FYSSTK4155

Project 3 - Yelp Star Rating

Erlend Lima & Per-Dimitri B. Sønderland & Vala M. Valsdottir

Classification is performed on Yelp reviews in an attempt to predict a rating based on previous business and user data. The methods employed are logistic regression, random forest and AdaBoost. All methods gave similar performance with logistic regression consistently performing slightly better. When ranking features by importance it is discovered that the average business rating and average user rating are by far the most important features, entailing that the models are of minimal assistance if one wishes to predict ratings for the purpose of helping businesses bettering their ratings.

CONTENTS

I. INTRODUCTION

Yelp is an internet company offering people to review businesses by assigning them a star rating 1 through 5. It was founded by Jeremy Stoppelman and Russel Simmons in 2004 and has grown to one of the biggest rating website and app in the world [1]. There are 5 million reviews of restaurants in their database and it includes the US, Europe and Asia. Due to Yelp's potential to make a business rise or fall, many are interested in maximizing their own rating. Fortunately Yelp provides big data sets containing user and business information alongside reviews. Utilizing this data it might be possible to supply some factors to which business owners should pay attention to in order to attain a good rating.

For this project we picked out 500 000 of these reviews and used classification machine learning algorithms to see if they can predict which rating the customers are most likely to give. Our hypothesis is to see if a classification analysis on meta data from reviews extracted from Yelp can be used to predict reviews for a business in question with machine learning algorithms. This project has the aim to see if machine learning can be used in a practical way and hopefully then offer a good statistical analysis for business owners to use in the future.

The GitHub repository is available here: https://github.com/Caronthir/FYSSTK4155/tree/master/projects/project3.

II. THEORY

A. Logistic Regression

The main point of Logistic Regression is that it takes in a classification problem as input and gives probabilities as output. In the case of a yes or no answer one can say that the output is $Y = 1$ or $Y = 0$. Then one can use the sigmoid-function

$$P = \frac{1}{1 + \exp\{-y\}}, \tag{II.1}$$

which gives values between 1 and 0, this can be used to classify if the answer is either yes or no. y is the equation of a line

$$\hat{y} = \hat{X}^T \hat{w} + \hat{\epsilon}, \tag{II.2}$$

and if its value is tending towards negative infinity the sigmoid-function will be equal to zero and if it is tending to positive infinity the sigmoid-function will be equal to one. As the first rule of thumb one can classify the output of the the sigmoid as: under 0.5 will give a no-answer and above 0.5 will give a yes-answer. For training our model we use Gradient Descent that converges to the best solution. For this we need a cost function which we get by the likelihood of our output being $Y = 1$ or $Y = 0$

$$\begin{aligned} p\left(y_i = 1|x_i, \hat{w}\right) &= \frac{\exp(x_i w_i)}{1+\exp(w_i x_i)} \\ p\left(y_i = 0|x_i, \hat{w}\right) &= 1 - p\left(y_i = 1|x_i, \hat{w}\right) \end{aligned}. \tag{II.3}$$

To obtain the total likelihood for all the possible outcomes of our dataset we use the Maximum Likelihood Estimation principle (MLE) [2]

$$P(\mathcal{D}|\hat{w}) = \prod_{i=1}^{n} \left[p\left(y_i = 1|x_i, \hat{w}\right)\right]^{y_i} \left[1 - p\left(y_i = 1|x_i, \hat{w}\right)\right]^{1-y_i}. \tag{II.4}$$

To obtain our cost/loss function or log-likelihood function is simply taking the log of the function above

$$\mathcal{C}(\hat{w}) = \sum_{i=1}^{n} \left(y_i \left(w_i x_i\right) - \log\left(1 + \exp\left(w_i x_i\right)\right)\right). \tag{II.5}$$

This function is commonly called *The Cross Entropy* error function. To minimize the cost-function, which in turn optimizes our weights. We thus need to find the derivative in terms of the weights, presented here in matrix form

$$\frac{\partial \mathcal{C}(\hat{w})}{\partial \hat{w}} = -\hat{X}^T(\hat{y} - \hat{p}). \tag{II.6}$$

A gradient descent method either the regular Gradient Descent (GD), Stochastic Gradient Descent (SGD) or the Stochastic Gradient Descent with mini batches (MB) can minimize this function. If more information is need on the Logistic Regression and Gradient Descent methods one can find it in Project 2 [3].

B. Decision Trees

In what follows is the theory of *Decision Trees* for *classification*. A decision tree is a predictive model that partitions a feature space of data into ordered levels of nodes usually, but not necessarily, connected in a binary manner.

Figure II.1 shows an example of such a decision tree where the final outcome is divided into two classes of high rating and low rating. The top red node of the tree is called the *root node* while the following blue nodes are called *internal nodes*. The green end nodes, with arrows only pointing to them, are called *leaf nodes*.

To be able to decide how to split and order a decision tree, a measure of *impurity* is needed. If a certain

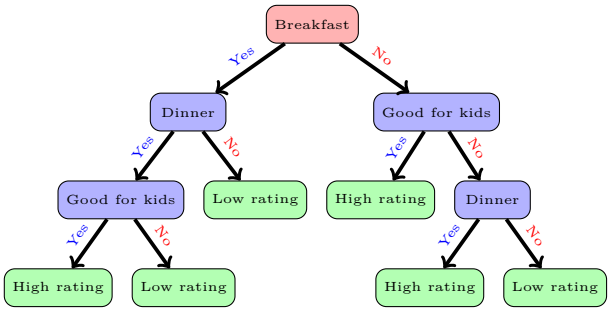


Figure II.1. Simple example of a *decision tree*, with binary features and two classes; *Class = (Low rating, High rating)*. The later nodes in this figure aren't necessarily connected to the values of figure fig. II.2

question, say the node "Breakfast" in the example fig. II.1, completely separates all the ratings in the data into either low rating or high rating it would have zero impurity. Meaning that the truth value of this question alone would be a perfect predictor for the rating. This is usually not the case and assumed to not be the case for any of the questions in the example, thus they all have a degree of impurity. By measuring this impurity with a common metric we can compare the nodes and decide how to order them in an optimal manner. There are several ways of doing this, but the method used in this article is the *Gini index* [4, p. 309].

$$I_G = \sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}\left(1 - \hat{p}_{mk}\right) \tag{II.7}$$

and using that $\sum_{k=1}^{K} \hat{p}_{mk} = 1$, Equation (II.7) can be written

$$I_G = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^2 \tag{II.8}$$

where $\hat{p}_{mk}$ is the proportion between given class k observations to the total observations N_m in that given node m, or put in a different way; $\hat{p}_{mk}$ is the probability of correct categorization of an outcome k given the truth value of a feature corresponding to the node m. $\hat{p}_{mk'}$ represent the probability of a mistaken categorization. The total observations of a given feature does not necessarily have to be equal for different features. Invoking aid by example, question "Dinner" in fig. II.1 has not been answered by the same number of raters as the question "Good for kids". Assuming that we have the two truth values, the total Gini impurity for a given feature is then the weighted average of the Gini indicies of the two possible truth values. For our fictional rating example we show in fig. II.2 the Gini impurities that determines which feature becomes the root node. These impurities were calculated by using Equation (II.8) in the following manner

$$\begin{aligned} I_G(yes) &= 1 - \hat{p}^2_{(yes,high)} - \hat{p}^2_{(yes,low)} \\ I_G(no) &= 1 - \hat{p}^2_{(no,high)} - \hat{p}^2_{(no,low)}, \end{aligned} \tag{II.9}$$

and then taking the weighted average to obtain the Gini impurity

$$I = w_{yes} \cdot I_G(yes) + w_{no} \cdot I_G(no). \tag{II.10}$$

where $w_{yes} = \frac{N_{yes}}{N_f}$ and similarly for w_{no}. Total observations for a given feature is $N_f = N_{yes} + N_{no}$. Looking at the Gini impurities in fig. II.2 we would pick "Breakfast" as the root node for the greater decision tree as it has the lowest Gini impurity. The Gini impurity calculation is then repeated for the two remaining features for the next two edges, respectively. An example of an end result is already presented in fig. II.1, where the final leaf nodes in this example is the class/outcome with highest probability for that path. A last point to be made; On the branch "No" to question "Dinner" on the left branch of our tree in fig. II.1, why didn't we add the question "Good for kids"? If the Gini impurity of an added node gives a higher Gini impurity than the node representing the relevant truth value to the prior question("No" to "Dinner"), then adding the new question("Good for kids") increases the impurity of that branch. We would thus defer from adding the new question and simply make the truth valued node of the prior question the final leaf node. We may now send test data that was not used to build the model through our decision tree to predict whether it would be given a predicted rating of high or low.

Decision trees are relatively easy and fast to construct, and they produce models which are intuitive and easy to interpret for smaller data sets. As we just saw in our toy model, new questions were rejected at certain branches. This illustrates how decision trees make internal feature selection easy and automatic, as it is a defining attribute of its construct. This property makes them highly resistant to the problem of including irrelevant predictor variables in a model. [4, p. 352].

C. Random Forest

Continuing the discussion from section II B above. There are downsides to Decision Trees, and the main issue is summed up well by Hastie, Tibshirani and Friedman "Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy"[4, p. 352]. Decision trees rarely provide the accuracy that we would want. The error on a test sample is commonly relatively big compared to the

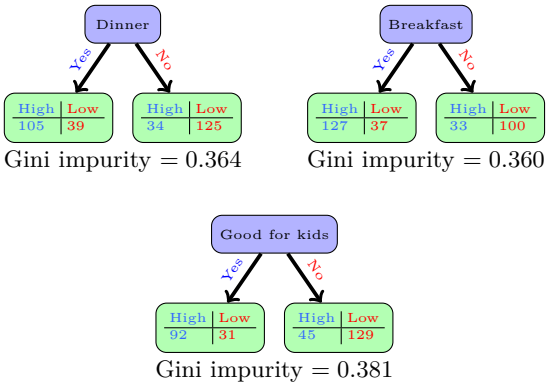


Figure II.2. Comparing the decision trees with some fictional numbers. The high/low values represent the number of people in the data set that gave high/low rating given the truth value yes/no they chose. Breakfast gives the lowest Gini impurity and would thus be placed as the root node, as was done in fig. II.1. The values are borrowed from, and the figure inspired by the educational videos at StatQuest [5]

error on the training sample. To tackle this we may give up some of the mentioned gains of Decision Trees for an acceptable cost. One category of approach is called *bagging*(portmanteau of bootstrap and aggregating), where the method of *Random Forest* resides as its proudest member. The cost of using Random Forest compared to Decision Trees is mainly realized in our computational expenses and to the ease of interpretation. The latter point simply means that when we bag a model, the simple structure of Decision Trees is lost, as it is no longer a tree [4, p. 352]. On the other hand, the advantage of Random Forests is that it greatly improves accuracy.

So what do we mean by bagging? Suppose we have some training data that we want to fit a model to, say

$$\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \tag{II.11}$$

. Where the x_i variable is our instance row of features, and y_i our target. We want to find a prediction $\hat{f}(x)$ for any instance input x. We begin by creating a *bootstrapped* data set out of our training data. This simply means we pick out at random a predefined number, N, of instances from our training data set and place them in our new bootstrapped data set $\mathbf{Z}^*$. We may continue creating several bootstrapped data set, giving $\mathbf{Z}^{*b}, b = 1, 2, \ldots, B$. We sample *with replacement*, meaning that we are allowed to pick the same instance twice. This also means that for a large data set we will by the law of large numbers have instances of the original data set that were not used in the bootstrapped data. We shall soon see that this is convenient.

In table II.1 we see a fictional example of the process of bootstrapping and then building a tree from the bootstrapped data set.

i	Breakfast	Dinner	Good for kids	High Rating
1	Yes	Yes	No	Yes
2	Yes	No	No	No
3	Yes	Yes	No	Yes
4	No	Yes	Yes	Yes

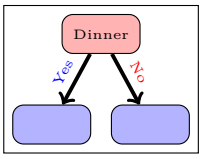
Step 1. Bootstrapping

i	Breakfast	Dinner	Good for kids	High Rating
1	Yes	Yes	No	Yes
1	Yes	Yes	No	Yes
4	No	Yes	Yes	Yes
2	Yes	No	No	No

Step 2. Randomly pick two feature-columns

i	Breakfast	Dinner	Good for kids	High Rating
1	Yes	Yes	No	Yes
1	Yes	Yes	No	Yes
4	No	Yes	Yes	Yes
2	Yes	No	No	No

Step 3. Lowest impurity of the two feature-columns gives the root node



Repeat from step 2. using the remaining columns, at each node respectively

i	Breakfast	Good for kids	High Rating
1	Yes	No	Yes
1	Yes	No	Yes
4	No	Yes	Yes
2	Yes	No	No

Table II.1. This figure shows the process of bootstrapping and building a tree from the bootstrapped data. The latter is done by picking a random subset of features-columns. In this case the subset size was chosen as two

Typically we would want to repeat the whole process from step 1. seen in table II.1 a great number of times. Meaning that we create many bootstrapped data sets $b = 1, 2, \ldots, B$ and create a tree for each one. Thus the example table II.1 would then correspond to $\mathbf{Z}^{*b=1}$. An imagined result of trees made from $\mathbf{Z}^{*b}, b = 1, 2, \ldots, B = 4$ bootstrapped data sets is shown in fig. II.3. Each of the trees represent our fitted models giving us the prediction $\hat{f}^{*b}(x)$.

Now, we can use this model with a test data set. Let's say our test data is just a single rater, e.g. as in table II.2.

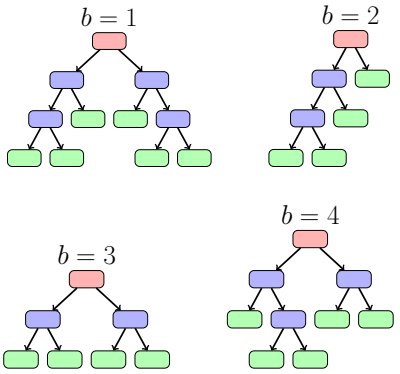


Figure II.3.

Breakfast	Dinner	Good for kids	High Rating
Yes	Yes	No	Yes

Table II.2.

Starting with the tree $b = 1$ we would now check which feature the first node corresponds to and evaluate the truth value according to the feature-values of the rater. This leads us down one branch path until we reach a leaf node and obtain a target value of either *High Rating* $= Yes/No$. Continuing with the same procedure for $b = 2, 3, 4$ we could end up with something like in table II.3.

High Rating	Low Rating
3	1

Table II.3.

.
For the *Aggregate* part of *Bagging* we have two possible cases;

- When predicting numerical outcomes the aggregation averages over the versions
  $\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$
- While when predicting a class the aggregation picks the most frequent outcome.

[6][4, p. 282] In our table II.3 we would thus pick *High Rating*.
One last point to be made is that through this method of bagging we have obtained a validation data set that is free to be used. As mentioned before we *sample with replacement* hence we will with a large data set most certainly have a large number of instances that were not included in the bootstrap data sets. These instances are called *out-of-bag* data set. In table II.1 we see that $i = 3$ was not used in the bootstrapped data set and this instance thus part of the *out-of-bag* data set. We may run our *out-of-bag* data set through all the trees in our model and obtain an *Out-of-bag Error*. This is essentially a validation error that helps us asses the quality of our model.

D. Ada Boost

In Random Forest we had no fixed set depth of our trees, as can be seen in fig. II.3. In *Ada Boost* we pre-set the number of nodes, usually only a root node with two terminal leaf nodes, as in fig. II.4. These types of trees are called *stumps*, and in Ada Boost we essentially have a forest of many weak learners, typically such stumps.

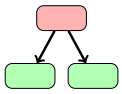


Figure II.4. Ada boost stump example. One root node, and two leaf nodes.

In the case of Random Forest we gave the same importance to every tree in the quest of predicting the target variable. So every tree in e.g. II.3 has an equal vote in the final election. Ada Boost on the other is not as democratic as it does not necessarily give an equal vote(weight) to the trees(shown figuratively in fig. II.5 with different colour contrasts representing weight).

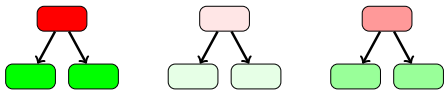


Figure II.5. Ada boost stump example. Where the contrast of the color represents different weight to the stump. The contrast/weights in this example are $100\%, 10\%, 40\%$ respectively

The third point of difference from Random Forest is that each stump considers the prior mistakes of other stumps. Before we go further into detail we can summarize the three defining properties of the method

1. Our model consist of *Weak Learners*, typically *Stump trees*
2. Each Stump tree has a different weight.
3. Each stump tree takes into consideration the prediction of the previous stump

The actual calculation algorithm is as follow

1. We begin by initializing weights on each instance of our data set. These are all simply set to $w_k = 1/N, k = 1, 2, \ldots, N$.

2. For every iteration $m = 1, 2, 3, \ldots, M$ we individually modify each weight in the following way

   (a) Calculate the Gini index for every stump in the data, pick the stump with the lowest Gini index as this round's stump. For $m = 2, 3, \ldots, M$ we use a *Weighted Gini Index* where we, simply put, calculate the probabilities of the Gini Index weighted by the relevant sample weights

   (b) Find the incorrect classifications and sum the corresponding weights. In the $m = 1$ example table II.4 this would simply be $err_1 = 1/4$ for "Good for kids", as there is only one instance leading to an incorrect prediction.

   (c) Calculate *The Amount of Say*,

   $$\alpha_m = \log\left((1 - \text{err}_m)\,/\,\text{err}_m\right) \tag{II.12}$$

   (d) Increase the sample weights of the instances i corresponding to an incorrect classification

   $$w_i \leftarrow w_i \cdot \exp\left(\alpha_m\right) \tag{II.13}$$

   (e) Decrease the sample weights of the instances j corresponding to a correct classification

   $$w_j \leftarrow w_j \cdot \exp\left(-\alpha_m\right) \tag{II.14}$$

   (f) Normalize sample weights so that $\sum_k^N \omega_k = 1$

[5][4, p. 339]

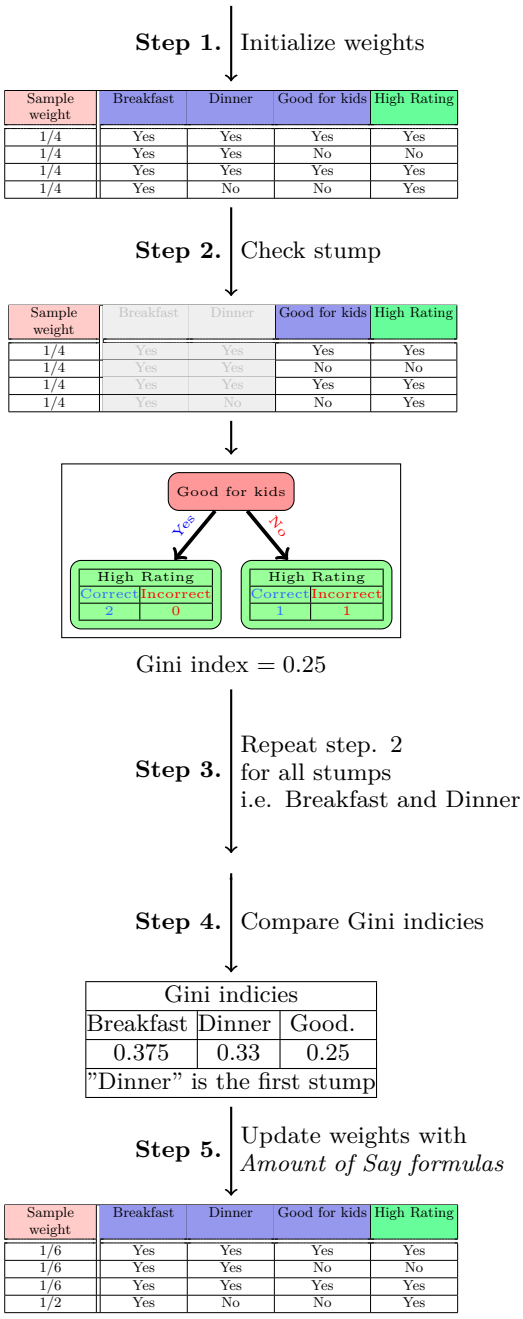


Table II.4. Rough sketch of first round($m = 1$) of an Ada Boost iteration scheme. Values are fictional and just for illustration. [5]

E. Metrics

There exists a plethora of metrics for quantifying the quality of classification. As this is a binary classification problem with class imbalances remedied, the metrics *accuracy*, *ROC-curve* and *confusion matrix* were chosen. Others such as *F1* and *precision-recall curves* were considered but left out as they do not provide any more insight.

1. Accuracy Score

The accuracy score measure the ratio of correct classification to the total population:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I\left(t_i = y_i\right)}{n}. \tag{II.15}$$

where t_i is the target data points and y_i is predicted data points. Accuracy score is widely used in classification methods, but it is not necessarily giving the wider picture. Therefore in this project we have made use of many different metrics to see the full picture of how our models perform.

2. ROC and AUC

The *Receiver operating characteristic*(ROC)[3] is a curve often used for graphical illustration to asses the quality of a model and to find the optimal classification. In the case of the Yelp data one might classify a prediction(a probability), as 'probably *top rated*(1)' or 'probably not top rated(0)'. To be able classify the

data one must define a threshold, say 50%. Meaning that those with over 50% chance will be classified as *top rated*. In this case there might be those that did are not top rated, but were incorrectly classified as *top rated*(above 50%), this is called *False Positives*. There might also be those who did where classified as top rated, but were incorrectly classified as such, these cases are called *False Negatives*. The two other possibilities is when we classify correctly: *True Positives* and *True Negatives*. If we change the threshold we will get different numbers of the four scores above, and this is exactly what is done with the ROC curve. A value on the horizontal axis, for a given threshold is calculated by

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{II.16}$$

and the vertical axis by

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}. \tag{II.17}$$

A diagonal line is also usually drawn to graphically show where $TruePositiveRate = False$. The ROC is simply put a graph that plots the *hit rate* against the *false alarm rate*

a. AUC The *Area Under the Curve*(AUC) is exactly that, the area under the curve of the ROC curve. This metric can be used to compare one ROC curve with another ROC curve to assess which one has the greatest area under the curve, and thus is the best model.

III. METHOD

A. Yelp Data

The goal of this project was to explore different classification algorithms to see how they perform on Yelp data and if they are able to classify if a business gets top rating or not. The methods explored were *Logistic Regression* with *Stochastic Gradient Descent*, *Decision Trees*, *Random Forest*, and *AdaBoost*. The data is taken from the Yelp data challenge which offers disparate datasets containing about 4.5 million reviews along with data about the business. A subset of 500 000 reviews were extracted and used in this analysis. The dataset was created by concatenating a subset of predictors from a business, a user and a review, with the `rating` variable of the review used as the response.

Considerable preprocessing was performed on the data, with the most important parts summarized listed below. A notebook is available in the GitHub repository to reproduce the preprocessing.

1. Preparing the data
   - The dataset contains information about the business which the customer can choose to give information about whilst rating. Some are deemed as categorical such as `restaurant take out`, `trendy` and `desserts`. These are encoded by creating indicator columns for NaN, $True$ and $False$.
   - The quantitative predictors such as `review count` were shifted by their mean and scaled by their standard deviation.
   - Several predictors such as `postal code` and `city` were dropped as no good encodings were found.

In order to examine the distribution of `rating`, its values are plotted in III.1. At a glance it is clear that there are large class imbalances. Class imbalances have been an active area of research the last decade, providing several methods to deal the problem. To limit the complexity of this project, a simple solution was employed, namely rebinning the data so that the classes become more equal. Whenever data is binned there is some arbitrariness introduced, unless one has a good reason for it. It our case, it is reasonable to consider a 5 star rating as being "outstanding", a 1 star rating as "awful", and in-between ratings as different shades of "ok". Two reasonable binnings are therefore "outstanding" vs "non-outstanding" and "outstanding" vs "ok" vs "awful. The former was chosen and achieved by binning all ratings less than 5 into a single < 5 bin, as seen in the second panel of III.1. The latter option was chosen against as the class imbalance problem remains unfixed. When splitting the data into training and test sets it was stratified so as to preserve the class ratios.

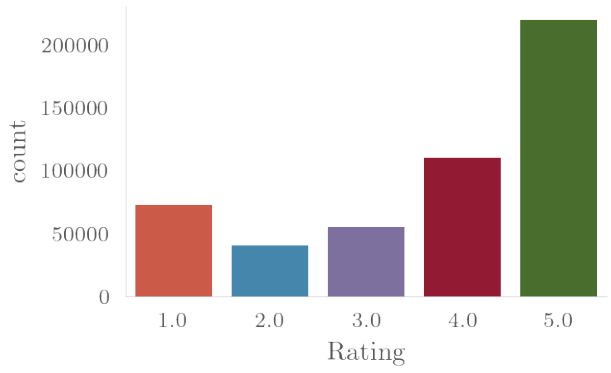


Figure III.1. Distribution of review ratings.

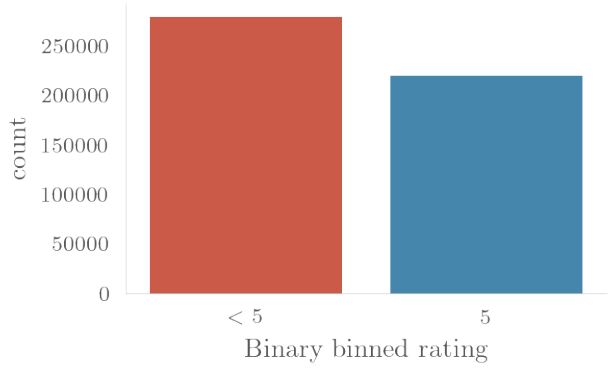


Figure III.2. Distribution of review ratings where $1, 2, 3, 4$ are binned into a single < 5 bin

Figure III.3. Distribution of review ratings where $2, 3, 4$ are binned into a single bin. Ratings 1 and 5 are left undisturbed.

1. *Logistic Regression*

The implementation of logistic regression follows the theory in section II A. The only hyperparameter tuned was penalization type (L_1 vs L_2) and regularization strength. The error was decomposed using bootstraping.

2. *Random Forest and AdaBoost*

These classification methods differ from the logistic regression because they are usually better and preferable when one has more than two classes.

1. Scikit Learn implementation

- The Random Forest can also be applied as a function in the Scikit Learn Library RandomForestClassifier. A Random Forest is a meta estimator that fits a number of decision tree classifiers on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [7]. Again there are various parameters to choose from, some are the same as for Decision Trees, like the criterion parameter. Another is the *n_estimators* which decides how many trees there should be in the forest. In this project we first iterated over this parameter and evaluated it against the accuracy score to find the optimal number of trees for our dataset.
- AdaBoost for classification, which in Scikit Learn has the name AdaBoostClassifier is also a meta estimator which firstly fits a classifier to the whole dataset but then also fits a copy of that classifier to the dataset and then aims to correct for incorrectly classified samples done by the first fit. So one of the parameters which is called *base_estimator* takes in the classifier that works as a base, say DecisionTreeClassifier, then the AdaBoost will literally boost the method trying to correctly classify the incorrect prediction of the Decision Tree method.

2. Evaluation

- All methods above where evaluated by confusion matrices using the function $confusion\_matrix$. This to see how the different methods performed with the different classes mentioned above.

All of the calculations can be reproduced by running the Jupyter notebooks.

IV. RESULTS AND DISCUSSION

A. Logistic Regression

Logistic regression was employed using both L_1 and L_2 regularization. The space over the regularization parameters was searched under bootstrapping to explore the consequence on bias and variance. The results are shown in fig. IV.3 and fig. IV.4.

The usual pattern is observed. Increasing the regularization strength initially reduces the error by decreasing the variance, but too high a regularization shrinks the solution space sufficiently such that the bias skyrockets. The change is most dramatic for L_1 as many predictors are set to 0.

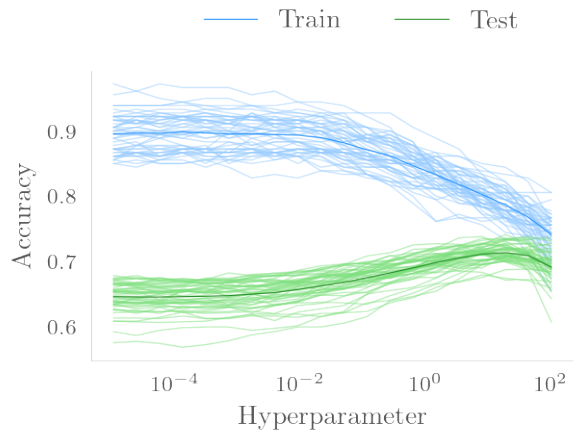


Figure IV.1. The accuracy for different bootstrap samples of training and test data.

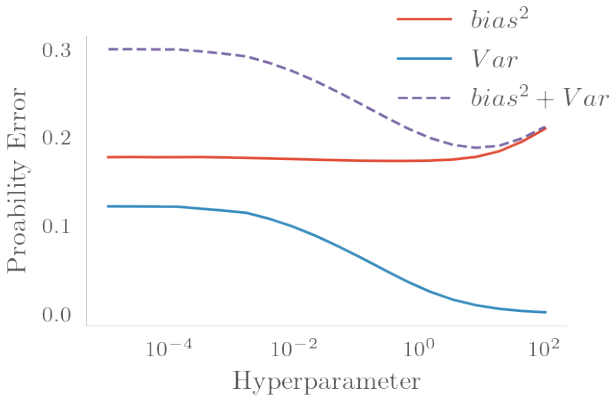


Figure IV.2. The error in predicted probability decomposed as bias squared and variance.

Figure IV.3. The quality of classification as function of L_2 regularization hyperparameter. Increasing the regularization prevents overfitting by decreasing variance, increasing the test accuracy. A too high regularization (> 1) causes an increase in bias, deteriorating the accuracy.

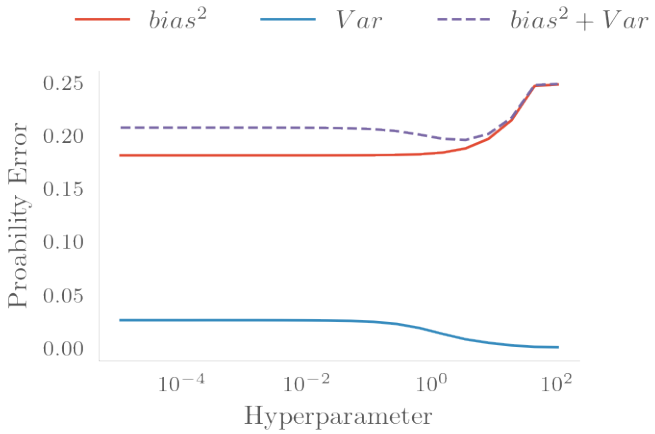


Figure IV.4. The decomposition of error as a function of L_1 regularization strength. The error changes slightly after $> 10^{-1}$ by decreasing the variance, but quickly jumps as bias grows at an increasing rate.

As L_2 and L_1 give practically identical results, only the results using L_2 regularization are shown, using the optimal hyperparameter found earlier. The resulting confusion matrix is shown in fig. IV.5. Many of the observations are classified correctly, quite a few are not. Considering that this is a binary classification, the number of 5 star ratings classified as < 5 is worrying. It gets worse when also taking into account that the number of < 5 wrongly classified as 5 is much smaller. A possible explanation for this is that the model defaults to classifying observations as < 5, requiring exorbitant high evidence to classify an observation as 5.

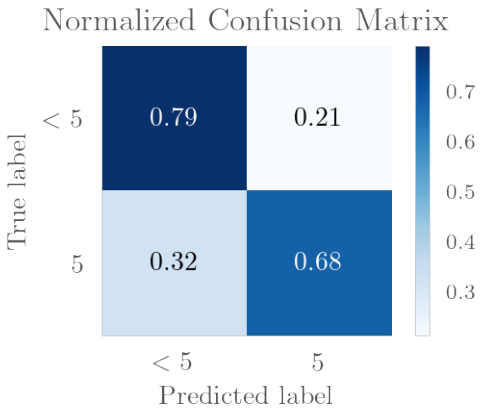


Figure IV.5. Confusion matrix for binary classification using logistic regression with L_2 penalization.

The corresponding ROC-curve is shown in fig. IV.6. The curves for both classes are nearly identical, show-

ing that class imbalances do not affect the classification to a disastrous extent.

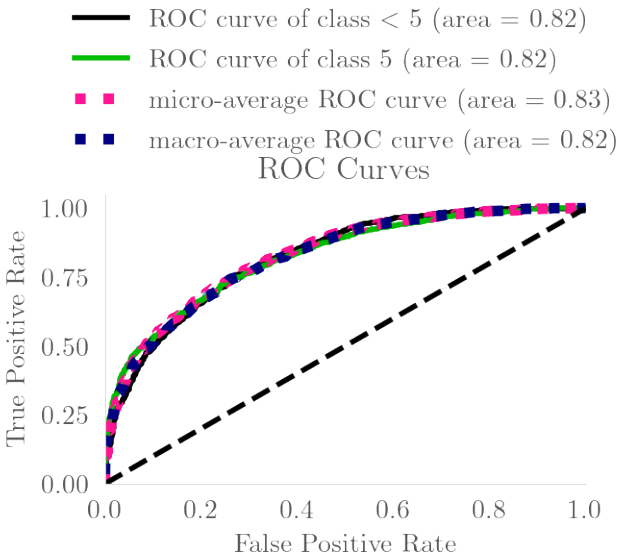


Figure IV.6. The ROC curve for logistic regression with L_2 penalization.

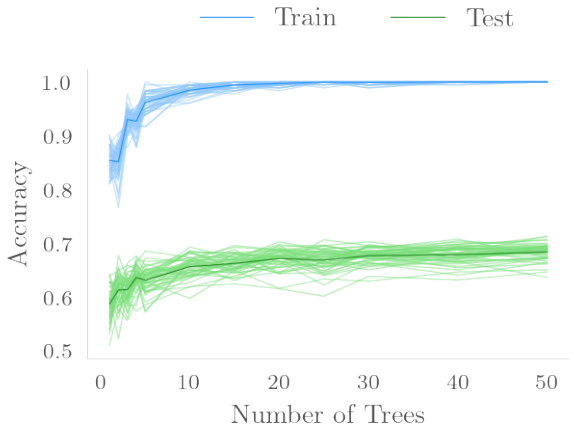


Figure IV.7. The accuracy of random forest as a function of number of trees. Notice the perfect training accuracy.

B. Random Forest

Random forests and other ensemble methods have many hyperparameters available for tuning. To limit the scope, only the number of trees and the maximum tree depth were explored. The error decomposition of each is shown in fig. IV.10. A bit surprisingly, random forest gives good results even with few trees of small depth. As there are more than a 100 predictors, this implies that many of them contain little information.

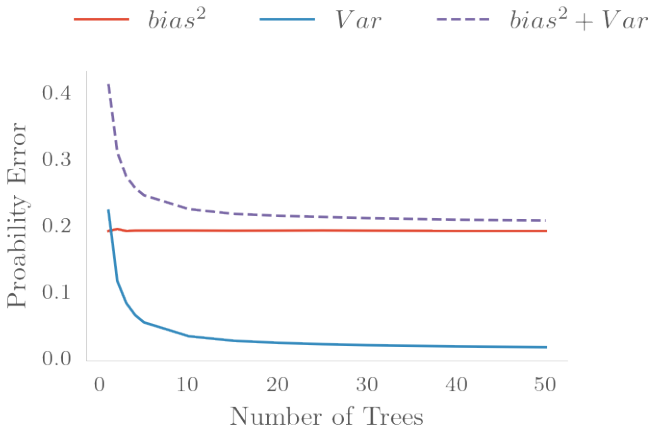


Figure IV.8. The error decomposition as function of number of trees. The depth is kept constant at 50.

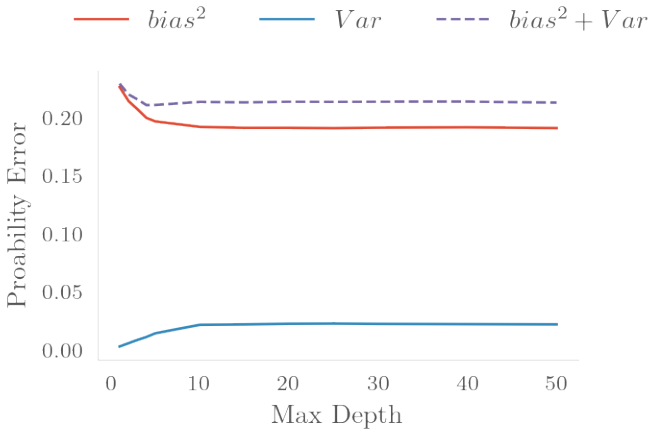


Figure IV.9. The error decomposition as function of number of depth. The number of trees is kept constant at 20

Figure IV.10. Error decomposition by bootstrapping for random forests.

It is fig. IV.7 that is the most alarming, however. Once the number of trees gets sufficiently high, the training accuracy becomes *perfect*. This indicates either that the method overfits the training set, or that the response is leaking into the predictors. As both the error decomposition and test accuracy does not deteriorate, overfitting is not the culprit. The investigation of response leakage is postponed till section IV D.
The resulting confusion matrix using the found optimal hyperparameters is shown in fig. IV.11. It is nearly identical to logistic regression fig. IV.5, being slightly better at < 5 but worse at 5. If our suspicion of defaulting to < 5 as discussed earlier is correct, it implies than random forests performes worse than logistic regression. The associated ROC curve is not shown as it is nearly identical to that of logistic regression.

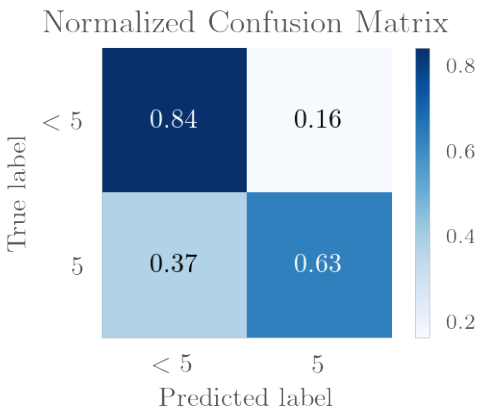


Figure IV.11. The confusion matrix for random forest.

C. AdaBoost

Just like random forest, AdaBoost provides a plethora of tuneable hyperparameters. The number of trees and learning rate were chosen, with their error decomposition shown in cIV.15 and behavior of accuracy as function of number of trees shown in fig. IV.12. Neither hyperparameter give widely different performance, only slightly decreasing variance while slightly increasing bias.

The behavior of the accuracy however, is markedly different from the earlier methods, requiring a large number of trees to give "good" training accuracy. More pointedly, the test and training accuracy start out equal[WHY] with the test accuracy reaching a maximum at just 4 trees. In contrast to random trees, this does not point to response leakage, but overfitting.

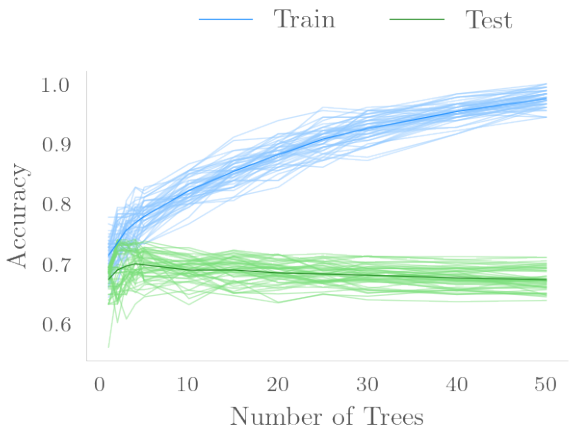


Figure IV.12. The accuracy of AdaBoost as a function of number of trees. Compare with the similar fig. IV.7.

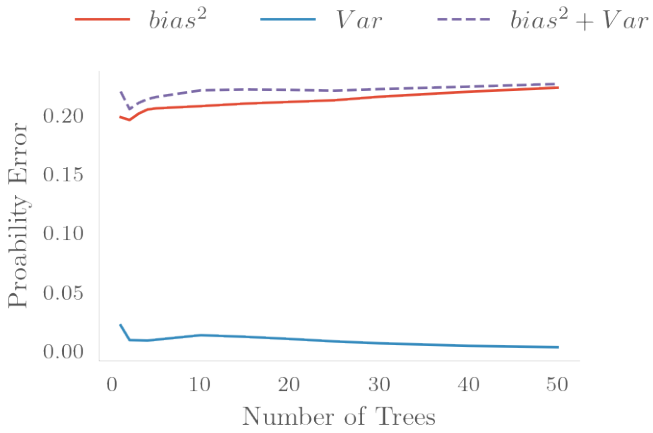


Figure IV.13. The error decomposition as function of number of trees. The depth is kept constant at 50.

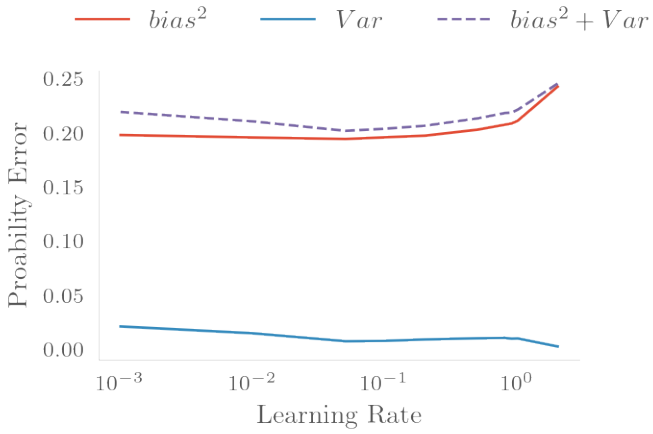


Figure IV.14. The error decomposition as function of learning rate. The number of trees is kept constant.

Figure IV.15. Error decomposition by bootstrapping using AdaBoost.

As before, the confusion matrix for AdaBoost was created used the found optimal hyperparameters. This is shown in fig. IV.16. The results are similar to and slightly better than random forest.

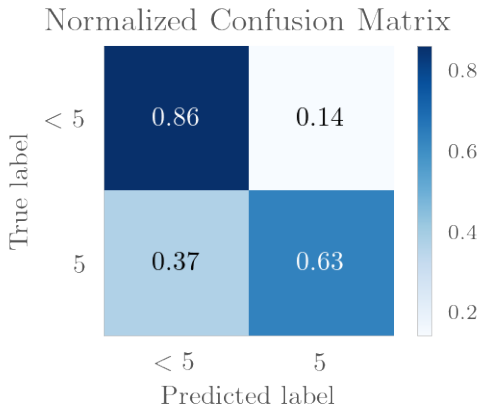


Figure IV.16. The confusion matrix from using AdaBoost

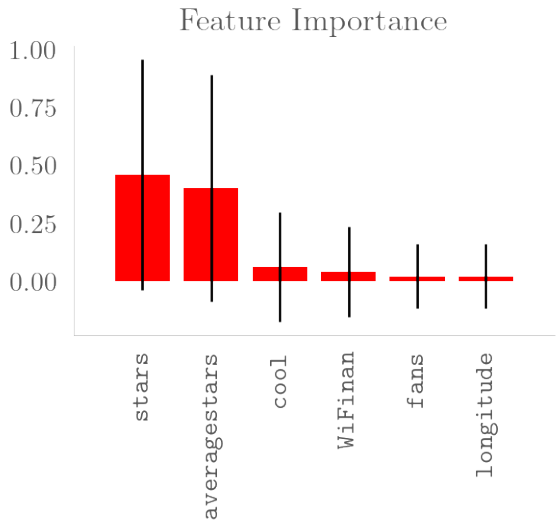


Figure IV.17. The features in the dataset ranked by importance by AdaBoost. All features not shown have been excluded by the method.

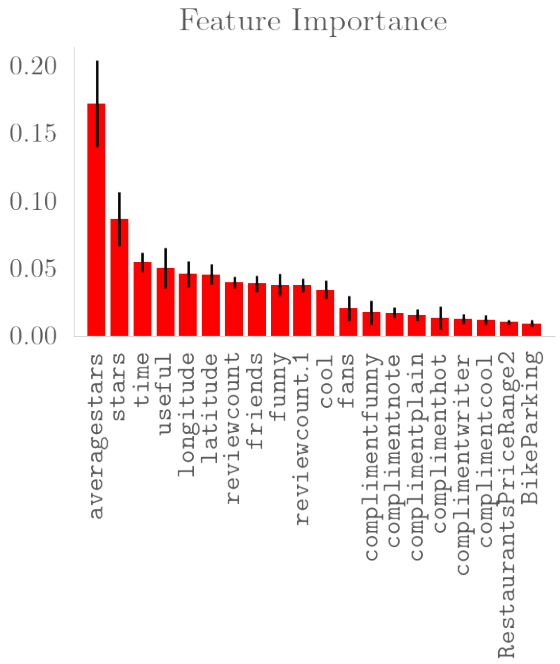


Figure IV.18. The features in the dataset ranked by importance by random forest.

D. Response Leakage

To investigate the possibility of the response leaking into the features, the random forest and AdaBoost models are used to extract the most important features used by each. This is shown in fig. IV.17. Immediately the possible culprits reveal themselves. Both methods rely heavily on `average stars`, describing the average number of stars given by the user, and `stars`, the average number of stars given to the business. If the number of `average stars` and `stars` is large, there should be no response leakage due to the fact that there is no way to connect which observation of one corresponds to the other. However, if the number is small, they would be highly predictive of the response simply because they *equal* the response.

To investigate the relationship between the mentioned features and the response `rating`, they are plotted by violin plots in fig. IV.19. For all values of `rating` there is significant spread in both `average stars` and `stars`, but also a clear correlation: users that give rating 1 to a business tends to give more 1s overall; vice versa for 5, and slightly weaker for the other ratings. Interestingly rating 3 and 4 are centered around 3 and 4, suggesting that users that give "ok" ratings are precisely those

that tend to give "ok" ratings. The same goes for the `stars`: businesses that tend to get 5 stars are more likely to get more 5 stars. There is a weaker correlation between the other `stars` ratings.

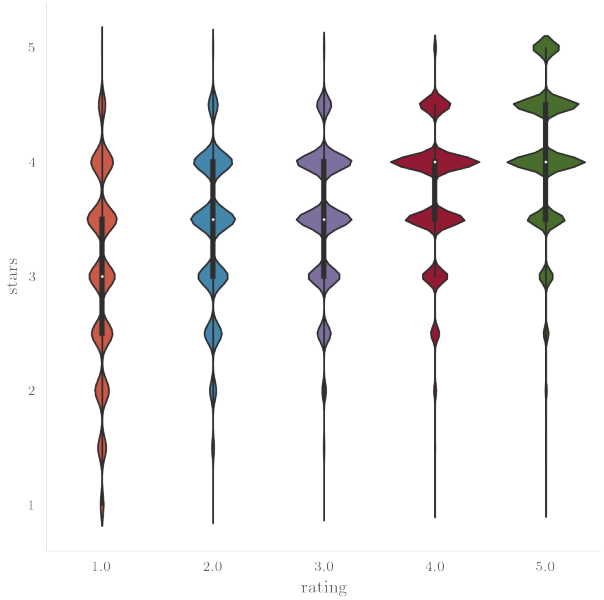


Figure IV.19. The response `rating` plotted against `stars`, the average number of stars given to the business.

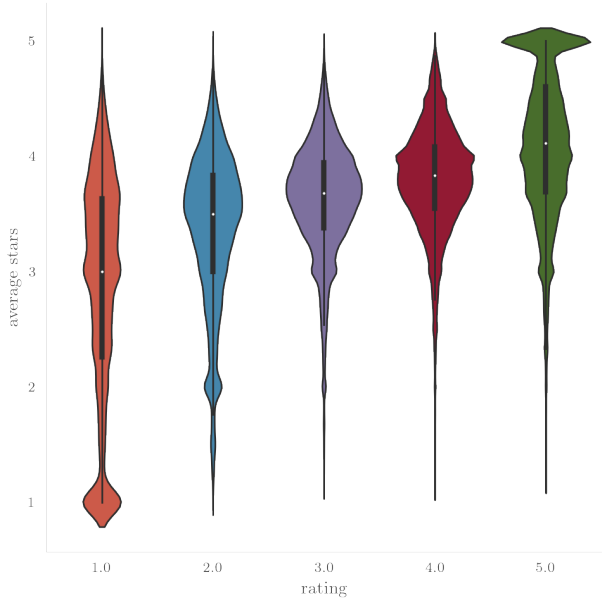


Figure IV.20. The response `rating` plotted against `average stars`, the average number of stars given by the user.

In addition, there is reason to not be too worried about response leakage as AdaBoost relies much more on these two features than random forest, but it is the random forest that gives perfect training accuracy while AdaBoost gives similar accuracy for both training and test.

To summary, there is the possibility of the response leaking into the features, but this is not strongly supported. The more likely explanation is a type of "rating inertia" for both businesses and users. Those businesses that have generally high ratings are more likely to get more favorable reviews, and users that tend to give high ratings are more likely to give favorable reviews. An interesting question then becomes: what happens when `average stars` and `stars` are removed from the dataset?

E. Hamstringing the Data

To investigate the effect of `average stars` and `stars`, they were removed from the dataset and the analysis was repeated. The results were very similar across the methods, with the most important behavior summarized the confusion matrix in fig. IV.21. The classification of < 5 is impressive, but alas, the classification of 5 is abysmal, not better than chance. This supports our previous suspicion that the algorithms defaults to < 5, requiring unreasonable evidence to classify something as 5.

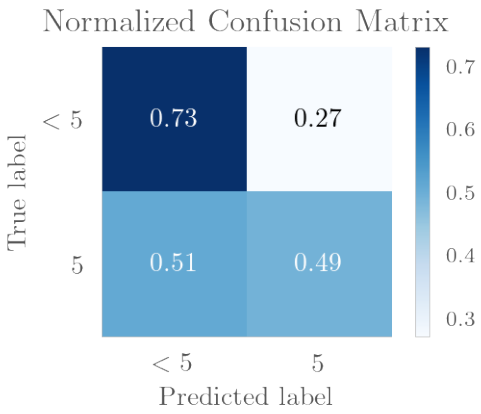


Figure IV.21. The confusion matrix for Yelp data with `stars` and `average stars` removed. The class 5 is nearly completely mislabeled. The method used was logistic regression with L_2 penalty. The ROC AUC was 0.62.

V. CONCLUSION

After applying various classification algorithms on the Yelp data, it was found that logistic regression was sufficient enough for this type of analysis, as more advanced methods provides no significant benefit.
There was found support for "rating inertia" where businesses with high rating are more likely to receive favorable reviews and users who tend to give high rating too are more likely to give favorable reviews. This leads to the rather unhelpful advice for businesses to avoid low ratings and bar low rating customers from entry.
Removing previous ratings from both business and user greatly deteriorates the performance of the classifiers, suggesting that the remaining factors rather uninformative. However, that a feature does not contribute to a model's predictive power does not mean that the feature is not important. Customers may very well enjoy having free WiFi, increasing their likelihood of giving favorable reviews, but perhaps omits this meta data from their review. In other words, it can be that the Yelp dataset is too incomplete to be useful in predicting rating from metadata. It may also be the case that our methods were too simple in analysing the data.
The analysis performed gives no support for our initial hypothesis, that one can use the business and user meta data to predict reviews for a business in question.

[1] Yelp, "An introduction to yelp metrics as of september 30, 2019," (2019).
[2] M. H. Jensen, *Data Analysis and Machine Learning: Logistic Regression* (Department of Physics, University of Oslo, 2019).
[3] V. M. V. . P.-D. B. Sønderland, *Classification and Regression, From Linear and Logistic Regression to Neural Network Project 2* (Department of Physics, University of Oslo, 2019).
[4] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning* (Springer, 2017).
[5] J. S. statquest.org, "Statquest," (Unknown).
[6] L. Breiman, Machine Learning (1996), 10.1007/BF00058655.
[7] scikit learn.org, "3.2.4.3.1. sklearn.ensemble.randomforestclassifier," (Unknown).