HAAG NLP Summarization Week 12

Michael Bock

November 2024

1 Slack Questions

What did you accomplish this week?

- Finished LLaMa training pipeline by carefully selecting validation set and using class weighting to correct for class imbalance
- Used TFIDF to improve classification results
- Identified documents that failed to OCR
- Identified empty documents

What are you planning on working on next?

- Get TFIDF working in PyTorch so I can use class weighting on it.
- Start on end of semester report that we will be giving to UPenn.
- Train on new data provided by OCR team
- If there is time, explore RNNs and seq2seq models again.

What is blocking you from progressing?

- None

2 Abstract

With the rapid growth of Text sentiment analysis, the demand for automatic classification of electronic documents has increased by leaps and bound. The paradigm of text classification or text mining has been the subject of many research works in recent time. In this paper we propose a technique for text sentiment classification using term frequency- inverse document frequency (TF-IDF) along with Next Word Negation (NWN). We have also compared the performances of binary bag of words model, TF-IDF model and TF-IDF with 'next word negation' (TF-IDF-NWN) model for text classification. Our proposed model is then applied on three different text mining algorithms and we found the Linear Support vector machine (LSVM) is the most appropriate to work with our proposed model. The achieved results show significant increase in accuracy compared to earlier methods.

2.1 Brief Analysis

Term Frequency Inverse Document Frequency (TFIDF) is a way to turn a document into a fixed length vector. It counts the number of times words appear in the document and weights rare words more heavily than common words. I was having trouble with LLaMa's size so I used TFIDF as features for a model and got good results. The fact that TFIDF goes from a full document to a fixed length feature vector reminds me a lot of sequence to sequence and I wonder if a tiny model could map onto a fixed length feture vector that is small enough for us to inference on. Dr. Alexander also had the idea of chunking, which is kind of similar to this TFIDF/seq2seq idea.

3 Scripts and Code Blocks

hf_training.py

```
import os
import ast
import pandas as pd
from labels import DNO_ISSUES
from datasets import Dataset
from transformers import AutoTokenizer
from peft import prepare_model_for_kbit_training, AutoPeftModel
from peft import LoraConfig, get_peft_model
import datetime
import os
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification, AutoModel
from transformers import TrainingArguments, Trainer, BitsAndBytesConfig
import numpy as np
import evaluate
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
from torch.utils.data import DataLoader
from torch import nn
import torch
from sklearn.utils.class_weight import compute_class_weight
from accelerate import Accelerator

accelerator = Accelerator()
device = accelerator.device

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_name = "meta-llama/Llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name, token = os.environ['HF_TOKEN'
    ])
batch_size = 2
df = pd.read_csv('dno_labels.csv').dropna()

df['labels'] = [[x[issue_name] for issue_name in DNO_ISSUES] for _, x in df.iterrows
    ()]

class_weights = {}
pos_weighting = []
for issue in DNO_ISSUES:
    if issue != 'Cyber':
```

```
        class_weights[issue] = compute_class_weight(class_weight="balanced", classes=np.array([0, 1]), y=df[issue])
        pos_weighting.append(class_weights[issue][1]/class_weights[issue][0])
    else:
        pos_weighting.append(0.0)

pos_weighting = torch.tensor(pos_weighting).to(device)

samples = {issue: sum(np.array(df[issue])) for issue in DNO_ISSUES}
samples = dict(sorted(samples.items(), key=lambda item: item[1], reverse=False))

train_dataset = {issue: [] for issue in df.columns}
val_dataset = {issue: [] for issue in df.columns}
for issue in samples:
    this_issue = df[df[issue] == 1.0]#.iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
    this_issue_train = this_issue.iloc[:int(len(this_issue) * 0.8)]
    this_issue_val = this_issue.iloc[int(len(this_issue) * 0.8):]

    for column in this_issue_train.columns:
        train_dataset[column].extend(this_issue_train[column])

    for column in this_issue_val.columns:
        val_dataset[column].extend(this_issue_val[column])

    df = df[df[issue] != 1.0]

df_train = pd.DataFrame(train_dataset)
df_val = pd.DataFrame(val_dataset)

train_dataset = Dataset.from_pandas(df_train)#.train_test_split(test_size=0.2)
val_dataset = Dataset.from_pandas(df_val)

#tokenizer = AutoTokenizer.from_pretrained("FacebookAI/xlm-roberta-base")
#tokenizer.add_special_tokens({'pad_token': '[PAD]'})
tokenizer.pad_token = tokenizer.eos_token

max_seq_length = tokenizer.model_max_length  # or model.config.max_position_embeddings
print(f"Model's maximum sequence length: {max_seq_length}")

def tokenize_function(examples):
    examples['Text'] = [e.replace('\n', '') for e in examples["Text"]]
    return tokenizer(examples['Text'], padding="max_length", max_length=max_seq_length//4, truncation=True, return_tensors='pt')

train_tokenized_dataset = train_dataset.map(tokenize_function, batched=True)
train_tokenized_dataset = train_tokenized_dataset.remove_columns(["Text"])
train_tokenized_dataset = train_tokenized_dataset.remove_columns(["Name"])
train_tokenized_dataset.set_format("torch")
train_tokenized_dataset = train_tokenized_dataset.filter(lambda example: len(example['input_ids']) > 1000)#.select(range(100))

val_tokenized_dataset = val_dataset.map(tokenize_function, batched=True)
val_tokenized_dataset = val_tokenized_dataset.remove_columns(["Text"])
val_tokenized_dataset = val_tokenized_dataset.remove_columns(["Name"])
val_tokenized_dataset.set_format("torch")
val_tokenized_dataset = val_tokenized_dataset.filter(lambda example: len(example['input_ids']) > 1000)#.select(range(100))
```

```
93
94  print('Filtered Train: ', len(train_tokenized_dataset))
95  print('Filtered Val:   ', len(val_tokenized_dataset))
96
97  quantization_config = dict(load_in_4bit=True, bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype="bfloat16")
98  #quantization_config=quantization_config
99  base_model = AutoModelForSequenceClassification.from_pretrained(model_name,
        num_labels=len(DNO_ISSUES), problem_type="multi_label_classification",
        torch_dtype = torch.bfloat16, attn_implementation = "flash_attention_2",
        device_map={"": torch.cuda.current_device()})
100
101 base_model = prepare_model_for_kbit_training(base_model)
102
103 lora_config = LoraConfig(r=8, lora_alpha=32, target_modules="all-linear",
        lora_dropout=0.01, task_type="SEQ_CLS",)
104
105 model = get_peft_model(base_model, lora_config)
106 model.config.pad_token_id = tokenizer.pad_token_id
107
108 def numpy_sigmoid(x):
109     return 1 / (1 + np.exp(-x))
110
111 def compute_metrics(eval_pred):
112     logits, labels = eval_pred
113     preds = (numpy_sigmoid(logits) >= 0.5).astype(int)
114     x = precision_recall_fscore_support(preds, labels, average='macro')
115     metrics = dict(precision=x[0], recall=x[1], f1=x[2])
116     x = precision_recall_fscore_support(preds, labels)
117     for i, label_name in enumerate(DNO_ISSUES):
118         metrics[f'{label_name}__f1'] = x[2][i]
119     print(logits, labels)
120     losses = nn.BCEWithLogitsLoss(pos_weight = pos_weighting)(torch.tensor(logits).
        to(device), torch.tensor(labels).to(device))
121     metrics['loss'] = losses
122     return metrics
123
124 now = datetime.datetime.now()
125 logdir = now.strftime('/home/hice1/mbock9/scratch/runs_hf/tensorboard/%Y%m%d_%H%M%S'
        )
126 savedir = now.strftime('/home/hice1/mbock9/scratch/runs_hf/checkpoints/%Y%m%d_%H%M%S
        ')
127
128 #eval_steps=1.0,
129 training_args = TrainingArguments(
130     output_dir=savedir,
131     num_train_epochs=1,
132     per_device_train_batch_size=batch_size,
133     per_device_eval_batch_size=batch_size,
134     gradient_accumulation_steps=1,
135     learning_rate=3e-4,
136     warmup_ratio=0.03,
137     save_strategy='steps',
138     eval_strategy='steps',
139     save_total_limit = 2,
140     load_best_model_at_end = True,
141     report_to='tensorboard',
142     logging_dir=logdir,
```

```
    logging_strategy = 'steps',
    overwrite_output_dir=True,
    ddp_find_unused_parameters = False,
    dataloader_num_workers = 4,
    fp16 = True,
    optim = 'adafactor'
)

class ClassWeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs = False):
        labels = inputs.pop("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        losses = nn.BCEWithLogitsLoss(pos_weight = pos_weighting)(logits.to(device),
     labels.to(device))
        return (losses, outputs) if return_outputs else losses

#train_dataloader = DataLoader(train_tokenized_dataset, shuffle=True, batch_size=8)
#eval_dataloader = DataLoader(val_tokenized_dataset, batch_size=8)

#model, training_dataloader, val_dataloader = accelerator.prepare(model,
    training_dataloader, val_dataloader)
model.gradient_checkpointing_enable(gradient_checkpointing_kwargs={"use_reentrant":
    False})
trainer = accelerator.prepare(ClassWeightedTrainer(model=model, args=training_args,
    train_dataset=train_tokenized_dataset, eval_dataset=val_tokenized_dataset,
    compute_metrics=compute_metrics,))
trainer.train()
```

tfidf_training.py

```
import os
import ast
import pandas as pd
from labels import DNO_ISSUES
from datasets import Dataset
from transformers import AutoTokenizer
from peft import prepare_model_for_kbit_training, AutoPeftModel
from peft import LoraConfig, get_peft_model
import datetime
import os
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification, AutoModel
from transformers import TrainingArguments, Trainer, BitsAndBytesConfig
import numpy as np
import evaluate
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
from torch.utils.data import Dataset, DataLoader
from torch import nn
from torch import optim
import torch
from sklearn.utils.class_weight import compute_class_weight
from accelerate import Accelerator
from sklearn.feature_extraction.text import TfidfVectorizer
from transformers import PreTrainedModel, PretrainedConfig
from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
29
30 df = pd.read_csv('dno_labels.csv').dropna()
31
32 df['labels'] = [[x[issue_name] for issue_name in DNO_ISSUES] for _, x in df.iterrows
       ()]
33
34 df['word_count'] = df['Text'].apply(lambda x: len(x))
35 df = df[df['word_count'] >= 2000]
36
37 df = df.drop(columns=['word_count'])
38
39 class_weights = {}
40 pos_weighting = []
41 for issue in DNO_ISSUES:
42     if issue != 'Cyber':
43         class_weights[issue] = compute_class_weight(class_weight="balanced", classes
       =np.array([0, 1]), y=df[issue])
44         pos_weighting.append(class_weights[issue][1]/class_weights[issue][0])
45     else:
46         pos_weighting.append(0.0)
47
48 pos_weighting = torch.tensor(pos_weighting).to(device)
49
50 samples = {issue: sum(np.array(df[issue])) for issue in DNO_ISSUES}
51 samples = dict(sorted(samples.items(), key=lambda item: item[1], reverse=False))
52
53 train_dataset = {issue: [] for issue in df.columns}
54 val_dataset = {issue: [] for issue in df.columns}
55 for issue in samples:
56     this_issue = df[df[issue] == 1.0]#.iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
57     this_issue_train = this_issue.iloc[:int(len(this_issue) * 0.8)]
58     this_issue_val = this_issue.iloc[int(len(this_issue) * 0.8):]
59
60     for column in this_issue_train.columns:
61         train_dataset[column].extend(this_issue_train[column])
62
63     for column in this_issue_val.columns:
64         val_dataset[column].extend(this_issue_val[column])
65
66     df = df[df[issue] != 1.0]
67
68 df_train = pd.DataFrame(train_dataset)
69 df_val = pd.DataFrame(val_dataset)
70
71 vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5, min_df=5, stop_words="
       english")
72 X_train = vectorizer.fit_transform(df_train['Text']).toarray().astype(np.float32)
73 X_val = vectorizer.transform(df_val['Text']).toarray().astype(np.float32)
74
75 class TfIdf(Dataset):
76     def __init__(self, train = True):
77         if train:
78             self.df = df_train
79             self.X = X_train
80         else:
81             self.df = df_val
82             self.X = X_val
83
```

```
    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        return self.X[idx], torch.tensor(self.df['labels'][idx])

#base_model = AutoModelForSequenceClassification.from_pretrained(model_name,
    num_labels=len(DNO_ISSUES), problem_type="multi_label_classification",
    device_map={"": torch.cuda.current_device()})
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_hidden_layers):
        super(MLP, self).__init__()

        layers = []
        # Input layer
        layers.append(nn.Linear(input_size, hidden_size))  # First hidden layer

        # Hidden layers
        for _ in range(num_hidden_layers - 1):
            layers.append(nn.ReLU())  # Activation function
            layers.append(nn.Linear(hidden_size, hidden_size))  # Next hidden layer

        layers.append(nn.ReLU())  # Activation function for last hidden layer
        layers.append(nn.Linear(hidden_size, output_size))  # Output layer
        layers.append(nn.Sigmoid())  # Softmax activation for multi-class
    classification

        # Create the model using nn.Sequential
        self.model = nn.Sequential(*layers)
    def forward(self, x):
        return self.model(x)

train_ds = TfIdf(train = True)
val_ds = TfIdf(train = True)

input_size = X_val.shape[1]  # Number of features (words)
hidden_size = 128  # Arbitrary hidden layer size
output_size = 51  # Number of labels

model = MLP(input_size, hidden_size, output_size, 3)

# Binary Cross-Entropy loss for multi-label classification
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch_size = 16
train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=True)
epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in tqdm(train_loader, total = len(train_loader)):

        outputs = model(inputs)

        loss = criterion(outputs, labels)

```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader)}")
```

tfidf notebook, which results are taken from

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import RidgeClassifier
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from labels import DNO_ISSUES
from sklearn.metrics import RocCurveDisplay
import numpy as np
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, roc_auc_score
df = pd.read_csv('./dno_labels.csv').dropna(how = 'any')

df = pd.read_csv("dno_labels.csv", sep=",").dropna()
samples = {issue: sum(np.array(df[issue])) for issue in DNO_ISSUES}
samples = dict(sorted(samples.items(), key=lambda item: item[1], reverse=False))
print(samples)
train_dataset = {issue: [] for issue in df.columns}
val_dataset = {issue: [] for issue in df.columns}
#train_df
for issue in samples:
    #print(df[df[issue] == 1.0].iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
    this_issue = df[df[issue] == 1.0]#.iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
    this_issue_train = this_issue.iloc[:int(len(this_issue) * 0.8)]
    this_issue_val = this_issue.iloc[int(len(this_issue) * 0.8):]

    for column in this_issue_train.columns:
        train_dataset[column].extend(this_issue_train[column])

    for column in this_issue_val.columns:
        val_dataset[column].extend(this_issue_val[column])

    df = df[df[issue] != 1.0]
    #print(df.iloc[:])

df_train = pd.DataFrame(train_dataset)
df_val = pd.DataFrame(val_dataset)
vectorizer = TfidfVectorizer(
        sublinear_tf=True, max_df=0.5, min_df=5, stop_words="english"
    )

X_train = vectorizer.fit_transform(df_train['Text'])
X_val = vectorizer.transform(df_val['Text'])

from tqdm import tqdm

plt.figure()
f1_scores = {}
for issue in tqdm(DNO_ISSUES):
    clf = MLPClassifier()
```

```
    if issue != 'Cyber':
        clf.fit(X_train, df_train[issue])

        pred = clf.predict(X_val)
        f1_scores[issue] = f1_score(df_val[issue], pred)

        prob = clf.predict_proba(X_val)[:, 1]
        fpr, tpr, thresholds = roc_curve(df_val[issue], prob)

        # Calculate AUC
        auc = roc_auc_score(df_val[issue], prob)

        # Plot ROC curve

        plt.plot(fpr, tpr, label=f'ROC Curve {issue} (AUC = {auc:.2f})')
        plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

how I fixed the problem with the green line where some classes had no validation data

```
df = pd.read_csv("dno_labels.csv", sep=",").dropna()
samples = {issue: sum(np.array(df[issue])) for issue in DNO_ISSUES}
samples = dict(sorted(samples.items(), key=lambda item: item[1], reverse=False))
print(samples)
train_dataset = {issue: [] for issue in df.columns}
val_dataset = {issue: [] for issue in df.columns}
#train_df
for issue in samples:
    #print(df[df[issue] == 1.0].iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
    this_issue = df[df[issue] == 1.0]#.iloc[:int(len(df[df[issue] == 1.0]) * 0.2)])
    this_issue_train = this_issue.iloc[:int(len(this_issue) * 0.8)]
    this_issue_val = this_issue.iloc[int(len(this_issue) * 0.8):]

    for column in this_issue_train.columns:
        train_dataset[column].extend(this_issue_train[column])

    for column in this_issue_val.columns:
        val_dataset[column].extend(this_issue_val[column])

    df = df[df[issue] != 1.0]
    #print(df.iloc[:])

df_train = pd.DataFrame(train_dataset)
df_val = pd.DataFrame(val_dataset)
```

4 Documentation

Two things have happened. First, in the LLaMa trainer I corrected the class imbalance problems I was having last week. K Folding wasn't the solution there, instead I very carefully put the dataset together by going through the dataset class by class and adding 20% of each class to the validation set, and the rest to the training set. This resulted in a train/val split of 80/20.

I trained a model using sklearn's TFIDF function, which produced a shorter fixed length feature vector for me to use. I then trained separate sklearn MLP Classifiers for each class and recorded their validation performance. I did this because when I tried using LLaMa, it took 11 hours per epoch, which was more than the 8 hour time limit on PACE. The 11 hour runtime limited my ability to experiment and was slowing progress too much, so we moved away from LLaMa because TFIDF ran faster.

5 Script Validation(Optional)

N/A, quicker script

6 Results Visualization

Issue Name	**TDIDF F1 Score**	**LLaMa F1 Score**
Bodily injury	0.285	0.0
Didn't settle when should have	0.129	0.0
Fraud/criminal/illegal conduct	0.2041	0.0625
Restitution/ disgorgement is not "Loss"	0.478	0.0
Settlement amount unreasonable	0.0	NaN
What is a "Claim"?	0.434	0.0
"Insured" v "Insured"	0.488	0.0
Failed to get insurer consent	0.269	0.0
Other "Loss" issues	0.348	0.0
Other issues arising from insurer settlement conduct	0.127	0.0
Property damage	0.129	0.0
What is a "Securities Claim"?	0.456	0.0
Libel or slander	0.0	0.0
Other issues arising from PH settlement conduct	0.0	0.0
Unjust enrichment (profits not entitled)	0.487	0.0
What is a "Related Claim" / "Interrelated Wrongful Act"?	0.548	0.0606
Employment practices	0.0	0.0
Prior claim / notice	0.352	0.0
What counts as "Loss"?	0.630	0.1667
Prior acts	0.1875	0.0
Professional services	0.583	0.0
Who is an "Insured"?	0.388	0.0
Fiduciary liability	0.0	NaN
Prior or pending litigation / proceeding	0.2	0.0
Wrongful act not in capacity as a director or officer of the insured	0.326	0.0
Cyber	0	NaN
Late notice or reporting issue	0.480	0.0588
Prior knowledge	0.2	0.0
Bump up	0.815	0.0
Retro date issue	0.167	0.0

Misrepresentation/Rescission	0.356	0.0
Regulatory	0.571	NaN
Insolvency	0.222	NaN
Market segmentation exclusion issues	0.552	0.0
Contract	0.439	0.0
Exclusion issues	0.745	0.4545
Antitrust/restraint of trade/unfair business practice	0.0	0.0
Severability	0.0	0.0
Insurer refused to pay defense	0.620	0.6900
Privacy/IP	0.0	0.0
Laser exclusion	0.16	0.0
PH failed to cooperate	0.319	0.0
PH settlement conduct	0.213	0.0
What counts as "Final Adjudication"	0.174	0.0
Insurer settlement conduct	0.128	0.0
Other exclusion issues	0.454	0.0
Other insurance	0.510	0.0
Allocation	0.468	0.0
Arbitration	0.0	0.0
Bad faith	0.658	0.2857
Other Coverage Issues	0.508	0.0

Table 1: TFIDF F1 vs LLaMa F1

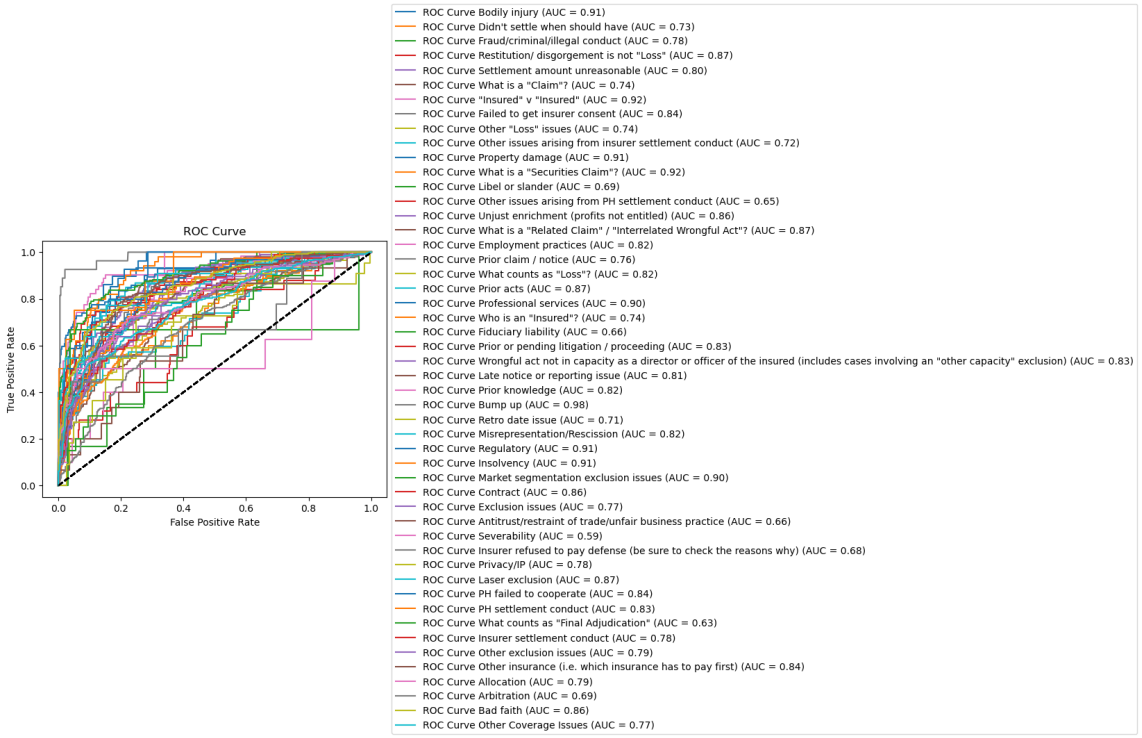

Figure 1: ROC Curves from TFIDF Model

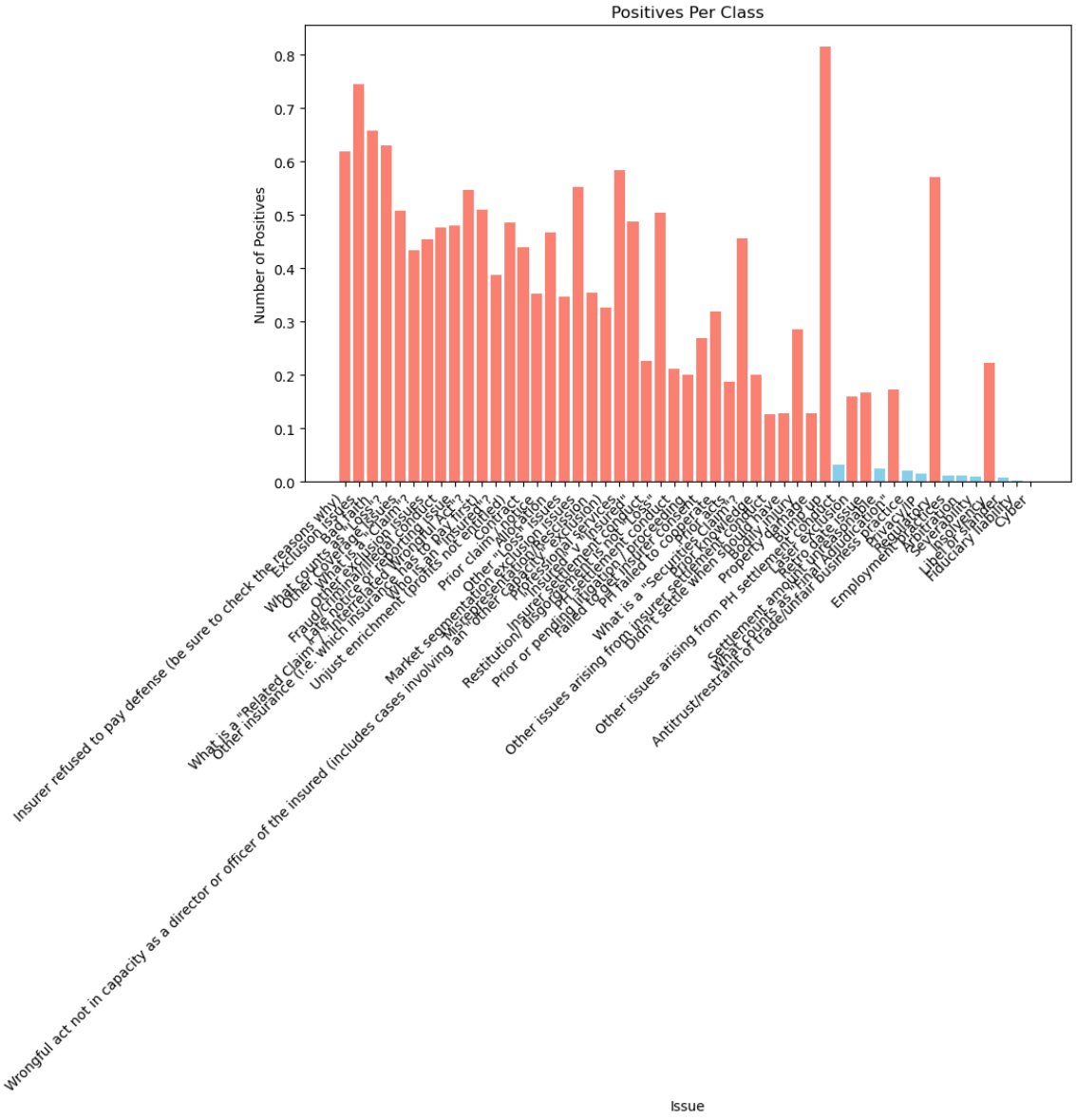

Figure 2: F1 Scores and Positive Rate

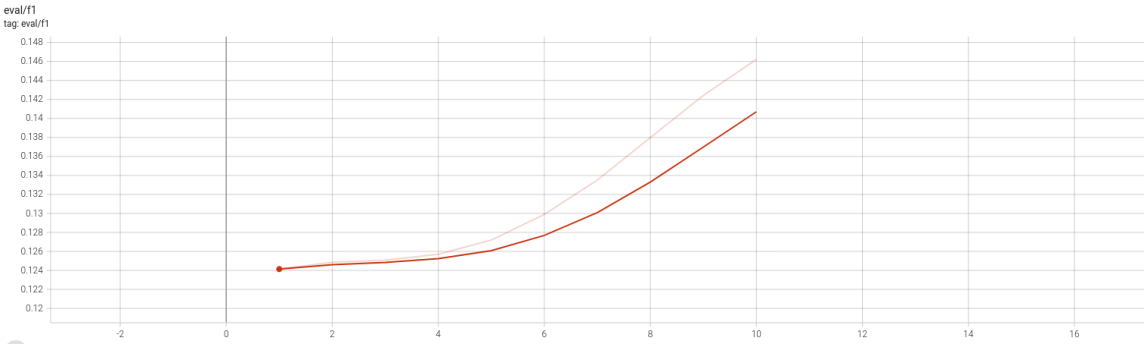

Figure 3: Eval F1 Rising overtime

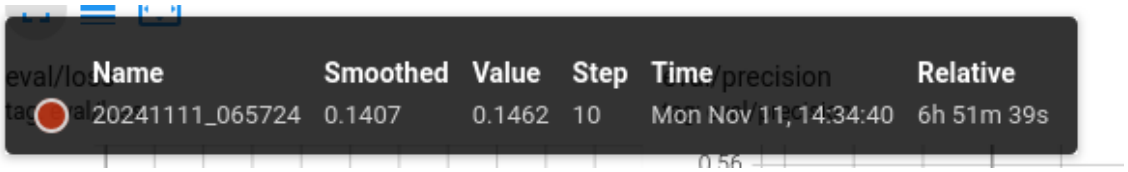

Figure 4: Runtime Too Long, this run was only out to around half an epoch, which is why it says 6 hours.

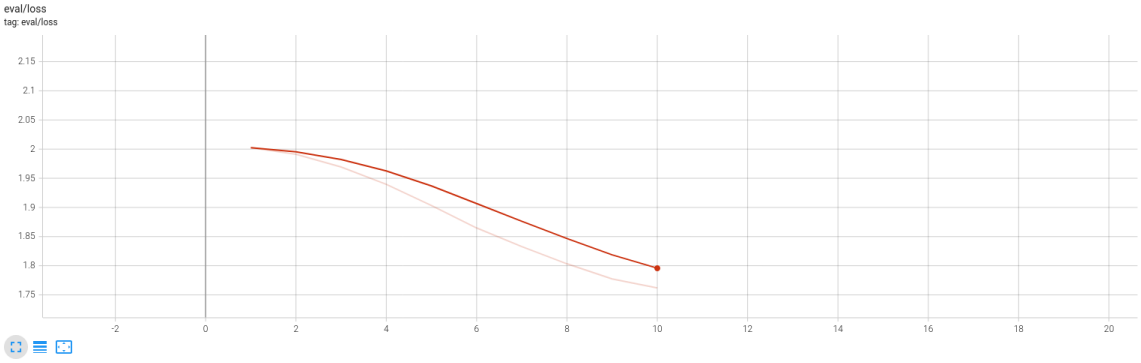

Figure 5: Eval loss goes down, model learns

7 Proof of work

I think the TFIDF results and the LLaMa results are reasonable. The Eval F1 should go up during training as more examples are seen by LLaMa so I'm not surprised by these. We haven't been using LLaMa lately because of its long runtime. The long runtime comes from not having a large batch size due to long texts and from the number of parameters LLaMa has.

I want to argue that TFIDF with a MLP on the back of it is a good enough classifier on DNO issues that this technique or techniques like it (seq2seq) should be pursued for the rest of the DNO project over techniques like LLaMa and Longformer. At the core of my argument is Figure 1. The point is that black dashed line is the result of randomly guessing; it's a random chance line. AUC stands for area under curve, which measures the integral of each ROC curve in the plot and is in the legend. The random chance line has an AUC of 0.5. So if you have an AUC that is more than 0.5, it means the classifier is better than randomly guessing. An AUC below 0.5 means that the model is worse than randomly guessing. All of the AUC scores - even for classes that have F1 scores of 0 in Table 1 have an AUC greater than 0.5, meaning that the classifier is at least better than randomly guessing for all of them. Additionally, at every point, almost all of the classes' ROC curve is above the random chance line. Some undergraduate students are looking into why certain issues like Severability are parts below the random chance line and we think it might have something to do with OCR failing. We are working to get a new dataset with more OCR'd cases and we are also trying to remove cases which only have cover pages.

8 Next Week's proposal

- Get TFIDF working in PyTorch so I can use class weighting on it.
- Start on end of semester report that we will be giving to UPenn.
- Train on new data provided by OCR team
- If there is time, explore RNNs and seq2seq models again.

HAAG Research Report
NLP - Sentencias / NLP - Gen Team
Week 13

Víctor C. Fernández
November 2024

1 WEEKLY PROJECT UPDATES

What progress did you make in the last week?

- Looked into prompt engineering to improve the Llama model's performance
- Completed presentation for call with Judge Miguel updating our project's current status
- Worked on a pipeline to orchestrate the whole process using prefect.

What progress are you making next?

- Filling in a preliminary version for the project's paper.
- Complete full implementation of a simple pipeline to run all processes end to end.
- Document and upload all code to the Sentencias private folder.

Is there anything blocking you from making progress?

No, no blockers right now.

2 ABSTRACTS

1. **Title:** Process mining-enabled jurimetrics: analysis of a Brazilian court's judicial performance in the business law processing
   - **URL:** https://dl.acm.org/doi/10.1145/3462757.3466137 (may require accessing with GaTech credentials to view actual paper)

   - **Abstract:** Improving judicial performance has become increasingly relevant to guarantee access to justice for all, worldwide. In this context, technology-enabled tools to support lawsuit processing emerge as powerful allies to enhance the justice efficiency. Using electronic lawsuit management systems within the courts of justice is a widespread practice, which also leverages production of big data within judicial operation. Some jurimetrics techniques have arisen to evaluate efficiency based on statistical analysis and data mining of data produced by judicial information systems. In this sense, the process mining area offers an innovative approach to analyze judicial data from a process-oriented perspective. This paper presents the application of process mining in a event log derived from a dataset containing business lawsuits from the Court of Justice of the State of Sao Paulo, Brazil - the largest court in the world - in order to analyze judicial performance. Although the results show these lawsuits have an ad hoc sequence flow, process mining analysis have allowed to identify most frequent activities and process bottlenecks, providing insights into the root causes of inefficiencies.

   - **Summary:** The paper explores the application of process mining to evaluate judicial performance in business law cases within the Court of Justice of São Paulo, Brazil. The study leverages process mining techniques to analyze an event log of over 4,700 cases and 266,000 events, identifying procedural bottlenecks and inefficiencies in the judicial process. Key findings include prolonged case durations and specific delays caused by resource constraints and procedural complexities. By employing a process-oriented approach, the study provides actionable insights into the root causes of inefficiencies and proposes potential areas for automation and performance improvement.

   - **Relevance:**This paper focuses on extracting dates and other critical procedural information from Dominican judicial decisions (sentencias) to analyze

court congestion. The application of process mining in this study demonstrates how procedural event logs can uncover bottlenecks and delays in judicial workflows. These techniques could be adapted to identify temporal patterns in the sentencias dataset, enabling a deeper understanding of delays caused by specific procedural steps. Furthermore, the process mining approach aligns with our goal of structuring judicial data to inform policy recommendations aimed at improving case resolution efficiency in the Dominican Republic.

3 SCRIPTS AND CODE BLOCKS

All scripts have been uploaded to the HAAG NLP Repo. Outputs files, processed sentencias and any other document that may contain sensitive information is located in the private NLP-Sentencias Repo.

For this week I've been aiming for multiple ways of querying the model to retrieve the right information only by modifying the prompt and how it is validated afterwards. All the code was executed within PACE given it required a larger memory and GPU for running the Llama 3.1 70B model.

1. Multiple prompts used for querying the model here.

```
Analiza el siguiente texto:

{{DOCUMENT_CONTENT}}

Por favor, según la información en el texto, sustituye
↪  "TO_BE_FILLED_IN" con la opción adecuada que represente lo que
↪  indica la fecha y devuelve solo un JSON.
Utilizando solo las siguientes opciones para la respuesta:

# Opciones:
{{OPTIONS}}

**Importante**: Incluye solo el JSON en la respuesta.

Estos son los datos a rellenar:
{{MODEL_OUTPUT_FORMAT}}
```

Code 1—Prompt version 1

```
Analiza el siguiente texto y, según la información proporcionada,
↪  sustituye "TO_BE_FILLED_IN" con la opción adecuada que
↪  represente lo que indica la fecha. Devuelve solo un JSON.

**Ejemplos**:

Texto: "El 10 de mayo de 2023 se presentó la demanda en el
↪  juzgado."
Opciones: ["fecha de presentación de demanda", ...]
Datos a rellenar:
{
  "date": "10 de mayo de 2023",
  "event": "TO_BE_FILLED_IN"
}
**Respuesta esperada**:
{
  "date": "10 de mayo de 2023",
  "event": "fecha de presentacion de demanda"
}

**Tu turno**:

Texto: "{{DOCUMENT_CONTENT}}"
Opciones:
{{OPTIONS}}
Datos a rellenar:
{{MODEL_OUTPUT_FORMAT}}

**Importante**: Incluye solo el JSON en la respuesta.
```

Code 2—Prompt version 2

```
Analiza el siguiente texto y, según la información proporcionada,
↪  sustituye "TO_BE_FILLED_IN" con la opción adecuada que
↪  represente lo que indica la fecha. Devuelve solo un JSON.

**Ejemplos**:

Texto: "El 10 de mayo de 2023 se presentó la demanda en el
↪  juzgado."
Fecha: "10 de mayo de 2023"
Opciones: ["opción ba", "opción rrr", ...]
Datos a rellenar:
  {"date event": "TO_BE_FILLED_IN"}

**Respuesta esperada**:
{"date event": "opción ba"}

**Tu turno**:

Texto: "{{DOCUMENT_CONTENT}}"
Opciones: {{OPTIONS}}
Fecha: "{{DATE}}"
Datos a rellenar:
{{MODEL_OUTPUT_FORMAT}}

**Importante**: Incluye solo el la opción correcta en la respuesta
↪  en formato JSON.
```

Code 3—Prompt version 3

4 DOCUMENTATION

The pipeline/flow we're currently following is the one below, where we first extract and clean the documents. Afterwards, a process takes care of diving the clean documents into smaller pieces that can be then passed as input to a new layer where a Bert based model in Spanish, that has been fine tuned to

better identify dates over legal documents for the Dominican Republic, is used to retrieve the dates from the corpus. Once these dates have been identified, they will be passed on to an additional model that will then retrieve the context of the date to identify what it is representing. Finally, all dates will be grouped and included in one file, representing the output of all the pieces of the original document being put together.

The following diagram represents this flow:

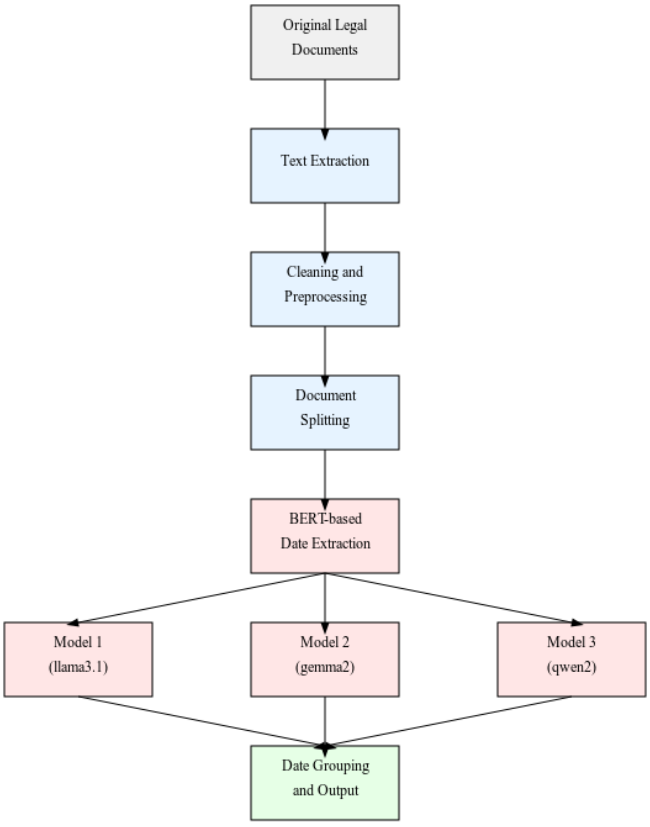

Figure 1—Full date extraction process

This week, my focus has been on the second to last step, with the goal of improving the models results given the low accuracy obtained when performing the benchmark. For the updates, I focused on the Llama models, running the changes mainly with Llama 3.1, Llama 3.2 and Llama 3.1 70b.

Date context extraction

- Input template generated in txt format to feed the model and retrieve the date context. This template contains placeholders to fill in:
  - Date retrieved by model in previous steps, copied exactly from the original text document.
  - 2000 characters window (1000 before and 1000 after the date) from the original text where the date is contained.
  - Options/clusters template containing the categories by which to classify the different dates retrieved.

  The output of the model will be a single text file containing a JSON object with the input date, a JSON object with the model output and a JSON object containing configuration details for the executed model such as hyperparameters used, model's name and execution time.

  Also generated an alternative format where the model only outputs the event related to the date without the original date to compare if this would return better results.

5 SCRIPT VALIDATION

The model was queried over a set of 5 files generating 10 outputs for each of the dates contained in the files. Obtaining additionally, performance metrics from the execution.

Results when benchmarking the updated logic with the 3 mentioned models were the following in the last run:


```
  [
{
  "model_name": "llama3.2",
  "hyperparameters": {
    "temperature": 1e-13,
    "top_k": 5,
    "top_p": 0.5,
    "seed": 42
  },
```

```
    "documents_evaluated": [
      "Demanda en designacion de administrador judicial-0164-2023"
    ],
    "total_correct": 1,
    "total_incorrect": 7,
    "total_false_positives": 72,
    "total_validation_dates": 8,
    "total_model_dates": 80,
    "average_precision": 0.0136986301369863,
    "average_recall": 0.125,
    "average_f1_score": 0.024691358024691357,
    "average_accuracy": 0.125,
    "total_processing_time": 24.869086265563965,
    "average_processing_time_per_execution": 0.3108635783195496
  },
  {
    "model_name": "llama3.1",
    "hyperparameters": {
      "temperature": 1e-13,
      "top_k": 5,
      "top_p": 0.5,
      "seed": 42
    },
    "documents_evaluated": [
      "Demanda en designacion de administrador judicial-0164-2023"
    ],
```

```
    "total_correct": 0,
    "total_incorrect": 8,
    "total_false_positives": 0,
    "total_validation_dates": 8,
    "total_model_dates": 2,
    "average_precision": 0.0,
    "average_recall": 0.0,
    "average_f1_score": 0.0,
    "average_accuracy": 0.0,
    "total_processing_time": 0.6077580451965332,
    "average_processing_time_per_execution": 0.3038790225982666
  },
  {
    "model_name": "llama3.1:70b",
    "hyperparameters": {
      "temperature": 1e-13,
      "top_k": 5,
      "top_p": 0.5,
      "seed": 42
    },
    "documents_evaluated": [
      "Demanda en designacion de administrador judicial-0164-2023"
    ],
    "total_correct": 2,
    "total_incorrect": 6,
    "total_false_positives": 59,
    "total_validation_dates": 8,
    "total_model_dates": 67,
    "average_precision": 0.03278688524590164,
    "average_recall": 0.25,
    "average_f1_score": 0.05797101449275363,
    "average_accuracy": 0.25,
    "total_processing_time": 87.11048579216003,
    "average_processing_time_per_execution": 1.3001565043605976
  }
]
```


Code 4—Benchmark results for Llama 3.1, Llama 3.1 70b and Llama 3.2

Execution would be carried out with the following hyperparameters:

- Temperature = 0.0000000000001,
- Top_k = 5,
- Top_p = 0.5
- Seed = 42

Here is a brief explanation of these hyperparameters:

- **Temperature**: A very low temperature (0.0000001) ensures that the outputs will be highly predictable. This is useful when we are looking for consistency and want results to be stable over time.
- **Top-k**: This limits the choices to only the top 5 probable words. This ensures that the model generates meaningful outputs without straying into highly unlikely predictions. It balances between randomness and relevance.
- **Top-p**: Combined with top-k, this gives fine control over the diversity of model output. A top_p value of 0.5 means the model will only consider words that make up 50% of the total probability distribution, ensuring more relevant results.
- **Seed**: Setting the seed makes the experiments reproducible, helpful for research purposes. With the same inputs and hyperparameters, in theory, we should get the same outputs every time (but in practice this doesn't always happen).

6 RESULTS VISUALIZATION

The following images provide the results compared between the different models when retrieving the context for the date given as an input to the model on the last run.

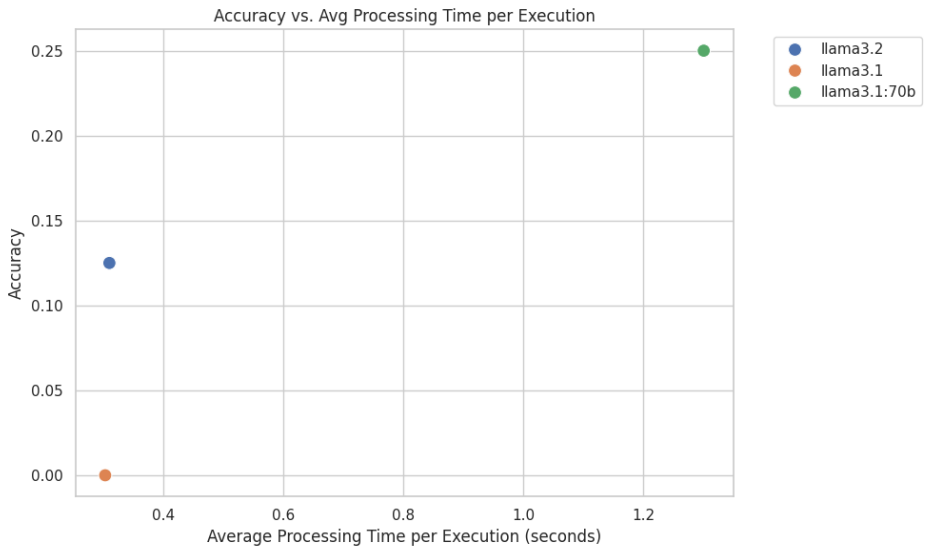

Figure 2—Accuracy vs. Processing time for each model

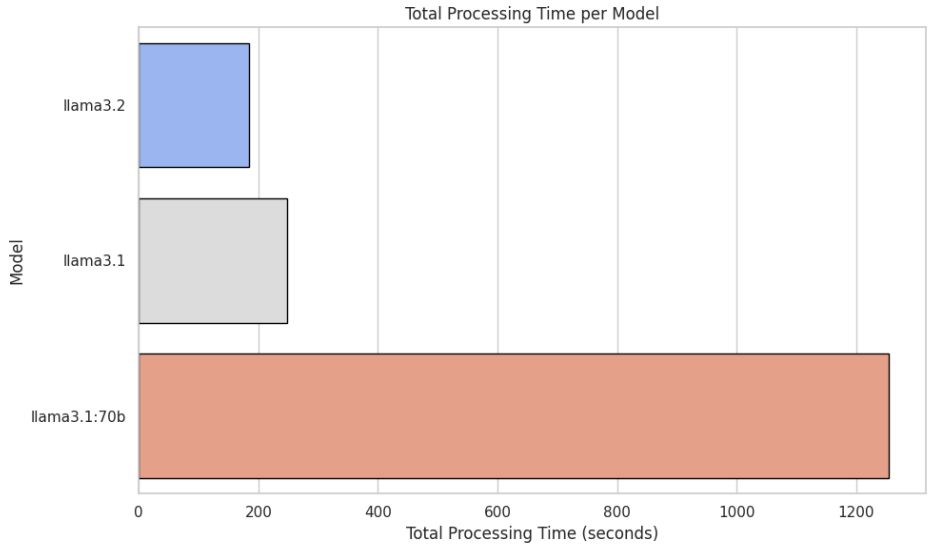

Figure 3—Total processing time for each model

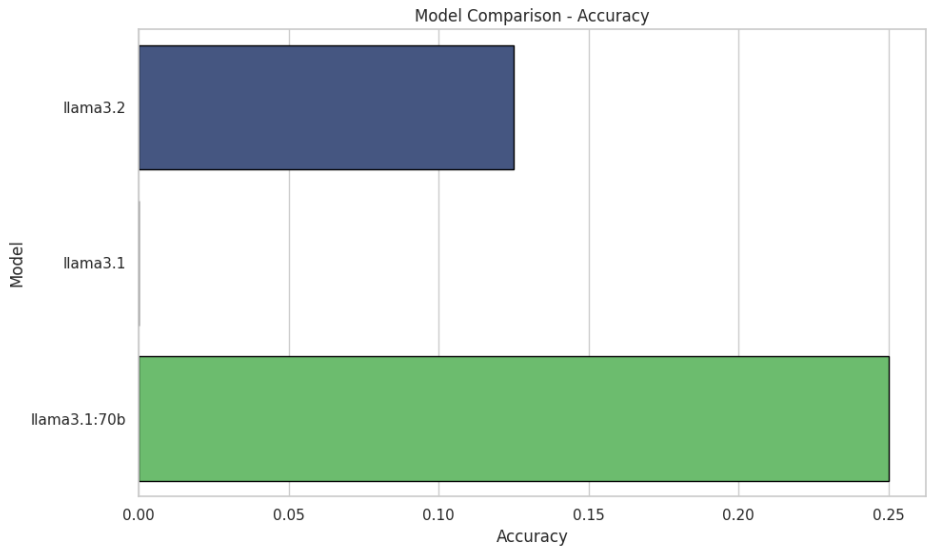

Figure 4—Accuracy comparison between models

7 PROOF OF WORK

Having the implemented function now a smaller piece of context to retrieve the class related to the date. This has improved the model's accuracy, Additionally, by modifying the prompt and the way the response was assessed, the Llama3.1 70b model is now able to achieve nearly 25% accuracy, which is approximately twice than before. Still, this isn't enough and will continue tweaking the prompt to obtain better results.

Below is an example of a date and context retrieved by the updated code to be used for verifying the model's new accuracy. Similarly to what was performed on the previous week, to ensure stability in the results, each input will be used 10 times to generate an output.

```
'seis (06) días del mes de enero del año dos mil
↪  veintitrés (2023)'
```

Code 5—Date to be classified

```
\nPRESIDENCIA DE LA CÁMARA CIVIL Y COMERCIAL DEL JUZGADO
↪ DE PRIMERA INSTANCIA DEL DISTRITO NACIONAL \n
↪ Ordenanza civil núm. 123-4567-ABCD-8901 Número único
↪ de caso (NUC) 1234-0158080 EN NOMBRE DE LA REPÚBLICA
↪ \n Ordenanza civil núm. 504-2023-SORD-0013 Número
↪ único de caso (NUC) 1234-0158080 En la ciudad de Santo
↪ Domingo de Guzmán, Distrito Nacional, capital de la
↪ República Dominicana, a los seis (06) días del mes de
↪ enero del año dos mil veintitrés (2023); años ciento
↪ setenta y nueve (179) de la Independencia y ciento
↪ sesenta (160) de la Restauración. \n \nPresidencia de
↪ la Cámara Civil y Comercial del Juzgado de Primera
↪ Instancia del Distrito Nacional, localizada en el
↪ primer piso del Palacio de Justicia del Centro de los
↪ Héroes de Constanza, Maimón y Estero Hondo, en el
↪ Distrito Nacional, República Dominicana, presidida por
↪ XXXXXXXXXX, quien dicta esta ordenanza en sus
↪ atribuciones de juez presidente de los referimientos y
↪ en audiencia pública constituida por la secretaria
↪ XXXXXXXX. \nXXXXXXXX, y el alguacil de estrados de
↪ turno. \n \nCon motivo de la demanda en referimiento
↪ sobre producción forzosa y entrega inmediata de
↪ certificado de matrícula interpuesta por la señora
↪ XXXXXXXX, dominicana, mayor de edad, titular de la
↪ cédula de identidad y electoral núm. 123-45678-1, con
↪ su domicilio en la calle XXXXXX, núm. 1, torre XXXXXX,
↪ apartamento núm. 1, urbanización XXXX, XXXX
```

Code 6—Window for retrieving date class

8 NEXT WEEK'S PROPOSAL

1. Filling in a preliminary version for the project's paper.
2. Complete full implementation of a simple pipeline to run all processes end to end.

3. Document and upload all code to the Sentencias private folder.

Week 13 | HAAG - NLP | Fall 2024

Alejandro Gomez

November 15th, 2024

1 Time-log

1.1 What progress did you make in the last week?

- This week was a pretty big effort - It was a ton effort to try to get everything wrapped up to be able to present to the stakeholders toady. I finished the large refactor of the NER pipeline that is intended to be submitted with our publication. This refactor is more modular and has configurations that can be tweaked in a config.py file for ease of use. Given this refactor and the previous help from Nathan, I was ready to deploy this NER model publicly to HuggingFace, so I shipped this. Additionally, I used HuggingFace Spaces leveraging the streamlit library to develop a proof of concept for the use of the NER model. There is a GUI that allows a user to submit a sentencia as a PDF or docx file and it will process the file and return the recognized named entities, i.e. the extracted dates. Lastly, I presented with the team to the judge from the Dominican Republic where we demonstrated our PoC and gathered further information on next steps for the contextual NER and how our developoments could best serve the judicial systems in all of Latin America.

1.2 What are you planning on working on next?

- I need to write up the documentation for the deployed NER model on HuggingFace, i.e. how to use it, expected behavior, etc.
- I need to do some deep QA and last minute adjustments on our deployed NER model on huggingface
- Major shift in focus to writing the paper with the team.

1.3 Is anything blocking you from getting work done?

N/A

2 Article Review

2.1 Abstract

Improving judicial performance has become increasingly relevant to guarantee access to justice for all, worldwide. In this context, technology-enabled tools to support lawsuit processing emerge as powerful allies to enhance the justice efficiency. Using electronic lawsuit management systems within the courts of justice is a wide- spread practice, which also leverages production of big data within judicial operation. Some jurimetrics techniques have arisen to eval- uate efficiency based on statistical analysis and data mining of data produced by judicial information systems. In this sense, the pro- cess mining area offers an innovative approach to analyze judicial data from a process-oriented perspective. This paper presents the application of process mining in a event log derived from a dataset containing business lawsuits from the Court of Justice of the State of Sao Paulo, Brazil – the largest court in the world – in order to analyze judicial performance. Although the results show these law- suits have an ad hoc sequence flow, process mining analysis have allowed to identify most frequent activities and process bottlenecks, providing insights into the root causes of inefficiencies doi[UNF+21]

2.2 Summary

This paper was shared to me by my teammate because of its similarity to our ongoing work. They're focusing on visualizing court cases in order to identify specific bottenecks; pretty much the same goal that we have at long term. Our goal being: extract dates and context, and use that information to develop visualizations and tools that will provide us information with bottlenecks. The researchers in this case mainly leveraged jurimetrics (statistical methods) while we will be focusing on NLP AI models for assistance so we will have distinctive approaches to pursue similar goals.

3 Scripts and Code Blocks

3.1 Code

```
import json

import streamlit as st

from core.ner_pipeline import FileProcessor, NerProcessor

# TODO: add typehints

st.set_page_config(
    page_title="NER DEMO",
    page_icon=":computer:",
    layout="centered",
    initial_sidebar_state="auto",
)

st.title(":robot_face: NER Demo :judge:")
st.subheader(" :date: Date Extraction for _Sentencias_ from DR :flag-do:  ")

html_temp = """
            <div style="background-color:{};padding:1px">

            </div>
            """

with st.sidebar:
    st.markdown("""
    # How to use:
    Upload a ‘pdf‘ or ‘docx‘ file and wait for the model to output JSON identifying the dates in your Spanish legal documents. This will be available for download.
    """)
    st.markdown(html_temp.format("rgba(55, 53, 47, 0.16)"),unsafe_allow_html=True)
    st.markdown("""
    Made by [HAAG x GT](https://sites.gatech.edu/human-augmented-analytics-group/)
    """)

uploaded_file = st.file_uploader(
    "Choose a PDF or DOCX file to extract text, clean it, and perform Named Entity Recognition (NER) for date extraction",
    type=["pdf", "docx"],
    accept_multiple_files=False,
)


if uploaded_file is not None:
    st.write(f"**Uploaded File:** {uploaded_file.name}")

    file_extension = uploaded_file.name.split(".")[-1].lower()
    raw_text = ""

    try:
        with st.spinner("Extracting text..."):
            if file_extension == "pdf":
                raw_text = FileProcessor.extract_text_from_pdf(uploaded_file.read())
            elif file_extension in ["doc", "docx"]:
                raw_text = FileProcessor.extract_text_from_docx(uploaded_file.read())
```

```
            else:
                st.error("Unsupported file type!")
    except Exception as e:
        st.error(f"Error extracting text: {e}")

    if raw_text:
        st.subheader("Extracted Text")
        st.text_area("Raw Extracted Text", raw_text, height=300)

        try:
            with st.spinner("Cleaning text..."):
                cleaned_text = FileProcessor.clean_document(raw_text)
        except Exception as e:
            st.error(f"Error cleaning text: {e}")
            cleaned_text = ""

        if cleaned_text:
            st.subheader("Cleaned Text")
            st.text_area("Cleaned Text", cleaned_text, height=300)

            try:
                with st.spinner("Performing NER..."):
                    ner_processor = NerProcessor()
                    ner_output = ner_processor.process_text(cleaned_text)
                    ner_json = json.loads(ner_output)

                st.subheader("NER Output (JSON)")
                st.json(ner_json, expanded=True)

                # Download buttons
                jsonl_str = json.dumps(ner_json, ensure_ascii=False)
                st.download_button(
                    label="Download NER Output as JSON",
                    data=jsonl_str,
                    file_name="ner_output.json",
                    mime="application/json",
                )

            except Exception as e:
                st.error(f"Error during NER processing: {e}")
        else:
            st.warning("Cleaning process resulted in empty text.")
    else:
        st.warning("No text extracted from the uploaded file.")
```

Listing 1: app.py on HuggingFace Spaces using streamlit

This is the main entrypoint for the NER demo shown during the demonstration in the presentation today. Streamlit's library provides helpful tools to abstract GUI design, keeping the code focused on the logic. If you follow the links later in the report you can also see the modules used to preprocess the uploaded file to prepare it for the NER model.

3.2 List of Scripts

- Full NER pipeline
  - The live demo for the NER model with links to the source code, too
  - The NER model deployed on HuggingFace, used in the NER demo
  - The tentative conference submission for the NER model. This is the refactoring I've been working on that finetunes the model and eventually deploys to HuggingFace

3.3 Documentation

Figure 1: pipeline visualization for code structure

Part of completing the documentation for the NER model is diagrams that showcase the process at high level. I honed on keeping the pipeline isolated to finetuning, so I moved the data preprocessing to it's own directory, i.e. its own pipeline. I will make the edits to this chart, but there is not much variation. I simply encapsulated the pipeline differently. The config worked out just as planned and pictured - beautifully.

3.4 Script Validation (optional)

N/A

3.5 Results Visualization

Figure 2: presentation

This is a screenshot of the NLP-DR teams intro slides for our presentation to Magistrado Miguel Angel.

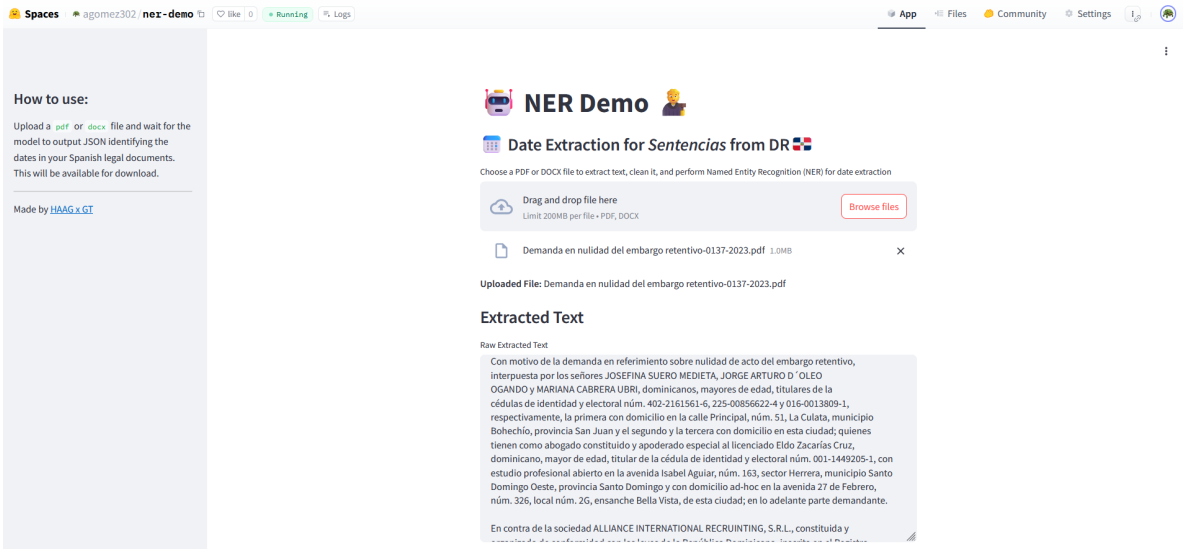

Figure 3: Image 1 of 2 of the NER demo

Figure 4: Image 2 of 2 of the NER demo

The two above images show the NER model demo where the user can upload a document and can retrieve JSON with the relevant information.

3.6 Results Visualization Summary

The demo, i.e the proof of concept is huge milestone. It showcased to our stakeholders a tangible view into our team's development and expected results for their use in the future. It also provided us a platform to build off as we navigate into part 2 of this project, that being the contextual NER.

3.7 Proof of Work

Scripts in GitHub Repo

4 Next Week's Proposal

- (See first section for full list. Brief summary below)
- Documentation for NER model including diagrams
- Make larger contributions toward the publication
- QA NER model further
- Update current documentation, e.g. NLP website.

References

[UNF+21] Adriana Jacoto Unger, José Francisco dos Santos Neto, Marcelo Fantinato, Sarajane Marques Peres, Julio Trecenti, and Renata Hirota. Process mining-enabled jurimetrics: analysis of a brazilian court's judicial performance in the business law processing. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, ICAIL '21, page 240–244, New York, NY, USA, 2021. Association for Computing Machinery.

HAAG NLP Sentencias — Week 13 Report
NLP-Gen Team

Karol Gutierrez

November 15, 2024

1 Weekly Project Update

1.1 What progress did you make in the last week?

- Improved clustering model by adding more data.
- Prepare for call with Sentencias specialists on Friday 11/15.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Continuous meetings with Dr. Alexander, Nathan and team to discuss progress on project and publication options, as well as internal meetings with team to sync on next steps.

1.2 What are you planning on working on next?

- Use feedback from call to improve clustering model.
- Work on paper with team.
- Continue fulfilling my role as Meet Manager/Documentor by working on the tasks expected for my position (gather notes from meetings and prepare recordings).

1.3 Is anything blocking you from getting work done?

No.

2 Literature Review

Paper: FlairNLP at SemEval-2023 Task 6b: Extraction of Legal Named Entities from Legal Texts using Contextual String Embeddings [RE23].

2.1 Abstract

Indian court legal texts and processes are essential towards the integrity of the judicial system and towards maintaining the social and political order of the nation. Due to the increase in number of pending court cases, there is an urgent need to develop tools to automate many of the legal processes with the knowledge of artificial intelligence. In this paper, we employ knowledge extraction techniques, specially the named entity extraction of legal entities within court case judgements. We evaluate several state of the art architectures in the realm of sequence labeling using models trained on a curated dataset of legal texts. We observe that a Bi-LSTM model trained on Flair Embeddings achieves the best results, and we also publish the BIO formatted dataset as part of this paper

2.2 Summary

The authors present an approach to extracting legal named entities from court case judgments using FlairNLP for the SemEval-2023 Task 6b. Their method involves several key steps:

- **Dataset Preparation**: Curating a dataset of legal texts annotated with named entities in a BIO format.
- **Model Architecture**: Using a Bi-LSTM model enhanced with Flair contextual string embeddings to capture deep contextual meaning.
- **Evaluation**: Measuring performance through standard metrics such as precision, recall, and F1-score.

The authors demonstrate that the Bi-LSTM model with Flair embeddings achieves optimal results in legal named entity recognition, outperforming other methods in the task.

2.3 Relevance

This paper is relevant to our Sentencias project because it's tailored to work with a specific benchmark (task 6b in a competition). They use NER and embeddings, which is an approach that we have explored. Both of our projects involve working with legal documents with specialized terminology. We can first try to replicate the results of this paper using versions of libraries and embeddings for the Spanish language.

3 Scripts and code blocks

The code is in the private repository repository. The progress for this week is in `./karol/week13/`.

3.1 Code developed

The full workflow of the code is shown in Figure 1.

- Existing script to use Hugging Face transformer to evaluate semantic similarity of context and then split it into five buckets, Figure 2. The method used is cosine similarity.
- Code to show compare results with test set in Fig 3. This script generates the plots shown in this report.

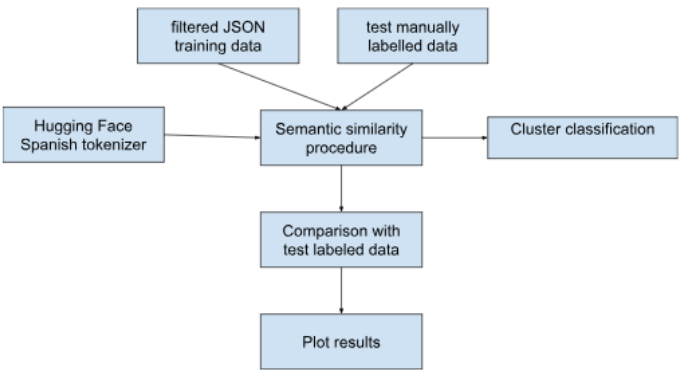

Figure 1: Code logic workflow to process data in Week 13.

4 Documentation

The documentation is present in the README.md file in the repository. No new dependencies were added for this week, all the scripts can be run in Python.

5 Script Validation

Figure 4 shows the generated contexts from the sentencias in Spanish. This document was used to validate the results and provide performance numbers.

6 Results Visualization

I increased the subset of the dates and manually labeled the categories, then compared the performance of my classifier with respect to this set. The results can be seen in Figure 5. The results are better than random by a reasonable margin but still it's not a reliable classifier, however, the accuracy decreased slightly when adding more data, indicating that we probably need a better selection of clusters.

Accuracy: 0.42 Precision: 0.47 Recall: 0.42 F1 Score: 0.39

7 Proof of Work

Figures 7 and Figure 8 show the final distribution of the data and samples of the classification during runtime, thus proving the work.

8 Next Week's Proposal

Refer to section 1.2 for details (avoid repetition).

References

[RE23] Vinay N Ramesh and Rohan Eswara. Flairnlp at semeval-2023 task 6b: Extraction of legal named entities from legal texts using contextual string embeddings, 2023.

sentencias / karol / week12 / **clusterization.py**

Code | Blame | 84 lines (74 loc) · 3.33 KB

```python
folder_path = "./dates_marzo"
output_path = "./plots"
os.makedirs(output_path, exist_ok=True)

# Load Spanish sentence transformer model
model = SentenceTransformer("hiiamsid/sentence_similarity_spanish_es")

# Define representative phrases for each category
category_phrases = {
    "Supreme Court Judgments": "sentencia de la Suprema Corte de Justicia",
    "Appeal Court Decisions": "decisión de la Corte de Apelación",
    "Initial Trial Judgments": "juzgado de primera instancia dicta sentencia",
    "Filing and Appeal Actions": "recurso de apelación interpuesto",
    "Scheduled Dates and Readings": "fecha de lectura de sentencia"
}

# Step 1: Calculate embeddings for each category phrase
print("Calculating category embeddings...")
category_embeddings = {category: model.encode(phrase) for category, phrase in category_phrases.items()}

# Load and encode contexts
contexts = []
context_embeddings = []
print("Loading and encoding contexts...")
for filename in tqdm(os.listdir(folder_path), desc="Files processed"):
    if filename.endswith(".json"):
        with open(os.path.join(folder_path, filename), "r", encoding="utf-8") as file:
            try:
                data = json.load(file)
                if isinstance(data, list):
                    for entry in data:
                        context = entry.get("context", "")
                        if context:
                            contexts.append(context)
                            context_embeddings.append(model.encode(context))
            except json.JSONDecodeError:
                continue

# Step 2: Classify each context based on semantic similarity
classified_contexts = []
print("Classifying contexts based on semantic similarity...")
for i, context_embedding in tqdm(enumerate(context_embeddings), total=len(context_embeddings), desc="Classifying contexts"):
    similarities = {
        category: util.cos_sim(context_embedding, category_embedding).item()
        for category, category_embedding in category_embeddings.items()
    }
    best_category = max(similarities, key=similarities.get)
    classified_contexts.append({"context": contexts[i], "category": best_category})

# Step 3: Save classified data
output_file = os.path.join(output_path, "classified_contexts.json")
with open(output_file, "w", encoding="utf-8") as f:
    json.dump(classified_contexts, f, ensure_ascii=False, indent=4)

# Optional: Visualize category distribution
category_counts = {category: 0 for category in category_phrases.keys()}
for entry in classified_contexts:
    category_counts[entry["category"]] += 1

plt.figure(figsize=(10, 6))
categories, counts = zip(*category_counts.items())
plt.bar(categories, counts, alpha=0.7)
plt.title("Number of Contexts in Each Category")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(os.path.join(output_path, "category_counts.png"))
plt.close()

# Display sample classifications
for category in category_phrases.keys():
    print(f"\nSample contexts for category: {category}")
    sample_contexts = [entry["context"] for entry in classified_contexts if entry["category"] == category][:3]
    for context in sample_contexts:
        print(f" - {context}")
```

Figure 2: Clustering of categories for context

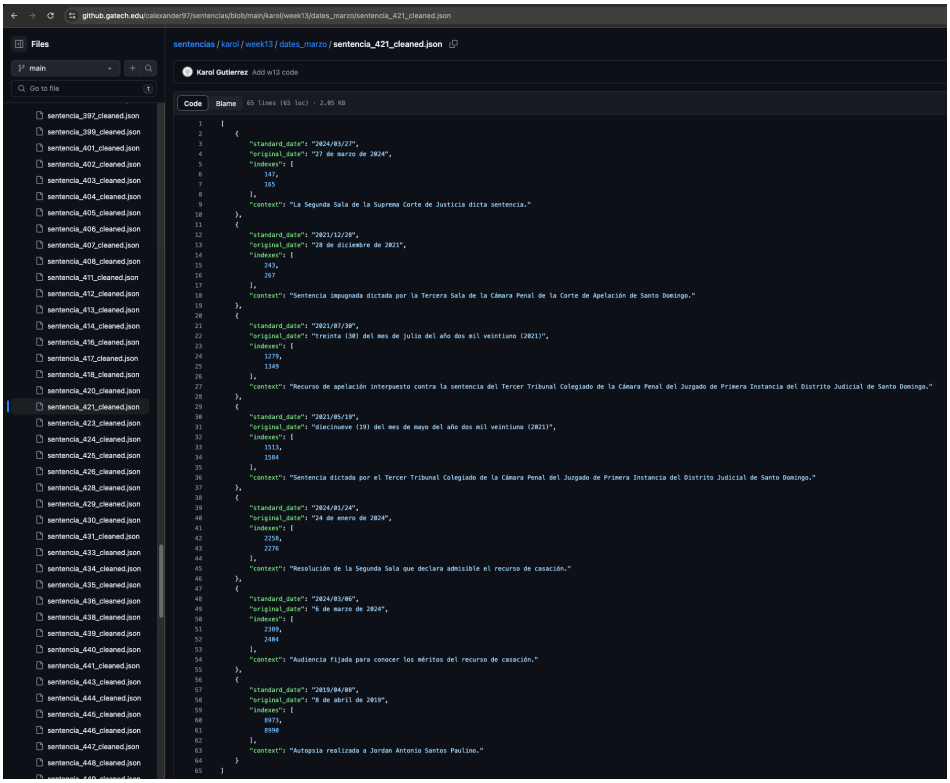

Figure 3: More sentencias data

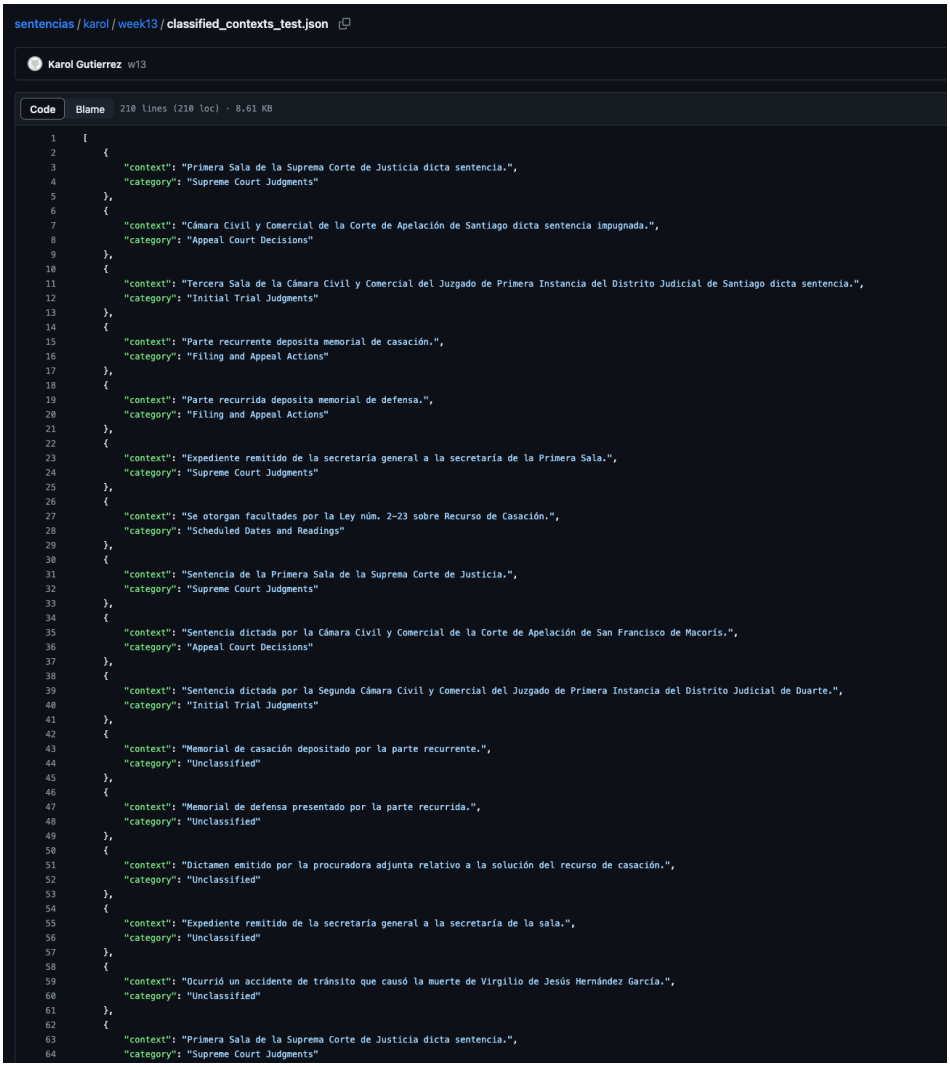

Figure 4: Sentencias used for final testing

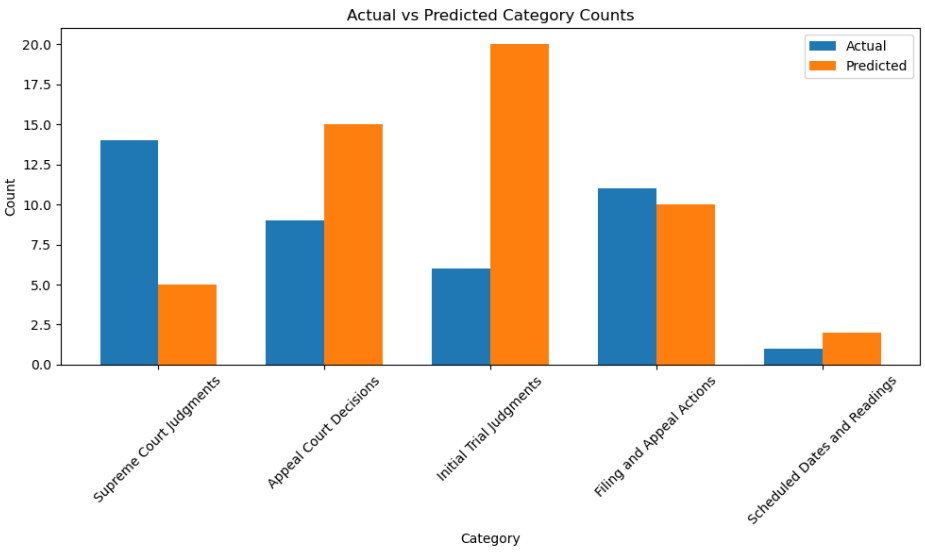

Figure 5: Comparison of results with test data

```
(haag-nlp) ➜  week13 git:(main) ✗ python comparison_test.py
/Users/karol/anaconda3/envs/haag-nlp/lib/python3.9/site-packages/huggingface_hub/file_download.py:1132: Fut
ureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume w
hen possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Calculating category embeddings...
Loading contexts and actual categories...
Classifying contexts based on semantic similarity...
Classifying contexts: 100%|                                        | 52/52 [00:16<00:00,  3.15it/s]
/Users/karol/anaconda3/envs/haag-nlp/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1531: U
ndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Accuracy: 0.42
Precision: 0.47
Recall: 0.42
F1 Score: 0.39
(haag-nlp) ➜  week13 git:(main) ✗
```

Figure 6: Accuracy results

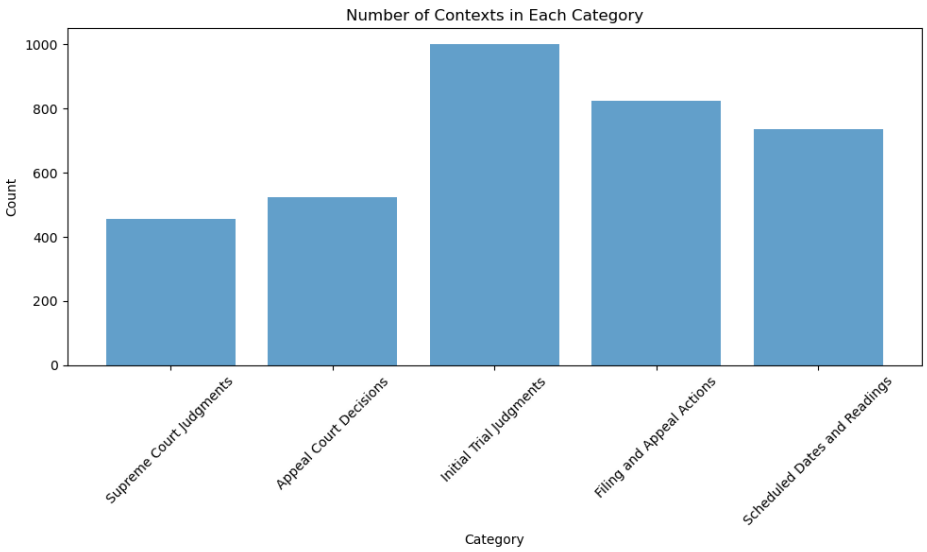

Figure 7: Distribution of categories in available updated data

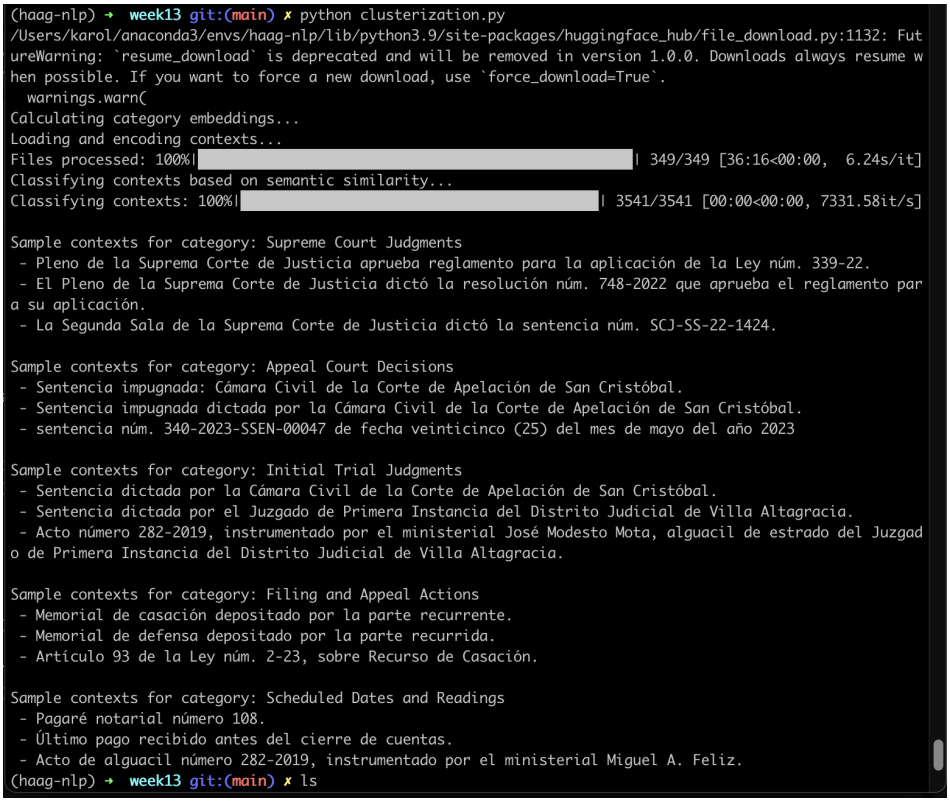

Figure 8: Code working and showing the classified contexts with updated data

Week 13 Report

Thuan Nguyen – Clearinghouse Summarization project

Friday, November 15, 2024

Summary

What progress did you make in the last week?

- In my experiments with summarizing court orders using GPT-4o-mini, I compared two approaches: (1) a structured Chain-of-Thought (CoT) method where the model answered specific legal questions before creating a summary and (2) a more flexible, direct summary generation approach without CoT.
- While the CoT method ensured focused, question-driven outputs, it often missed nuanced details that fell outside the predefined questions, resulting in more generic summaries.
- In contrast, the direct summary approach provided richer, contextually relevant content, capturing a broader range of legal nuances, which ultimately resulted in more detailed and coherent summaries.

What are you planning on working on next?

- Study up on LangGraph on Sam Pang's suggestions. Seems to be highly useful tool to learn for future projects/experiments in the lab.
- Study Sam Pang's previous LLaMA finetuning experiment. Try to extend it with synthetic data generated by gpt-4o (for learning purpose).

Is anything blocking you from getting work done?

- Nothing at the moment.

Abstract

Towards Monosemanticity: Decomposing Language Models With Dictionary Learning

https://transformer-circuits.pub/2023/monosemantic-features/index.html

The "Towards Monosemanticity" paper explores the concept of aligning AI model neurons to have single, clear functions (monosemanticity) to improve interpretability and safety.

It argues that current models often have neurons that respond to multiple, unrelated concepts, making them difficult to understand and control.

The authors propose techniques to train models to develop monosemantic neurons, aiming to isolate specific features or concepts each neuron should represent.

This approach is seen as a step towards making AI systems more transparent and predictable, reducing the risk of unexpected behaviors.

Work done this week – further details

Previously, in my prompt engineering experiment from last week, I asked gpt-4o-mini to generate short details along with verbatim quotes to respond to each question.

Observations: But it turned out that this approach was too restrictive. The model had to rephrase some ideas into short sentences. When combined, these short sentences turn into generic summaries, where a lot of context and concrete details are lost.

This approach generates worse summaries than the simpler method, which just prompts the model to “please summarize this” and “by the way focus on these questions.
https://3.basecamp.com/5835116/buckets/38747617/messages/8026679714#__recording_8030561515

See my scripts for specific approaches.

https://github.com/thuann2cats/NLP-Summarization-Experiment-with-Documents-from-U-Michigan-Civil-Rights-Litigation-Clearinghouse/blob/main/legal_doc_summarizer_Chain_of_Thought.py

https://github.com/thuann2cats/NLP-Summarization-Experiment-with-Documents-from-U-Michigan-Civil-Rights-Litigation-Clearinghouse/blob/main/legal_doc_summarizer.py

Scripts - Documentation - Script Validation - Results Visualization - Proof of Work

For further details, please refer to my scripts posted on GitHub or the information above.

Next week's proposal

- Study up on LangGraph on Sam Pang’s suggestions. Seems to be highly useful tool to learn for future projects/experiments in the lab.
- Study Sam Pang’s previous LLaMA finetuning experiment. Try to extend it with synthetic data generated by gpt-4o (for learning purpose).

Week 13 Research Report

Thomas Orth (NLP Summarization / NLP Gen Team)

November 2024

0.1 What did you work on this week?

1. Been running through different prompts to extract settlement information.
2. Adjusted the feedback from law students to extract more information that comes up in court summaries for the settlement.
3. Worked with the interview team to review summaries and determine the best workflow.
4. Met with the Clearinghouse technical POC for integration discussion.
5. Ran an experiment similar to Thuan's to see if a more open-ended summary technique works better for settlements.

0.2 What are you planning on working on next?

1. Continue refining settlement summaries.
2. Work to provide data schema examples for Jasmine for integration purposes.
3. Start reviewing multi-agent frameworks and landscape.

0.3 Is anything blocking you from getting work done?

1. None currently

1 Abstracts

- Title: Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks. Conference / Venue: Preprint. Link: https://www.microsoft.com/en-us/research/uploads/prod/2024/11/MagenticOne.pdf
- Abstract: Modern AI agents, driven by advances in large foundation models, promise to enhance our productivity and transform our lives by augmenting our knowledge and capabilities. To achieve this vision, AI agents

must effectively plan, perform multi-step reasoning and actions, respond to novel observations, and recover from errors, to successfully complete complex tasks across a wide range of scenarios. In this work, we introduce Magentic-One, a high-performing open-source agentic system for solving such tasks. Magentic-One uses a multi-agent architecture where a lead agent, the Orchestrator, plans, tracks progress, and re-plans to recover from errors. Throughout task execution, the Orchestrator also directs other specialized agents to perform tasks as needed, such as operating a web browser, navigating local files, or writing and executing Python code. Our experiments show that Magentic-One achieves statistically competitive performance to the state-of-the-art on three diverse and challenging agentic benchmarks: GAIA, AssistantBench, and WebArena. Notably, Magentic-One achieves these results without modification to core agent capabilities or to how they collaborate, demonstrating progress towards the vision of generalist agentic systems. Moreover, Magentic-One's modular design allows agents to be added or removed from the team without additional prompt tuning or training, easing development and making it extensible to future scenarios. We provide an open-source implementation of Magentic-One, and we include AutoGenBench, a standalone tool for agentic evaluation. AutoGenBench provides built-in controls for repetition and isolation to run agentic benchmarks in a rigorous and contained manner – which is important when agents' actions have side-effects. Magentic-One, AutoGenBench and detailed empirical performance evaluations of MagenticOne, including ablations and error analysis are available at https://aka.ms/magentic-one.

- Summary: This technical report describes a generic multi-agent system that generalizes to different tasks. It leverages the Autogen framework from microsoft to orchestrate agents.
- Relevance: I wanted to explore multi-agent systems next semester so this type of work would be a useful avenue to explore.

2 Relevant Info

- Summary Chain of Thought (CoT) is a technique to prompt LLMs for information to provide context for summarization. I took a domain centric approach in this experiment to extract entities the Clearinghouse is looking for specifically.
- Llama 3.2 is a popular LLM given its performance
- Ollama is a way to serve LLMs locally
- Langchain is a popular library for interacting with LLMs
- Anthropic is a company that produces the Claude family of models that compete with GPT-4.

- The two best models in terms of accuracy and cost tradeoff is Claude 3.5 Sonnet and Claude 3 Haiku

3 Scripts

1. All scripts uploaded to https://github.com/Human-Augment-Analytics/NLP-Gen
2. Scripts were run with the following file for testing: `https://gatech.box.com/s/foejfx8hly8diex99m5smldvnh7ly4by`
3. Thomas-Orth/anthropic/settlements/domain_specific_scot_chunked.py
   - Brief Description: Run a domain specific version of Summary Chain-of-thought (CoT) on settlements with Anthropic models.
   - Status: Tested by running the pipeline to completion without issue
   - Important Code Blocks:
     (a) First block: Read in CSV file, choose document
     (b) Second block: Run through prompts, chunking documents, save summaries
     (c) Third Block: Evaluate via manual inspection
   - Screenshot of code: No screenshots provided due to the code being largely the same as previous weeks, just with different prompts. Prompts will be pasted at the bottom of the report.
4. Flow Diagram:

Figure 1: Flow diagram

5. Running scripts:
   (a) Download the scripts, the csv from the box link and llm.requirements.txt
   (b) Run: python -m pip install -r llm.requirements.txt
   (c) Sign up for an Anthropic account, generate an API Key, and set "ANTHROPIC_API_KEY" in your environment.
   (d) Run: python (chosen python script)

4 Documentation

1. Download CSV file and
2. Update scripts to point to CSV file
3. Run script to output generated summaries as CSVs
4. Manually evaluate summary

5 Results

5.1 Example Settlement Summary

Below is the example of a settlement summary from Claude Sonnet:

"On March 23, 1990, a court order was issued dismissing a case concerning juvenile detention practices in Iowa. The case was dismissed as moot after Iowa took steps to achieve compliance with the Juvenile Justice and Delinquency Prevention Act (JJDPA) by submitting a plan for policy changes and reducing juvenile jailing in adult facilities. As part of the settlement, attorney fees and costs totaling $276,163.09 were awarded, with the State Defendants being responsible for 90% of the amount. The fees were divided among three attorneys: Harry Swanger received $187,407.90, Blake Parker received $59,203.04, and John Bird received $29,552.15. The settlement marked a significant change in Iowa's approach to juvenile detention practices."

5.1.1 Difference from before

I changed to Sonnet because our interview team evaluated the Haiku summary and said while it was more concise, it would omit some details.

I also am investigating if a separate extraction step is needed for settlements. Thuan noticed for orders and opinions, that going right to summaries performed better than doing a seperate extraction then summarize.

I haven't included those results until I can do a more in-depth review.

5.1.2 Evaluation

The summaries currently are evaluated mainly on the infromation points I added to the prompts. Our interview team will review the summaries to ensure factual correctness, the data points in the prompts are done as well as compare to any additional clearinghouse criteria.

5.2 Prompts

Below are the prompts used by the anthropic model. First prompt will extract key details. The second will take that information to make a summary.

First prompt:

You are a law student tasked with extracting key information from a chunk of a settlement agreement. Your goal is to identify and summarize specific elements of the agreement. Here is the settlement chunk you will analyze:

```
<settlement_chunk>
{document}
</settlement_chunk>
```

Please extract the following information from the settlement chunk:

1. **Actions to be Taken by Defendants:** Describe who has agreed to do what. Be very detailed in providing this information.
2. **Damages (Money):** Identify who is paying for what, including attorney fees. For the money to be paid to plaintiffs, do not name the plaintiffs and report the total sum to be paid to plaintiffs.
3. **Implementation and Enforcement:** Note if there's a court-appointed "monitor" or other oversight.
4. **Duration:** How long the settlement is in effect.
5. **Conditional Agreements:** Mention any conditions for the settlement (e.g., "will only agree IF ...").
6. **Policy Adoptions:** Note any agreement to adopt policies and provide any relevant details about those policies. Do not omit important information and describe in detail.
7. **The Date of the Settlement:** This is typically the document's filing date, the date the document is dated, or the date of execution.
8. **The Type of Settlement:** This is the type of settlement that was entered by this document.

For each piece of information you extract, include a citation of the text from the settlement chunk that supports your conclusion. Use the following format:

```
<citation>[Exact quote from the text]</citation>
```

If any of the requested information is not present in the settlement chunk, state *"Not Specified"* for that item.

If any acronyms are present and their definitions are defined, please spell out the acronym the first time it is used.

After extracting the information, provide a brief summary of your findings.

Important: Do not extract or include the following types of information:

- Introductory and Boilerplate Information
- Reporting Information (how parties must report progress)

- Notice for Class Actions (how parties must give notice to consumers for class action suits)
- Giving Up Claims or Admitting Fault (it's a given that settling parties must give up claims)

Present your findings in the following format:

```
<extracted_information>
1. Actions to be Taken by Defendants:
[Your summary]
[Citation if applicable]

2. Damages (Money):
[Your summary]
[Citation if applicable]

3. Implementation and Enforcement:
[Your summary]
[Citation if applicable]

4. Duration:
[Your summary]
[Citation if applicable]

5. Conditional Agreements:
[Your summary]
[Citation if applicable]

6. Policy Adoptions:
[Your summary]
[Citation if applicable]

7. Date of the Settlement:
[Your info]
[Citation if applicable]

8. Type of Settlement:
[Your info]
[Citation if applicable]
</extracted_information>

<summary>
[Your brief summary of the key points found in the settlement chunk]
</summary>
```

Second Prompt:

You are a law student skilled at distilling sets of extracted information and partial summaries into informative summaries. You will be provided with a set of extracted information and a partial summary about a legal settlement. Your task is to create a concise, one-paragraph summary of the settlement.

Here is the set of extracted information and partial summary:

```
<extracted_info_and_summary>
{chunks}
</extracted_info_and_summary>
```

Using the provided information, create a summary of the settlement following these guidelines:

1. Begin with a sentence describing when the settlement was entered, including the specific date and the type of settlement that was entered.
2. If the case was not dismissed in the settlement, include information on the following aspects, if available:
    - Actions to be Taken by Defendants
    - Damages (Money)
    - Implementation and Enforcement
    - Duration
    - Conditional Agreements
    - Policy Adoptions
3. If the settlement was dismissed, talk about why it was dismissed and what the outcome was.
4. Keep the summary to one paragraph.
5. If any information provides a citation, do not use that information in your summary.
6. Do not omit any of the actions or policy adoptions noted.
7. Write the summary in past tense.
8. If for the requested information, all of the chunks say *"Not Specified"*, do not include that information in the summary.

Carefully review the extracted information and partial summary to ensure you capture all relevant details. Focus on presenting the most important aspects of the settlement in a clear and concise manner.

Please provide your summary within the following tags:

```
<summary>
[Your concise one-paragraph summary here]
</summary>
```

6 Proof of work

The prompts were generated using Anthropic Workbench and ran using their LLMs, so the results are relatively reliable.

6.1 Known Limitations

Currently this is using Claude models. According to our interview team, the best commercial model workflow we've presented has been Gemini. So I need to see if switching to that model with some prompt engineering will help with the summary quality.