CS5014 - Practical 2 - 160020220

Introduction

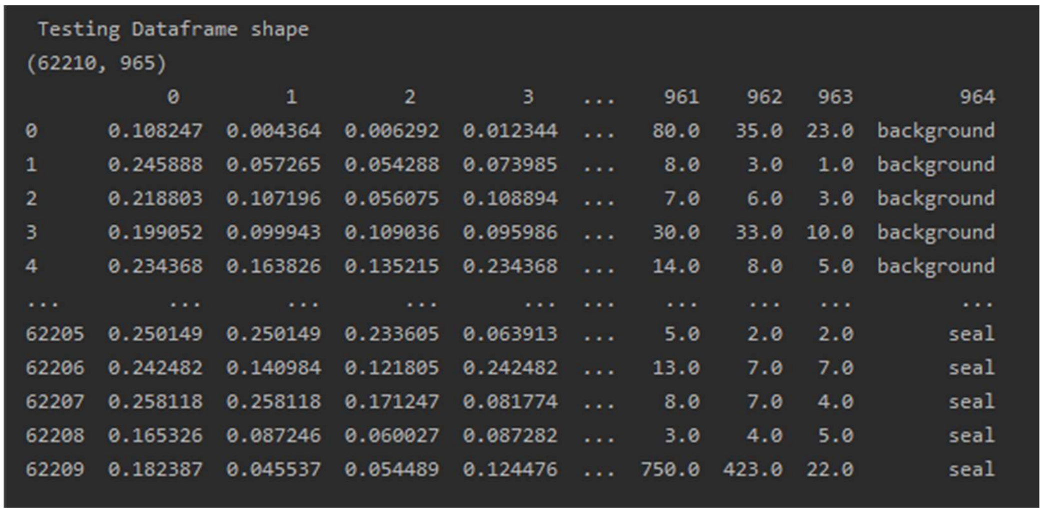
The aim of this practical is to implement, test and discuss classification models on real world data. Classification is an area of supervised machine learning that tries to predict which class or category some entity belongs to, based on its features. The focus in this practical is on the two main types of classifications, binary and multiclass. The difference between the two is whether the output belongs to multiple classes of is binary, hence the name.

In this practical, all specification and extensions have been attempted. This report will describe how the data has been loaded and pre-processed after the accompaniment of its analysis and visualisation. Following this, design and implementation decisions will be discussed with, in particular, the description and comparison between the different models submitted. Finally, testing of the models and overall program will be proposed.

The extension attempt here is the implementation of multiple binary classification models and their comparison.

Data analysis and visualisation

This section will only take the most important features that are kept after pre-processing into account.



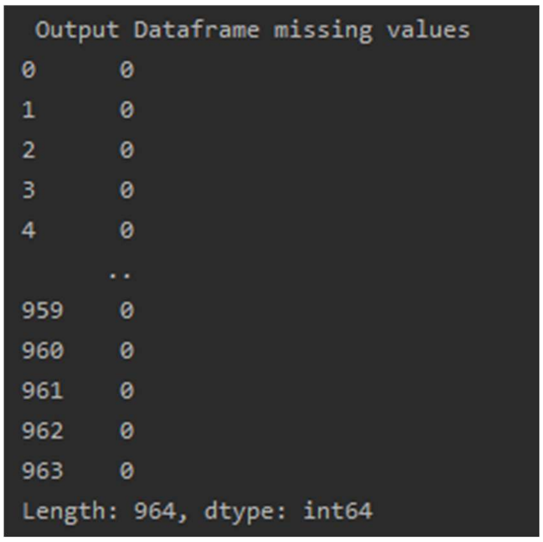
```
 Testing Dataframe shape
(62210, 965)
              0         1         2         3  ...    961    962   963         964
0      0.108247  0.004364  0.006292  0.012344  ...   80.0   35.0  23.0  background
1      0.245888  0.057265  0.054288  0.073985  ...    8.0    3.0   1.0  background
2      0.218803  0.107196  0.056075  0.108894  ...    7.0    6.0   3.0  background
3      0.199052  0.099943  0.109036  0.095986  ...   30.0   33.0  10.0  background
4      0.234368  0.163826  0.135215  0.234368  ...   14.0    8.0   5.0  background
...         ...       ...       ...       ...  ...    ...    ...   ...         ...
62205  0.250149  0.250149  0.233605  0.063913  ...    5.0    2.0   2.0        seal
62206  0.242482  0.140984  0.121805  0.242482  ...   13.0    7.0   7.0        seal
62207  0.258118  0.258118  0.171247  0.081774  ...    8.0    7.0   4.0        seal
62208  0.165326  0.087246  0.060027  0.087282  ...    3.0    4.0   5.0        seal
62209  0.182387  0.045537  0.054489  0.124476  ...  750.0  423.0  22.0        seal
```

Design and implementation

Loading the data and pre-processing

Loading the data is done by using 2 pandas data frames to store the input set and the output set as 2 csv files are provided in the specifications. The handy thing here is that pandas provides a reading function to load the csv files directly in data frames. One thing to note here is that the header is set to None as the data provided does not have any. This allows the data frames to set a unique id to each column of data which represent the features.

Now, we must ensure that the file doesn't have missing values. Applying the following to the input data frame ensures the above: lambda x: sum(x.isnull()), axis=0)



```
 Output Dataframe missing values
0      0
1      0
2      0
3      0
4      0
      ..
959    0
960    0
961    0
962    0
963    0
Length: 964, dtype: int64
```

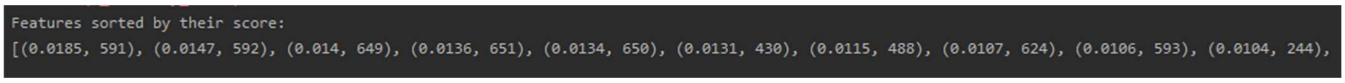
All values are set to 0 and that indicates that the file in clear from any missing values.

The output file contains the output column, its content is tricky to handle as it is a string and our models would prefer numbers for dinner. For this reason, after loading the data, pre-processing of such data must take place. A solution to this problem is Dummy encoding, or one hot encoding, which transforms categorical variables into a series of binary columns. However, the 2 newly produced dummy variables are completely dependent on each other (Multicollinearity). We can naturally drop one of the dummy variables to avoid this trap whilst not losing any valuable information. This is done by simply setting the drop first to true.

Choosing suitable features

In this section, two choices were available. Either I would choose features depending on the data provided OR I would implement a piece of code to choose the most suitable feature out of any data file. I chose to go with the second option as it would more scalability and robustness. Indeed, this method allows my submission to be reused with other input/output data files. Even if the features chosen can be (not necessarily) less relevant, the scalability and robustness trade-off is worth it in my opinion. Of course, This choice is made as the precision of the models are very good and close to expected results.

Choosing suitable features is done using a random forest classifier. It allows a correlation between features and actual output to produce and important index. Here is a sorted list of the most impactful features:

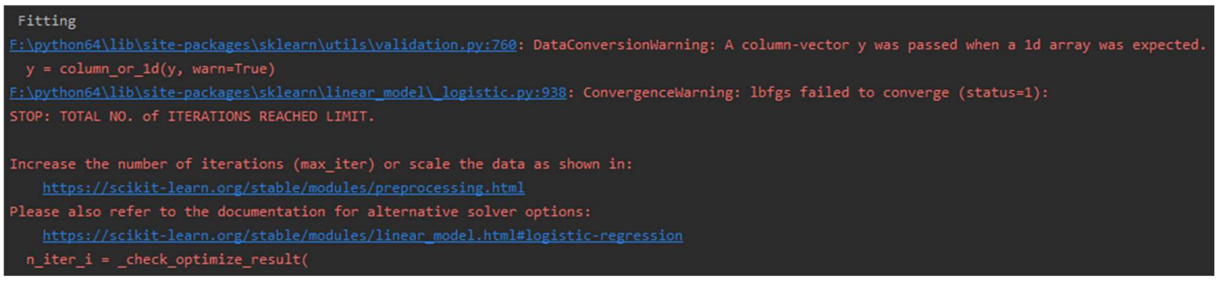


```
Features sorted by their score:
[(0.0185, 591), (0.0147, 592), (0.014, 649), (0.0136, 651), (0.0134, 650), (0.0131, 430), (0.0115, 488), (0.0107, 624), (0.0106, 593), (0.0104, 244),
```

Now, instead of choosing the 10 most important features, I noticed that the importance index would plummet after the 0.01 value. Thus, SelectFromModel allows here the reprocessing of our data while keeping only the features that have and importance index greater than 0.01. We end up keeping the 9 most important features instead of using the 964 provided in the csv input file.

Furthermore, pre-processing of data is not quite yet finished. The input set must be split to spot and prevent issues such as overfitting and underfitting. Overfitting is a situation when a model shows almost perfect accuracy when handling training data. This situation happens when the model has a complex set of rules. When a model is overfitting, it can be inaccurate when handling new data. Underfitting is when a model doesn't fit the training data due to sets of rules that are too simple. You can't rely on an underfitting model to make an accurate prediction. To train and test the model, 2 (or 3) sets are needed. Here, I only split the inputs into a training and testing set as I will use the Sklearn cross validation method thus not requiring a validation set. The train test split method is used here to have an 80% training set and a 20%testing set.

Putting a threshold to process only the most important values also allows to not cap the maximum of iterations:



```
 Fitting
F:\python64\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
  y = column_or_1d(y, warn=True)
F:\python64\lib\site-packages\sklearn\linear_model\_logistic.py:938: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Binary classification model

Logistic regression model

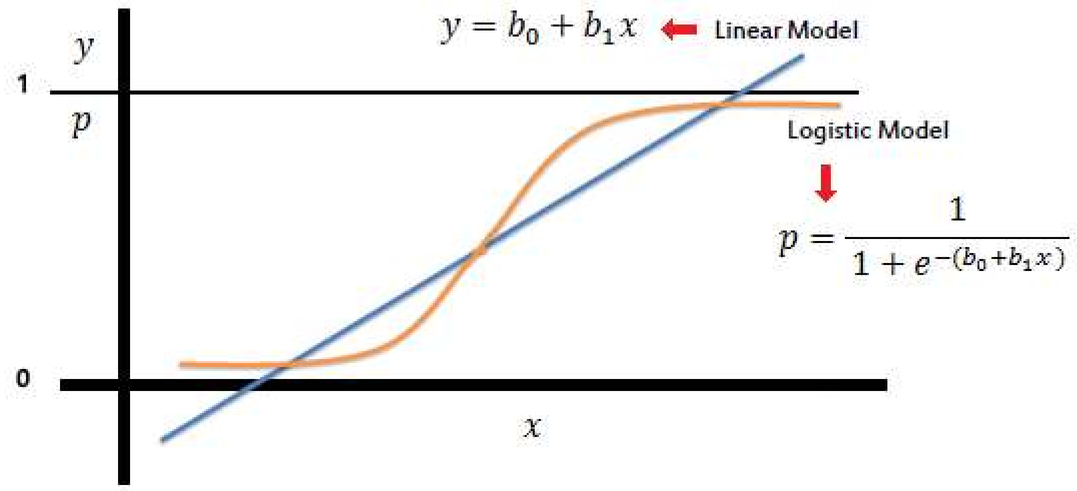
Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems thus being relevant in this practical. The logistic function is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1 defined by the following formulae:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}},$$

Ref: https://en.wikipedia.org/wiki/Logistic_function

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values to predict an output value (y). In other words, we use logistic regression to see how independent variables can influence a main result variable or how those variables can be related. This can allow some kind of forecasting that can be used in various fields.A key difference from linear regression is that the output value being modelled is a binary value (0 or 1) rather than a numeric value.

“a logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the “odds” of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.”



Ref = https://www.saedsayad.com/logistic_regression.htm

The implementation of this model in this practical has been done using sklearn.

Decision tree model

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making and can be used in a classification problem such as this practical. “Growing a tree involves deciding on which features to choose and what conditions to use for splitting, along with knowing when to stop” [ref: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052]

Decision tree builds a classification model in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The result is a tree with decision nodes and leaf nodes. A decision node can have multiple branches whereas a leaf node represents a binary decision in this case. The topmost decision node in a tree which corresponds to the best predictor called root node.

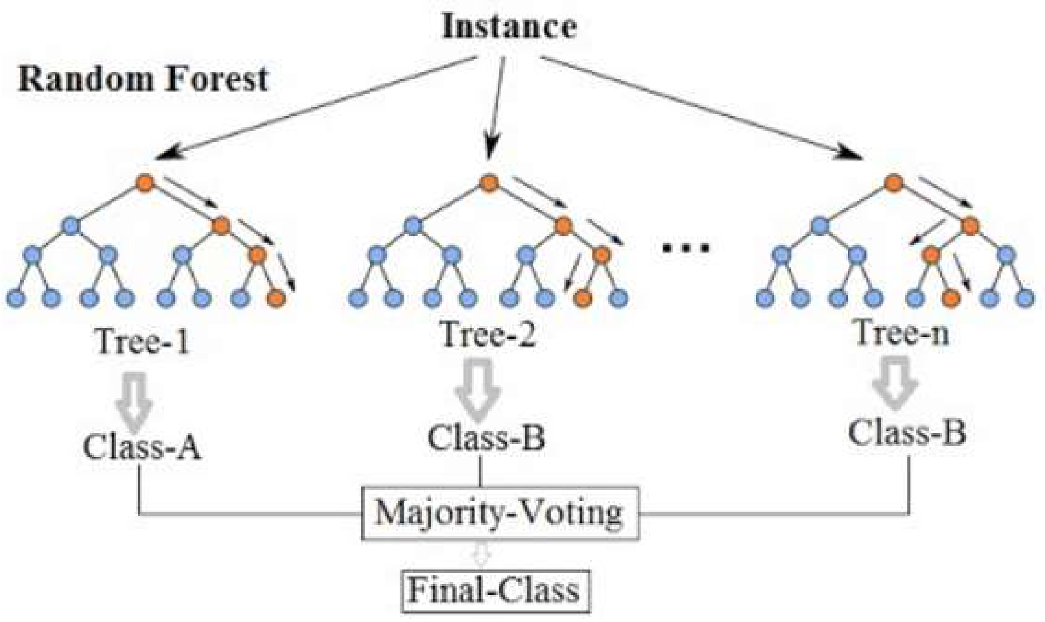
The core algorithm for building decision trees employs a top-down, greedy search through the space of possible branches with no backtracking. This algorithm is called ID3 by J. R. Quinlan.

Ref: Utgoff, P.E. Incremental Induction of Decision Trees. Machine Learning 4, 161–186 (1989).

It uses Entropy and Information Gain to construct a decision tree. Indeed, the input data is partitioned into subsets that contain instances with similar values. Thus, entropy is used to calculate the homogeneity of a sample. Then, the tree is built using the information gain which is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain

Random forest model

Following the above information, a random forest model is an ensemble tree-based learning algorithm. The Random Forest Classifier is a set of decision trees from randomly selected subset of training set. It aggregates the votes from different decision trees to decide the final class of the test object.

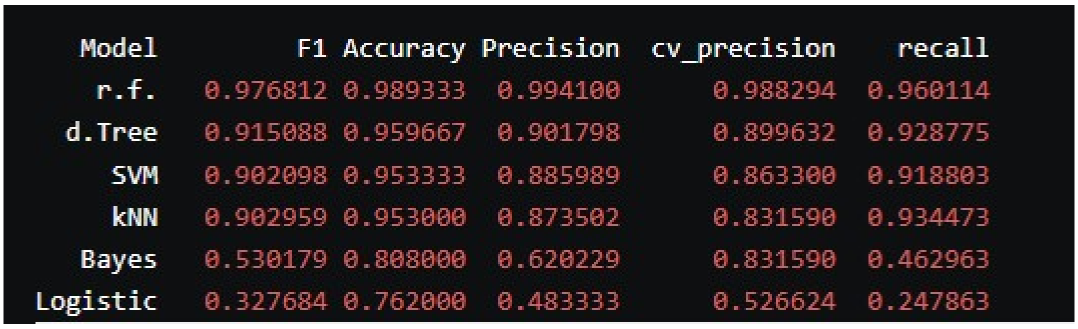


Structure of Random Forest Classification

Ref: https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0

The random forest model is one of the most accurate learning algorithms available and runs efficiently on large databases. It generates an internal unbiased estimate of the generalization error as the forest building progresses. However, Random forest models are likely to overfit for some datasets with noisy classification tasks. Thus, A cross validation must be done to make sure that the above doesn't happen. In theory, I would choose the random forest model fir my final prediction.

In addition to all this, I have found an online test for a similar task online where the author compares different classification models:



```
   Model          F1  Accuracy  Precision  cv_precision    recall
    r.f.    0.976812  0.989333   0.994100      0.988294  0.960114
  d.Tree    0.915088  0.959667   0.901798      0.899632  0.928775
     SVM    0.902098  0.953333   0.885989      0.863300  0.918803
     kNN    0.902959  0.953000   0.873502      0.831590  0.934473
   Bayes    0.530179  0.808000   0.620229      0.831590  0.462963
Logistic    0.327684  0.762000   0.483333      0.526624  0.247863
```

Ref: https://lukesingham.com/whos-going-to-leave-next/

This list translates the clear superiority of the random forest model based on accuracy and precision mainly.

Multi-class classification

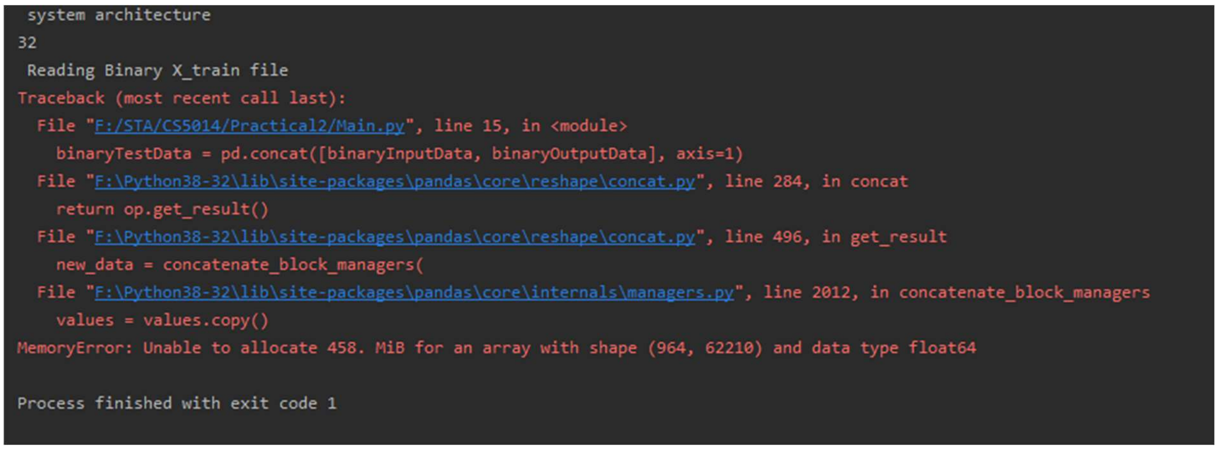
The main difference between a binary classification and a multiclass classification is that the outputs are not binary but belong to a set of classes. The theory behind classification remains the same however.

Final Prediction

Finally, what is left to be done is to do a prediction of the set given in the specifications. Choosing the random forest model as it is the most performant here, a csv file is from the prediction data frame. The data frame is also processed to have a string label instead of an integer value for formatting purposes.

Problems encountered

During this practical, I encountered multiple problems. The biggest problem was that I was using python 32 bits and had a memory limitation which cause a memory error every time I wanted to handle the data. Spotting this issue and fixing it took the majority of the time I had to do the practical as it required a reinstallation of python which was not instinctive to do at the start.



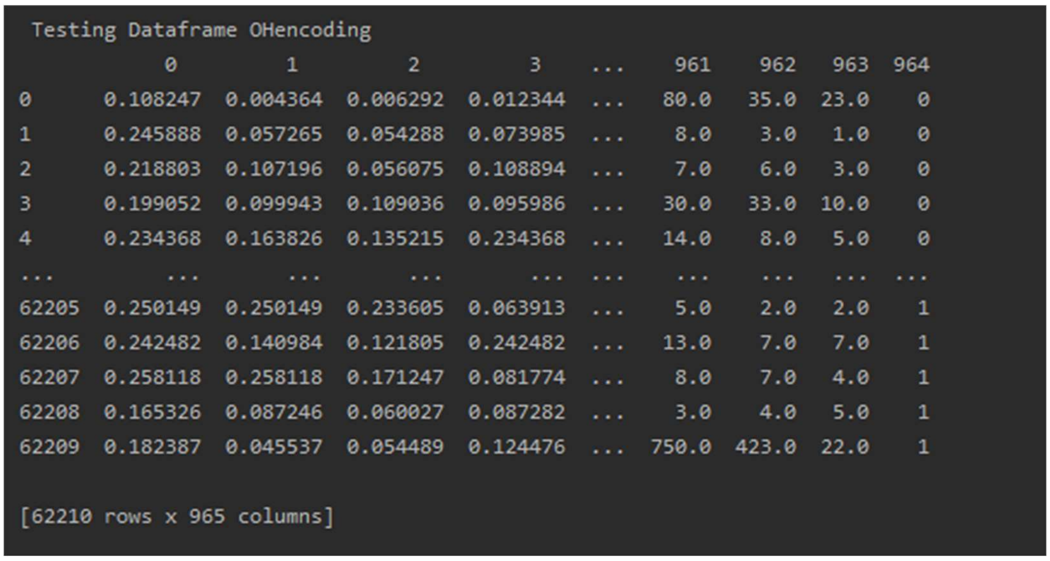
```
 system architecture
32
 Reading Binary X_train file
Traceback (most recent call last):
  File "F:/STA/CS5014/Practical2/Main.py", line 15, in <module>
    binaryTestData = pd.concat([binaryInputData, binaryOutputData], axis=1)
  File "F:\Python38-32\lib\site-packages\pandas\core\reshape\concat.py", line 284, in concat
    return op.get_result()
  File "F:\Python38-32\lib\site-packages\pandas\core\reshape\concat.py", line 496, in get_result
    new_data = concatenate_block_managers(
  File "F:\Python38-32\lib\site-packages\pandas\core\internals\managers.py", line 2012, in concatenate_block_managers
    values = values.copy()
MemoryError: Unable to allocate 458. MiB for an array with shape (964, 62210) and data type float64

Process finished with exit code 1
```

I encountered other hardware issues as I got a disk error when important sklearn. As I could not use the lab machines and my personal computer, setting up the project was a real struggle. I can honestly say that the most difficult part of the practical was to fix hardware bugs.

Testing

Binary classification

The first thing we need to check is that the content of the data frame is correct. We see that the shape and format of the data frame is logical. These outputs need to be encoded to a machine learning understandable format. To do this, one hot encoding is used. Here, we can see that the output value has been encoded correctly.

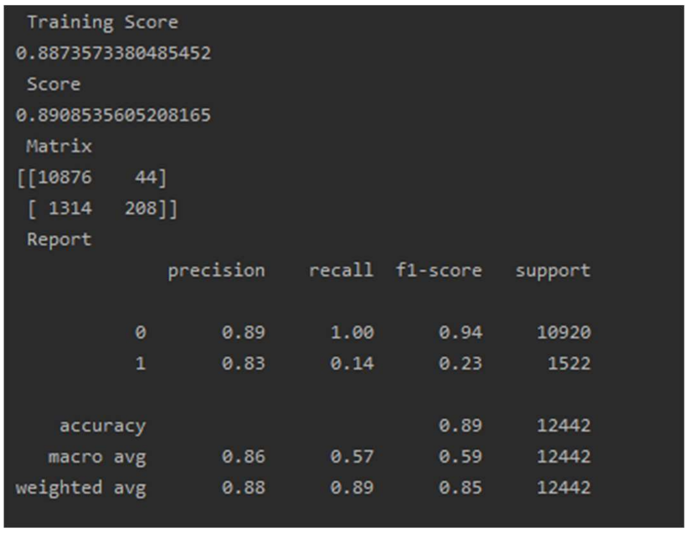


```
 Testing Dataframe OHencoding
              0         1         2         3  ...    961    962   963  964
0      0.108247  0.004364  0.006292  0.012344  ...   80.0   35.0  23.0    0
1      0.245888  0.057265  0.054288  0.073985  ...    8.0    3.0   1.0    0
2      0.218803  0.107196  0.056075  0.108894  ...    7.0    6.0   3.0    0
3      0.199052  0.099943  0.109036  0.095986  ...   30.0   33.0  10.0    0
4      0.234368  0.163826  0.135215  0.234368  ...   14.0    8.0   5.0    0
...         ...       ...       ...       ...  ...    ...    ...   ...  ...
62205  0.250149  0.250149  0.233605  0.063913  ...    5.0    2.0   2.0    1
62206  0.242482  0.140984  0.121805  0.242482  ...   13.0    7.0   7.0    1
62207  0.258118  0.258118  0.171247  0.081774  ...    8.0    7.0   4.0    1
62208  0.165326  0.087246  0.060027  0.087282  ...    3.0    4.0   5.0    1
62209  0.182387  0.045537  0.054489  0.124476  ...  750.0  423.0  22.0    1

[62210 rows x 965 columns]
```

Now let's compare the performances of the models implemented.

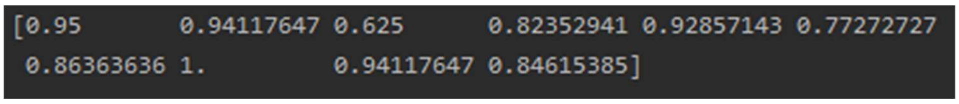
The main way to compare the models is to check not only the score of said models, but also the different averages of precision, recall, f1-score and support.

Logical regression



```
 Training Score
0.8873573380485452
 Score
0.8908535605208165
 Matrix
[[10876    44]
 [ 1314   208]]
 Report
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     10920
           1       0.83      0.14      0.23      1522

    accuracy                           0.89     12442
   macro avg       0.86      0.57      0.59     12442
weighted avg       0.88      0.89      0.85     12442
```

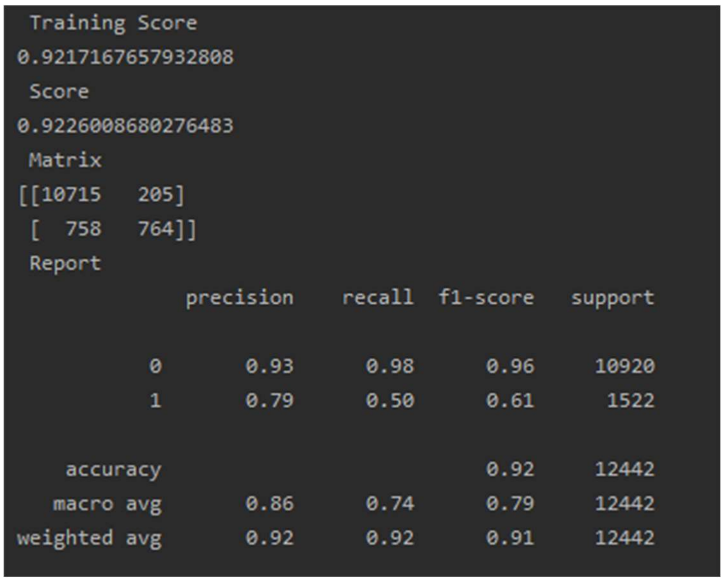
Cross val



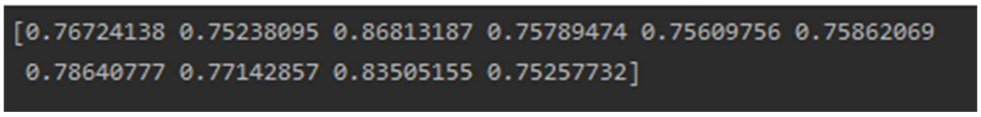
```
[0.95       0.94117647 0.625      0.82352941 0.92857143 0.77272727
 0.86363636 1.         0.94117647 0.84615385]
```

Decision tree

Metrics

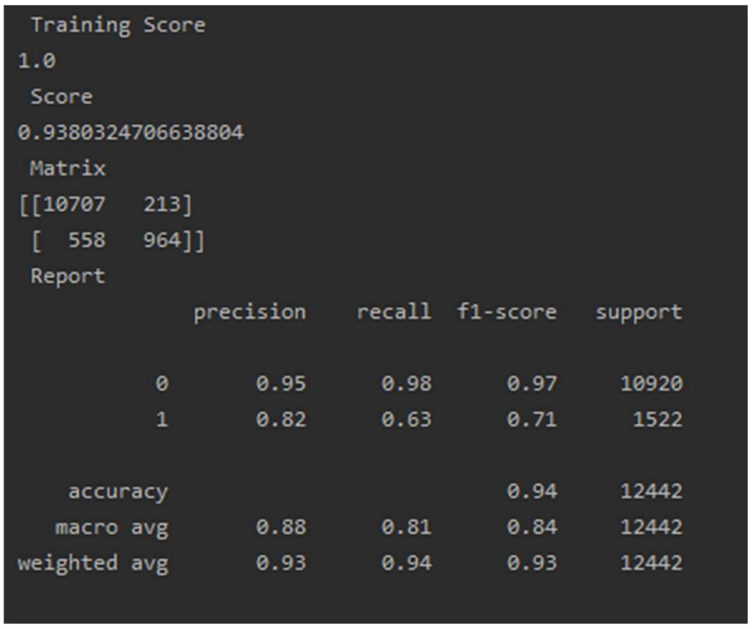

```
 Training Score
0.9217167657932808
 Score
0.9226008680276483
 Matrix
[[10715   205]
 [  758   764]]
 Report
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     10920
           1       0.79      0.50      0.61      1522

    accuracy                           0.92     12442
   macro avg       0.86      0.74      0.79     12442
weighted avg       0.92      0.92      0.91     12442
```

Cross val

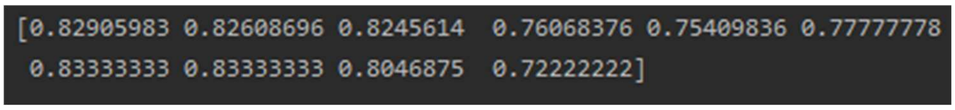

```
[0.76724138 0.75238095 0.86813187 0.75789474 0.75609756 0.75862069
 0.78640777 0.77142857 0.83505155 0.75257732]
```

Random forest

Metrics

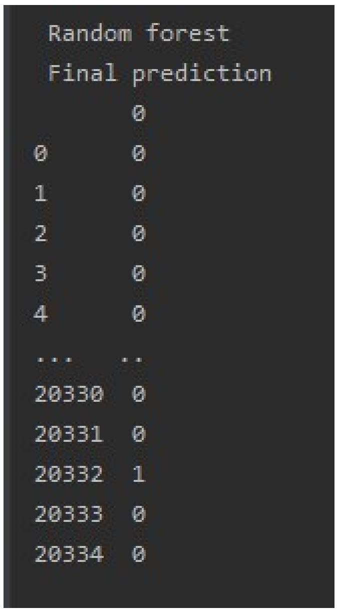

```
 Training Score
1.0
 Score
0.9380324706638804
 Matrix
[[10707   213]
 [  558   964]]
 Report
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     10920
           1       0.82      0.63      0.71      1522

    accuracy                           0.94     12442
   macro avg       0.88      0.81      0.84     12442
weighted avg       0.93      0.94      0.93     12442
```

Cross val

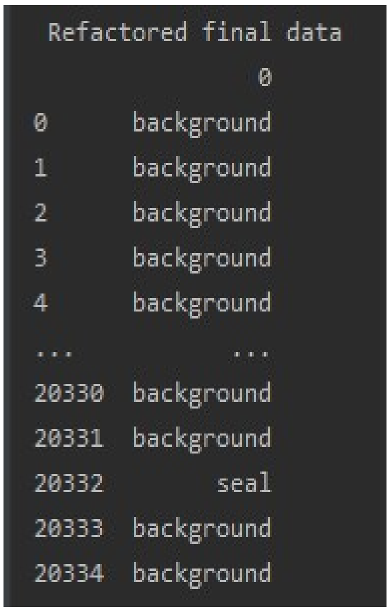

```
[0.82905983 0.82608696 0.8245614  0.76068376 0.75409836 0.77777778
 0.83333333 0.83333333 0.8046875  0.72222222]
```

As we can see, the random forest model behaves as expected and outperforms the other models.

Final prediction output



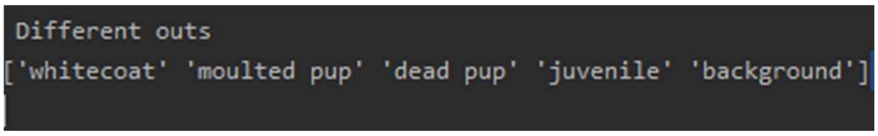
```
  Random forest
  Final prediction
          0
 0        0
 1        0
 2        0
 3        0
 4        0
 ...     ..
 20330    0
 20331    0
 20332    1
 20333    0
 20334    0
```



```
  Refactored final data
                0
0      background
1      background
2      background
3      background
4      background
...           ...
20330  background
20331  background
20332        seal
20333  background
20334  background
```

MultiClass classification

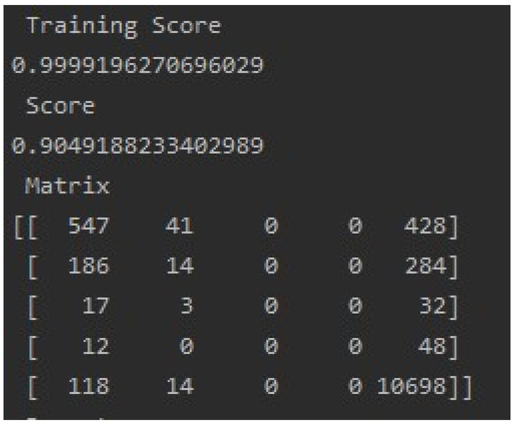
The different possible outputs are as follow.



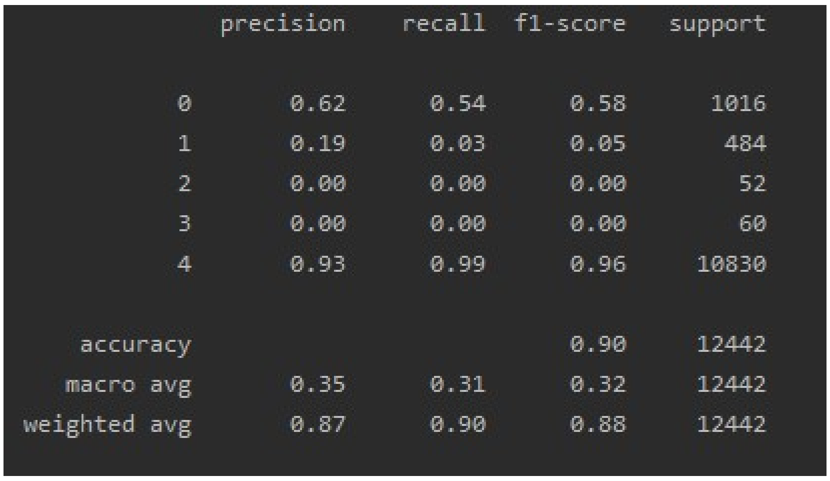
```
 Different outs
['whitecoat' 'moulted pup' 'dead pup' 'juvenile' 'background']
```

Using encoding, the outputs are categorized and parsed to integers whibetween 0 and 4. These values are used for the random forest training

Random Forest



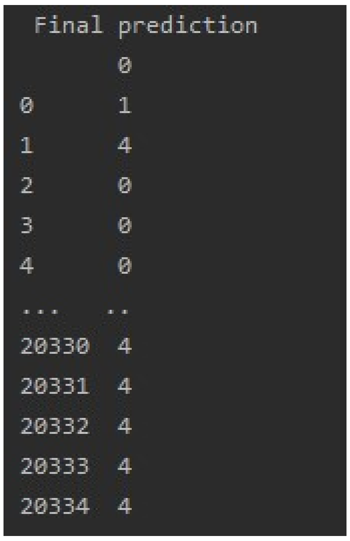
```
 Training Score
0.9999196270696029
 Score
0.9049188233402989
 Matrix
[[  547    41     0     0   428]
 [  186    14     0     0   284]
 [   17     3     0     0    32]
 [   12     0     0     0    48]
 [  118    14     0     0 10698]]
```



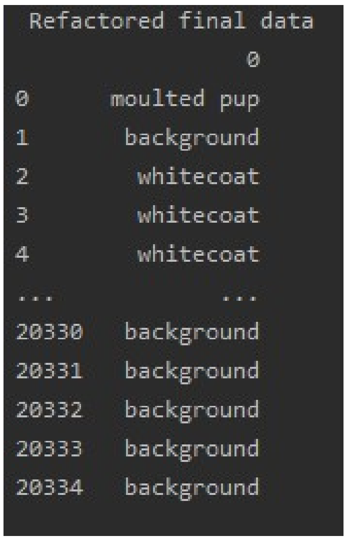
```
              precision    recall  f1-score   support

           0       0.62      0.54      0.58      1016
           1       0.19      0.03      0.05       484
           2       0.00      0.00      0.00        52
           3       0.00      0.00      0.00        60
           4       0.93      0.99      0.96     10830

    accuracy                           0.90     12442
   macro avg       0.35      0.31      0.32     12442
weighted avg       0.87      0.90      0.88     12442
```

The random forest model is also behaving as expected in multi class classification and performs very well.

Final prediction output



```
 Final prediction
       0
0      1
1      4
2      0
3      0
4      0
...   ..
20330  4
20331  4
20332  4
20333  4
20334  4
```



```
 Refactored final data
                0
0      moulted pup
1       background
2        whitecoat
3        whitecoat
4        whitecoat
...            ...
20330   background
20331   background
20332   background
20333   background
20334   background
```

As we can see again. The parsing is correct

Conclusion

Practical conclusion

Overall, I am very happy with my submission relatively to the struggle I had with the submission. The practical was very fun to do. The data training files had to be removed for submission.

(The files for testing are the Y_test.csv)

How did Covid-19 impact my work

Covid-19 had a huge impact on my submission. The main one being a late submission due to being denied access to my personal computer and forced to work in Cyber-cafés and brother's work laptop. This cause a lot of bugs detailed above. And above all else, it diminished the quality of my report.

References

https://en.wikipedia.org/wiki/Logistic_function https://en.wikipedia.org/wiki/Logistic_regression

https://lukesingham.com/whos-going-to-leave-next/

https://stackabuse.com/classification-in-python-with-scikit-learn-and-pandas/

https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_forest/

https://www.bitdegree.org/learn/train-test-split#overfitting-and-underfitting

https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Utgoff, P.E. Incremental Induction of Decision Trees. Machine Learning 4, 161–186 (1989).

https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0

https://chrisalbon.com/machine_learning/preprocessing_structured_data/one-hot_encode_features_with_multiple_labels/