.0.1:8000"
      ],
      "django": true
    }
  ]
}
```

The first setting will launch the backend and enable you to use breakpoints in your python backend code.

The second setting will launch a chrome browser in developer mode and attach it to you project, so you can use breakpoints in your JS code (You still must manually launch npm start before doing this).

10. Data and Database Architecture

10.1. JSON Item

Most important to mention here is that we use the same JSON object on the frontend and on the backend and in the file database. The object is called „Item“. It is generated in the frontend passed to the backend which will add the explain information and other metadata, and then saved to the database via API. In following is an Example JSON item

An example can be found in the “item.json” file. To support different LIME extensions, the exact file format can differentiate between each LIME version.

10.2. File DB

We use a very simplified database in our project with used file append in JSON Lines format. Through the file append we don ‘t have to write the whole file again and again on saving one item but just to append it to the top. This means that we lose a bit of performance in the GETALL operation because we filter the double items and deleted items out there via a python for loop, but the performance is far enough to scale for this project.

We decided for this solution, because it is so simple and does not require any additional software or expensive cloud databases, but still performs well and is completely dynamic during development. If you need more performance, you can still adapt the items API to MongoDB which also based on JSON documents and very simple.

10.3. Azure File Storage

By setting the environment setting WEBAPP_AZURE_STORAGE_CONNECTION_STRING you will automatically enable the database stored not in a local folder, but in an Azure Blob Storage. The setting WEBAPP_AZURE_STORAGE_CONTAINER_NAME defined which container inside the blob it will use. We use this setting to distinct between production and development.

11. Code Architecture

11.1. Explain Engines

Important to notice here is that, because of simplicity of explain engines, we do not have abstractions for them. We run the explain engines directly in api/explain.py (see section backend)

11.2. Machine Learning

In this part we have generated a framework which allows us to save a whole trained model with its settings and a specific tokenized into one single (Pickle) file. The purpose on this is to check the trained file via unit tests and provide a hash code which can prove that the file is generated from these specific settings and therefor up to date.

11.2.1. Abstract

In the folder classes Abstract you will find the major files to the ML of this project.

- `Abstract_ml.py` is a basic abstraction for all ml models in this project. It combines the preprocessing, the tokenizer, ml training and testing into one single class, but also provides data splitting, time measurements and logging for a better training experience. There are several methods which are defined by `assert False` which make this class abstract. By deriving from it you must define your specific ML training, but not all the operations around which makes it easy to create several models.
Very important to notice here is that by deriving from this class your ML will be potentially compatible to the predefined unit tests in `Abstract_ml_testing.py` but also to the training class wrapper `Abstract_ml_testing.py`
- `Abstract_keras_ml.py` is a second layer abstraction which enables to easily, build a sequential Keras model that is in any case compatible with the Training and Unit Testing by simply defining the layers and the optimizer in the compile step. The Keras model is also set to train deterministic here.
- `Abstract_ml_training.py` is the wrapper that enables the model to be trained and saved into a single file. It also provides the settings for the training and generated the training data from the actual dataset.
- `Abstract_ml_testing.py` is the advanced set of unit tests for a deterministic trainer with a hash code. It will check if the trained model has alternating results and it matches the saved trained model file by applying a short test with a subset of the training data.
- `Abstract_ml_testing_pretrained.py` is a basic set of unit tests for a non-deterministic trainer. It will just check if the saved model provides satisfying results and is therefore trained well. We use this basically to test the Distilbert Classifier because we cannot change the pretrained settings on.

11.2.2. ML Models

In the folder `classes/ml` contain the ml classes which we can train and select in the Form to use them for explaining. We basically support 3 different MLs. Others are just experimental or from older versions.

- `Classes/abstract/ml/keras_nn_softmax_classifier.py` is a simple neural network that provides a multiple class output
- `Classes/abstract/ml/lstm_classifier.py` is a simple LSTM classifier
- `Classes/abstract/ml/hugging_face_pretrained_classifier.py` is a ml model that provides loading a pretrained model from the filesystem and refine it. You can load any model here unless it is from the official hugging face model (<https://huggingface.co/models>)

11.2.3. Training

All classes in the root folder matching the pattern `train_*.py` are the classes that will train a model and save it to a file for further use. Every train file combines the dataset, preprocessing, tokenizer, hyperparameters into a single processable class. Changing any of those setting would result in a different hash code. On the bottom of any of these classes is a `__main__` method which is the entry.

If you want to retrain a class or train a new created one, then just run `python3 train_[name].py` and it will perform the ml training.

11.2.4. Data Preprocessing

In an earlier stage we have processed the data preprocessing tokenizer in a separate step which results in the `datasets/data.csv`. Since this file is cleaner summarized and allows us to provide a custom data split, we use this `data.csv`. Therefor we also keep the files `data_preprocessing.py` and its unit tests. The tokenized fields in the generated file are not used any more.

11.2.5. LIME Extensions

This work implements the results from our work on LIME extensions for better performance on large text sequences. The LIME variant MultiStepLIME and IterativeLIME are implemented in a simplified way such that the methods can be used without the complexity in which they can be normally used. For example, in MultiStepLIME arbitrary large pipelines could be created. As the dataset we used to train our machine learning models can only be tokenized to words or sentences, only a 2 step approach is implemented.

11.2.6. Fixed Files

- Datasets/raw contains the source dataset from Kaggle
- Datasets/data.csv is the summarized and preprocessed file which we use in this project.
- Models contains the pickle files of the trained model results. We load this files in the frontend to provide the
- NLP contains the spacy models which we had to save outside because they don't provide a deterministic hash otherwise.
- Pretrained contains the pretrained models from the hugging face classifier. We had to download and save them to support a deployment as a website. Currently the is only the distilbert-base-cased pretrained model and tokenizer inside.

11.3. Backend (Django)

The Django application is using ASGI Mode and providing WebSocket's via Channels.

11.3.1. Root Files

In the root you find several files with webapp_*.py pattern. They are used to configure Django and don't have to be modified any more since you have the environment settings for that.

If you want to know further information on that, please check the official Django documentation.

<https://docs.djangoproject.com/en/3.2/>

The file webapp.py is the main file that launches the web application.

11.3.2. API Folder

This folder is the actual backend that is called. Every file here sets up a response on a specific URL.

- Auth.py is used to handle the authentication requests and to check the GitHub tokens on every request.
- Item.py handles the database requests for the CRUD operations but also to display a list of the provided requests onto the frontend.
- Explain.py is the main file on this project because it runs the ai predictions and explainer engines. After the run the json item is enriched with the result property which contains all variables about the ML and Explain predictions and runtimes.

11.3.3. API Sockets Folder

The API-Sockets folder contains the item WebSocket notification container. This is used to notify other clients about changed added or deleted items.

11.4. Frontend (React)

The main file here is package.json in the root folder which defines the npm settings to run the frontend, but also all the additional JavaScript packages and their versions. For a proper documentation of the React Framework please visit the official documentation (<https://reactjs.org/docs/getting-started.html>).

11.4.1. Folder src

The src folder contains the frontend files which are compiled via webpack over the create-react-app package. We use create-react-app because it already provides an advanced webpack configuration with all features needed. React Components are camel cased while other JS Libraries are lowercased. Each React files has a same named CSS file which contains its specific formatted style.

- App.js in the center file for the react application which also defines the Routes and combines them to Pages
- NavMenu.js is the Navigation Menu
- Login.js handles the login process
- api.js is the abstracted backend access
- components/* are commonly used controls
- pages/* are the pages
 - pages/EditItem.js contains the main form to select the explain settings and push them to the backend API.

11.4.2. Folder build and public

The public folder contains the static frontend files like images and the index.html file.

The build folder contains the compiled webpack result of all the JavaScript files in the project when running npm run-script build.

12. Testing

The main purpose for the testing implementation is to check if the saved models are trained properly, trained with the current set up hyperparameters in the train file and with the same data.

Every training file in the root folder has a testing file which has the same file name, but with a suffix. You can simply run the file or use the python unit test engine to select multiple files to run tests for.

12.1. Major Challenge

The main challenge to achieve a proper testing is that a model needs a lot of time to train which is not easy to reproduce on a device without a GPU like an automatic testing VM.

To avoid this long run training, we provide the following 2 solutions.

12.1.1. Smaller set of data to test if training runs.

The testing class calls "self.custom_init" in the constructor and provides the actual training class plus a unit_test_data_filter_scale parameter which is about 0.01. The parameter defines the amount of data to use for the subset of training to check if the training performs and provides reasonable data. The test also checks if the models run deterministic by running different instances with the same settings and comparing the results.

12.1.2. Hash code

The more difficult part was to check if a new training would produce the same saved result as the currently saved data. To provide this we generate a hash code just before the training starts. This hash code is calculated from the input data and all the saved hyperparameters of the model. On the test run we therefore do not run the train but just `get_hash_code` method with the original amount of training data and test if they result in the same hash code. Since the model is already tested for determinism in the previous tests the model will result in the same predictions if we would retrain it.

12.2. Other Challenges

12.2.1. TensorFlow Session

It can happen that a TensorFlow session is not cleared. The next test run with another model will be affected by this. To avoid it we clear the session before the hash generation to ensure this session is new. It works for all sequential Keras models. But the pretrained ones do not listen to this setting. TensorFlow has documented many of such issues and is currently changing several triggers which are still there from version 1.0, but unless it does not provide working triggers for every setting this issue cannot be fixed in common for all TensorFlow models.

For this project try to run exotic TensorFlow Combinations in another run than the sequential models. This is the only fix for this issue.

12.2.2. Pretrained Bert Models determinism

On the pretrained hugging face models we have in general that they are not deterministic. Since this setting seems not to be reversible and the common TensorFlow Determinism Trigger does not work here we have to accept this for our Bert Model. The solution for this is to provide a shorter testing where there is no check for hash code or for determinism. This test is also run in a separate test run after the sequential model tests and simply checks for alternating prediction results (That is not 0 or 1 for any input)

12.3. Run Tests

To run the tests, you can enter the following

- `python3 -m unittest *_test.py` for the sequential model tests
- `python3 -m unittest *_test-pretrained.py` for the pretrained shorter model tests

13. Application

By using constantly Bootstrap 5 we designed the app to be responsive in all sizes. This means it can also run on an iPhone Mini as well as on a 4k screen.

13.1. Home Screen – Item List

The screenshot shows the Home Screen of the application. On the left is a list of 10 items, each with a text description, a model name, and 'Show' and 'Delete' buttons. The items are grouped by color: pink, light blue, and light purple. On the right is a configuration panel for the 'Machine Learning Model'.

Item Description	Model	Show	Delete
Email US President Barack Obama and former secreta - BERT - MultiStepLIME	BERT	✓	✗
Email US President Barack Obama and former secreta - BERT - IterativeLIME	BERT	✓	✗
Email US President Barack Obama and former secreta - BERT - LIME	BERT	✓	✗
The overwhelming consensus of the punditry across - TfIdf - MultiStepLIME	BERT	✓	✗
His name is King Leopold II of Belgium and the liv - LSTM - IterativeLIME	BERT	✓	✗
NETHERLANDS has a new Muslim political party calle - BERT - IterativeLIME	BERT	✓	✗
President Trump on Tuesday nominated Judge Neil M. - BERT - MultiStepLIME	BERT	✓	✗
Satanic Temple YES, Ten Commandments NO? October 2 - LSTM - MultiStepLIME	BERT	✓	✗
Anthem, the nation's second largest health insura - BERT - LIME	BERT	✓	✗
Posted by Alex Cooper Nov 25, 2016 Breaking Ne - LSTM - LIME	BERT	✓	✗
WASHINGTON — Just days before Election Day, and - BERT - LIME	BERT	✓	✗

Machine Learning Model

BERT

Explaining Method

MultiStepLIME

MultiStepLIME

Combination Setting

Combined

Sentence Level

Minimum number of samples

10

Maximum number of samples

1000

Distance Metric

MAE

Error Cutoff

0,001

Word Level

Minimum number of samples

10

Maximum number of samples

1000

Distance Metric

MAE

Error Cutoff

0,001

When you have proceeded the GitHub logon you will be automatically redirected on the start page which provides a list of already run explanations sorted descending by their modified date.

When you click on one item it will open the detail view as well as select it for further procedure.

13.2. Home Tasks

The screenshot shows the Home Tasks section. It features a single item card with a text description, a model name, and 'Show' and 'Delete' buttons. Above the card are three icons: a refresh icon (green), a plus icon (cyan), and a delete icon (red).

Refresh (green) | New Item (cyan) | Delete (red)

Email US President Barack Obama and former secreta - BERT - MultiStepLIME

Show ✓ Delete ✗

On the home screen you can run following operations.

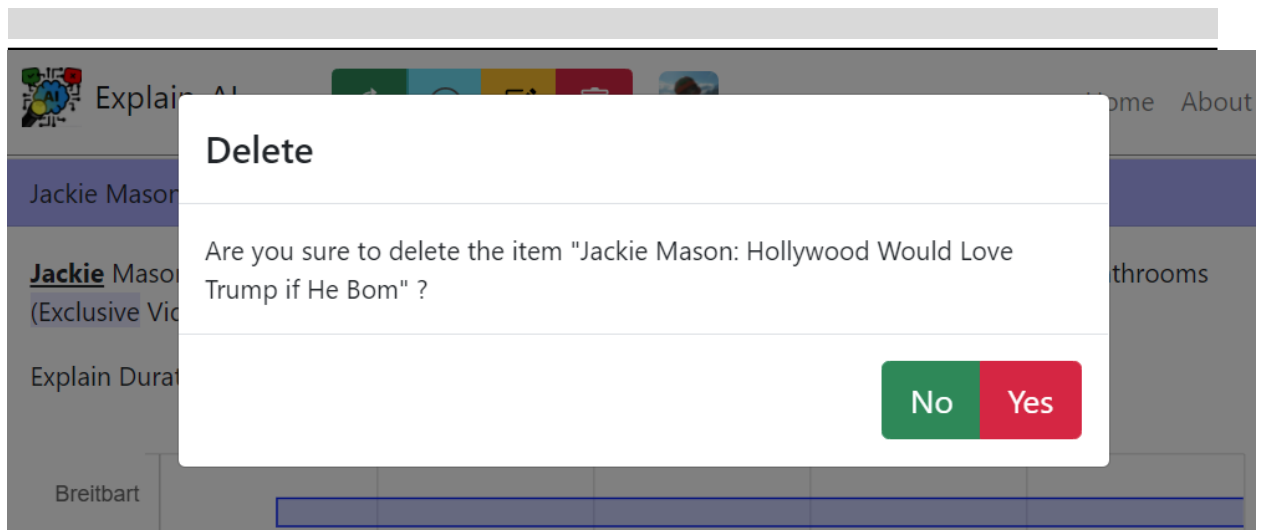
- Refresh – green – will refresh the list
- New Item – cyan – will open edit form with a new entry

Every item in the list further provides the following operations:

- Show – cyan – will open the details for the explanations in which the item can also be edited
- Delete – red – will open the delete prompt to delete the item

13.2.1. Delete Prompt

The delete Prompt looks as following



You will have to select yes to finally delete the item from database.

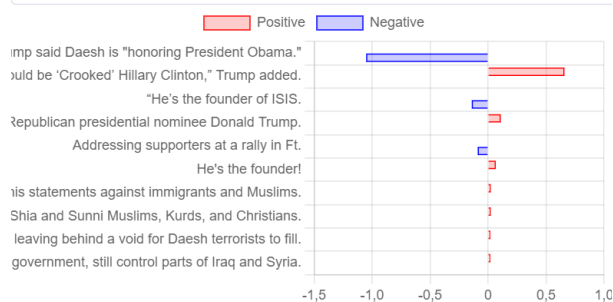
13.3. Show/Edit Form – View Mode

When the “Show” operation is used on an item, the complete text will be displayed which also highlights the corresponding explanations. When scrolling down, further details can be seen such as method specific operations, such as changing the level in MultiStepLIME can be seen. The edit button allows to change explanation parameters.

Explanation

Details

Sentence Level



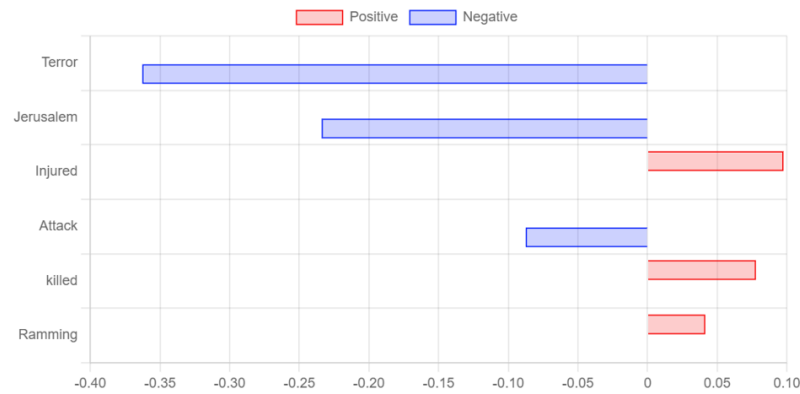
Text

Email US President Barack Obama and former secretary of state Hillary Clinton “founded” the Daesh (ISIL) terrorist group in the Middle East region, according to Republican presidential nominee Donald Trump. Addressing supporters at a rally in Ft. Lauderdale, Florida, on Wednesday, Trump said Daesh is “honoring President Obama.” “He’s the founder of ISIS. He’s the founder! He founded ISIS,” the real estate tycoon said, using an alternative acronym for the terrorist group. “I would say the co-founder would be ‘Crooked’ Hillary Clinton,” Trump added. He went on to criticize Obama’s decision to withdraw American military forces from Iraq and leaving behind a void for Daesh terrorists to fill. “We should never have gotten out the way we got out,” he said. “We unleashed terrible fury all over the Middle East.” “Instead of allowing some small forces behind to maybe, just maybe, keep it under control, we pulled it out,” he continued. Daesh terrorists, many of whom were initially trained by the CIA in Jordan in 2012 to destabilize the Syrian government, still control parts of Iraq and Syria. They have been engaged in crimes against humanity in areas under their control. They have been carrying out horrific acts of violence such as public decapitations and crucifixions against all communities, including Shia and Sunni Muslims, Kurds, and Christians. In a statement last week, Mike Pence, Trump’s vice presidential candidate, also said that the policies of Obama and Clinton led to the rise of Daesh. He blamed Obama and Democratic presidential nominee Hillary Clinton for the “disastrous decisions” that led to the death of Captain Humayun Khan in Iraq. On July 28, Captain Khan’s father, Khizr Khan, addressed the Democratic National Convention, denouncing Trump as unpatriotic and selfish over his statements against immigrants and Muslims.

Edit Text

Four killed, 10 Injured in Jerusalem Truck-Ramming **Terror** Attack

Explain Duration: 00:01:31.471 GMT



13.3.1. Detail view

The detail view provides you all prediction results in all detail levels as well as the runtimes for the specific operations. Local Score and Samples are only available for LIME and IterativeLIME.

Explanation

Details
Model Prediction Label: 1
Model Prediction: -0.22723981738090515,0.34722867608070374
Duration ML: 1.4427852630615234
Duration Lime: 7.663795709609985
Duration Total: 9.106580972671509
Local Score: -
Samples: -

13.4. Edit Mode – Edit

In Edit mode the results will be deleted (not saved yet) and the view changes into a form where you can select the options for a new explain prediction. The same form is opened when a new item is created.

Text

Email US President Barack Obama and former secretary of state Hillary Clinton "founded" the Daesh (ISIL) terrorist group in the Middle East region, according to Republican presidential nominee Donald Trump. Addressing supporters at a rally in Ft. Lauderdale, Florida, on Wednesday, Trump said Daesh is "honoring President Obama." "He's the founder of ISIS. He's the founder! He founded ISIS," the real estate tycoon said, using an alternative acronym for the terrorist group. "I would say the co-founder would be 'Crooked' Hillary Clinton," Trump added. He went on to criticize Obama's decision to withdraw American military forces from Iraq and leaving behind a void for Daesh terrorists to fill. "We should never have gotten out the way we got out," he said. "We unleashed terrible fury all over the Middle East." "Instead of allowing some small forces behind to maybe, just maybe, keep it under control, we pulled it out," he continued. Daesh terrorists, many of whom were initially trained by the CIA in Jordan in 2012 to destabilize the Syrian government, still control parts of Iraq and Syria. They have been engaged in crimes against humanity in areas under their control. They have been carrying out horrific acts of violence such as public decapitations and crucifixions against all communities, including Shia and Sunni Muslims, Kurds, and Christians. In a statement last week, Mike Pence, Trump's vice presidential candidate, also said that the policies of Obama and Clinton led to the rise of Daesh. He blamed Obama and Democratic presidential nominee Hillary Clinton for the "disastrous decisions" that led to the death of Captain Humayun Khan in Iraq. On July 28, Captain Khan's father, Khizr Khan, addressed the Democratic National Convention, denouncing Trump as unpatriotic and selfish over his statements against immigrants and Muslims.

Explain Instance

Cancel

Machine Learning Model

BERT

Explaining Method

MultiStepLIME

MultiStepLIME

Combination Setting

Combined

Sentence Level

Minimum number of samples

10

Maximum number of samples

1000

Distance Metric

MAE

Error Cutoff

0,001

Word Level

Minimum number of samples

10

Maximum number of samples

1000

Distance Metric

MAE

Error Cutoff

0,001

The form provides the following options:

- Machine Learning Model
 - Chooses the prediction model which classifies the text. The following options are available: TfIdf, LSTM, BERT
- Explaining Method
 - Chooses the explanation method which explains the results and has a direct correlation with runtime. The following options are available: LIME, IterativeLIME, MultiStepLIME
- Model Specific settings
 - LIME
 - Number of features changes the number of features which are explained
 - Number of samples: number of samples which are created from the text to create the explanation
 - IterativeLIME
 - Number of features: same as LIME
 - Minimum number of samples: minimum number of samples which are created for the explanation
 - Maximum number of samples: maximum number of samples which are created for the explanation
 - Sample Step Size: Number of increments in samples for each step
 - Distance Metric: Metric to which the last explanation is compared with the new explanation with increased sample size
 - Error Cutoff: If the difference is smaller than the error cutoff, IterativeLIME will stop to create new samples

-
- MultiStepLIME
 - Combination Setting: setting in which defines how the results of deeper levels are combined. Combined will create one combined string containing each higher-level string determined important by feature size (fixed at 10 for simplification).
Individual will perform deeper level explanation individually for each feature.
 - Levels: Same as IterativeLIME but with two levels of parameters. The first level describes sentence-based tokenization, the second word level tokenization
-

14. Progressive Web App

The application is designed as an PWA. This means it contains a manifest.xml which defines the settings to be installed as an App, for example on a smartphone. When the server is hosted on Https then the App will behave like a native app so you will not see any URL line or anything that looks like the web browser.

14.1. Android

To install a PWA on an Android Device please follow the following instructions.

<https://support.google.com/chrome/answer/9658361?hl=de&co=GENIE.Platform%3DAndroid>

14.2. IOS

To install a PWA on a IOS device you can follow the following instructions.

<https://superpwa.com/doc/test-pwa-ios-devices/>
