Stockholms universitet

Classifying BBC News Articles with Random Forest and eXtreme Gradient Boosting.

Leon Voss Gustavsson

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2022:5
Matematisk statistik
Juni 2022

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2022:5**
http://www.math.su.se

Classifying BBC News Articles with Random Forest and eXtreme Gradient Boosting.

Leon Voss Gustavsson*

June 2022

Abstract

With the modern invention and improvement of machine learning methods, the scope of their use is increasing, with new fields of application. One of these fields is natural language processing (NPR). Within NPR one task is text classification. In this thesis we will classify news articles from BBC as belonging to one of the categories sport, politics, entertainment, business or tech. The dataset that we use consists of 2225 observations/articles. Classifying the articles will be done trough analyzing word frequencies from all articles. Furthermore, we will compare different ways of selecting these exact words and study how many words are needed to reach satisfactory results. We will then use Random forest as well as the eXtreme Gradient Boosting (XGBoost) method. In the end we mainly end up with three different factors to evaluate the models by; the number of words used, the computation time, and the prediction accuracy. Despite the reputation of XGBoost, random forest produces somewhat higher prediction accuracy (a fraction of 0.963 correct classification with 10 fold cross validation, compared to 0.957 for XGBoost), and it also takes considerately less time to train. Furthermore, at around 500 words we start to see convergence towards high prediction accuracy for random forest as well as for XGBoost. After having reached such a high degree of prediction accuracy we investigate which words where the most important for classifying an article. Not surprisingly we found that "coach" was meaningful for classifying an article as sport, "shares" for business, and "film" for entertainment and so on. In the end we thus obtained models that where both interpretable and reached high prediction accuracy.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: leon.voss.g@gmail.com. Supervisor: Ola Hössjer, Nils Engler.

Contents

1 Introduction

During the 21st century, machine learning methods have been drastically improving. This has a lot to do with increasing amounts of data being collected. In the supervised learning field of machine learning, the models are directly trained using labeled data and can then hopefully perform well on new data that have not been seen before.

This can be useful for a wide range of purposes. For this thesis, we will specifically try to classify news articles as belonging to one of five different categories (business, tech, entertainment, sport, and politics). All this is made possible through a dataset from BBC consisting of 2225 articles. The aim is then to train a model that will reach the best possible accuracy when classifying articles. In this process, we will investigate the relationship between how many words from the articles are needed for a well-performing model and the impact this has on computation time. Furthermore, we will investigate which words are most important for classifying an article as belonging to a given class.

There is a wide range of machine learning methods available for this task but in this thesis, only random forest and XGBoost are used. Both are tree based methods that are known to perform well, especially XGBoost. However, despite XGBoosts popularity, as we will see, random forest performs better on our particular BBC dataset.

The thesis is organised as follows, we will present the theory section (section 2) about why and how our chosen models work. There we will also see the importance of some parameters that we will have to tune for optimal performance, this will be discussed later in section 4. Then we shall take a look at the original dataset in section 3 and see how we manipulate it to create subsets of different sizes. In section 5 we will then present the results, and this will be followed up by a discussion about these results in section 6.

2 Theory

In this section, we will cover the theory behind the machine learning models used for analyzing our dataset. Mostly we will follow the theory from the book The Elements of Statistical Learning(Hastie et al., 2017) [4]. One notable exception is the theory of section 2.7 which is based on the paper XGboost: A Scalable Tree Boosting System by Chen Guestrin (2016) [2]. Inspiration for notation regarding the XGboost section has been taken from Classification of Music Genres with eXtreme Gradient Boosting by Jesper Muren (2019) [6].

2.1 The Bias-Variance Tradeoff

2.1.1 The Bias-Variance decomposition

This section will be based on section 2.9 in Hastie et al (2017) [4].

A reoccurring theme during this thesis will be choosing the right parameters, parameters that will achieve optimal performance. And to a high degree, choosing the right values amounts to reducing the expected prediction error (EPE). In other words, we want a model that will have a low expected error for data it has not yet seen.

We denote the response variable Y, the vector of inputs X and the prediction model for the expected value $E(Y|X) = f(X)$ of the response by $\hat{f}(X)$. To measure how well $\hat{f}$ performs we use different loss functions $L(Y, \hat{f}(X))$.

If we were to consider the EPE with squared error loss as a loss function we would for a data point x_0 get

$$\text{EPE}(x_0) = E[(Y - \hat{f}(x_0))^2].$$

Doing some algebraic manipulation leads to the following decomposition;

$$\text{EPE} = \sigma^2 + [\text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\hat{f}(x_0))], \tag{1}$$

where Bias $= E[\hat{f}(x_0) - f(x_0)]$. So the bias squared stands for the error originating in the difference between the "true" f and systematic departures from this function in the implemented model $\hat{f}$. The variance from (1) stands for how much our model will vary across different data sets. In the data there will also be some inherent randomness beyond our control, hence the term $\sigma^2 = \text{Var}(Y|X)$.

Making the model $\hat{f}$ more complex reduces the bias, however, too complex would mean that it does not work on new data. In other words, it would have high variance $\text{Var}(\hat{f}(x_0))$. This phenomenon will be reoccurring and it is called overtraining.

In conclusion, the best model have to balance between bias and variance for optimal performance. How is this done? One way is through K-fold cross validation.

2.1.2 *K*-fold cross validation

This section will be based on section 7.1 in Hastie et al (2017) [4].

The idea here is to split the data into K equally sized folds. A model is then trained on $K - 1$ of these folds and then tested on the kth fold (also called the hold-out-set). The procedure is repeated so that all K folds of the data have been used as both hold-out and training sets, i.e. $k = 1, \ldots, K$. We then combine the K estimators.

We write this using an indexing function $\kappa : [1, ..., N] \to [1, ..., K]$ that denotes the fold to which each observation $i = 1, \ldots, N$ ends up in. Furthermore let $\hat{f}^{-k}(x)$ be the fitted function trained on all folds but the kth. The K-fold cross validation prediction error is

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i)).$$

As earlier mentioned choosing the right parameters α for our models will be crucial for this work. We now have a method to find the optimal parameters $\hat{\alpha}$.

Let $f(x, \alpha)$ be a set of models that is indexed by α. Denote $\hat{f}^{-k}(x, \alpha)$ as the αth model fit with the kth fold removed. We thus get

$$CV(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha)).$$

This function provides a curve of accuracy depending on α. We can then choose $\hat{\alpha}$ simply as the value of α that minimizes $CV(\hat{f}, \alpha)$. This would give us the final model $f(x, \hat{\alpha})$.

However, sometimes other aspects than accuracy also have to be considered, such as computation cost. This will be further discussed later on in this thesis.

2.2 Decision Tree

This section will be based on Section 9.2 in Hastie et al (2017) [4]. There exist different algorithms for creating decision trees. In this thesis, we will only be using and describing the CART method.

As we shall see trees are easy to interpret and understand. However, they have one problem which is inaccuracy. And since this work aims to provide the best prediction possible, decision trees are not a suitable method. Nevertheless, decision trees are a stepping stone to understanding both random forest and XGBoost as both are tree based methods. Hence we shall cover the underlying theory below.

Consider a regression problem where we have N observations used as training data (x_i, y_i) for $i = 1, ..., N$. Furthermore we have p predictors so $x_i = (x_{i1}, x_{i2}, ...x_{ip})$. Our aim is to make a prediction of the response y_i given x_i.

We do this by dividing the space of possible values (the prediction space) for $x = x_1, ..., x_p$ into M separate regions $R_1, ..., R_M$. For an observation that falls in a region R_m, we make the prediction as the mean of the response values from the training data that lies in R_m, call this constant γ_m.

The regression model can thus be written as

$$f(x) = \sum_{m=1}^{M} \gamma_m I(x \in R_m). \tag{2}$$

Here, $I(A)$ denotes an indicator function for the event A.

In the case of classification we instead make the prediction based on the majority vote in the region. The question then arises of how to grow a regression tree.

If we are to minimize the sum of squares $\sum(y_i - f(x_i))^2$ it is computationally infeasible to consider all possible separations of the feature space into M boxes. Therefore recursive binary splitting is used. This is a *top-down, greedy* method. It is *top-down* since it starts at the top of the tree, and then, one step at a time, splits the predictor space. It is *greedy* as the best partition of the feature space is made at each split. This means that future splits are not considered in the choice of new regions.

Consider a splitting variable j and a split point s. In the first step, two regions would be created

$$R_1(j,s) = \{x \mid x_j \leq s\} \text{ and } R_2(j,s) = \{x \mid x_j > s\}.$$

We want the regions that give the greatest reduction in sum of squares, this results in the following minimization problem

$$\min_{j,s} \Big[\min_{\gamma_1} \sum_{x_i \in R_1(j,s)} (y_i - \gamma_1)^2 + \min_{\gamma_2} \sum_{x_i \in R_2(j,s)} (y_i - \gamma_2)^2 \Big].$$

The inner minimization is solved by

$$\hat{\gamma}_1 = \text{ave } (y_i \mid x_i \in R_1(j,s)) \text{ and } \hat{\gamma}_2 = \text{ave } (y_i \mid x_i \in R_2(j,s)).$$

This process is continued until we reach a stop criterion. One possibility is to stop when a predetermined number of regions are created. Or that a new split does not surpass a predefined threshold for a decrease in sum of squares.

Both of these stop criteria are problematic. The first since it is hard to know how many regions we want beforehand. The second since a worthless split might lead to a second split that is desirable.

In general, caution is needed when growing a tree. The reason is that a too large tree will overfit the model to the training data (high variance) and will not generalize well to new data. A small tree on the other hand might miss some important structure, and thus also make a poor prediction (high bias).

Moving from regression to classification where we have K outcomes is straightforward. The main difference is the criterion for which we did the

split. Now we need another criterion than the sum of squares to be minimized. Commonly used criteria are the Gini index and the deviance.

Before presenting these criteria let N_m be the number of observations in region R_m. Also the proportion of class k in node m (representing region R_m) is written as

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k).$$

The Gini index is then given by

$$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

and the deviance, also called Cross-entropy, is given by

$$-\sum_{k=1}^{K} \hat{p}_{mk}(\log(\hat{p}_{mk})).$$

There also exists a missclassification rate. It is the proportion of observations of a node that do not belong to the majority class $k(m)$. This is written as

$$1 - \hat{p}_{mk(m)}.$$

The first two criteria are differentiable and thus better suited for numerical optimisation, something that will be of importance later.

One way to try to solve the problem that we discussed earlier, that of tree depth (number of nodes), is through pruning. However, this generally will not produce accuracy on the same level as that of the other models we will use. Hence I will not discuss this concept. But there are some key takeaways from this section that are important for the coming models. A large tree has low bias and high variance, and a shallow tree has high bias and low variance.

In figure 1 we see how a classification tree based on two predictors, how many times the word investment and bank occur, might work. In the end, the yes/no question would lead the tree to classify the data as belonging to one of the three classes Sport, Politics, or Economy.

Note that this tree is not realistic for the prediction of the earlier mentioned data. The reason is that the tree is too shallow and contains too few predictors. It does however provide a good example of how a tree gives rise to a classifier.

2.3 Bagging

This section will be based on Section 8.7 in Hastie et al (2017) [4]. Before introducing the model of random forest we need a discussion about bagging.

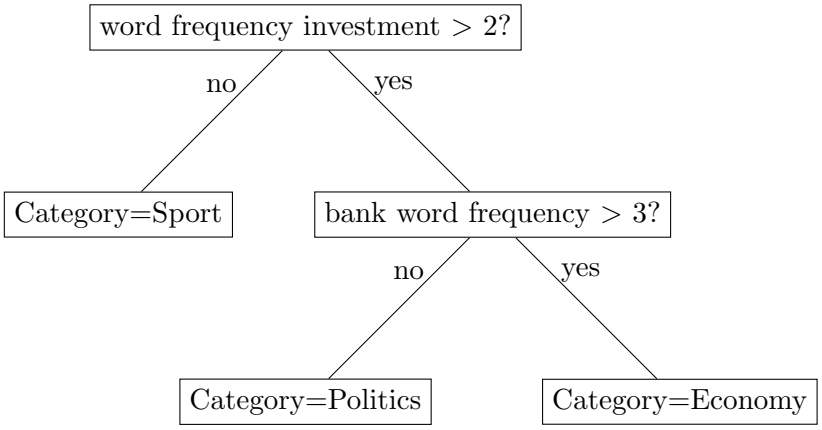

Figure 1: Visualizing a decision tree with 3 nodes

Consider a regression problem where we fit a model to our training data

$$\boldsymbol{Z} = \big\{(x_1, y_2), (x_2, y_2), ..., (x_N, y_N)\big\}.$$

This fitted model generates the prediction $\hat{f}(x)$ at an input x. Now we take B bootstrap samples, meaning we resample the training data from our original sample so that each bootstrap sample is of the same size as our original set. Denote the b:th of these samples by $\boldsymbol{Z}^{*b}$. For each $b = 1, 2, ...B$ we fit our model and get the prediction $\hat{f}^{*b}(x)$. The bagging estimate is then defined by

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

2.4 Random Forest

This section will be based on Section 15 in Hastie et al (2017) [4].

As discussed earlier we saw that decision trees, if grown large enough, have low bias. However, they are often quite noisy and won´t generalize well to new data. The idea is then to combine the low bias with variance reduction. This is where bagging comes in.

Each tree in bagging is identically distributed. Thus an expectation of any one of them is the same as the average of B of them. Furthermore, the variance of the sample mean of B identically distributed variables with pairwise correlation ρ is equal to

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{3}$$

When looking at this expression we can directly see that the second expression goes towards zero as B gets larger, thus the variance is reduced.

So bagging increases the performance of the model through variance reduction. But despite that, a greater reduction is still possible. And this is where random forest comes in. The reduction is done by lowering the correlation ρ between the trees.

Having a look at the first expression in (3) we see that reducing B does not affect this term. Instead what we can do is to reduce ρ. Note that ρ in our case is the correlation factor between the trees.

So if we, when growing the trees, only consider a subset with m of the p predictors in each split, we will decorrelate the trees. However, having too few predictors considered at each split might result in an oversimplified model. This means that m is a tuning parameter that is best found through K-fold cross validation.

Algorithm 1 Random Forest for Regression or Classification

1: **for** $b = 1$ to B **do**
2: (a) Draw a bootstrap sample $\mathbb{Z}^{*b}$ of size N from the training data.
3: (b) Grow a random forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached.
4: (I) Select m variables at random from the p variables.
5: (II) Pick the best variable/split-point among the m.
6: (III) Split the node into two daughter nodes.
7: **end for**
8: Output the ensemble of trees $\{T_b\}_1^B$
9: To make a prediction at new point x:
10: $Regression : \hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
11: $Classification$: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{rf}^B(x) =$ majority vote $\{\hat{C}_b(x)\}_1^B$.

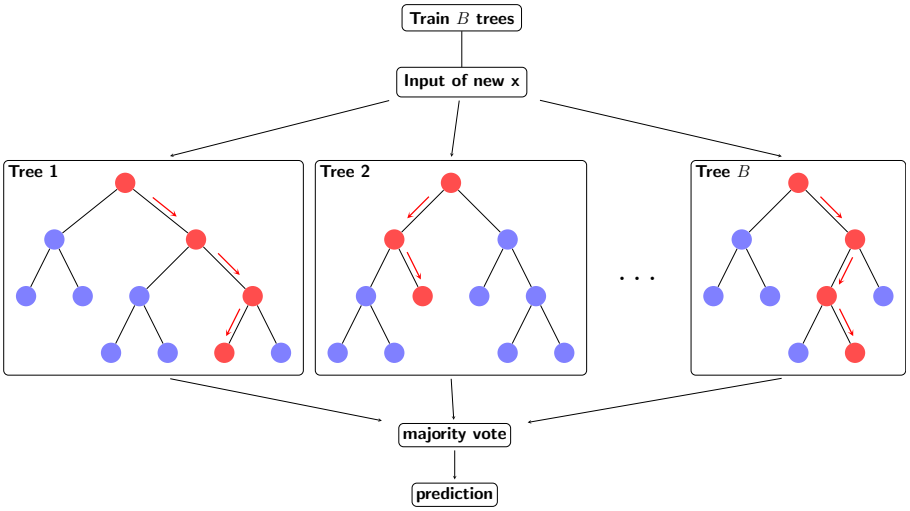

Figure 2: Visualizing a random forest with B trees for classification.

Random forest is robust to datasets consisting of many predictors, something that will be useful for us as we shall see later. The reason for this is that random forest uses only a few percentages of all possible predictors in each step of the tree building. For this reason, computation time is reduced. Moreover, important variables still have a high chance of showing up in the splitting process, thus a useful split will probably be executed somewhere in the trees (as long as the tree is deep enough).

2.5 Boosting Trees

This section will be based on section 10.9 in Hastie et al (2017) [4].

Boosting is a method that will create many weak learners (small trees). The combination of them will then become a competent predictor for new data. This works through a sequential process where more and more trees are added to the model. However, unlike random forest, where the trees do not influence each other, we here grow a new tree to the model, a tree that improves the performance of the earlier mix of trees.

For an intuition about how this works for regression, consider algorithm 2. What we see is a model that grows new trees to the previous errors (residuals). For each new tree, a learning rate λ is used so that in each iteration we (hopefully) make an incremental improvement.

Remember from (2) that a tree with J regions could be expressed as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j). \tag{4}$$

Algorithm 2 Boosting algorithm for regression trees

1: Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2: **for** $m = 1, 2, ..., M$ **do**
3: Fit a tree $\hat{f}^m$ with d splits to the training data $\{(x_i, r_i)\}_{i=1}^N$.
4: Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^m(x).$$

5: Update the residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^m(x).$$

6: **end for**
7: Output the boosted model:

$$f_M(x) = \sum_{m=1}^{M} \lambda \hat{f}^m(x)$$

Here $\Theta = \{R_j, \gamma_j\}_{j=1}^J$ are the parameters of the model. The sum of these trees defines a boosted tree model and can be written as

$$f_M(x) = \sum_{m=1}^{M} T(x; \hat{\Theta}_m).$$

Finding the optimal parameters is done trough the following minimization problem

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j).$$

Thus finding the best regions $R_j = R_{jm}$ and constants $\gamma_j = \gamma_{jm}$ within these regions for $j = 1 \ldots, J_m$ at each step m of the model building, amounts to solving

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)). \tag{5}$$

Solving this gives us $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ for the next tree given $f_{m-1}(x)$. Given that we found the optimal regions we also need to pick the most favorable constant for each region. This is typically straightforward. In the case of regression, $\hat{\gamma}_{jm}$ is equal to the average of all y_i in the region. In classification, it is the most commonly occurring class.

Furthermore, for squared error loss, the solution is simply, as we saw in algorithm 2, the regression tree that best predicts the current residuals $r_i = y_i - f_{m-1}(x_i)$.

2.6 Numerical optimisation with gradient boosting

This section is based on section 10.10-10.12 in Hastie et al (2017) [4].

The idea of gradient boosting is the following. We use a differentiable loss function $L(f)$, for instance the Gini index or deviance from section 2.1. We can then solve (5) through numerical optimization. The loss in predicting y in the training data with $f(x)$ is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)).$$

Then the goal is to minimize $L(f)$ with respect to f, we write this as

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}) \tag{6}$$

with $\mathbf{f} = \{f(x_1), f(x_2), ..., f(x_N)\}^T$ being the prediction at each datapoint. Using numerical optimization, (6) is solved as a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m$$

where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess. Each subsequent $\mathbf{f}_m$ is equal to $\mathbf{f}_{m-1} + \mathbf{h}_m$. Again the idea is that $\mathbf{h}_m$ improves the performance at each step.

2.6.1 Steepest descent

The question arises about how we choose $\mathbf{h}_m$. In steepest descent $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ where $\rho_m > 0$ is a scalar and $\mathbf{g}_m$ is the gradient of $L(\mathbf{f})$ at $\mathbf{f} = \mathbf{f}_{m-1}$. This works since the negative gradient gives the direction at which a function decreases most quickly. Furthermore, the components of the gradient $\mathbf{g}_m$ are

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)} \tag{7}$$

and for $i = 1, \ldots, N$. The optimal step length ρ_m is found trough

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m).$$

We then get

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m.$$

Note that this is a greedy strategy. We only minimize $L(\mathbf{f})$ at each step individually. Future steps are not taken into consideration.

2.6.2 Gradient boosting

Let us take a look at the prediction from boosting trees $T(x_i; \Theta_m)$ with tree components

$$\mathbf{t}_m = \{T(x_1; \Theta_m), T(x_2; \Theta_m), ..., T(x_N; \Theta_m)\}^T.$$

Building these trees while minimizing (5) is as we discussed in section 2.1 a greedy top down approach. So both steepest descent and boosting tree algorithms are greedy approaches. An important consequence is that the predictions $T(x_i; \Theta_m)$ are analogous to the components of the negative gradient (7).

Minimizing $L(y, f(x))$ with respect to $f(x)$, when $f(x)$ is updated according to steepest descent (7) is easily done if we choose a differentiable loss function. However, the gradient is defined only for the training data $x_1, \ldots, x_N$. Since we need a model that generalizes effectively to new data, this approach is flawed. To solve this issue one can do the following. Create a tree $T(x; \Theta_m)$ at the m:th iteration. The predictions of t_m of this tree should then be as close as possible to the negative gradient $-g_{im}$. Using squared error as a measurement of distance we want to solve the following

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2. \tag{8}$$

Solving (8) does however not create exactly the same regions as when solving (5). Even so, they are similar enough to serve our purpose.

2.6.3 Loss functions

We have not yet discussed the choice of loss function. There is a wide range of choices for this. Different factors, such as the problem one wishes to solve and the nature of the dataset one works with calls for consideration about what loss function one ought to choose. Nonetheless, it is outside the scope of this work to investigate this aspect in detail. Hence, since our task is to classify objects, the most frequently used loss function for this purpose, the deviance, will be used as loss function. The definition goes as follows:

$$L(y, p(x)) = -\sum_{k=1}^{K} y_k \log(p_k(x)), \tag{9}$$

where $p(x) = (p_1(x) \ldots, p_K(x))$. Note that we have a response variable that take values in $G = \{G_1, ..., G_K\}$, meaning that we have K classes to classify. This is usually coded as a binary response vector $y = \{y_1, ..., y_K\}$ with y_k taking the value 1 if it belongs to class G_k and 0 otherwise. Also denote $p_k(x) = P(Y_k = G_k|x)$. Now we want a function $p_k(x)$ that fulfills

$0 \leq p_k(x) \leq 1$ and $\sum_{l=1}^{K} p_l(x) = 1$ so that the class probabilities sum to one. The softmax function, defined as

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{l=1}^{K} \exp(f_l(x))},$$

solves this. With the softmax function we can, using basic laws of logarithms, simplify (9) as

$$L(y, p(x)) = -\sum_{k=1}^{K} y_k f_k(x) + \log\left(\sum_{l=1}^{K} \exp(f_l(x))\right).$$

By differentiating this function with respect to $f_k(x)$, we note that the m:th component of the gradient, with K least square trees, is equal to

$$g_{ikm} = \left[\frac{\partial L(y_i, f_1(x_i), ..., f_K(x_i))}{\partial f_k(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} = y_{ik} - p_k(x_i).$$

The actual algorithm then works as seen below. Also let each class k have its own function f_k.

Algorithm 3 Gradient Tree boosting algorithm for K-class classification

1: Initialize $f_{k0}(x) = 0$ for $k = 1, ..., K$.
2: **for** $m = 1$ to M **do**
3: Set $p_k(x) = \frac{\exp(f_{k,m-1}(x))}{\sum_{l=1}^{K} \exp(f_{l,m-1}(x))}, k = 1, ..., K.$
4: **for** $k = 1$ to K **do**
5: (I) Compute: $g_{ikm} = y_{ik} - p_k(x_i), i = 1, ..., N.$
6: (II) Fit a regression tree to $-g_{ikm}, i = 1, ..., N.$, according to (8)
7: (III) With regions from the trees in step (II) R_{jkm}, j=1,...,J_{km}, compute

$$\gamma_{jkm} = \arg\min_{\gamma} \sum_{x_i \in R_{jkm}} L(y_i, f_{k,m-1}(x_i) + \gamma), j = 1, ..., J_{km}.$$

8: (IV) Update

$$f_{km}(x) = f_{k,m-1}(x) + \lambda \sum_{j=1}^{J_{km}} \gamma_{jkm} I(x \in R_{jkm}).$$

9: **end for**
10: **end for**
11: Output $\hat{f}_k(x) = f_{kM}(x), k = 1, ..., K$

2.6.4 Parameter choices

There are three hyperparameters that play an important role in the performance of algorithm 3.

Firstly we have the tree size J. As we discussed in section 2.1 too large trees are prone to overfitting and additionally have an extra computational cost. In boosting, the idea is instead to grow small trees that improve on the previous errors. There are no exact rules behind the choice of tree size, but typical sizes for the trees are 4-8 terminal nodes.

We also have the choice of the shrinkage parameter λ, defined by scaling. We then scale the contribution of each new tree by $0 \leq \lambda \leq 1$. This corresponds to step (IV) of the Gradient Tree boosting algorithm (algorithm 3). This algorithm looks as follows:

$$f_{km}(x) = f_{k,m-1}(x) + \lambda \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm}).$$

This is called the learning rate and we saw an earlier example of this in algorithm 2. The point of the learning rate is to take small steps in the right direction. If we take too large steps we might overfit. The reason is that the trees are grown in a greedy way to reduce the error for the training data. However, they might do this too well so that the model is overspecified. If we instead let each tree have a smaller contribution to the final fit, a single overfitted tree will not matter as much in the mixture of all trees.

The last parameter in this discussion is M. This is the number of iterations that we allow, or in other words, how many trees we grow. Like before there exists a certain risk of overfitting. Growing too many trees may cause the model to be too specific to the training data.

As discussed we see that all parameters have their own impact on the bias variance tradeoff. For instance, if we choose smaller trees we will have higher bias and lower variance. We can then decrease bias by growing more trees.

In the end cross validation is a good way to determine the best combination of learning rate and tree size. We can then see after how many iterations M we reach our optimal model, and we can also detect when overfitting starts to be a concern. Unfortunately, each iteration has a computation cost.

Further discussion about the final choice of parameters, for our specific dataset, will be provided later.

2.7 XGBoost

This section will be based on Guestrin (2016) [2]. Despite the long discussion about gradient boosting, this method will not be implemented. Instead, eXtreme gradient boosting (XGboost) is the method we will use. Hoewever,

XGBoost is, as one might suspect from the name, a natural continuation of gradient boosting.

Like with gradient boosting we want to minimize a loss function L. However, with XGBoost, this is done differently. The aim is to minimize

$$\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \tag{10}$$

at step m. This is done by approximating L with a second order Taylor expansion. Remember that a linear approximation of $f(x)$ around a is according to Taylor´s theorem given by

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2. \tag{11}$$

The key here is then to differentiate each term $L(\cdot, \cdot)$ in (10) with respect to its second argument and view $f(x)$ as the loss function for $f_{m-1}x_i + T(x_i, \Theta_M)$, $f(a)$ as the loss function for $f_{m-1}x_i$, and $(x-a)$ as the tree $T(x_i; \Theta_m)$ that we want to add in (10). If we let the second and first derivatives from (11) be denoted as follows

$$g_m(x_i) = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} \text{ and } h_m(x_i) = \frac{\partial^2 L(y_i, f_{m-1}(x_i))}{\partial^2 f_{m-1}(x_i)}$$

we can approximate (10) trough

$$\sum_{i=1}^{N} \Big[L(y_i, f_{m-1}(x_i)) + g_m(x_i) T(x_i; \Theta_m) + \frac{h_m(x_i) T(x_i; \Theta_m)^2}{2} \Big]. \tag{12}$$

Let T be a tree (2.7) and define $\Omega(T) = \Gamma J + \frac{1}{2}\eta \sum_{j=1}^{J} \gamma_j^2$. In particular, if $T = T(\cdot; \Theta_m)$ we can write this term as $\Omega(\Theta_m)$. If we add this term to (12) we get the following expression we want to minimize in the m:th step of the XGBoost algorithm.

$$\sum_{i=1}^{N} \Big[g_m(x_i) T(x_i; \Theta_m) + \frac{h_m(x_i) T(x_i; \Theta_m)^2}{2} \Big] + \Omega(\Theta_m). \tag{13}$$

Note that $L(y_i, f_{m-1}(x_i))$ is a constant and therefore it is not relevant. Now you should wonder why the expression $\Omega(T)$ has appeared. The reason is that it is a regularization term to prevent overfitting. Note, just as before J is the number of terminal nodes in a tree and γ_j are the coefficients for the regions. Both Γ and η are hyperparameters that are tuned to regulate the bias variance tradeoff, and we will return to them later in the parameter discussion. Now recall from () that we write a tree with J regions as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j).$$

Also remember that all regions are disjoint and that $I(x_i \in R_j)$ is an indicator function so that the terms with $x_i \notin R_j$ are all zero. Using this and the definition of $\Omega(T)$ we write (13) as

$$\sum_{j=1}^{J_m} \Big[\big(\sum_{x_i \in R_{jm}} g_m(x_i) \big) \gamma_{jm} + \frac{1}{2} \big(\sum_{x_i \in R_{jm}} h_m(x_i) \big) \gamma_{jm}^2 \Big] + \Gamma J_m + \frac{1}{2} \eta \sum_{j=1}^{J_m} \gamma_{jm}^2.$$

Furthermore, if we write

$$G_{jm} = \sum_{x_i \in R_{jm}} g_m(x_i) \text{ and } H_{jm} = \sum_{x_i \in R_{jm}} h_m(x_i)$$

and combine all terms containing γ_{jm} we get the following expression to minimize at each step m:

$$\sum_{j=1}^{J_m} \Big[G_{jm} \gamma_{jm} + \frac{1}{2} \Big(H_{jm} + \eta \Big) \gamma_{jm}^2 \Big] + \Gamma J_m. \tag{14}$$

Now say we have a predefined tree structure, we then want optimal values for γ_{jm}. These are easily found by differentiating (14) with respect to γ_{jm} and setting the corresponding derivative equal to zero. Thus we get

$$\tilde{\gamma}_{jm} = -\frac{G_{jm}}{H_{jm} + \eta}.$$

Plugging in this expression into (14) yields

$$-\frac{1}{2} \sum_{j=1}^{J_m} \frac{G_{jm}^2}{H_{jm} + \eta} + \Gamma J_m. \tag{15}$$

Expression (15) can then be used to measure how good predictions a given tree produces, referred to as the score function.

The score function plays a vital role in constructing the trees in XGBoost. When adding new regions we evaluate their contribution based on what score they correspond to. Like in section 2.1, the number of splits to consider based on this score function criterion is too large. Hence we use the earlier described top-down greedy approach.

Let $\frac{G_L^2}{H_L + \eta}$ be the score from a potential left leaf, let $\frac{G_R^2}{H_R + \eta}$ be the same for a right leaf. Furthermore, let $\frac{G^2}{H + \eta} = \frac{(G_L + G_R)^2}{H_L + H_R + \eta}$ be the score representing no split at all.

We can then get an expression to minimize when doing splits in order to improve the model. We define

$$L_{split} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \eta} + \frac{G_R^2}{H_R + \eta} - \frac{G^2}{H + \eta} \right] - \Gamma. \tag{16}$$

Create a new split when $L_{split} \geq 0$. However, sometimes no additional split is created at all, and this happens if $L_{split} < 0$.

Algorithm 4 XGboosting algorithm for K-class classification

Initialize $f_{k0}(x) = 0$ for $k = 1, ..., K$.
for $m = 1$ to M **do**
    for $k = 1$ to K **do**
        (I) Compute: $g_{km}(x_i) = \frac{\partial L(y_i, f_{k,m-1}(x_i))}{\partial f_{k,m-1}(x_i)}, i = 1, ..., N$
        (II) Compute: $h_{km}(x_i) = \frac{\partial^2 L(y_i, f_{k,m-1}(x_i))}{\partial^2 f_{k,m-1}(x_i)}, i = 1, ..., N$
        (III) Fit a tree $T(x_i, \Theta_{jkm})$ for $i = 1, ..., N$ that for each split maximizes

$$L_{split} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \eta} + \frac{G_R^2}{H_R + \eta} - \frac{G^2}{H + \eta}\right] - \Gamma.$$

        (IV) With splits from III compute:

$$\tilde{\gamma}_{jkm} = -\frac{G_{jkm}}{H_{jkm} + \eta}.$$

        (V) Update

$$f_{km}(x) = f_{k,m-1}(x) + \lambda \sum_{j=1}^{J_{km}} \tilde{\gamma}_{jkm} I(x \in R_{jkm}).$$

    end for
end for
Output $\hat{f}_k(x) = f_{kM}(x), k = 1, ..., K$

2.7.1 Parameter choice

As in Gradient boosting we again have the parameters tree size J, learning rate λ, and the number of iterations m. For a discussion about them and their impact on the model see section 2.5.3.

Two new parameters for XGBoost are η and Γ. Both of them work as regularization parameters. Increasing them will make the model less prone to growing large trees. The reason is simply that η and Γ reduce L_{split} as seen in (16) and having $L_{split} < 0$ means that no further splits are done.

There are more parameters of XGBoost (Guestrin, 2016)[2]. However, the ones mentioned above are the ones I will use in this work. A longer discussion on their impact follows later.

2.8 Variable importance

In both random forest and XGBoost accuracy is increased from a single decision tree by using an ensemble of trees. However, a price is paid in form of interpretability. And in the end, it is desirable to get some insight into why the model works as it does.

Some understanding can however be achieved by using the variable importance metric. What this does is to give each predictor a value. The value is based on how much this predictor contributes to the accuracy of the model.

For a single tree this contribution was proposed by Breiman et al. (1984) [1] to be measured as

$$I_\ell^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbf{I}(v(t) = \ell)$$

for each predictor X_ℓ, where $v(t) \in \{1, \ldots, p\}$ tells which predictor variable that is involved in split t of the tree.

The $\hat{i}_t^2$ is the improvement in squared error risk that comes from the best possible way to define split t. By summing the improvement over all internal nodes of T for which X_l was chosen in the corresponding splits, we obtain its squared relative importance.

This can then be generalized to additive tree expansions by averaging over the trees

$$I_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} I_\ell^2(T_m). \tag{17}$$

For K-class classification, K different models

$$f_k(x) = \sum_{m=1}^{M} T_{mk}(x)$$

are brought about. Note that we sum over the M trees. Thus (17) generalizes to

$$I_{\ell k}^2 = \frac{1}{M} \sum_{m=1}^{M} I_\ell^2(T_{km}).$$

By averaging over the classes we then get the overall relevance of X_ℓ

$$I_\ell^2 = \frac{1}{K} \sum_{k=1}^{K} I_{\ell k}^2. \tag{18}$$

3 Data

For this thesis news article data from BBC will be used. The goal will be to predict what category a certain article belongs to given which words are used, and how often these words occur. Furthermore, the data is divided into two sets. One set is used to train the model. Another smaller set is used to evaluate the model.

3.1 Datasets

3.1.1 Cleaned dataset

As the data from [2] first consisted of 2225 .txt files there was a need of cleaning the data. To a high degree the code for doing this was borrowed from [5]. After the data cleaning process all stopwords were removed. The remaining words were then transformed to lower case and all .txt files were merged to one dataset. Thus we had a dataset with 2225 observations (one for each article). The response variable was then the category of the article. Within the response variable there are 5 categories, namely business (510 observation), sport (511 observations), tech (401 observation), politics (417 observations) and entertainment (386 observations). Bellow in table 1 there is a description of the different variables. Furthermore, each word occurring in the text column became its own variable. This variable then took values corresponding to the frequency of the word in each individual article. The idea was then to identify what category the article belonged to based on both which words occurred, and their frequency. An example of how this works is provided in table 1.

Table 1: Example of observations from the new format.

poll	housewives	grand	slam	economic	Category
1	0	0	0	1	business
0	0	1	1	0	sport
0	1	0	0	0	entertainment

This format of the dataset does however introduce one problem, namely the number of predictors. There are as many as 34099 different words in the training set. Thus the new dataset has 34099 new columns, even after the removal of the stopwords. Dealing with so many predictors is computationally infeasible, and in addition, most columns are probably worthless for prediction. For instance, the word "abdomen" occurs only once in all articles. But with XGBoost and random forest, columns that contain no valuable information are still considered in the algorithm, hence they bring additional computation cost without any benefit.

3.1.2 Different datasets

On account of this, we created datasets with different subsets of the 34099 columns. A total of 18 datasets were created. Of these sets, 12 were chosen to be the most commonly occurring words from all articles. The smallest set consisted of the 10 most frequent words, the largest of the 5000 most frequent words.

The remaining 6 sets were instead created based on frequency from each category separately. The smallest incorporated the 25 most common words of each category and the largest the 350 most occurring words per category. Note however that this approach leaves us with duplicates since many of the most common words for one category also appear in other categories. For example the set with the 350 most common words for each category in the end only had 1068 predictors and not 1750. And the more words that were selected from each category, the more they resembled the firstly mentioned method of selecting words, and for this reason no larger sets were created with this second approach.

Using dataset with different number of predictors introduced not only the opportunity of comparing XGBoost with random forest but it also made it possible to see how many words are needed to be analyzed by the models to produce satisfactory accuracy. Furthermore, we will also compare computation time and which parameters are suitable for the different datasets.

The datasets are then named based on how many predictors they include and whether these predictors are selected as the most frequent words per category or for all sets. For more details see table 4.

Table 2: Description and size of the 18 datasets created from the original set of 2225 articles

Name	Description	Number of predictors
C10	Subset 10 words	10
C25	Subset 25 words	25
C50	Subset 50 words	50
C75	Subset 75 words	75
C100	Subset 100 words	100
C150	Subset 150 words	150
C200	Subset 200 words	200
C250	Subset 250 words	250
C500	Subset 500 words	500
C1000	Subset 1000 words	1000
C2500	Subset 2500 words	2500
C5000	Subset 5000 words	5000
PC33	Subset most 10 common words per category	33
PC87	Subset most 25 common words per category	87
PC174	Subset most 50 common words per category	174
PC338	Subset most 100 common words per category	338
PC510	Subset most 155 common words per category	510
PC1068	Subset most 350 common words per category	1068

3.1.3 Data split

Before training the model on the data we partitioned the dataset with 2225 observations. Thus one training set consisting of 80% of all articles, whereas 20% of the articles were then saved for validation of our data. After the split, the training and validation data consisted of 1780 and 445 observations respectively.

The idea behind this procedure is to be sure that the original models generalize well. Despite using cross validation, there is a small risk of over-training, hence we try the fitted models on data they have not seen before.

4 Modeling

As mentioned earlier in the theory section, finding appropriate parameters is of utmost importance for a well-performing model. In this thesis, we will be using grid search. This means that we create a grid of different parameter values to obtain the best-performing model. The grid of parameter values

will be based on creating optimal accuracy without creating too computationally expensive models. Exactly which parameters we will try will be specified and motivated in the sections below.

4.1 Parameter tuning for random forest

Here we will briefly revisit the parameter discussion from section 2.4.

When choosing parameters for the random forest model the key is to look at the expression

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

from section 2.4 regarding the variance of the sample mean of B terms.

The second term goes to zero as we increase B (the number of trees grown). However, after a while, we are still stuck with the first term. Hence growing enough trees is only one part of reducing the prediction error. Furthermore, convergence is guaranteed by including enough trees. For this reason, we will not fine-tune this parameter, instead, we will use the default setting of random forest, which is $B = 250$, and in a plot assure ourselves of convergence.

What we mainly will focus on is the correlation coefficient ρ which is regularized by m where m is the number of variables considered in each split. Having a too large m will increase ρ and thus increase the variance. For classification, the typical value is $m = \sqrt{p}$ where p is the number of predictors used. However, often the optimal value is quite a bit lower or higher. We will hence try a grid of values for m. Moreover, we will try different parameters for different datasets.

For the smallest set (10 predictors) we try all values ranging from 1 to 10. For the sets with 25 - 250 predictors we test

$$m \in \{1, 5, 10, 15, 20, 25\}.$$

The remaining sets are tested with

$$m \in \{5, 10, 20, 40, 80, 120\}.$$

Note that there exists yet another parameter to tune, namely tree depth. However, since deep trees have low bias, and variance reduction is achieved through tuning B and m, we will not regularize the size of the trees.

4.2 Parameter tuning for XGBoost

This part is based on sections 2.6.3 and 2.7.1, to which we refer for more mathematical details.

Earlier we mentioned that numerous iterations M come with the benefit of incrementally increasing the performance on the training data. The

downside was the computation cost and the risk of overfitting. However, by using cross validation, we should be able to see when overtraining is becoming an issue. Furthermore, there exists a parameter "early stopping rounds" that we set equal to 10. This means that the model stops if no improvement is reached during ten consecutive iterations.

We also have the learning rate λ. Higher values of λ means faster convergence. This does, however, come with a risk of not finding the values of λ that reduce error the best, i.e not finding the global minimum of the loss function. The standard for XGBoost is 0.3 but we will use 0.1 and thus trade some extra computation time for extra error reduction.

Then there is the max tree depth. Larger trees as we discussed earlier may lead to overfitting. In addition to that, there is also the computational aspect. The default for XGBoost is a max tree depth equal to 6, however, we will also try 2 and 10.

Furthermore, we have the parameter Γ from algorithm 4. Larger values of Γ lead to a more conservative model less prone to overfitting. For exact details of what this parameter does we refer to section 2.7. We are going to try $\Gamma \in \{0.5, 1, 1.5\}$.

There exists many more parameters to regulate for XGBoost but we will stick to using default values for other parameters than the ones mentioned above. Indeed since we already have many datasets (with some of them being quite large) we must consider computational aspects and stick to a few chosen parameters of the model to tune.

The interested reader can go to XGBoost documentation [5] for more information.

5 Results

Here we will take a look at the performance of our different models. A discussion about why we have these results and a comparison between them will follow in the next section.

Furthermore, for a description of the different datasets see table 2 of section 3.1.2.

5.1 Results for Random Forest

We can see in table 3 that the highest accuracy obtained through 10-fold cross validation was with dataset PC1055 and it was equal to 0.964. The highest validation set accuracy was 0.991 and was again obtained with the dataset PC1055.

Table 3: Highest achieved CV and validation set accuracy of random forest for each dataset. Shown is also the final value of m, the number of variables considered in each split, and computation time.

Dataset	CV accuracy	VS accuracy	m	Computation time
C10	0.527	0.530	2	1.8 min
C25	0.679	0.685	5	1.9 min
C50	0.814	0.811	5	3.5 min
C75	0.872	0.879	5	5.2 min
C100	0.897	0.899	5	6.9 min
C150	0.920	0.921	10	10.3 min
C200	0.924	0.933	5	13.7 min
C250	0.939	0.960	5	16.5 min
C500	0.953	0.969	5	27.5 min
C1000	0.959	0.978	10	51.2 min
C2500	0.959	0.978	20	120.4 min
C5000	0.961	0.978	10	220.3 min
PC32	0.851	0.964	5	2.3 min
PC91	0.926	0.982	5	5.8 min
PC177	0.948	0.984	5	11.0 min
PC336	0.956	0.984	10	19.2 min
PC514	0.962	0.987	5	27.6 min
PC1055	0.963	0.991	5	53.2 min

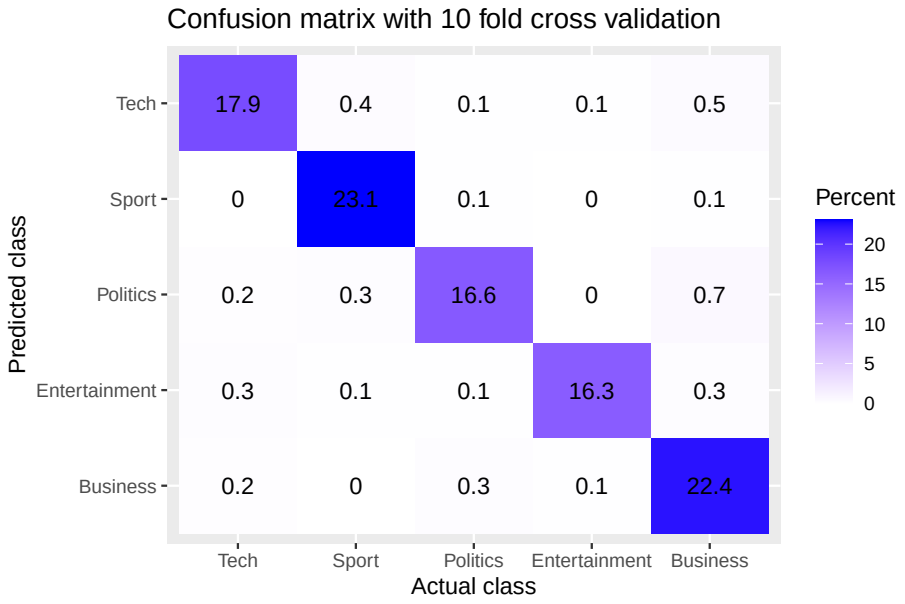

Figure 3: Cross validation confusion matrix using random forest. Entries are percentual average cell counts across resamples.

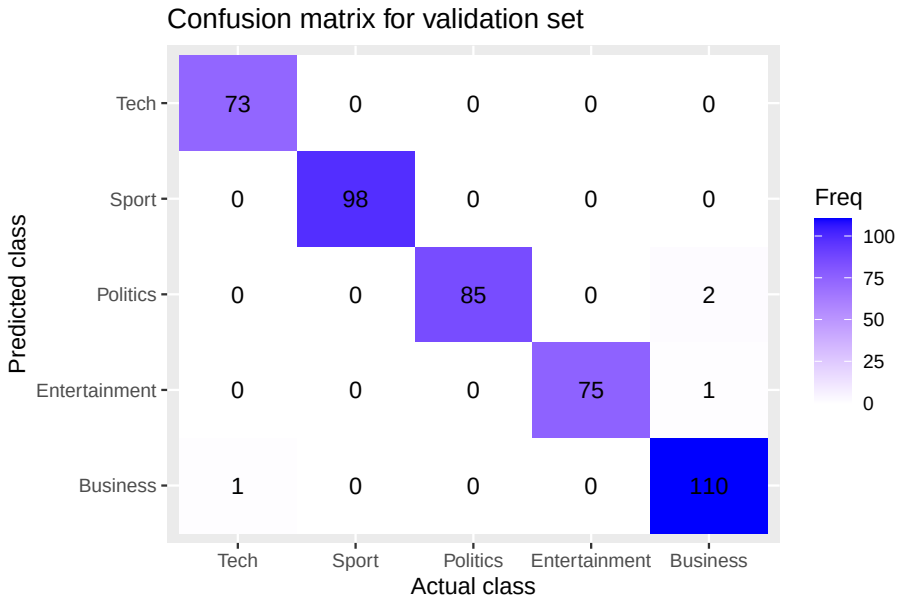

Figure 4: Confusion matrix using random forest. Predictions are made on the validation set on data that the models were not trained on. Entries are actual cell counts.

5.2 Results for XGBoost

In table 4 we see that the best accuracy for 10-fold cross validation is equal to 0.959, which is achieved with the PC510 dataset. As for the validation dataset, again PC510 gave the highest accuracy 0.989.

Table 4: Highest achieved CV and validation set accuracy of XGBoost for each dataset. Displayed for each dataset is also the optimal tree depth and the parameter Γ

.

Dataset	CV accuracy	VS accuracy	Depth	Γ	Computation time
C10	0.517	0.503	10	1	2.2 min
C25	0.683	0.647	6	0.5	3.9 min
C50	0.808	0.827	6	0.5	6.9 min
C75	0.867	0.872	6	0.5	9.7 min
C100	0.881	0.899	2	0.5	12.5 min
C150	0.911	0.912	2	0.5	18.3 min
C200	0.916	0.921	10	0.5	24.3 min
C250	0.927	0.944	2	1	29.6 min
C500	0.953	0.957	2	0.5	64.1 min
C1000	0.956	0.964	2	0.5	130.4 min
C2500	0.954	0.973	2	0.5	324.5 min
C5000	0.954	0.964	2	1	639.4 min
PC33	0.842	0.928	6	0.5	4.6 min
PC87	0.925	0.948	6	0.5	11.1 min
PC174	0.947	0.980	2	0.5	20.6 min
PC338	0.952	0.978	2	1	38.8 min
PC510	0.959	0.989	2	0.5	64.3 min
PC1068	0.958	0.987	2	1	137.6 min

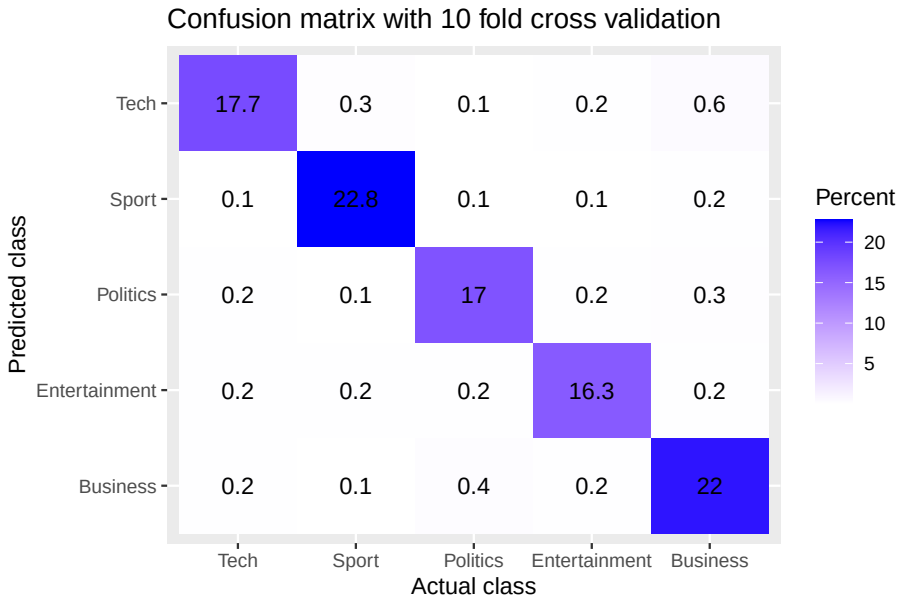

Figure 5: Cross validation confusion matrix using XGBoost. Entries are percentual average cell counts across resamples.

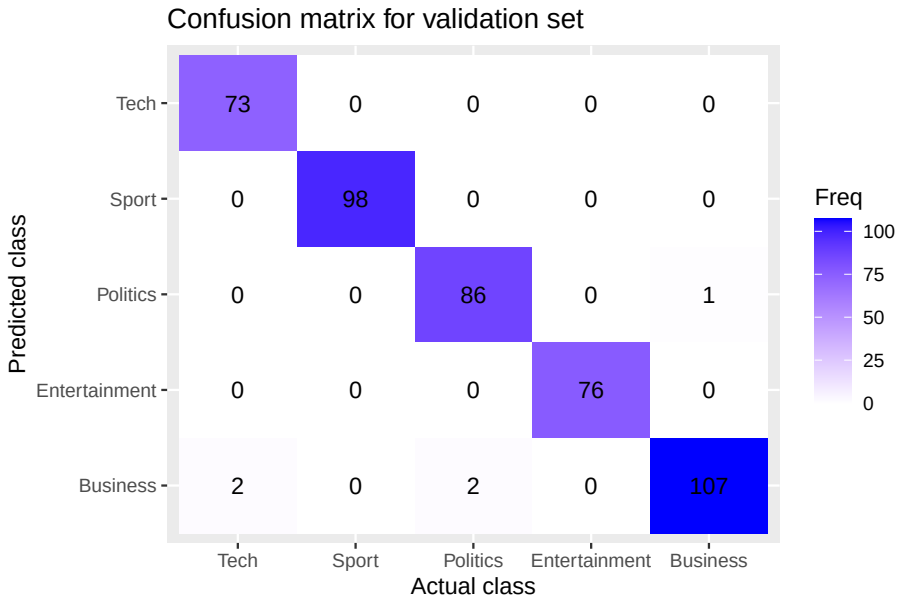

Figure 6: Confusion matrix using XGBoost. Predictions are made on the validation set on data that the models were not trained on. Entries are actual cell counts.

6 Discussion

6.1 Comparison between the datasets

As we can see in figure 7 as few as 10 predictors can give us a prediction accuracy of 0.53 for random forest. After that, the results converge to a final result sligthly above 0.95. This occurs when about 500 predictors are used.

Another interesting result from figure 7 is that the blue dots, representing the datasets where we choose words from all categories separately, perform somewhat better than the red dots. Remember that the red dots mean that the predictors were chosen to be the most commonly occurring words from all classes. When trying the model on the validation set as seen in figure 8, this trend becomes even clearer. For the validation set, even the blue dots with a low number of predictors outperforms the red dots with much higher number of predictors.

Another noteworthy trend from figures 7-10 is that when there are fewer than 500 predictors, the difference between the red and blue dots are clearer. This can possibly be explained by the fact that the per-category subsetting technique guarantees important words to be included in all news categories. These important words may be missed when subsetting from the combined set. However, as discussed earlier, when including more and more predictors, the different subset techniques of creating datasets start to resemble each other more and more. This may explain why we see that the accuracy from the red and blue dots get more similar when the number of predictors grows.

These figures indicate that the subsetting per category method is preferable, especially when building less complex models.

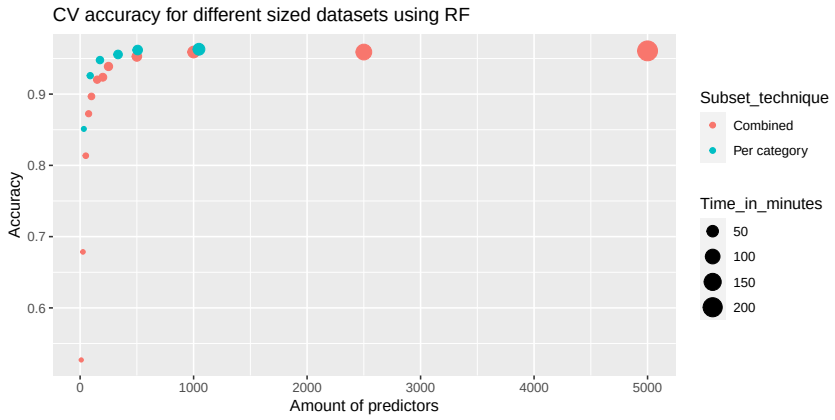

Figure 7: Cross validation accuracy, for random forest, as a function of the number of predictors with dot size representing computation time.

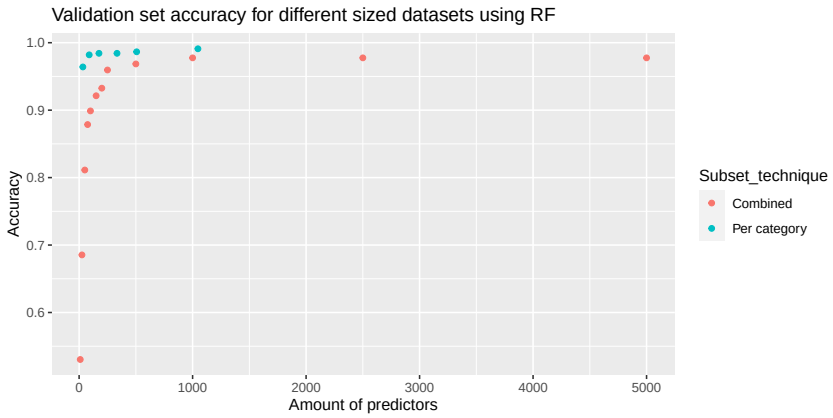

Figure 8: Validation set accuracy, for random forest, as a function of the number of predictors.

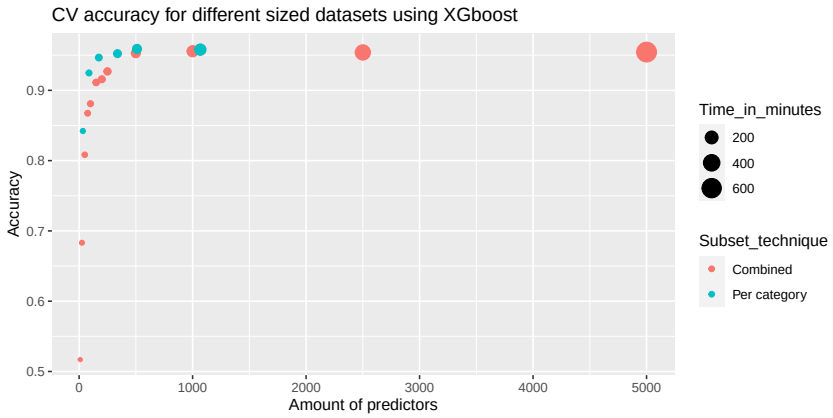

Figure 9: Cross validation accuracy, for XGBoost, as a function of the number of predictors with dot size representing computation time.

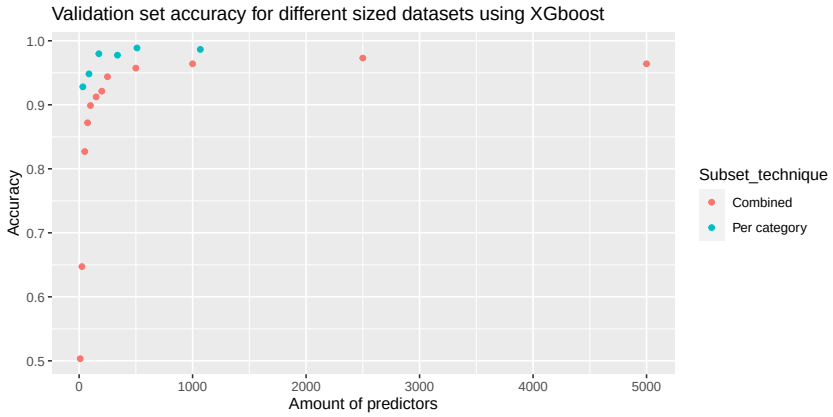

Figure 10: Validation set accuracy, fir XGBoost, as a function of the number of predictors.

6.2 Comparison between Random Forest and XGboost

Suprisingly, considering the reputation of XGBoost, is that random forest surpasses XGBoost (by a small margin) in terms of accuracy. This holds true regarding both cross validation and validation set accuracy. And as seen in figures 11 and 12, this phenomenon is not restricted to any particular dataset, it is true for all of them. This observation might have to do with our dataset being more suited for the random forest model. But on the other hand, there are simply too many parameters to consider to state with certainty the superiority of one method over the other. And as discussed in sections 4.1 and 4.2, there are many more possible tuning parameters that I have not explored.

In addition to comparing accuracy, we can also contrast the computation time of both models. As seen in figure 11 and figure 13, the computation time of XGBoost blows up in a way the computation time of random forests does not. This is however somewhat expected since random forest is robust to datasets with many predictors, as discussed in section 2.4.

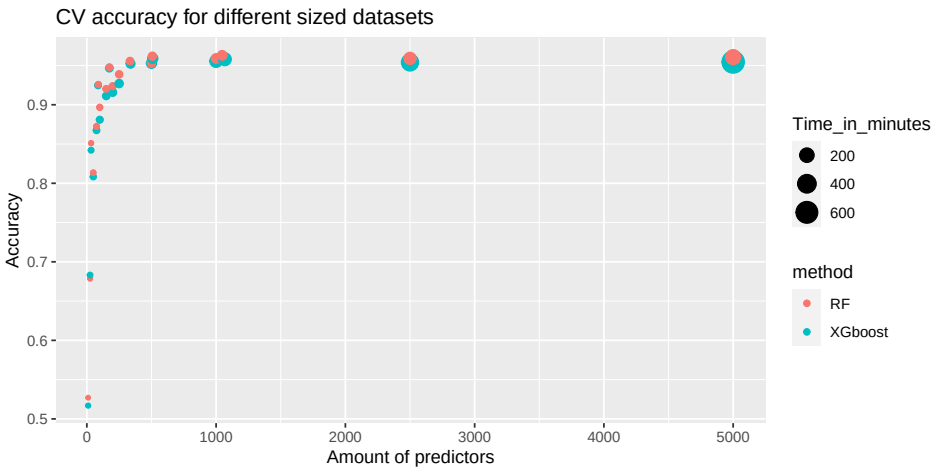

Figure 11: Cross validation accuracy as a function of the number of predictors for random forest and XGBoost with dot size representing computation time.

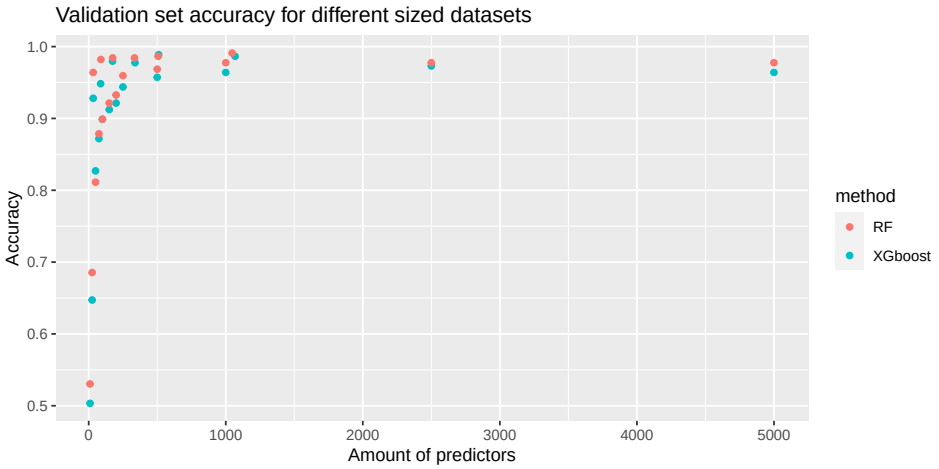

Figure 12: Validation set accuracy as a function of the number of predictors for random forest and XGBoost.

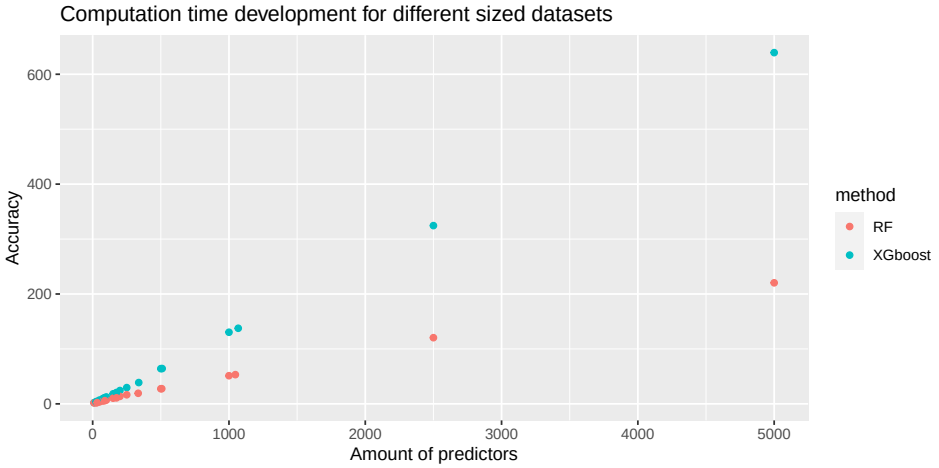

Figure 13: Computation time as a function of the number of predictors for random forest and XGBoost.

6.3 Variable importance

Our best-performing models all had an accuracy exceeding 96%. In this section, we will discuss which variables made this classification rate possible. Also, note that the plots of figure 14 and 15 are based on results from the models trained on dataset PC1068 for random forest and C510 for XGBoost (for dataset descriptions see table 2).

6.3.1 Variable importance for Random Forest

In random forest, in contrast to XGBoost, there is a more broad range of variables that are used to create splits in the trees. This follows from the

fact that a random subsample of all possible predictors is used in each split. Furthermore, as discussed in section 2.8, feature importance for random forest is based on how much the prediction accuracy is influenced by choosing a given feature as a predictor.

From figure 14 we can observe which variables were the most important for each and every category.

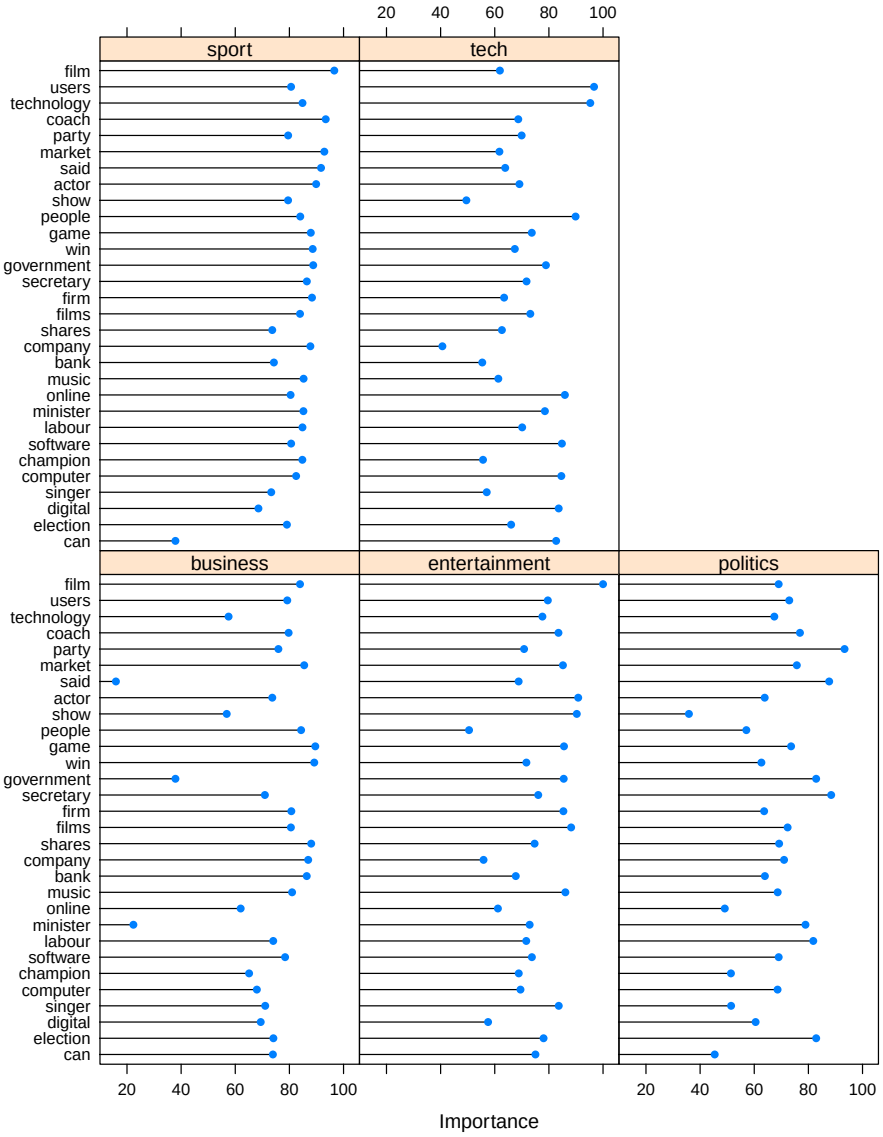

Figure 14: Variable importance (measured by (18)) for the different categories using RF.

Many of the results in figure 14 are quite intuitive. For instance, the frequencies of the words "party", "election", and "secretary" play an important role in deciding if the article belongs to the politics category. Likewise "film", "show", and "actor" are key predictors for the entertainment category.

6.3.2 Variable importance for XGBoost

Before interpreting the result it is worth noting that variable importance for XGBoost is a bit different from that of random forest. Firstly there are three different metrics. We will however only use the one named Gain, this represents how well the chosen predictor reduces the loss function. Secondly, the scale is different from that in figure 14 so direct comparisons can not be made. The interested reader can learn more about the different variable importance choices in XGBboost documentation [5].

As with random forest, the results are quite intuitive. For instance, shares is a word that provides useful insight into whether or not an article belongs to the business category.

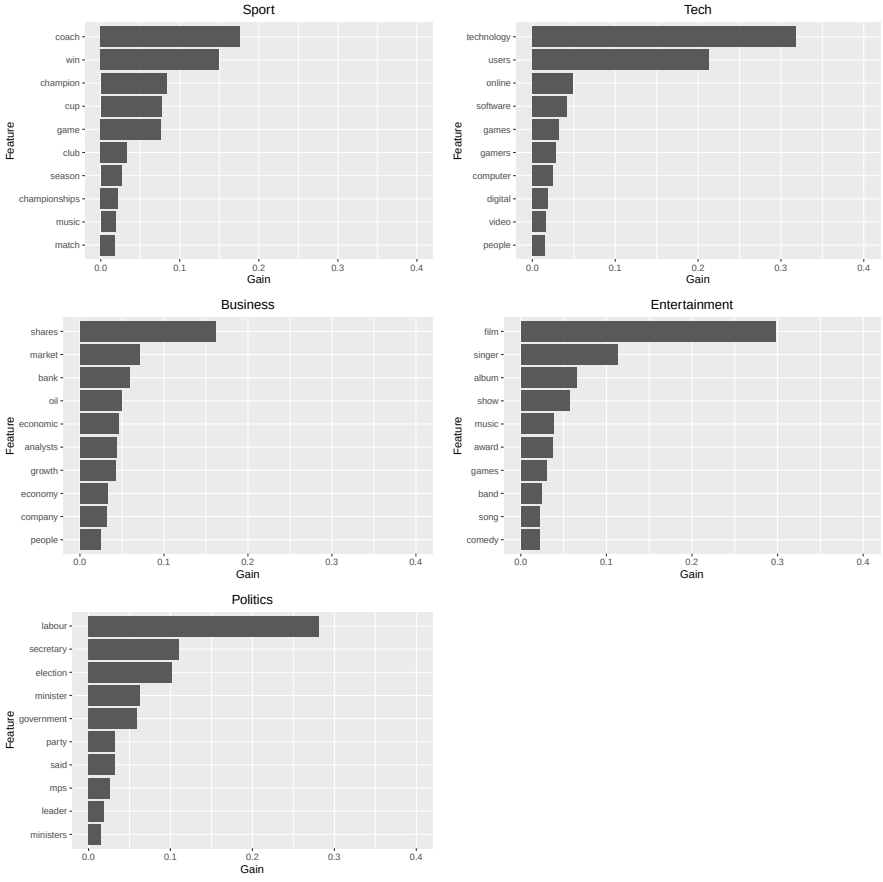

Figure 15: Variable importance (measured by (18)) for the different categories using XGBoost.

6.4 Performance discussion

The highest performing models reached an accuracy above 96% for both cross validation and the validation set. This is quite a high success rate, especially considering that the distinction between articles is not always that clear. Sometimes in real life an article can belong both to the business and tech categories. Also worth noting is the fact that the validation dataset got accuracy rates higher than with 10 fold cross validation. This might suggest that our final model is close to perfection in evaluating new data. However, it might also mean that our models are overtrained for the validation set. The best way to control this would be to have yet another set of observations (testing set) to try the model on, but that is outside the scope of this thesis.

Furthermore, the final models are interpretable in terms of variable importance and have been built by finding meaningful structures in the dataset.

6.5 Potential improvements

In the end, as mentioned, the finally chosen random forest and XGBoost models reach an impressive prediction accuracy. An obvious way to increase the accuracy even more would be to restrict the usage of the models for certain categories, categories that are more clearly separable, i.e. excluding the business class.

Another obvious way, as often is the case with machine learning, would be to collect more data. The models perform well by picking up patterns, so one can easily imagine that these patterns become even clearer with additional observations. This would also allow us to save a test set to rule out overtraining to the validation set as mentioned in section 6.4.

Then there is also the case of parameter choice, as discussed earlier. Firstly there exists a larger grid of values to test, especially for XGBoost, and there is a chance that some of them would increase accuracy somewhat more. Secondly, the method used to obtain our final parameter values has been grid search. One could also try random search.

Yet another possible improvement would be to modify the method for creating the different datasets in section 3.1.2. One could for example allow for a certain number of unique words from each category, or one could filter for words that exist in a certain percentage of all articles from the different categories.

7 Conclusion

As stated in the introduction the main aim of this thesis was to create a model that could classify news with high accuracy. This was achieved and many of the models reached a prediction accuracy as high as $95 - 96\%$. Furthermore, we saw that despite XGBoost's reputation, random

forest actually performed better both in terms of computation time and accuracy. We also learned that of all possible words existing in the articles, only a few are needed as predictors in order to attain high accuracy.

References

[1] Breiman, L., Friedman, J., Olshen, R and Stone, C. (1984). Classification and regression trees, Wadsworth, New York.

[2] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. arxiv.org/abs/1603.02754. Available at :

https://arxiv.org/abs/1603.02754

[3] D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006. Available at :

http://mlg.ucd.ie/datasets/bbc.html

[4] Hastie, T and , Tibshirani R. and Friedman, J. (2017). The Elements of Statistical Learning. Second edition, Springer Series in Statistics, Springer New York Inc., New York.

[5] Miriam Chou (2018). BBC-News-Classifier. Available at :

https://github.com/mijchou/NLP–BBCNewsClassifier

[6] Muren, J (2019). Classification of Music Genres with eXtreme Gradient Boosting. Department of Mathematics, Stockholm University. Available at:

https://www.math.su.se/publikationer/uppsatsarkiv/tidigareexamensarbeten-i-matematisk-statistik/kandidat/kandidatarbeten-imatematisk-statistik-2019-1.422151.

[7] XGboost documentation (2016). Available at:

https://xgboost.readthedocs.io/en/latest/.