# NLP – Title Detection

## Project Outline

This project aims to build a machine learning model to detect whether a line in a document represents a title or not, based on the text as well as its placement on the page, etc.

## Results

Performance metrics from running the model against the test set:

* Accuracy: ~97.7%
* F1-score: ~97.7%

## Solution

### Pre-processing the Data

In the effort to normalise the data, ensuring each feature is equally impactful during training, continuous values were rescaled to fit between 0-1. Moreover, discrete values were encoded from [true, false] to [0, 1]. Lastly, the bag-of-words model was used to numerically encode the texts via vectorisation, taking term frequency into consideration. Consequently, the challenge of combining text-data with both discrete and continuous numerical data had been solved.

Another challenge encountered in the pre-processing stage was dealing with the class-imbalance. The dataset wasn’t big enough to do undersampling and risk losing significant information, so the use of oversampling was explored as a result. The SMOTETomek method was deemed the best for the purpose of this project as it creates new *plausible* data points by connecting known points in the feature space, as opposed to duplicating existing points from the underrepresented class. However, the generation of a high number of new data points in this fashion might have caused the model to overfit to the underrepresented class, as the performance of the classifier was not significantly improved. Vectorising the text-feature may have made the feature-space too high-dimensional for this approach, and the method was therefore not used in the end. Nonetheless, the final classifier had a very high accuracy rate and most importantly f1-score.

### Selecting the Model

A support vector machine was initially selected as the classification model since it works effectively for high-dimensional data, i.e., vectorised text-data. The hyperplane (decision boundary) can be drawn for complex non-linear data and, via the kernel trick, it maximises the margins to the support vectors, lowering the risk of overfitting.

Regardless, the use of a deep neural network was explored as it is also known to work efficiently with high-dimensional data. Furthermore, it has proven to continue to increase in performance as more data is fed into it, as opposed to plateauing. Importantly, overfitting is effectively avoided via added dropout-layers for regularisation. Via deep learning techniques, many layers can be added to a neural network to ensure the entire scope of the data has been captured in the training stage.

When compiling the model, a binary cross-entropy function was used which is effective for binary classification problems and the ‘adam’ optimiser was used, a stochastic gradient descent algorithm. A pre-determined number of fully connected layers was used, found via a trial-and-error approach, but the most optimal number of nodes in each layer was found via a random search using a cross-validation approach. The ReLU activation function, which overcomes the vanishing gradient-problem, was used for all 4 layers except the last one which implemented the sigmoid activation function with 1 node, adaptable to binary classification problems. Two dropout layers were added with a 20% dropout rate, which was deemed effective enough for the purpose of regularisation.

### Tuning the Model

The random search found the following number of nodes to be the most optimal for each layer:

* Layer 1: 120
* Layer 2: 60
* Layer 3: 40
* Layer 4: 60

Subsequently, the number of epochs and size of batches were explored in the hyperparameter tuning, which was performed via a cross-validation grid search of parameter-values on the network with the most optimal topology found earlier. The search found 100 to be the most optimal batch size, with 100 epochs.

The final model, with achieved optimal network topology and hyperparameter values, was saved as ‘clf\_model’.

## Conclusion and Future Areas of Improvement

The final model achieved very high accuracy and f1 scores, indicating that it is an accurate and unbiased classifier. However, certain measures can still be taken to improve it. Given more time, another approach to dealing with the class-imbalance could be explored, performing random oversampling, or performing SMOTETomek oversampling *in combination* with some undersampling.

For the neural network, a topology with different *types* of layers, such as LSTM, convolutional and max-pooling layers could be experimented with, as e.g., recurrent layers would add long-term memory retention which is especially usefully when learning the meaning of a sequence of characters in a word.

Lastly, the solution could be parallelised with Apache Spark to reduce the computational time, since the current version of the solution requires a lot of time to find the best network topology and hyperparameter-values.

## Time

The model required 2.5h to find the most optimal network topology, tune the network hyperparameters and train on the training set to find the best node weights. The saved model ‘clf\_model’ can be reloaded to make predictions on the test set in roughly 7 seconds.

Roughly 20 hours were spent working on this project, distributed across 3 days. A considerable amount of this time was spent researching models and methods to explore in the solution.

SVM RESULTS

* Best accuracy: 0.9614491734083714
* Best parameters: {'C': 1, 'gamma': 0.1}
* F1 score: 0.9791006966434453
* Confusion matrix: [[1164, 23], [10, 382]]

QUESTIONS

* How much preprocessing should I have done?
  + Are there any cases when it's better to stemmatize instead of lemmatize?
  + Should I also have removed symbols?
  + Should I not have lowercased the words?
  + Is there anything else I should've done?
  + How else could I have dealt with class imbalance?
  + How could I have built a data pipeline to process the entire dataset (including both numerical and text values)?
  + Word2Vec, transformers, glove, bert
  + They do their own feature extraction sometimes and sometimes within the model like nn
* How could I have improved the performance of the network?
  + Would an LSTM layer (adding recurrence) have done that?
  + CNN (or just for computer vision)?
  + How do you decide on the best network topology for a project?
  + LSTM Good for context, remember information from previous
  + Bi-lstm
  + Cnn : images
* Have you fully let go of keras and moved to sci-kit learn; are they equally good?
* Do you only use deep learning or other ML models as well?
* What other prediction models could I have used?
* What other libraries could I have used? Spacy
* How did you generate the CSVs?
* Do you parallelize the computation in your projects with big data frameworks?
* What do you do when your code is running?
* How do you generate a setup script?
* What sort of business problems do you solve with NLP?
* Will I only work on NLP projects? Data structuring from unstructured detect table, doc segmentation, hierarchy detection, ner, header footer detection, new: data capturing (text summary, q and a), computer vision, object detection
* Does the client always just provide documents?

Bonus Qs

1. Use api, request, get
   1. Build services, use api’s to interact
2. Postgres, db browser for sqlite
3. Sql statement
4. Unit testing and Integration test
5. 0.95, use cases, requirements