University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Visualising Important Sensitivity Classification Features

Mohib Akoum

March 5, 2022

Abstract

Freedom of information act legislates that governmental entities must release a number of documents to the public. However, due to the sensitive nature of these documents, each document needs to be reviewed to identify and protect any sensitivities. Therefore, we created a web-based application that helps users with the classification task of whether a document is sensitive or not using machine learning models. However, to trust these black-box classifiers, users must understand how they make their decisions. The application implements two different algorithms to interpret our models: LIME and SHAP. We implement these two algorithms using SHAP, ELI5 and lime libraries. SHAP uses both LIME and SHAP algorithms; lime and ELI5 both implement the LIME algorithm. However, the ELI5 library implements the algorithm differently since it uses a classifier for explaining predictions, while the lime library fits a regression model on the probability output. We use multiple libraries because they explain classifiers differently, and thus, we want to see which explanations are the most useful for our documents. Furthermore, we create these explanations for three types of models: linear-based, tree-based and deep learning-based. Comparing inherently different machine learning models will allow us to find which kind of model outputs the most valuable features. Moreover, including multiple classifiers can assist reviewers in identifying documents that are difficult to judge for sensitivity if the classifiers disagree about the prediction. Our user study concluded that ELI5 is the most useful for explaining our classifiers. Users chose it 67% of the time for its text highlighting explanations and 49% of the time for its feature importance visualisations. Furthermore, we found that the linear-based classifier is the preferred model for users because they chose it 44% of the time. The deep learning model was the second most selected with 32% and lastly, the tree-based model with 24%. Nevertheless, more work needs to be done to improve the classifiers and their explanations.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Mohib Akoum Date: 5 March 2022

Contents

1 Introduction

This chapter will introduce our project, discussing the motivation and aim behind our system. Furthermore, we give a brief overview of our system and discuss this paper's outline.

1.1 Motivation

Machine learning classifiers have made groundbreaking innovations and breakthroughs in recent years. However, unfortunately, these advances came at the expense of interpretability. Therefore, to trust these models' predictions, we need to understand how they make their classifications. Precisely, what features/terms they utilise to make their decision. Furthermore, it is unwise to use its predictions without understanding how it works because it might be heavily over-fitted on the specific data set, and when deployed in the real world, it will fail. Solving this problem is vital because machine learning is used daily to automate and solve many tasks.

Imagine you work for the government, and they tasked you with labelling sensitive documents. While working, three coworkers are helping you with your task; all three of them tell you their prediction of whether they believe a document is sensitive or not. Moreover, they give you different explanations for their choice to convince you. After enough time, you will find that some explanations are more helpful than others. This project is trying to show precisely that; do multiple classifiers give different predictions?; is it beneficial to have various explanations?; which of the explanation strategies is most valuable?; which of the classifiers is the most helpful in aiding your decision making?

Like friends will differ in their decision-making, different machine learning algorithms differ in how they approach their classification. Therefore, we tested three different classifier types: linear-based, tree-based and deep learning-based. Having multiple classifiers aids the user in deciding which document is sensitive since it gives the user numerous viewpoints. Furthermore, users might find that some classifiers make more sense than others. Finally, if many classifiers have the same prediction, the user can be more confident. However, when classifiers disagree, the user has a more difficult judgement to make and, therefore, needs to know which explanations are most helpful in their decision making. Hence, we utilised multiple explanation libraries to interpret the classifiers, specifically lime, ELI5 and SHAP. Utilising multiple libraries instead of just one will give the user a broader view and potentially more practical explanations.

1.2 Aim

This project aims to provide users with a system that helps them classify documents' sensitivities using interpretable classifiers. However, to accurately measure how good the classifications and explanations are, we have to ask the people who are likely to be using them. Therefore, many individuals have evaluated the classifiers and their predictions on multiple documents. The evaluation gave us data on whether the classifiers' explanations help the users and, if so, which classifiers and explanations are the most useful. The user evaluation also gave us input on whether the visualisations libraries are sophisticated enough to help users understand classifiers and whether different classifiers will result in other explanations.

1.3 System

To achieve our aims, we created a system that contains all of the functionalities previously discussed. This system explains classifiers using feature importance graphs and text highlightings. A feature importance graph visualises the top features (terms) that impacted the classifier prediction. This visualisation shows two types of features, the features that influenced the classifier into thinking that the document is sensitive and the features that impacted the classifier into thinking that the document is non-sensitive. Similarly, the text highlighting also displays these features but highlights them in the document's text instead of showing them on a graph. This web-based application created contains multiple pages to ensure a user-friendly experience. We discuss the different pages in more detail in the list below.

- Collection Sensitivity Analysis: This page contains general information about a classifier and includes visualisations of the critical features on the corpus level. The user can change classifiers using a drop-down list to view specific details about each classifier.
- Sensitive and Non-Sensitive Documents Explanations: This page has the same functionality as the previous page. The only difference is that users can only view documents that at least one classifier predicted as non-sensitive. In other words, if all three classifiers predict a given document to be sensitive, then the document will not appear on this page. This page is helpful because it allows users to filter documents by sensitivity.
- Non-Sensitive Documents Explanations: Similarly to the previous page, this page contains the same functionality. The only difference is that users can only view documents that at least one classifier predicted as non-sensitive. In other words, if all three classifiers predict a given document to be sensitive, then the document will not appear on this page. This page is helpful because it allows users to filter documents by sensitivity.
- Sensitive Documents Explanations: Inversely to the previous page, this page only shows documents that at least one classifier predicted as sensitive.

1.4 Outline

This chapter discussed our motivation and aim behind developing this sensitivity classification system. We also gave an overview of the main pages that the users can utilise. In Further chapters, we discuss specific project areas in more detail. This paper is structured as follows:

- **Chapter 2** discusses related work similar to this project and how our paper improves on previous work. Furthermore, the chapter introduces the fundamentals of machine learning and its explainability.
- **Chapter 3** outlines the software specification process and discusses user stories and feature requirements.
- **Chapter 4** presents the website's high-level design by looking at the architecture alongside some of the frameworks and tools utilised.
- **Chapter 5** shows a low-level view of the implementation of each component in this system.
- **Chapter 6** discusses the testing practices followed in this project.
- **Chapter 7** gives an overview of the user study, including the documents used to evaluate the users and the questions asked. Afterwards, the chapter summarises the results attained from the user evaluation through plots and graphs.
- **Chapter 8** analyses the findings from the previous chapter and discusses the outcomes. Moreover, the chapter also evaluates some weaknesses in our approach.
- **Chapter 9** concludes this dissertation and examines potential future work

2 Background

This chapter discusses related work similar to this project, specifically, papers discussing digital sensitivity reviewing, papers discussing the explainability of machine learning and papers discussing the explainability of legal documents. Furthermore, we discuss how our approach differs from other papers. Afterwards, we discuss the fundamentals of machine learning and its explainability.

2.1 Related work

Multiple papers, similarly to this project, created classifiers for legal documents (Undavia et al. 2018; Howe et al. 2019). Furthermore, some papers created classifiers for sensitive documents specifically (McDonald et al. 2014; 2017) and achieved state-of-the-art performance. However, these papers do not discuss the explainability of their classifiers. Nevertheless, recently, there have been multiple papers acknowledging the lack of explainability in machine learning models and displaying why explainable systems are a critical problem in the field of natural language processing (NLP) (Danilevsky et al. 2020; Liu et al. 2019). Bhambhoria et al. (2021) interpreted two classifiers for legal documents using lime. The authors used a deep learning (Longformer) and a tree-based (XGBoost) model. They concluded that XGBoost and Longformer achieved similar scores and that XGBoost's predictions were more logical after viewing lime's output. However, one thing that Bhambhoria et al. did not consider is the usage of multiple different visualisations simultaneously. Moreover, the authors only looked at the top five common classifier-impacting words, which is a limited view of the overall explainability. Therefore, this paper compares three different classifiers on both the document and corpus levels. For each one of the classifiers, we created three different visualisations (LIME, SHAP, ELI5) methods alongside two different text highlighting methods (LIME, ELI5) to understand the models better. Furthermore, the model outputs are tested on five different documents and evaluated by many users. Finally, this is an end to end product, unlike the papers discussed; this means that the user can utilise the website created and get real-time explanations for the different classifiers and explanations on many documents.

2.2 Supervised classification

This section will view the three different classifiers implemented in this project. Then, we will look at the logic behind how they work and the main differences between them.

2.2.1 Linear classifier

The linear classifier we implemented in this project is a logistic regression (LR) classifier. Logistic regression was first created by David Cox in 1958 (DATA 2019) and is, therefore, one of the oldest classification algorithms. This algorithm works by creating a linear boundary that separates two classes from each other by using a logistic sigmoid function. It then uses that line to calculate the probability of a particular set of data points belonging to one of the two classes. Furthermore, the probabilities are normalised to a range from zero to one. Classification can then occur by

choosing a threshold within this range; for example, if we take a threshold of 0.5, we can say that any data point that lands below that threshold (<0.5) is a non-sensitive document, while any data point that lands above the threshold (>0.5) is a sensitive document. Therefore, this allows logistic regression to do binary classification. Figure 2.1 shows an example of how logistic regression works.

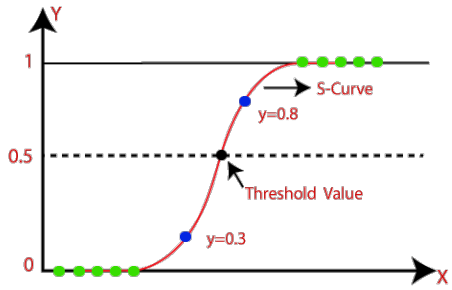


Figure 2.1: *Logistic regression curve with a 0.5 threshold value. Adapted from Javatpoint (2021).*

2.2.2 Tree classifier

The tree classifier our system uses is extreme gradient boosting (XGB). XGB is an effective and widely used machine learning method that has achieved better results than simple tree-models (Chen and Guestrin 2016). To fully explain XGB, we must first discuss decision trees.

Decision trees Decision trees follow a set of if-else statements to decide; the if-else statements are decided based on the structure that maximises the model's accuracy. For an example, view 2.2.

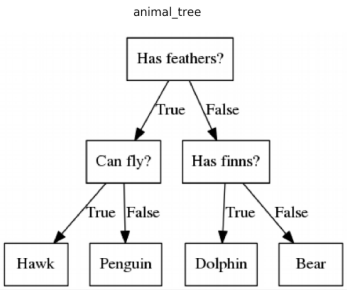


Figure 2.2: *Decision tree for predicting animals. Adapted from Davuluri (2020).*

Random forests Random forests are an ensembling method applied to decision trees, meaning that a random forest consists of many decision trees. The algorithm works by creating many decision trees and using the majority vote as a final prediction. For example, if a forest had five trees and three predicted that the document is sensitive while the other two predicted that the document is non-sensitive, then the algorithm predicts that the document is sensitive because it takes the majority's vote. Therefore, this method assumes that multiple models will perform better than one (one tree).

However, if the trees are identical, this would make forests redundant since the trees will always make the same predictions; therefore, the technique needs to ensure that the trees are decorrelated. It achieves this by using two different decorrelation methods, specifically bootstrap aggregating (bagging) and feature randomness.

- Bootstrap aggregating: the bootstrapping part of bagging is an algorithm that samples from the data to create multiple subsets of the data. This technique then trains each tree on a different subset. This process ensures decorrelation since the algorithm trains each tree using a different part of the data. Afterwards, this algorithm aggregates the predictions of all models together, which is the aggregation part of bagging.
- Feature randomness: similarly to bagging, feature randomness takes a subset of all the features (instead of the data set). Again, this process ensures decorrelation since we train the trees on different features.

Gradient boosting The main difference between the final XGB model and random forests is that XGB utilises a boosting method. Boosting attempts to improve random forests by creating weak decision trees sequentially so that each new tree improves on the previous tree. The algorithm evaluates how well each tree performs using a loss function, and the goal is to have each new tree minimise that loss function further to improve the model's predictions. The most common loss function used for XGB models is cross-entropy loss. We can see the formula for it below.

$$L_{log}(y, p) = -(ylog(p)) + (1 - y)log(1 - p),, \tag{2.1}$$

where $L_{log}(y, p)$ is the cross-entropy loss, y is the actual label, and p is our prediction.

2.2.3 Deep learning classifier

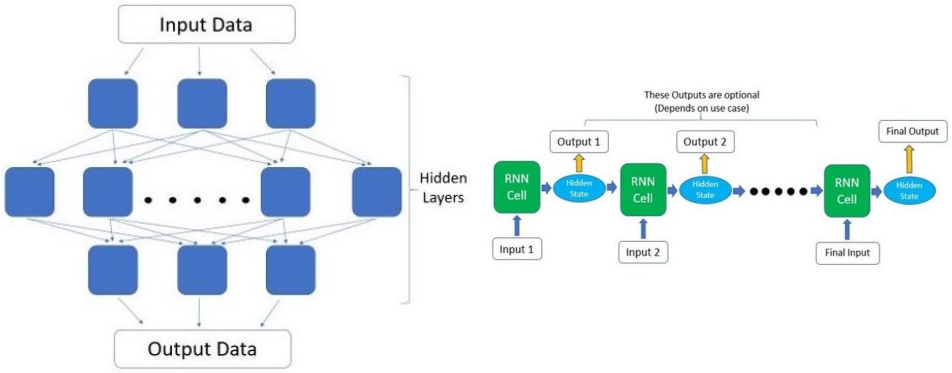
The deep learning classifier we used is long short-term memory. It is a deep neural network built from the traditional recurrent neural network architecture. Therefore, to explain LSTM's architecture, we must first discuss neural networks, then recurrent neural networks and finally how LSTM improves on the recurrent network architecture.

Neural networks Neural networks transform an input into output by passing the input through hidden layers. These hidden layers are composed of neurons, and each neuron contains its weight and bias and, when given an input, produces an output. This output is calculated using the neuron's weight and bias and its received input. The output produced is then usually passed to an activation function. The activation function is a threshold function that decides whether a neuron will be activated or not. If the neuron is activated, it transmits data to the subsequent layers. The final layer in a neural network is the output layer, resulting in the network's prediction. This process is called forward-propagation. Finally, the network adjusts its weights and biases to minimise the number of labels it predicts incorrectly by looking at the output received. This process is called backpropagation.

Recurrent neural networks Recurrent neural networks operate similarly to neural networks, but in addition, they reuse every output created by the neural network with each new input. Therefore, we will have more than one input in future iterations since we use a new input alongside our previous output. The diagram below 2.3 gives a direct comparison between the traditional neural network and the recurrent one.

LSTM One issue in the recurrent neural network architecture is that it does not store the information of large sequences well. As a result, the input of many sequences after enough iterations leads to many variables' multiplications, resulting in the loss of information. We call this phenomenon the vanishing gradient when the input becomes too small due to the multiplication of small variables. Otherwise, we call it the exploding gradient when the input becomes too large due to large due to the multiplication of large variables.

LSTM fixes the issue of the vanishing/exploding gradient problem since it does not automatically input all the features into the neural network. Instead, it selects which information to forget and which to output to the network. Thus, it does not require the multiplication of all inputs together.



(a) *Traditional Neural Network.* ***(b)*** *Recurrent Neural Network.*

Figure 2.3: *Direct comparison between two architectures. (a) shows a traditional neural network. (b) shows a recurrent neural network. Adapted from floydhub (2019).*

2.3 Machine learning explainability

This section will discuss the background of the two model interpretation algorithms (SHAP and ELI5) implemented in this project that all three of our explainers (LIME, SHAP, ELI5) utilise. Afterwards, we discuss additional potential techniques for explaining natural language processing classifiers and why they were unsuitable for this project.

2.3.1 LIME

LIME is an algorithm that explains the prediction of classifiers by approximating them locally with an interpretable model (Ribeiro et al. 2016). It creates these explanations by removing certain keywords and observing the impact on the output. To understand LIME further, we can look at what the word LIME stands for:

- Local: LIME's output must accurately represent how the model behaves locally in the predicted instance's proximity. Furthermore, this does not imply global fidelity (Ribeiro et al. 2016).
- Interpretable: Simple enough that any user can understand its output
- Model-agnostic: LIME treats the model as a black box, and therefore, it does not make any predictions about what the model is. This disregard ensures that LIME can predict all different types of models.
- Explanations: providing explanations by displaying textual or visual artefacts to the user.

2.3.2 SHAP

SHAP is a unified framework that interprets predictions by assigning each of their features an importance value (Lundberg and Lee 2017). Similarly to LIME, it is model-agnostic and to further understand it, we can look at what SHAP stands for:

- SHapley: shapley values are a solution concept in game theory. It was named after Lloyd Shapley, who introduced this concept (Shapley 1953). Shapley values display the expected contribution of each feature in the classification and whether it contributed positively or negatively (made the document more or less sensitive),
- Additive: the addition of all the feature contributions will result in the final prediction score of the model.
- exPlanations: providing explanations by displaying textual or visual artefacts to the user.

2.3.3 Yellowbrick

Yellowbrick is an open-source visual steering tool (Bengfort and Bilbro 2019). This tool provides visual interpretations for machine learning models' predictions. Yellowbrick is slightly different to the previous two explainers discussed since it is a library, not an algorithm. Nevertheless, this tool can create many explanations and visualisations for a given classifier; we can view the complete list of visualisations in the package's documentation [1]. Yellowbrick provides many useful visualisations and explanations; however, it also has some downsides:

- Yellowbrick can only interpret scikit-learn compatible models. This limitation is problematic because only our logistic regression model is a sci-kit learn model. Therefore, our system would need to significantly modify our other two models to make them compatible.
- The library appears to be more targeted toward software engineers and data scientists than regular users. The visualisations it produces are helpful but have a learning curve; a user must have a good machine learning background knowledge to comprehend the visualisations.

We concluded that it is wiser to not utilise this library due to the significant downsides it contains.

2.3.4 Skater

Skater (2017) is a unified framework that enables model interpretation for all forms of models. The authors designed it to learn the structures of black-box models both globally and locally. We discuss how the library generates its explanations below.

- Skater uses partial dependence and feature importance to interpret classifiers globally. However, using these two techniques, the library's generated plots do not give as much information as the SHAP plots. Furthermore, skater's plots are also harder to understand for users with no background knowledge.
- Skater uses lime and integrated gradient algorithms to interpret classifiers locally. However, the lime technique is redundant since we already utilise lime directly from the lime library. Furthermore, the skater's integrated gradient technique can only be used for deep neural networks (DNNs).

Since two of our models are not DNNs, the library's local interpretability will not give many benefits. Furthermore, skater does not seem to add much value to global interpretations. And Lastly, skater is still in beta phase. Therefore, because skater will not give many benefits to our project, and our system already has three explainers that are more sophisticated, we decided not to implement this library.

2.4 Summary

This chapter reviewed related work that helped this project and how our paper builds from them. Furthermore, we looked at the logic behind our three supervised classifiers. Finally, we looked at different options for machine learning explainability in natural language processing. In the following chapter, we will discuss the project's architecture alongside some necessary tools to implement the algorithms we had just discussed.

[1] `https://www.scikit-yb.org/en/latest/index.html`

3 Requirements

This chapter will introduce user stories, functional requirements and non-functional requirements. At the start of the project, we determined these requirements and continued developing them as we implemented new features and the project advanced. Furthermore, this chapter will show the project's wireframe.

3.1 User Stories

In agile software development, creating user stories is a common practice to better understand what users want from our system. Furthermore, user stories allow us to describe a software feature from a user's perspective; this creates a simplified explanation of our requirement.

In discussion with the client (project supervisor), we discussed several scenarios in which our prospective users would use our application. Based on these conversations, we came up with these user stores:

- *As* Tom, *I want to* know a document's sensitivity, *so that* I can quickly label the document.
- *As* Alice, *I want to* understand why a classifier made its prediction, *so that* I can trust the classifier's decision.
- *As* Bob, *I want to* see multiple predictions on a given document, *so that* I can be more confident in the document's sensitivity
- *As* Jennifer, *I want to* be able to use the website online, *so that* I can use the website from anywhere in the world.
- *As* Jack, *I want to* see the classifier's accuracy, *so that* I know how much to trust the classifier.
- *As* Megan, *I want to* see the explanations highlighted directly on the document, *so that* I can see the context of the terms.
- *As* Alex, *I want to* see the most impactful terms on the collection level, *so that* I know what words to look out for when reviewing documents.

3.2 Functional Requirements

Since we have our user stories, we need a way to prioritise them. Therefore, we utilise the MoSCow method. This method allows us to give different importance to our requirements by splitting them into four categories:

- Must Have: These requirements are crucial for the success of this project. These are the minimal features required for the system to be usable.
- Should Have: These requirements are important for improving the user experience and making our product better, but they are not vital.
- Could Have: These requirements are desirable but do not substantially impact the overall usefulness of our system.

- Won't Have: These requirements will not be achieved during this project because their value is not worth the time and effort required for implementing them. Nevertheless, it is helpful to record them to help clarify the project's scope.

Furthermore, we split our MoSCow requirements into two sections, functional and non-functional requirements. In this section, we discuss functional requirements. These requirements describe how the system must work and verify the software's functionality. We represent our MoSCow requirements using **MH**, **SH**, **CH**, **WH**, which corresponds to *Must Have*, *Should Have*, *Could Have* and *Won't Have* respectively. Furthermore, we initialise each requirement with a different label to track it throughout the project. This tracking helps us differentiate requirements when referring to them and clarifies where we implement each requirement in our system. We represent the functional requirements below.

- **MH1**: Users can view a classifier's prediction for a given document.
- **MH2**: Users can view an explanation for the classifier's prediction.
- **SH1**: Users can view multiple classifiers' predictions for a given document.
- **SH2**: Users can view many explanations for the classifier's prediction.
- **SH3**: Users can view each classifier's accuracy scores.
- **SH4**: Users can view an explanation for the classifier's prediction as highlighting in the document's text.
- **CH1**: Users can use the web-based app online.
- **CH2**: Users can view the most impactful terms on the collection level.
- **CH3**: Users can give feedback on the essential terms in a given document to reweigh classifiers' term feature importance value.
- **WH1**: Users can view predictions generated by a transformer-based classifier.

Transformers are deep learning models that adopt the mechanism of self-attention. They are discussed in more detail further in this paper.

3.3 Non-Functional Requirements

Non-functional requirements correspond to our system's qualities, such as performance and usability. We represent the non-functional requirements for our system below.

- **MH3**: The system must have a User-friendly interface where users can interact with the platform to change documents easily and view a different prediction and explanation for each document.
- **SH5**: The platform should have low latency when loading predictions and explanations or changing documents.
- **SH6**: The app is intuitive and easy to use without explanation.
- **SH7**: The web-based platform is visually appealing.
- **CH4**: The system is responsive, and a user can use it on both desktop and mobile devices.

3.4 Wireframe

Firstly, to successfully implement most of the requirements discussed in the previous sections, we must build the correct structure for our system, precisely, an outlook on the components needed for the website's success. For example, the number of pages, how to link them and what to include on each page. Thus, near the start of the project, we created a wireframe using a graphics editor tool called Figma. We can see the specific diagram we created in figure 3.1.

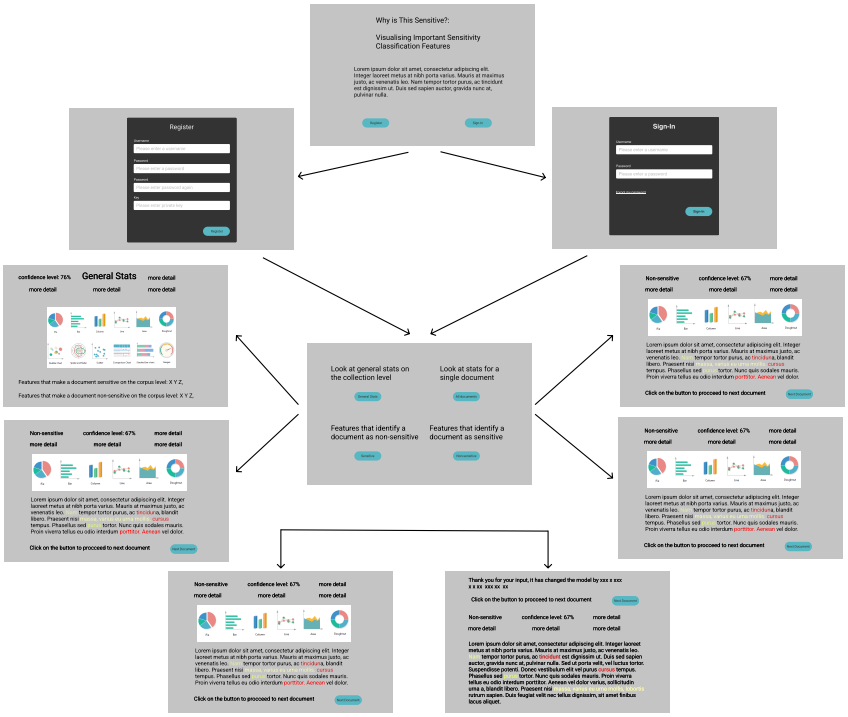


Figure 3.1: *The wireframe plans the application's structure and displays the different sections required to fulfil the user requirements.*

3.5 Summary

This chapter overviewed different user stories that our system aims to implement. Furthermore, we further improved the requirements by utilising the MoSCow method and splitting each feature into functional or non-functional. Finally, we displayed the system's wireframe. In the following chapter, we will present a system design capable of implementing these requirements.

4 Architecture

This chapter will discuss the website architecture used alongside some of the frameworks and tools we utilised to build this project successfully.

4.1 Website architecture

We can see the project's architecture diagram in figure 4.1. This diagram shows the entire workflow of our project. Our system utilises a micro web framework written in Python for its web app architecture. Specifically, it utilises the Flask framework, which does not follow the stereotypical web architecture since it does not implement the model view controller architecture as discussed in more detail in the paper from Mufid et al. (2019). The system uses Flask because when comparing it versus Django (the top competitor), Flask seemed like a better choice for the type of web application needed for this project because Flask is a better choice for deploying a small-scale website with machine learning models.

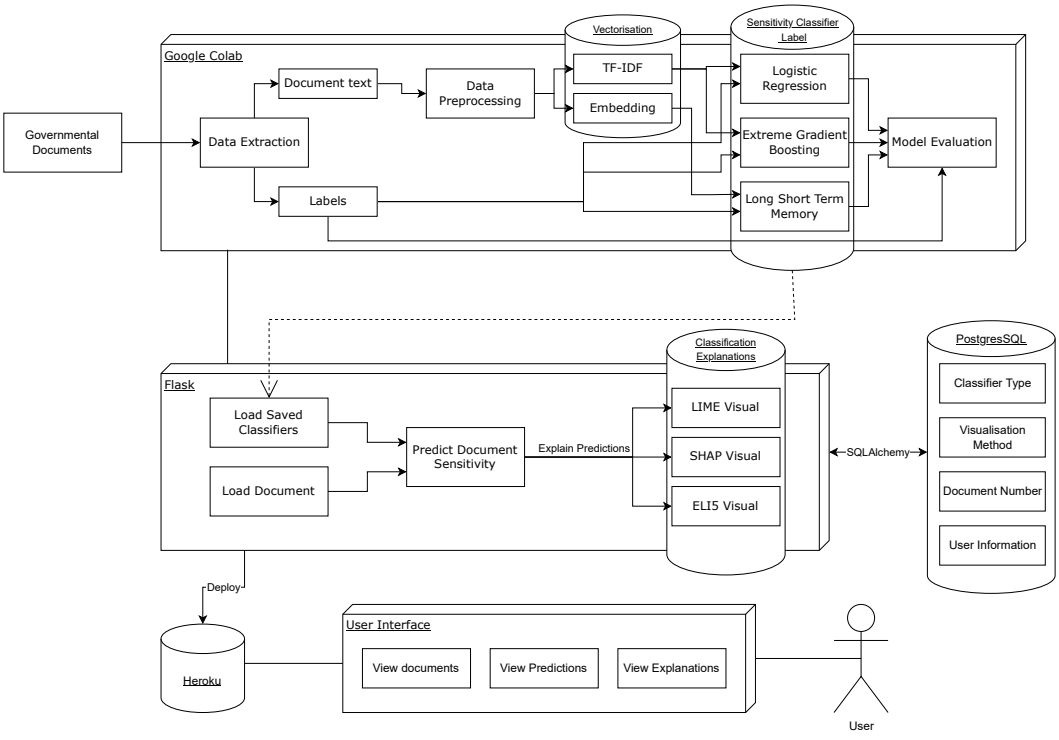


Figure 4.1: *Architecture diagram showing the complete workflow of the project. This diagram displays the project's main components starting from the data extraction for training the classifiers up until the deployment of the website to reach the end-users.*

Furthermore, for our end to end website, a database is needed. Therefore, our system utilised the SQLAlchemy library to connect a database with our project to facilitate the communication between a database to our Flask program. The application uses this library as an Object Relational Mapper (ORM) to connect python code with SQL statements. The specific database set up in our project is PostgreSQL. PostgreSQL is a suitable database because it is a reliable, fast, open-source database that supports ACID properties (Atomicity, Consistency, Isolation and Durability. Furthermore, we deployed our system on a cloud platform called Heroku, and this platform uses PostgreSQL as its primary database. We chose Heroku as our hosting platform because it is a known and reliable hosting platform for Flask web applications. Furthermore, Flask, in its documentation, directly recommends Heroku as an excellent hosting option (Flask 2010). Hosting the application fulfils the **CH1** requirement.

We are utilising Flask to create a RESTFUL API that users can interact with to view different governmental documents and classifiers. In addition, we can view the source code outline of our project in appendices A.

4.2 Tools utilised

Version control is also crucial in every software project due to its significant benefits. It allows programmers to keep track of their commits and version histories. Version control also allows them to create multiple branches that facilitate working on individual components. Moreover, version control is critical for teamwork and code merging. Therefore, we utilised GitHub as our version control system in this project.

Since training classifiers is slow and computationally expensive, we used google Colaboratory for our model training and evaluation and the creation of explanations since it provides additional computing resources, including GPUs (Colab 2017).

4.3 Summary

This chapter overviewed the system architecture and discussed its workflow. Furthermore, we discussed some of the essential tools utilised in this project. In the following chapter, we will discuss how our system implements all the different components discussed in this chapter using the architecture and tools mentioned.

5 Implementation

In this chapter, we discuss how our system implements each step in the architecture diagram 4.1 in detail.

5.1 Data extraction

The first step in the workflow is extracting information from the files we received at the project's start. The files received contained 3801 governmental documents in HTML format and a text file containing the governmental documents' ground truths (labels).

Afterwards, the system extracted each document's text by taking all of the data within the <body> tags of the HTML files. The platform then structured the data better by creating a new data object that contains each document's text with its corresponding label. We created this object (data frame) using pandas' library. Afterwards, we saw from our data frame that there were 502 sensitive documents and 3299 non-sensitive documents, which signified a class imbalance.

After completing the data extraction, the system saves the new data frames with a .pkl extension. This extension is standard for storing data frames. Furthermore, this allows the platform to load the data frame in the future without rerunning the data extraction procedure.

5.2 Data preprocessing

The system splits the data set into training and test sets for preprocessing. Two data sets are necessary because if the platform trains a classifier using the entire data set, it cannot evaluate it. Moreover, the system must evaluate the models because they could be overfitted to the data they were trained on and, therefore, will not perform well when predicting new unseen documents. Therefore, in this project, the platform used our training set to train the model; afterwards, it used our test set to measure how well the models performed. The training set holds 80% of the data while the test set contains the remaining 20%. This four to one training to testing ratio is a common practice in machine learning.

5.2.1 Cross-validation

Choosing 20% of the data set is difficult; it is possible to have a more biased test set depending on which documents the system chooses. Cross-validation helps fix this problem; it allows the platform to test the classifiers on the entire data set. It achieves this by creating five training and test sets, each taking a different part of the corpus. Thus, it created five different classifiers, each trained on a different 80% of the data set and tested on a different 20%. The system also stratified the splits across all five cross-validations. Meaning it trained each of the five classifiers on an equal number of sensitive documents to maintain class distribution. Using stratified cross-validation is a common practice when dealing with imbalanced data. Another benefit of cross-validation is that it allows the platform to display all 3801 documents since the system tests the entire data set. However, since we only require five documents for the user evaluation, the platform takes

one unique document and classifier from each of our five cross-validations folds to ensure that it utilises all trained classifiers.

Unfortunately, the platform is still unable to directly train the models using the documents' text (string) in the five training sets. A preprocessing step is required to transform the strings into a format that the system can input into the model. The first preprocessing step is usually tokenisation.

5.2.2 Tokenization

Tokenisation is the process of splitting a string into a list of tokens. For example, tokenising the string:

'natural language processing'

will return this list

['natural', 'language', 'processing']

The system can then input the tokenised list into a preprocessing technique that transforms the given strings into a vector of dictionaries mapping terms to feature indices. The system can then use this vector to train the models. Our system's two preprocessing techniques in this project are TF-IDF and word embedding.

5.2.3 TF-IDF

TF-IDF is a statistical vectorisation method commonly used to measure the importance of a given word in a corpus (collection of documents). TF-IDF is short for term frequency-inverse document frequency. We can further understand this vectorisation technique by looking at the underlying maths, which we can separate into two parts.

- Term frequency is the number of times a term occurs in a given document.
- Inverse document frequency is the total number of documents over the number of documents that contain a given term, then calculating the logarithm of the result. For example:

$$idf(t) = log\frac{n}{1 + df(t)}, \tag{5.1}$$

  where idf(t) is the inverse document frequency, n is the total number of documents in our corpus and df(t) is the number of documents in our corpus containing the term t.

However, in this project our app does not use this standard textbook notation, it uses a slightly different implementation. Specifically, scikit-learn's implementation (scikit learn 2007):

$$idf(t) = log\frac{1 + n}{1 + df(t)} + 1. \tag{5.2}$$

Scikit-learn is a machine learning library that provides many valuable features. Thus, our platform utilises it in many parts of this project, specifically, pre-training, logistic regression classification algorithm and model evaluation.

Finally, our application calculates the TF-IDF score by multiplying the term frequency (TF) with the inverse document frequency (IDF). The result gives more weight to rarer words, which appear less frequently. Giving more importance to rare words is helpful because it eliminates the dominance of stopwords that appear in all documents but add no value to the document's meaning.

To preprocess our data set using this technique, our system used scikit-learn's TF-IDF fit function to learn a vocabulary and IDF from our training set. Afterwards, the platform used scikit-learn's

transform function for the training and test set and transformed them into a document-term matrix using the newly created vocabulary. It used these two document-term matrices later in the project for training and evaluating our linear and tree classifiers. We call training and test sets that we transformed using a vectoriser, training features and test features, respectively.

5.2.4 Embedding

Embedding (word embedding) is another vectorisation technique that modifies a term into a numerical value. Unlike TF-IDF, it does not give a single numerical value to each term; instead, it gives them an array of numbers. Arrays that are close in vector space will be considered more similar.

These arrays are created by initialising an array of a fixed size with random values. Afterwards, the vectorisation technique iteratively modifies the values in the array to group similar words in a close vector space. Afterwards, it modifies these values by comparing different terms with their vector space. For example, if the words 'dog' and 'bark' occur in the same sentence in many of our documents, but their vector representation is unrelated, then the embedding algorithm modifies both vector representations to become closer. After enough iterations of this procedure with different terms, the algorithm returns a vector representation that groups similar words together.

Our platform uses the TensorFlow library's embedding layer within our neural network architecture to preprocess our documents into embeddings to utilise word embeddings. Our system later used these embeddings to train and evaluate the deep learning model.

Nevertheless, this word embedding implementation has downsides:

- It is unable to distinguish between words with multiple meanings. For example, the word 'bark' will have one vector representation even though 'bark' can have two implications. For example, bark has a different meaning in 'dog bark' than in 'tree bark'.
- It is corpus dependent. Our training data could have biases that the word embeddings will learn.

5.2.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a state of the art encoder that solves the downsides of traditional word embedding techniques. BERT is built from a transformer architecture and represents word meanings bidirectionally. Devlin et al. (2019) discuss BERT in more detail in their paper.

Using a pre-trained BERT model then fine-tuning it to a given task has been shown to work well. However, BERT has a limit of 512 tokens (Devlin et al. 2019) and thus, can only take as input small documents that have less than 512 tokens in total. Similarly to this project, Howe et al. (2019) also created classifiers for large legal documents, and the authors discussed that they could have potentially used BERT with their legal documents by splitting each long document into shorter segments. However, Howe et al. left this for future experimentation and did not attempt it in their paper. Nevertheless, we tried to use BERT as our vectorisation method for this project by splitting our documents into multiple smaller documents that are less than 512 tokens in total before inputting them into BERT and then combining the documents afterwards. However, this proved problematic when using our entire data set since we split the large documents into many new documents, which became too computationally expensive for Colab. Therefore, we concluded that using more than 10% of our training set was not feasible since our ram crashed due to insufficient memory. This conclusion is connected to our **WH1** requirement.

5.3 Classifiers

Our platform trained and evaluated our models using the acquired features from the previous section. Because the system utilised used cross-validation, each model (LR, XGB, LSTM) has five different classifiers. The usage of multiple classifiers, satisfies the **MH1** and **SH1** requirements.

The first classification algorithm our system implemented is logistic regression (LR). The system first imported LR from scikit-learn's linear models and then trained it by inputting our training data's TF-IDF document-term matrix. However, our data set is imbalanced since we have 3299 non-sensitive documents and only 502 sensitive ones. Therefore, our system needs to ensure that our model does not overfit and predict all documents as non-sensitive. The platform resolved this issue by giving the sensitive class more weight (importance). It modified the weight of the class by changing one of the hyperparameters, specifically the classifier's class_weight parameter, to give around seven times more weight to the sensitive class since there are seven times fewer sensitive documents in the corpus. This trained model then predicted the test data set's TF-IDF document-term matrix by inputting it into the model's prediction function.

The second classifier the platform implemented is extreme gradient boosting (XGB). Our system imported XGB from a library called XGBoost and then trained it similarly to logistic regression. It followed the same steps for training and evaluation. Moreover, the application also changed the weight parameters to ensure seven times more weight for the sensitive class.

The final classifier our system implemented is Long short-term memory (LSTM). Our platform imported the necessary algorithms and helper functions to implement LSTM from the TensorFlow library. Moreover, similarly to the previous models, our system modified the weights of the classes to ensure that the model classifies some sensitive documents correctly. However, having the sensitive class's weight seven times more than the non-sensitive class was insufficient and, therefore, our platform modified it to be fourteen times more. In addition, the system trained this model using word embeddings as input instead of a TF-IDF document-term matrix. Furthermore, the platform used a dropout layer in the network of the LSTM model to remove a different group of features for each sample. This elimination prevents the overfitting of the model because it forces the model to not rely solely on a few of its inputs.

5.3.1 Experimental setup

After training the classifiers, the platform evaluated their performance based on the following metrics:

- Accuracy: How many documents did our classifier predict correctly over all documents

$$\frac{TP + FP}{TP + FP + TN + FN}, \tag{5.3}$$

  where T stands for true, P stands for positive, F stands for false and N stands for negative.

- Recall: The classifier's ability to find all (non) sensitive samples

$$\frac{TP}{TP + FN}, \tag{5.4}$$

- Precision: The classifier's ability to not label a class (sensitive or non sensitive) incorrectly.

$$\frac{TP}{TP + FP}, \tag{5.5}$$

- F1 score: The harmonic mean of a class's (sensitive or non sensitive) mean and recall.

$$2 * \frac{precision * recall}{precision + recall}. \tag{5.6}$$

Table 5.1: *LR, XGB and LSTM models evaluated on accuracy, recall, precision and f1 scores for sensitive and non-sensitive documents. The results are taken as percentages*

		Non-Sensitive			Sensitive		
	Accuracy	Recall	Precision	F1	Recall	Precision	F1
LR	70 ± 2	70 ± 3	94 ± 0	80 ± 2	68 ± 2	26 ± 2	37 ± 1
XGB	81 ± 1	87 ± 1	91 ± 0	89 ± 1	44 ± 3	34 ± 3	39 ± 3
LSTM	68 ± 10	72 ± 15	90 ± 1	79 ± 9	46 ± 19	21 ± 3	27 ± 3

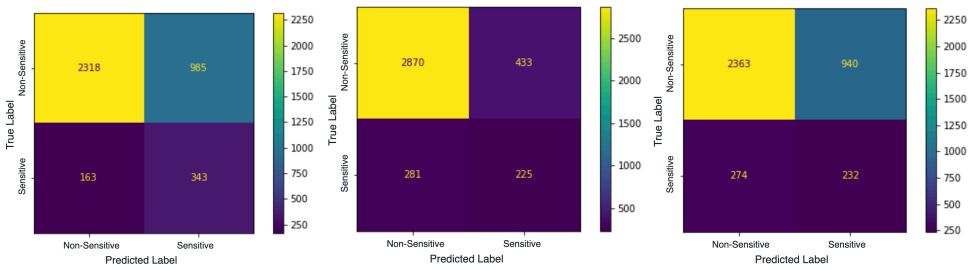
The system evaluates each classifier using cross-validation 5.2.1. Therefore, our platform tests each classifier using 3801 unique governmental documents. Of these documents, 502 are sensitive, and 3299 are non-sensitive. We discuss the results of the evaluations in the next section. We also add these results to our end-to-end platform to satisfy the **SH3** requirement.

5.3.2 Classifier effectiveness

We compare the performance of each classifier using the metrics discussed previously. We can view the results in table 5.1. We can see from the table that none of the models outperforms the other ones on all metrics. This result signifies that the models have different strengths and weaknesses. However, LSTM did not outperform on any of the metrics. Nevertheless, it also did not underperform on all of the metrics, meaning that it predicts better than other classifiers in some areas. Furthermore, we can deduce from the table that:

- XGB achieved the highest accuracy.
- XGB has the best recall for non-sensitive documents.
- LR achieved the highest precision for non-sensitive documents.
- XGB has the best F1 score for non-sensitive documents.
- LR achieved the highest recall score for sensitive documents.
- XGB has the best precision score for sensitive documents.
- XGB achieved the highest F1 score for sensitive documents.

We can further understand how well our classifiers predict documents by looking at their confusion matrices. We can view the results in figure 5.1. We can see a bias in XGB and LSTM models; they perform significantly better on non-sensitive documents than sensitive ones. This bias is less evident in our LR model.



(a) LR confusion matrix. *(b) XGB confusion matrix.* *(c) LSTM confusion matrix.*

Figure 5.1: *Confusion matrix plot displaying the true document's sensitivity versus each of the three classifiers' predicted sensitivity over all 3801 documents.*

5.4 Explainers

This section will discuss the different libraries our system uses to explain the classifiers.

5.4.1 Lime

Lime is a library based on the LIME algorithm previously discussed 2.3.1. This library can be used as an interface to quickly implement the LIME algorithm without going into the low-level details. We can further understand how the library explains our models by viewing an example from their documentation, view figure 5.2.

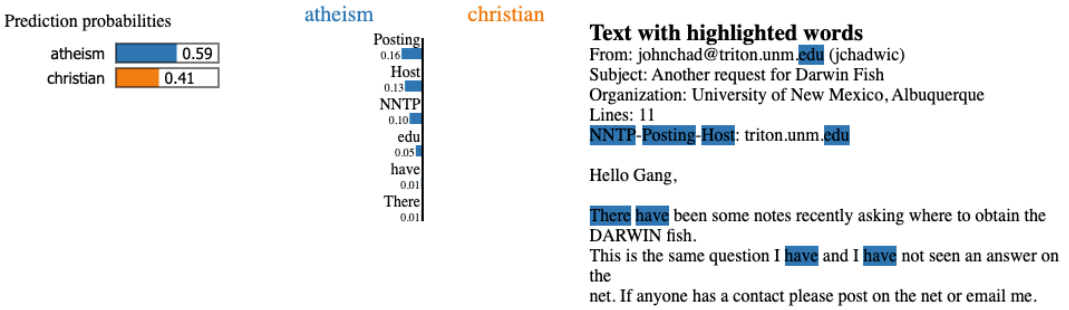


Figure 5.2: *Example model prediction explanation from lime's library. The possible predictions for this task are 'atheism' and 'christian', and the document evaluated is an email. Adapted from Ribeiro (2016).*

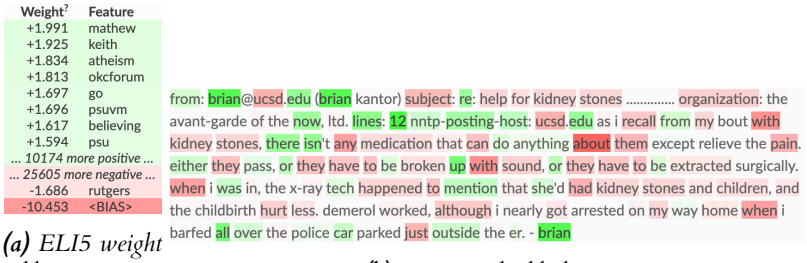
To explain our classifiers using lime, the system first imported a python class from lime called 'LimeTextExplainer'. Then, the application initialised the class with the labels (non-sensitive and sensitive). Afterwards, it used one of its methods called 'explain_instance' to create the explanations. This method takes two inputs, the document we want to explain and the classifier's predictor function. The classifier predictor function takes a variable as an input, vectorises that input then predicts that input's label as a probability. Our system then explained our classifiers by using three different classifier predictor functions. It required three because each function requires a different vectoriser and classifier to represent the model. Since lime's implementation gives our system text highlighting as seen in figure 5.2, our platform successfully completed the **MH2** and **SH4** requirements.

5.4.2 ELI5

Similarly to the lime library, the ELI5 (Explain it Like I am 5) library also implements the LIME algorithm. However, ELI5 implements it with slight modifications, resulting in different explanations. The main difference is that ELI5 uses a classifier trained using cross-entropy loss to explain predictions of probabilistic classifiers, while canonical libraries fit a regression model on the probability output. We can see more details about ELI5's implementation in the library's documentation [1]. We can further understand how the library explains our models by viewing an example from their documentation, view figure 5.3.

Our system used ELI5 to explain the classifiers, similarly to how our platform implemented the lime library. First, it created a python class from ELI5 called 'TextExplainer' that takes the document we want to predict and the system's classifier predictor function as input. Afterwards, the platform uses a method called 'show_weights' to create the weight table and a method called 'show_prediction' to create the text highlighting. Again, just like when using lime, the system had to use three different classifier predictor functions that correspond to our classifiers. Finally, because ELI5 is our second implemented explainer, our system completed the **SH2** requirement.

[1] `https://eli5.readthedocs.io/en/latest/blackbox/lime.html`



Weight[2]	Feature
+1.991	mathew
+1.925	keith
+1.834	atheism
+1.813	okcforum
+1.697	go
+1.696	psuvm
+1.617	believing
+1.594	psu
... 10174 more positive ...	
... 25605 more negative ...	
-1.686	rutgers
-10.453	<BIAS>

(a) ELI5 weight table.

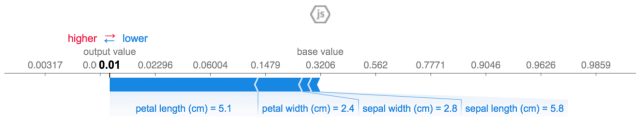
from: brian@ucsd.edu (brian kantor) subject: re: help for kidney stones ..... organization: the avant-garde of the now, ltd. lines: 12 nntp-posting-host: ucsd.edu as i recall from my bout with kidney stones, there isn't any medication that can do anything about them except relieve the pain. either they pass, or they have to be broken up with sound, or they have to be extracted surgically. when i was in, the x-ray tech happened to mention that she'd had kidney stones and children, and the childbirth hurt less. demerol worked, although i nearly got arrested on my way home when i barfed all over the police car parked just outside the er. - brian

(b) ELI5 text highlighting.

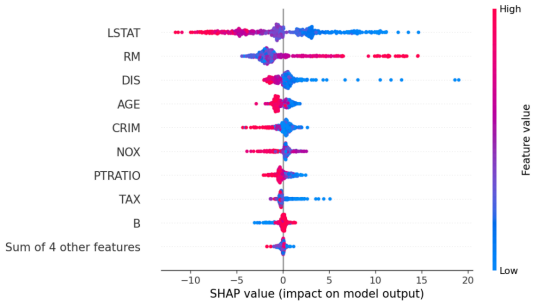
Figure 5.3: *Example model prediction explanation from ELI5's library. (a) shows a weight table that displays the terms that impacted the model's decision making the most. (b) shows the specific words in the text that influenced the model's decision making. Adapted from ELI5 (2017).*

5.4.3 SHAP

SHAP is a library that implements both LIME algorithm 2.3.1 and SHAP algorithm 2.3.2. The library also uses additional methods; we can view the full list in the library's documentation [2]. Our system implements this library to visualise information on the document and corpus levels for this project. Therefore, we completed the **CH2** requirement. We can see examples from SHAP's documentation in figure 5.4.



(a) SHAP base value plot.



(b) SHAP summary plot.

Figure 5.4: *Example model prediction explanation from SHAP's library. (a) shows the features contributing to pushing the model's output from the base value (the average model output over the training data set). (b) shows the top features that impact the model's output on the corpus level. Adapted from Lundberg (2022).*

SHAP implementation is slightly different from the previous implementations. Our system first creates explanations for our logistic regression model using a python class from SHAP called 'Explainer' that takes our model, training features, and vectoriser feature names as input. Afterwards, the system passes the test features in this newly created class. Finally, it uses the resulting output to create our visualisations. The platform creates explanations on the document level using the base value plot by calling SHAP's 'plots.force'. The system also explains classifiers on the collection level using the summary plot by calling 'plots.beeswarm' and inputting our 'Explainer' class in both.

[2] `https://github.com/slundberg/shap`

Our system created explanations for the extreme gradient boosting model in a similar way. It used SHAP's 'TreeExplainer' class instead of 'Explainer' and did not pass the training features in the class, and again the platform passed the test features in this python class. When calling the 'plots.force' SHAP method, the system passed three parameters: the average of the classifier's output over the training set; the 'TreeExplainer' class; and vectoriser feature names. Our system created the summary plot with identical syntax to the creation of LR's plot, except using 'TreeExplainer' instead of 'Explainer'.

As for our neural network model, our application initialised a python class from SHAP called 'DeepExplainer' that took the model and training data as input. It then created its base value plot visualisation the same way it did for XGB. However, for the summary plot, the system was unable to use the beeswarm method with the deep learning model due to an internal issue in the library (Newton 2020). Nevertheless, one of the solutions suggested using SHAP's 'summary_plot' method, which outputs the same summary graph but takes in different inputs. Therefore, the platform inputted into this method the required inputs: the average of the classifier's predictions over the training set, the test features and the vectoriser feature names.

5.5 End to end website

So far, our system has implemented everything solely on google Colab. Therefore, it was impossible to display any of the information we created to the end-user. Therefore, we set up a locally hosted website using the Flask framework to solve this. One of the initial steps was installing and importing all libraries that our classifiers utilise into our system. To visually see where we are in the project workflow, we can view this diagram again 4.1.

The visualisations our system created using lime and ELI5 look good when viewed on google Colab; however, when implemented in our application, the explanations did not fit well with the other components because of how rigid the visualisations were. Furthermore, lime's library does not offer any high-level functions that allow us to split the three visualisations into separate components. Therefore, we overcame this problem by modifying the library's internal codebase and manually splitting the three visualisations seen in 5.2 into components that we can move separately. Moreover, to ensure consistency, we modified lime's text highlighting to have the same styling and shape as ELI5's text highlighting. Similarly to lime, we also changed the structure of ELI5's visualisations to suit our web application.

5.5.1 Pickling models

Our system relies on classifiers to perform predictions. However, setting up and training the classifiers each time is time-consuming and inefficient. Therefore, the system pickled (stored) all classifiers and saved them inside the Flask website. The system can then load these pickled models rapidly and utilise them to predict document sensitivities.

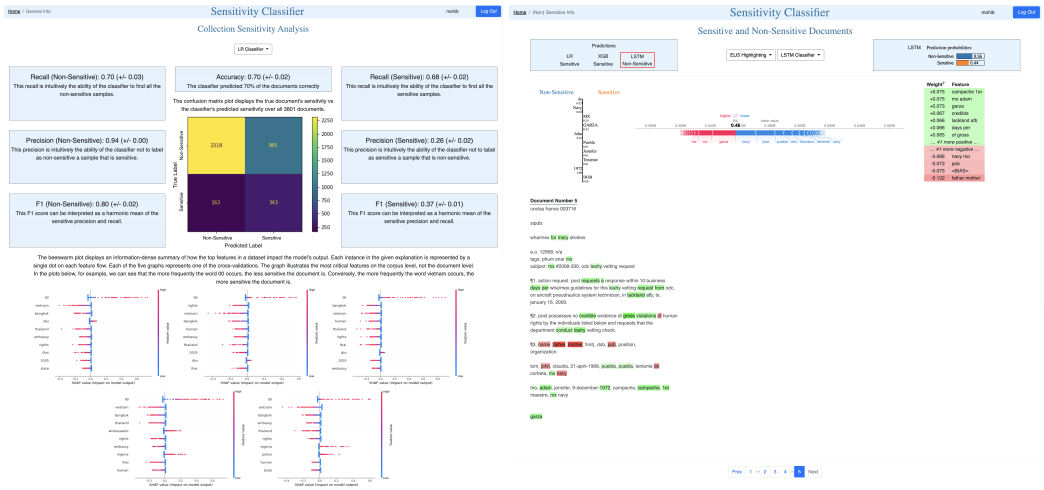
5.5.2 User interface

It is not sufficient to have the models saved on our website; the user must be able to access the classifiers and interact with them. Therefore, our system contains a user interface to allow the user to view documents, classifiers' predictions and the explanations made for each of them. Furthermore, the website allows the user to toggle classifiers; this results in all explanations and visualisations altering accordingly. Moreover, the user can toggle text highlighting; this allows the user to view a particular library's text highlighting of the essential features.

Figure 5.5 shows the primary two pages in our system. The page on the left, the collection sensitivity page, gives users more information about each classifier. This information includes a confusion matrix and metrics to evaluate our classifiers like accuracy, recall, precision and f1 score.

Furthermore, there are five plots at the bottom of the left page. SHAP generated these plots to visualise the most critical features that impacted our classifier's prediction on the collection level. Furthermore, there are five plots because our system performed cross-validation and generated five classifiers for each of our classifier types. These graphs also show that classifiers in different folds do not generate identical graphs. This contrast is because our system trained the classifiers using different documents. Moreover, the user can use the drop-down menu at the top of the page to change which classifier to view the metrics and SHAP plots for.

The page on the right in figure 5.5, the documents explanations page, allows the user to view explanations on the document level. In the top left of this page, the user can view each classifier's prediction of a document's sensitivity. If a classifier makes a different prediction from the other two models, the system highlights it with a red box. There are two drop-down menus on the top centre of the page that the user can modify. The first one corresponds to which text highlighting the website should use, and the second corresponds to which classifier to see explanations for. Furthermore, the website displays the user's chosen classifier's prediction probability on the top right. In other words, how confident the classifier is with its prediction. Below this, the application displays ELI5's weight table to show the terms that impacted the classifier the most. The red highlighted terms made the classifier think the document is more sensitive, and inversely, the green highlighted terms made the classifier think the document is less sensitive. On the left of this table, the system shows the SHAP base value plot; it acts similarly to ELI5's table since it shows the terms that affect the model into thinking a document is sensitive or non-sensitive. With terms in red being more sensitive and terms in blue being less sensitive. On the left of this plot, the website displays the lime plot; again, this works the same way as the previous two visualisations, with the orange colour meaning more sensitive and the blue colour meaning less sensitive. At the bottom of this plot, the user can view the governmental document. Moreover, as discussed previously, the user can change the text highlighting method using the drop-down menus to see different highlights on the text. Finally, the application provides document navigation at the bottom of the page. The user can utilise this to change documents, which will result in the generation of entirely new predictions and explanations.



(a) *Collection sensitivity page.* ***(b)*** *Documents explanation page.*

Figure 5.5: *Main two pages in the website. (a) displays statistical information about how well each classifier is performing. (b) shows all classifier predictions alongside explanations for these classifications.*

To ensure a good user experience design that is visually appealing, our system utilised Bootstrap, an open-source CSS framework that facilitates the creation of responsive front-end components.

5.5.3 Database

As discussed in the previous section, we want the user to be able to change documents, classifiers and text highlightings. Therefore, a database is necessary to allow these functionalities. Firstly, we created a single table in our PostgreSQL database that holds each user's unique id, username, password, document number, classifier and highlighting method. It was unnecessary to create multiple tables since having one table is sufficient for our task and simplifies the structure of our database. The website requires the username and password for login, the document number tracks which document the user is on, the classifier tracks which model the user wants to view and the highlighting method tracks which text highlighting the user wants to see. We set the primary key to be the id, which will hold a different value for each user.

Whenever the user requests to change the document or classifier or highlighting, the system updates the database accordingly. The range of possible values for 'document number' is between one and five, the possible classifiers are 'LR', 'XGB' and 'LSTM', and the highlighting methods are 'None', 'ELI5' and 'LIME'. The 'None' option displays the document without text highlighting.

5.5.4 Heroku

Heroku must first compress a copy of the application's git repository alongside the installed packages to host our website. The compilation of this compression has a specific size called a slug size. Heroku has a maximum slug size of 500 MB, meaning that Heroku will not host any project that exceeds this limit. Our project initially exceeded that size due to the installed packages; specifically, TensorFlow's package for our LSTM model alone is larger than 500 MB. A potential solution was storing our packages in an asset storage like AWS. However, this was not optimal since it would add a layer of complexity to the project since we would spend time setting up the storage and creating a new account. Thus, we attempted a different method; we avoided the slug size issue by preloading and saving all the tasks that utilised the larger packages and, therefore, did not need to install the packages at compilation time. This method was feasible because our user evaluation was a controlled environment where we knew which documents to show the users. Hence, it was possible to create and save explanations for these documents before the user evaluation. We can view some of the hosted website's pages in the following figure 5.5.

5.6 Summary

In this chapter, we discussed how our system implements each process in the architecture diagram (figure 4.1). We also explained the importance of each implementation and the issues that occurred when implementing them and how we overcame these problems. In the following chapter, we will present the tests created to ensure our product's robustness.

6 Testing

Testing is a crucial part of every software product; it saves software engineers time long term and ensures a robust and bug-free code base. In this chapter, we will discuss the tests implemented in this project and the requirements that they cover.

6.1 Unit testing

The project initially took a test-driven development approach. First, we wrote the tests, then implemented code that made the test cases pass and afterwards refactored the code. This method is a fundamental practice for reducing costs and improving code quality (Borle et al. 2018).

We created 54 tests for our system. These tests verified and evaluated the four main components of our project, namely, the factory, database, authentication and classifications/explanations.

Factory: The test in factory checks that we successfully created a flask application in testing mode.

Database: we have two tests set up for our database; the first test checks whether we can open the database, and the second test checks whether the database can be closed. These tests are essential to ensure that our database is constantly working.

Authentication:

- Checking that user can go from register to login page.
- Checking that the user can go from the login page to the register page.
- Registering a user with a random name and password, then checking whether the system redirects them correctly and adds them to the database.
- Checking that the correct error message shows up when the user attempts to register with no name or no password or attempts to register with an already registered username.
- Logging in a user and then checking whether the system redirects them correctly and whether it logs them into the website's session.
- Logging in a user with a username-password combination that was not registered in the database and checking whether the correct error message appears.
- Logging the user out and checking whether the application removed them from the website's session.

Classifiers:

- Testing that a non-registered user is unable to access all pages created. Conversely, the platform checks that the logged-in user can access all four pages.
- Testing that the user can view different documents and change between them. Furthermore, our system tests the extremes of the document range, for example, when the user attempts to access the first or the last available document and ensuring that the website responds appropriately MH3.
- Checking that the user can view a classifier's prediction MH1 and also able to change which classifiers SH1.

- Checking that the user can view a classifier's explanation MH2 and also able to change explanations SH2.
- Testing that the user can view text highlighting for explaining the classifiers SH4.

Having many tests for our system ensures that we cover a large portion of our codebase. Throughout most of this project, we maintained a high codebase coverage to ensure a bug-free system. Furthermore, our unit tests contain many of our functional requirements. Our system passes all 54 tests that we just discussed with a 100% success rate, as seen in this image 6.1.

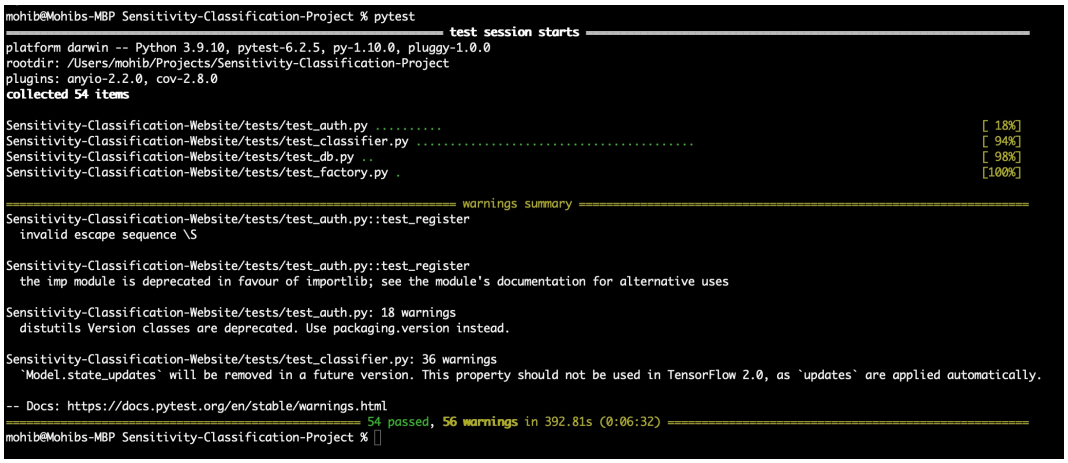


Figure 6.1: *Screenshot taken from our terminal displaying the 54 unit tests created passing.*

After having all of these tests set up that ensured our website was robust and reliable, we ran the user study. We discuss the specific details of the evaluation in the next chapter.

7 User study

This chapter will explain the experimental design and then discuss the results attained from the user evaluation.

7.1 Experimental design

As discussed in the background chapter (2), to evaluate our classifiers and explainers and see how useful they are, we must ask those who are likely to be using them. Therefore, in this section, we discuss the user study that we ran on fourteen users to get feedback about our end to end platform, our classifiers and the explanations made by the libraries we used. In addition, we can view the ethics checklist in appendices B.

7.1.1 Governmental documents

As previously discussed in the cross-validation section 5.2.1; our system had to create five different classifiers for each classifier algorithm to test on all documents. Therefore, to ensure that we evaluate each classifier at least once, we need five documents minimum, each having a different classifier predict its sensitivity. However, due to how time-consuming it would be to have more than five documents in the user evaluation, we ran the user evaluation on five documents only. These documents are a mixture of sensitive non-sensitive. The exact ratio is two sensitive documents and three non-sensitive documents. The second and fifth documents' ground truth is sensitive, while the rest are non-sensitive.

However, due to the sensitive nature of this project, we must protect the information within the sensitive documents before we allow the participants to view them. Therefore, we modified the names of all countries and individuals mentioned in these documents.

7.1.2 Questionnaire

We created a total of twelve unique questions for the user study. Eight of which are questions on the document level and are thus, repeated five times. Seven of the twelve questions are quantitative, while the other five are qualitative.

We asked quantitative questions because these questions are helpful for comparison and seeing what works. However, they do not give us concrete advice on the user experience and what specific things to modify within our platform. Therefore, it was also essential to ask the users qualitative questions. We can view the complete list of questions that we asked the users in our appendices C

7.2 User study results

This section will discuss the user study results and show some plots and graphs to better represent and understand the data.

7.2.1 Documents labeled correctly by participants

We can view the overall predictions and how well the users performed by looking at this diagram, figure 7.1. The diagram shows that users were more likely to classify a given document as sensitive. Users, on average, classified that all documents were more likely to be sensitive except document number four. Furthermore, the ground truth is that documents numbers two and five are the only sensitive documents; thus, users mislabeled many documents. Specifically, users classified 47 documents incorrectly; therefore, they only classified 23 correctly. Hence, the average user's accuracy is 33%.

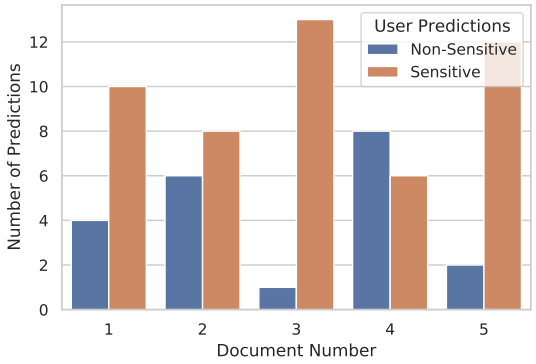


Figure 7.1: *Bar chart showing user predictions for the documents' sensitivities.*

7.2.2 Text highlighting preference

There were eleven responses of users changing their prediction due to text highlighting, which is around 16% of the time. Furthermore, there were nine unique users from these responses, which means that three out of the fourteen members evaluated did not change their prediction based on the text highlighting.

Users also chose which text highlighting was the most helpful in making their decision. The results can be viewed in this figure 7.2. There is a clear preference for ELI5. Users chose ELI5 54 times and lime 16 times, meaning that users chose ELI5 77% of the time.

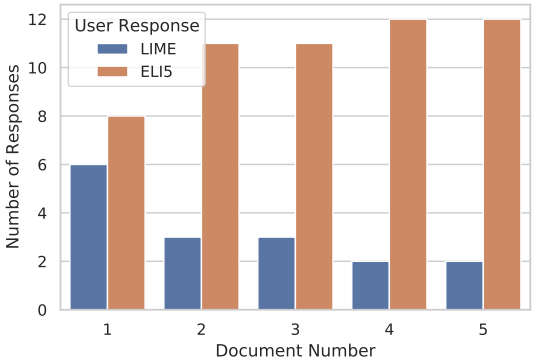


Figure 7.2: *Bar chart showing user responses for which text highlighting library was the most useful.*

Furthermore, we asked users whether they found the text highlighting useful at the end of every document. The results can be viewed in figure 7.3. The graph shows a pattern where users find text highlighting more useful with every new document they view, except documents 1 to 2. Overall, users found text highlighting useful 67% of the time.

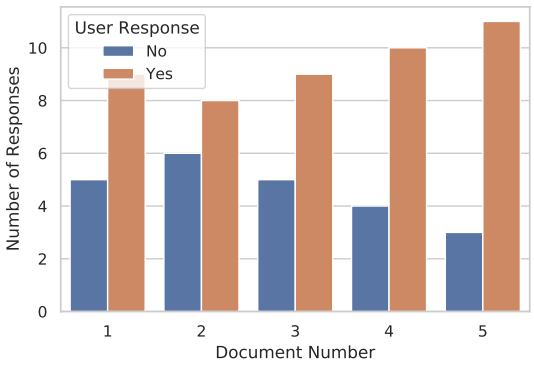


Figure 7.3: *Bar chart showing user responses for whether they found the text highlighting useful.*

7.2.3 Visualisations preference

We also tracked which visualisation method users found most helpful in making their decision. We can view the results in this figure 7.4. Again, there is a clear preference for ELI5 in all documents except the first one. Users chose ELI5 34 times (49%), lime 22 times (31%) and SHAP 14 times (20%).

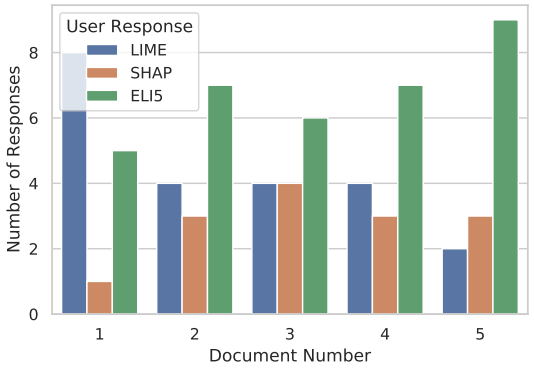


Figure 7.4: *Bar chart showing user responses for which visualisation they found most useful over all documents.*

In addition, we asked users whether they found the visualisations helpful at the end of every document. We can view the results in figure 7.5. Similarly to text highlighting, most of the responses claim that the visualisations aided them. However, for the visualisations, there does not appear to be an improvement the more documents the users view; it seems mostly constant. Users found the visualisations helpful 69% of the time.

7.2.4 Classifiers preference

For comparing classifiers, we asked users which classifier they found most helpful in making their predictions. We can view the results in figure 7.6. Users chose logistic regression the most with 31 votes (44%); they also chose LSTM 22 times (32%) and XGB 17 times (24%).

7.2.5 Collection Analysis usefulness

For our last quantitative data, we asked users whether the collection sensitivity analysis page (photo showing page 5.5) aided them. We can view the results in this figure 7.7. Again, we can

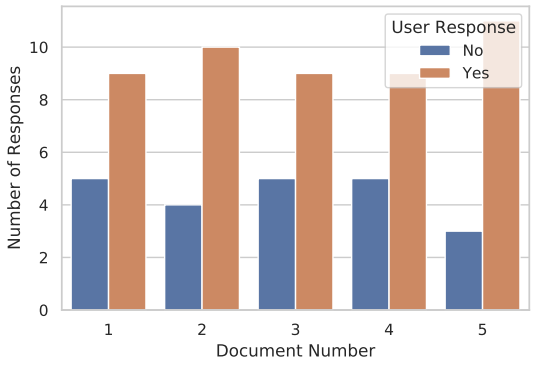


Figure 7.5: *Bar chart showing user responses for whether they found the visualisations helpful.*

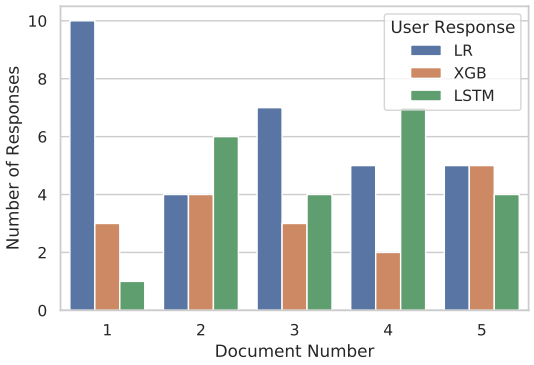


Figure 7.6: *Bar chart showing user responses for which classifier they found most useful.*

see that most users found the page helpful; specifically, 12 out of 14 participants found it helpful.

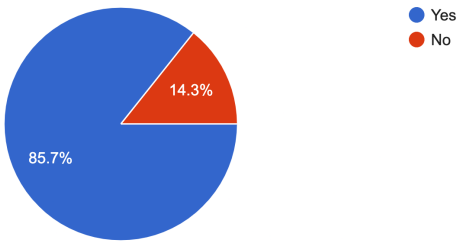


Figure 7.7: *Pie chart showing whether users found the collection sensitivity analysis page helpful.*

7.2.6 Qualitative feedback

In addition, we asked users in our study multiple qualitative questions:

- Write the top five features (words) that you found most helpful: this question is crucial because we can utilise the responses to improve and refine our classifiers. We asked this question at the end of each document; this question refers to the **CH3** requirement. From the question, we received feedback for reweighing the classifiers' term features. However, we did not complete this requirement fully since we could not utilise the terms to reweigh the models due to time constraints.
- What did you think of the website's front-end: we asked this question to see how we could further improve the visual appearance of our web app.

- Discuss whether you did not understand some parts of the website: this question helps us ensure that the platform is straightforward and easy to use.
- Discuss any additional information you would like to share: this was the final question in our questionnaire. This question is valuable because it allows the user to input information about anything that we have not asked about directly.

These questions resulted in us getting feedback on some unclear parts in the survey, unclear parts in the main website and overall feedback on the parts that require better explanation and detail. Some of the example feedback discussed that: 'confusion matrices with 0 and 1 on the axis are unclear'; 'it is sometimes difficult to know where to go next in parts of the survey'; 'it was unclear where the document starts'. We can view the complete list of user responses for all qualitative questions in our appendices D. Furthermore, these qualitative questions are helpful because they allow us to see how well our system accomplishes the non-functional requirements.

7.3 Summary

In this chapter, we discussed the experimental design, the documents chosen, and the evaluated questions. Furthermore, we summarised the results attained from the user study using multiple plots. In the following chapter, we will analyse these results in depth and draw conclusions from them.

8 Analysis

In this chapter, we evaluate the results of the previous chapter and discuss some of the findings and correlations found. Moreover, we analyse some of the problems in our system and how they could have impacted our results.

8.1 User study

Our user study contains a minuscule portion of the population, and thus, there is a high likelihood of bias. Nevertheless, there were some clear patterns and correlations that we can see from our quantitative data.

The first thing we will discuss is the accuracy of users' predictions. As discussed in 7.2.1, users, on average, achieved an accuracy of 33%. This accuracy is low considering that a dummy model that randomly predicts whether the document is sensitive will get, on average 50% accuracy. Many reasons could have caused this low accuracy, one of which is that the users have the wrong understanding of what document sensitivity is and thus are labelling based on the wrong reasons. Another plausible theory is that the users relied too heavily on the classifiers, which themselves were making the wrong predictions. We can test this by looking at how our classifiers performed against the ground truths (only the second and fifth documents are sensitive) by looking at table 8.1. Comparing this table with our bar chart that displays user predictions 7.1, we can see an overlap between user predictions and classifier predictions on all documents except the first. However, due to the users predicting exceptionally badly the first and third documents, their average accuracy was significantly reduced. Furthermore, if the users relied solely on the classifiers' predictions, considering the majority rule (taking the prediction that at least two classifiers agree on), they would have achieved 80% accuracy, with the only falsely predicted document being the third one.

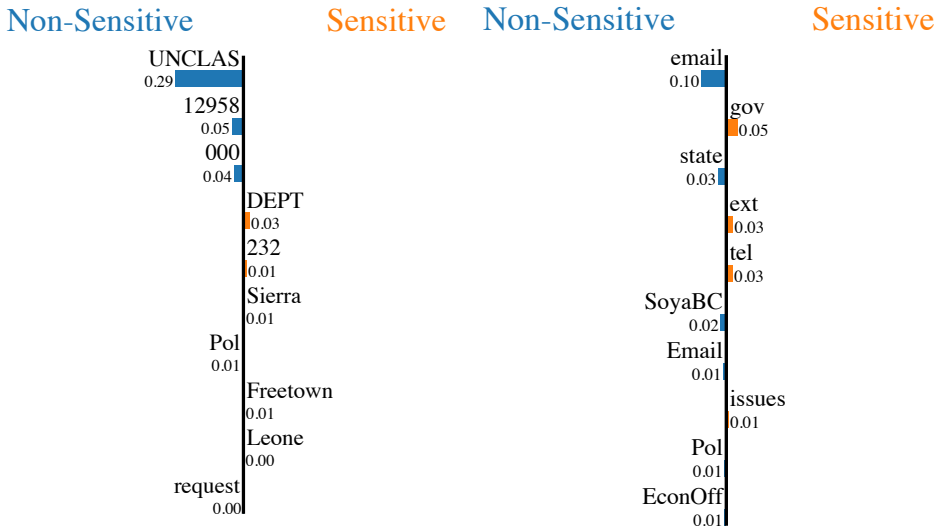
The second result we viewed was the text highlighting 7.2.2. Users, on average, found text highlighting useful, and their preferred text highlighting method is ELI5. This finding concludes that ELI5's implementation of the LIME algorithm was more valuable than lime library's implementation for text highlighting. The third result we viewed was the visualisations (explanations) 7.2.3. Again, on average, users found that the explanations were helpful, and their preferred visualisation is ELI5. This result concludes that ELI5's visual explanations aided the users in our study more than SHAP and lime's explanations. Furthermore, both lime and ELI5 scored higher

Table 8.1: *Table displaying each classifier's predictions over all documents in our user study.*

	LR	XGB	LSTM
Document 1	Sensitive	Non-Sensitive	Non-Sensitive
Document 2	Sensitive	Sensitive	Sensitive
Document 3	Sensitive	Sensitive	Non-Sensitive
Document 4	Non-Sensitive	Non-Sensitive	Non-Sensitive
Document 5	Sensitive	Sensitive	Non-Sensitive

than SHAP, and since both utilise the LIME algorithm, we can conclude that LIME's algorithm helped more users than SHAP's for our specific project. Furthermore, since ELI5 scored the highest for text highlighting and feature visualisations, it is the most helpful library in our project.

The fourth result we viewed was the classifiers 7.2.4. LR being the preferred classifier makes sense considering that it is the most well-rounded classifier since it predicts both sensitive and non-sensitive documents fairly well 5.1. Especially since it predicts sensitive documents correctly more often than the other classifiers. However, it is unexpected to see that LSTM helped more users than XGB considering that XGB outperformed LSTM on almost all metrics. This result could be due to LSTM learning features that are more logical and important to the user than XGB. By looking at one of the explanations created by LIME in figure 8.1, we can view that XGB does indeed give substantial importance to words that do not make much sense to the user, unlike LSTM.



(a) *Lime explanation of XGB prediction.* ***(b)*** *Lime explanation of LSTM prediction.*

Figure 8.1: *Comparing explanations made by lime for XGB and LSTM. (a) shows the main features that impacted XGB's prediction. (b) shows the main features that impacted LSTM's prediction.*

Interestingly, some of the users' responses were slightly contradictory. When choosing what label the document is and what classifier was the most helpful, the users often chose a classifier that predicted the document inversely to what they predicted. This phenomenon occurred twenty-three times (33%) during the user study. Thus, users found a classifier helpful in aiding their decision making due to its features even though its final prediction opposed what they thought was the correct prediction.

The fifth result we viewed was related to the collection sensitivity analysis page 7.2.5. The majority of users found this page helpful; this is logical since this page is supposed to give more information about how well each model performs and where its weaknesses are. Furthermore, the page also shows the top features that affect each classifier's predictions on the corpus level.

Finally, we looked at the qualitative results we received. The information we got from these questions was beneficial, especially the first two user evaluations. They gave detailed feedback that we were able to act upon before we undertook the rest of the user studies. We modified our website accordingly to ensure no ambiguity and that subsequent users will not experience similar problems. Furthermore, these qualitative measurements allowed us to evaluate our system to the non-function requirements initialised earlier in this paper:

- The system successfully implemented **MH3** since all users managed to change documents and view predictions and explanations for each document.
- The platform successfully implemented **SH5** because no user gave feedback about the platform being slow.
- The system implemented **SH6** since most users found the platform intuitive. However, a few users struggled with understanding the graphs. Therefore, we should improve the clarity of the project further by utilising the feedback from the user studies.
- The system successfully implemented **SH7** because the majority of the users said that the application was visually appealing.
- The platform unsuccessfully completed **CH4** since it was unresponsive, and therefore, the participants were unable to use the system on their mobile devices

Overall, our results showed that the explanations made by the libraries we utilised did indeed help most users, especially the ELI5 library. Furthermore, users chose the linear classifier as their preferred choice and the tree-based classifier as the least useful.

8.2 Classifiers

For the purposes of this project, our classifiers were sufficient. Nevertheless, there are many ways to improve them further:

- Consistency between models: Unfortunately, our models did not have identical accuracies and did not perform equally well on the test sets. This difference could have affected the comparison and made it harder to decipher whether a model was better, mainly because of its type.
- Implementing transformers: we were unsuccessful in implementing a more sophisticated model such as BERT due to ram issues 5.2.5. A possible way to overcome this issue would be to access more ram. Another solution would be to use a transformer that google created called Big Bird (Zaheer et al. 2020). This transformer for longer sequences overcomes BERT's memory issues due to its reduction of BERT's quadratic dependency to linear.
- Data augmentation: our data set is imbalanced since it has significantly more non-sensitive documents. This imbalance negatively impacted our models and led them to perform poorly when predicting sensitive documents. We can attempt to further improve our models by using a method called data augmentation. This method fixes the data imbalance issue by generating new sensitive documents that are slightly modified until we reach the same number of sensitive documents and non-sensitive documents.
- Hyperparameter tuning: another way to improve our classifiers would be to optimise their hyperparameters. We can drastically modify the classifiers and their predictions by modifying their hyperparameters. Grid search is a common approach for parameter optimisation as it allows the trial of many parameter combinations to see which perform the best.
- Getting more data: one of the best methods to improve our classifiers would be to gather more sensitive and non-sensitive documents. The more training data we have, the more features our classifiers learn and the less bias they will likely learn. Furthermore, less bias will result in the model predicting unseen real-world data more accurately.

8.3 Explainers

Similarly to our classifiers, our explainers generally helped the users, but there are still some ways to improve them further:

- SHAP missing features: in some explanations, the SHAP visualisation appears bugged; it does not show specific features; it only shows blue numbers at the bottom left of the visualisation as seen in this image 8.2. We could not find out why this happens as this does not appear to be a common issue in the library.
- ELI5 inconsistency: for only non-sensitive documents, features in ELI5's weight table have the opposite colour to its highlighting, as seen in this image 8.3. This inconsistency is problematic since the user cannot tell whether red signifies more or less sensitivity due to the inconsistency. Moreover, because we were running a user study, we had to inform the users about this issue. However, we did not inform them that this only happens to non-sensitive documents because then they can instantly tell the document's true label. Therefore, we informed the users that the colours in the weight table are inconsistent and that red does not always signify more sensitivity and vice versa.
- Unhelpful Features: for some of the explanations, users gave feedback that the terms they saw highlighted were useless and felt random. These features could be created by either the classifier learning unhelpful terms due to bias or the explanations libraries being unable to represent critical features accurately.

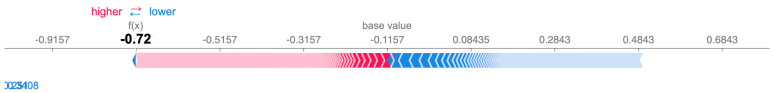


Figure 8.2: *Displaying that the SHAP plot does not show any features when explaining LR on the fourth document in our user study.*

+0.126	ms
+0.073	s 23
+0.069	august 20
+0.067	jordanian
… 66 more positive …	
… 68 more negative …	
-0.066	york
-0.067	embassy amman
-0.075	6
-0.087	on ms
-0.087	date in
-0.093	coordinate with
-0.101	department for

(a) ELI5 weight table

Document Number 3

unclas amman 003971

sipdis

fsi for m/fsi/spas/pol - ellen cosgrove
department for nea/arn - susan ziadeh

e.o. 12958: n/a
tags: aper afsi afsn pgov jo
subject: confirmation that embassy amman polfsn sireen khalifeh will attend fsi's september 16-27 political training course for fsns

ref: fsinfatc 1805

¶1. (u) post confirms that polfsn sireen khalifeh will attend fsi's september 16-27, 2002 political training course for fsns. embassy point of contact is ms. khalifeh's supervisor, political officer samuel kotis. information requested reftel para 9 is as follows:

(b) ELI5 highlighting

Figure 8.3: *Comparing ELI5's weight table and text highlighting on the third document in our user study. This table shows that the weight table and text highlighting are giving opposite colours.*

8.4 Summary

In this chapter, we evaluated the responses from the user study and concluded that ELI5 is the users' preferred explanation method, and LR is their preferred classifier. Furthermore, we discussed some imperfections in our system that could be improved. In the following chapter, we will summarise this dissertation.

9 Conclusion

This paper comparatively benchmarked the explanations created for three different types of classifiers, namely, linear-based, tree-based and deep learning-based on the classification of governmental documents. Three libraries generated these explanations: lime, SHAP and ELI5. Our user study concluded that our system's explanations successfully aided users since the participants found text highlighting and feature importance visualisations helpful 67% and 69% of the time, respectively. Moreover, users preferred ELI5 over other libraries for both of these explanations. Users also chose logistic regression (LR) as their preferred classifier over extreme gradient boosting (XGB) and long short-term memory (LSTM). Specifically, users chose LR 31 times, LSTM 22 times and XGB 17 times. Furthermore, 87% of users found that learning more detail about each classifier helped them in their classification task. In addition, our system fulfiled all of the functional and non-function requirements for MHs and SHs. However, our system does not satisfy all CHs since our platform does not implement CH3 and CH4.

9.1 Future work

During earlier chapters, we briefly discussed transformers and why they would be beneficial. Potential future work is to implement one of the transformer architectures and compare the explanations generated for it to our more traditional models.

Furthermore, SHAP and ELI5 libraries have problems that make them less user friendly; we discussed the full details in 8.3. Therefore, fixing these issues will be beneficial for future work.

Moreover, we measured our model performance on the test sets and found that the classifiers perform better on non-sensitive documents than on sensitive ones. A potential improvement is implementing hyperparameter tuning and data augmentation to improve our classifiers.

Participants in the user study also gave valuable feedback on some improvements to the system. For example, many users suggested adding additional information about the explanation graphs that show the top features. The participants suggested adding details about the diagrams inside the web-based app directly so that the user can hover on the graph and get a description of what the graph does. In addition, some participants suggested that it would be beneficial to have the ability to expand the explanations graphs so that they can see more terms instead of just the top ten. Multiple users have recommended these two suggestions. Therefore, adding them to our application would be helpful.

On March 15, LinkedIn (2022) made a blog post discussing the open-sourcing of FastTreeShap that is based off of Yang's paper (Yang 2021). The author claims that this algorithm can result in as many as 3x faster explanations. This new algorithm could prove beneficial for our project since the explanations that SHAP generates for XGB are slow.

Another future work that could be beneficial is allowing users to upload their legal documents and view the explanations created by our classifiers for these documents. This new feature would also be helpful because it would allow us to evaluate how well our models and explanations perform on new, unseen documents.

Acknowledgments

We thank Dr Graham McDonald for providing us with governmental documents and helpful advice throughout this project. We also thank the users that participated in our user study for their valuable comments: Annika Borrmann, Mario Killmann, Evicka Kupcova, Robin Sund, Abdel-Nasser Akoum, Nour Akoum, Randa Jamal, Ibrahim Akoum, Mohamad Mawlawi, William Hull, Omar Akoum, Marty O'Neill, Sonali Bhaskar, Emma Taleb.

A Source code outline

End to end website: Sensitivity Classifier Website

```
flaskr
├── model
│   ├── LRCrossValStats.pkl
│   ├── XGBCrossValStats.pkl
│   └── LSTMCrossValStats.pkl
├── static
│   └── images
├── templates
│   ├── auth
│   │   ├── register.html
│   │   └── login.html
│   ├── classifier
│   │   ├── generalSensitivitiyPage.html .4 index.html
│   │   ├── nonSensitiveInfo.html
│   │   ├── sensitiveInfo.html
│   │   └── singleDocumentSensitivity.html
│   └── base.html
├── init.py
├── auth.py
├── classifier.py
├── extensions.py
├── user.py
└── tests
    ├── testAuth.py
    ├── testClassifier.py
    ├── testDb.py
    └── testFactory.py
├── Procfile
└── wsgi.py
```

B Ethics checklist

User study we ran: User evaluation

We provide the signed user evaluation ethics checklist form below.

School of Computing Science
University of Glasgow

Ethics checklist form for assessed exercises (at all levels)

This form is only applicable for assessed exercises that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.
   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.

2. The experimental materials were paper-based, or comprised software running on standard hardware.
   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.
   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.
   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

Figure B.1: *Signed ethics checklist part 1 of 2.*

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Course and Assessment Name _____Level 4 individual project_____

Student's Name _____Mohib Akoum_____

Student Number _____2431135a_____

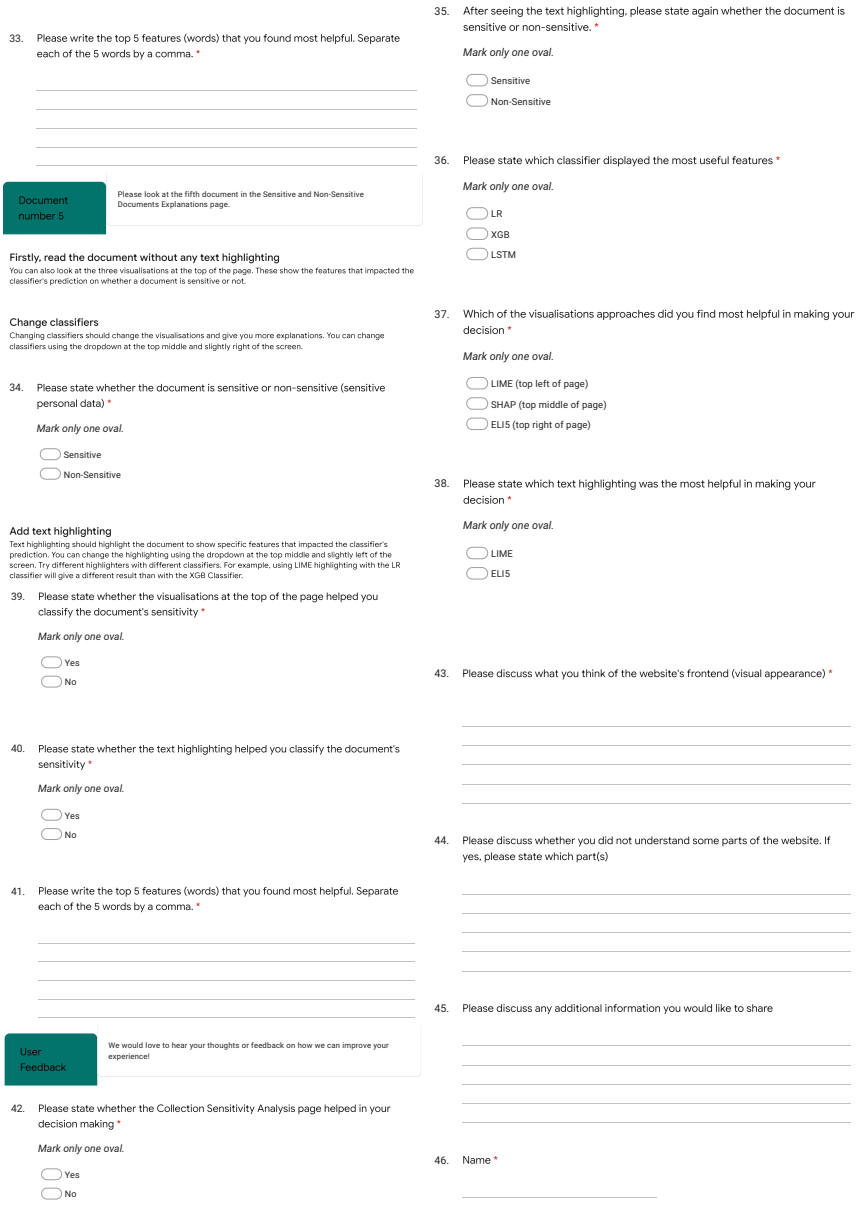
Student's Signature _____  **_____**

Date _____23/2/2022_____

Figure B.2: *Signed ethics checklist part 2 of 2.*

C | Questionnaire

This appendix contains the full questionnaire that the participants filled out.



33. Please write the top 5 features (words) that you found most helpful. Separate each of the 5 words by a comma. *

Document number 5

Please look at the fifth document in the Sensitive and Non-Sensitive Documents Explanations page.

Firstly, read the document without any text highlighting

You can also look at the three visualisations at the top of the page. These show the features that impacted the classifier's prediction on whether a document is sensitive or not.

Change classifiers

Changing classifiers should change the visualisations and give you more explanations. You can change classifiers using the dropdown at the top middle and slightly right of the screen.

34. Please state whether the document is sensitive or non-sensitive (sensitive personal data) *

Mark only one oval.

- Sensitive
- Non-Sensitive

Add text highlighting

Text highlighting should highlight the document to show specific features that impacted the classifier's prediction. You can change the highlighting using the dropdown at the top middle and slightly left of the screen. Try different highlighters with different classifiers. For example, using LIME highlighting with the LR classifier will give a different result than with the XGB Classifier.

39. Please state whether the visualisations at the top of the page helped you classify the document's sensitivity *

Mark only one oval.

- Yes
- No

40. Please state whether the text highlighting helped you classify the document's sensitivity *

Mark only one oval.

- Yes
- No

41. Please write the top 5 features (words) that you found most helpful. Separate each of the 5 words by a comma. *

User Feedback

We would love to hear your thoughts or feedback on how we can improve your experience!

42. Please state whether the Collection Sensitivity Analysis page helped in your decision making *

Mark only one oval.

- Yes
- No

35. After seeing the text highlighting, please state again whether the document is sensitive or non-sensitive. *

Mark only one oval.

- Sensitive
- Non-Sensitive

36. Please state which classifier displayed the most useful features *

Mark only one oval.

- LR
- XGB
- LSTM

37. Which of the visualisations approaches did you find most helpful in making your decision *

Mark only one oval.

- LIME (top left of page)
- SHAP (top middle of page)
- ELI5 (top right of page)

38. Please state which text highlighting was the most helpful in making your decision *

Mark only one oval.

- LIME
- ELI5

43. Please discuss what you think of the website's frontend (visual appearance) *

44. Please discuss whether you did not understand some parts of the website. If yes, please state which part(s)

45. Please discuss any additional information you would like to share

46. Name *

Figure C.1: *User study questionnaire.*

D Qualitative feedback

This appendix contains the all qualitative feedback we received from the users.

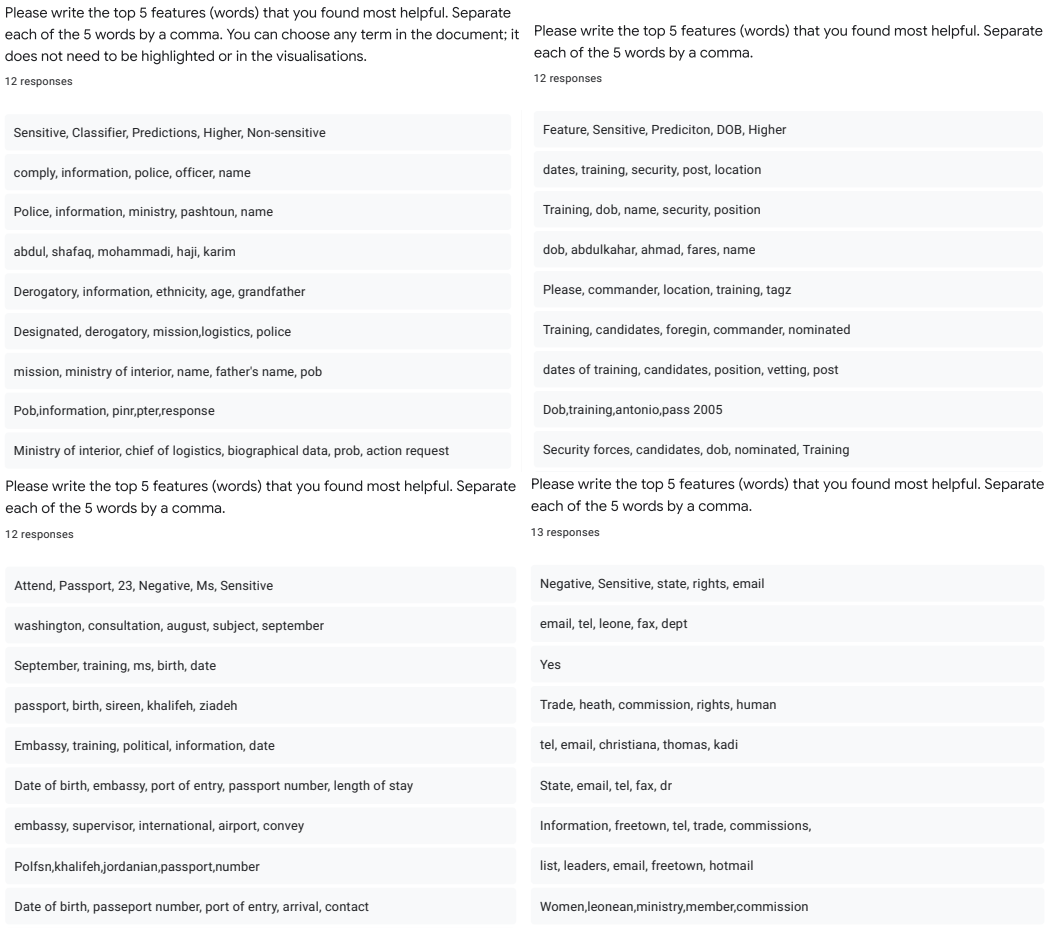


Figure D.1: *User study questionnaire part 1 of 2.*

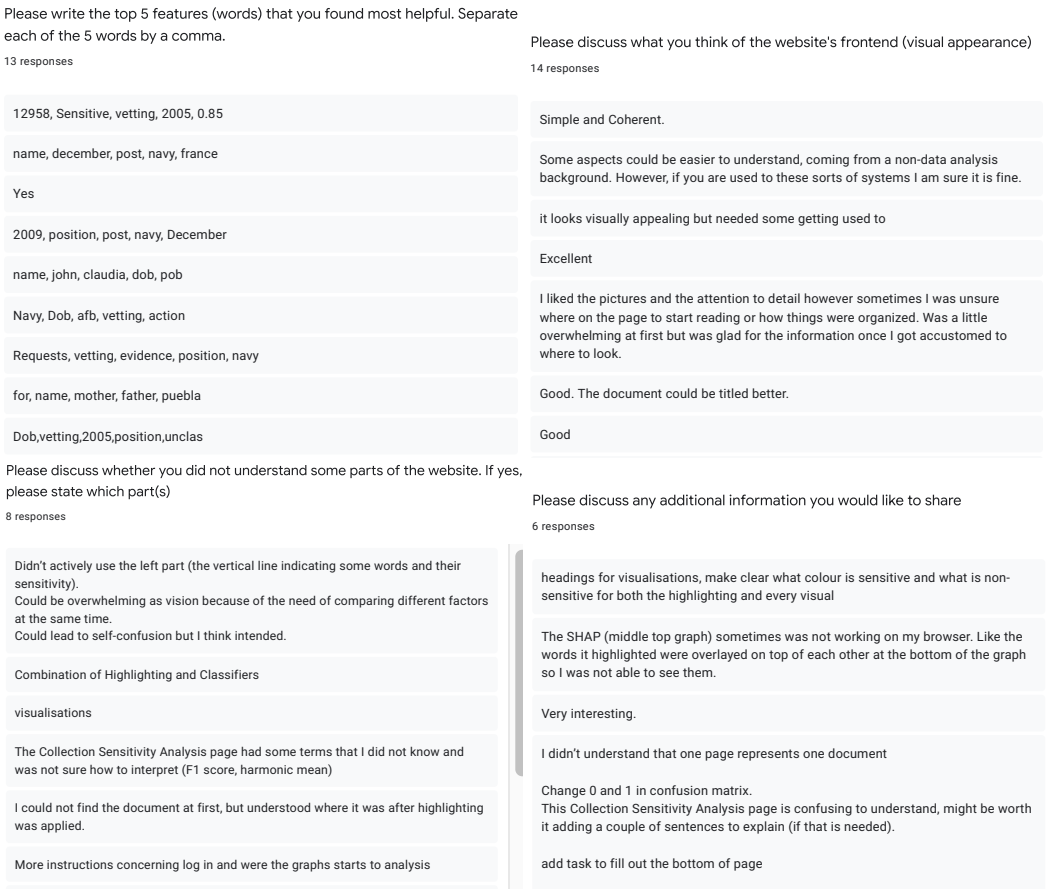


Figure D.2: *User study questionnaire part 2 of 2*

Bibliography

Bengfort, B. and Bilbro, R. (2019), 'Yellowbrick: Visualizing the Scikit-Learn Model Selection Process', **4**(35).

Bhambhoria, R., Dahan, S. and Zhu, X. (2021), 'Investigating the State-of-the-Art Performance and Explainability of Legal Judgment Prediction', *Proceedings of the Canadian Conference on Artificial Intelligence* .

Borle, N. C., Feghhi, M., Stroulia, E., Greiner, R. and Hindle, A. (2018), 'Analyzing the effects of test driven development in GitHub', *Empirical Software Engineering* **23**(4), 1931–1958.

Chen, T. and Guestrin, C. (2016), XGBoost: A Scalable Tree Boosting System, *in* 'Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, San Francisco California USA, pp. 785–794.

Colab (2017), 'Google colab faq', `https://research.google.com/colaboratory/faq.html`. Last accessed: 2022-03-31.

Danilevsky, M., Qian, K., Aharonov, R., Katsis, Y., Kawas, B. and Sen, P. (2020), 'A Survey of the State of Explainable AI for Natural Language Processing', *arXiv:2010.00711 [cs]* .

DATA (2019), 'Logistic regression', `https://datascience.eu/mathematics-statistics/logistic-regression/`. Last accessed: 2022-03-31.

Davuluri, H. (2020), 'Decision Trees Explained With a Practical Example – Towards AI — The World's Leading AI and Technology Publication'.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', *arXiv:1810.04805 [cs]* .

ELI5 (2017), 'Debugging scikit-learn text classification pipeline', `https://eli5.readthedocs.io/en/latest/tutorials/sklearn-text.html`. Last accessed: 2022-03-31.

Flask (2010), 'Deployment Options — Flask Documentation (2.0.x)', `https://flask.palletsprojects.com/en/2.0.x/deploying/`. Last accessed: 2022-03-31.

floydhub (2019), 'Beginner's Guide on Recurrent Neural Networks with PyTorch', `https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/`. Last accessed: 2022-03-31.

Howe, J. S. T., Khang, L. H. and Chai, I. E. (2019), 'Legal Area Classification: A Comparative Study of Text Classifiers on Singapore Supreme Court Judgments', *arXiv:1904.06470 [cs]* .

Javatpoint (2021), 'Logistic Regression in Machine Learning - Javatpoint', `https://www.javatpoint.com/logistic-regression-in-machine-learning`. Last accessed: 2022-03-31.

LinkedIn (2022), 'FastTreeSHAP: Accelerating SHAP value computation for trees', https://engineering.linkedin.com/blog/2022/fasttreeshap--accelerating-shap-value-computation-for-trees. Last accessed: 2022-03-31.

Liu, H., Yin, Q. and Wang, W. Y. (2019), 'Towards Explainable NLP: A Generative Explanation Framework for Text Classification', *arXiv:1811.00196 [cs]* .

Lundberg, S. (2022), 'Shap documentation', https://github.com/slundberg/shap. Last accessed: 2022-03-31.

Lundberg, S. M. and Lee, S.-I. (2017), A Unified Approach to Interpreting Model Predictions, *in* 'Advances in Neural Information Processing Systems', Vol. 30, Curran Associates, Inc.

McDonald, G., Macdonald, C. and Ounis, I. (2017), Enhancing Sensitivity Classification with Semantic Features Using Word Embeddings, *in* J. M. Jose, C. Hauff, I. S. Altıngovde, D. Song, D. Albakour, S. Watt and J. Tait, eds, 'Advances in Information Retrieval', Vol. 10193, Springer International Publishing, Cham, pp. 450–463.

McDonald, G., Macdonald, C., Ounis, I. and Gollins, T. (2014), Towards a Classifier for Digital Sensitivity Review, *in* M. de Rijke, T. Kenter, A. P. de Vries, C. Zhai, F. de Jong, K. Radinsky and K. Hofmann, eds, 'Advances in Information Retrieval', Vol. 8416, Springer International Publishing, Cham, pp. 500–506.

Mufid, M. R., Basofi, A., Al Rasyid, M. U. H., Rochimansyah, I. F. and rokhim, A. (2019), Design an MVC Model using Python for Flask Framework Development, *in* '2019 International Electronics Symposium (IES)', pp. 214–219.

Newton, A. (2020), 'Inconsistent usage of 'shap_value' in beeswarm · Issue #1460 · slundberg/shap', https://github.com/slundberg/shap/issues/1460. Last accessed: 2022-03-31.

Ribeiro, M. T. C. (2016), 'Lime', https://github.com/marcotcr/lime. Last accessed: 2022-03-31.

Ribeiro, M. T., Singh, S. and Guestrin, C. (2016), "Why Should I Trust You?": Explaining the Predictions of Any Classifier, *in* 'Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '16, Association for Computing Machinery, New York, NY, USA, pp. 1135–1144.

scikit learn (2007), 'Feature extraction', https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction. Last accessed: 2022-03-31.

Shapley, L. S. (1953), "a value for n-person games", *in* 'In: Contributions to the Theory of Games', pp. 307–317.

Skater (2017), 'Skater documentation', https://oracle.github.io/Skater/overview.html. Last accessed: 2022-03-31.

Undavia, S., Meyers, A. and Ortega, J. E. (2018), A Comparative Study of Classifying Legal Documents with Neural Networks, *in* '2018 Federated Conference on Computer Science and Information Systems (FedCSIS)', pp. 515–522.

Yang, J. (2021), 'Fast TreeSHAP: Accelerating SHAP Value Computation for Trees', *arXiv:2109.09847 [cs, stat]* .

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L. and Ahmed, A. (2020), Big bird: Transformers for longer sequences, *in* H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin, eds, 'Advances in Neural Information Processing Systems', Vol. 33, Curran Associates, Inc., pp. 17283–17297.