CS5014 P2

160004864

5 May 2020

1 Introduction

The aim of this practical is to give us experience with working with a real-life experimental dataset that has not been analysed before. The dataset consists of features extracted from aerial images obtained during seasonal surveys of islands in the North Sea, and we are required to build and train a classification model to predict what type of seal pup is contained within these images. In this project, we will be performing two types of classification, namely binary classification and multi-class classification.

2 Data Loading

The first step to completing the task at hand is to load the data we will be working with. The dataset is split into two directories, `binary` and `multi` containing the data for our binary and multi-class classification task, respectively.

Each of these directories contain three files:

- X_train.csv: This file contains a set of comma-separated values where each row represents an image, and the columns represent the features extracted from the image.
- Y_train.csv: Each row of this file contains the corresponding class ID for each sample row from `X_train.csv`.
- X_test.csv: This file contains data in the same format as `X_train.csv` and it serves as the test test. This data will be used to produce the `Y_train.csv` file, which will contain the predictions our model makes, as required by the specification.

The process of loading the data is carried out by two methods, `load_binary_data` and `load_multi_data` with the first loading the dataset used for binary classification and the latter loading the dataset used for multi-class classification.

We specifically use the `read_csv` method of the `pandas` module[1][2] in order to load the datasets into `DataFrame` data structures. These data structures are particularly useful as they can hold two-dimensional, size-mutable and potentially heterogeneous tabular data. Furthermore, we chose to use this data structure as it provides us with helpful methods for analysing, manipulating and transforming data[3].

Once the `DataFrame` data structures are created, one for each of our `.csv` files, we create a dictionary with the keys `X_train`, `X_test` and `y_train` and it is returned by our method, to be used in other parts of the program.

3 Data Cleaning

In order to determine whether data cleaning is necessary, we first need to understand what our training dataset is made up of. As we mentioned in the previous section, the files `X_train.csv` and `X_test.csv` are

composed of rows, each of which represent an image and columns that represent the features extracted from the image. As outlined in the specification, each row has 964 columns and they're made up as follows:

- The first 900 columns correspond to a histogram of oriented gradients (HoG) extracted from the image.
- The next 16 columns are drawn from a normal distribution.
- The last 48 columns correspond three colour histograms extracted from the image one for each channel (red, green, blue), with 16 bins per channel.

The first thing we chose to do was to check whether the dataset is well formed and has the correct dimensionality. For this, I used the `pandas.DataFrame.info` method. This revealed the following information:

- For the binary classification dataset:
  - `X_train`: Contains 62209 entries with 964 columns each, containing values of type `float64`.
  - `y_train`: Contains 62209 entries with each entry either being `background` or `seal`.
  - `X_test`: Contains 20334 entries with 964 columns each, containing values of type `float64`.
- For the multi-class classification dataset:
  - `X_train`: Contains 62209 entries with 964 columns each, containing values of type `float64`.
  - `y_train`: Contains 62209 entries with each entry either being one of the following `background`, `whitecoat`, `juvenile`, `moulted pup` and `dead pup`.
  - `X_test`: Contains 20334 entries with 964 columns each, containing values of type `float64`.

There are several observations that we can use to conclude that the datasets are well formed. The first observation is that the entries of our `X_train` datasets match the entries of their respective `y_train` datasets. Secondly the `X_train` datasets have the correct number of columns, matching the number of extracted features for each image, and they also only contain values of type `float64` which means that there are no `null` values present or values of some other type that would make an entry invalid.

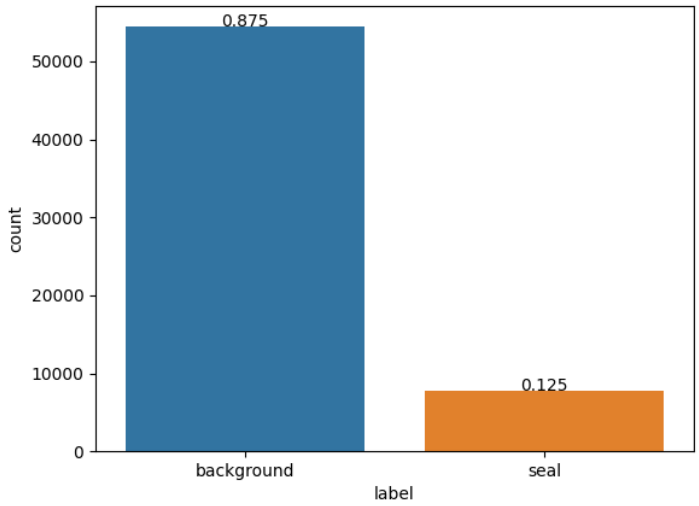
With that being said, there doesn't seem to be any relationship between the values drawn from a normal distribution and the image an entry represents. For this reason, I decided to remove these 16 columns from our datasets. As a result, the resulting `X_train` and `X_train` datasets will now be made up of 948 columns instead of the initial 964.

Note: It seems quite peculiar that the `X_train` and `X_test` datasets for both classification tasks contain the same number of entries (62209 and 20334 respectively). To check whether these datasets are the same for both tasks or not, we sorted them and we used the `DataFrame.equals` method. This returned `False` meaning that the datasets are indeed not the same.

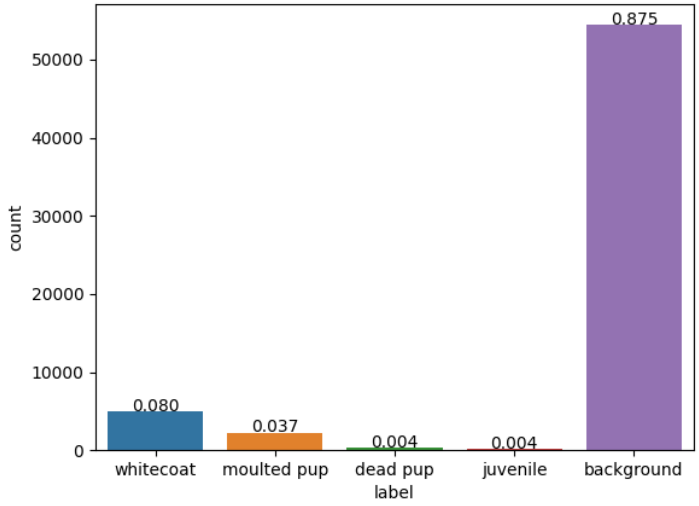
4 Data Analysis & Visualisation

At this point we will set aside the testing sets and we will focus on the training sets only. We will not perform any analysis or data visualisation on the testing sets as that would constitute data leakage. Data leakage occurs when information outside the training set affects decision making with regards to our classification model and thus invalidating the ability of said model to generalise[7].

In order to get a better understanding of the training data, I plotted a `countplot` which is essentially shows the count of each of the classification labels in our datasets[8].



(a) Binary Classification Dataset



(b) Multi-Class Classification Dataset

Figure 1: Countplots of the classification labels from our datasets.

From the above figures it is easy to see that the vast majority of our training samples are images of the `background` class, with that being 87.5% for both of our datasets. In the dataset for our multi-class classification task, the frequency of each type of seal image varies, from 8% for images of the class `whitecoat` to 3.7% for images of class `moulted pup` and 0.4% for images of class `dead pup` and `juvenile`.

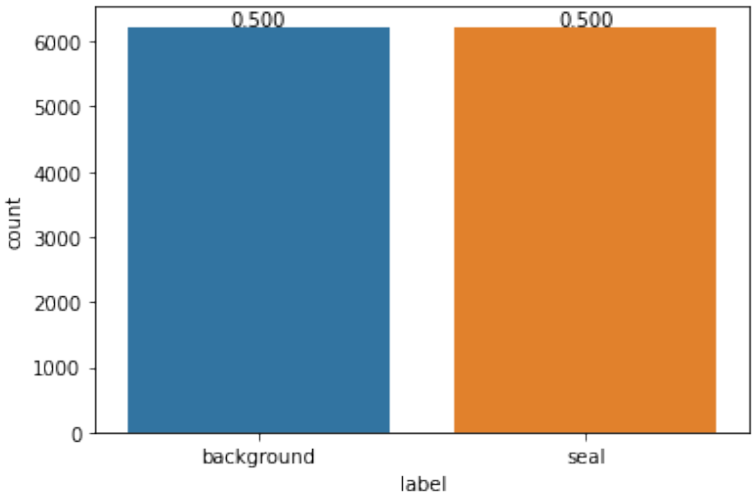
5 Validation Set

Since we don't have the `y_test` dataset, we cannot use the `X_test` in order to evaluate the performance of our classifiers. For this reason, I have created an additional set by splitting the training dataset into two parts, the set of data that will be used for training and the set of data that can be used for evaluation which we will refer to as the validation set. This set was obtained by using the `sklearn.model_selection.train_test_split`[9], setting the `test_size` parameter as 0.2, meaning that the validation set we obtain will be 20% of the training data.

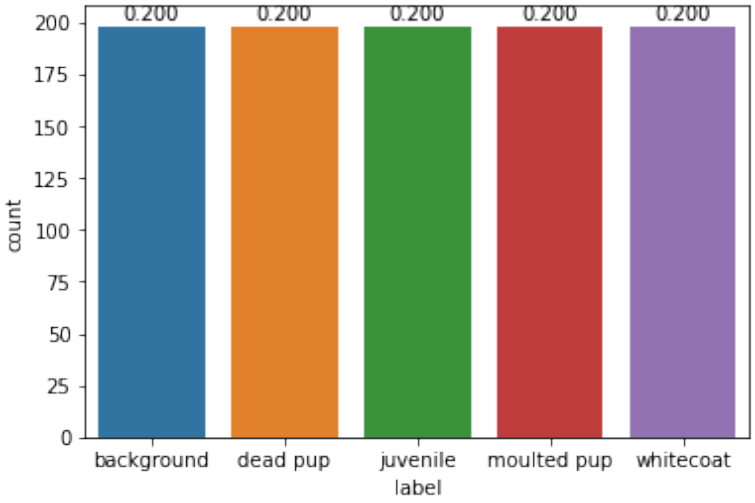
6 Dataset Imbalance

By looking at Figure 1, we can see that our datasets are severely imbalanced, a fact that may cause come problems down the line, such as a poor performance on predicting the minority classes on each of the tasks which, in our case, are the classes we are most interested in identifying. One remedy of imbalanced datasets is resampling our dataset so that it is more balanced by either oversampling, which is adding copies of the under-represented classes or by undersampling, which is deleting instances of the over-represented class[10]. Given the fact that the size of our dataset is large, I chose to carry out undersampling.

In the binary dataset, I have performed random undersampling of the background class by using the `imblearn.under_sampling.RandomUnderSampler` class of the `imbalanced-learn` library[11]. The resulting dataset is one that consists of 12444 samples, 6222 for each of the two classes. For the purposes of the dataset used for multi-class classification, we performed random undersampling for the majority classes, that are the `background`, `whitecoat` and `moulted pup` classes. The resulting dataset is one that consists of 990 samples, with 198 samples for each of the 5 classes.



(a) Binary Classification Undersampled Dataset



(b) Multi-Class Classification Undersampled Dataset

Figure 2: Countplots of the classification labels from our undersampled datasets.

Note: I have not performed undersampling on the testing set or the validation set, since we except the distribution of classes in a potential deployment scenario to be very similar to the one found in our original dataset. As a result if we were to perform undersampling on those sets we would not get evaluation metrics that would represent the classifier's performance in a real-life setting.

7 Preparing Inputs & Choosing Features

The first step I take is to use `sklearn.StandardScaler`[12] in order to standardise the features by removing the mean and scaling to unit variance, that is, the distribution of our input features have a mean value of 0 and a standard deviation equal to 1. This is important as many machine learning estimators expect features to be standardised, such as algorithms that use gradient descent as their optimisation technique, while other distance based algorithms are highly sensitive to the range of features[13].

Following that, I use the Principal Component Analysis (PCA) technique in order to reduce the dimensionality of the data. Essentially, this procedure converts a set of related variables into a set of unrelated ones by using an orthogonal transformation[14]. The motivation behind this choice is the mitigation of the problems that come with the 'curse' of high-dimensionality such as the problem of overfitting which reduces a model's ability to generalise[15]. To do this, we used the `sklearn.decomposition.PCA` class[16], with attributes `svd_solver='full'` and `n_components=0.99` so that the number of components selected is such that their explained variance is greater than 99%. This technique reduces the dimensionality of the input from 948 down to 473 in the original dataset. However when the dataset is undersampled, PCA reduces the number of components to 489 in the binary classification dataset and 383 in the multi-class classification dataset.

8 Cross Validation

During the model training process, I've incorporated the cross-validation resampling procedure[17], by using the `sklearn.model_selection.KFold` class[18]. What this procedure does it it takes the training data and it is split into k folds (in our case $k = 5$). Then, for each fold the following steps are carried out:

1. Take one fold and use it as the validation set.
2. Take the remaining 4 folds and use them as the training set.
3. The model is then trained on the training set and its performance is evaluated on the validation set.
4. The model's evaluation score is stored and the model is discarded.

After this process is complete we can use the scores collected to evaluate the validation performance of our model. This is useful as it gives us a rough idea on how well this model would perform on unseen data, a fact that can be used when faced with the decision of choosing between different classifiers.

The evaluation metric used in the cross-validation procedure is the balanced accuracy score[19] which is essentially the average of the recall for each class. This is used due to the unbalanced nature of our dataset. Had we used a different metric such as accuracy can give us misleading results as we may report a very high accuracy even if the model is only good at identifying the majority class in our dataset. Furthermore, this choice can be further justified by the fact that the classes we are most interested in are the minority classes, which are the classes representing seals. Our model would be no good if it was unable to detect these in a potential deployment scenario.

9 Classification Models

In this section I will be discussing the classifiers I've chosen for this practical.

Note: The hyperparameters used for the classifiers can be found in the appendix of this document.

9.1 K-Nearest Neighbours

The k-Nearest neighbours (KNN) algorithm is a rather simple yet widely used machine learning algorithm in which the dataset is stored and when a prediction is required, the k nearest samples of the training dataset are located and a prediction is made based on the most common outcome[20]. For the purposes of this practical, I used the `sklearn.neighbors.KNeighborsClassifier` implementation[21].

9.2 Random Forest

A random forest classifier is an ensemble which consists of a number of decision trees which are trained on sub-samples of the dataset, and it then uses averaging to improve its performance and to reduce the problem of overfitting. I have decided to use this classifier since decision trees have been reported to perform well on imbalanced datasets due to the fact that by learning through following a hierarchy of essentially if/else statements, they have to address all of the classes present in the dataset. [22]. For the purposes of this practical I have used the `sklearn.ensemble.RandomForestClassifier` implementation[23].

9.3 SVM

Support vector machines (SVM) models are widely used and they're some of the most popular machine learning models. Support vector machines essentially use a hyperplane to separate the dataset into the different classes. While there can be many ways to use a hyperplane to separate a dataset, it is chosen in a way such that it maximises the margin between a subset of data points, the support vectors, which are the data points that are the most difficult to classify[24]. For the purposes of this practical we have used the `sklearn.svm.LinearSVC` implementation, which is an SVM that uses a linear kernel. It is worth noting that when performing multi-class classification, this SVM will train one-vs-rest classifiers, instead of training one-vs-one classifiers for each pair of classes, as some other SVM implementations do.

10 Evaluation & Model Comparison

For this practical we were required to train classifiers to perform two classification tasks - a binary and a multi-class one. In this section I will be outlining, comparing and discussing the performance of the models on both of the above tasks. As mentioned earlier in section 5, we have set aside a sub-set of the data set to be used as a validation set, and it is by using this set that we evaluate the performance of our models.

Note: The values reported for each metric are the macro-averages across the different classes. We do so due to the fact that the dataset is imbalanced towards the class that we are the least interested in. As a result, weighted averages are not very meaningful, since they will be inflated and unrepresentative of how the classifiers perform on the minority classes. Since macro-averages are imbalance insensitive, we chose to report these in this section. The full classification reports which contain the values for the class weighted metrics as well, can be found in the `results` directory of the submission. Furthermore, I have included the confusion matrix for each of the experiments in the appendix of this document which show how the well the classifiers perform on each class.

10.1 Binary Classification

At first we will compare the performance of the different classifiers at the binary classification task, when trained on the original, non under-sampled dataset.

Classifier	Accuracy	Precision	Recall	f1-score
KNN	0.94	0.96	0.78	0.84
RF	0.94	0.96	0.77	0.83
SVM	0.97	0.95	0.9	0.92

Table 1: Results for the binary classification task, when trained on the original dataset.

The following are the results when the classifiers were trained on the undersampled dataset, which was obtained as described in section 6:

Classifier	Accuracy	Precision	Recall	f1-score
KNN	0.95	0.94	0.83	0.88
RF	0.86	0.73	0.89	0.77
SVM	0.94	0.83	0.93	0.87

Table 2: Results for the binary classification task, when trained on the undersampled dataset.

10.2 Multi-Class Classification

Now, we will comparing the performance of the different classifiers at the multi-class classification task, when trained on the original, non under-sampled dataset.

Classifier	Accuracy	Precision	Recall	f1-score
KNN	0.92	0.6	0.35	0.4
RF	0.91	0.34	0.29	0.31
SVM	0.94	0.66	0.47	0.48

Table 3: Results for the multi-class classification task, when trained on the original dataset.

The following are the results when the classifiers were trained on the undersampled dataset:

Classifier	Accuracy	Precision	Recall	f1-score
KNN	0.87	0.45	0.43	0.38
RF	0.68	0.31	0.57	0.31
SVM	0.56	0.28	0.49	0.26

Table 4: Results for the multi-class classification task, when trained on the undersampled dataset.

11 Critical Discussion

11.1 Dataset

The nature of our datasets presented several difficulties, due to the fact they are very imbalanced, with the vast majority of the samples belonging to the `background` class. To make matters worse, in the multi-class dataset there is a big difference between the frequency for each class of seals, with `whitecoat` dominating the remaining three labels, and `dead pup` having the least amount of samples along with `juvenile`. As described in section 6, we used undersampling in an attempt to remedy this.

In addition the original dataset contained contained some imperfections, specifically 16 features that were drawn from a normal distribution. These did not seem to be related to the dataset in any way, and that is why we decided to remove these, as described in section 3.

Furthermore, the high-dimensionality of the input also presented some difficulties. This is an issue referred to as the 'curse of dimensionality' which entails high sparsity in our dataset and an increase in our algorithm's time and storage requirements[26]. As described in section 7 we attempted to reduce the effects of these issue through applying the PCA dimensionality reduction technique.

11.2 Approach

The approach I followed was designed to cope with the nature of the provided dataset. Most decisions I have taken such as the decision to experiment with an undersampled dataset, applying PCA and even the choice of our classifiers were motivated from our dataset's traits. Furthermore, the validation process I have followed is quite robust, as I have taken a subset of the training set aside before doing anything else and I have used this as a mock test set to evaluate the final performance of our classifiers, as reported in section 10. Lastly, by choosing to optimise for the balanced accuracy metric, I have ensured that my classifiers are able to adequately classify samples belonging in the minority classes as well as the majority class, something which I deemed very important as the minority classes are of higher interest to us given the intended application domain of the classifiers.

11.3 Binary Classification Results

In section 10.1 I have outlined the results for the binary classification task. By looking at the tables we can make the following observations:

- When the classifiers were trained on the undersampled dataset, their macro-average precision increased while their precision has decreased. The f1-score increased only for the KNN whereas it decreased for the RF and the SVM. Accuracy followed the opposite trend, where undersampling increased the accuracy score only for the KNN (although by only 1%) but it decreased for the RF and the SVM. By looking at the results, we can deduce that if high recall is important, undersampling an imbalanced dataset is an effective way to achieve that.
- When the classifiers were trained on the original data, the SVM scored the highest accuracy, recall and f1-score while KNN and RF tied for the highest precision score, beating the SVM for just 0.01. From examining this data we can conclude that the SVM is the superior choice for a classifier of the three.
- When the classifiers were trained on the undersampled data, the KNN achieved the highest accuracy, precision and f1-score, while the SVM achieved the highest recall. We can see that on a smaller dataset, the KNN is generally performing better out of the three classifiers. With that being said, the SVM achieves the highest recall in this case as well.

11.4 Multi-Class Classification Results

In section 10.2 I have outlined the results for the multi-class classification task. By looking at the tables we can make the following observations:

- In this classification task, undersampling the dataset causes the accuracy, precision and f1-score to fall by a significant amount. The poor performance is due to the fact that the smallest classes, 'dead pup' and 'juvenile', down to which all the other classes are undersampled, make up just 0.4% of the dataset, a tiny portion of the original dataset. As a result the entire dataset consist of 198 samples for each class, which given the dimensionality of the input it is very small.
- When the classifiers were trained on the original dataset, the SVM achieved the highest scores in all of the metrics. We can thus come to the conclusion that the SVM is the superior option of the three for performing multi-class classification on an imbalanced dataset. The second best classifier was the KNN while RF achieved the lowest scores out of the three.
- When the classifiers were trained on the undersampled dataset however, the SVM perform the poorest across all metrics. In fact KNN achieved the highest accuracy, precision, recall and f1-score while RF achieved the highest recall. Once again, we that KNN performs better on the smaller dataset, an observation we made when were analysing the performance of our classifiers on the binary classification task.

11.5 Overall Results

Overall, we can conclude that the SVM shows the most promising results achieving high scores across all metrics in both of our classification tasks. It is if for this reason we will be using it to produce the required predicted output files. I have entered the predictions of my best performing classifier in the competition server set up for this practical, with which I achieved an accuracy score of **0.9823** and **0.97472** in the binary and multi-class tasks respectively. These are very promising results which proves that our approach was successful.

Furthermore, we can conclude that while undersampling a dataset has some merit, as it has improved the recall of our classifiers, it has adversely affected all other metrics, especially in the multi-class classification task and in this case, it may not be an ideal way to deal with the imbalance in our dataset.

12 Evaluation & Conclusion

Overall I'm very happy with the work I've produced as I've successfully created three different classification models which I've trained using a number of machine learning techniques, completing all of the requirements set out by the specification. Furthermore, I am very confident in the approach that I followed as my best performing classifier achieved very scores in the competition organised for this practical.

If I had more time, I would have performed a systematic grid search which unfortunately due to time restrictions and hardware limitations of my personal machine, I was not able to carry out. In addition, I would have liked to explore additional classifiers such as XGBoost, which has been used to achieve state of the art results in many classification tasks. Lastly, I would have explored additional resampling techniques in order to deal with the dataset imbalance, such as the SMOTE oversampling technique.

This practical was very successful in achieving its learning aims as I have gained knowledge and practical experience of how to deal with a real-life and imperfect dataset, a skill that is particularly useful as this is a trait shared by most datasets in existence.

References

[1] pandas.read_csv — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

[2] pandas - Python Data Analysis Library. (2020). Retrieved 17 April 2020, from https://pandas.pydata.org/

[3] pandas.DataFrame — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

[4] (2020). Retrieved 17 April 2020, from https://studres.cs.st-andrews.ac.uk/CS5014/Practicals/P2/P2.pdf

[5] pandas.DataFrame.info — pandas 1.0.1 documentation. (2020). Retrieved 3 March 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html

[6] pandas.DataFrame.equals — pandas 1.0.3 documentation. (2020). Retrieved 17 April 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.equals.html

[7] Brownlee, J. (2020). Data Leakage in Machine Learning. Retrieved 17 April 2020, from https://machinelearningmastery.com/data-leakage-machine-learning/

[8] seaborn.countplot — seaborn 0.10.0 documentation. (2020). Retrieved 17 April 2020, from https://seaborn.pydata.org/generated/seaborn.countplot.html

[9] sklearn.model_selection.train_test_split — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[10] Brownlee, J. (2020). 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset. Retrieved 22 April 2020, from https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[11] 3. Under-sampling — imbalanced-learn 0.5.0 documentation. (2020). Retrieved 23 April 2020, from https://imbalanced-learn.readthedocs.io/en/stable/under_sampling.html

[12] sklearn.preprocessing.StandardScaler — scikit-learn 0.22.2 documentation. (2020). Retrieved 22 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[13] Standardization, F. (2020). Feature Scaling — Standardization Vs Normalization. Retrieved 22 April 2020, from https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

[14] ML — Principal Component Analysis(PCA) - GeeksforGeeks. (2020). Retrieved 22 April 2020, from https://www.geeksforgeeks.org/ml-principal-component-analysispca/

[15] The Curse of Dimensionality in Classification. (2020). Retrieved 22 April 2020, from https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

[16] sklearn.decomposition.PCA — scikit-learn 0.22.2 documentation. (2020). Retrieved 22 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[17] Brownlee, J. (2020). A Gentle Introduction to k-fold Cross-Validation. Retrieved 4 March 2020, from https://machinelearningmastery.com/k-fold-cross-validation/

[18] sklearn.model_selection.KFold — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 March 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[19] sklearn.metrics.balanced_accuracy_score — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html

[20] Brownlee, J. (2020). Develop k-Nearest Neighbors in Python From Scratch. Retrieved 23 April 2020, from https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

[21] sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.22.2 documentation. (2020). Retrieved 23 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[22] Methods for Dealing with Imbalanced Data. (2020). Retrieved 4 May 2020, from https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

[23] 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[24] (2020). Retrieved 4 May 2020, from http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

[25] sklearn.svm.LinearSVC — scikit-learn 0.22.2 documentation. (2020). Retrieved 4 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[26] Pore, P. (2020). Must-Know: What is the curse of dimensionality? - KDnuggets. Retrieved 5 May 2020, from https://www.kdnuggets.com/2017/04/must-know-curse-dimensionality.html

A Classifier Hyperparameters

A.1 K-Nearest Neighbours

- algorithm : auto
- leaf_size : 30
- metric : minkowski
- metric_params : None
- n_neighbors : 3
- p : 2
- weights : uniform

A.2 Random Forest

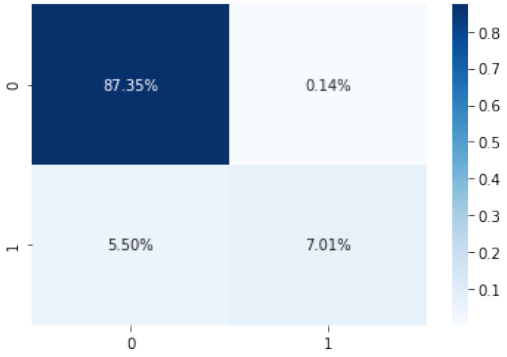
- bootstrap : True
- class_weight : None
- criterion : gini
- max_depth : None
- min_impurity_decrease : 0.0
- min_impurity_split : None
- min_samples_leaf : 1
- min_samples_split : 2
- min_weight_fraction_leaf : 0.0

A.3 SVM

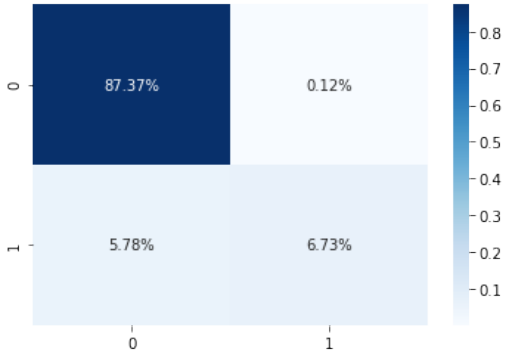
- C : 1.0
- class_weight : None
- dual : False
- fit_intercept : True
- intercept_scaling : 1
- loss : squared_hinge
- max_iter : 1000
- multi_class : over
- penatly : l2
- tol : 0.0001

B Confusion Matrices

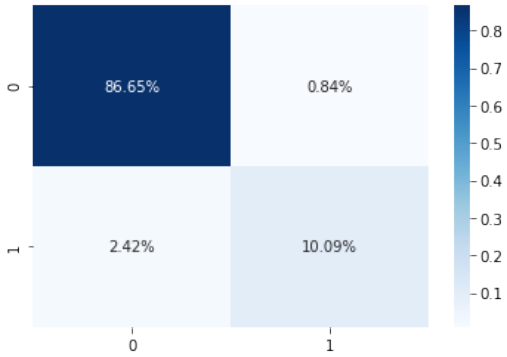
B.1 Binary Classification Task



(a) KNN

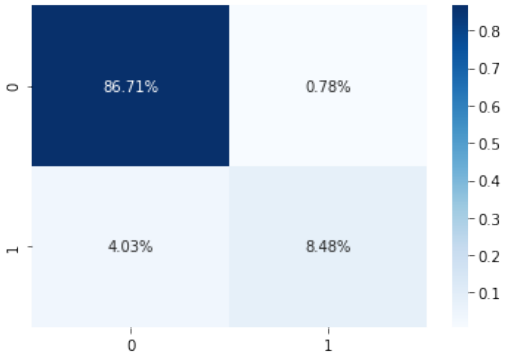


(b) RF

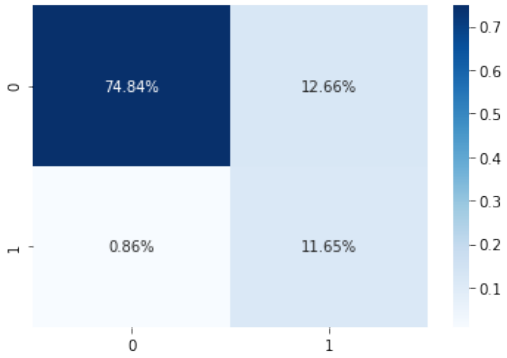


(c) SVM

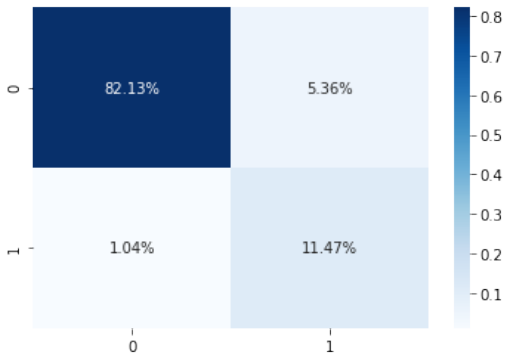
Figure 3: Confusion Matrices for the binary classification task, when trained on the original dataset.



(a) KNN



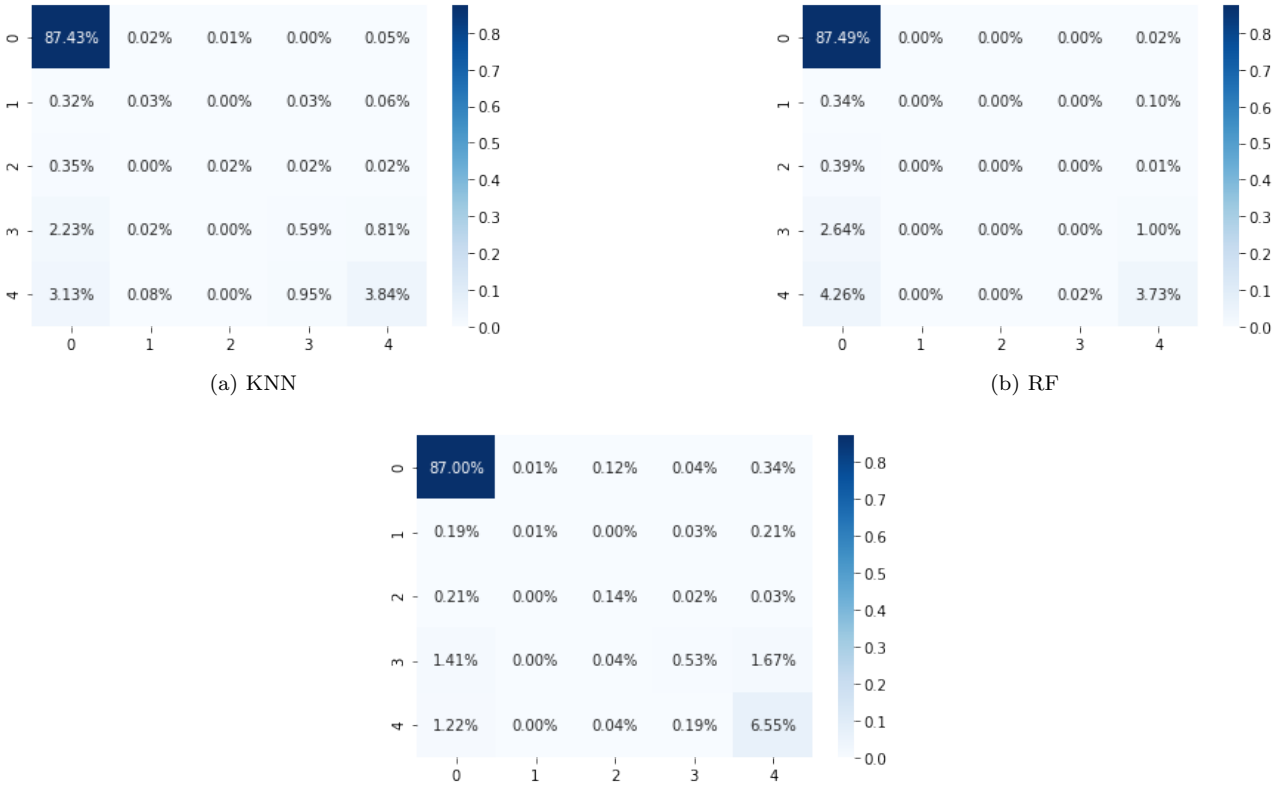
(b) RF



(c) SVM

Figure 4: Confusion Matrices for the binary classification task, when trained on the undersampled dataset.

B.2 Multi-class Classification Task

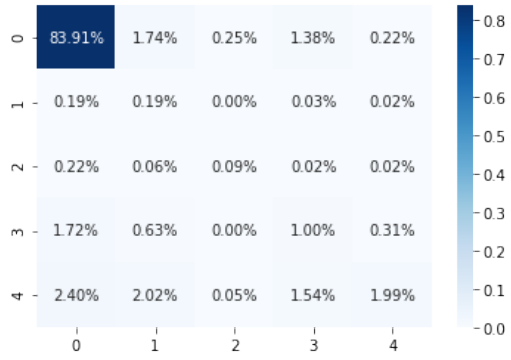


(a) KNN

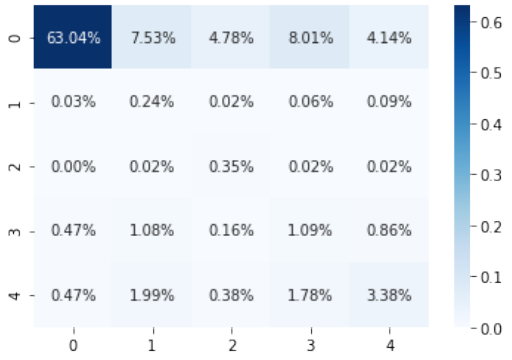
(b) RF

(c) SVM

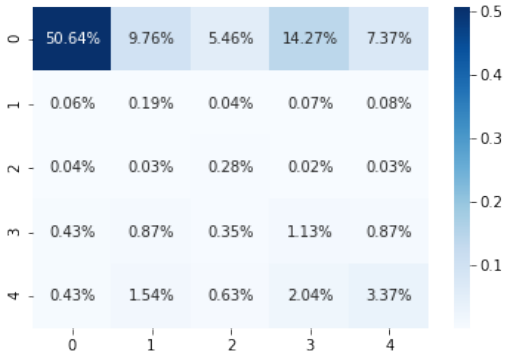
Figure 5: Confusion Matrices for the multi-class classification task, when trained on the original dataset.



(a) KNN



(b) RF



(c) SVM

Figure 6: Confusion Matrices for the multi-class classification task, when trained on the undersampled dataset.