**CO600 Terms and Conditions Analyzer**

Adam Preston (ap720), Euel Domingo (ejd29), Kester Gideon (kg327), Marcus Robinson-Bailey(mer25)

*School of Computing*

*University of Kent*

*Canterbury, United Kingdom*

**Supervised by**: Alex Freitas ([A.A.Freitas@kent.ac.uk](mailto:A.A.Freitas@kent.ac.uk))

**Abstract**

This report describes the stages and processes involved in creating the terms and conditions analyser. The analyser is a user-friendly website that highlights any risky statements in the document. With the help of machine learning, the analyser will scan through the document inserted and compare with a trained set of words we deemed to be risky. The paper will begin by describing how the concept came about then move on to provide more information on the background and aims of the project. Further description will be given on the product development and testing of the final product. In addition to this the report we will provide a technical description of the classification algorithms used, quality assurance and challenges faced.

**1. Introduction**

There have been countless occasions where individuals sign documents or contracts with no intention of reading the terms and conditions. According to a report published by the European Commission, 66.8% of consumers had problems relating to online purchases for having insufficient knowledge of the Terms and Conditions. Furthermore, 52.7% of consumers consider their problem as serious and 57.9% did not contact or take any action against the trader due to their lack of understanding of the Terms and Conditions.[1] In relation to this current issue, we decided to create a terms and conditions analyser. The purpose of this would be to would allow users to insert their document in the form of plain text into the web interface which would then indicate the risky statements by displaying them in a modal. The risky sentences displayed allows the user to review the document before taking further action. Research was undertaken in the initial stages into different terms and conditions, machine learning API, pre-processing scripts. Following this we planned our project by setting out aims for the project and a design plan.

**2. Background**

Before any project, market research is an essential part to building a foundation on your project and gather information on your target market. Below include several companies that already exist in contract analyzing field.

**2.1 Market Research**

After conducting market research on our specific topic, we came across applications that held the same attributes as the ones we wanted to include in our software (EULAlyzer, Terms of Service Didn’t Read, Polisis). Terms of Service Didn’t Read was the one main application that mirrored features that we wanted to implement. The purpose of performing market research was to gather and formulate ideas for our university project. It also allowed us to brainstorm ideas but more importantly inspired us to make an application that first met user requirements and secondly pushed the boundaries of our data mining knowledge.

**2.1.1 Terms of Service Didn’t Read**

Terms of Service Didn’t Read [2] is our biggest competitor within the market we have chosen. This is so, because it both analyses the terms and conditions of a website and also provides browser add on compatibility. This feature would label website policies with class ratings ranging from Class A (good) – Class E (very bad) through a browser add-on. This would then inform the user about the rights they have on the specific website they’re signing up to. From Terms of Service Didn’t Read we acknowledged that there were aspects of the application that we didn’t want to implement for the reason that the feature would be very time consuming, expensive to implement and would slow down production of our main features.

**2.1.2 EULAlyzer**

EULAlyzer [3] is another software that possesses the same functionality as the one we wanted to implement. Created by Brightfort the software prides itself on identifying important elements of terms and conditions. Even though EULAlyzer wasn’t our main competition like Terms of Service Didn’t Read, it again helped with research and understanding with regards to what is essential for our project idea.

**2.1.3 Polisis**

A smaller advertised application was Polisis [4], created by an independent developer. The application visualised privacy policies using artificial intelligence. It highlights information that a website is collecting from you and possibly sharing to external agencies.

Researching a spectrum of applications, reassured us that there would be limitations in our project, and we shouldn’t label the lack of features as a constraint.

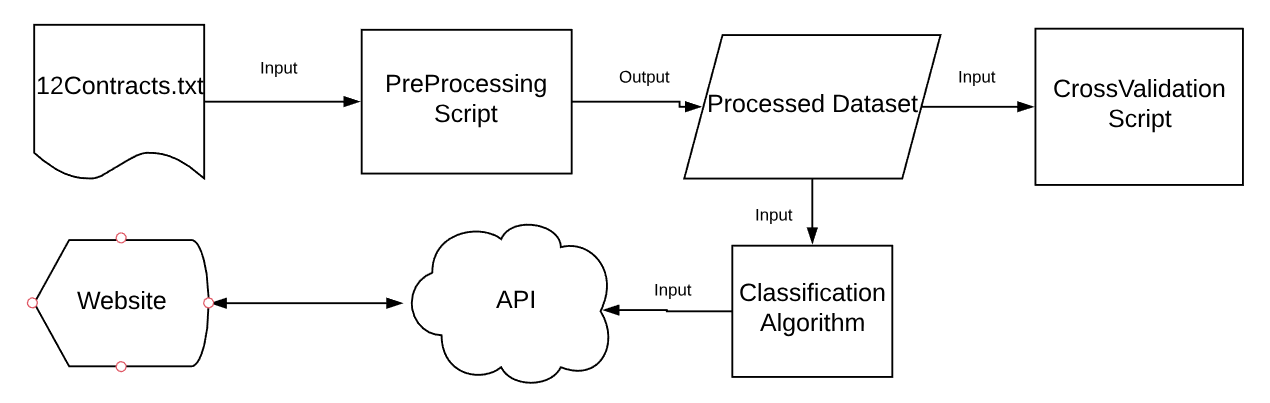
**3. Aims**

Our aim was to build a web application that will simplify the process of reviewing contracts. The application will have a parameter where it takes a copy of a contract, extract the risky statements out from the contract, and then display these statements to the user for them to review. In addition, the display will include statements where users might want to have a second look at. This will speed up the process of reading through policies and reduce the risk of users getting into problems.

Our core components are a concise dataset and a suitable classification algorithm. To build the dataset, we would need a pre-processing application that will meticulously run through several contracts. The aim is to produce high precision and recall scores which will be a good determinant for a positive user experience with our classifier.

**4. Development**

**4.1 General System Design**

We opted for a modular system design to our project. The modules are independent from each other and possess only the functionality required for the modules purpose.

The modular system meant that changes in the functionality of one module did not affect any of the other modules. This was especially useful in our project for two reasons.

Firstly, continual improvements were made to the ‘PreProcessing Script’ throughout the project in order to optimize the performance of the classification algorithm. If changes to the preprocessing script forced modifications throughout the system, then resources would have been spent invested in rework instead of optimizing the classification algorithm.

We have 4 developers in the team, therefore the modular system allowed for an effective distribution of work. Developers could work on different areas of the system with the confidence that they will not be affecting each other.

**4.2 Language and Environment**

For the purposes of our ‘PreProcessing Script’, ‘Classification Algorithm’, ‘CrossValidation Script’ and ‘API’ we chose to use Python as our programming language. There are several reasons why we chose to use this.

In terms of the system complexity, our application is relatively simple. Our application does not have to deal with threads, memory allocation issues or complex back-end server logic. Therefore, we believed that a lightweight, general purpose programming language such as Python would allow us to implement everything required but wouldn’t come at the cost of excluding group members from development because they were unfamiliar with a specific language.

Python (along with R) is quickly becoming the first-choice language for Machine Learning / Data Science enthusiasts. Python hosts a large collection of libraries that provide easy to implement and up to date machine learning algorithms. Considering the objectives of our project were focused on applying machine learning techniques rather than implementing our own, we believed Python would be a good choice for our project.

In addition, we also needed a language that would support access to our classification model through an API. We are aware that mathematical languages such as R, MATLAB and Octave do not have as many API packages available therefore we were apprehensive of choosing any of these languages for our project. A limitation of API packages means there is less fallback if issues arise with the current API package we are using. However, Python has many API packages available, providing greater security in the usability of our project.

**4.3 Data Collection**

In preparation for creating the dataset, we reviewed a variety of contracts. These came from a variety of companies. We highlighted statements that we deemed risky. By reviewing these, we were able to recognise patterns within the contracts. As we reviewed more contracts, we were able to find and mark risky statements consistently. We were able to develop a guideline in highlighting the risky statements. Thus, this improved our consistency when specifying the risky statements. After reviewing a significant number of contracts, we extended our reviewing process to include more companies.

**4.4 Preprocessing Script**

The model responsible for classifying sentences as either risky/safe needs first to be trained by a classification algorithm using a large dataset which indicates the distinction between ‘risky’ and ‘safe’ sentences. To provide the required dataset, a preprocessing script had to be developed that transformed our interpretations of risky/safe sentences into a table that could be interpreted by a classification algorithm. In addition, the preprocessing script is also responsible for manipulating the dataset in order to enhance the classification algorithms ability to generate a suitable model. To give an example, we may program the preprocessing script to deliberately exclude certain words being included as attributes. As humans we judge that attributes within a preprocessing script have no relevance to the model we wish to generate and therefore excluding them can improve both speed and accuracy results.

*See Figure A.1*

The columns (excluding ‘Is\_Risky’) represent words that the model would check the presence of, while the rows represent one sentence extracted from contracts. A cell can contain either a 1 or a 0 and indicates the presence of a word in a sentence. The ‘Is\_Risky’ column labels the sentence as either Risky (‘Yes’) or Safe (‘No’).

When examining the contracts, for each risky sentence we would include a codeword within the sentence (0cf333), as presented below.

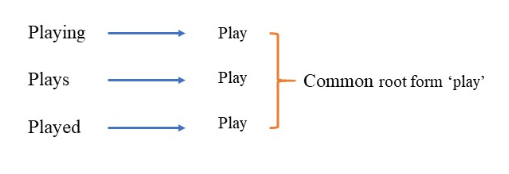
*See Figure A.2*

After the preprocessing script has used the nltk toolkit to extract the sentences from the provided documents, it looks for the codeword within each sentence. If the codeword is found, then the preprocessing script can store the corresponding sentence in a collection of risky sentences. If the codeword is not present, then the sentence is stored in a collection of safe sentences.

The preprocessing script then performs a filtering process on all words recorded when examining the documents to generate the attributes which will be included as the column headers. The script removes duplicates, punctuation, excluded words, numbers and stop words.

Stop words are words that are considered to add no extra value to a sentence/query when processed by a natural language algorithm. Examples of stop words include ‘your’, ‘that’, ‘the’, ‘when’. Non-stop words are usually keywords that help define the sentence, e.g. ‘information’, ‘payment’, ‘privacy’. Removing stop words can improve the accuracy of the model. By removing stop words, excess noise in the data is reduced and the feature space is focused on the most important attributes. However, removing certain stop words can reduce the semantic information of sentences and have a detrimental impact on the model’s performance. For example, ‘not’ is commonly included as a stop word in most stop word datasets. However, consider the following example when removing ‘not’ from the sentence: *‘This website does not share your personal information with third parties.’* Removing ‘not’ from this sentence would result in a sentence which would be classified as ‘risky’ by the classifier. Therefore, in this example, ‘not’ needs to be included as an attribute so that the classifier learns during training that ‘not’ used in conjunction with a risky sentence usually results in a safe sentence.

Stemming is a technique employed by the preprocessing script to recognize the form of the word by reducing the word to its stem. An example is given below.



This improves the accuracy of the model because it reduces the number of words that may have been disregarded/missed during training or testing. For example, the preprocessing script may record the word ‘playing’ 100 times when examining the documents, but only record ‘played’ a few times. Despite ‘playing’ and ‘played’ both possessing the same meaning, the preprocessing script might discount the word ‘played’ because it occurred too few times despite being relevant to the model.

Suppose also, that a model is trained using a dataset where ‘playing’ occurs 100 times and ‘playing’ is an indicator of a risky sentence. However, ‘played’ does not occur once during training. A user then submits a document where the word ‘played’ occurs several times. The sentences where ‘played’ is present are misclassified as safe because the model is trained on recognizing the relationship between ‘playing’ and the presence of a risky sentence. Stemming resolves these issues by classifying all forms of the word ‘play’ under the same word/attribute. After filtering and stemming has been applied to the sentences in the contracts, the risky and safe sentences are re-examined and the presence of attributes within each sentence are recorded before being exported to a csv format to be interpretable by a classification algorithm.

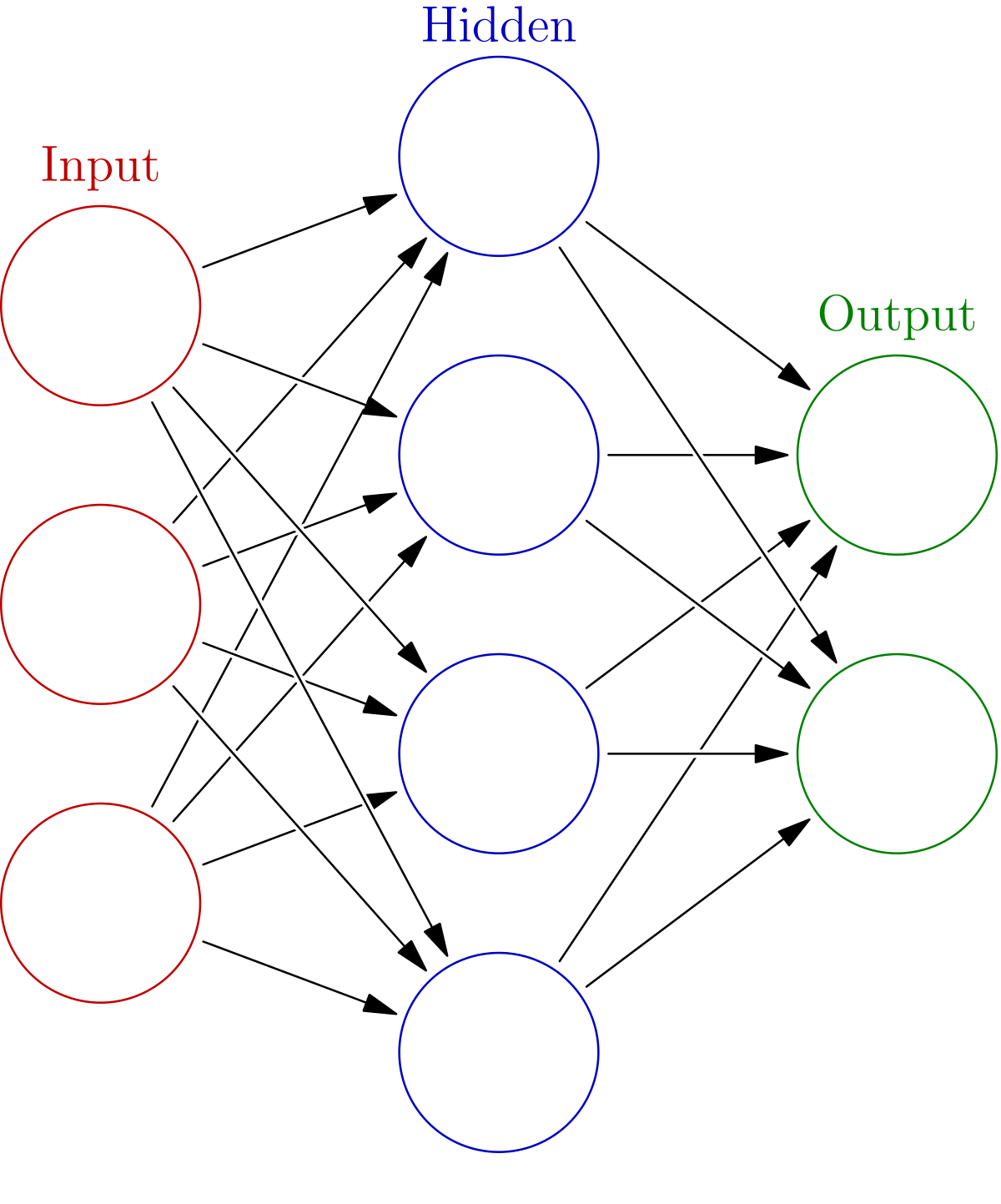
**4.5 Deciding on the classification algorithm**

Our first hurdle when deciding the classification algorithm was understanding which classification algorithms were applicable to our project. We decided against employing any unsupervised learning algorithms as we believed the differences may be too subtle for the classifier to distinguish between risky and safe sentences. As our project requires a classification problem, we excluded any algorithms that generate continuous output.

The remaining supervised learning algorithms available to us were:

* Neural Networks
* Support Vector Machines (SVM)
* Decision Trees
* Naïve Bayes
* Logistic Regression

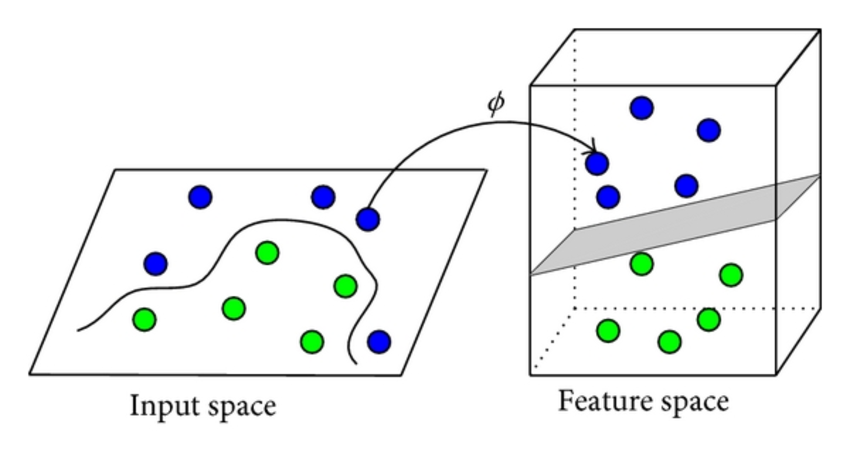
A Neural Network is a machine learning algorithm which consists of multiple units in multiple layers with each unit connecting to every other unit in the next layer. The structure of a neural network is always composed of one input layer, at least one hidden layer and an output layer. From a high-level perspective, the purpose of hidden layers is to combine the weighted inputs from the first layer in order to capture additional complexity from the data.



We believe that a neural network would be a good algorithm to evaluate more closely because often the distinction between a risky sentence and a safe sentence is very subtle. Therefore, a more complex model may be required to differentiate between the nuances in the data.

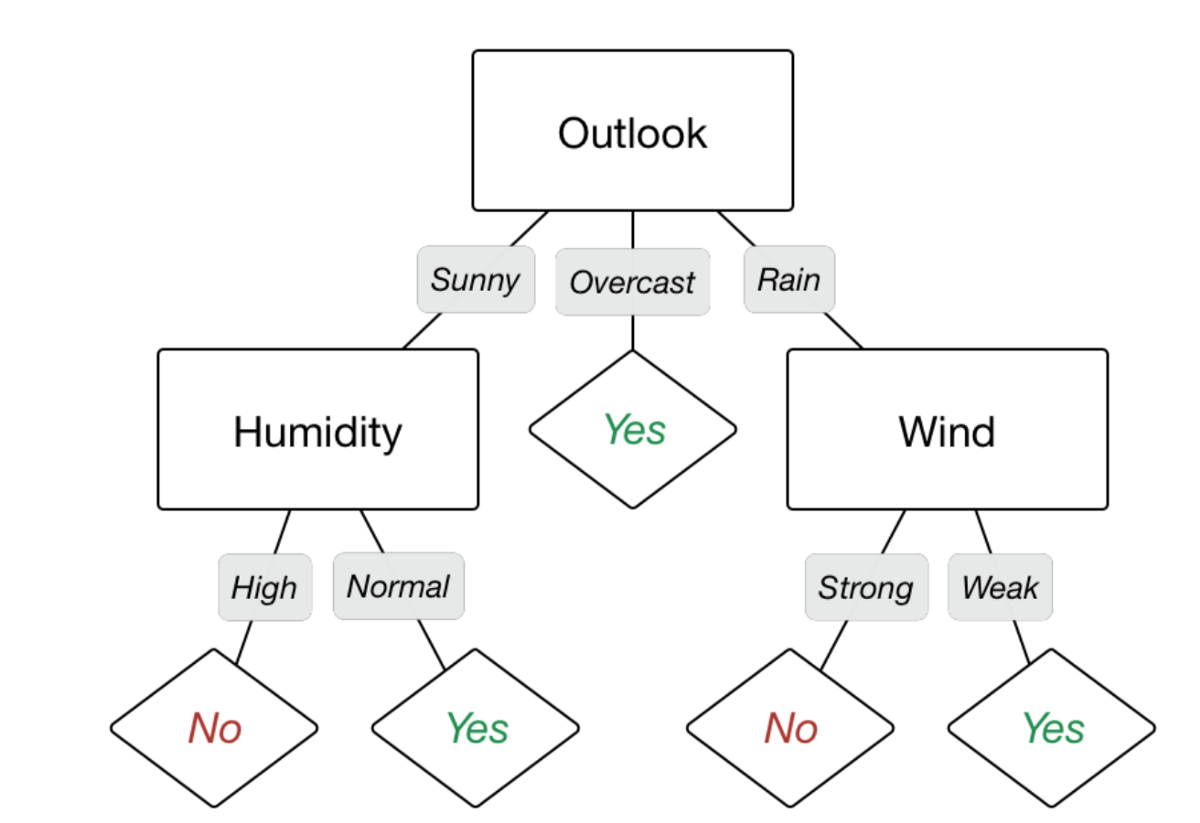
A Support Vector Machine separates two classes by finding the optimal hyper plane between them. The primary advantage of SVMs is that they can separate a non-linear function easily. By using a technique called the ‘kernel trick’, non-linear space can be transformed into linear space, enabling the two classes to be separated easily.

For example, suppose there are two classes which are governed by the function f(x^2). Currently these classes are non-linearly separable. However, when the input space is transformed by the sqrt(X), the separability between the two classes becomes linear.



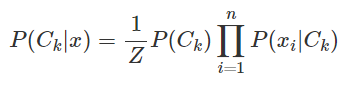
Our task may require a non-linear boundary between the two classes due to the complexity of the classification problem. Therefore, we considered this algorithm for further evaluation.

Decision trees in machine learning are constructed by examining the entropy of the attributes. Entropy is a measure of the homogeneity of the samples. An entropy of 0 indicates high homogeneity while an entropy of 1 indicates low homogeneity. Attributes of low entropy are more favorable to a decision tree because they offer a more attractive split to help distinguish between the two sets of classes. For example, an attribute of ‘gender’ which has 4 examples labelled to the positive class and 4 examples labelled to the negative class does not provide us anymore insight regarding our ability to distinguish between the two classes.



Although Decision Trees struggle to match the complexity of Neural Networks or SVMs, the interpretability of the model appealed to us. The model’s decision process can be revealed to users of the website to support a more transparent application.

Naïve Bayes is a simpler learning algorithm that focuses on utilizing bayes theorem to calculate the probability of a class. A new example is classified by multiplying the calculated probabilities of discrete attribute values occurring for a certain class, then comparing the returned probabilities of the different classes occurring given the example.



The Naïve Bayes classifier assumes independence between its attributes. Therefore, each attribute’s contribution to the model can be interpreted by examining the calculated conditional probabilities. This makes the model very interpretable for a human examiner. In addition, a Naïve Bayes classifier is inexpensive to train. For these reasons, we decided to investigate this algorithm further and evaluate its performance using our dataset.

Logistic Regression uses a sigmoidal function on the attribute values to differentiate between the two sets of classes. However, we did not decide to perform further evaluation on this algorithm. As a group we decided that its decision boundary was too close to that of SVMs, while SVMs provided better classification results in nearly all cases.

**4.6 Developing the classification algorithm**

Assessing the best machine learning algorithm to use is difficult because of the many different combination of parameters available for the different learning algorithms. To help find the optimum learning algorithm we performed AutoML on our neural network. This involved developing a script that iterated through different combinations of hidden layer sizes and returning the precision, recall and accuracy scores on each iteration. By using *Figure A.5*, we can examine the precision, recall and accuracy scores for the different combinations of hidden layer sizes to help identify the optimum parameters for our neural network. The optimum parameters using the current results are highlighted (1 hidden layer, 30 hidden units).

The ratio of safe sentences to risky sentences in our dataset is approximately 13:1.

*See Figure A.3*

Therefore, our task of identifying risky sentences in contracts is a classic example of a minority class problem. Datasets containing minority classes may produce misleading results for the accuracy scores. This is because, during training, the model’s objective is to minimize its error in order to achieve the best accuracy score.

However, a classifier can still achieve a high accuracy score by predicting every example as the majority class. For our classification task, this is not useful because the user only wants to know about the risky sentences present in the document. The metrics associated with understanding how well a model performs on minority class examples are called precision and recall scores. Recall is the percentage of minority class examples correctly predicted out of the total number of minority class examples in the dataset. Precision represents the total number of examples predicted correctly out of all examples predicted as the minority class.

One technique used to deal with minority classes is called ‘SMOTE Oversampling’. Synthetic Minority Over-sampling Technique (SMOTE) synthetically creates new examples for the minority class that are similar to current examples assigned to the minority class. This forces the classification algorithm to consider the minority class when minimizing its error and produce higher recall scores as a result. As shown in *Figure A.4*, after implementing oversampling using the sklearn library, recall scores for the positive class (risky) did increase for 3 out of the 4 algorithms. SVM’s in particular appear to be unusable without oversampling applied. However, precision did decrease for the positive class for all 4 algorithms when applying oversampling. We decided that the tradeoff between recall and precision was more favorable when applying oversampling with regard to our project’s objective.

Understanding that users operating on our site are concerned about their privacy, we believed it was better to return a greater number of risky statements with more misclassified safe sentences rather than return fewer risky statements with less misclassified safe sentences.

The performance of the algorithms listed in *Figure A.4* may change over time as a result of changes in data format or improvements to the existing algorithms. However, as evident in *Figure A.4,* currentlyour best performing algorithm is a neural network using oversampling with 1 hidden layer and 30 hidden units. This will be used as our chosen classification algorithm

**4.7 Classification API**

The motivation for developing an API is so that our website can communicate with our classifier and is essential for enabling normal users to analyse contracts.

When the API starts, it runs the classification algorithm to generate a predictive model. Once a request is received, the API parses the given document and extracts the sentences. Each sentence is then run through the classifier to obtain a prediction (risky/safe). All risky sentences are returned to the website in JSON format so they can be displayed to the user.

**4.8 User Interface (Website)**

**4.8.1 Iteration 1**

Our plan at the start of the project was to design a user-friendly website that allowed users to review their contracts for any risky or concerning statements. Our initial design requirements included the following:

* Text box: for user input where statements would be highlighted to insinuate, they were risky
* Submit button: to initiate the classification models and perform action of highlighting risky statements
* Languages: HTML, CSS and JavaScript
* Cross Browser Compatibility

Due to its simplicity and overall appearance we used the implementation as a template for future iterations. But the advantage of the design was that we were able to create a core plan for how we wanted our interface to look.

*See Figure A.6*

**4.8.2 Iteration 2**

After reviewing the website and discovering an issue regarding the text area, we decided to update the content of the website and how the interface will highlight the risky statements. The issue that we discovered in the text area during testing is that the risky statements are not highlighted despite the server returning the list of risky statements. We found out that if the list of risky statements returned by the server are too many, then the highlight text function stops working. So instead of highlighting the text on the text area, we decided to return the list of risky or concerning statements. A modal was implemented to show the list of risky statements. A modal is a dialog box that is displayed in from of the web page. When the user clicks the ‘Analyse’ button, this triggers the modal and shows the list of risky statement(s).

In addition, we decided to add sections such as the Header and the About section. The Header section introduces the user interface with the title and a brief explanation of the website. The About section will be a short and simple explanation regarding how the text entered in the user interface is processed. We also changed the overall look of the website by applying plain and light colours such white, shades of grey and light blue. This choice of colours is to increases the quality of the website by making the text stand out from the background and having a consistent style throughout.

The website communicates with the server that runs the classification algorithm using a jQuery AJAX method. This method allows data to be exchange with the server and update the modal without reloading the whole page or website. When the user clicks the ‘Analyse’ button, it calls the JavaScript function which takes text from the text area and transforms it into a Json file. The Json file is then processed by the AJAX method and sent to the server. When the server responds, the method receives the response, process it, and if it is successful it will return a list of risky statements. An error function has also been implemented to catch any errors.

*See Figure A.7*

**5. Testing**

**5.1 Cross Validation**

Testing your trained model using one subset of the dataset is not enough. The subset used may be favorable to the model, resulting in optimistic accuracy scores. Ten-fold Cross Validation provides realistic accuracy scores by testing the model using ten different subsets and therefore eliminating the chances of testing the model using a favorable test set.

We used WEKA to check for accuracy scores. Weka has an application ‘Explorer’ where we used to test and check scores. First, we upload the dataset in csv format on the ‘Preprocess’ tab and then we selected ‘Is\_Risky’ as our class. Finally, under ‘Classify’ tab, we chose a classifier, selected ‘cross-validation’ in 10 ‘Folds’ and pressed ‘Start’ button to run the classifier. After the application has completed the task, it generates a results buffer which will show the accuracy scores.

Unfortunately, oversampling does not work correctly on Weka. The whole dataset is oversampled before applying the train test split meaning there are often duplicate examples between the training and test sets resulting in invalid accuracy scores. To resolve this dilemma, we developed our own Cross Validation script that oversamples correctly.

The cross validation oversampling algorithm we developed works as follows:

1. Split the dataset into two different subsets, the test set (10%) and the training set (90%) while keeping a record of which examples have been used in each test set.
2. Oversample the training set
3. Create a model using the oversampled training set and test that model using the current test set.
4. Repeat 10 folds of steps 1 to 3 ensuring a new test set is used each time. Sanity checks are used in the code to ensure a new test set is used on each iteration and that there are no examples of the test set leaking into the training set.
5. Calculate average recall, precision and accuracy scores across all 10 folds.

The cross-validation results for each algorithm are shown in *Figure A.4*

**5.2 Web Testing**

**5.2.1 Functionality Testing**

We made sure that the website is valid by running it through <https://validator.w3.org/>. This also checks for HTML syntax errors. All buttons on the website have been tested to make sure they correctly perform their tasks. Since the website is only one page, we made sure that the appropriate buttons navigate to the correct section of the page.

When the ‘Analyse’ button is pressed, it checks whether the text area is empty or there is enough text/document to be analysed. This prevents the website from sending empty or insufficient data to the server. In addition, it is of capable of catching errors and displaying them to the user if it arises.

*See Figure A.8*

**5.2.2 Usability Testing**

The content of the website is proofread. We used a web application that strips the html tags and proofreads the content. The colours on the website are plain and do not contrast with one another. In Addition, a site map for the website is unnecessary.

**5.2.3 Compatibility Testing**

The website can run on different browsers, mainly Internet Explorer, Google Chrome, Mozilla Firefox and Safari with the most recent versions. It is also compatible with tablet and mobile devices. The @media is a CSS feature where it detects the size of the browser window and automatically adjusts the content and style of the website to adapt with the browser to maintain high quality content throughout the website.

**6. Challenges**

**6.1 Accuracy**

One major challenge that we experienced was producing accurate data. This involved accuracy in our pre-processing data and making sure that all the data collected followed the same guidelines. Originally, we had an issue with inconsistent data where we all had conflicting views on what was considered “risky”. This was a problem for us but in order to resolve this problem we created Terms and Conditions guidelines to keep us focus on what should be considered “risky” and prevent us from swaying in opinions.

**6.2 Expectation exceeding reality**

No matter how much we planned, gathered more data or how much time we spent on our project, we always came with the overwhelming feeling of having set the bar too high for what we expect to complete. As the idea was based around machine learning, an experimental area in computer science, we encountered hurdles. For example, the research required to develop aspects of the project exceeded our expectations. Even smaller features of the project like collecting data became a longer process than we first imagined. Looking back, ideas we brainstormed at the start of the project could’ve been refined to help us progress further into the analyser and expand its means of functionality.

**6.3 Algorithms**

Another challenge that we faced was choosing the right classification algorithm to apply to our model. With such a wide range of classification algorithms within machine learning e.g. Linear regression, decision trees, neural network, naïve bayes etc. we had a problem with choosing an algorithm that met our model requirement.

To have a model that easily could distinguish between “risky” and “non-risky” sentences we needed an algorithm that produced high accuracy and recall results. As a group we had issues with low results from the chosen algorithms and were struggling to find an algorithm that produced a high enough accuracy and recall results to meet requirements. One of the solutions to this issue was to collect more training data i.e. review more contracts. However, due to time constraints this wasn’t always feasible.



**Conclusion**

**7.1 Project Reflection**

We built a system that was able to identify risky statements within terms and conditions contracts using a neural network in combination with other data mining techniques. As a group we feel as though we were able to hit the targets that we envisioned. Documents can be analyzed by returning feedback to the user by presenting them with the risky statements found in the terms and conditions contracts. We were able to improve our classifier so that the website returned enough relevant statements to fulfil the purpose of the product.

**7.2 Product Comparison**

After comparing our finished product to Polisis and EULAlyzer, there are few advantages and disadvantages of using these applications instead of using our project for analyzing terms and conditions contracts.

After reviewing Polisis’ technical report, I found that their F1 score for identifying risky statements is 0.81. Their accuracy when identifying risky sentences is visibly greater than our projects, which has a F1 score of 0.46. However, Polisis is only able to analyze a range of privacy policies (21214) while our project is more flexible, allowing users to paste in any terms and conditions contract.

EULAlyzer has an intuitive format when returning risky sentences after performing analysis. It categorizes the risky sentences, allowing a user to more effectively review how their data is being used. However, EULAlyzer fails to use AI when analyzing contracts. Instead it identifies risky statements by searching for keywords in sentences. This results in a large number of safe sentences being classified as risky leading to weaker accuracy scores than our project.

**7.3 Future Improvements**

**7.3.1 User Feedback**

A future improvement could include sending an email of risky statements found to the users so they can further evaluate it. Adding a registration feature would mean users would have an account which would further enable us to send them updates about new features added to the analyser.

**7.3.2 Browser Add-Ons**

Future enhancements would include making the analyser a web browser add-on. Its purpose is to provide in-browser notifications reporting any risky statements found. The number of risky statements would be displayed within the add-on icon when navigating to a new website. The risky statements found would be displayable via clicking the browser add-on icon.

**7.3.3 Cost sensitive classification**

To improve the results of our classification algorithm, we could use Cost Sensitive Learning. Currently, without Cost Sensitive Learning, all the misclassifications are treated equally during training. However, Cost Sensitive Learning penalises misclassifications of the minority class more heavily by increasing the penalty on false negatives. Although this may increase the number of safe sentences misclassified as risky, the trade off will be more beneficial because users will be notified of a higher proportion of risky sentences.

**Final Words**

To sum up, our project prompted a deeper inquiry into data mining and neural networks and we were able to achieve the core aims of our project. As a group we accept there are aspects of this field that could not be explored in this field and we hope this project ignites a greater interest.

**Acknowledgements**

The team would like to thank Alex for his support and showing great interest in the success of the project. The team would also like to thank all companies whose contracts were used and the developers of sklearn for providing such a powerful framework for us to work with.

**8. References**

[1]<https://ec.europa.eu/info/sites/info/files/terms_and_conditions_final_report_en.pdf>

(Viewed 20/3/2019)

[2] <https://tosdr.org/>

(Viewed 6/2/2019)

[3]<https://www.brightfort.com/eulalyzer.html>

(Viewed 6/2/2019)

[4] <https://pribot.org/polisis>

(Viewed 6/2/2019)

<https://machinelearningmastery.com/improve-deep-learning-performance/>

<https://www.w3schools.com/jquery/jquery_ajax_get_post.asp>

<https://christophm.github.io/interpretable-ml-book/other-interpretable.html>

<https://www.google.co.uk/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjplfePr53hAhWv2OAKHdaDCIgQjRx6BAgBEAU&url=https%3A%2F%2Fbecominghuman.ai%2Fdecision-trees-in-machine-learning-f362b296594a&psig=AOvVaw28Kdvgfqt4VKOmceXNosbS&ust=1553605886676172>

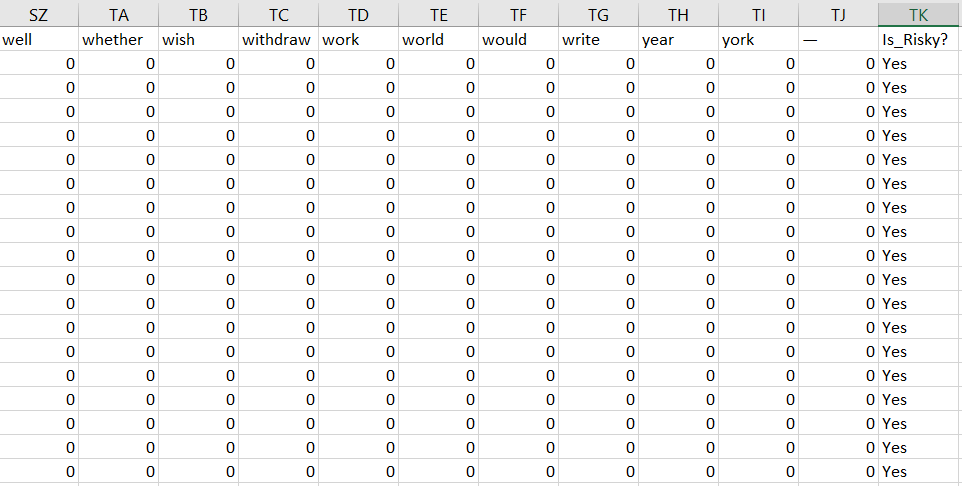
<http://i.imgur.com/WuxyO.png>

<https://www.google.co.uk/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjUjd36p6DhAhUI3RoKHQ_OD3MQjRx6BAgBEAU&url=https%3A%2F%2Fwww.datacamp.com%2Fcommunity%2Ftutorials%2Fstemming-lemmatization-python&psig=AOvVaw32CTElWeic4HU9lr7Jx8JL&ust=1553707045178325>

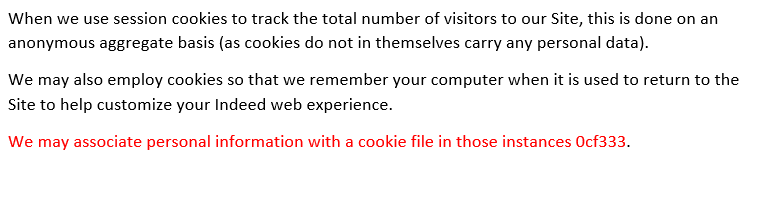
<https://www.google.co.uk/search?biw=1536&bih=792&tbm=isch&sa=1&ei=V16aXMyvIa6I1fAPxYa-8Ao&q=neural+network&oq=neural+&gs_l=img.1.0.35i39j0i67j0j0i67j0j0i67j0l4.35502.37325..38101...2.0..0.69.545.9......1....1..gws-wiz-img.7-U5LcaASFU#imgrc=JHhP7TOdaWcwbM:>

**Appendices**

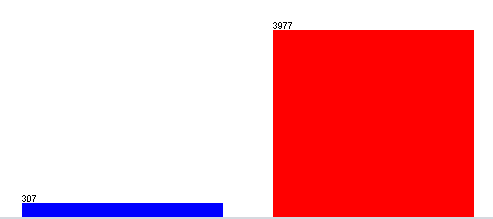
**Figure A.1**



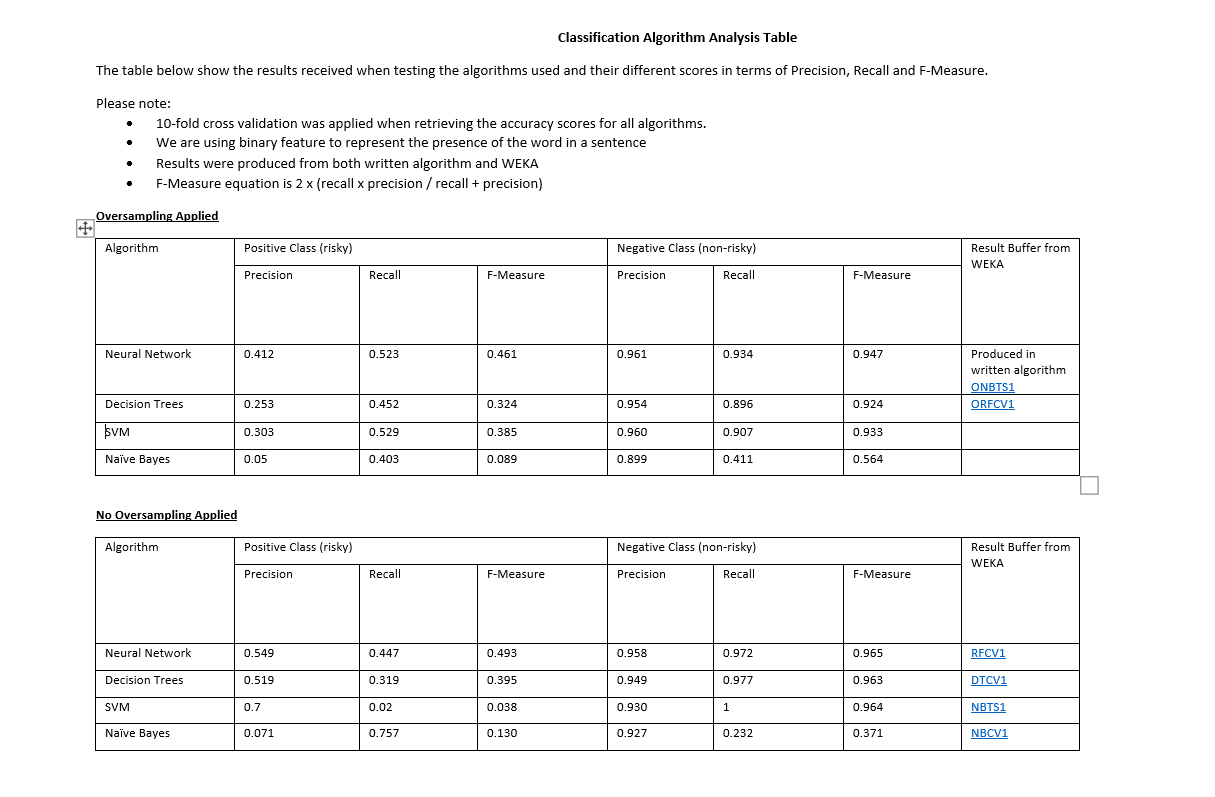
**Figure A.2**



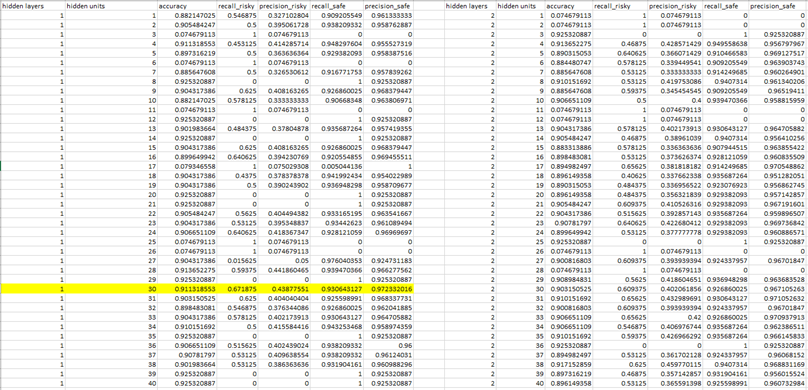
**Figure A.3**



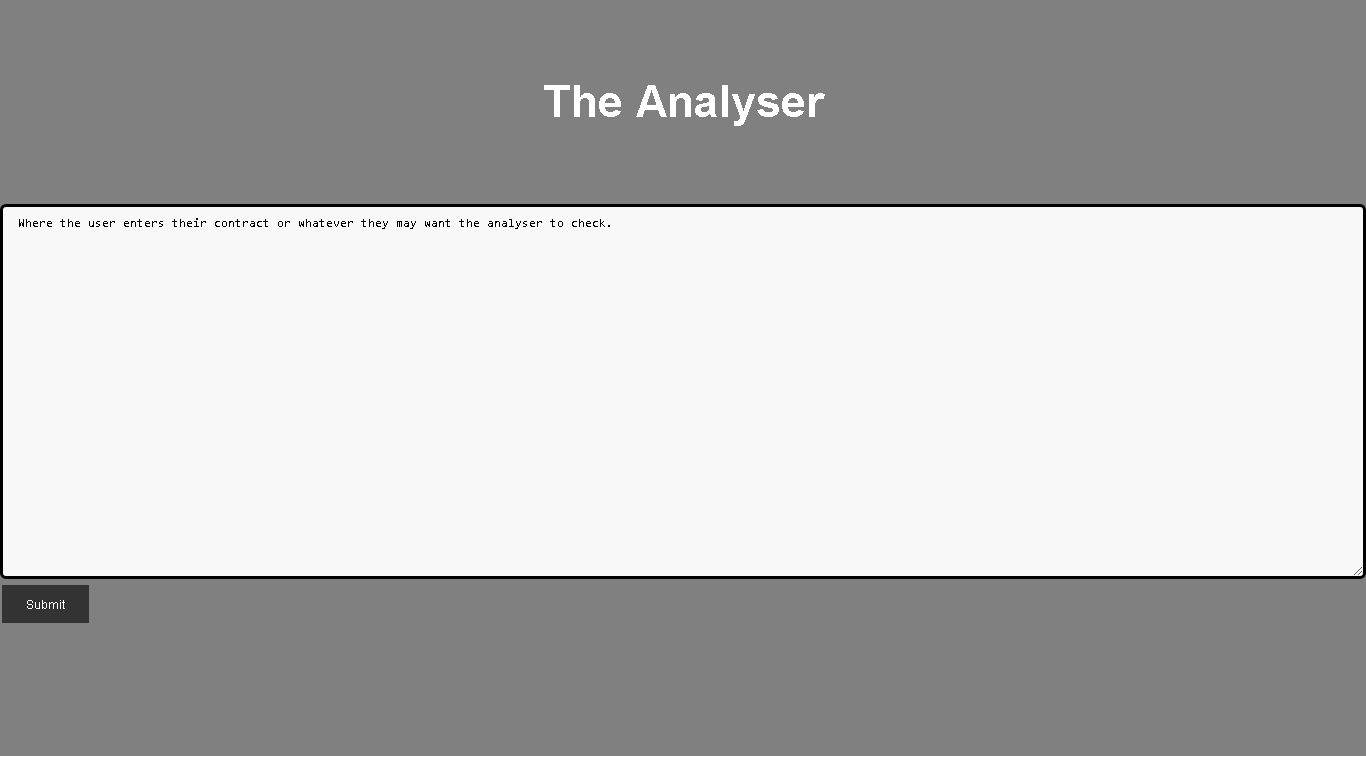
**Figure A.4**



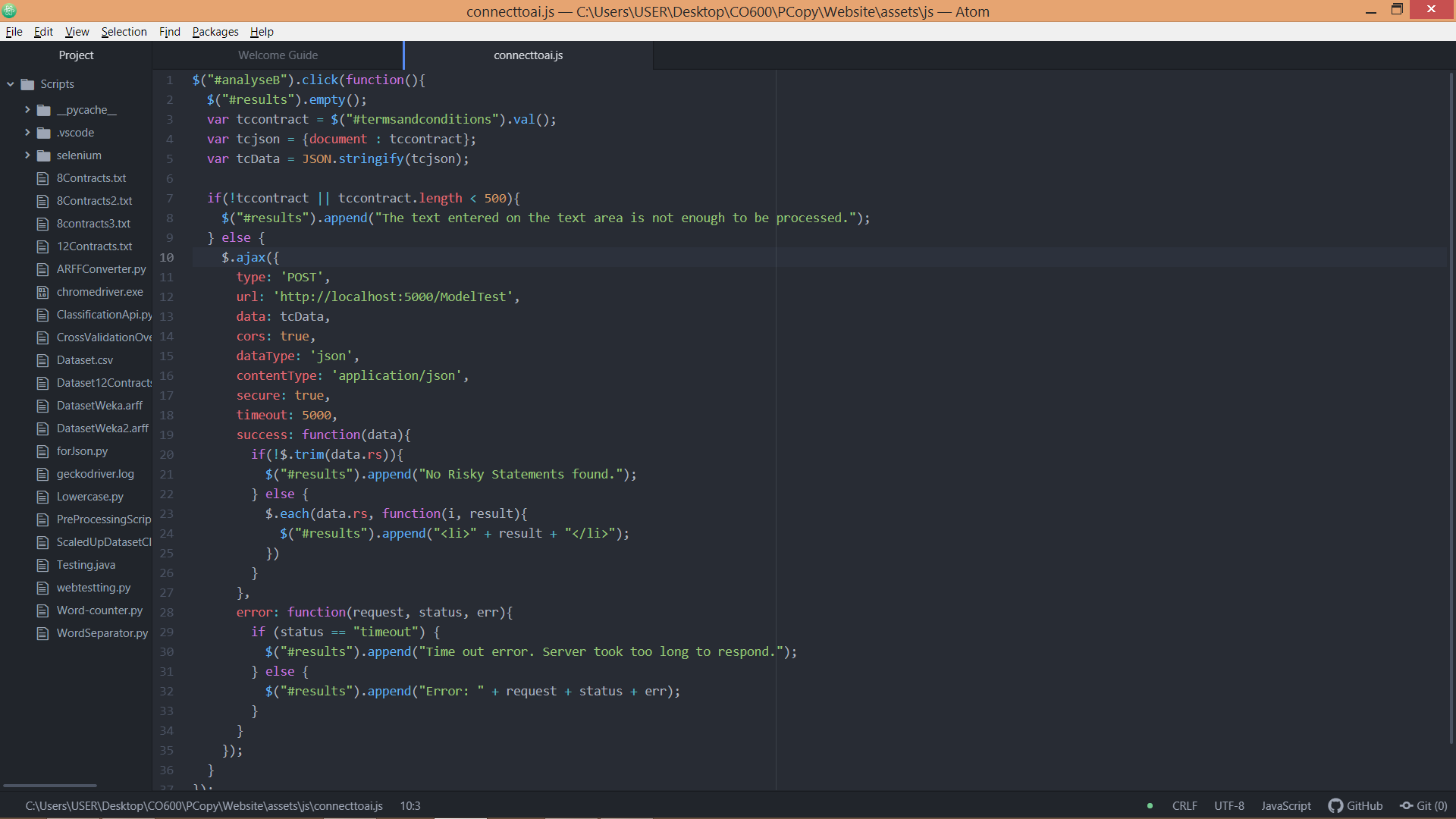
**Figure A.5**



**Figure A.6**



**Figure A.7**



**Figure A.8**

