

Configuration Manual

MSc Research Project
Data Analytics

Christopher Signorelli
Student ID: 19181027

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Christopher Signorelli
Student ID:	19181027
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	19/9/2022
Project Title:	Configuration Manual
Word Count:	2715
Page Count:	39

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	18th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Christopher Signorelli
19181027

1 Introduction

This configuration manual describes the hardware/software platforms, data sources, code, figures, and files associated with the research project described in (Signorelli; 2022).

- OBJ-1: Create a sufficiently large training dataset that combines gene-expression data, drug chemical properties, and cardiotoxicity outcome labels.
- OBJ-2: Perform feature selection for dimension reduction, and minimal model performance degradation.
- OBJ-3: Implement a MLC to match the state-of-the-art, capable of predicting multiple cardiotoxicity outcomes.
- OBJ-4: Automate the hyperparameter search for optimising the MLC, for training a robust MLC using the optimised hyperparameters.
- OBJ-5: Demonstrate the efficacy of the robust MLC and automation process, relative to other models.
- OBJ-6: Benchmark the computation times of the training experiments.

Hardware, Software and Installation: All code was implemented and run on a standard home laptop, with the main hardware and software specifications shown in Table 1. Before installing the required Anaconda virtual environment, Anaconda needs to be installed on the local machine (ideally the version in Table 1). A Windows batch script, *install_signorelli_env.bat*, has been written to automate this process, and is included in the artefacts zip file. The script creates a new environment, installs the dependencies defined in *requirements_signorelli.txt*, then starts up Jupyter Notebook.

Notebook Files: The code has been modularised into logically separated Notebook files that need to be run sequentially, as shown in Figure 1, and detailed in *README.pdf*. Two of these Notebook files need to (optionally) have a small number of constants modified prior to execution, and are clearly marked at the top of the relevant Notebooks. These control whether the code runs a short demo mode that can complete in approximately 20 min, OR the full set of training experiments that takes approximately 1 week. By default it is set to short demo. The step-by-step instructions for using the installation script and running the project code are provided in the *README.pdf* file, in the artefacts zip file.

Standalone Machine	HP Pavilion laptop
Operating System	Windows 10 Home
Processor	6-core AMD Ryzen 5, 4500U, 2.375 GHz
Graphics Card	Radeon
RAM	8 GB
Hard Drive (HDD)	500 GB
Minimum Free HDD Space	20 GB
Anaconda	4.10.1
Jupyter	1.0.0
Python	3.8.13
Python Module: scikit-learn ¹	1.1.1
Python Module: cmappy ²	4.0.1
Installation Script	install_signorelli_env.bat

Table 1: Hardware and Software Specifications.

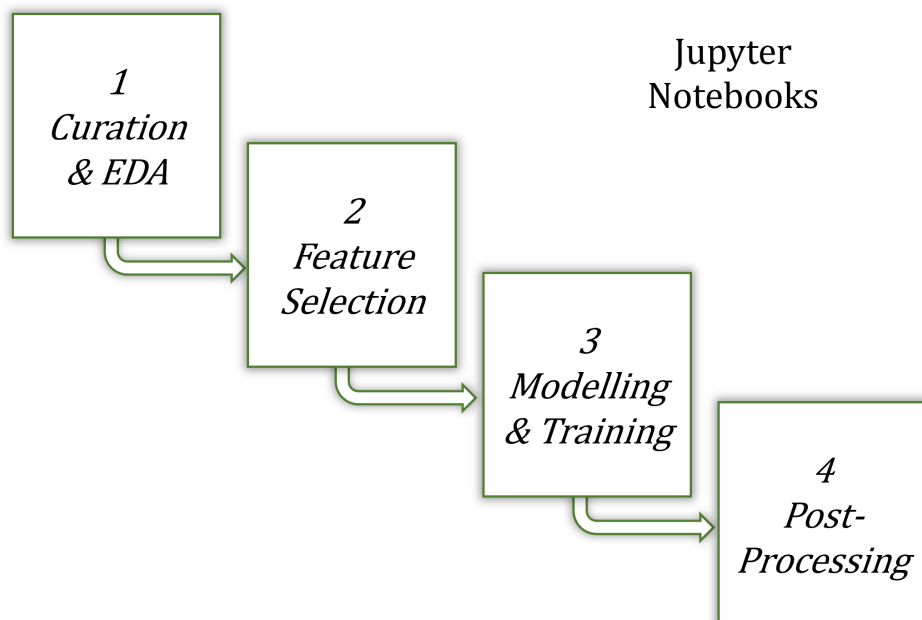


Figure 1: Notebook Execution Order.

The functionality of each Notebook is summarised as follows:

1. **Curation and Exploratory Data Analysis:** This Notebook downloads the required data from the public NCBI database³, and loads the Mamoshina data from the project execution path. EDA is performed on the data which are merged, cleaned, and standardised.
2. **Feature Selection:** Recursive Feature Elimination (RFE) is performed on the data from the previous step to decrease feature dimensionality and produce as parsimonious a model as possible.
3. **Modelling and Training:** The selected features from the previous step are used to train several individual classifier chains, using random forest classifiers as the base model.
4. **Post-Processing:** The individual chains are imported, and analysed to determine the best possible hyperparameter values to train the *Best Means Chain*, which is then trained. Performance is compared with an Ensemble chain, and all individual chains. Various performance metrics are compared, with analyses conducted for ensemble size and computation times.

³<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE92742>

2 Curation and Exploratory Data Analysis Notebook

The main steps in this Notebook are:

1. **Download CMap gene-expression data** (Figure 2). Due to the large size of the CMap dataset, the code downloads it from the NCBI FTP server. It usually takes less than 3 min to download, and less than that to unzip it, which is also implemented in the code. Note that the perturbation (*sig-info*) and gene (*gene-info*) meta data are also downloaded and unzipped in the same manner.

Download and Unzip CMap Gene-Expression Data

- The CMap data is downloaded from the web due to being too big to submit with this project.

```
In [7]: ▶ 1 # CMap data source.
2 source_url_cmap_data = 'https://ftp.ncbi.nlm.nih.gov/geo/series/GSE92nnn/GSE92742/
3 dest_cmap_file = f"{PROJECT_PATH}/{source_url_cmap_data.split('/')[-1]}"
4
5 # Unzipped dataset file name.
6 GSE92742_gctx_file_level_2 = dest_cmap_file.split('.gz')[0].split('/')[-1]
```

```
In [8]: ▶ 1 # Download CMap datafile.
2 print('Download should take approximately 3 min.....')
3 download_url_file(source_url_cmap_data, dest_cmap_file)
4
5 # Unzip file.
6 print('\nUnzipping should take approximately 2 min.....')
7 unzip_file(dest_cmap_file)
```

Download should take approximately 3 min.....
Downloading: ./GSE92742_Broad_LINCS_Level2_GEX_epsilon_n1269922x978.gctx.gz from:
https://ftp.ncbi.nlm.nih.gov/geo/series/GSE92nnn/GSE92742/suppl/GSE92742_Broad_LINCS
Download complete. Time taken: 78.96 s.

Unzipping should take approximately 2 min.....
Unzipping: ./GSE92742_Broad_LINCS_Level2_GEX_epsilon_n1269922x978.gctx.gz ...
Unzipping complete. Time taken: 76.77 s.

Figure 2: CMap Data Download.

2. **Load Mamoshina data.** Figure 3 shows the a snippet of the imported Mamoshina gene features and labels. These data were obtained directly from the Authors of (Mamoshina et al.; 2020) via an email-provided link. Note that the Mamoshina gene features are discarded, as discussed in (Signorelli; 2022).

Load Mamoshina Training and Test Data

```

1 x_train = pd.read_csv(f'{PROJECT_PATH}/m_x_train.txt', delimiter='\t')
2 y_train = pd.read_csv(f'{PROJECT_PATH}/m_y_train.txt', delimiter='\t')
3 x_test = pd.read_csv(f'{PROJECT_PATH}/m_x_test.txt', delimiter='\t')
4 y_test = pd.read_csv(f'{PROJECT_PATH}/m_y_test.txt', delimiter='\t')
5
6 display(x_train.head())
7 display(y_train.head())
8 display(x_test.head())
9 display(y_test.head())

```

	AARS	ABCB6	ABHD6	ACLY	ADCK3	ADRB2	ALDOA	ANO10	
1	5132.125916	467.804172	48.646005	2477.175182	22.821061	24.135693	578.397678	159.513293	539.
2	2940.207696	429.410662	67.929398	704.461539	16.707369	27.266940	552.135698	96.830051	674.
3	2074.853176	341.142882	64.650364	2417.293726	16.947446	15.376390	955.924296	114.906136	647.
4	3110.488907	259.737644	54.893478	1825.930595	24.339630	21.907402	611.936578	141.073582	533.
5	1784.303987	204.233738	55.133039	3525.667627	16.763129	13.969898	390.147670	85.828144	1370.

	Vascular.disorders	Cardiac.disorder.signs.and.symptoms	Cardiac.arrhythmias	Heart.failures	Coronary.a
1	0	0	0	0	
2	0	0	0	0	
3	0	1	1	0	
4	0	1	1	0	
5	0	1	1	0	

Figure 3: Imported Mamoshina Data.

3. **Extract the genes and drugs from the Mamoshina data.** From the loaded Mamoshina data, the set of genes are extracted from the header columns, as well as the molecular descriptors, fingerprints, and labels. The drug ID codes are extracted from the ID column. Figure 4 shows a snippet of the extracted information.

index	Mamoshina Columns	Feature Type	Mamoshina Gene in CMap	
0	0	AARS	Gene	True
1	1	ABCB6	Gene	True
2	2	ABHD6	Gene	True
3	3	ACLY	Gene	True
4	4	ADCK3	Gene	True
...
336	336	V76	Fingerprint	False
337	337	V77	Fingerprint	False
338	338	V78	Fingerprint	False
339	339	V79	Fingerprint	False
340	340	id	Chemical ID	False

341 rows × 4 columns

Figure 4: Dataset Column Types.

4. **Find the corresponding CMap indexes in the meta data, using the Mamoshina genes and drugs.** Figure 5 shows the structure of the CMap data, where the columns represent the drug perturbations (signatures), the rows represent the genes, and the cells in the center are the gene-expression values. Using the gene and drugs information obtained in the previous step, the CMap ID numbers are extracted, which can then be used to parse only the desired data from the large CMap dataset. Figure 6 shows a snippet of the data in the *sig-info* meta data.

	A	B	C	D	E	F	G	H	I
1	#1.3								
2	10	6	2	5					
3	id	pr_gene_symbol	pr_is_lmark	LPROT001_A375	LPROT001_A375	LPROT001_A375	LPROT001_A375	LPROT001_A375	LPROT001_A375
4	pr_gene_symbol	-666	-666	DMSO	DMSO	BRD-K52313696	BRD-K52313696	BRD-K52313696	BRD-K77908580
5	pr_is_lmark	-666	-666	DMSO	DMSO	tacedinaline	tacedinaline	tacedinaline	entinostat
6	pr_treatment	-666	-666	A375	A375	A375	A375	A375	A375
7	pr_time	-666	-666	6	6	6	6	6	6
8	pr_dose	-666	-666	-666	-666	2	2	2	2
9	5720	PSME1	1	8.7980	8.9395	8.9561	9.4491	9.1994	9.2937
10	55847	CISD1	1	9.8349	9.5334	9.7543	9.9203	9.8904	10.0666
11	7416	VDAC1	1	12.5431	12.3479	12.4662	12.5892	12.7088	12.6397
12	10174	SORBS3	1	8.1017	8.5660	8.3563	8.6225	8.5800	8.3666
13	25803	SPDEF	1	11.0651	11.0922	11.3566	11.1527	11.0557	10.7427
14	466	ATF1	1	6.6887	6.8780	6.6684	7.1662	6.7492	6.7492
15	6676	SPAG4	1	3.5195	3.8840	3.6936	3.4822	3.2684	3.7300
16	1870	E2F2	1	4.5798	4.6260	4.4156	4.2644	4.4611	4.4611
17	6009	RHEB	1	11.7969	12.3234	12.0216	11.9772	12.0580	12.0216
18	3480	IGF1R	1	8.9370	10.0307	9.5279	10.0616	8.9231	8.6343

Figure 5: CMap Data Structure.

Get the Rows in sig_info that only have Mamoshina Drugs

```

1 mammo_drugs_train = list(x_train['id'].unique())
2 mammo_drugs_test = list(x_test['id'].unique())
3 mammo_drugs_train_test = mammo_drugs_train + mammo_drugs_test
4
5 print(f'NUM UNIQUE DRUGS IN mammo_drugs_train:\t\t{len(mammo_drugs_train)}')
6 print(f'NUM UNIQUE DRUGS IN mammo_drugs_test:\t\t{len(mammo_drugs_test)}')
7 print(f'NUM UNIQUE DRUGS IN mammo_drugs_train_test:\t\t{len(mammo_drugs_train_test)}')
8
9 # Find overlap between Mamoshina and CMap drugs.
10 mask_sig_info_mammo_drugs = sig_info['pert_id'].str.contains('|'.join(mammo_drugs_train_test))
11 print(f'Num drugs in both Mamoshina and CMap: {sum(mask_sig_info_mammo_drugs)}')
12
13 sig_info_mammo_drugs = sig_info[mask_sig_info_mammo_drugs]
14 display(sig_info_mammo_drugs)

```

```

NUM UNIQUE DRUGS IN mammo_drugs_train:      291
NUM UNIQUE DRUGS IN mammo_drugs_test:       66
NUM UNIQUE DRUGS IN mammo_drugs_train_test: 357
Num drugs in both Mamoshina and CMap: 12425

```

	sig_id	pr_gene_symbol	pr_is_lmark	pr_treatment	pr_time	pr_dose	pr_dose_unit	pr_idose	pr_time	
90	AML001_CD34_24H:BRD-K27316855:0.37037	BRD-K27316855	1	calcitriol	trt_cp	CD34	0.37037	µM	500 nM	24
91	AML001_CD34_24H:BRD-K27316855:1.11111	BRD-K27316855	1	calcitriol	trt_cp	CD34	1.11111	µM	1 µM	24
92	AML001_CD34_24H:BRD-K27316855:10	BRD-K27316855	1	calcitriol	trt_cp	CD34	10.0	µM	10 µM	24
93	AML001_CD34_24H:BRD-K27316855:3.33333	BRD-K27316855	1	calcitriol	trt_cp	CD34	3.33333	µM	3 µM	24
102	AML001_CD34_24H:BRD-K43389675:0.37037	BRD-K43389675	1	daunorubicin	trt_cp	CD34	0.37037	µM	500 nM	24

Figure 6: CMap Signature Meta Data.

5. **Parse the CMap gene-expression data using the indexes.** Figure 7 shows a snippet of the extracted (parsed) CMap data using the Mamoshina subsets of genes and drug IDs. The *cid* values are the CMap indexes for the drug perturbation signatures, which are found using the *pert_id* values shown in Figure 6.

Parse Data

```

1 gene_expr_level_2 = parse(
2     file_path=GSE92742_gctx_path_level_2,
3     rid=gene_rid_vals,
4     cid=drug_cid_vals
5 ).data_df.T
6
7 # Replace Gene IDs with Gene Names
8 gene_expr_level_2 = gene_expr_level_2.rename(columns=mamo_genes_id_name_lut)
9
10 # Sort column names to be in same order as Mamoshina dataset.
11 gene_expr_level_2 = gene_expr_level_2[sorted(gene_expr_level_2.columns)]
12
13 display(gene_expr_level_2)

```

	rid	AARS	ABCB6	ABHD6	ACLY	ADCK3	ADRB2	ALDOA
	cid							
	AML001_CD34_24H_X1_F1B10:C03	296.0	247.0	293.0	384.0	219.0	262.0	128.0
	AML001_CD34_24H_X1_F1B10:C04	157.0	275.0	395.0	659.0	248.0	432.0	258.0
	AML001_CD34_24H_X1_F1B10:D03	710.0	438.0	319.0	746.0	162.0	517.0	200.0
	AML001_CD34_24H_X1_F1B10:D04	453.0	283.0	354.0	637.0	142.0	401.0	235.0
	AML001_CD34_24H_X1_F1B10:D12	369.0	324.0	248.0	577.0	142.0	208.0	180.0

	RAD001_PC3_6H_X1_F1B5_DUO52HI53LO:J07	951.0	518.0	509.0	701.0	365.0	580.0	1127.0
	RAD001_PC3_6H_X1_F1B5_DUO52HI53LO:J08	1073.0	715.0	590.0	686.0	325.0	641.0	977.0
	RAD001_PC3_6H_X1_F1B5_DUO52HI53LO:J09	890.0	569.0	630.0	536.0	315.0	535.0	717.0
	RAD001_PC3_6H_X1_F1B5_DUO52HI53LO:J10	866.0	609.0	714.0	708.0	343.0	511.0	866.0
	RAD001_PC3_6H_X1_F1B5_DUO52HI53LO:J11	405.0	296.0	276.0	323.0	133.0	235.0	368.0

12425 rows x 252 columns

Figure 7: Parsed CMap Data.

6. **Join the gene-expression data with the perturbation meta data.** This is done to add potential features such as dose time, dosage amount, and cell lines.
7. **Join the features and labels from the Mamoshina dataset.** Figure 8 and Figure 9 partially show the resulting data frames after performing this and the last step.

	ALogP	ALogp2	AMR	MW	XLogP	apol	TopoPSA	tpsaEfficiency	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V1
1	-2.0096	4.038492	61.9671	254.101505	-1.876	34.543102	102.28	0.402516	0	0	0	0	0	0	1	0	1	0	0	1	1	
2	-2.0096	4.038492	61.9671	254.101505	-1.876	34.543102	102.28	0.402516	0	0	0	0	0	0	1	0	1	0	0	1	1	
3	-1.4962	2.238614	119.1990	415.272259	3.221	72.979341	81.95	0.197340	0	0	0	0	0	0	0	0	1	0	0	1	1	
4	-1.4962	2.238614	119.1990	415.272259	3.221	72.979341	81.95	0.197340	0	0	0	0	0	0	0	0	1	0	0	1	1	
5	-1.4962	2.238614	119.1990	415.272259	3.221	72.979341	81.95	0.197340	0	0	0	0	0	0	0	0	1	0	0	1	1	
...
1534	1.8073	3.266333	70.9511	282.061612	1.615	34.356137	62.22	0.220590	0	0	0	0	0	0	0	0	0	0	0	1	0	
1535	1.8073	3.266333	70.9511	282.061612	1.615	34.356137	62.22	0.220590	0	0	0	0	0	0	0	0	0	0	0	1	0	
1536	1.8073	3.266333	70.9511	282.061612	1.615	34.356137	62.22	0.220590	0	0	0	0	0	0	0	0	0	0	0	1	0	
1545	1.3098	1.715576	90.1276	265.157898	2.085	45.889067	27.63	0.104202	0	0	0	0	0	0	0	0	1	0	0	1	0	
1546	1.3098	1.715576	90.1276	265.157898	2.085	45.889067	27.63	0.104202	0	0	0	0	0	0	0	0	1	0	0	1	0	

1419 rows × 88 columns

Figure 8: Joined Dataset with Molecular Descriptor and Fingerprint Data.

rhythmias	Heart.failures	Coronary.artery.disorders	Pericardial.disorders	Myocardial.disorders	pert_time	AARS	ABCB6	ABHD6	ACLY	ADCK3
0	0	0	0	0	1.088940	-0.853271	-0.864225	-0.478906	-0.077411	-0.614872
0	0	0	0	0	-0.918324	-0.888629	-1.160543	-0.528463	-0.646796	-0.732607
0	0	0	0	0	1.088940	-0.890102	-0.965946	0.338791	5.156601	-0.312859
0	0	0	0	0	-0.918324	-0.059199	-1.209193	-0.451030	-0.662652	-0.558565
0	0	0	0	0	1.088940	-0.465811	-0.561272	-0.029792	-0.266245	0.183672
...
0	0	0	0	0	-0.918324	-0.038573	-0.041608	0.561799	-0.159576	0.398665
0	0	0	0	0	1.088940	-0.498222	-0.532524	-0.735985	-1.085005	-0.635348
0	0	0	0	0	-0.918324	-1.364483	2.545655	0.459587	0.156108	-0.256551
0	0	0	0	0	1.088940	1.218094	0.026944	0.202508	-0.333995	1.299588
0	0	0	0	0	-0.918324	-0.641126	-0.999116	-0.252801	-0.865901	-0.435712

Figure 9: Standardised Dataset.

8. Conduct EDA to get feature and label distributions.
9. Filter the data based on EDA. This is done to extract approximately 10k rows of training data. From the EDA, Figure 10 and Figure 11 show the distribution of cell lines in extracted CMap data. An important aspect of this project is increasing the size of the Mamoshina training set, which only had approximately 1000 rows. By including extra cell lines the row count increased to over 9000. The criterion for including extra cell lines was arbitrarily that they were associated with at least 100 rows.

Cell Line Counts

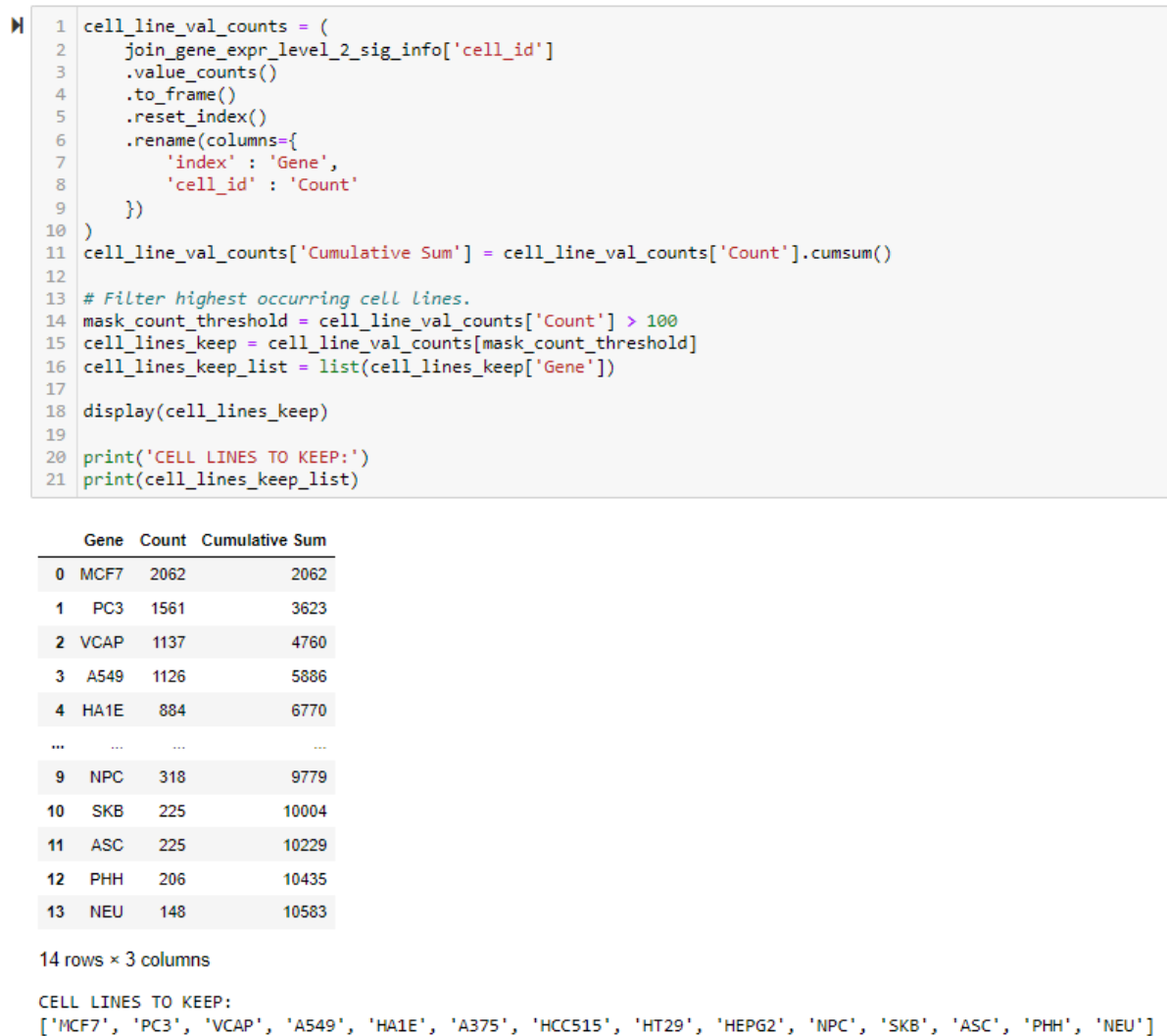


Figure 10: Cell Line Counts.

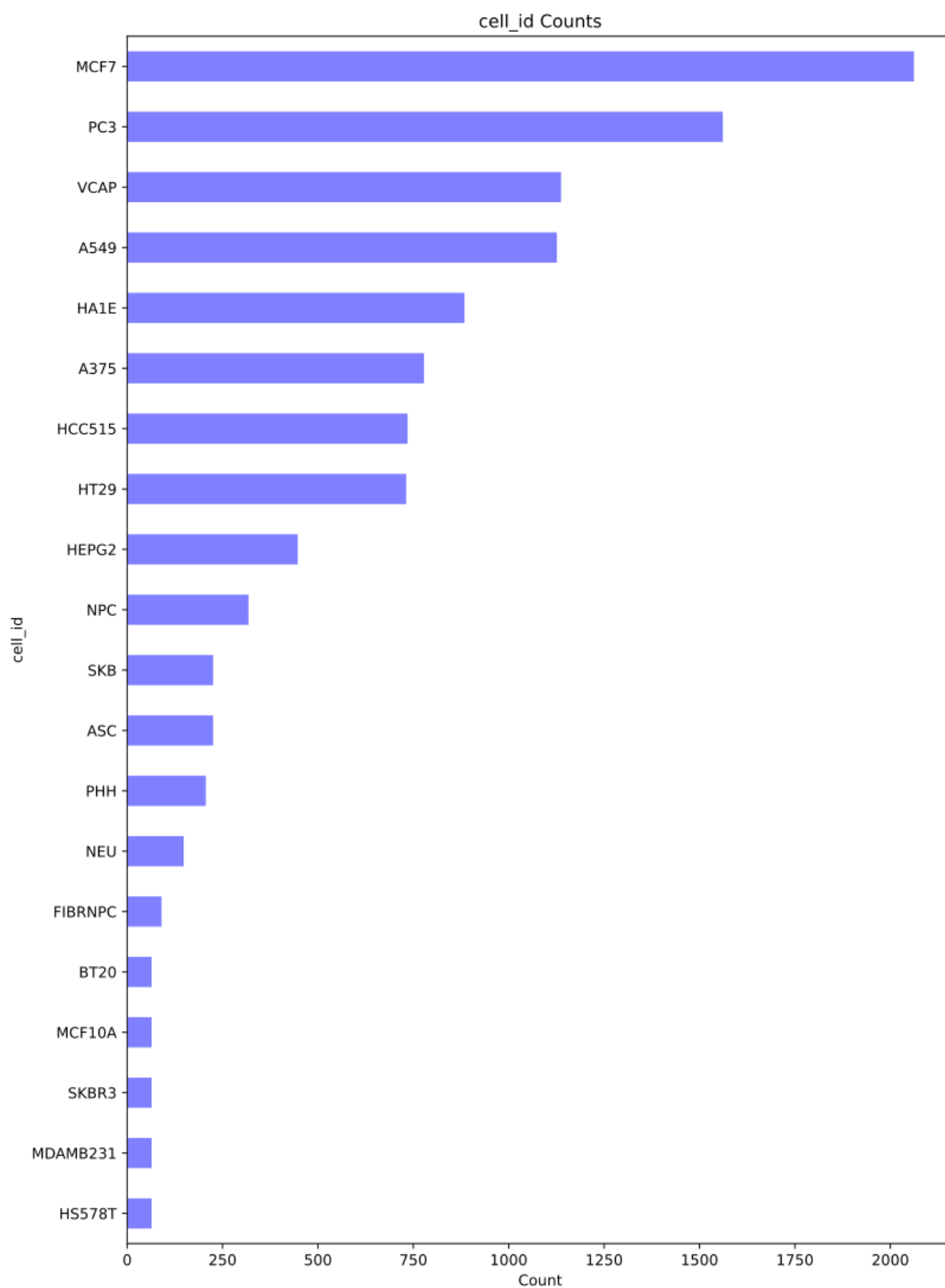


Figure 11: Cell Line Counts.

Figure 12 shows the distribution of the perturbation (dose) times, which has values of 6, 24, and 48 hours. While the counts for 6 and 24 are evenly balanced, the count for 48 is very low. For this reason, all rows with 48 hour dose times are removed. Figure 13 shows only 4 of the 254 gene-expression values in the dataset. A small amount of right-skew can be seen, however the distributions looks near-normal. The vast majority of genes show similar distributions. Figure 14 shows the level of imbalance in the label data, and high variability between each label. A proportion value of 0.5 indicates perfect balance between the positive and negative cases.

pert_time BEFORE Removing 48 Hours

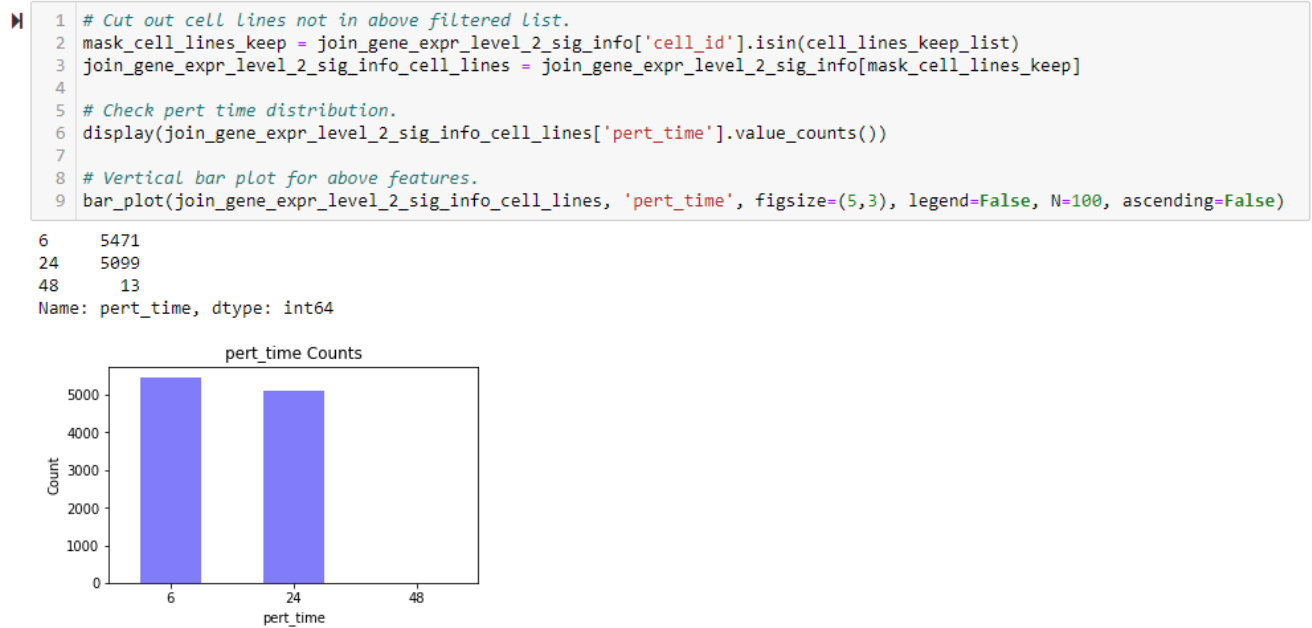


Figure 12: Imbalance in Perturbation (Dose) Time.

10. **Standardise the data** to help the machine learning algorithm's optimisation process. A snippet of this can be seen in Figure 9.

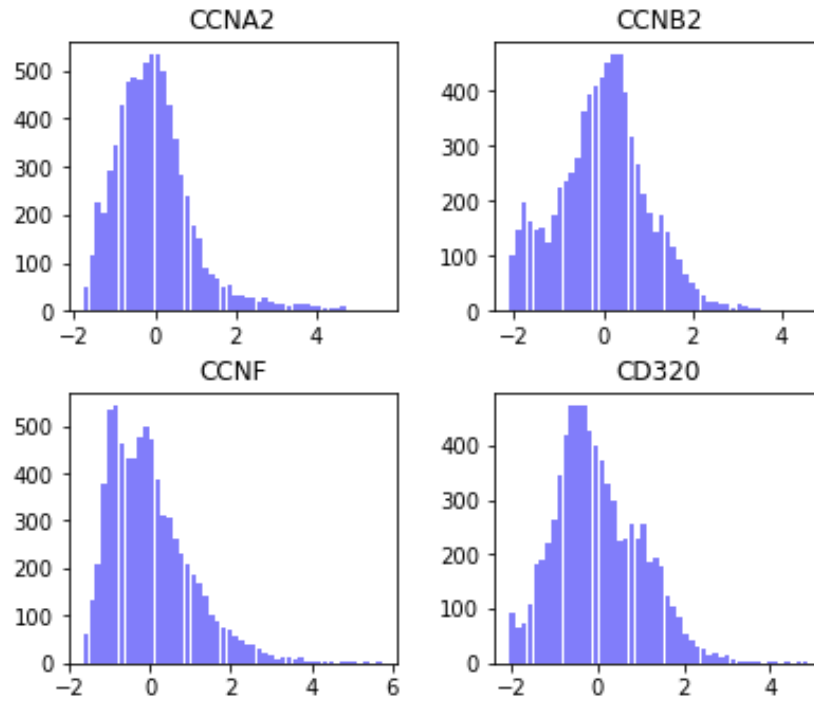


Figure 13: Gene-Expression Distributions.

	Positive Case Proportion
Vascular.disorders	0.148913
Cardiac.disorder.signs.and.symptoms	0.483308
Cardiac.arrhythmias	0.331210
Heart.failures	0.165605
Coronary.artery.disorders	0.292774
Pericardial.disorders	0.023281
Myocardial.disorders	0.032067

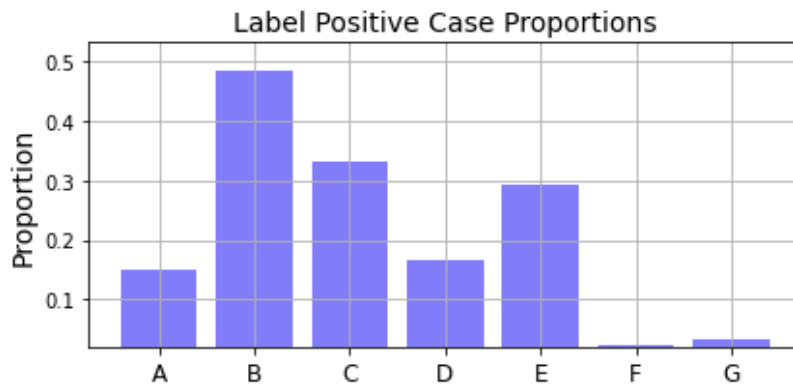


Figure 14: Variation in Proportion of Positive / Total Label Cases.

Input and Output Data Files: Table 2 summarises the data files related to the *Curation and EDA* Notebook. The work completed in the *Curation and Exploratory Data Analysis* Notebook satisfies the objective **OBJ-1**.

File	Type
GSE92742_Broad_LINCS_Level2_GEX_epsilon_n1269922x978.gctx.gz	Input
GSE92742_Broad_LINCS_sig_info.txt.gz	Input
GSE92742_Broad_LINCS_gene_info.txt.gz	Input
m_x_train.txt	Input
m_y_train.txt	Input
m_x_test.txt	Input
m_y_test.txt	Input
train_v2_2.pkl	Output
test_v2_2.pkl	Output
scaler.pkl	Output

Table 2: Input and Output Data Files for Curation and EDA Notebook.

3 Feature Selection Notebook

The main steps in this Notebook are:

1. **Set run mode.** Figure 15 shows the code where it is possible to switch into *Short Demo Mode*, such that the whole process can execute relatively quickly (in about 20 min), as opposed to roughly 1 week for the full experiment run. This is located in the second Notebook cell from the top. Figure 16 shows the section of code, about halfway down the Notebook where the dataset is shortened. This propagates through all Notebooks, via intermediate file sets that are written and read between Notebooks.

Switch between Short Demo Mode OR Full Experiments Mode as in Report

- Short demo can run within 20 min or so.
- The full set of experiments takes about 1 week.

```
1 SHORT_DEMO_ONLY = True # Change to False to run the full set of experiments.
```

Figure 15: Switch for Short Demo Mode.

2. **Reshape Data.** To suit the random forest models, the data needs to be reshaped. Figure 17 shows the code that performs this before the recursive feature elimination (RFE) process. It also shows the dimensions of the short demo datasets.

SHORT DEMO ONLY!!!.... Shorten Training Data for Quick Testing

- Change `SHORT_DEMO_ONLY` (boolean) at top of Notebook to switch between full dataset and experiments.

```
1 if SHORT_DEMO_ONLY:
2     v22_train = v22_train.sample(500, random_state=42)
3     v22_train = v22_train.iloc[:, :50]
4
5     v22_test = v22_test.sample(500, random_state=42)
6     v22_test = v22_test.iloc[:, :50]
```

Figure 16: Code for Shortening the Training Data.

```
1 def split_output(data):
2     y1 = np.array(data.pop('Vascular.disorders'))
3     y2 = np.array(data.pop('Cardiac.disorder.signs.and.symptoms'))
4     y3 = np.array(data.pop('Cardiac.arrythmias'))
5     y4 = np.array(data.pop('Heart.failures'))
6     y5 = np.array(data.pop('Coronary.artery.disorders'))
7     y6 = np.array(data.pop('Pericardial.disorders'))
8     y7 = np.array(data.pop('Myocardial.disorders'))
9
10    return y1, y2, y3, y4, y5, y6, y7
```

```
1 v22_train_y = split_output(v22_train)
2 v22_test_y = split_output(v22_test)
3
4 print(f'Mean of output 1: {np.mean(v22_train_y[0])}')
5 print(f'Mean of output 2: {np.mean(v22_train_y[1])}')
6 print(f'Mean of output 3: {np.mean(v22_train_y[2])}')
7 print(f'Mean of output 4: {np.mean(v22_train_y[3])}')
8 print(f'Mean of output 5: {np.mean(v22_train_y[4])}')
9 print(f'Mean of output 6: {np.mean(v22_train_y[5])}')
10 print(f'Mean of output 7: {np.mean(v22_train_y[6])}')
11
12 print(f'v22_train_y[0] shape: {v22_train_y[0].shape}')
13 print(f'v22_test_y[0] shape: {v22_test_y[0].shape}')
```

```
Mean of output 1: 0.14
Mean of output 2: 0.508
Mean of output 3: 0.332
Mean of output 4: 0.182
Mean of output 5: 0.266
Mean of output 6: 0.018
Mean of output 7: 0.028
v22_train_y[0] shape: (500,)
v22_test_y[0] shape: (500,)
```

```
1 v22_train_x = np.array(v22_train)
2 v22_test_x = np.array(v22_test)
3
4 print(f'v22_train_x shape: {v22_train_x.shape}')
5 print(f'v22_test_x shape: {v22_test_x.shape}')
6
7 display(v22_train_x)
```

```
v22_train_x shape: (500, 43)
v22_test_x shape: (500, 43)
```

Figure 17: Reshaped Data for Random Forest Classifier.

3. **Run RFE for each label.** Recursive feature elimination was performed using the *RFECV()* function from the *scikit-learn* module in tandem with a random forest classifier, within a *Pipeline* object. For the full experiment runs, execution takes about 45 min for each of the 7 labels. For the short demo it takes less than 15 sec per label. Figure 18 and Figure 19 show the code for the pipeline definition and the algorithm execution respectively.

Recursive Feature Elimination (RFE) Pipeline

- Full dataset takes approximately 0.5 hours for each of the 7 labels.
- Takes about 15 seconds when using the shortened dataset for testing purposes only.

```
1 # Classifier for feature selection.
2 clf_featr_sele = RandomForestClassifier(
3     n_estimators=30,
4     random_state=42,
5     class_weight='balanced'
6 )
7
8 # Recursive feature elimination object with cross-validation.
9 rfecv = RFECV(
10     estimator=clf_featr_sele,
11     step=1,
12     cv=5,
13     scoring = 'roc_auc'
14 )
15
16 # Final classifier.
17 clf = RandomForestClassifier(
18     n_estimators=10,
19     random_state=42,
20     class_weight='balanced'
21 )
22
23 # Feature selection pipeline.
24 pipeline = Pipeline([
25     ('feature_sele', rfecv),
26     ('clf', clf)
27 ])
```

Figure 18: Recursive Feature Elimination Definition.

4. **Evaluate metrics and choose feature subset.** The RFE process outputs a set of model artefact files for each label, which can be interrogated to assess the model performance on single random forest classifiers when using the selected features. Since, single label classifiers are used, a ROC curve and AUC score can be obtained, in addition to a confusion matrix and classification report. These are shown in Figure 20 and Figure 21, with the code used to create them in Figure 22. The ROC curve shown corresponds to label *Cardiac.arrythmias*, and is chosen due to having the best AUC, where at least one of the F1 scores was well above 0.5. Another label also has the same AUC score, however both of the F1 scores are close to 0.5.

Perform RFE for each Label

```
1 for OUT_NUM_RFC in range(7):
2     train_X = v22_train_x
3     train_Y = v22_train_y[OUT_NUM_RFC]
4
5     start_time = time()
6     pipeline.fit(train_X, train_Y)
7     end_time = time()
8     time_diff = end_time - start_time
9     print(f'Elapsed time: {time_diff} s.')
10
11     # Save model pipeline.
12     with open(f'{PROJECT_PATH}/pipeline_model_output_{OUT_NUM_RFC}_v22.pkl', 'wb') as p:
13         pickle.dump(pipeline, p, protocol=4)
14
15     # Save
16     with open(f'{PROJECT_PATH}/rfecv_{OUT_NUM_RFC}_v22.pkl', 'wb') as r:
17         pickle.dump(rfecv, r, protocol=4)
```

```
Elapsed time: 14.059834003448486 s.
Elapsed time: 14.295321702957153 s.
Elapsed time: 14.83909559249878 s.
Elapsed time: 12.717202186584473 s.
Elapsed time: 13.019178867340088 s.
Elapsed time: 11.136342763900757 s.
Elapsed time: 11.110288381576538 s.
```

Figure 19: RFE Iterations.

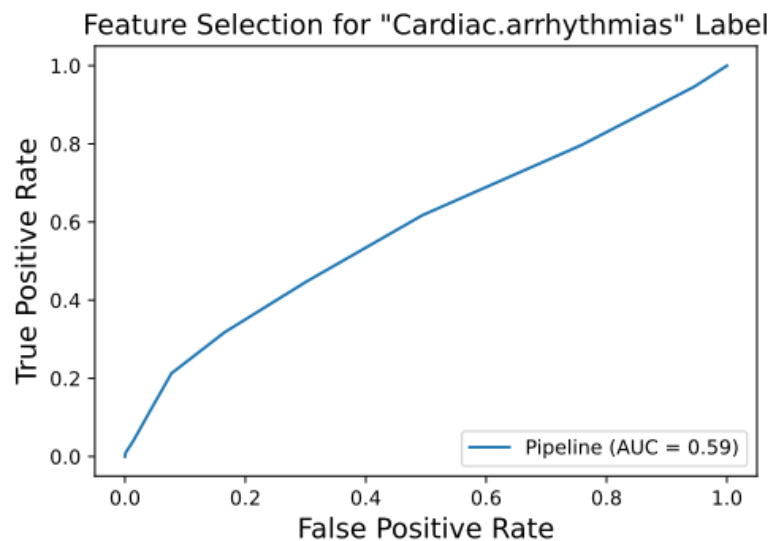


Figure 20: ROC Curve and AUC Score.

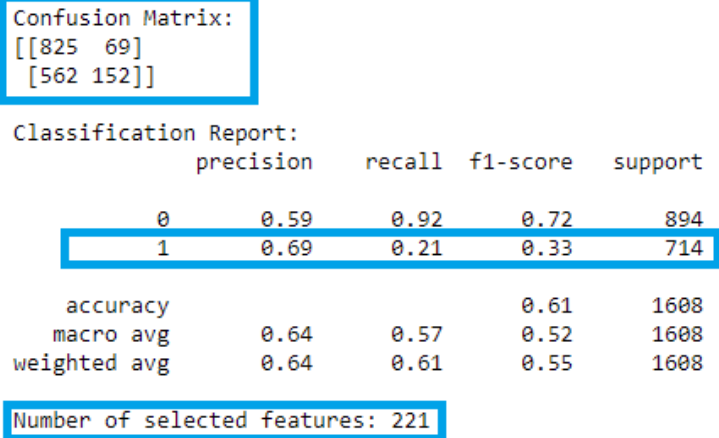


Figure 21: RFE Performance Metrics.

5. **Finalise datasets for subsequent classifier chain experiment runs.** The final dataset is created by simply using the 221 selected features from previously created dataset.

Input and Output Data Files: Table 3 summarises the data files related to the *Feature Selection* Notebook. The work completed in the *Feature Selection* Notebook satisfies the objective **OBJ-2**.

File	Type
train_v2_2.pkl	Input
test_v2_2.pkl	Input
v22_train_x_cc.pkl	Output
v22_train_y_cc.pkl	Output
v22_test_x_cc.pkl	Output
v22_test_y_cc.pkl	Output

Table 3: Input and Output Data Files for Feature Selection Notebook.

Extract RFE Performance Metrics and Best Features

```
1 def get_selected_features(output_num=0, verbose=False, results_path=None):
2     # Load
3     with open(f'{PROJECT_PATH}/pipeline_model_output_{output_num}_v22.pkl', 'rb') as p:
4         pipeline_loaded = pickle.load(p)
5
6     # Load
7     with open(f'{PROJECT_PATH}/rfecv_{output_num}_v22.pkl', 'rb') as r:
8         rfecv_loaded = pickle.load(r)
9
10    train_X = v22_train_x
11    train_Y = v22_train_y[output_num]
12    test_X = v22_test_x
13    test_Y = v22_test_y[output_num]
14
15    # Make predictions for the test set
16    y_pred_test = pipeline_loaded.predict(test_X)
17
18    support_bools = rfecv_loaded.support_
19    support_cols_df = pd.DataFrame({
20        'Feature Name' : v22_train.columns,
21        'Support Bool' : support_bools,
22    })
23
24    selected_features_df = (
25        support_cols_df[support_cols_df['Support Bool']]
26        .reset_index()
27        .rename(columns={'index' : 'Feature Index'})
28    )
29
30    # Optionally print out the performance metrics for the specified label (output).
31    if verbose:
32        print(f'Accuracy: {accuracy_score(test_Y, y_pred_test)}')
33        print(f'\nConfusion Matrix:\n{confusion_matrix(test_Y, y_pred_test)}')
34        print(f'\nClassification Report:\n{classification_report(test_Y, y_pred_test)}')
35        print(f'Number of selected features: {rfecv_loaded.n_features_}')
36        display(selected_features_df[['Feature Index', 'Feature Name']])
37        pipeline_disp = RocCurveDisplay.from_estimator(pipeline_loaded, test_X, test_Y)
38        plt.show()
39
40    return(selected_features_df)
```

Figure 22: Function for Extracting RFE Results and Selected Features.

4 Modelling and Training Notebook

The main steps in this Notebook are:

1. **Set run mode.** In this Notebook there are two constants to adjust based on the run mode (Figure 23), although it is not a binary selection process, as they can be set to any integer value. For short demo mode, the best values are shown in Table 4.

CONSTANTS for Short Demo Only OR Full Set of Experiments as in Report

- Short demo can run within 20 min or so.
- The full set of experiments takes about 1 week.

```
1 # Set to desired interrupt timeout in seconds.
2 # Short Demo: 10
3 # Full set of experiments: 3600
4 TIMEOUT_SEC = 10
5
6 # Limit the number of training experiments.
7 # Short Demo: 10
8 # Full set of experiments: 1000. Need to manually interrupt the process (ctrl-C) when desired experiments reached.
9 NUM_LIMIT = 10
```

Figure 23: Constants for Changing between Run and Full Experiment Modes.

Mode:	Short Demo	Full Experiment
TIMEOUT_SEC	10	3600
NUM_LIMIT	10	1000

Table 4: Recommended Constant Values for each Run Mode.

2. **Create a large hyperparameter search space to be randomly sampled.** As described in (Signorelli; 2022), the hyperparameter space are chosen to be randomly distributed with predefined ranges (Figure 24). The $n_estimators$ values are chosen to match those used by Mamoshina et al. (2020). The remaining hyperparameter ranges, taken from (Koehrsen; 2018), are considered to be a reasonable starting point in the absence of prior knowledge of optimal ranges.

Define Large Set of Hyperparameters

```
1 # Most important hyperparameters according to [1,2]:
2 # - n_estimators
3 # - max_features
4 #
5 # In [2], they use a grid search over:
6 # n_estimators = number of trees in the forest
7 # max_features = max number of features considered for splitting a node
8 # max_depth = max number of levels in each decision tree
9 # min_samples_split = min number of data points placed in a node before the node is split
10 # min_samples_leaf = min number of data points allowed in a leaf node
11 # bootstrap = method for sampling data points (with or without replacement)
12 #
13 # [1]: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
14 # [2]: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn
15
16 bootstrap_vals = [True, False]
17 max_depth_vals = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]
18 max_features_vals = [None, 'sqrt']
19 min_samples_leaf_vals = [1, 2, 4]
20 min_samples_split_vals = [2, 5, 10]
21
22 SAMPLE_LARGE = 1000
23 SEED = 142
24 np.random.seed(seed=SEED)
25
26 bootstrap_array = np.random.choice(bootstrap_vals, SAMPLE_LARGE)
27 max_depth_array = np.random.choice(max_depth_vals, SAMPLE_LARGE)
28 max_features_array = np.random.choice(max_features_vals, SAMPLE_LARGE)
29 min_samples_leaf_array = np.random.choice(min_samples_leaf_vals, SAMPLE_LARGE)
30 min_samples_split_array = np.random.choice(min_samples_split_vals, SAMPLE_LARGE)
31 n_estimators_unique = np.array(list(set([400,150,850,50,800,400,300,100,1050,1050,1400,1050,700,700])))
32 n_estimators_unique.sort()
33 n_estimators_array = np.random.choice(n_estimators_unique, SAMPLE_LARGE)
34
35 hyperparams_large_set = {
36     'bootstrap': bootstrap_array,
37     'max_depth': max_depth_array,
38     'max_features': max_features_array,
39     'min_samples_leaf': min_samples_leaf_array,
40     'min_samples_split': min_samples_split_array,
41     'n_estimators': n_estimators_array
42 }
43
44 # Save dict to pickle.
45 with open(f'{PROJECT_PATH}/hyperparams_large_set.pkl', 'wb') as hp_lrg:
46     pickle.dump(hyperparams_large_set, hp_lrg, protocol=4)
47
48 hyperparams = hyperparams_large_set.copy()
```

Figure 24: Large Hyperparameter Set Generation.

3. **Create the base model for the classifier chains to be trained.** Each classifier chain needs a base model, which in this case is a random forest classifier. Figure 25 shows the function created to instantiate a new base classifier, for each new classifier to be trained. The *test_num* value is the chain ID, which is used to index the hyperparameter arrays.

Function to Create Base Model for Classifier Chain (Random Forest)

- Creates the model, given hyperparameter dictionary, and array index (test number).
- The test numbers are recorded in the file names of the experiment artifact files, e.g. `rfc_chain_100_v22.pkl`.

```
1 def create_rfc_model(hyperparams, test_num):
2     return(
3         RandomForestClassifier(
4             bootstrap = hyperparams['bootstrap'][test_num],
5             max_depth = hyperparams['max_depth'][test_num],
6             max_features = hyperparams['max_features'][test_num],
7             min_samples_leaf = hyperparams['min_samples_leaf'][test_num],
8             min_samples_split = hyperparams['min_samples_split'][test_num],
9             n_estimators = hyperparams['n_estimators'][test_num]
10        )
11    )
```

Figure 25: Base Model Definition.

4. **Wrap the model training process within a timeout interrupt class.** Given the long training time (1 week) of the full set of experiments, a robust approach for completing a sufficiently high number of experiments has been developed. A timeout interrupt framework (thread-based class), adapted from this *stackoverflow* solution⁴, is wrapped around the classifier training steps. The timeout period, defined by the `TIMEOUT_SEC` constant, is set to 1 hour. This means that any training experiments will be aborted after 1 hour if they do not converge. The next model in the sequence is then trained until enough experiments are successfully completed. Figure 26 shows the training steps inside the timeout wrapper.

⁴<https://stackoverflow.com/questions/56315296/provide-a-timeout-in-python-program>

Timeout Class for Skipping Non-Converging Experiments

- Adapted from: <https://stackoverflow.com/questions/58315290/provide-a-timeout-in-python-program>

```
1 # Python program raising exceptions in a python thread.
2
3 chain_order = [
4     'Vascular.disorders',
5     'Cardiac.disorder.signs.and.symptoms',
6     'Cardiac.arrhythmias',
7     'Heart.failures',
8     'Coronary.artery.disorders',
9     'Pericardial.disorders',
10    'Myocardial.disorders'
11 ]
12
13 class thread_with_exception(threading.Thread):
14     def __init__(self, name, rfc_num):
15         threading.Thread.__init__(self)
16         self.name = name
17         self.rfc_num = rfc_num
18
19     def run(self):
20
21         # Target function of the thread class
22         try:
23             print(f'\nFitting RFC chain: {self.rfc_num} ....')
24
25             ##### Training Loop #####
26             start_time = time()
27
28             # Set up RFC models.
29             rfc = create_rfc_model(hyperparams, self.rfc_num)
30
31             # Build RFC models into Classifier Chains.
32             rfc_chain = ClassifierChain(rfc, order=range(len(chain_order)), cv=5)
33
34             # Fit models.
35             rfc_chain.fit(v22_train_x_cc, v22_train_y_cc)
36
37             # Measure training time.
38             end_time = time()
39             time_diff = end_time - start_time
40             ##### Model training #####
41
42             # Save current model chain as backup.
43             rfc_num_str = str(self.rfc_num).zfill(3)
44             print(f'Saving: rfc_chain_{rfc_num_str}_v22.pkl')
45             with open(f'{PROJECT_PATH}/rfc_chain_{rfc_num_str}_v22.pkl', 'wb') as ch:
46                 pickle.dump(rfc_chain, ch, protocol=4)
47
48             ##### End of training Loop #####
49
50         finally:
51             print('Training thread ended.')
52
53     def get_id(self):
54
55         # returns id of the respective thread
56         if hasattr(self, '_thread_id'):
57             return self._thread_id
58         for id, thread in threading._active.items():
59             if thread is self:
60                 return id
61
62     def raise_exception(self):
63         thread_id = self.get_id()
64         res = ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, ctypes.py_object(SystemExit))
65         if res > 1:
66             ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
67             print('Exception raise failure')
```

Figure 26: Timeout Interrupt Class.

5. **Train all individual classifier chains.** The code in Figure 27 shows how the experiment *thread_with_exception* class is iteratively used, with the first few chain output messages displayed. Note that where there is no line saying “Saving: rfc_chain-⟨ID⟩_v22.pkl”, this corresponds to where an experiment is aborted.

Train All Individual Classifier Chains

- Loops through all hyperparameter sets to train each chain, using class defined above.
- For the full sized datasets, the timeout is set to 1 hour. The overall training process took approximately 1 week to run continuously.
- There are 1000 hyperparameter sets, so training was manually halted once 100 experiments successfully ran.
- If the process terminates abnormally, the `START_NUM` just needs to be set to the next test number in the list and the process resumed.
- Each experiment is saved before the next one commences, so no work is lost if there is an unexpected crash, e.g. Windows auto-updates... which did happen once during the week of training.
- Other than that, the process ran continuously without stopping.
- Classifier Chain example: https://scikit-learn.org/stable/auto_examples/multioutput/plot_classifier_chain_yeast.html

```
1 START_NUM = 0      # Change START_NUM only if training halts unexpectedly. Set to next number in sequence.
2
3 for i in range(START_NUM, len(hyperparams['bootstrap']))[:NUM_LIMIT]:
4     t1 = thread_with_exception(f'Thread {i}', i)
5     t1.start()
6     sleep(TIMEOUT_SEC)
7     t1.raise_exception()
8     t1.join()
```

```
Fitting RFC chain: 0 ....
Saving: rfc_chain_000_v22.pkl
Training thread ended.
```

```
Fitting RFC chain: 1 ....
Training thread ended.
```

```
Fitting RFC chain: 2 ....
Training thread ended.
```

```
Fitting RFC chain: 3 ....
Training thread ended.
```

```
Fitting RFC chain: 4 ....
Saving: rfc_chain_004_v22.pkl
Training thread ended.
```

```
Fitting RFC chain: 5 ....
Saving: rfc_chain_005_v22.pkl
Training thread ended.
```

Figure 27: Training each Individual Classifier Chain.

Input and Output Data Files: Table 5 summarises the data files related to the *Modelling and Training* Notebook. The work completed in the *Modelling and Training* Notebook satisfies the objectives **OBJ-3** and **OBJ-4**.

File	Type
v22_train_x_cc.pkl	Input
v22_train_y_cc.pkl	Input
v22_test_x_cc.pkl	Input
v22_test_y_cc.pkl	Input
hyperparams_large_set.pkl	Output
rfc_chain_{ID}_v22.pkl	Output

Table 5: Input and Output Data Files for Modelling and Training Notebook. Note: ID ranges from 0 to the last experiment number.

5 Post-Processing Notebook

1. **Extract the hyperparameters used for each experiment.** During the post-processing ingestion process, the code loads training experiment results into the Python dictionary *rfc_chains*. This facilitates searching for training experiments by name rather than index, ensuring that correct metrics and hyperparameters are linked correctly. Figure 28 shows the code that extracts the hyperparameters from this dictionary. The first set of hyperparameters are shown in the cell output.
2. **Calculate various performance metrics for each experiment.** For comparative purposes, the code calculates various performance metrics (precision, recall, F1 score, MCC, Jaccard score) for the experiment runs, linking them with the experiment hyperparameters. Correlation analysis, and identification of the *Best Means* hyperparameters are carried out from the data, loaded into the *all_results* data frame. Throughout this project, *Pandas* data frames are used, due to their ease of applying the necessary analysis functions, for example *corr()* and *groupby()*. Note that these performance metrics (including Jaccard score) are at the level of single-label classifiers. Figure 29 shows the code used to calculate the performance metrics, and Figure 30 shows the code used to load the data into the data frame. A snippet of the data frame is shown in Figure 31.

Extract Hyperparameters from Saved Model Artifact Files

```
1 def get_used_hyperparams(all_params):
2     h_dict = {}
3     h_dict['n_estimators'] = all_params['base_estimator__n_estimators']
4     h_dict['max_features'] = all_params['base_estimator__max_features']
5     h_dict['max_depth'] = all_params['base_estimator__max_depth']
6     h_dict['bootstrap'] = all_params['base_estimator__bootstrap']
7     h_dict['min_samples_leaf'] = all_params['base_estimator__min_samples_leaf']
8     h_dict['min_samples_split'] = all_params['base_estimator__min_samples_split']
9
10    return(h_dict)
11
12    n_estimators_array = []
13    max_features_array = []
14    max_depth_array = []
15    bootstrap_array = []
16    min_samples_leaf_array = []
17    min_samples_split_array = []
18
19    hyperparams = {}
20
21    # Only Loop through chain numbers where model didn't timeout.
22    for i in good_train_nums:
23        try:
24            chain_params = get_used_hyperparams(rfc_chains[f'RFC Chain {i}'].get_params())
25
26            hyperparams[i] = {
27                'bootstrap' : chain_params['bootstrap'],
28                'max_depth' : chain_params['max_depth'],
29                'max_features' : chain_params['max_features'],
30                'min_samples_leaf' : chain_params['min_samples_leaf'],
31                'min_samples_split' : chain_params['min_samples_split'],
32                'n_estimators' : chain_params['n_estimators']
33            }
34
35        except:
36            pass
37
38    print(f'Number of extracted hyperparameters: {len(hyperparams)}')
39    print('FIRST HYPERPARAMETER SET:')
40    display(hyperparams[0])
```

Number of extracted hyperparameters: 5

FIRST HYPERPARAMETER SET:

```
{'bootstrap': False,
 'max_depth': 60,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 100}
```

Figure 28: Code to Extract Hyperparameters from Trained Chains.

Get Predictions and Calculate the Performance Metrics

```
1 def get_binary_classification_metrics(X_test, Y_true, Y_pred, verbose=True):
2
3     # Accuracy score.
4     acc = accuracy_score(Y_true, Y_pred)
5
6     # Confusion matrix.
7     conf_mat = confusion_matrix(Y_true, Y_pred)
8
9     # Classification report.
10    report = classification_report(Y_true, Y_pred, output_dict=True)
11
12    # MCC
13    mcc = matthews_corrcoef(Y_true, Y_pred)
14
15    # Jaccard score - single label.
16    jacc_score = jaccard_score(Y_true, Y_pred, average='binary')
17
18    metrics = {
19        'Precision [1]' : report['1']['precision'],
20        'Recall [1]' : report['1']['recall'],
21        'F1 [1]' : report['1']['f1-score'],
22        'MCC' : mcc,
23        'Jaccard' : jacc_score
24    }
25
26    if verbose:
27        print(f'Accuracy: {acc}')
28        print(f'\nConfusion Matrix:\n{conf_mat}')
29        print(f'\nClassification Report:\n{classification_report(Y_true, Y_pred)}')
30        print(f'\nMatthews Correlation Coefficient (MCC):\n{mcc}')
31        print(f'\nJaccard Score (single label):\n{jacc_score}')
32
33    return(metrics)
```

Figure 29: Code to Extract SLC Performance Metrics.

Build All Performance Metrics and Hyperparameters into DataFrame for Analysis

```
1 chain_order = [  
2     'Vascular.disorders',  
3     'Cardiac.disorder.signs.and.symptoms',  
4     'Cardiac.arrythmias',  
5     'Heart.failures',  
6     'Coronary.artery.disorders',  
7     'Pericardial.disorders',  
8     'Myocardial.disorders'  
9 ]  
10  
11 all_results = pd.DataFrame()  
12  
13 # for cnum, chain_num in enumerate(good_train_nums):  
14 for chain_num in good_train_nums:  
15  
16     # Get next chain.  
17     rfc_model_chain = rfc_chains[f'RFC Chain {chain_num}']  
18  
19     for model_num in range(len(chain_order)):  
20  
21         print(f'Processing Performance Metrics for (Chain ID, Model ID) ({chain_num},{model_num}):')  
22  
23         # Get predictions for next model in chain.  
24         Y_pred_N = np.array(rfc_model_chain.predict(v22_test_x_cc))[:, model_num]  
25         Y_true_N = 1*v22_test_y_cc[model_num].values  
26  
27         # Metrics for current model.  
28         metrics = get_binary_classification_metrics(v22_test_x_cc, Y_true_N, Y_pred_N, verbose=False)  
29  
30         metrics_df = pd.DataFrame([metrics])  
31         metrics_df.loc[0, 'Chain ID'] = chain_num  
32         metrics_df.loc[0, 'Model ID'] = model_num  
33         metrics_df = metrics_df.set_index(['Chain ID', 'Model ID']).reset_index()  
34  
35         # Hyperparameters  
36         metrics_df.loc[0, 'n_estimators'] = hyperparams[chain_num]['n_estimators']  
37         metrics_df.loc[0, 'max_features'] = hyperparams[chain_num]['max_features'] # In R mtry is static!!!  
38         metrics_df.loc[0, 'max_depth'] = hyperparams[chain_num]['max_depth']  
39         metrics_df.loc[0, 'bootstrap'] = hyperparams[chain_num]['bootstrap']  
40         metrics_df.loc[0, 'min_samples_leaf'] = hyperparams[chain_num]['min_samples_leaf']  
41         metrics_df.loc[0, 'min_samples_split'] = hyperparams[chain_num]['min_samples_split']  
42  
43         # Append results.  
44         all_results = pd.concat([all_results, metrics_df])  
45  
46         # Jaccard score - multilabel.  
47         Y_pred_all = np.array(rfc_model_chain.predict(v22_test_x_cc))  
48         Y_true_all = 1*v22_test_y_cc.values  
49  
50         jacc_score_chain = jaccard_score(Y_true_all, Y_pred_all, average='samples')  
51         print(f'\nJaccard Score (chain):\n{jacc_score_chain}')  
52  
53 print(f'Number of hyperparameter sets: {len(hyperparams)}')  
54 print(f'Last chain number: {chain_num}')  
55 print(f'Number of good chains: {len(good_train_nums)}')
```

Figure 30: Code to Build All Results into Data Frame for Analysis.

All Results DataFrame

```

1 display(all_results.sort_values(['Chain ID', 'Model ID']))
2 display(all_results.info())

```

Chain ID	Model ID	Precision [1]	Recall [1]	F1 [1]	MCC	Jaccard	n_estimators	max_features	max_depth	bootstrap	min_samples_leaf	min_samples_split
0	0	0.229508	0.141414	0.175000	0.029474	0.095890	100	sqrt	60	False	1	5
0	1	0.590909	0.458333	0.516245	-0.066164	0.347932	100	sqrt	60	False	1	5
0	2	0.417391	0.206897	0.276657	-0.051079	0.160535	100	sqrt	60	False	1	5
0	3	0.264151	0.117647	0.162791	0.021145	0.088608	100	sqrt	60	False	1	5
0	4	0.500000	0.229167	0.314286	0.021260	0.186441	100	sqrt	60	False	1	5
...
9	2	0.485294	0.284483	0.358696	0.026100	0.218543	150	sqrt	-999	False	1	10
9	3	0.275862	0.134454	0.180791	0.032206	0.099379	150	sqrt	-999	False	1	10
9	4	0.546154	0.295833	0.383784	0.078488	0.237458	150	sqrt	-999	False	1	10
9	5	0.000000	0.000000	0.000000	-0.032078	0.000000	150	sqrt	-999	False	1	10
9	6	0.000000	0.000000	0.000000	-0.015793	0.000000	150	sqrt	-999	False	1	10

Figure 31: Snippet of All Results Data Frame.

3. **Identify the Best Means hyperparameters.** To identify the *Best Means hyperparameters*, the *all_results* data frame is grouped by each hyperparameter, and the Jaccard score averaged for each group. The hyperparameter value with the largest score is selected as the *Best Means Hyperparameter*. Figure 32 shows the identified *Best Means Hyperparameters* for the full experiment run, and Figure 33 shows the code used to determine them. Figure 34 shows the averaged Jaccard scores for each hyperparameter.

Hyperparameter	
Chain ID	131
Model ID	1
n_estimators	50
max_features	n_features
max_depth	40
bootstrap	False
min_samples_leaf	2
min_samples_split	2

Figure 32: Identified Best Means Hyperparameters.

Grouping of Hyperparameters and Aggregation (mean) of Jaccard Metric

```
1 # Figure properties.
2 fig_col_rot_widths = [
3     ('Chain ID', 45, 15),
4     ('Model ID', 0, 5),
5     ('n_estimators', 0, 5),
6     ('max_features', 0, 5),
7     ('max_depth', 0, 5),
8     ('bootstrap', 0, 5),
9     ('min_samples_leaf', 0, 5),
10    ('min_samples_split', 0, 5)
11 ]
12
13 # List of performance metrics.
14 metrics = [
15     'Precision [1]',
16     'Recall [1]',
17     'F1 [1]',
18     'MCC',
19     'Jaccard'
20 ]
21
22 best_feature_values = {}
23 mean_metrics = {}
24
25 # Loop through hyperparameters and plot sorted performance metrics.
26 for c_r_w in fig_col_rot_widths:
27     col = c_r_w[0]
28     rot = c_r_w[1]
29     width = c_r_w[2]
30
31     grp_mean_metrics = all_results.groupby(col)[metrics].mean()
32     mean_metrics[col] = grp_mean_metrics
33     grp_mean_metrics_sorted = grp_mean_metrics.sort_values('Jaccard', ascending=False).reset_index()
34     best_feature_values[col] = grp_mean_metrics_sorted.loc[0, col]
35     display(grp_mean_metrics_sorted)
36
37     x_vals = grp_mean_metrics_sorted.index
38     y_vals = grp_mean_metrics_sorted['Jaccard'].values
39
40     x_labels = grp_mean_metrics_sorted[grp_mean_metrics_sorted.columns[0]]
41     y_label = 'Mean Score'
42     title = f'Mean Jaccard Score across {col}'
43
44     bar_plot(x_vals, y_vals, x_labels, title, col, rot, width)
45
46 if best_feature_values['max_features'] == 'n_features':
47     best_feature_values['max_features'] = None
48
```

Figure 33: Code to Group and get Aggregate Performance Scores.

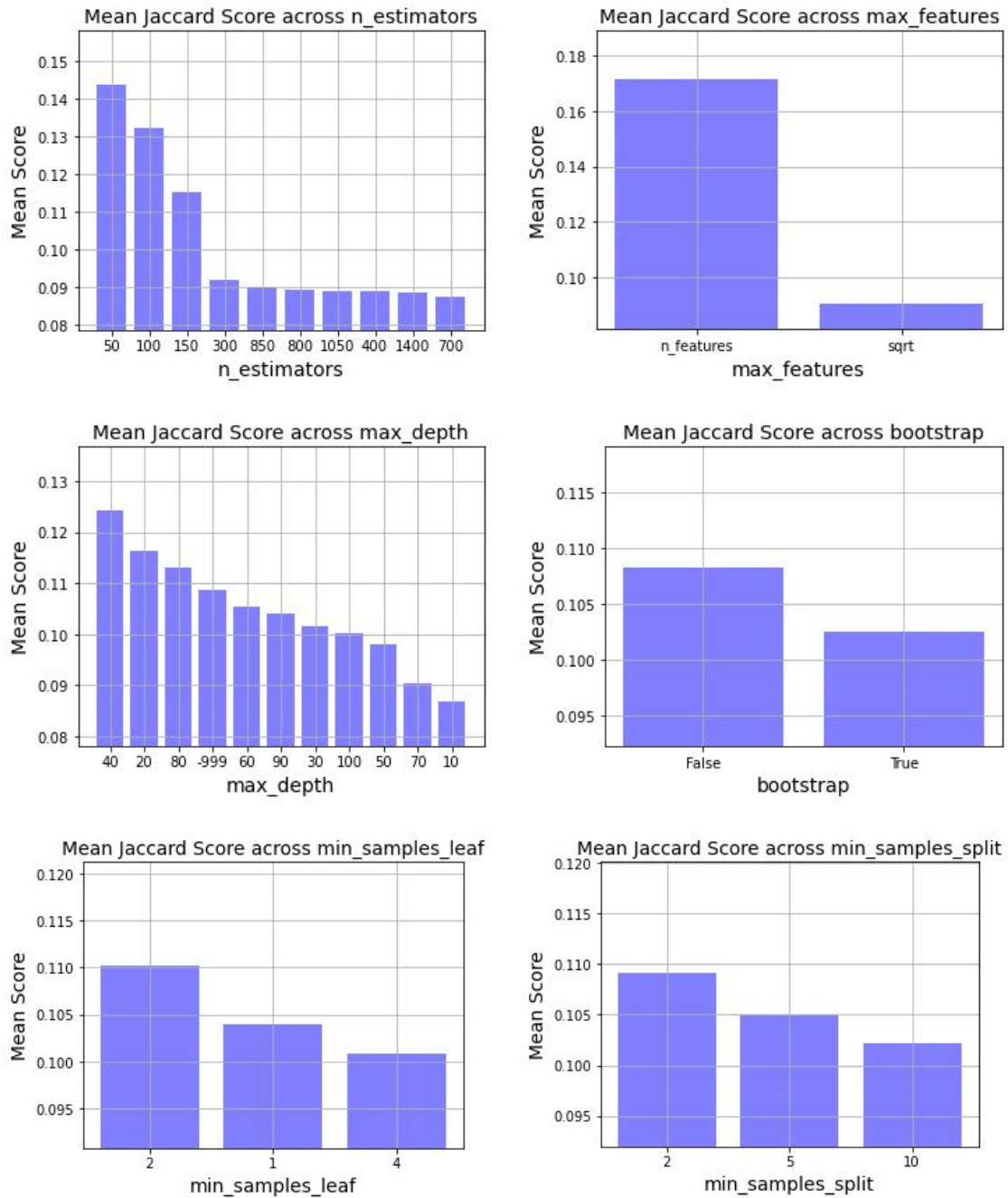


Figure 34: Grouped Hyperparameter Jaccard Scores (Full Set of Experiments).

4. **Train a new single classifier chain.** Using the *Best Means hyperparameters*, the final *Best Means Chain* is trained. Figure 35 shows the code for this. Note that this classifier chain is not wrapped within an interrupt class, as it is only one chain. Also, the `-999` constant is used to represent *auto* and default values for *max_depth* and *min_samples_split* respectively.

Fit New RFC with Best Means Hyperparameters

```
1 # Cleaning.
2 if best_feature_values['max_depth'] == -999:
3     best_feature_values['max_depth'] = None
4
5 # Cleaning.
6 if best_feature_values['min_samples_split'] == -999:
7     best_feature_values['min_samples_split'] = 2
8
9 # Set up base RFC with Best Means Hyperparameters.
10 rfc_best_mean = RandomForestClassifier(
11     n_estimators = best_feature_values['n_estimators'],
12     max_features = best_feature_values['max_features'],
13     max_depth = best_feature_values['max_depth'],
14     bootstrap = best_feature_values['bootstrap'],
15     min_samples_leaf = best_feature_values['min_samples_leaf'],
16     min_samples_split = best_feature_values['min_samples_split']
17 )
18
19 # Build RFC models into Classifier Chain.
20 rfc_best_mean_chain = ClassifierChain(rfc_best_mean, order=range(len(chain_order)), cv=5)
21
22 # Train new RFC model (Best Means Chain).
23 start_time_bmch = time()
24 rfc_best_mean_chain.fit(v22_train_x_cc, v22_train_y_cc)
25 end_time_bmch = time()
26 print(f'Training time for Best Means Chain: {end_time_bmch - start_time_bmch} s.')
```

Figure 35: Code to Train the Best Means Chain.

5. **Create the Ensemble chain and calculate all chain scores.** Adopting a similar approach to this example⁵, the *Ensemble* chain is created from the 100 individual chains, using the code in Figure 36. The code also calculates, and stores the multi-label Jaccard scores for all chains into the *model_scores* data frame.
6. **Compare all chain performances.** The ranked chain performances for the individual chains, *Ensemble* chain, and *Best Means Chain* are shown in Figure 37 (full set of experiments), and Figure 38 (short demo runs). It can be seen how the *Best Means Chain* performs relatively well, even for the short demo run, demonstrating the robustness of the *Best Means* approach. Figure 39 shows the code used to produce the plots.

⁵https://www.typeerror.org/docs/scikit_learn/auto_examples/multioutput/plot_classifier_chain_yeast

Comparison of each Chain with Ensemble

- Note: the Ensemble implements direct-voting of all chain predictions.
- Code in this cell was adapted from: https://www.typeerror.org/docs/scikit_learn/auto_examples/multioutput/plot_classifier_chain_yeast

```
1 # RFC chains.
2 Y_pred_chains = np.array([chain.predict(v22_test_x_cc) for chain in list(rfc_chains.values())])
3
4 chain_jaccard_scores = [jaccard_score(v22_test_y_cc, Y_pred_chain >= .5, average='samples')
5                         for Y_pred_chain in Y_pred_chains]
6
7 # Ensemble chain.
8 Y_pred_ensemble = Y_pred_chains.mean(axis=0)
9 ensemble_jaccard_score = jaccard_score(v22_test_y_cc, Y_pred_ensemble >= .5, average='samples')
10
11 # Best mean chain.
12 Y_pred_best_mean = rfc_best_mean_chain.predict(v22_test_x_cc)
13 best_mean_jaccard_score = jaccard_score(v22_test_y_cc, Y_pred_best_mean >= .5, average='samples')
14
15 model_scores = chain_jaccard_scores
16 model_scores.append(ensemble_jaccard_score)
17 model_scores.append(best_mean_jaccard_score)
18
19 model_names = [f'{i}' for i in good_train_nums] + ['Ensemble', 'B.M. Chain']
```

Figure 36: Code to Calculate Ensemble Chain and Build Comparison Data Frame.

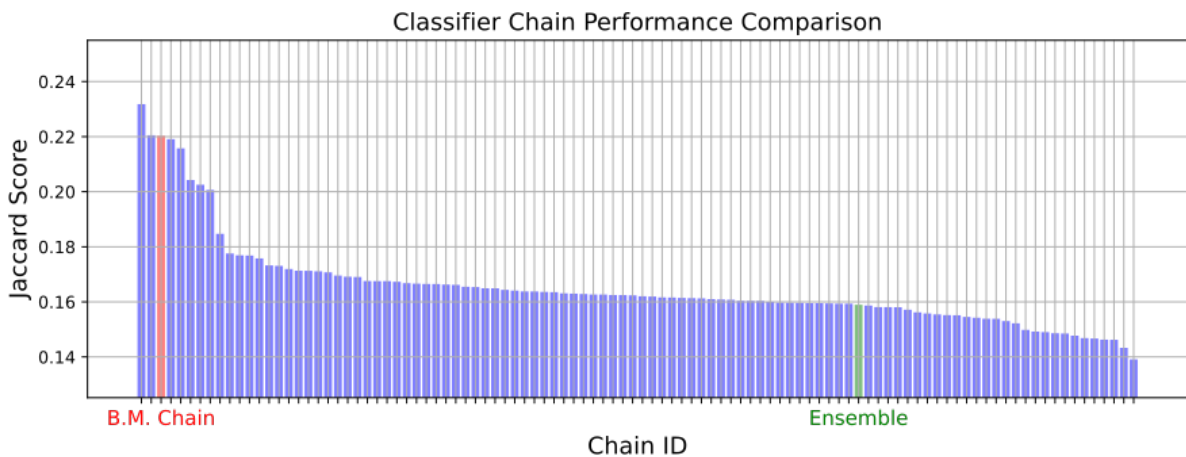


Figure 37: Chain Rankings (Full Experiment Set).

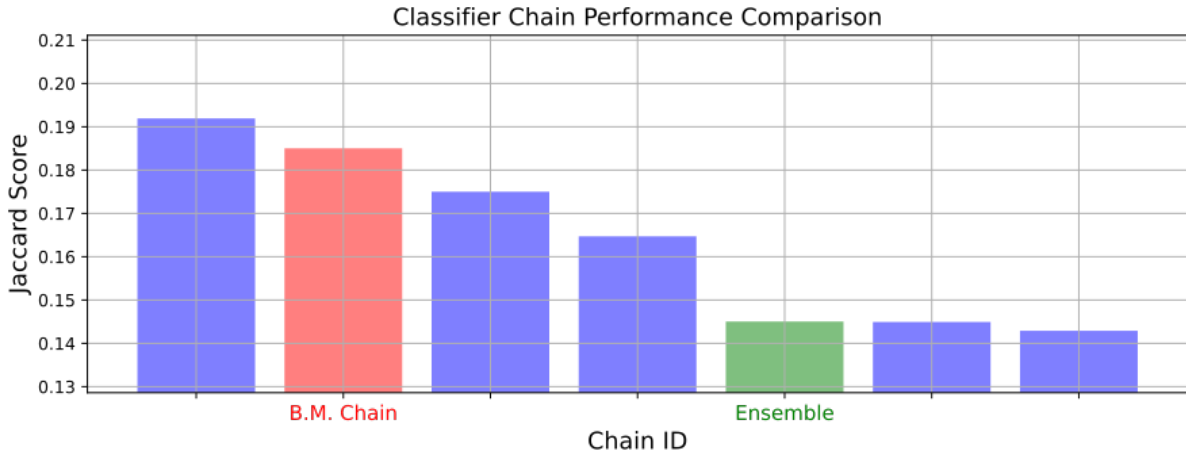


Figure 38: Chain Rankings (Short Demo).

7. **Identify correlations between the various performance metrics.** Correlation coefficients are calculated directly from the *all_results* data frame using the *Pandas corr()* function. The results are shown in (Signorelli; 2022).
8. **Analyse the execution times of the experiment runs.** Experiment computation times (training + file saving) are obtained, using the *glob* module to interrogate the file statistics. Capturing the combined training and file write times is preferred over just the training time, since the combination significantly affects the total experiment run time for all chains. Note that the file sizes for each experiment vary dramatically. The computation times are calculated from the time differences between each of the individual chain timestamps, and file sizes are also captured. Using the chain ID numbers in the model artefact file names, the Jaccard scores are joined to obtain correlation coefficients. From these results, a diminishing return curve is plotted (Figure 40), with the results presented in (Signorelli; 2022). Figure 41 shows the code.

Plot All Comparative Chain Performances

```

1 model_scores_df_bck = model_scores_df.copy()
2
3 # Abbreviate Best Means Chain for clarity.
4 model_scores_df['Model Name'] = model_scores_df['Model Name'].str.replace('Best Means', 'B.M.')
5
6 # Remove xtick labels for clarity.
7 mask_indiv_names = (
8     (model_scores_df['Model Name'] != 'B.M. Chain') &
9     (model_scores_df['Model Name'] != 'Ensemble')
10 )
11 model_scores_df.loc[mask_indiv_names, 'Model Name'] = ''
12
13 # Get position for Best Means Chain and Ensemble colour.
14 mask_x_pos_ensmb = model_scores_df['Model Name'] == 'Ensemble'
15 mask_x_pos_bm_chain = model_scores_df['Model Name'] == 'B.M. Chain'
16
17 x_pos_ensmb = int(model_scores_df.loc[mask_x_pos_ensmb, 'X Pos'])
18 x_pos_bm_chain = int(model_scores_df.loc[mask_x_pos_bm_chain, 'X Pos'])
19
20 len_scores = len(model_scores_df)
21
22 # Plot the Jaccard similarity scores for the each of the chains, and the ensemble.
23 fig, ax = plt.subplots(figsize=(10, 4))
24 ax.grid(True)
25 ax.set_title('Classifier Chain Performance Comparison', fontsize=FONT_SIZE)
26 ax.set_xticks(np.array(model_scores_df['X Pos']))
27 ax.set_xticklabels(model_scores_df['Model Name'], rotation=0)
28 ax.set_xlabel('Chain ID', fontsize=FONT_SIZE)
29 ax.set_ylabel('Jaccard Score', fontsize=FONT_SIZE)
30 ax.set_ylim((min(model_scores_df['Jaccard']) * .9, max(model_scores_df['Jaccard']) * 1.1))
31 colors = ['b']*x_pos_bm_chain + ['r'] + ['b']*(x_pos_ensmb-x_pos_bm_chain-1) + ['g'] + ['b']*(len_scores-x_pos_ensmb-1)
32 ax.bar(model_scores_df['X Pos'], model_scores_df['Jaccard'], alpha=0.5, color=colors)
33
34 # Make xtick labels bigger.
35 for tick in ax.xaxis.get_major_ticks():
36     tick.label.set_fontsize(FONT_SIZE-2)
37
38 # Set label colours.
39 labels = [item.get_text() for item in ax.get_xticklabels()]
40 for t, tick_label in enumerate(ax.xaxis.get_ticklabels()):
41     if labels[t] == 'B.M. Chain':
42         tick_label.set_color('red')
43     elif labels[t] == 'Ensemble':
44         tick_label.set_color('green')
45
46 plt.tight_layout()
47 plt.show()
48 fig.savefig(f'{PROJECT_PATH}/chain_rankings.svg', bbox_inches='tight')

```

Figure 39: Code to Plot Best Means, Ensemble and Individual Chain Performances.

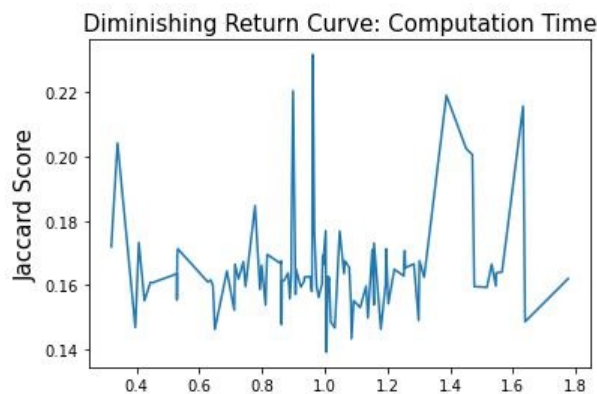


Figure 40: Diminishing Return Curve for Computation Time.

Execution Time and File Size Analysis

```
1 # Get list of all model artifact files.
2 saved_rfc_chains = glob.glob(f'{PROJECT_PATH}/rfc_chain_*')[ :NUM_TO_LOAD]
3 fnames = [f.split('\\')[-1] for f in saved_rfc_chains]
4
5 timestamps = []
6 sizes = []
7
8 # Extract the file properties from the directory listing.
9 for file in saved_rfc_chains:
10     fname = pathlib.Path(file)
11     assert fname.exists(), f'No such file: {fname}' # check that the file exists
12
13     mtime = datetime.datetime.fromtimestamp(fname.stat().st_mtime)
14     size = fname.stat().st_size
15     timestamps.append(mtime)
16     sizes.append(size)
17
18 # Build info into DataFrame.
19 ts_df = pd.DataFrame({
20     'File' : fnames,
21     'Timestamp' : timestamps,
22 }).sort_values('Timestamp')
23
24 # Calculate relative times between file-modified times.
25 ts_df['T-prev'] = pd.NaT
26 ts_df.iloc[1:, 2] = ts_df.iloc[:-1, 1]
27 ts_df['T-diff'] = ts_df['Timestamp'] - ts_df['T-prev']
28 ts_df['Computation Time (hr)'] = ts_df['T-diff'] / np.timedelta64(1, 'h')
29
30 # Calculate file sizes in MB.
31 ts_df['Size (MB)'] = np.array(sizes)/1e+6
32 ts_df['Size (MB)'] = ts_df['Size (MB)'].round(0).astype(int)
33
34 # Extract chain number from file name.
35 ts_df['Chain Number'] = ts_df['File'].str.extract(r'_(\d+)_').astype(int)
36
37 # Get difference between chain numbers.
38 ts_df['Chain Number Prev'] = pd.NaT
39 ts_df.iloc[1:, 7] = ts_df.iloc[:-1, 6]
40 ts_df['Chain Number Diff'] = ts_df['Chain Number'] - ts_df['Chain Number Prev']
41 ts_df['Computation Time (hr)'] = ts_df['Computation Time (hr)'] / ts_df['Chain Number Diff']
42
43 # Impute training time with mean training time after correction.
44 ts_df['Computation Time (hr)'] = ts_df['Computation Time (hr)'].fillna(ts_df['Computation Time (hr)'].mean())
45
46 # Clean up.
47 ts_df.drop(['T-diff', 'T-prev', 'Chain Number', 'Chain Number Prev', 'Chain Number Diff'], axis=1, inplace=True)
48
49 display(ts_df)
50
51 # Remove any negative time durations (shouldn't exist) from box plot calculation.
52 mask_positive = ts_df['Computation Time (hr)'] > 0
53 fig, ax = plt.subplots(figsize=(6, 4))
54 ts_df[mask_positive].boxplot(column=['Computation Time (hr)'], ax=ax)
55 x = np.random.normal(1, 0.04, size=len(ts_df[mask_positive]))
56 plt.plot(x, ts_df.loc[mask_positive, 'Computation Time (hr)'], 'r.', alpha=0.2)
57 fig.savefig(f'{PROJECT_PATH}/computation_boxplot.jpg', bbox_inches='tight')
```

Figure 41: Code to Extract RFC Chain Artefact File Statistics.

9. **Create diminishing return curve for Ensemble size.** Figure 42 shows the diminishing return curve for varying ensemble sizes, that is, where varying numbers of individual chains are used in its calculation. Figure 43 shows the code used to calculate each ensemble run.

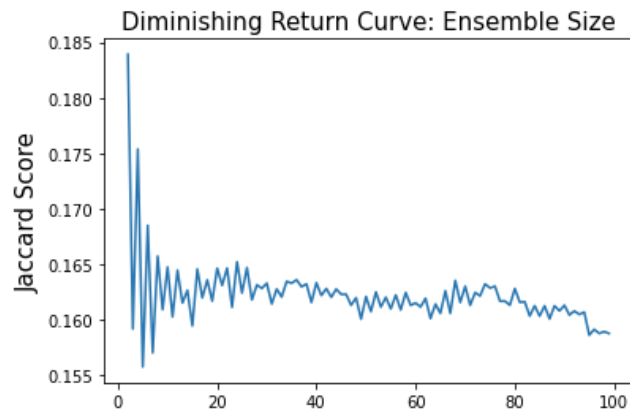


Figure 42: Diminishing Return Curve for Ensemble Size.

Diminishing Return Curve for Performance vs. Ensemble Size

```
1 ensmb_sizes = range(2, len(rfc_chains))
2 ensmb_scores = []
3
4 # Iterate through Ensemble sizes from 2 to the number of individual chains... takes a Long time!
5 for e in ensmb_sizes:
6     print(f'Ensemble size: {e}')
7     # Get the chain predictions for each chain, up to a limit of the current max ensemble size.
8     Y_pred_chains = np.array([chain.predict(v22_test_x_cc) for chain in list(rfc_chains.values())[:e]])
9
10    chain_jaccard_scores = [jaccard_score(v22_test_y_cc, Y_pred_chain >= .5, average='samples')
11                           for Y_pred_chain in Y_pred_chains]
12
13    # Ensemble chain.
14    Y_pred_ensemble = Y_pred_chains.mean(axis=0)
15    ensemble_jaccard_score = jaccard_score(v22_test_y_cc, Y_pred_ensemble >= .5, average='samples')
16
17    ensmb_scores.append(ensemble_jaccard_score)
```

Figure 43: Code to Calculate Performance for Different Ensemble Chain Sizes.

Input and Output Data Files: Table 6 summarises the data files related to the *Post-Processing* Notebook. The work completed in the *Post-Processing* Notebook satisfies the objectives **OBJ-5** and **OBJ-6**.

File	Type
v22_train_x_cc.pkl	Input
v22_train_y_cc.pkl	Input
v22_test_x_cc.pkl	Input
v22_test_y_cc.pkl	Input
rfc_chain_{ID}_v22.pkl	Input
all_results.pkl	Output
rfc_best_mean_chain.pkl	Output

Table 6: Input and Output Data Files for Post-Processing Notebook. Note: ID ranges from 0 to the last experiment number.

Acknowledgements

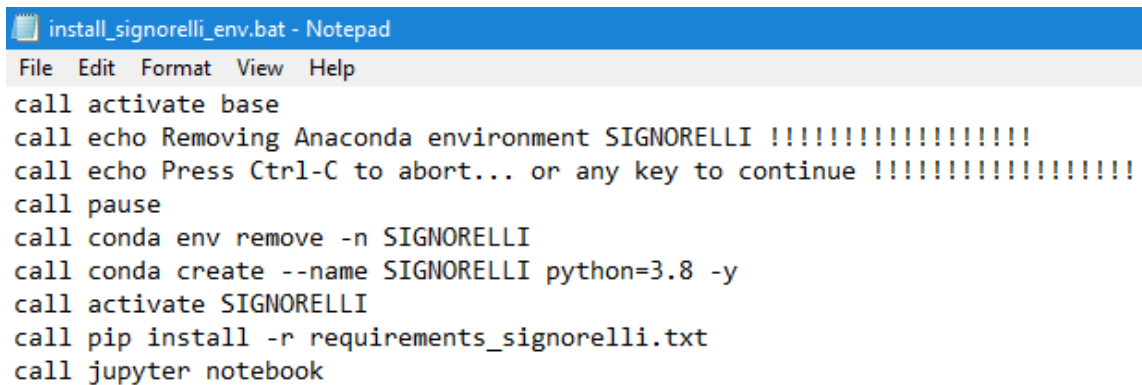
Deep gratitude is expressed to Dr. Vladimir Milosavljevic whose supervision was clear, concise, and very helpful in maintaining an efficient research path. The same gratitude is expressed to the authors of Mamoshina et al. (2020) for providing the ‘Mamoshina’ data referred to in this configuration manual and (Signorelli; 2022). This project could not have been achieved in its current form without that data. Dr. Polina Mamoshina was very helpful with her email correspondence.

References

- Koehrsen, W. (2018). Hyperparameter tuning the random forest in python.
URL: <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74>
- Mamoshina, P., Bueno-Orovio, A. and Rodriguez, B. (2020). Dual transcriptomic and molecular machine learning predicts all major clinical forms of drug cardiotoxicity, *Frontiers in Pharmacology* **11**.
- Signorelli, C. (2022). *Sub-optimal hyperparameter selection for multi-label classifier chains predicting cardiotoxicity from gene-expression data*, Master’s thesis, NCI, Dublin.

Appendix

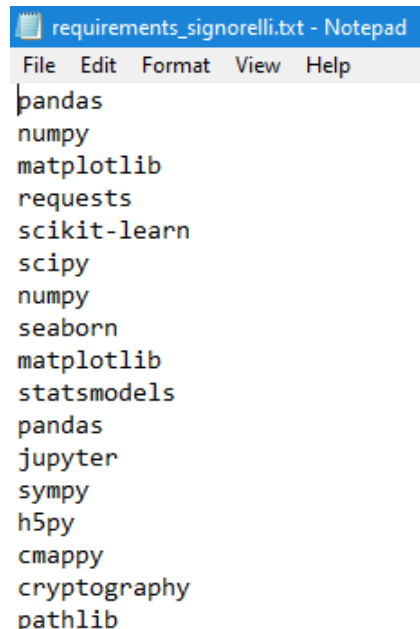
Installation Batch Script: Figure 44 shows the commands in the installation script (*install_signorelli_env.bat*) to run after the project artefacts .zip file has been extracted. The script sets up a new anaconda environment called SIGNORELLI, installs all Python dependencies, then opens up Jupyter Notebook, ready to open and run the project code.

A screenshot of a Notepad window titled "install_signorelli_env.bat - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content of the batch script is as follows:

```
call activate base
call echo Removing Anaconda environment SIGNORELLI !!!!!!!!!!!!!!!!!!!!!!!
call echo Press Ctrl-C to abort... or any key to continue !!!!!!!!!!!!!!!!!!!!!!!
call pause
call conda env remove -n SIGNORELLI
call conda create --name SIGNORELLI python=3.8 -y
call activate SIGNORELLI
call pip install -r requirements_signorelli.txt
call jupyter notebook
```

Figure 44: Installation Batch Script.

Anaconda Requirements.txt File: Figure 45 shows the requirements file (*requirements_signorelli.txt*) that the installation script uses when setting up the SIGNORELLI anaconda environment.

A screenshot of a Notepad window titled "requirements_signorelli.txt - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content of the requirements file is as follows:

```
pandas
numpy
matplotlib
requests
scikit-learn
scipy
numpy
seaborn
matplotlib
statsmodels
pandas
jupyter
sympy
h5py
cmappy
cryptography
pathlib
```

Figure 45: Requirements.txt File.