## Pipeline

One of the very first changes we made to our model after the baseline implementation was to switch from the CountVectorizer to the TfidfVectorizer which resulted in a small performance increase. [1] The difference between the two is only a small one, the value of TFIDF increases proportionally to count, but is offset by the frequency of the word in the corpus. This means less importance is given to extremely common words such as “the”, and more importance is given to rarer and more unusual words.

In addition to the change mentioned above, we also set use\_idf=True in the TfidfTransformer [2], this enables inverse document frequency reweighting when normalizing the count matrix from the previous step. This also led to a small performance boost.

Towards the end, we tested all the relevant text classification algorithms to see which offered the highest prediction accuracy, we tested the following algorithms [2]:

* (Linear) SGDClassifier is the algorithm we decided on in the end due to its impressive performance increase of around 20% compared to Naïve Bayes (the algo we started off with). It also has a large selection of loss functions for further fine-tuning (will get to that later).  
  accuracy: 78%
* (Linear) LogisticRegression is known as a crowd favourite among text classification circles and came in at a close second in terms of accuracy, making a strong case for linear-type classifiers being the best choice for our kind of problem. On a technical level, Logistic Regression is extremely similar to SGDClassifier, the only difference being that the latter has a Stochastic Gradient Descent solver.  
  accuracy: 70%
* (ensemble) RandomForrestClassifier methods correct for the overfitting found in decision trees.  
  accuracy: 62%
* (naïve bayes) MultinomialNB is a probabilistic classifier which depends on the Bayes theorem which statistically calculates an unknown parameter based on already known information but with a strong emphasis on making the features strongly independent. This algorithm performed ok and made a great baseline to compare to, however was just not suitable for our data.  
  accuracy: 54%
* (nearest neighbours) KNeighboursClassifier is better known for more loose data classification problems such as categorising hand written text due to its way of finding the next nearest match. However, in the English language, the next nearest match isn’t beneficial to the model and may even cause problems.  
  accuracy: 46%

Finally, we tried out various loss functions on the SGDClassifier, in the end we found modified\_huber to be the best [3] – not only because it had the highest accuracy but also because it allowed us to return the % certainty of the prediction. This meant making small changes to the API and GUI, as well as our model class in order for this to become visible to the user [4].

## File structure

Initially, our project was just a python notebook file, we quickly realised that making the program modular and readable on GitHub commits, we would require the use of standard python files. [5] We later decided that the program is better split up into multiple files, the first sign of us doing so is when we imported the data loader from an external file into our model script. [6] We followed this archetype of separating code by function into separate files thought our project.

## Class Structure

Part of the success of separating files out is that we can have a neat and organised class structure. When importing a class from a different file it is as easy as:

* from file import class

This allows us to instantiate classes from other files as objects, which was extremely useful when using the model in the API.py for example on lines 4, 14 and 23; in model.py on lines 14, 18 and on virtually every line of the demonstration script. This development choice paves the way towards our model being used as a library. You can see where we switched from a linear system to a class based system in [9].

## Conda environment

As advised by our tutors, creating an environment .yml file is an effective way to make sure the environment we run our code in such as installed packages is reproducible across various machines. [7] For users who do not use Conda, we included pip installation instructions in the readme. [8]

## Demonstration script

To show off our code in all its modular nature, we created a demonstration script that works in a similar way to how a user would work with any other python library, they import the model class from the model file, instantiate it with the relevant parameters etc. Once a model has been loaded or trained, it opens the GUI and starts the API and the two cooperate instantly.

## Commit bibliography

[1] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/e4f802d07219dffa513882d902b9191702628ddf>

[2] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/e361aae1a36b260bb40d4d8d4a3629148af3a8f0>

[3] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/9aceea75963a6c217af16ecb73a84f4d2f068c90>

[4] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/2f808e8ea82f1f2210da3addc4522d143c808fd9>

[5] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/2d4c9b246674c16bc8b54b4bc85d1b62b8046c02>

[6] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/d40f817548d9b370e6228201bc9d2e2f64a606d0>

[7] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/00b5b2338d3e6bfb213d0408f365f0f8b3c7802d>

[8] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/86060792c19aa1c8a9b3652eb5245f10f83d5f7c>

[9] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/e9a51a0f44a21d71bb2cc28c0be891dc1bb4bb91>

[10] <https://github.com/sisygoboom/fools-and-horses-group-proj/commit/6d3d482878bf6617b92754de8ed6b19bc70d6f03>