A Supervised Learning Approach For Heading Detection

Preprint · September 2018

CITATIONS
0

READS
2,716

2 authors:

Sahib Singh Budhiraja
Lakehead University Thunder Bay Campus
3 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Vijay Mago
Lakehead University Thunder Bay Campus
117 PUBLICATIONS **864** CITATIONS
SEE PROFILE

A Supervised Learning Approach For Heading Detection

Sahib Singh Budhiraja and Vijay Mago
(sbudhira,vmago)@lakeheadu.ca

Lakehead University, 955 Oliver Rd, Thunder Bay, ON P7B 5E1

Abstract. As the Portable Document Format (PDF) file format increases in popularity, research in analysing its structure for text extraction and analysis is necessary. Detecting headings can be a crucial component of classifying and extracting meaningful data. This research involves training a supervised learning model to detect headings with features carefully selected through recursive feature elimination. The chosen classifier has an accuracy of 95.83%, sensitivity of 0.981 and a specificity of 0.946. This research into heading detection contributes to the field of PDF based text extraction and can be applied to the automation of large scale PDF text analysis in a variety of professional and policy based contexts.

Keywords: Heading Detection · Text Segmentation · Supervised Approach.

1 Introduction

As the amount of information stored within PDF documents increases worldwide, the opportunities for large scale text based analysis requires increasingly automated processes, as the amount of document processing is time consuming and labour intensive for human professionals. Systematic processing and extraction of textual structure is increasingly necessary and useful as demonstrated in El-Haj et al.'s work involving 1500 financial statements[7]. Categorizing data into seperate sections is quite easy for humans, as they rely on visual cues such as headings to process textual information. Machines, despite being able to process large amounts information at high speeds, require effort to classify and interpret text based data. This paper explores the application of supervised classifiers to operationalize a system that would aid in the identification of headings. PDF documents are a visually exact digital copy that displays text by drawing characters on a specific location [10] and present a challenge in analysis because the files do not provide enough information on how the text is organized and formatted.

A supervised classifier that is trained on labelled data provides one solution to categorizing PDF text as it tells the classifier how to make predictions based on the data provided. This research involved comparing and systematically testing

a variety of classifiers for the purpose of selecting classifiers best suited to this application. *Recursive feature elimination*[8]is used to ensure the classifiers only use the best and minimum number of features for making predictions. Cross validation is used to tune the hyper parameters of a given machine learning algorithm for increased performance before testing it out on test data. The final trained classifier is currently being applied to detect headings in course outline documents and extract learning outcomes. The extracted learning outcomes are being used for automating the process of developing university/college transfer credit agreements by using semantic similarity algorithms[3].

2 Related Work

While PDF format is convenient as it preserves the structure of a document across platforms, extracting textual layout information is required for detecting headings and further analysis. One solution to extracting layout information is to convert the PDF into HTML and use the HTML tags for further analysis. Once converted to HTML all the information related to text formatting required for the analysis, like font size and boldness of text, can be easily extracted. A variety of PDF to HTML document tools are available and have been assessed based on the text and structural loss associated with each tool[4]. Additional work includes PDF to HTML text detection approaches that maintain layout and font information[2], table detection, extraction and annotation[1] and analysis using white spaces[5]. HTML conversion is clearly a well established approach to analyzing PDF layout and content.

Previous research provides insight into processes related to extracting the heading layout of a HTML document[6]. In Manabe's work, headings are used to divide a document at certain locations that indicate a change in topic. Document Object Model(DOM) trees are used to sort candidate headings based on their significance and to define blocks. A recursive approach is applied for document segmentation using the list of candidate headings and evaluate with good results using a manually labelled dataset.

El-Haj et al. provide a practical application of document structure detection through the analysis of a large corpus of UK financial reports including 1500 annual reports from 200 different organizations. A list of 'gold standard' section names was generated from 50 randomly selected reports and used to match with corresponding sections of every document page in the dataset. Section matches were then extracted and evaluated using sensitivity, specificity and F1 score in addition to being reviewed by a domain expert for accuracy[7].

Current research has taken steps towards a system which analyses a document's textual structure. But there is a need to have an approach that can efficiently and accurately analyse the textual layout of a document and divide it into con-

tent sections to automate the process of extracting text from a PDF documents. We present out supervised learning approach for heading detection as a solution for it.

3 Methodology

3.1 Data Collection

Our data set consisted of 500 documents[1] downloaded from Google using *Google Custom Search API* [11]. To extract the correspoding formatting/style information the documents were converted from PDF to HTML using *pdf2txt*, which is a PDFMiner wrapper available in Python [12]. This is illustrated in Fig 1 which shows some sample text and its corresponding HTML tags generated using the conversion process. The final data points are also shown in the Fig 1, which was generated by parsing the HTML tags using regular expressions. A regular expression is string of characters used to define a search pattern[13]. The regular expressions used for parsing the tags are as follows:

To extract Font size and corresponding text:

$r`< \s*?span[^>]*font-size:(\w*)px[>]*>(.*?)<\/span\b[>]*>'$

To check if text is bold we look for the following regular expression for the word bold in the starting tag:

$r`[Bb]old'$

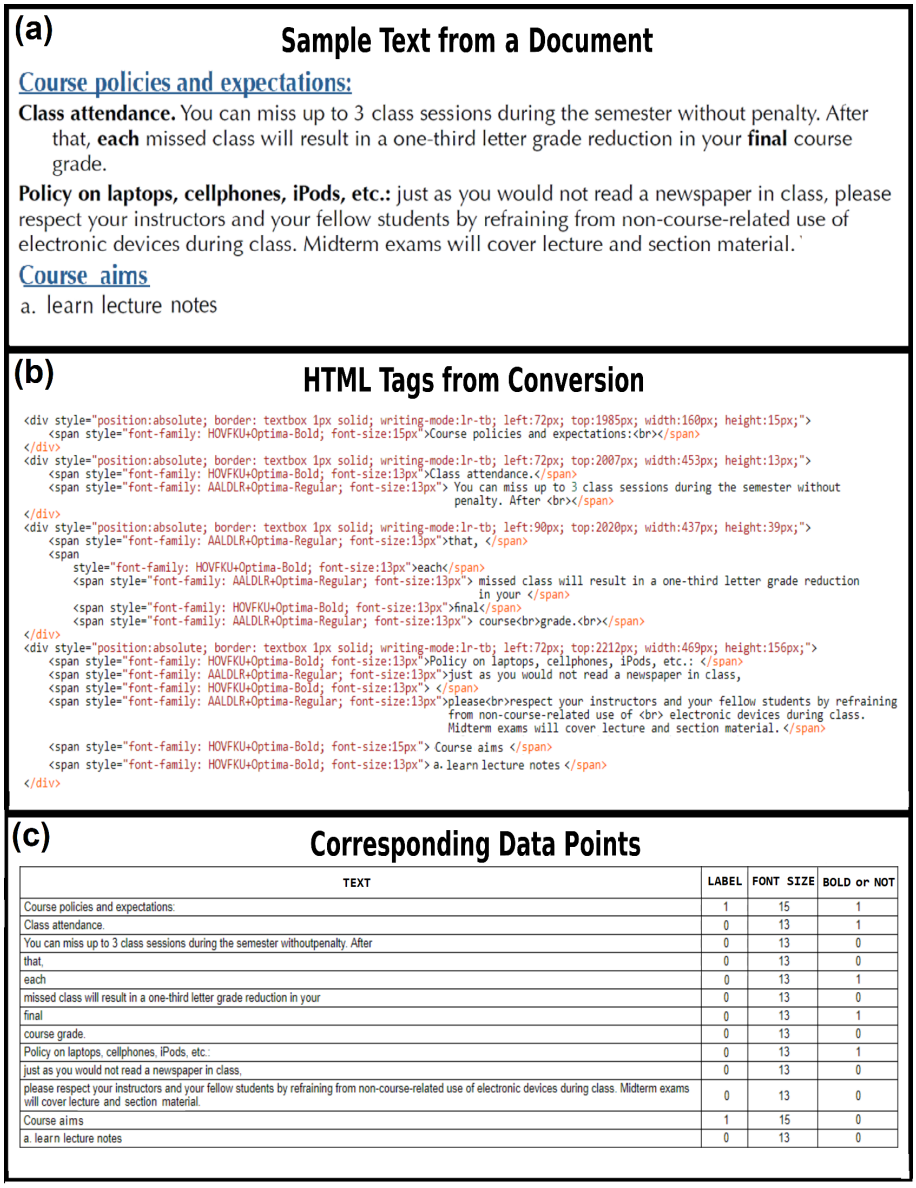
Each data point contains some text, font size and a flag which is either 1 or 0 depending on the corresponding text being bold or not. The whole process yielded 83,194 data points, which was then exported into an Excel file for further pre-processing.

3.2 Data Preprocessing

The process of transforming raw data into usable training data is referred to as data preprocessing. The steps of data preprocessing for this research are as follows:

Data Labelling: Data labelling refers to the process of assigning data points labels, this makes the data suitable for training supervised machine learning models. All the 83914 data points are manually labelled by cross referring to the

[1] Repository available at: https://github.com/sahib-s/Heading-Detection-PDF-Files



(a) Sample text from PDF file, (b) Corresponding HTML code for the text and (c) Corresponding data points from parsing HTML tags

NOTE: The 'LABEL' column in (c) is manually tagged and not generated via parsing.

Fig. 1. Extraction of Data from Documents

documents as both training and testing data needs to be labelled. If the text in the data point was a heading the label was set to 1 otherwise 0. Labelling data is one of the most important steps of preprocessing because the performance of the model depends on how well the data is labelled. Example of labelled data points is provided in Fig 1(c).

Balancing The Dataset: The dataset is considered imbalanced if the prevalence of one class is more than the other. The number of headings in our dataset is very less as compared to non-headings, this is because of the fact that the number of headings in a document is far less than the number of other text elements. Sklearn's implementation for Synthetic Minority Over-sampling Technique (SMOTE) is used to balanced the dataset, which does so by creating synthetic data points for the minority class to make it even [18, 16].

Data Transformation: The process of transforming data into a form that has more predictive value is known as data transformation. The purpose of data transformation is to convert raw data points into 'features' that contribute to more predictive value in training and decision making related to heading identification. For example, font size and text are two features from the raw data which, in their base form, do not have much value but can be transformed into useful features for training an efficient model. The list of transformed data fields are as follows:

- Font Flag: Headings tend to be larger in terms of font size as compared to the paragraph text that follows. Therefore, a higher font size increases the probability that the text is a heading. However, since each document is unique, there can not be a single threshold applied across all instances.
  Thresholds are calculated for each document by measuring the frequency of each font size where each character with a particular font size is counted as one instance. The font size which has the maximum frequency is used as the threshold. This approach relies on the assumption that the most frequently used font size is the one that is being used for the paragraph text, so having any font size above that increases the probability of that text being a heading. Fig 2 shows that the most frequently used font size is for the paragraph text with size 9 and all other text above it has more chances of being a heading. *Font Flag* can take two possible values 0 and 1. If the font size for that data point is less than the corresponding threshold then the value is set to 0, otherwise it is set as 1.
- Text: The text is transformed into the following feature variables, which are also listed in Table 1.
  - Number of Words: The number of words in the text can be used for training, as headings tend to have less words when compared to regular sentences and paragraphs.
  - Text Case: Headings mostly use title case, while sometimes they are in upper case as well. This variable tells whether the text is in upper case

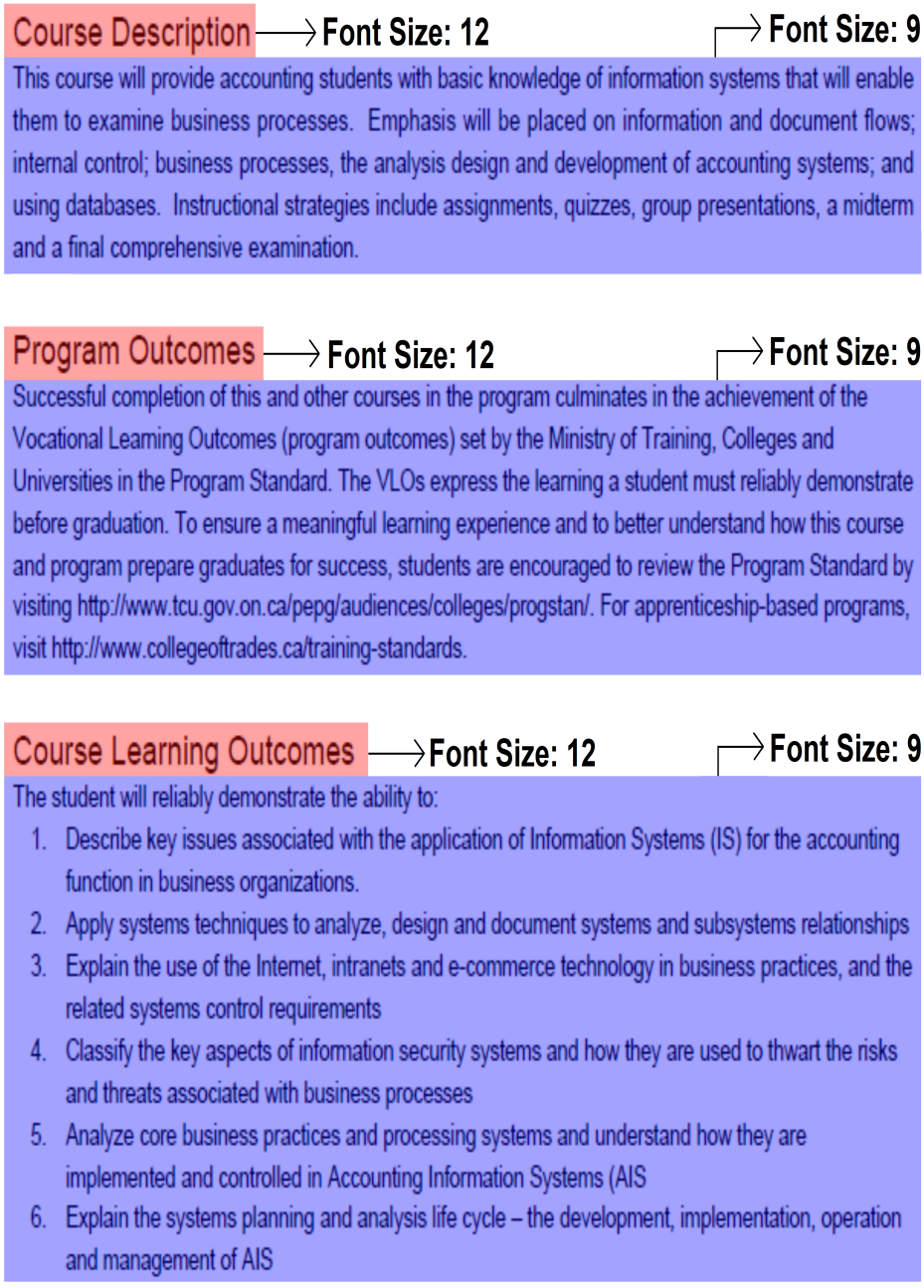


Fig. 2. Font Size Threshold Assumption Example

(all letters in upper case), lower case (all letters in lower case), title case (first letter of all words in uppercase) or sentence case (only the first letter of the text in uppercase).

- Features From Parts of speech(POS) Tagging: POS Tagging is the process of assigning parts of speech (verb, adverb, adjective, noun) to each word, which are referred to as tokens. The text from each data point is first tokenized and then each token is assigned a POS label [9].

The POS frequencies provides the model with information on the grammatical aspect of the text and can be used to exploit the frequency of these labels in a text to identify headings and contribute to the accuracy of the model. For example, headings tend to have no verbs in them, though some might have them but absence of verbs increases the probability of the text being an heading. All frequency data collected from POS tagging is analysed in the feature selection process to differentiate between useful and irrelevant features collected through it. The frequency for each POS label is calculated and used to calculate the frequency of each POS tag in the text for each data point. These frequencies serve as potential features for the model.

All these features brings the count of total number of features generated using the text to 11, 9 from POS tagging the text and 2 using its physical properties.

Table 1. List of all features

All features are integers, except for *Bold or Not* and *Font Threshold Flag* which are binary.

Feature Name	**Description**
Characters	Number of characters in the text.
Words	Number of words in the text.
Text Case	Assumes the value 0,1,2 or 3 depending on the text being being in lower case, upper case, title case or none of the three respectively.
Bold or Not	Assumes the value 1 or 0 depending on the text being bold or not.
Font Threshold Flag	Assumes the value 1 or 0 depending on the font size of the text being greater than the threshold or not.
Verbs	Number of verbs in the text.
Nouns	Number of nouns in the text.
Adjectives	Number of adjectives in the text.
Adverbs	Number of adverbs in the text.
Pronouns	Number of pronouns in the text.
Cardinal Numbers	Number of cardinal numbers in the text.
Coordinating Conjunctions	Number of coordinating conjunctions in the text.
Predeterminers	Number of predeterminer in the text.
Interjections	Number of Interjections in the text.

3.3 Feature Selection

After pre-processing, 14 training features are established. There is a need to select the top features for building each individual model with maximum accuracy. Table 1 lists all the features we are choosing from. To achieve this we used *Recursive feature elimination* with Cross-Validation (RFECV), which recursively removes weak attributes/features and uses the model accuracy to identify features that are contributing towards increasing the predictive power of the model[8]. The selection process is performed using the machine learning library, "scikit-learn".

Cross validation is done by making 10 folds in the training set where one feature is removed per iteration. As per this analysis the accuracy does not increase on choosing to train the Decision Tree classifier with more than the following seven features:

- Bold or Not
- Font Threshold Flag
- Number of words
- Text Case
- Verbs
- Nouns
- Cardinal Numbers

The same process is repeated for all the classifiers and their individual set of chosen features are listed in Table 2.

3.4 Grid Search

Tuning each classifiers parameters for optimal performance is performed using accuracy from cross validation as a measure. We use various combinations of classifiers parameters and choose the combination with the best cross validation accuracy. This process is performed on various classifiers to choose their corresponding parameters. The description along with the final selected tuning parameters for each classifier used in this research are discussed in the next section.

Table 2. Selected features for each classifier

Classifier Name	**Selected Features**
Decision Tree	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers
SVM	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adjectives, Adverbs
k-Nearest Neaighbors	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Cardinal Numbers, Coordinating Conjunctions
Random Forest	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adverbs, Cardinal Numbers, Coordinating Conjunctions
Gaussian Naive Bayes	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Cardinal Numbers, Coordinating Conjunctions
Quadratic Discriminant Analysis	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Coordinating Conjunctions
Logistic Regression	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adverbs, Coordinating Conjunctions
Gradient Boosting	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers
Neural Net	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers

3.5 Training

After the most suitable features and parameters for each classifier have been selected, we can proceed with training the classfiers using scikit-learn [18].

Decision Tree Decision trees are the most widely used amongst classifiers as they have a simple flow-chart like structure starting from a root node. It branches off to further nodes and terminating at a leaf node. At each non-leaf node a decision is made, which selects the branch to follow. The process continues to the point where a leaf node is reached, which contains the corresponding decison[14]. Gini impurity is used as a measure for quality of a split, which tells if the split made the dataset more pure. Using Gini makes it computationally less expensive as compared to entropy which involves computation of logarithmic functions. The "best" option for strategy chooses the best split at each node. The minimum number of samples required to split an internal node is set to 2 and the minimum number of samples needed to be at a leaf node is set to 3.

The code snippet for training this classifier with the chosen parameters is given in Box 1

Box 1: Code Snippet for Training Decision Tree Classifier

```
treeclf = DecisionTreeClassifier(criterion = 'gini', splitter = 'best',
min_samples_split = 2, min_samples_leaf = 3)
treeclf = treeclf.fit(traindata, truelabels)
```

Support Vector Machine (SVM) It is a classifier that uses multi-dimensional hyperplanes to make classification. SVM also uses kernel functions to transform the data in sucha way that it is feasible for the hyperplane to effectively partition classes[15]. The kernel used is radial basis function(rbf), degree of the polynomial kernel function is set to 3 and gama is set to "auto". The shrinking heuristics were enabled as they speed up the optimization. Tolerance for stop criteria is set to $2e-3$ and 'ovr'(one vs rest) decision function is chosen for decision function shape. The code snippet for training this classifier with the chosen parameters is given in Box 2.

Box 2: Code Snippet for Training Support Vector Machine Classifier

```
svmclf = SVC(kernel='rbf', degree=3, gamma='auto', shrinking=True,
tol=0.002, decision_function_shape='ovr')
svmclf = svmclf.fit(traindata, truelabels)
```

k-Nearest Neighbors The main idea behind k-Nearest Neighbors is that it takes into account the class of its neighbors to decide how to classify the data point under consideration. Each neighbors class is considered as their vote towards that class and the class with the most votes is assigned to that data point[17]. The number of neighbours used to classify a point is set to 10. Each neighbours are weighed equally as weights is set to 'distance'. Minkowsky distance function used as the distance metric. The code snippet for training this classifier with the chosen parameters is given in Box 3.

Box 3: Code Snippet for Training k-Nearest Neighbors Classifier

```
neighclf = KNeighborsClassifier(n_neighbors = 10, weights = 'distance',
metric = 'minkowski')
neighclf = neighclf.fit(traindata, truelabels)
```

Random Forest This classifier works by choosing random data points from the training set and creating a set of decision tress. The final decision regarding the class is made by aggreggation of the outputs from all the trees[19]. The number of trees in the forest is set to 2 and 'gini' is used as a measure for quality of a split. The maximum depth of trees is set to 5 and the maximum number of features to be considered while searching for the best split is se to 'auto'. The minimum number of samples required to split an internal node is set to 2 and the minimum number of samples needed to be at a leaf node is set to 3. The number of parallel jobs to running for both fit and predict is set to 1. The code snippet for training this classifier with the chosen parameters is given in Box 4.

Box 4: Code Snippet for Training Random Forest Classifier

```
RandomForestClassifier(n_estimators = 2, criterion = 'gini', max_depth
= 5, max_features='auto', min_samples_split=2, min_samples_leaf=3,
n_jobs=1)
rndForstclf = rndForstclf.fit(traindata, truelabels)
```

Gaussian Naive Bayes This classifier works by using Bayesian theorem with assumption of strong independence between the predictors(features). It is very useful for large data sets as it is quite simple to build and has no complicated iterative parameters[22]. This classifier does not have much to set when it comes to configuring parameters. Prior probabilities of the classes is set to [0.5, 0.5] as the number of headings is less as compared to other text. The code snippet for training this classifier with the chosen parameters is given in Box 5.

Box 5: Code Snippet for Training Gaussian Naive Bayes Classifier

```
gaussianclf = GaussianNB(priors = [0.5, 0.5])
gaussianclf = gaussianclf.fit(traindata, truelabels)
```

Quadratic Discriminant Analysis It works under the assumption that the measurements for each class are normally distributed while not assuming the covariance to be identical for all the classes. Discriminant analysis is used to choose the best predictor variable(s) and is more flexible than linear models making it better for a variety of problems[24]. Prior probabilities of the classes is set to [0.5, 0.5] as the number of headings is far less as compared to other text. The threshold used for rank estimation is set to $1e-4$. The code snippet for training this classifier with the chosen parameters is given in Box 6.

Box 6: Code Snippet for Training Quadratic Discriminant Analysis Classifier

```
quadclf = QuadraticDiscriminantAnalysis(priors = [0.5, 0.5], tol = 0.0001)
quadclf = quadclf.fit(traindata, truelabels)
```

Logistic Regression It is a discriminative classifier, therefore it works by discriminating amongst the different possible values of the classes[23]. Penalization method is set to l2. The tolerance for stopping criteria is set to $2e-4$. The parameter 'fit_intercept' is set to true adding a constant to the decision function. The optimization solver used is 'liblinear' and the maximum number of iterations taken for the solvers is set to 50. Multiclass is set to 'ovr' fitting a binary problem for each label. The number of CPU cores used for parallelizing over classes is set to 1. The code snippet for training this classifier with the chosen parameters is given in Box 7.

Box 7: Code Snippet for Training Logistic Regression Classifier

```
logisticRegr = LogisticRegression(penalty=l2, tol=0.0002, fit_intercept = True, solver='liblinear', max_iter=50, multi_class = 'ovr', n_jobs=1)
logisticRegr = logisticRegr.fit(traindata, truelabels)
```

Gradient Boosting This classification method uses an ensemble of weak prediction models in a stage wise manner. In each stage, a weak model is introduced to make up for the limitations of the existing weak models[21]. The loss function to be optimized is set as 'deviance' and learning rate is set to 0.1. The minimum number of samples required to split an internal node is set to 2, the minimum number of samples needed to be at a leaf node is set to 1 and maximum depth of the individual regression estimators set to 3. The number of boosting stages is set to 150 and the measure of quality of a split is set to 'friedman_mse'. The code snippet for training this classifier with the chosen parameters is given in Box 8.

Box 8: Code Snippet for Training Gradient Boosting Classifier

```
grdbstcf = GradientBoostingClassifier(loss = 'deviance', learning_rate = 0.1, min_samples_split = 2, min_samples_leaf = 1, max_depth = 3, n_estimators = 150, subsample = 1.0, criterion = 'friedman_mse')
grdbstcf = grdbstcf.fit(traindata, labels)
```

Neural Net This classifier works by imitating the neural structure of the brain. One data point is processed at a time and the actual classification is compared to the classification made by the classifier. Any errors recorded in the classification process are looped back into algorithm to improve classification performance in future iterations[27, 25]. The classifier is configured to have one hidden layer with 100 units. The activation function used for the hidden layer is 'tanh'. The solver used for weight optimization is 'lbfgs'. The batch rate is set to 'auto' and the initial learning rate is set to 0.001. The parameter 'max_iter' is set to 300, which for 'adam' solver defines the number of epochs. Sample shuffle is set to true, which enables sample shuffling in each iteration. The exponential decay rates for estimates of the first and second moment vector is set to 0.9 and 0.999 respectively. The code snippet for training this classifier with the chosen parameters is given in Box 9.

Box 9: Code Snippet for Training Neural Net Classifier

```
nurlntclf = MLPRegressor(hidden_layer_sizes = (100, ), activation =
'tanh', solver = 'lbfgs', learning_rate = 'invscaling', batch_size = 'auto',
learning_rate_init = 0.001, max_iter = 300, shuffle = True, beta_1 = 0.9,
beta_2 = 0.999)
nurlntclf = nurlntclf.fit(traindata, labels)
```

4 Evaluation

Evalution is going to be based on the following measures:

4.1 Training and Prediction Time

When dealing with a large number of documents, the time required to train a model and make predictions is important and is dependant on the type of classifier used, the number of features and the amount of data points. In this research all classifiers are trained using the same number of features and data points, therefore 'time taken' provides a good measure of variations in training and prediction speed associated with each different classifier being used. Of note, the training time for a classifier should be considered in context, as training only needs to be performed once and can be saved for later use. Therefore, a model that takes a long time to train can still be practical so long as it does not take a lot of time to make predictions.

4.2 Confusion matrix

Confusion matrix is used to represent a classifier's performance with respect to a data set having known true values, an example representation of such matrix

is shown in Fig 3. A confusion matrix is generated for each classifier, serving as a summary of prediction results for it and further used to calculate the evaluation parameters. These calculated evaluation parameters are used to compare these classifiers to each other and are mentioned below [20].

n = 200	Predicted NO	Predicted YES	
Actual No	600 (TN)	121 (FP)	721
Actual Yes	79 (FN)	1200 (TP)	1279
	679	1321	

Fig. 3. Confusion Matrix Example

- Sensitivity: It is also known as recall or true positive rate and is the odds of getting a positive test outcome given a positive case.
- Specificity: It is also known as true negative rate and is the odds of getting a negative test outcome given a negative case.
- Precision: It gives the odds of that the outcome classified as positive is actually positive.
- F1 Score: It provides a score for the balance between precision and recall.
- Total Accuracy: It is the measure of how many predictions in total are correct or the measure of trueness of the results.

These parameters are calculated as follows:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1Score = \frac{2TP}{2TP + FP + FN}$$

$$TotalAccuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where,
TP means true positives,
TN means true negatives,
FP means false positives, and
FN means false negatives

4.3 AUC

A receiver operating characteristics(ROC) curve is used to visualize the trade-offs between sensitivity and specifity. These graphs are used for performance based selection of classifiers. The graph can be reduced to a numerical measure, AUC(or AUROC) which is the area under the ROC graph with values ranging from 0 to 1[26].

5 Test Results

5.1 Training and Prediction Time

Fig 4 shows time required for training and making predictions using these classifiers. Time shown is average of 10 observations, which is done to reduce the effect of programs running in the background on the comparison.

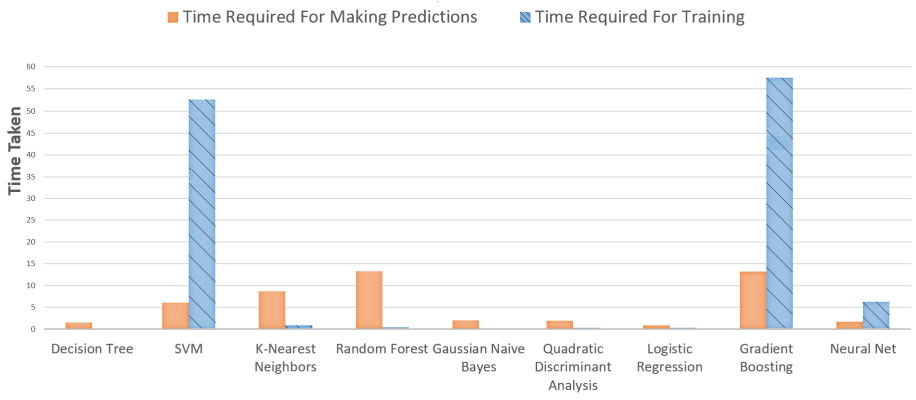


Fig. 4. Time (in seconds) required to train classifiers and run predictions on test data

5.2 Confusion Matrix Based Evaluation

Table 3 shows the results of this evaluation. Gradient Boosting classifier shows the best specificity, precision and net accuracy. Gaussian Naive Bayes shows the best sensitivity and SVM shows the best F1 score.

5.3 AUC

Table 4 shows the AUC scores for the classifiers used in this research. The discussion section provides more information on how we used AUC score to select the best classifier.

Table 3. Classifier Accuracy

Highest Value For Each Measure is Bold

Classifier	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**Accuracy**
Decision Tree	0.972	0.944	0.897	0.933	95.37 %
SVM	0.970	0.933	0.881	0.930	95.09 %
K-Nearest Neighbors	0.940	0.932	**0.907**	0.923	94.82 %
Random Forest	0.971	0.935	0.882	0.926	94.85 %
Gaussian Naive Bayes	0.960	0.911	0.847	0.909	93.47 %
Quadratic Discriminant Analysis	0.963	0.912	0.848	0.910	93.52 %
Logistic Regression	0.956	0.919	0.855	0.903	93.17 %
Gradient Boosting	**0.981**	**0.946**	0.902	**0.940**	**95.83** %
Neural Net	0.980	**0.946**	0.901	0.939	95.80 %

Table 4. AUC Values for all Classifiers

Classifier	**AUC**
Decision Tree	**0.98**
SVM	0.97
K-Nearest Neighbors	0.96
Random Forest	0.97
Gaussian Naive Bayes	0.96
Quadratic Discriminant Analysis	0.96
Logistic Regression	0.95
Gradient Boosting	**0.98**
Neural Net	**0.98**

6 Discussion & Future Work

6.1 Overall Results

We recorded the time (in seconds) required for training each classifier and also time for making predictions as shown in Fig 4. Time taken by a classifier to make predictions is important when processing documents in bulk as it can increase the processing time. Time taken to train a classifier only has to be done once therefore it is not given that much importance. The Decision Tree Classifier took the least time for training while Gradient Boosting took the most. On comparing the prediction time Logistic Regression takes the least time and Random Forest takes the most. While prediction time is not the most important factor while choosing a classifier we take it into consideration when two classifiers are performing approximately the same.

The top three classifiers based on net accuracy are Decision Tree, Gradient Boosting, and Neural Network, however classifier selection can not solely rely on accuracy [28, 29]. Therefore, we also weigh the metrics like AUC, F1 score, sensitivity, and specificity to choose the best suited classifier for detecting headings.

The top three classifiers in terms of F1 score, precision, sensitivity and specificity are Decision Tree, Gradient Boosting, and Neural Network and the top 3 in terms of AUC as shown in Table 4 are again Decision Tree, Gradient Boosting, and Neural Network. The system is going to be dealing with documents in bulk and the prediction time for Decision Tree is better when compared to both Gradient Boosting and Neural Network. Therefore, we would be choosing our configuration of the Decision Tree for making the classifications.

6.2 Testing The Generalizability

Testing the chosen classifier on a general set of documents is important to show that it performs well on documents other than course outlines. We tested the chosen Decision Tree classifier on 12,919 data points collected from documents like reports and articles[2]. These data points were manually tagged using a survey. All the participants were graduate students from computer science department and were asked to point out headings and subheadings in the documents. Table 5 shows the results which are equivalent as compared to when tested on course outlines.

[2] Repository available at: https://github.com/sahib-s/Generalizability/

Table 5. Test Results For General Set

Category	Value
Total Data points	12919
Sensitivity	0.928
Specificity	0.966
Precision	0.964
F1 SCORE	0.946
Accuracy	94.73 %
AUC	0.97

Table 6. Pearson Correlation Coefficient Between Each Feature Used in the Selected Classifier and Final Decision Labels

Feature Name	Pearson Correlation Coefficient
Bold or Not	0.7022
Font Threshold Flag	0.2385
Words	0.1389
Verbs	0.1229
Nouns	0.1207
Cardinal Numbers	0.1201
Text Case	0.0660

6.3 Analysing The Results

The discussed configuration of Decision Tree is best suited to detect heading as discussed in Section 4.5.1. Analyzing the contribution of each feature towards the final decision made by the classifier is also important to understand the implications of the results. Table 6 shows the pearson correlation coefficient for all the features used in the selected classifier and final decision label. The list is in descending order of pearson correlation coefficient, therefore the top feature in the table contribute the most towards the final decision. Each feature was removed from the classifier one at a time and drop in evaluation metrics also verify the order of contribution presented by using the pearson correlation coefficient. Therefore, the top three contributing features are the ones that rely on the physical attributes of the text.

6.4 Extending The Classifier

The extension of this work includes tagging of multiple labels like heading, paragraph text, header/footer text and table text. While classifying paragraph text is possible using the existing features, for properly classifying table text and header/footer text more data features are necessary. We are currently looking features from our white space detection approach discussed in chapter 5 and

bounding box data from PDF to XML conversion to provide the model with what it needs to make this classification.

7 Conclusion

This research has provided a structured methodology and systematic evaluation of a heading detection system for PDF documents. The detected headings provide information on how the text is structured in a document. This structural information is used for extracting specific text from these documents based on the requirements of the field of application. This supervised learning approach has demonstrated good results and we are currently applying our configuration of the Decision Tree classifier in the field of post-secondary curriculum analysis to identify headings and extract learning outcomes from course outlines for a research being conducted at DATALAB, Lakehead University, Canada.

Acknowledgment

This research would not have been possible without the financial support provided by Ontario Council on Articulation and Transfer (ONCAT) through Project Number-2017-17-LU. We would also like to express our gratitude towards the datalab.science team and Andrew Heppner for their support.

References

1. Khusro, Shah, Asima Latif, and Irfan Ullah. "On methods and tools of table detection, extraction and annotation in PDF documents." Journal of Information Science 41.1 2015.: 41-57.
2. Jiang, Deliang, and Xiaohu Yang. "Converting PDF to HTML approach based on Text Detection." Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. ACM, 2009.
3. Lakehead University. Learning Objective Automated Gap Analysis. 2018, www.loaga.science/.
4. Goslin, Kyle, and Markus Hofmann. "Cross Domain Assessment of Document to HTML Conversion Tools to Quantify Text and Structural Loss during Document Analysis." Intelligence and Security Informatics Conference (EISIC), 2013 European. IEEE, 2013.
5. Rahman, Fuad, and Hassan Alam. "Conversion of PDF documents into HTML: a case study of document image analysis." Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on. Vol. 1. IEEE, 2003.
6. Manabe, Tomohiro, and Keishi Tajima. "Extracting logical hierarchical structure of HTML documents based on headings." Proceedings of the VLDB Endowment 8.12 2015.: 1606-1617.
7. El-Haj, Mahmoud, et al. "Detecting document structure in a very large corpus of UK financial reports." 2014.: 1335-1338.
8. Guyon, I., Weston, J., Barnhill, S. et al. Machine Learning 2002. 389422.

9. Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.
10. Bienz, Tim, Richard Cohn, and Adobe Systems (Mountain View, Calif.). Portable document format reference manual. Reading, MA, USA: Addison-Wesley, 1993.
11. Google. Custom Google Search API. Google, Google, 2018, developers.google.com/custom-search/json-api/v1/overview.
12. Shinyama, Yusuke. "PDFMiner: Python PDF parser and analyzer."
13. Goyvaerts, Jan, and Steven Levithan. Regular expressions cookbook. O'reilly, 2012.
14. Lior, Rokach. Data mining with decision trees: theory and applications. Vol. 81. World scientific, 2014.
15. Ben-Hur, Asa, and Jason Weston. "A users guide to support vector machines." Data mining techniques for the life sciences. Humana Press, 2010. 223-239.
16. Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.
17. D. T. Larose, "k-nearest neighbor algorithm" in Discovering Knowledge in Data: An Introduction to Data Mining, Hoboken, NJ, USA:Wiley, pp. 90-106, 2005.
18. Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.
19. Louppe, Gilles. "Understanding random forests: From theory to practice." arXiv preprint arXiv:1407.7502 2014.
20. Parikh, Rajul, et al. "Understanding and using sensitivity, specificity and predictive values." Indian journal of ophthalmology 56.1 2008.: 45.
21. Ridgeway, Greg. "The state of boosting." Computing Science and Statistics 1999.: 172-181.
22. Rish, Irina. "An empirical study of the naive Bayes classifier." IJCAI 2001 workshop on empirical methods in artificial intelligence. Vol. 3. No. 22. IBM, 2001.
23. Pohar, Maja, Mateja Blas, and Sandra Turk. "Comparison of logistic regression and linear discriminant analysis: a simulation study." Metodoloski zvezki 1.1 2004.: 143.
24. Sueyoshi, Toshiyuki. "DEA-discriminant analysis: methodological comparison among eight discriminant analysis approaches." European Journal of Operational Research 169.1 2006.: 247-272.
25. Graupe, Daniel. Principles of artificial neural networks. Vol. 7. World Scientific, 2013.
26. Fawcett, Tom. "An introduction to ROC analysis." Pattern recognition letters 27.8 2006.: 861-874.
27. Mago, Vijay Kumar, ed. Cross-Disciplinary Applications of Artificial Intelligence and Pattern Recognition: Advancing Technologies: Advancing Technologies. IGI Global, 2011.
28. Huang, Jin, and Charles X. Ling. "Using AUC and accuracy in evaluating learning algorithms." IEEE Transactions on knowledge and Data Engineering 17.3 2005.: 299-310.
29. Ling, Charles X., Jin Huang, and Harry Zhang. "AUC: a better measure than accuracy in comparing learning algorithms." Conference of the canadian society for computational studies of intelligence. Springer, Berlin, Heidelberg, 2003.