Machine Learning Project: The Simpsons Challenge

Felicia Burtscher, Thomas Hadler, Hanna Wulkow

***Abstract*—In this project, a data set of Simpsons character pictures was used to test the validity of a random forest implementation. Decision trees are a machine learning method based on the idea of human decision making. Random forest are an expansion of that. The pictures from the data set were preprocessed using hog features. The resulting random forests were created using a training set and validated using a smaller test set. The results were outstandingly good: even for small random forests well more than 50% of the characters could be classified correctly.**

I. Task Description

Our data set consists of images of 20 characters of *The Simpsons* with labels. The data originates from the Kaggle contest *Simpsons by the Data*. There are 200 to 1500 examples per character offering an unbalanced data set. The images have different shapes and the characters are scattered throughout the images. Our goal is to develop an algorithm which given an input image containing a Simpson character is capable to classify, i.e. recognize the Simpsons character in the image correctly. In *The Simpsons* data set this is the character the image shows. We are looking for a function that maps the input images to the labels of the images.

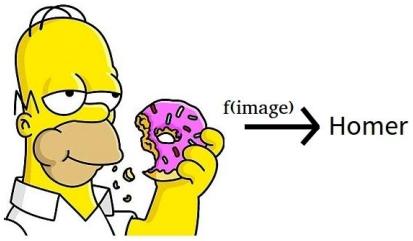

Figure 1: Homer Simpson

II. Mathematical Description of Images

We work with images as input to our classification algorithm. Thus we describe the nature of our input here first. Images are represented as tensors of rgb values. *The Simpsons* data set consists of images of approximately 110×110 pixels, each with three values in $\{0, ..., 255\}$ representing the red, green and blue intensity of the specific pixel. This means that each image can be mapped to a 36300 dimensional vector with values in $\{0, ..., 255\}$. These vast dimensions illustrate the difficulty of this problem and show that it is not trivially solvable. We have 256^{36300} different possible vectors. Estimating a vector in a 36 300 dimensional vector space would be hard enough, estimating a vector of this dimension with only a couple thousand examples makes this practically impossible without exploiting additional information.

III. Preprocessing the Data

We use several preprocessing steps to transform the images to useful input data. The first step deals with the rescaling of the images to same shapes. The second step reduces the dimensionality of the problem by exploiting features. The resulting feature image is then the input to the algorithm.

A. Rescaling the Data

As mentioned in the paragraph before, initially the images come in different shapes. The width is between 70 and 170 pixels, the height is between 80 and 180 pixels. Since the classification algorithm takes a fixed number of input values, the images must be transformed into the same shape.

Figure 2: Unscaled Nelson (left), Scaled Nelson (right). Best viewed on a monitor.

We took the simplest route to achieving this and used bilinear interpolation to scale all images to the shape 110×110 with three rgb values each. This then results in the 36300 values mentioned before.

B. Features

This section elaborates on features in general first. Then we move on to Histograms of Oriented Gradients (HOG), a feature descriptor developed by Dalal and Triggs in 2005 for human detection in images. We use this algorithm to create the feature inputs for our classifier.

1) Definition: A feature is a piece of information which is relevant for solving the computational task at hand. The idea of a feature is to reduce the vast amount of information given with the data to the relevant information within it. The assumption of a feature is that a large amount of the given information is essentially redundant or inconveniently expressed and can be reduced to a far more compact representation without losing a large amount of the overall information. This compact representation is called a feature.

2) Computer Vision Features: When working with images, video, ct-scans, etc. a computer vision specific dependency begins to take the spotlight. Pixels that lie close together tend to interact more strongly than pixels far apart. Within an image the pixel in the bottom left corner and the top right corner have little to do with each other. But pixels within a region can form larger features like corners, circles, noses, mouths or a fur-like texture. This locality- or neighborhood-characteristic is what is commonly used to create features in computer vision problems.

3) Histogram of Oriented Gradients: We recommend reading the original HOG-paper since it is very thorough and the exact execution of steps would be hard to elaborate on here. The procedure for creating the HOG-image from the original is roughly as follows:
1) Compute the gradients
2) Weighted vote into spatial and orientation bins
3) Contrast normalize over overlapping spatial blocks

Compute the gradients
Dalal and Triggs tested several derivative masks but the simple mask $(-1, 0, 1)$ for x-direction and $(-1, 0, 1)^T$ for y-direction proved to be best. This mask is applied to every pixel's colors.

Weighted vote into spatial and orientation bins
Now histograms of oriented gradients are created. To do this, the image is divided into cells of $n \times n$ pixels. For each cell, a bin is created with a certain number of orientations. Every orientation spans a certain range of gradient direction. For example, four bins could mean that bin one collects all gradients within the cell that have an angle between $0°$and $90°$. Every gradient is now interpolated between the closest orientations and between the closest cells it belongs to.

Contrast normalize over overlapping spatial blocks
Instead of simply using these pooled interpolated gradients as inputs to the classifier the histograms are normalized several times with their neighboring cells and concatenated to a vector for each cell.

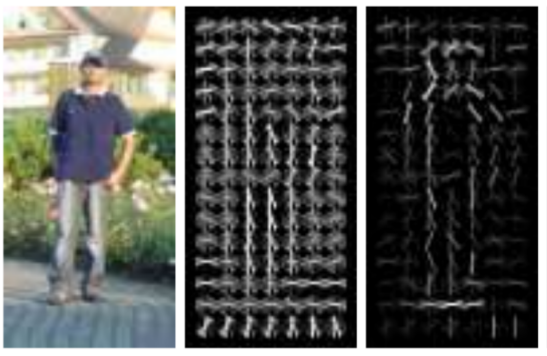

Figure 3: Visualization of HOG feature-image. Extracted from [3]. Best viewed on a monitor.

IV. Methods

The main algorithm we use is *Random Forest* which is an adaptation of decision trees.

A. Decision Trees

Decision trees are built by a *greedy* algorithm. The algorithm searches through each independent variable (here attributes) to find a value of a single variable that best splits the data into two (or more) groups. Typically, the best split minimizes the impurity of the outcome in the resulting data subsets. For these two resulting groups, the process is repeated until a stopping criterion applies.
Decision trees are easy to understand and interpret and robust to noisy data. However, using information gain they are biased in favor of those attributes that give lots of subsets. Also, calculations can get very complex, particularly if many values are uncertain. At the same time they often suffer from high variance, i.e. they tend to overfit. A decision tree keeps splitting up the data until it ends up with pure sets, so it will always classify the training example perfectly. As the tree gets bigger and bigger, it becomes more accurate on the training data but at some point it will become less accurate on future test data, that is data we have not seen before. For a bias-variance trade-off we want to introduce randomization: we combine the prediction of several randomized trees into a single model.

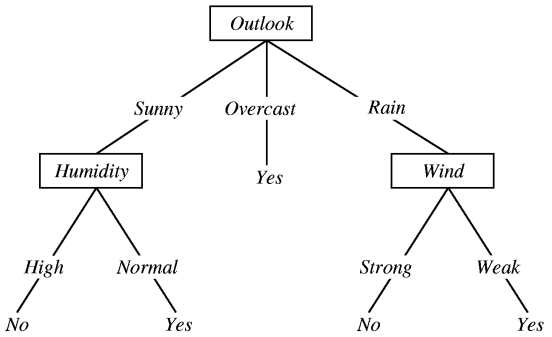

Figure 4: Example of a decision tree from [5]

B. Random Forest

Instead of growing a single tree, Random Forest grows a large number of different decision trees. There are two main differences to the construction of decision trees: First, from the set of training data D we choose a random sample D_i. Second, at each node we choose a random subset of features, S, and only consider splitting on those features. Each tree is fully grown and not pruned.
Using majority vote, the decision tree then finds the joint set of variables that produce the strongest classification model.
Random Forest is one of the most robust and effective machine learning algorithms for many problems.

We use the C4.5 algorithm that uses information gain as the splitting criterion. Growing the decision tree from the root down, it greedily selects the next best attribute using information gain and then recursively calls the algorithm on its child nodes. The growing stops when all instances belong to a single value of target feature – we call this a "pure" subset – or when the best information gain is not greater than zero

or any other stopping criteria defined such as tree depth.
It does not apply any pruning procedures nor does it handle numeric attributes or missing values. The following pseudo code taken from [1] explains the algorithm.

Algorithm 1 C4.5(D)

Require: an attribute-values data set D
  Tree = {}
  if D is "pure" OR other stopping criteria met **then**
    terminate
  end if
  for all attribute a $\in D$ **do**
    Compute information-theoretic criteria if we split on a
  end for
  a_{best} = Best attribute according to above computed criteria
  Tree = Create a decision node that tests a_{best} in the root
  D_v = Induced sub-data sets from D based on a_{best}
  for all D_v **do**
    Tree$_v$ = C4.5(D_v)
    Attach Tree$_v$ to the corresponding branch of Tree
  end for
  return Tree

The idea of decision trees is rather simple; the question that is not so trivial is: How to select the best attribute?

C. Information Gain

For the sake of of simplicity, assume we have a binary decision problem. Our hypothesis is "Homer Simpson is in the picture". We therefore have positive examples (i.e. supporting our hypothesis) and negative examples in our subset S. Our goal is to get items in pure sets, that is either only positive or only negative examples. This explains why a simple posterior probability computation falls short as it is not symmetric and *all negative* is as pure as *all positive*. As an alternative we can use entropy from information theory.

1) Entropy: Entropy measures the uncertainty of a class in a subset of examples S. It is defined by

$$H(S) = -p_+ \cdot log_2 p_+ - p_- \cdot log_2 p_-$$

where S is the subset of training examples and p_+ and $-p_-$ are the positive and negative examples in S.
In our multi-class classification problem ("What character is seen on the picture?") this equation generalizes to

$$H(S) = -\sum_{i=1}^{n} p_i \cdot log_2 p_i.$$

We can interpret the entropy as follows: What is the amount of bits we need to convey, i.e. how many bits do we need to spend to tell if x is positive or negative assuming that it belongs to X? As a result, entropy tells us how pure (entirely "negative" or "positive") or impure one subset is. For instance, if a subset is pure, we have 0 bits whereas if it is (fully) impure (say 50% positive, 50% negative) we need 1 bit to be certain. In general, binary entropy looks like the plot in Fig. 5.

Now we want to aggregate information from several different subsets, e.g. we are splitting on a certain attribute with

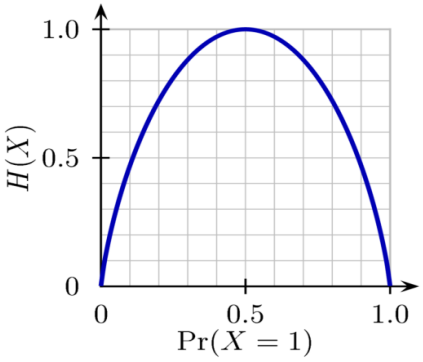

Figure 5: Binary entropy plot from [6]

several different subsets and each of these subsets will have different purity value. When putting them together we want to average their purity values. Since we also want to take into consideration how much splitting we might have left, we weigh each subset according to its size, so the proportion that fall into this subset, namely $\frac{|S_V|}{|S|}$.
Our aim is to have items in pure sets. In order to decide whether to split on a feature a or not, we compute the expected drop in entropy, *Gain(S,a)*, after the split

$$Gain(S, a) = H(S) - \sum_{V \in values(a)} \frac{|S_V|}{|S|} H(S_V)$$

where V are the possible values of a, S is the set of examples X and S_V is the subset where $X_a = V$. This is also known as *mutual information* between attribute a and class labels of S.

2) Gini index: As an alternative to entropy we can use the *Gini index* as a measurement for information gain but we will not go into this any further for now.

V. Project Structure

We implemented the Random Forest as implied above, following the pseudo code in a python implementation. This implementation was handed in as well. The following UML diagram explains the relationship between the classes used in the implementation:

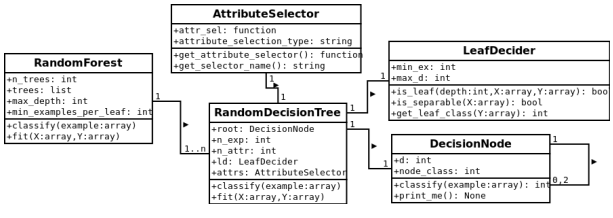

Figure 6: UML Diagram, best viewed on monitor

A Random Forest carries a list of Random Decision Trees. The Random Forest's `fit()` method calls the `fit()` method of each Random Decision Tree in its list. The `classify()` method lets the trees in the Random Forest's list democratically vote on the classification.
The Random Decision Tree carries at an instance each

of `LeafDecider` and `AttributeSelector`. The tree's `fit()` method constructs a tree as implied above. The tree's `classify()` method propagates the example to be classified through the tree structure.
The `AttributeSelector` contains all the code concerning how a node decides on how to choose the attribute on which it splits the data. A Decision Tree's `AttributeSelector` would pick the "best" splitting value of all possible attributes to split the data on. A Random Decision Tree's `AttributeSelector` would pick the "best" splitting value of $\log_2$(number of all possible attributes) to split the data on. The `LeafDecider` is a class that gives all nodes the capability of deciding whether they're a leaf (termination of recursion) or not. The `LeafDecider` checks things such as whether the maximum tree depth is reached, the number of examples in the leaf are still statistically significant, etc.

VI. Results

A. Evaluation Criteria

In the following chapter, the efficacy of the random forest approach is evaluated. We will be applying two classifier metrics to our random forests. We will first define true and false positives, true and false negatives. We will then apply the two metrics, accuracy and confusion matrix to two very different random forests, one with many deep trees and one with few shallow trees.

1) Fundamentals:
True Positives: examples that belong to class A and are classified as belonging to A
False Positives: examples that do not belong to class A but are classified as belonging to A
True Negatives: examples that belong to class B and are classified as such
False Negatives: examples that do not belong to class B but are classified as such

2) Accuracy:
The accuracy can be understood as the percentage of correctly classified examples. It is defined as:

accuracy(Examples) = (#TP + #TN) / (#Examples)

The table in Fig. 10 shows the accuracy of different combinations of the two parameters. The test data consists of almost 1000 pictures of the Simpsons characters, with about 50 pictures per person. The random forest that is built has never seen these pictures before.
The table shows that the accuracy is almost always monotonously increasing in the direction of both the depth and of the number of trees in the forest.
The best result can be found in the entry corresponding to 50 trees with depth 50, with an accuracy of 83%. This is a convincing result.

3) Confusion Matrix: The confusion matrix is a kind of contingency table. It has two axes, real class and predicted class. It then contrasts all examples by real class to the class predicted by the classifier. In our case the confusion matrix has 20 rows and 20 columns. The entry (i, j) corresponds to the percentage of the character i's classification as character j. It is noteworthy that this matrix is not typically symmetrical: mistaking Marge for Homer is not the same as mistaking Homer for Marge.
The accuracy shows positive results. But the quality of information given to us by the confusion matrix is vastly superior to the information of the accuracy alone. In other words, if the forest struggles to classify Marge Simpson, that would not be a good sign because Marge should be easily distinguished thanks to her blue hair 7. If the forest confuses Bart with Lisa, on the other hand, this can be borne, given that they both have spiky hair.
The matrix is plotted in heat map style, which - instead of the actual percentage - darkens the color of a cell if a cell has a high number and lightens the cell if the number is low. For example, if the cell $i \times j$ is white, the character i has not been classified as character j at all.
If the method works well, the diagonal of the confusion matrix should hold the highest values; in other words, the characters should have been classified as themselves most of the time.
For the combination 50 trees with depth 50 the confusion matrix in Fig. 11 looks very much like just described. The diagonal entries are of a dark shade of blue whereas most of the other cells are filled with whitish blue. Homer Simpson seems to have been classified especially well. Possible reasons for that are to be given shortly.
The combination 50×50 worked well, so now the confusion matrix for the combination 15×10 is shown in Fig. 12. The accuracy for this combination had just been over 50%. The actual values of the matrix can be found in Fig. 13.
The matrix shows a distinct diagonal, with some cells much

Figure 7: Marge Simpson

darker than the others. Homer Simpson's diagonal entry is again the darkest and the corresponding value in the matrix is 82%, i.e. of the Homer Simpson pictures in the test set, the random forest classified 82% of the pictures as Homer. Other characters who did well were Sideshow Bob who is easily distinguished by his wild reddish hair. The third best candidate is Ned Flanders, who is presumably recognizable


(a) Homer Simpson


(b) Sideshow Bob


(c) Ned Flanders

Figure 8: The characters who were the easiest to classify correctly


(a) Nelson Muntz


(b) Lenny Leonard


(c) Mayor Quimby

Figure 9: The characters who were the hardest to classify correctly

by his green sweater.

The character who did worst was Nelson Muntz, who was correctly classified only 2% of the time. As there are 50 pictures per character in the test set, only one picture of Nelson was classified as Nelson by the random forest. The other two worst characters are Lenny Leonard and Mayor Quimby, with 6 and 16% respectively.

Out of the 20 characters, 11 characters were classified as themselves more than 50% of the time, which is well satisfactory.

The nice thing about the confusion matrix, though, is that it is easy to see which characters the Simpsons have been confused with most of the time, i.e. what are the highest numbers in each row aside from the diagonal entry. It can be observed, for example, that Nelson Muntz who did badly on the diagonal has been classified as Homer Simpson 40% of the time. Another high value can be found in the entry corresponding to Lenny Leonard and Ned Flanders, stating that Lenny Leonard was classified as Ned Flanders 34% of the time.

As humans, the reason for this could easily be explained by the characters' appearances. Lenny Leonard is brown-haired and wears a green sweater, just like Ned Flanders. Nelson Muntz, though, has seemingly nothing in common with Homer Simpson. So why is he classified as Homer so often?

Again, an unscientific explanation would be that, since Homer is the main character but still a very ordinary person in the show, he was given the most ordinary Simpsons features. It is easy to see that his son Bart, for example, has the same face except for Homer's beard and Bart's hair. So one possible reason that all of the characters were confused with Homer so often might that he shares common face or clothes features with all of the other characters.

The mathematical reason can be found looking at the data structure of the training set, though. The training set consists of folders of pictures for each character, as was downloaded from Kaggle. But the folders are not equally big, in other words, in Homer's folder there are much more pictures than in Nelson's, around 2300 pictures to 220 to be exact. So the decision tree, if it does not have a big enough depth, learns much more about Homer's features and concentrates more on classifying Homer correctly than it cares about Nelson. Of course it would have been much more appropriate from a machine learning point of view to have equally sized folders. Another way would have been to put weights on the characters to make up for any difference in number of pictures in the training set.

This also explains why Ned Flanders is classified correctly most of the times: in the training set, his folder contains 1500 pictures.

To take a look at Marge Simpson: She has been classified correctly 68% of the time, so she is in the top seven. She was classified as Chief Wiggun 4% of the time, which is a surprisingly small value, given that they both show a lot of blue color.

Looking at Fig. 14, a very interesting thing to finish with is the heat map for the combination 15 trees with depth 5. The accuracy is not as good now: the diagonal does not seem as dark as for the other combination before. But it seems that the most prominent features of the characters are more important when there is less depth in the tree: Homer is still classified the best at 80%, possibly due to the reason just explained, but now Marge Simpson, Sideshow Bob and Chief Wiggun, the most easily recognizable characters, lead the way with 76, 59 and 42% correct classifications. This means that the decision trees, when constrained in their depth, look for the broader features to classify the characters, whereas if the tree is allowed to grow longer, other details become more important. This can also be seen in the corresponding table in Fig. 15.

Interestingly enough, the same effect does not appear when the size of the forest is reduced instead of the depth: the heat map becomes more diverse and it is harder to distinguish the diagonal. It is harder to classify the characters as themselves and even Homer is only recognized 42% of the time.

B. Summary

The random forest method yields convincing results if the number of trees in the forest and their depths is high enough. It is quite dependent on the training set, though, as can be seen looking at Homer Simpson: he does well because his folder in the training sets contains the most pictures. The characters with the highest accuracy are either main characters - meaning they have many pictures in the training set's folder - or have significant features such as Sideshow Bob. If you reduce the depth of the tree, the significant features are more important and suddenly characters like Marge and Chief Wigguns do well.

VII. Outlook

There are other tree methods that yield even better results than the random forests. There are the so-called *Extra trees* [2] which are a highly randomized version of the random forest. They differ from the random forest in two important facts: the splitting of the nodes is completely random and the whole training sample is used for creating the trees. In practice, the extra trees performed better than the random forest for the Simpson test data set. For a combination of enough trees with enough depth, they reached a 100% accuracy for the test set.

Appendix

The following pictures are best viewed on a monitor!

	1	5	10	15	20	50
1	0.07	0.07	0.08	0.06	0.07	0.07
5	0.17	0.25	0.26	0.27	0.27	0.27
10	0.25	0.43	0.5	0.53	0.54	0.58
15	0.33	0.55	0.63	0.66	0.72	0.76
20	0.31	0.56	0.66	0.74	0.76	0.83
50	0.36	0.61	0.68	0.75	0.79	0.85

Figure 10: Accuracy for combinations of number of trees and depth of the trees. The rows correspond to the depth, the columns to the number of trees. The accuracy in most of the time increasing in the direction of the depth of the trees and the number of trees.

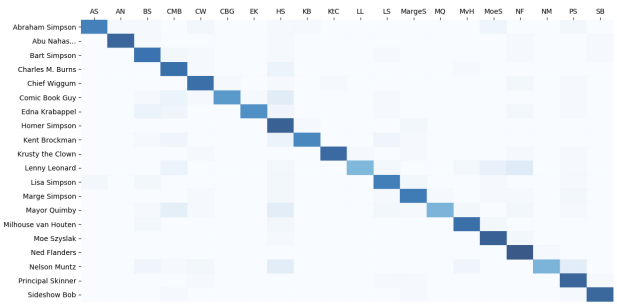

Figure 11: Confusion matrix heatmap for the combination 50 trees of depth 50. The diagonal is very distinct and shows that most of the characters are classified as themselves with a high percentage.

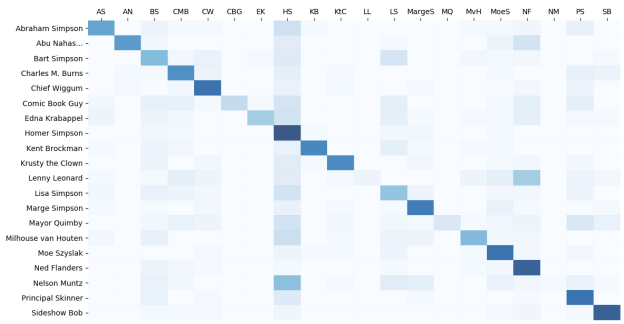

Figure 12: Confusion matrix heatmap for the combination 15 trees of depth 10. The diagonal is not as distinct as in the 50×50 case but it is still recognizable.

	AS	AN	BS	CMB	CW	CBG	EK	HS	KB	KtC	LL	LS	Marge	MQ	MvH	MoeS	NF	NM	PS	SB
AS	52	0	6	2	2	0	0	20	2	0	0	0	0	0	2	2	4	0	8	0
AN	0	56	4	0	0	0	0	12	0	0	0	2	0	0	0	6	20	0	0	0
BS	0	0	42	4	8	0	2	14	0	0	0	18	0	0	4	2	4	0	0	2
CMB	0	2	2	60	6	0	0	10	0	2	0	0	0	0	0	2	0	0	8	6
CW	0	2	0	4	72	0	0	8	0	2	0	0	2	0	0	0	4	0	6	0
CBG	4	0	8	8	2	24	2	18	0	0	0	10	0	0	0	2	10	0	10	0
EK	6	0	6	4	0	0	34	20	0	0	0	10	2	0	2	2	10	0	2	2
HS	0	0	4	2	0	0	0	82	2	0	0	4	4	0	0	2	0	0	0	0
KB	2	0	6	4	0	0	0	8	64	0	0	10	2	0	2	2	0	0	0	0
KtC	0	0	6	0	4	0	0	12	2	62	0	0	4	0	0	2	4	0	4	0
LL	2	0	2	10	6	0	0	12	0	2	6	0	0	0	6	10	34	0	6	4
LS	4	0	8	6	4	0	0	20	0	0	0	38	6	0	0	4	4	0	6	0
MargeS	2	0	2	0	4	0	0	8	0	2	0	2	68	0	0	8	2	0	0	2
MQ	0	0	4	8	6	0	0	20	0	4	0	4	2	16	2	4	6	0	16	8
MvH	4	0	8	0	0	0	0	22	0	2	0	6	6	0	42	4	4	0	0	0
MoeS	0	0	0	0	2	0	0	6	2	2	0	6	0	0	2	72	4	0	4	0
NF	0	0	6	4	2	0	0	2	0	0	0	2	0	0	2	4	77	0	0	0
NM	0	0	8	4	0	0	0	40	0	2	0	12	10	0	0	8	4	2	8	2
PS	0	0	8	0	2	0	0	14	0	0	0	0	2	0	0	0	2	0	72	0
SB	0	0	2	2	2	0	0	4	0	0	0	0	0	0	0	2	4	0	2	78

Figure 13: Confusion matrix for the combination 15 trees of depth 10.

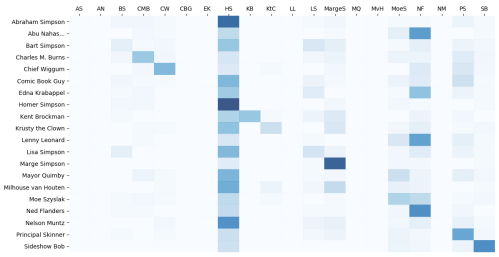

Figure 14: Confusion matrix heatmap for the combination 10 trees of depth 5. The diagonal is not easily distinguishable. Single cells on the diagonal show a deeper color and those cells correspond to characters with prominent features such as Marge.

	AS	AN	BS	CMB	CW	CBG	EK	HS	KB	KtC	LL	LS	Marge	MQ	MvH	MoeS	NF	NM	PS	SB
AS	0	0	4	4	2	0	0	72	0	0	0	2	4	0	0	2	2	0	6	2
AN	0	0	0	4	2	0	0	26	0	0	0	0	2	0	0	10	54	0	0	2
BS	0	0	10	2	4	0	0	36	0	0	0	16	10	0	0	2	10	0	6	4
CMB	0	0	4	35	4	0	0	16	0	0	0	4	6	0	0	6	6	0	12	4
CW	0	0	0	4	42	0	0	14	0	2	0	0	4	0	0	0	14	0	16	4
CBG	0	0	4	2	2	0	0	42	0	0	0	6	2	0	0	6	12	0	20	2
EK	0	0	2	4	0	0	0	34	0	0	0	12	0	0	0	2	38	0	6	2
HS	0	0	4	4	0	0	0	80	0	0	0	0	4	0	0	4	2	0	0	2
KB	0	0	2	0	0	0	0	30	36	2	0	6	14	0	0	0	6	0	4	0
KtC	0	0	0	2	2	0	0	38	0	22	0	0	16	0	0	4	10	0	2	4
LL	0	0	0	2	0	0	0	16	0	0	0	2	0	0	0	16	52	0	10	2
LS	0	0	10	0	0	0	0	42	0	0	0	18	6	0	0	4	12	0	6	2
MargeS	0	0	0	0	0	0	0	12	0	0	0	2	76	0	0	4	4	0	0	2
MQ	0	0	0	6	2	0	0	44	0	0	0	2	4	0	0	22	6	0	10	4
MvH	0	0	0	0	2	0	0	46	0	6	0	4	24	0	0	8	6	0	0	2
MoeS	0	0	2	4	2	0	0	26	0	2	0	0	2	0	0	30	26	0	2	4
NF	0	0	2	2	0	0	0	24	0	0	0	2	2	0	0	4	59	0	4	0
NM	0	0	0	0	4	0	0	57	0	0	0	6	8	0	0	4	10	0	8	2
PS	0	0	2	0	2	0	0	22	0	0	0	4	6	0	0	6	4	0	50	4
SB	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	6	4	0	6	59

Figure 15: Confusion matrix for the combination 10 trees of depth 5. The diagonal is not easily distinguishable. Single cells on the diagonal show a deeper color and those cells correspond to characters with prominent features such as Marge.

REFERENCES

[1] http://www.otnira.com/2013/03/25/c4-5/, 18/07/2017
[2] http://www.montefiore.ulg.ac.be/ẽrnst/uploads/news/id63/extremely-randomized-trees.pdf, 18/07/2017
[3] Histograms of Oriented Gradients for Human Detection, by N. Dalal, B. Triggs, in CVPR '05
[4] Random Decision Forests, by Tim K. Ho, 1995 in Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC
[5] http://cloudmark.github.io/images/kotlin/ID3.png, 18/07/2017
[6] https://upload.wikimedia.org/wikipedia/commons/c/c9/Binary_entropy_plot.png, 18/07/2017