VRIJE UNIVERSITEIT BRUSSEL

ANIMAL CLASSIFICATION

Machine Learning

NAME: ROBIN DE HAES
STUDENT NUMBER: 0547560

1. Problem analysis

The problem consists of assigning images to 1 of 12 animal types: chicken, elephant, fox, German Shepherd, golden retriever, horse, jaguar, lion, owl, parrot, swan, tiger. Intuitively, it is to be expected that some types will be easier to distinguish from each other than others. A lion and a tiger will probably have more features in common than a lion and an owl. Training images are labeled by their containing folder's name. Since we provide labeled training data to the learner to classify samples into one of multiple classes, the problem is *a supervised multiclass classification problem*.

The evaluation metric is also fixed to be the Cross-Entropy Loss: $L=-\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_n^c log(p_n^c)$. In short, this scoring metric takes the uncertainty of model predictions into account based on how much it varies from the actual class label. This Log Loss should be minimized. Just obtaining a high accuracy alone is not enough. The confidence for predictions is taken into account.

2. Simple preprocessing & Feature extraction

Folder names are used as string representations of the class labels. To facilitate learning these string labels will be further encoded as integers.

Pickle files containing features extracted from the images have been provided. After some testing with the feature descriptors, *SIFT feature vectors* seem to give good results overall and SIFT features will be used throughout this report. Feature values are in the same range, so they will not require normalization. SIFT feature vectors are actually already normalized to unit length to obtain contrast invariance, i.e., to reduce the effects of illumination changes in the images.

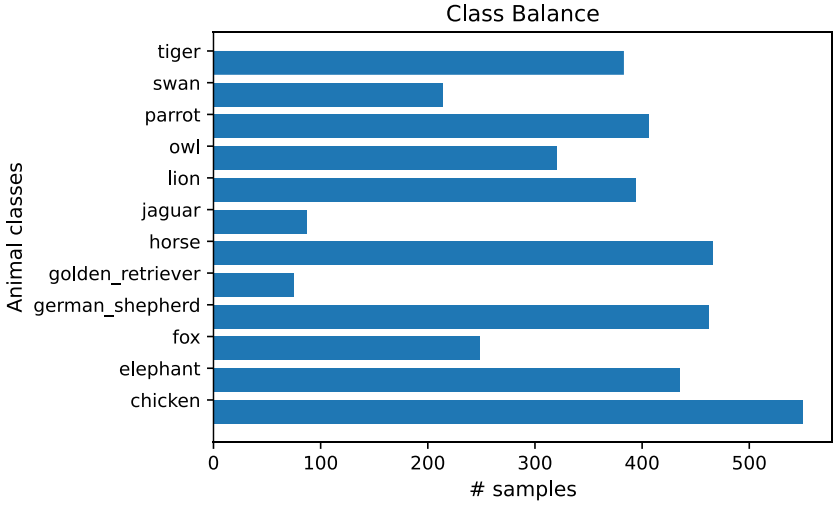
3. Data analysis

The results presented in this section are based on visual inspection of the images themselves and the SIFT feature vectors obtained with a codebook size of 500. A similar analysis occurred for a codebook size of 750 and 1000, which gave similar results and therefore will not be discussed here.

Images

The samples for each class are images that were scraped from the web. They are quite diverse in scale, viewpoint, etc. Some images are drawings, while others are actual pictures. Some images have a lot of background noise, e.g., humans or other animals that are in the picture. Images sometimes contain different species of the same animal type. They can also contain the same type of animal multiple times.

If we count the number of images per class folder, we can get a view on class imbalance.



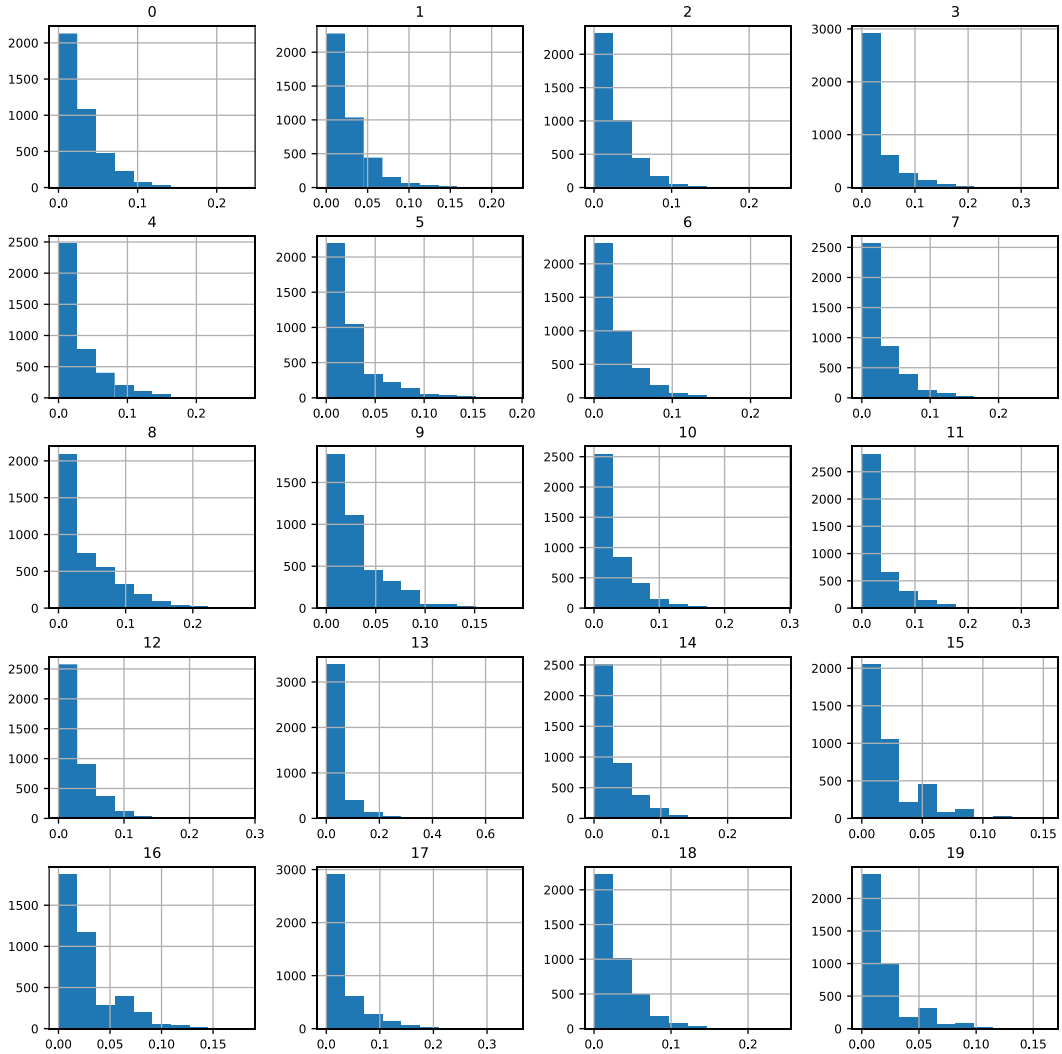
Some classes are clearly represented more than others, but it is known that the distribution of classes in the test set will be similar to the distribution of classes in this training data. Therefore, compensations for class imbalance will have a smaller effect, since 99% correctness in the imbalanced data should be representative for 99% correctness in the imbalanced population. However, very low confidence scores when classifying minority classes should be avoided since this increases the Cross-Entropy Loss. Minority classes should thus still be taken into account when training models.

SIFT: Missing values & No variance

First we checked if there are any missing values. As expected, there were no missing values. We also checked if there are features without any variance that are thus not important when making predictions. However, there also did not seem to be any such features.

SIFT: Univariate analysis

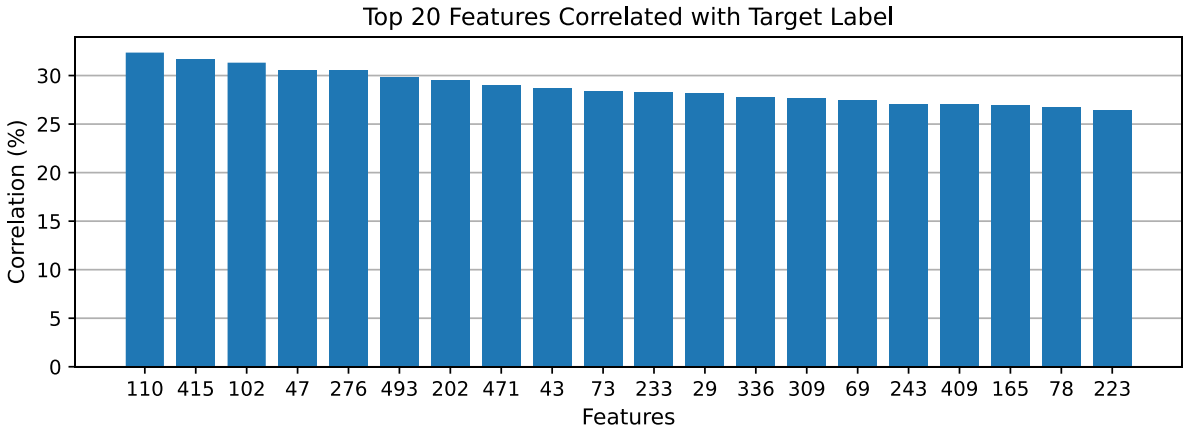
A significant imbalance in the target classes has already been observed. We can also still look further into the feature distributions. The distributions of all 500 features were plotted, but we will only present plots of the first 20 features for visual clarity. These 20 distributions show the general trend for most features. There were 4042 samples used when plotting these histograms.



All feature values are between 0 and 1. The distributions are generally right-skewed, but the tail length and/or fatness differs between features. These observations are further confirmed by skewness values that can vary from 1.3 to 7.4, indicating varying tail lengths, and kurtosis values that can vary from 1.7 to 74.7, indicating varying tail fatness. It should be kept in mind that the presence of certain “outliers” could in this case just follow from class imbalance.

SIFT: Bivariate analysis

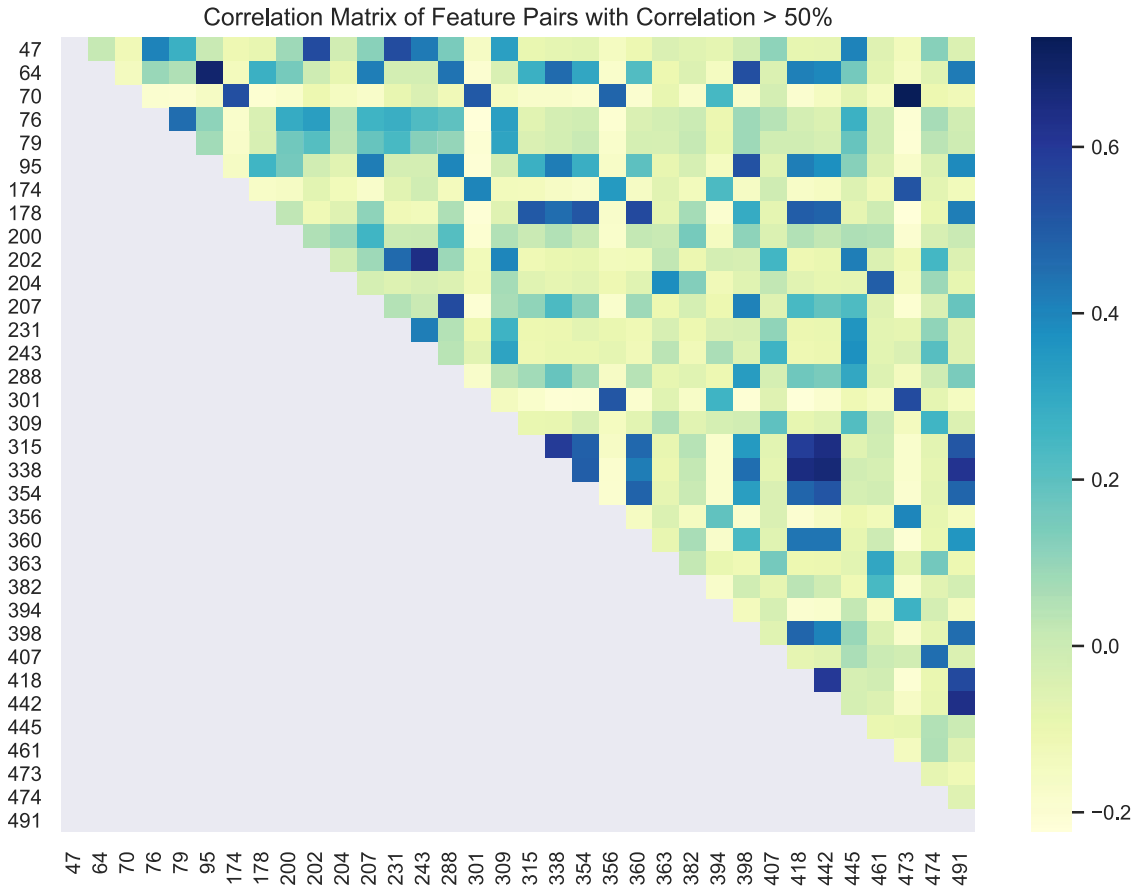
We explored the *correlation* every feature had *to the target label*. Below we present the 20 features with the highest correlation to the target. After these top 20 features, correlation drops below 25%.



There seem to be multiple features with a correlation to the target label of around 30%. These are important features, but only focusing on them might ignore features that are correlated with a minority that is less present in the data.

Next we looked at feature *collinearity*. If a pair of features are highly correlated, either positively or negatively, one of the two features could be removed without losing too much information.

A heatmap of all features was made, but high dimensionality hindered visually identifying correlated features this way. Nevertheless, it showed there is some correlation present since the scale goes from around -0.2 to 0.65. To produce a more useable heatmap, we selected feature pairs that have at least a correlation of 0.5 (or 50%). There were 34 such features. The heatmap's bottom half has been removed, since it does not provide additional information due to the matrix symmetry.



It is immediately clear from this heatmap that rather highly correlated features exist. However, a lot of features remain that are not very (linearly) correlated. The reduction in number of features by only dropping one of both features from these correlated pairs might therefore only lead to a small benefit. Further feature engineering, selection and/or dimensionality reduction methods will be needed.

4. Training/Validation/Test split

A split between training data and test data has already been made. There are 4042 training samples and 4035 test samples. The training data will be further split up in training data and validation data. Some class imbalance exists in our training dataset, which makes using a *stratified* approach a good idea. A fraction of each class is forced to be present in the training and validation set.

The data was randomly scraped from the web and we do not know if there are many relevant subgroups in our dataset. Some samples, however, differ in scale, viewpoint, etc. so *random shuffling* before splitting the data is a good idea. We chose to split our training data using the well-known split of *80% training data and 20% validation data*. Considering the amount of data we have and the fact that we have relatively sparse feature vectors (which is common for bag-of-words model), this split seems suitable here. The effects of preprocessing, feature selection and further finetuning will only occur on the 80% training data to *avoid* any biases due to *information leakage*.

5. Performance metric & Cross-validation

Our performance metric is the same as the evaluation metric, namely *multiclass Log Loss*. A negative version is used to allow a maximization approach instead of minimization, but it is the same formula multiplied by -1. In results, however, the absolute value of this Log Loss will be presented.

The Log Loss metric needs probability estimates so some models (like SVMs) have to perform cross-validation to obtain these. This slows down training, but our Kaggle score will be helped a lot by being able to compare models not only on output but also on their probabilistic outcomes. Computational overhead was mitigated by letting some models, like SVM, train overnight.

In general we will use *cross-validation with 5 stratified folds* when finetuning and testing the chosen models. We will use a stratified approach to ensure each class is somewhat represented in each fold despite our imbalanced data. These cross-validations will mostly happen in *grid searches*. Most grid searches will start with a general range for hyperparameters after which they might be repeated for more specific, interesting ranges or with certain hyperparameters added, removed or fixed.

6. Logistic Regression

It seems like a good idea to start off with a simple, linear model before trying more complex models. It might already perform well and we can test out the efficacy of certain steps at a faster rate.

There are 4 hyperparameters that are commonly tuned for logistic regression models:

- *C*, which is the regularization parameter
- *Penalty*, which specifies the norm used in penalization. Both the L1 norm and the L2 norm were tested when possible, but it should be noted that not all solvers support the L1 norm.
- *Solver*, which specifies the algorithm to use in the optimization.
- *Max iterations*, specifying the iteration count a solver has available for convergence to an optimal solution.

Using a codebook size of 500, grid searches were performed with 5-fold cross-validation and the following hyperparameter ranges:

- C was tested in a range from 10^{-5} to 10^{4} in logarithmic steps.
- Penalty = L1, combined with the following solvers: Newton-CG, L-BFGS, SAGA
- Penalty = L2, combined with the following solvers: Newton-CG, L-BFGS, SAGA, SAG, LibLinear

Cross-validation with 5 folds gives as best average cross-entropy score 1.591 with a standard deviation (SD) of 0.043 for the combination C=10, Penalty=L1, Solver=L-BFGS and 100 max iterations. This grid search was also repeated with algorithmic compensation for class imbalance, which improved the log loss score for the same hyperparameter combination to *1.570 (SD: 0.034).* This score difference is inside the standard deviation, but the balanced model will nevertheless be used as baseline.

Our baseline model for Logistic Regression has a mean log loss of 1.570 with hyperparameters:

- *C = 10*
- *Penalty = L1*
- *Solver = L-BFGS*
- *Max iterations = 100*
- *Algorithmic compensation for imbalance*

Feature selection

Word-of-bag models have high dimensionality and sparse feature matrices. Since our available training data is limited, reducing the number of features and dimensionality is a good idea. The results described here were obtained using a codebook size of 500, but were repeated with sizes 750 and 1000. Standardization was also tried out in combination with feature selection but this did not lead to further increases.

Filter

A filter technique selects a subset of features based on information measuring the usefulness of the set. This can happen quite fast, but perhaps allows a less fine-grained selection than other techniques.

A simple application uses the *correlation and collinearity*, which is available from our data analysis. First, a minimum threshold with regard to class label correlation was set to remove less correlated features. Next, we removed one of both features occurring in pairs of collinear features. Class correlation thresholds between 0.1 and 0.3 and feature collinearity between 0.4 to 0.7 were tried out in steps of 0.05.

The best results were obtained with a threshold of 0.15 and a feature collinearity of 0.6. These thresholds reduced the number of features to only *135 features.* The same baseline hyperparameter combination performed the best, but a log loss score of only *1.640 (SD: 0.016)* was obtained. This is not an improvement, but nevertheless remarkable considering we only used around one fourth of the original features.

Wrapper

A wrapper technique adds or removes features while using a provided machine learning algorithm's performance as evaluation criterion to decide if adding (forward selection) or removing (backward elimination) that feature improves or deteriorates performance. A combination of forward and backward elimination is also possible as well as recursive feature elimination which trains a model to iteratively assign weights to features in a set and prune less important features to obtain a smaller set until a specified number of features is left. It is computationally heavier than filter techniques.

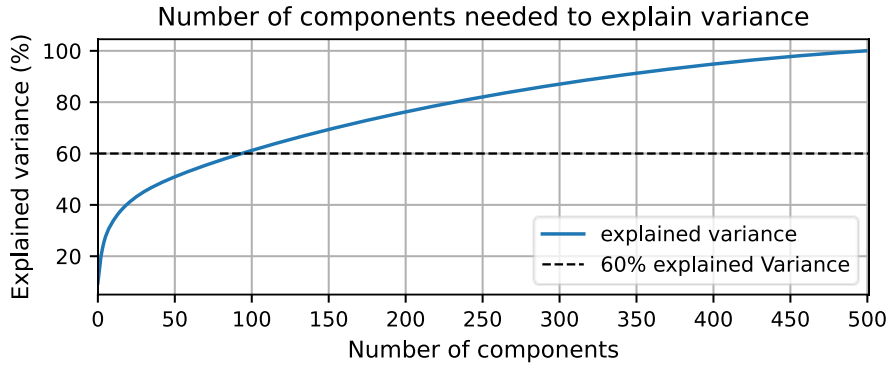
Backward elimination was tried out from 500 to 300 kept features in steps of 25 features. The best result was found for *375 features, C=10, penalty=L1, solver = L-BFGS and algorithmic imbalance compensation*. The log loss score was *1.531 (SD: 0.041)*.

RFE was tried out with a linear kernel SVM, a Random Forest Classifier and logistic regression as feature importance determinators. Using logistic regression for both importance determination and as trained model, seemed to overfit the data which is why we focused on the other two. Number of features between 200 and 400 were tested in steps of 50, with smaller steps of 5-25 between 250 and 350. The best result was found using an *SVM (linear kernel, C=1, imbalance compensation) to specify feature importance and keeping the 300 most important features*. *Logistic Regression with C=10, penalty = L1, solver = L-BFGS without algorithmically compensating for imbalance* performed the best. A log loss of *1.503 (SD: 0.027)* was obtained, which is a clear improvement over our previous baseline.

Dimensionality reduction

Instead of selecting specific features, we could also transform the feature space into a lower dimension. We mostly used PCA for dimensionality reduction. PCA basically projects the data features into a given number of new features that are orthogonal and can still explain a certain amount of variance seen in the data. We will use a grid search on the percentage of explained variance to obtain a suitable dimensionality reduction. Since the new features are linearly independent, checking for correlation between the features after performing PCA seems superfluous.

The original grid search was repeated, but now included PCA component tuning to explain 50 to 90 percent of the variance (in steps of 10 percent). The best results were found for *an explained variance of 60% and logistic regression with C=10, penalty=L1, solver=L-BFGS and no imbalance compensation*. As shown on the graph below, an explained variance of 60% means around 100 PCA components were used. This lead to a log loss score of *1.448 (SD: 0.024)* which is the best result so far. From the feature selection and dimensionality reduction it is clear that some features are not that informative and can be removed for better results.



Over- and undersampling

Another way to deal with class imbalance is over- and undersampling. Oversampling can be done by just randomly selecting samples of minority classes and refeeding them, but this makes these specific samples more influential in the obtained score. Another way is using SMOTE, which creates new samples by selecting samples from a minority class and combining their features to obtain a new sample that resembles their characteristics. We used *SMOTE* to oversample some of the minority classes and later combined it with undersampling of the majority classes.

Oversampling of classes to have equal samples for every class led to the LibLinear solver, with penalty L2, a C of 100 and 300 maximum iterations performing the best with a mean log loss score of *1.278*. However, *a large standard deviation in performance of 0.410* makes this model too unreliable.

We also tried *combining undersampling* of the majority classes *and oversampling* of the minority classes. First we tried giving every class the same amount of samples, i.e. the same samples as the median class. This gave a log loss score of 1.91, which is an overall bad performance. Some other combinations, preserving the original class sizes more, were tried and performed better but *did not lead to an improvement* on the previous baseline performance either.

Codebook size

The approach of hyperparameter tuning and feature selection was repeated for codebook sizes of 750 and 1000. Using a codebook size of 750 gave better results than 500 and led to our final selected model with PCA and an explained variance of 0.6.

Final model

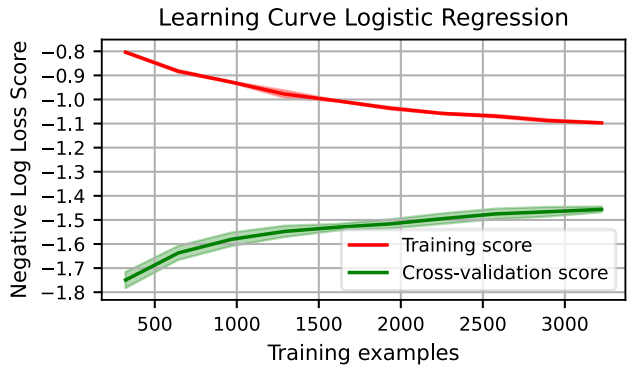
The final model that was selected is Logistic Regression with:

- C=10, Solver = L-BFGS, Penalty = L1, Max-iter = 100
- Codebook size of 750, reduced to 168 features through PCA with an explained variance of 0.6.

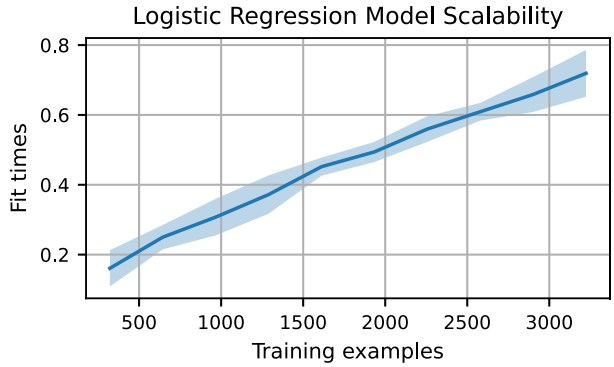
The obtained cross-validation score was 1.453 (SD: 0.024). The obtained Kaggle score was 1.484.

Model analysis

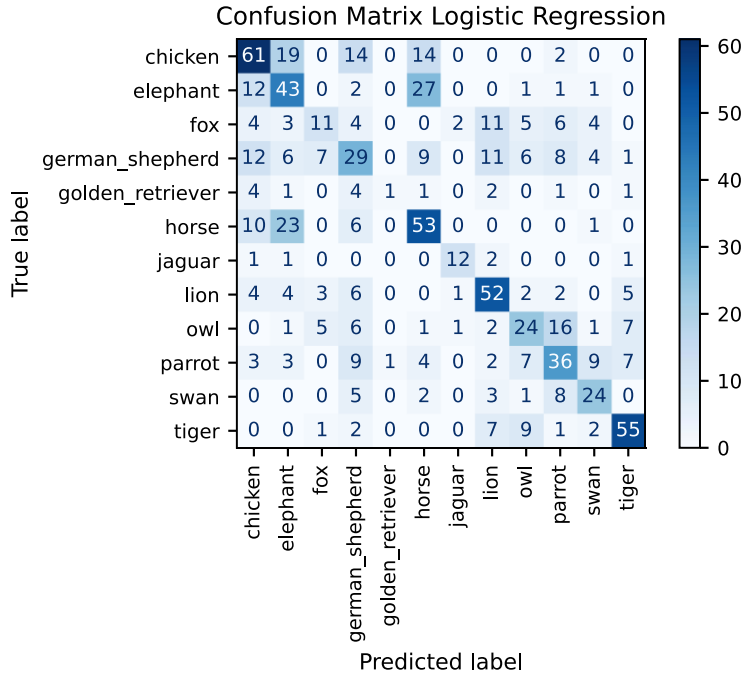
It is important to note that the negative log loss score is discussed and plotted in these learning curves, since we found it intuitively easier to interpret these negatively scored curves.



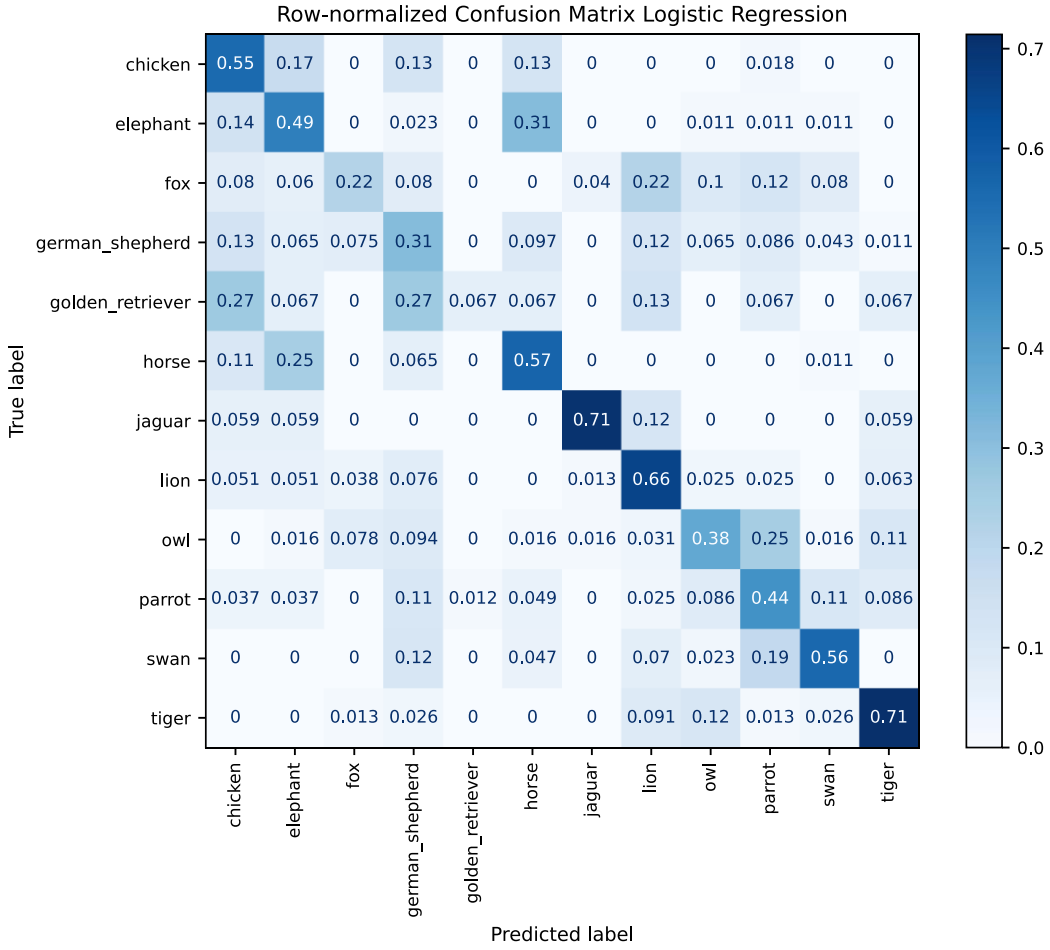
This model seems to suffer from high variance, but also a bit of bias. Despite tuning, it still does not represent the underlying model that well leading to a rather low cross-validation score. We will probably benefit from having more training data, since the cross-validation score is still rising, however it is seemingly already slowly stagnating.



This graph shows how the fitting time of the model scales with regard to the number of training examples used. This seems to be more or less linear, so it can be trained with more data in that regard.



We also show a row-normalized confusion matrix, since it shows the percentage of class samples (mis)classified as a specific class more clearly.



The consequences of class imbalance are clearly visible. Golden retriever, which has the lowest prevalence, is almost never classified correctly. It is often classified as the majority classes chicken or German Shepherd. Fox also has a low correct prediction rate, this class is mostly confused with lion. Interestingly, jaguar which is also a minority class has a very high correct classification rate. It is even higher than the majority classes. Those images probably contain highly distinguishable patterns.

As a result of this confusion matrix, SMOTE was reinvestigated to only double the lowest minority class without over- or undersampling the other classes. This improves the classification of Golden Retriever, but this does not improve the overall log loss score. Nevertheless, it might be interesting to have a model that can classify the minority classes well and then combine this with a more general model into an ensemble model. This, however, has not been tried out.

7. Support vector machines (SVM)

SVMs represent each sample as a point in the state space and then try to create a separating hyperplane between the different classes of points in that space.

SVMs normally perform binary classification, but they are expandable to multi-class via a one-vs-one or a one-vs-rest approach. A one-vs-one will train a separate classifier for each different pair of classes, while a one-vs-rest approach will just train a classifier for one class. Clearly a one-vs-one approach trains more classifiers and will thus be computationally more expensive. It didn't seem to have a huge effect in our case, so a one-vs-rest approach was used. Standardization of the feature vector also didn't have a beneficial effect so it was not added to our pipeline.

The hyperparameters to tune depend on the kernel used. The kernel decides how the separation between the classes is shaped, i.e., linear, polynomial, radial basis functions, sigmoid.

In general, there are 3 common hyperparameters in SVMs for tuning:

- *C*, the regularization parameter which influences the margin of class separation
- *Degree*, the degree of the polynomial that is used. This is only relevant for *polynomial* kernels.
- *Gamma*, which influences the narrowness of the function and thus how far the influence of a single sample point should reach. This is *not* relevant for a *linear* kernel.

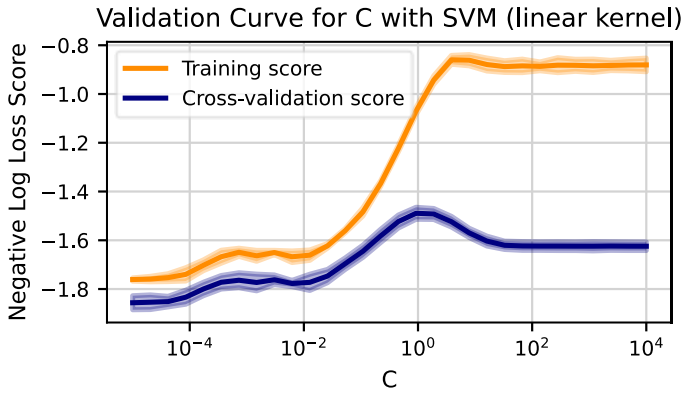
It should be noted that SVMs don't compute probabilities of outcomes by default. They need cross-validation to approximate probabilities which is computationally very expensive. We chose to do this to be able to use log loss as a scoring metric which will be the most reliable towards our Kaggle score.

The following hyperparameters grid searches occurred with 5-fold cross-validation:

Linear kernel

- C was tested in a range from 10^{-5} to 10^{4} in logarithmic steps.

The best score was found for C's starting from 1 and higher. The performance for C=1 and C=10 seems similar. If we take a look at the validation curve below, we clearly see our cross-validation score goes down around C=1 though. We performed another grid search with C going from 0.6 to 6 (in steps of 0.1 until 1 is reached and then in steps of 1), but the highest cross-validation score was still obtained with *C=1*. A log loss score of *1.465 (SD: 0.025)* was obtained.



Polynomial kernel

- C and gamma were tested in a range from 0.001 to 1000 in logarithmic steps. Gamma also had 'scale' as an option.
- Degree was tested in a range from 2 to 8 in steps of 2.

The best parameters were *degree of 2, gamma of 1 and C of 10*. Cross-validation with 5 folds gave an average log loss score of *1.474 (SD: 0.042)*.

Sigmoid & Radial basis function

- C and gamma were tested in a range from 0.001 to 1000 in logarithmic steps. Gamma also had 'scale' as an option.

The best score was found for RBF with C=10 and gamma='scale'. A new grid search with C going from 2 to 14 in steps of 1 gave *C=12 and gamma='scale'* as best parameters. A log loss score of *1.431 (SD: 0.033)* was obtained.

This hyperparameter tuning was repeated with algorithmic imbalance compensation, but this did not improve results.

Codebook size

The influence of codebook size for the linear and RBF kernels was tested, together with small grid searches around the best hyperparameter values found earlier, i.e., their best value and one logarithmic step above and below that value.

Kernel	Codebook size	Mean Log Loss	Standard deviation	Best parameters
RBF	250	1.463	0.042	C=1, gamma = 1
	500	1.431	0.033	C=12, gamma = 'scale'
	750	1.418	0.032	C=10, gamma= 'scale'
	1000	1.414	0.027	C=10, gamma = 'scale'
	1500	1.411	0.028	C=10, gamma = 'scale'
Linear	250	1.507	0.031	C=1
	500	1.465	0.025	C=1
	750	1.462	0.025	C=1
	1000	1.450	0.022	C=1
	1500	1.444	0.032	C=1

It seems for both kernels the *best performance* is found for *higher codebook sizes*. It should be mentioned that encoding time also increases a lot with higher codebook sizes, so going much higher than 750 should only be done if there is a clear benefit. Although quite a bit of score variance exists for the RBF kernel, the best hyperparameters seem to consistently be around C=10 and gamma='scale'. For linear kernels, the best C value seems to be C=1 overall.

Feature selection & Dimensionality reduction

Less extensive experimentation was performed for SVMs than for Logistic Regression (due to computational and time restrictions). As a *baseline* we used the *log loss score for an SVM with RBF kernel, C=10, gamma='scale' and a codebook size of 750*. This gave a score of *1.418 (SD: 0.032)*.

First, *filtering* techniques were applied on a codebook size of 750 and an SVM with the baseline hyperparameter combination. Chi² and mutual information, which is a measurement for dependence between features and thus information redundancy, were used. Between 300 and 500 features were selected in steps of 50, which led to a score of *1.401 (SD: 0.030) for 400 features* selected with the mutual information approach.

Next, *PCA* was tested for an SVM with the baseline hyperparameter combination. This was tested for explained variances between 0.5 and 0.8 (in steps of 0.1) and codebook sizes of 500, 750 and 1000. The best results for each codebook size are presented in the table below, together with the best performing explained variance.

Codebook size	**500**	**750**	**1000**
Explained variance	0.7	0.6	0.6
Number of components	176	196	233
Log loss score	1.401 (SD: 0.025)	1.382 (SD: 0.020)	1.360 (SD: 0.023)

Finally, standardization was tried out in combination with this dimensionality reduction but this did not lead to further increases.

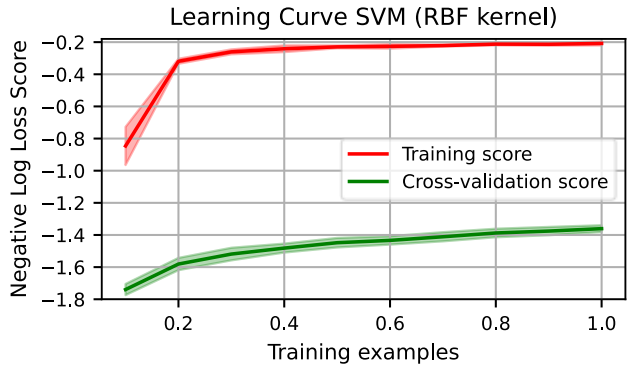
Final model

The final model that was selected is SVM with:

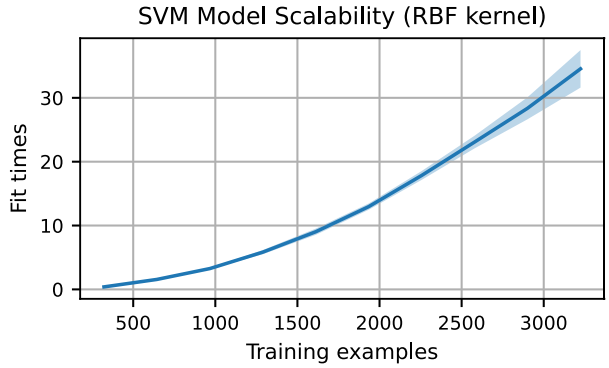
- RBF kernel, C = 10, scaled gamma
- No algorithmic compensation for imbalance
- Codebook size of 1000, reduced to 233 features via PCA with an explained variance of 0.6

The obtained cross-validation score was 1.360 (SD: 0.023). The obtained Kaggle score was 1.357.

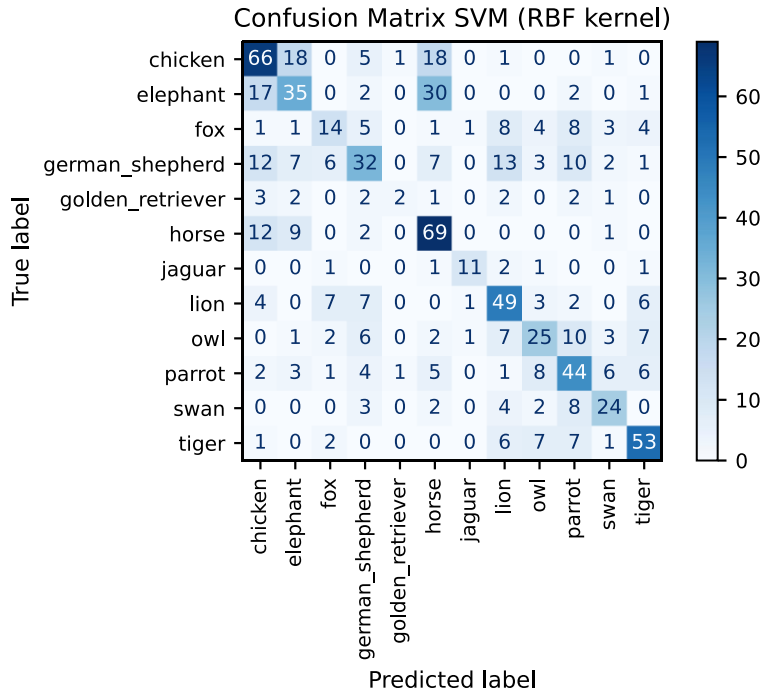
Model analysis



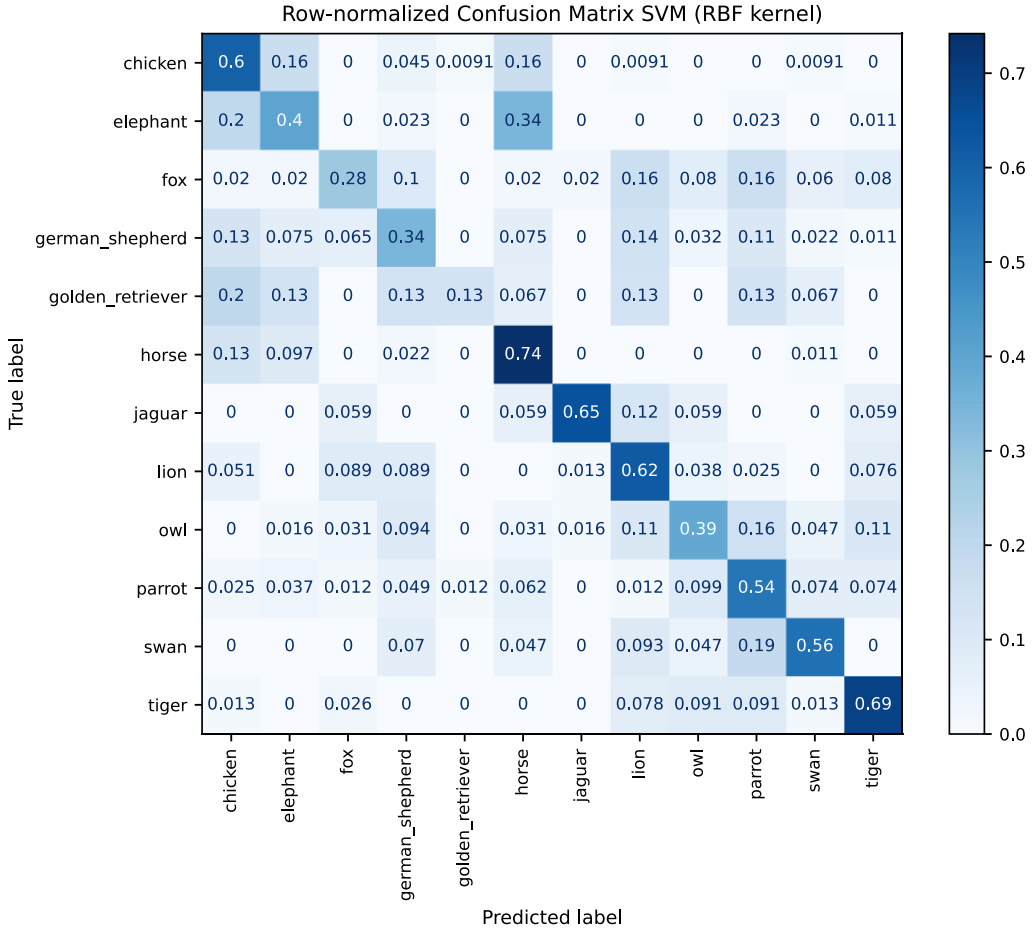
This model seems to be overfitted and suffer from high variance. Its negative log loss score is consistently high, while it's cross-validation score only goes up slowly. At this point the model is not generalizable enough. Having additional training data might help our model to better detect the actual underlying data model instead of noise, although the cross-validation score only rises rather slowly.



The fitting time in this graph is calculated without cross-validation for probability estimates. The fitting time seems to scale exponentially with the number of samples, which means its scalability is limited in this regard since having a lot more data will cause a large increase in time it takes to train the model.



Again, we also show a row-normalized confusion matrix to show the percentage of class samples (mis)classified as a specific class more clearly.



The consequences of class imbalance are visible, although its predictive power for minority classes is better than the logistic regression model. Golden retriever is still often misclassified as a chicken, but in 13% of the cases it is correctly classified. However, this is actually only one correct classification more than in the logistic regression model. Overall, it seems some correct class predictions went down while others went up. Therefore, it seemed interesting to use an ensemble *Voting Classifier* with a *soft voting* approach to combine the predictions of *both Logistic Regression and SVM* to see if this would increase the log loss score. This led to a score of *1.381 (SD: 0.028)* which is not an improvement.

8. Other classifiers

Gaussian Naive Bayes is a simple non-linear model that sometimes performs well. We tuned the hyperparameter variance smoothing, which smoothens the curve so more samples that are further from the distribution mean are accounted for. A grid search occurred for variance smoothing between 10^{-9} and 1 (logarithmic steps), combined with codebook sizes of 250 to 1000 (steps of 250) and PCA with explained variances from 60% to 100% (steps of 10%). The results seemed to prefer a variance smoothing of 0.1, a codebook size of 1000 and 60% explained variance. This led to a mean log loss score of *1.701 (SD: 0.040)*. This model was not investigated further.

The K-Nearest Neighbors Classifier has been tried out with some hyperparameter tuning for the number of neighbors, weights, leaf size and the power parameter used, combined with some dimensionality reduction through PCA. However, the classifier seemed to have low predictive power for this problem. This can be explained by the fact that we're using sparse high-dimensional feature vectors. Since we have so many dimensions, it is hard for similar points to actually be closer to each other than other points along all these axes.

Random Forest Classifier has a lot of possible hyperparameters to tune in relatively large ranges. Only a randomized search with cross-validation was performed to sample this search space. Max tree depth (10 to 100), min samples in leaf (1 to 10), min samples split (2 to 20), class weight (balanced, balanced subsample, none), number of estimators (200 to 2000) were searched with 100 iterations of random combinations. This only led to a mean log loss of 1.791. More fine-grained search and feature selection might have a huge influence though, but this was not performed due to time restrictions.

9. Closing remarks

Initially, we spent a bit too much time on extensive finetuning of model hyperparameters instead of focusing more on preprocessing, feature selection, etc. Spending more time on the latter would probably have led to better scores. Both logistic regression and SVM seemed to actually end up with hyperparameters close to the default ones and finetuning steps like dimensionality reduction seemed to have a larger influence.

Exploring steps that occur before hyperparameter finetuning more thoroughly could have had a larger impact on our Kaggle scores. We're thinking of steps like combining the highest feature correlations for each separate target label value instead of just using the highest correlation to the target label (in feature selection), combining multiple features (in feature engineering), using black-and-white images to mitigate color differences between subspecies of the same animal (in preprocessing) or just using an ensemble of classifiers perhaps combined with SMOTE for some. In conclusion, this report only shows interesting starting points but further exploration is definitely still possible.

10. Bonus Question

What would you change to your model if the distribution of the test set was uniform over the 12 classes? Would you use the same loss function, performance metric, train/validation/test splits etc. as you did before?

If the distribution of the test set was uniform over the 12 classes, the imbalance in our training data will have a more deleterious effect on the Cross-Entropy Loss. The less common classes of our training data will be misclassified often in the test data and lead to high penalties. Macro-averaged performance metrics that give equal weight to all classes, independent of their size, could be used. Train/validation/test splits should still be stratified, but oversampling and undersampling approaches will become more important.