Detecting PII (Personally Identifiable Information) in Student Essays

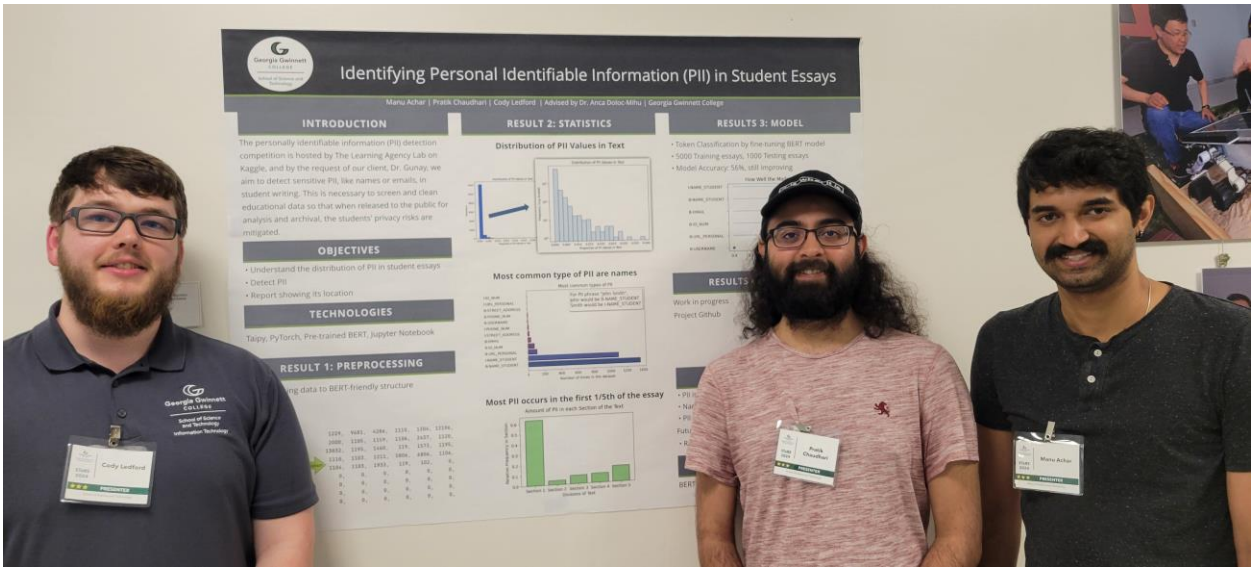
By Algorithm Allies

Cody Ledford, Pratik Chaudhari, Manu Achar

Description

Our team is the Algorithm Allies! We are working on a Kaggle project via their Challenges section. The Kaggle Competition we are participating in is the PII Data Detection hosted by The Learning Agency Lab. The objective of the project is to create an AI (Artificial Intelligence) model that detects personal identifiable information (PII) so it can be censored. This is important when releasing educational material to the public to protect the identity of students. The data is contained in JSONs of student essays that were tokenized using spaCy.

Team



(From Left to Right) Cody Ledford, Pratik Chaudhari, Manu Achar

Client Presentation

Presentaion

Team Plan

First Sprint

Completed

- Explore and clean dataset provided by Kaggle
- Find new datasets to potentially use to train after the initial training set
- Research frameworks and how we will build and implement the model

Second Sprint

Completed

- Visualizations
- More exploration of data
- Choosing a model to train
- Preprocessing the data to fit our model
- Build Taipy site to host model

Unfinished

- Add model to the Taipy site

Third Sprint

Completed

- Rebalance the dataset
- Change the tokenizer to handle labeled and unlabeled data
- Starte using the Hugging face TrainerArguments and Trainer in the model
- Add argmax (a method to find the highest probability) for prediction selection

- Build the submission report code for Kaggle
- Clean up visualizations
- Build the static site

Unfinished

- Find a host for the Taipy site
- Add model to Taipy site
- Improve model accuracy
- Try additional training with other datasets

Roles

Team Manager – Planning for sprints, Assigning issues, Jira Management, Framework Selection

Client Liaison – Communications with client, Handling client requirements, demos for client

Project Documenter – Document project progress. Prepare presentation

Technologies

Jupyter Notebooks, PyTorch, BERT pre-trained model, Taipy, JavaScript, HTML/CSS, Bootstrap, Docker, DockerHub, Google Colab, Kaggle, GitHub, GIT LFS

- BERT – A language model based on transformer architecture
- PyTorch – Machine Learning framework
- Taipy – A tool to create dynamic webpage using Python
- GIT LFS - Large file storage used by Git to store and transfer files larger than 100 MB

Timeline

Sprint 1

Sprint 2

Sprint 3

Issue	Week 9	Week 11	Week 12	Week 13	Week 14
Webpage Embeded Graphs	Cody				
Taipy Hosted on GGC Server			Cody		
Rebalance Data			Pratik		
Update README with Taipy			Cody		
Update README with Model				Manu	
Rework Encoder		Full Team			
CREATE/STARS Poster	Full Team				
Github/GitLFS Troubleshooting			Manu		
Analytics Webpage HTML/CSS				Cody	
Kaggle Submission				Pratik	Full Team
Statistics Notebook			Pratik	Full Team	
Analytics Webpage on GitHub Pages					Cody

Collections

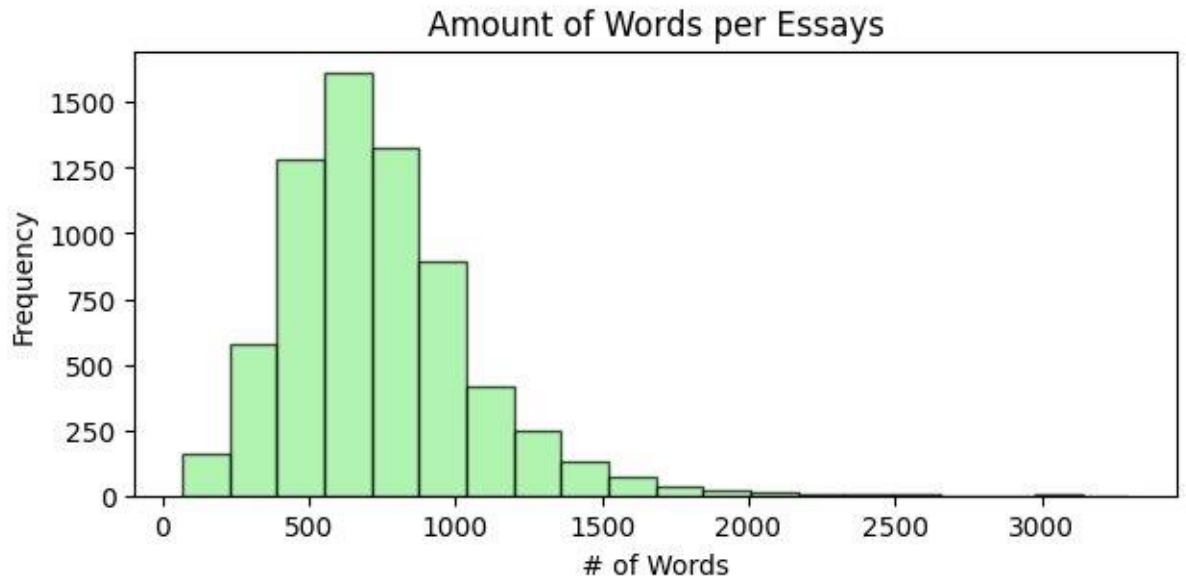
The dataset used for this project was from Kaggle. While the only dataset used has been the Official dataset from Kaggle, other datasets from student essay were collected.

Datasets:

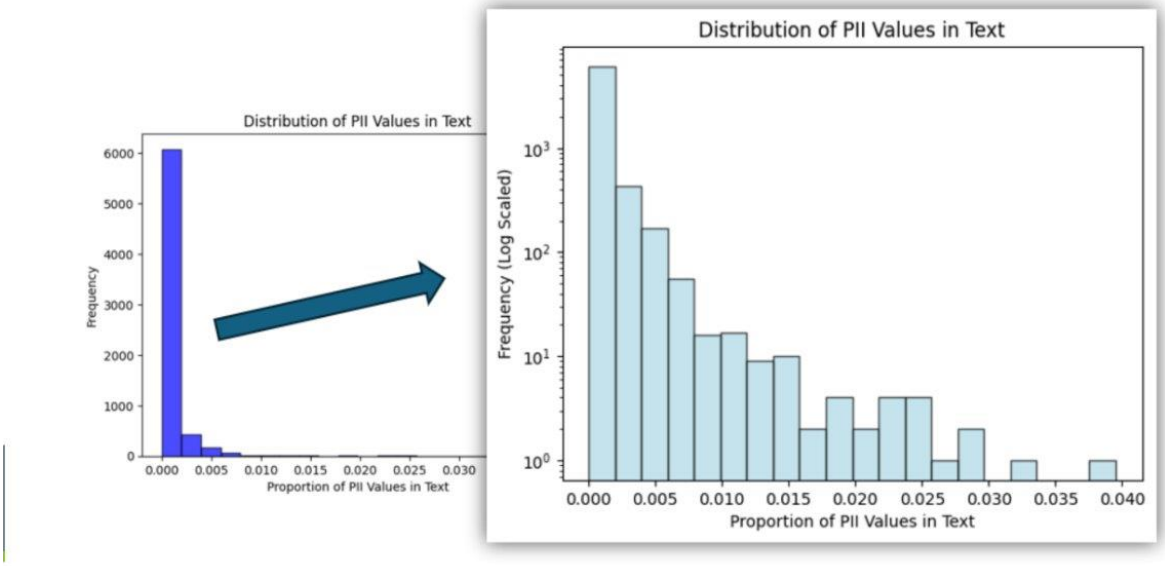
- Official Training
- Official Test
- External Datasets

Methods

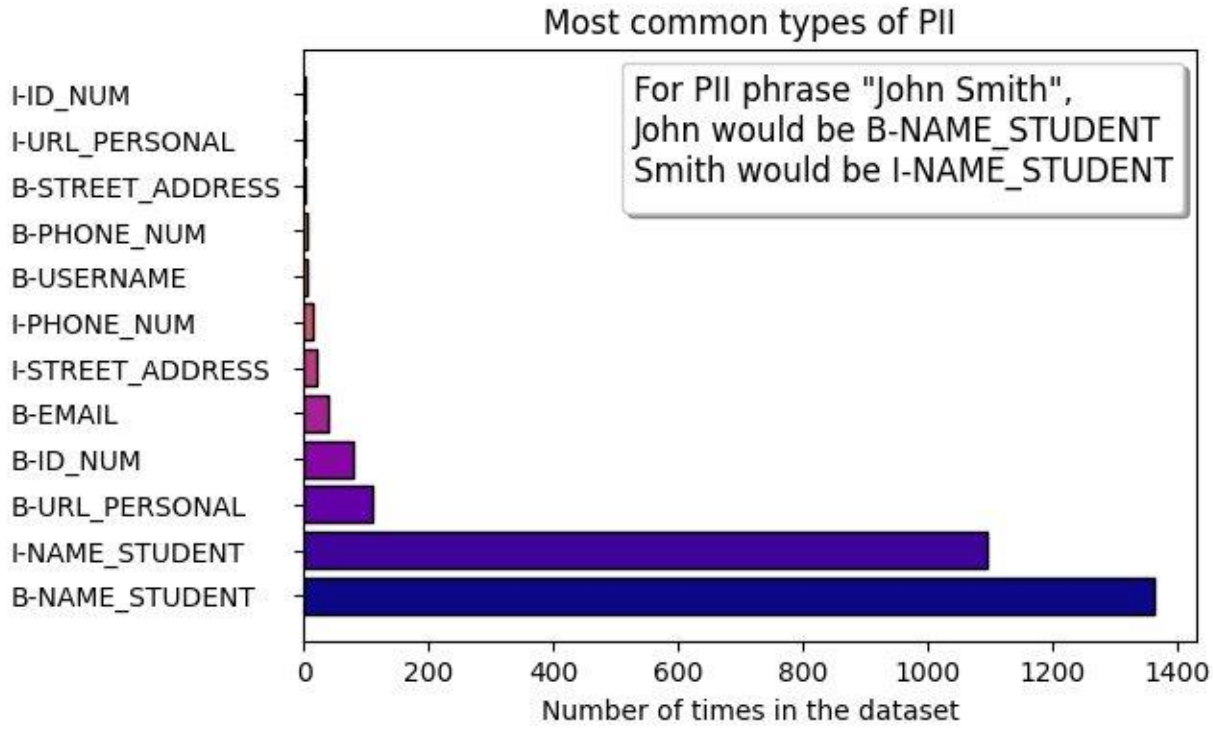
First a visualization for how many words is in each essay. We did this by graphing out the length of each essay.



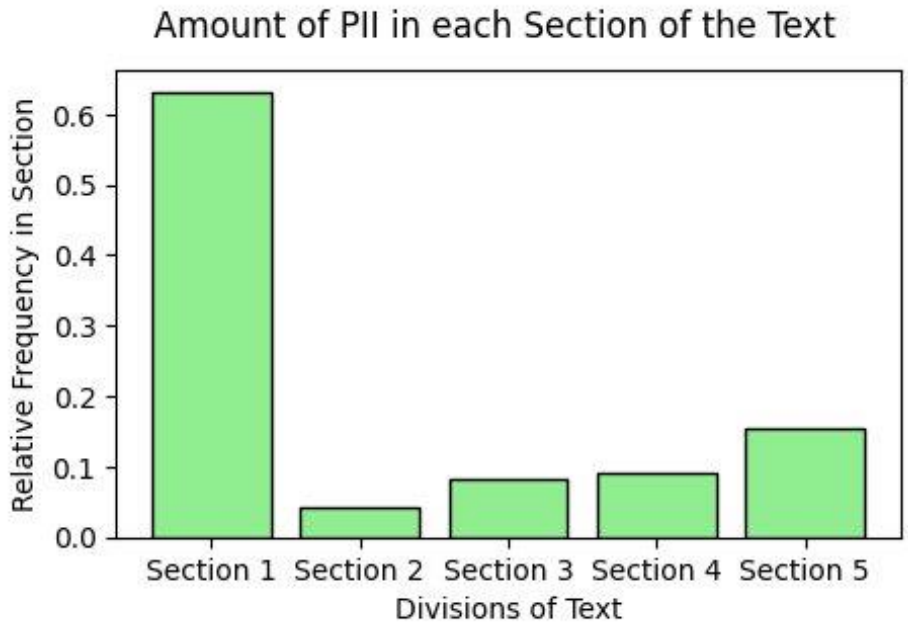
Next, we investigated as to what proportion of the essays were PII data. We did this by measure the proportion of PII in each essay



Next, we wanted to see how much of each type of PII was in the training dataset. We measured the labels and counted the number of each PII in the entire dataset



Next, we were looking to see the location of the PII in the essays. So, we split each essay into five parts and charted the locations of the PII



To preprocess the data for our model we needed to decrease the input size as our model only took up to 512, but there were many essays that were longer.

```python
def make_smaller_inputs(dataframe, type):
    """Splits the entire essays into MAX_LEN size blocks and remaps tokens and labels
    """
    df_out = pd.DataFrame(columns = ['tokens','labels','document','document_location'])
    idx_df = 0
    max_len = config['MAX_LEN']

    for _,line in dataframe.iterrows():
        location_counter = 0
        tokens = line.tokens
        if type == 'train':
            labels = line.labels
        document = line.document
        items = range(0,len(tokens),max_len)

        for i in items:
            df_out.at[idx_df,'tokens'] = tokens[i:i+max_len]
            if type == 'train':
                df_out.at[idx_df,'labels'] = labels[i:i+max_len]
            df_out.at[idx_df,'document'] = document
            df_out.at[idx_df,'document_location'] = location_counter
            location_counter += 1
            idx_df += 1

    return df_out
```

```python
df_model_input_train = make_smaller_inputs(df_usable_train,'train')
print(len(df_model_input_train.index))
df_model_input_train.head(2)
```

Next, we created the tokenizer

```python
def tokenize(line, tokenizer, type):

    tokens = line.tokens


    if config['ignore_subwords']:
        length = config['MAX_LEN']
    else:
        length = math.ceil(config['MAX_LEN'] * 1.2)


    encoding = tokenizer(tokens,
                         is_split_into_words= True,
                         return_offsets_mapping= True,
                         padding= 'max_length',
                         max_length= length)

    item = {key: torch.as_tensor(val) for key, val in encoding.items()}

    if type == 'train' or type == 'eval':
        word_labels = line.labels

        temp_list = [0 for _ in range(length - len(word_labels))]
        labels = [labels_to_ids[label] for label in word_labels] + temp_list

        encoded_labels = np.ones(len(encoding["offset_mapping"]), dtype=int) * -100
        i = 0

        if config['ignore_subwords']:
            #         Ignore subword labels
            for idx, mapping in enumerate(encoding["offset_mapping"]):
                if mapping[0] == 0 and mapping[1] != 0:
                    # overwrite label
                    encoded_labels[idx] = labels[i]
                    i += 1
        else:
            #         Extend subword labels
            for idx, mapping in enumerate(encoding["offset_mapping"]):
                if mapping[0] == 0:
                    encoded_labels[idx] = labels[i]
                    i += 1

        item['labels'] = torch.as_tensor(encoded_labels)

        return {**item}
```

```python
    if type == 'predict':
        document = line.document
        location = line.document_location

        encoded_labels = np.ones(len(encoding["offset_mapping"]), dtype=int) * -100
        labels = np.zeros(len(tokens), dtype=int)
        i = 0

        if config['ignore_subwords']:
            #         Ignore subword labels
            for idx, mapping in enumerate(encoding["offset_mapping"]):
                if mapping[0] == 0 and mapping[1] != 0:
                    # overwrite label
                    encoded_labels[idx] = labels[i]
                    i += 1
        else:
            #         Extend subword labels
            for idx, mapping in enumerate(encoding["offset_mapping"]):
                if mapping[0] == 0:
                    encoded_labels[idx] = labels[i]
                    i += 1
        item['labels'] = torch.as_tensor(encoded_labels)

        return {**item, 'document': document, 'location': location}
```

This tokenizer uses BERT's Word Piece algorithm to split the input into the most likely combinations of characters. The different sections of this custom function are used in the different steps of the process, training, evaluation, and prediction.

Next, we set up the training

```python
training_args = TrainingArguments(
    output_dir= config['model_path'],
    # overwrite_output_dir = True,
    do_train = True,
    # do_eval = False,
    # per_device_eval_batch_size=1,
    auto_find_batch_size=True,
    report_to="none",
    num_train_epochs = config['EPOCHS'],
    learning_rate = config['LEARNING_RATE'],
    save_strategy = 'no',
    disable_tqdm= False,
    no_cuda = False,
    metric_for_best_model="f1",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset = train_dataset,
    # eval_dataset = eval_dataset,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

```python
# Training
if training_args.do
    train_result = trainer.train()
    metrics = train_result.metrics
    trainer.save_model()  # Saves the tokenizer too for easy upload

    metrics["train_samples"] = len(train_dataset)

    trainer.log_metrics("train", metrics)
    # trainer.save_metrics("train", metrics)
    # trainer.save_state()
```

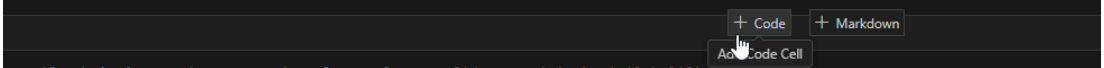
(variable) trainer: Any

The training arguments set hyperparameters used in the training loop and the number of EPOCHS, the learning rate for optimizer. This Class is from Huggingface. The Trainer class contains all the mathematical functions used to train the model.

```python
# Predict
if training_args.do_predict:
    # logger.info("*** Predict ***")

    predictions, labels, metrics = trainer.predict(predict_dataset, metric_key_prefix="predict")
    predictions = predictions.argmax(-1)

    # Remove ignored index (special tokens)
    true_predictions = [
        [ids_to_labels[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

#     trainer.log_metrics("predict", metrics)
#     trainer.save_metrics("predict", metrics)
```

Loading final predictions into and converting the numerical representations of the predictions into string representations

```python
results = []
for preds, location, doc in zip(
    true_predictions,
    predict_dataset["location"],
    predict_dataset["document"]
):
    results.extend([(doc, location * config['MAX_LEN'] + idx, label) for idx, label in enumerate(preds) if label != 'O'])
```

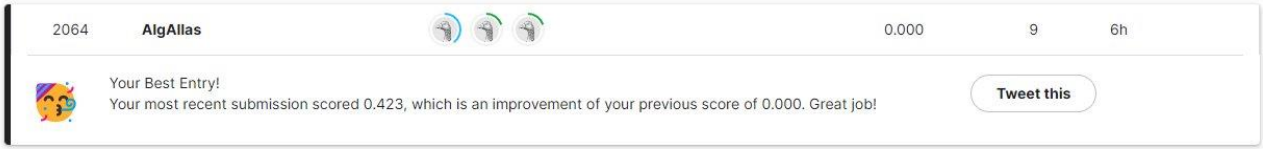



```python
df_submission = pd.DataFrame(results, columns = ['document','token','label'])
df_submission['row_id'] = df_submission.index
df_submission_final = df_submission[['row_id','document','token','label']]
df_submission_final
```

	row_id	document	token	label
0	0	7	9	B-NAME_STUDENT
1	1	7	10	I-NAME_STUDENT
2	2	7	482	B-NAME_STUDENT
3	3	7	483	I-NAME_STUDENT
4	4	7	741	B-NAME_STUDENT

Building the submission file from the true predictions and the location of each input sequence recorded during preprocessing. It excludes non-PII guesses.

We submitted to Kaggle with our model and submission



Summary of Results

Sprint 1

- Exploration of essay dataset provided by Kaggle
- New datasets for additional training on the model
- Taipy first page/hello world was created

Sprint 2

- Preprocessing
- Chose BERT pre-trained model as an initial model
- Initial model training
- Built Taipy site based on the wireframe. All pages were added with placeholders.

Sprint 3

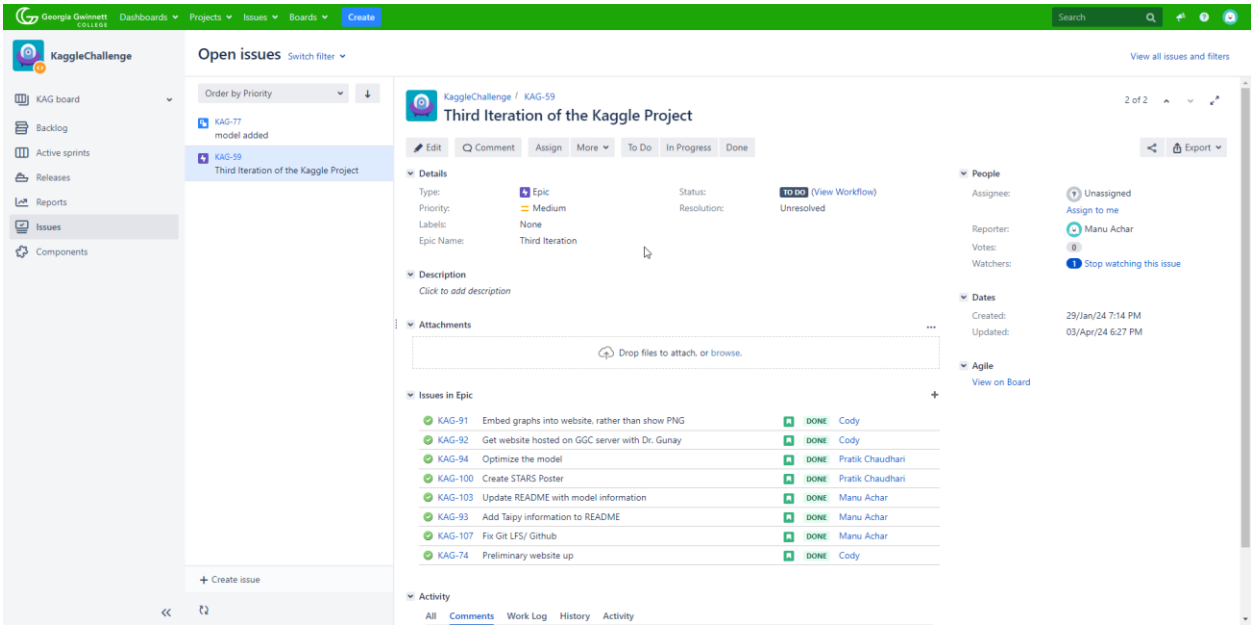
- Rebalanced the dataset
- Updated the tokenizer to handle labeled and unlabeled data
- Changed the training process by using Hugging face Trainer
- Built submission for Kaggle competition
- Built a static site (due to problems hosting Taipy)
- Setup Taipy to be ready to have the model added for full functionality

GitHub

All the project files are located on a public Github Repository. Our static website is located here.

Jira

Dashboard



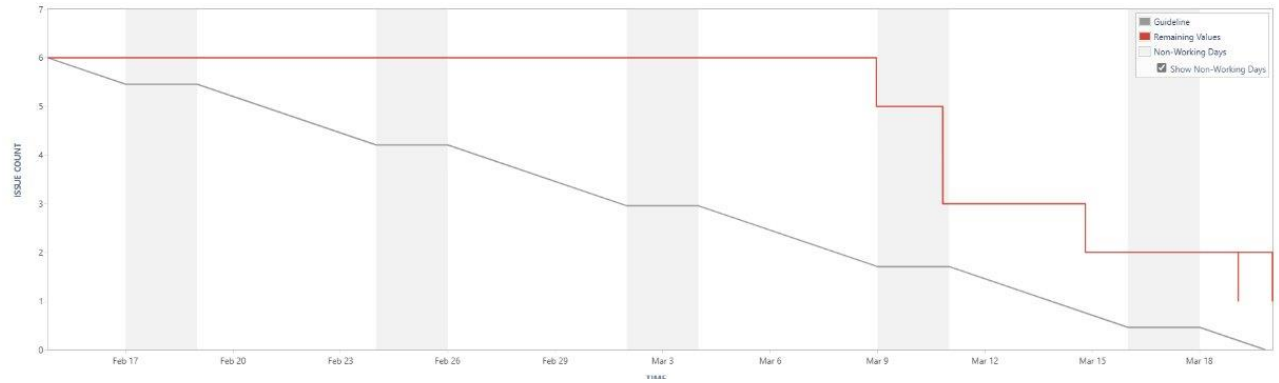
Velocity Chart

Only 1 bar because 2nd sprint issues were not assigned story points

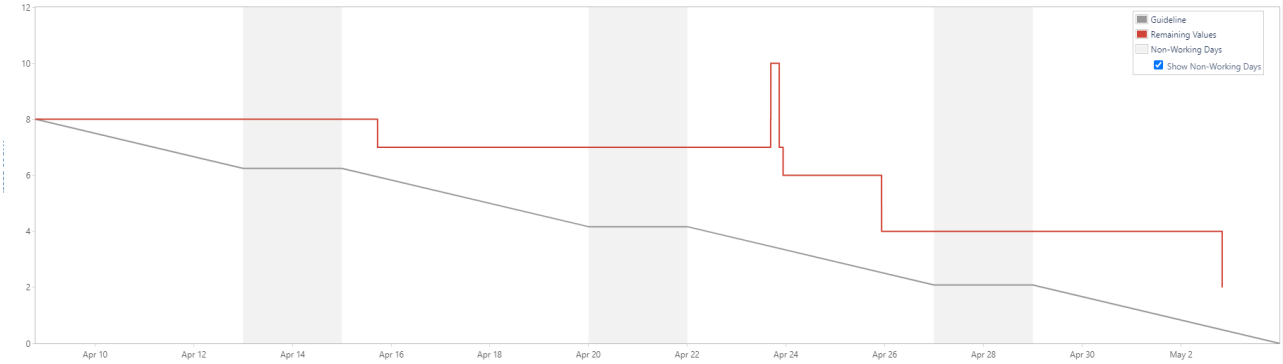
Burndowns Charts

Sprint 1

Sprint 2



Sprint 3



Vlog

This is a short video running through our project: [Vlog]

Features Implemented

- Built visualization to supplement understanding of the dataset
- Trained a model to detect PII in essays
- Built a python-based web application through Taipy
- Built a static website with JavaScript

Known Issues

- No hosting for the Taipy site
- Model has not been added to Taipy site

- The model's accuracy can be improved

TODO

- Use different preprocessing methods
- Try more rebalancing or different hyperparameters
- Try to use a different pre-trained model like RoBERTA or distilBERT
- Supplement training with the additional datasets found
- Add model to Taipy site
- Find a host for the Taipy site – Docker/Container
- Create a Docker image for the site to be deployed anywhere that containers can be run
- Upload Docker image to DockerHub for distribution