BMEG 3105 Project Report

Diagnosis of Thyroid Disease from Blood Work Results

Chan Cheuk Ka 1155174356

Table of Contents

1 Problem Statement

460 million people worldwide suffer from thyroid diseases, and this prevalence is higher in poorer countries and increases with age [1]. Thyroid diseases can manifest a wide variety of symptoms, and these symptoms often overlap with mundane problems like sleep deprivation [2]. Even abnormalities in blood test results can be easily obscured by other mundane factors [3, 4]. These reasons lead to a high probability of thyroid diseases being misdiagnosed or completely missed.

This report investigates using an AI model to diagnose thyroid diseases with blood test results. The comprehensive account of dataset CSVs, source code, terminal log, and heatmap output images can be accessed via this link.

2 Methodology

2.1 Data Acquisition

The original blood test dataset is from the UCI Machine Learning Repository [5], but a version reconciling everything into one file was used [6]. The dataset is a 9172×31 matrix with columns on the ID, age, sex, surgical history, medication, blood marker levels, and medical diagnoses of the patients. Refer to *Table 2* below for a comprehensive list of the columns and their functions.

2.2 Data Sanitisation

As can be observed in *Table 2* below, there are multiple irrelevant columns of data and many nullable data entries. This called for the need to first sanitise the data before using it.

Below is the code for data sanitisation — excerpt from `processing.py`.

```
# remove unrelated coloums
dataset = dataset.drop(columns=["patient_id"])
dataset = dataset.drop(columns=["referral_source"])

# change the column "sex" from "M" and "F" to 0 and 1
dataset["sex"] = dataset["sex"].map({"M": 0, "F": 1})
# change other columns from "f" and "t" to 0 and 1
dataset = dataset.replace({"f": 0, "t": 1})

# fill outlier age with average
MAX_ALLOWED_AGE = 100
average_age = dataset[dataset["age"] ≤ MAX_ALLOWED_AGE]["age"].mean()
dataset.loc[dataset["age"] > MAX_ALLOWED_AGE, "age"] = average_age

# fill missing values
dataset["sex"] = dataset["sex"].fillna(0.5)
dataset = dataset.fillna(0)
```

2.2.1 Column Purging

Since we are concerning the diagnosis of thyroid diseases from blood samples, only the data immediately surrounding the disease are relevant, and data such as patient ID and patient referral sources are purged.

Below is the code for column purging — excerpt from `processing.py`.

```
6  # remove unrelated coloums
7  dataset = dataset.drop(columns=["patient_id"])
8  dataset = dataset.drop(columns=["referral_source"])
```

2.2.2 Data Conformation

As can be observed in *Table 2* below, some of the data columns are presented with Boolean values and strings. To modify it so that the dataset can be used to train a model, such data is conformed to numeric values. For example, the “M” and “F” values for sex are conformed into “0”s and “1”s, and the “t” and “f” Boolean values are conformed into “1”s and “0”s.

Below is the code for data conformation — excerpt from `processing.py`.

```
10  # change the column "sex" from "M" and "F" to 0 and 1
11  dataset["sex"] = dataset["sex"].map({"M": 0, "F": 1})
12  # change other columns from "f" and "t" to 0 and 1
13  dataset = dataset.replace({"f": 0, "t": 1})
```

2.2.3 Outlier Substitution

There are some outlier values of 455, 65511, 65512, and 65526 for the age. These values were replaced with the average age of the remaining samples.

Below is the code for outlier substitution — excerpt from `processing.py`.

```
15  # fill outlier age with average
16  MAX_ALLOWED_AGE = 100
17  average_age = dataset[dataset["age"] ≤ MAX_ALLOWED_AGE]["age"].mean()
18  dataset.loc[dataset["age"] > MAX_ALLOWED_AGE, "age"] = average_age
```

2.2.4 Void Filling

As can be observed in *Table 2* below, some data entries are undefined. There are two cases of this behaviour.

> Below is the code for void filling — excerpt from `processing.py`.

```
20  # fill missing values
21  dataset["sex"] = dataset["sex"].fillna(0.5)
22  dataset = dataset.fillna(0)
```

2.2.4.1 Empty Sex Values

There are entries for sex where it is simply empty. To fill the voids, values of 0.5 were entered as neutral values.

> Below is the code for filling sex value voids — excerpt from `processing.py`.

```
21  dataset["sex"] = dataset["sex"].fillna(0.5)
```

2.2.4.2 Unmeasured Blood Marker Levels

As there are columns indicating whether each blood marker level was measured, the unmeasured blood marker levels lacked values. For this case of voids, zeros are filled in. Since some columns indicate whether the markers were measured, the zeros should not be ambiguous.

> Below is the code for filling blood marker level voids — excerpt from `processing.py`.

```
22  dataset = dataset.fillna(0)
```

2.3 Multi-Label Binarisation

In the dataset, each diagnosis is represented by a character, as detailed in *Table 3* below. Each patient can be diagnosed with more than one medical diagnosis. Thus, the dataset is multi-label. Multi-label binarisation is used to turn this into a usable format, where each diagnosis is given its separate column, similar to one-hot encoding. However, they are not mutually exclusive,

unlike one-hot encoding. *Table 1* below demonstrates the restructuring from raw representation to finalised representation. A and B are used here as arbitrary examples.

Table 1: Diagnosis multi-label binarisation reference.

Initial Representation	Interpretation	Binarised Value of A	Binarised Value of B
`A`	A is diagnosed	`1`	`0`
`AB`	Both A and B are diagnosed	`1`	`1`
`A|B`	A is the more likely diagnosis, but B is also possible	`1`	`1`

Notice that while binarisation requires the output of a binary value, the actual dataset actually includes a representation for uncertain diagnoses. However, for the sake of simplicity and binarisation, both likely diagnoses are treated as certain diagnoses.

Below is the code for multi-label binarisation — excerpt from `processing.py`.

```python
# change the column "target" to use multi-label binarisation
def confidence_encoding(target):
    encoding = {}
    if "|" in target:
        parts = target.split("|")
        encoding[parts[0]] = 1
        encoding[parts[1]] = 1
        # encoding[parts[0]] = 2/3
        # encoding[parts[1]] = 1/3
    else:
        for char in target:
            encoding[char] = 1
            # encoding[char] = 1.0
    return encoding
encoded_targets = dataset["target"].apply(confidence_encoding)
encoded_df = pd.DataFrame(encoded_targets.tolist()).fillna(0)
dataset = pd.concat([dataset.drop(columns=["target"]), encoded_df], axis=1)
```

2.4 Data Augmentation

Some diagnoses are inherently rarer than others, and this can lead to a bias in the model output prediction. To mitigate this problem, sample data with rarer diagnoses is augmented via duplication to reach the average sample count.

Below is the code for data augmentation — excerpt from `processing.py`.

```python
# augment the dataset such that every target class has the same number of samples
# by duplicating the samples with less than the average number of samples
classes = ['-', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S']
# get the samples with the class == 1 into a separate dataframe
class_samples = {}
print("Before augmentation:")
for class_name in classes:
    class_samples[class_name] = dataset[dataset[class_name] == 1]
    count = len(class_samples[class_name])
    percent = round(count/len(dataset)*100, 2)
    print(f"Class {class_name} has {count} ({percent}%) samples")
# get the average number of samples
average_samples = sum([len(class_samples[class_name]) for class_name in classes]) / len(classes)
# duplicate the samples with less than the average number of samples
for class_name in classes:
    while len(class_samples[class_name]) < average_samples:
        class_samples[class_name] = class_samples[class_name]._append(class_samples[class_name])
# concatenate the samples back into the dataset
dataset = pd.concat(class_samples.values())
#
print("After augmentation:")
for class_name in classes:
    count = len(class_samples[class_name])
    percent = round(count/len(dataset)*100, 2)
    print(f"Class {class_name} has {count} ({percent}%) samples")
```

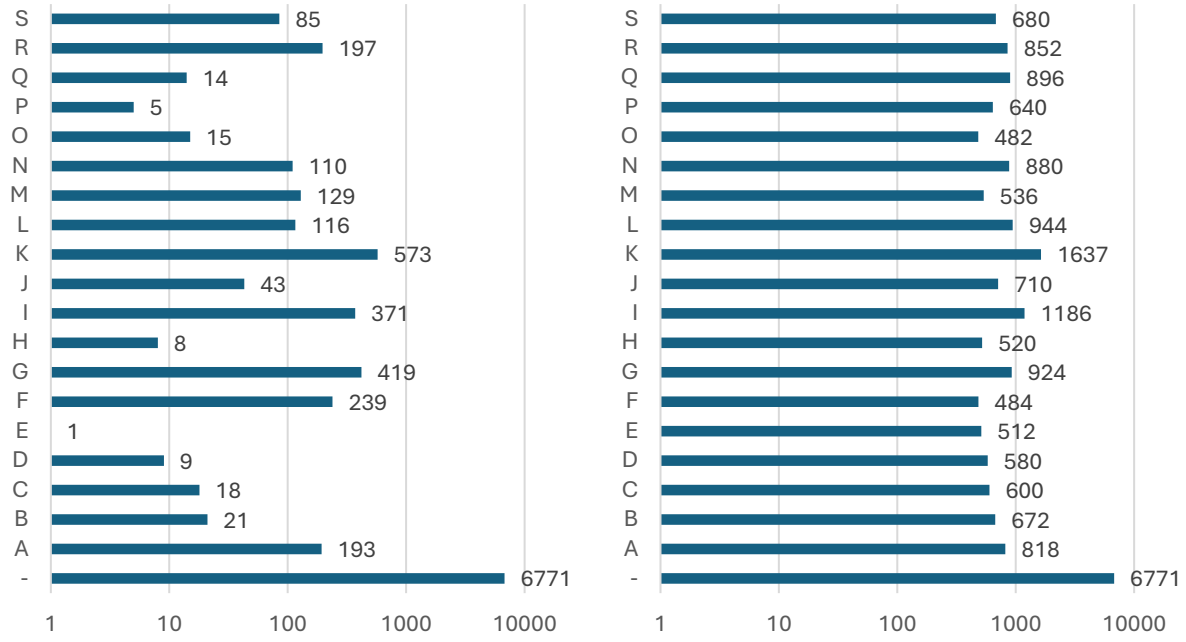

Figure 1: Diagnosis sample count before (left) and after (right) data augmentation.

As can be observed in *Figure 1* above, the data augmentation helped immensely with ensuring appropriate representation of each diagnosis type. The data augmentation increased the dataset sample count from 9172 to 19518, which equates to a 113% increase. Note that the diagnosis "T" was not shown since it has zero entries in the dataset despite being in the key. There are a total of 20 different valid diagnoses, excluding "T".

2.5 Model Architecture

2.5.1 Random Forest Ensemble

A random forest ensemble with 100 decision trees was chosen as the model for classification. Each decision tree would generate a prediction, and the ensemble would congregate the decisions and generate a result via majority voting.

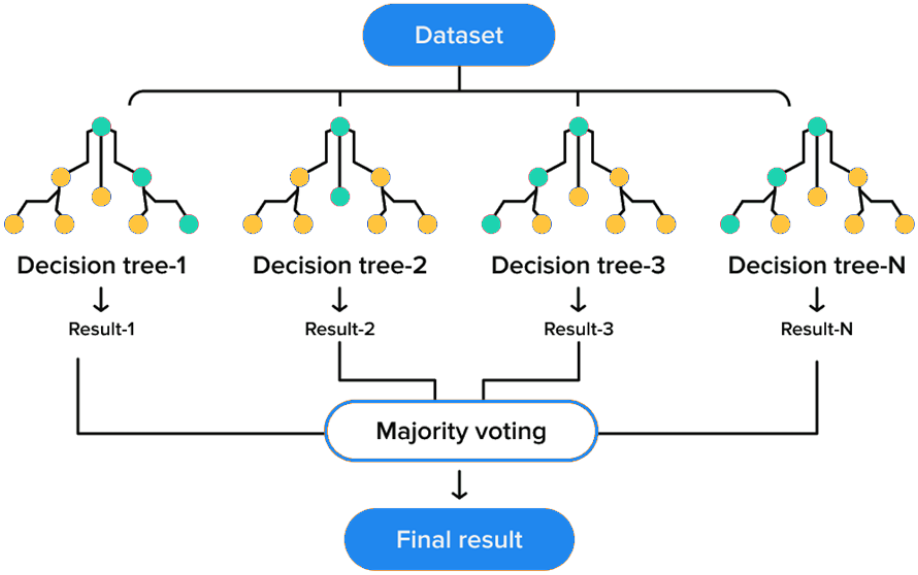

Figure 2: Diagram of a random forest ensemble.

2.5.2 One vs Rest Classification

Since the dataset is multi-label, a one vs rest classification was used. Each classifier (random forest ensemble) would produce a binary result to classify one class against the rest.

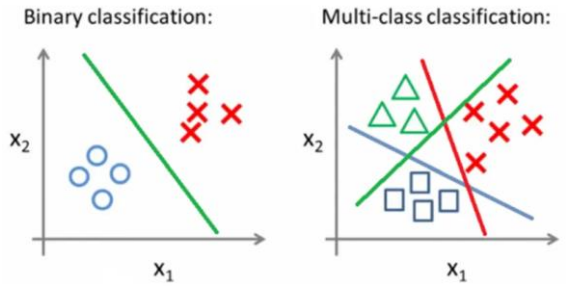

Figure 3: Diagram of one vs rest classification.

2.5.3 Classifier Chaining

Since each classifier is only responsible for classifying one class, 20 different classifiers would be necessary to classify the 20 different diagnoses. These 20 classifiers would be chained together to produce a final prediction. Classifier chaining is done by first having the first classifier produce a prediction for one of the diagnoses, then feeding that prediction as an extra feature to the next classifier, and so on. Each classifier down the chain will hence receive an increasing number of features. For our case, there are 28 features initially, which the first classifier will receive, then the next 29 features, and eventually the last one will receive 47 features.

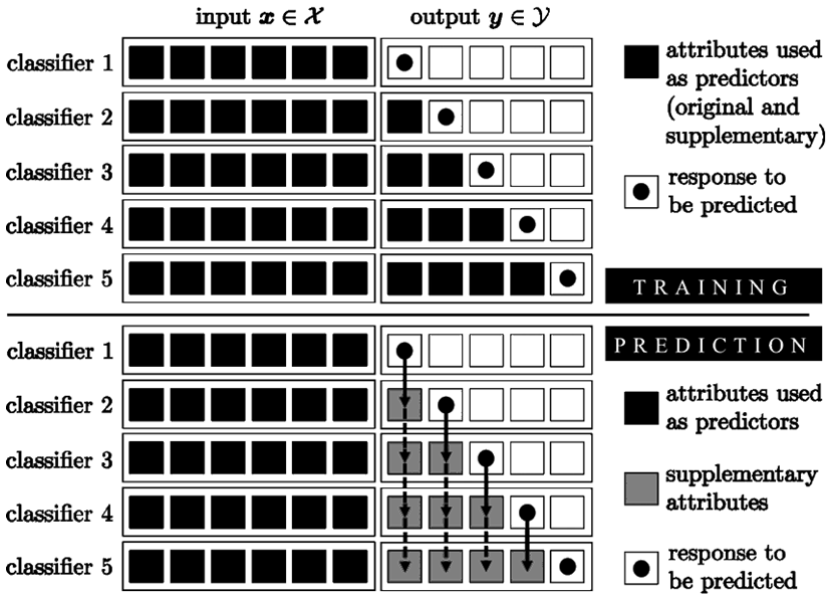

Figure 4: Diagram of classifier chaining.

2.5.4 Training-Testing Data Split

Considering the significance of duplicated data, as shown in *2.4 Data Augmentation* above, a larger testing-to-training data ratio of 1:1 was used, meaning that 50% of the overall dataset was each isolated to be the training and testing data. This decision was made to mitigate the potential of overfitting as much as possible to retain integrity. The data split was also randomised to prevent biases.

3 Results

For the training data, the model produced perfect 100% accuracy, precision, and recall scores. The confusion matrix for the training dataset can be seen in *Figure 5* below. A comprehensive display of the confusion matrices for each diagnosis can be seen in *Figure 7* below.

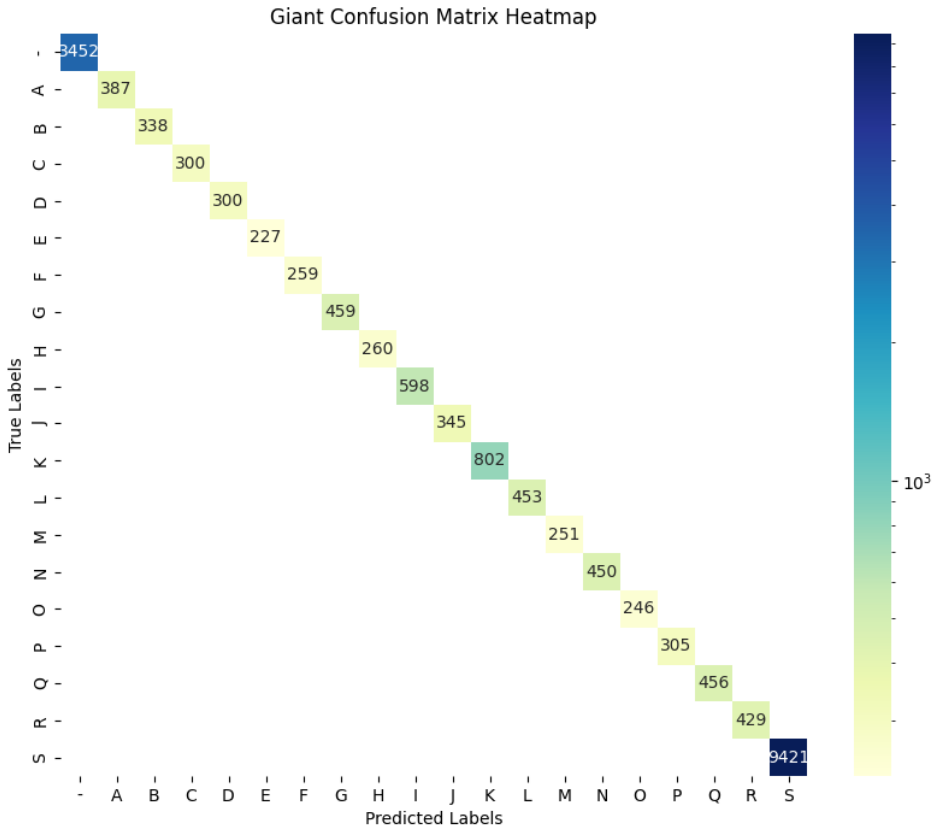

Figure 5: Confusion matrix heatmap for the training data.

For the testing data, the model produced 97.3% accuracy, 98.2% precision, and 97.5% recall scores. The confusion matrix for the testing data can be seen in *Figure 6* below. A comprehensive display of the confusion matrices for each diagnosis can be seen in *Figure 8* below.

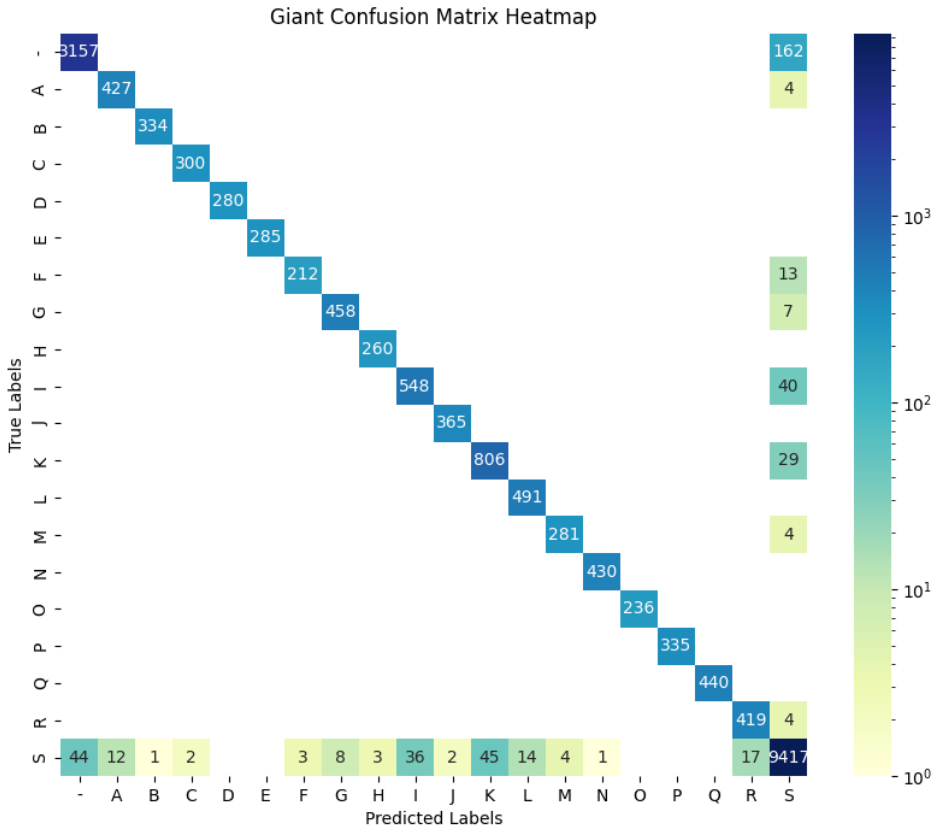

Figure 6: Confusion matrix heatmap for the testing data.

We can conclude that AI models can, in fact, be an effective and accurate method of diagnosing thyroid diseases given blood test results at 97% accuracy.

4 Discussion

While the current trial shown in this report can produce a satisfactory result, numerous limitations exist.

Firstly, the rarity of certain diagnoses led to the necessity of data augmentation, as demonstrated in *2.4 Data Augmentation* above. This obscures the variations and patterns on blood tests that a certain diagnosis may produce, leading to the model possibly overfitting for those diagnoses. Future work could overcome this problem by simply including more data points.

Secondly, while this model could predict the variation of thyroid diseases from blood test results, this only solves part of the current problem. As alluded to earlier in *1 Problem Statement* above, a big part of the problem with thyroid disease diagnosis is the overlapping of symptoms and markers, leading to difficulties in discriminating differential diagnoses. Future work could alleviate this problem by compiling a dataset with more features with increased diversity and more diagnoses of often-confused diseases and training a model specifically for discriminating differential diagnoses.

Thirdly, while a binary diagnosis can aid medical judgement, the model can provide much better information by predicting the disease's stage or severity and suggesting treatment plans. However, obtaining a dataset encompassing these may be difficult due to the broad scope necessary.

In conclusion, this project proved the viability of diagnosing thyroid diseases using blood work results with a high accuracy.

5 References

[1] M. P. J. Vanderpump, “Epidemiology of Thyroid Disorders,” in *The Thyroid and Its Diseases: A Comprehensive Guide for the Clinician*, 1 ed., M. Luster, L. H. Duntas and L. Wartofsky, Eds., Springer Cham, 2019, pp. 75-85.

[2] B. S. Harris and J. L. Eaton, “Postpartum Thyroiditis,” in *Thyroid Disease and Reproduction: A Clinical Guide to Diagnosis and Management*, 1 ed., J. L. Eaten, Ed., Springer Cham, 2019, pp. 183-188.

[3] M. Soundarrajan and P. A. Kopp, “Thyroid Hormone Biosynthesis and Physiology,” in *Thyroid Disease and Reproduction: A Clinical Guide to Diagnosis and Management*, 1 ed., J. L. Eaton, Ed., Springer Cham, 2019, pp. 1-17.

[4] V. V. Garla, L. L. Y. Cardozo and L. F. Lien, “Hypothyroidism,” in *Thyroid Disease and Reproduction: A Clinical Guide to Diagnosis and Management*, 1 ed., J. L. Eaton, Ed., Springer Cham, 2019, pp. 19-43.

[5] R. Quinlin, “Thyroid Disease,” UCI Machine Learning Repository, [Online]. Available: https://doi.org/10.24432/C5D010..

[6] E. F. Werr, “Thyroid Disease Data,” Kaggle, [Online]. Available: https://www.kaggle.com/datasets/emmanuelfwerr/thyroid-disease-data.

6 Appendix

Table 2: Comprehensive list of the original dataset columns and their meanings.

Column Name	Data Type	Data Meaning
age	number	Age of the patient
sex	(“M” | “F”)?	Sex of the patient
on_thyroxine	“t” | “f”	Whether the patient is on thyroxine
query_on_thyroxine	“t” | “f”	Whether the patient is on thyroxine?
on_antithyroid_meds	“t” | “f”	Whether the patient is on anti-thyroids
sick	“t” | “f”	Whether the patient is sick
pregnant	“t” | “f”	Whether the patient is pregnant
thyroid_surgery	“t” | “f”	Whether the patient has undergone thyroid surgery
I131_treatment	“t” | “f”	Whether the patient is undergoing I131 treatment
query_hypothyroid	“t” | “f”	Whether the patient believes they have hypothyroid
query_hyperthyroid	“t” | “f”	Whether the patient believes they have hyperthyroid
lithium	“t” | “f”	Whether the patient is on lithium
goitre	“t” | “f”	Whether the patient has goitre
tumor	“t” | “f”	Whether the patient has a tumour
hypopituitary	“t” | “f”	Whether the patient has hypopituitary
psych	“t” | “f”	Unclear?
TSH_measured	“t” | “f”	Whether TSH was measured
TSH	number?	Measured TSH levels
T3_measured	“t” | “f”	Whether T3 was measured
T3	number?	Measured T3 levels
TT4_measured	“t” | “f”	Whether TT4 was measured
TT4	number?	Measured TT4 levels
T4U_measured	“t” | “f”	Whether T4U was measured
T4U	number?	Measured T4U levels
FTI_measured	“t” | “f”	Whether FTI was measured
FTI	number?	Measured FTI levels

TBG_measured	"t" | "f"	Whether TBG was measured
TBG	number?	Measured TBG levels
referral_source	string	Patient referral source
target	string	Medical diagnoses of the patient
patient_id	number	Patient ID

Table 3: A comprehensive list of the diagnoses and their character representations.

Character	Occurrence		Medical Diagnosis	Diagnosis Category
-	6771	73.82%	Healthy	Healthy
A	193	2.10%	Hyperthyroid	Hyperthyroid conditions
B	21	0.23%	T3 toxic	
C	18	0.20%	Toxic goitre	
D	9	0.10%	Secondary toxic	
E	1	0.01%	Hypothyroid	Hypothyroid conditions
F	239	2.61%	Primary hypothyroid	
G	419	4.57%	Compensated hypothyroid	
H	8	0.09%	Secondary hypothyroid	
I	371	4.04%	Increased binding proteins	Binding protein abnormalities
J	43	0.47%	Decreased binding proteins	
K	573	6.25%	Concurrent non-thyroidal illness	Other diseases
L	116	1.26%	Consistent with therapy	Replacement therapy evaluation
M	129	1.41%	Under-replaced	
N	110	1.20%	Over-replaced	
O	15	0.16%	Anti-thyroid drugs	Anti-thyroid treatment evaluation
P	5	0.05%	I131 treatment	
Q	14	0.15%	Surgery	
R	197	2.15%	Discordant assay results	Miscellaneous
S	85	0.93%	Elevated TBG	
T	0	0.00%	Elevated Thyroid hormones	

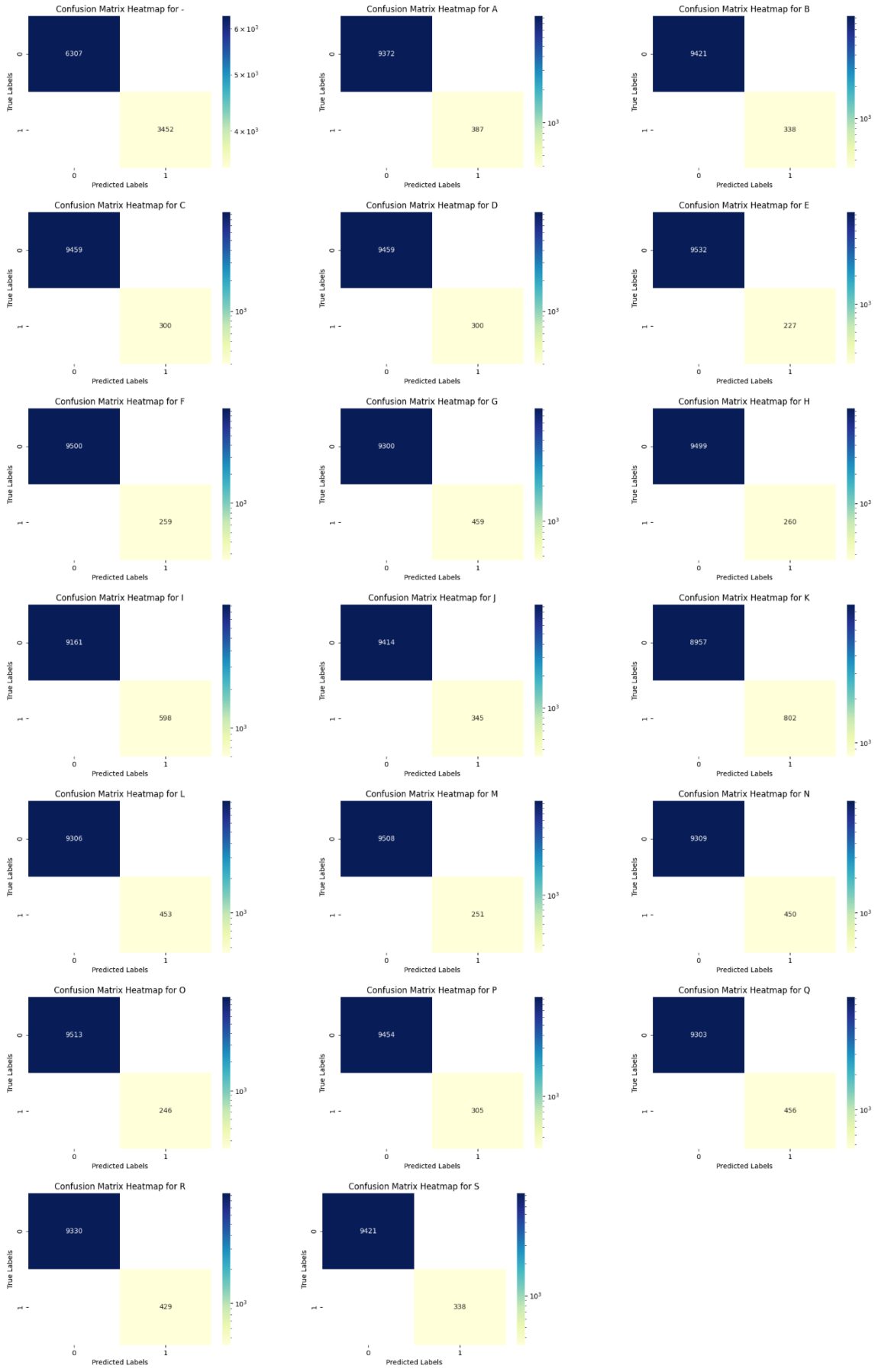

Figure 7: Comprehensive display of confusion matrices of each diagnosis for the training data.

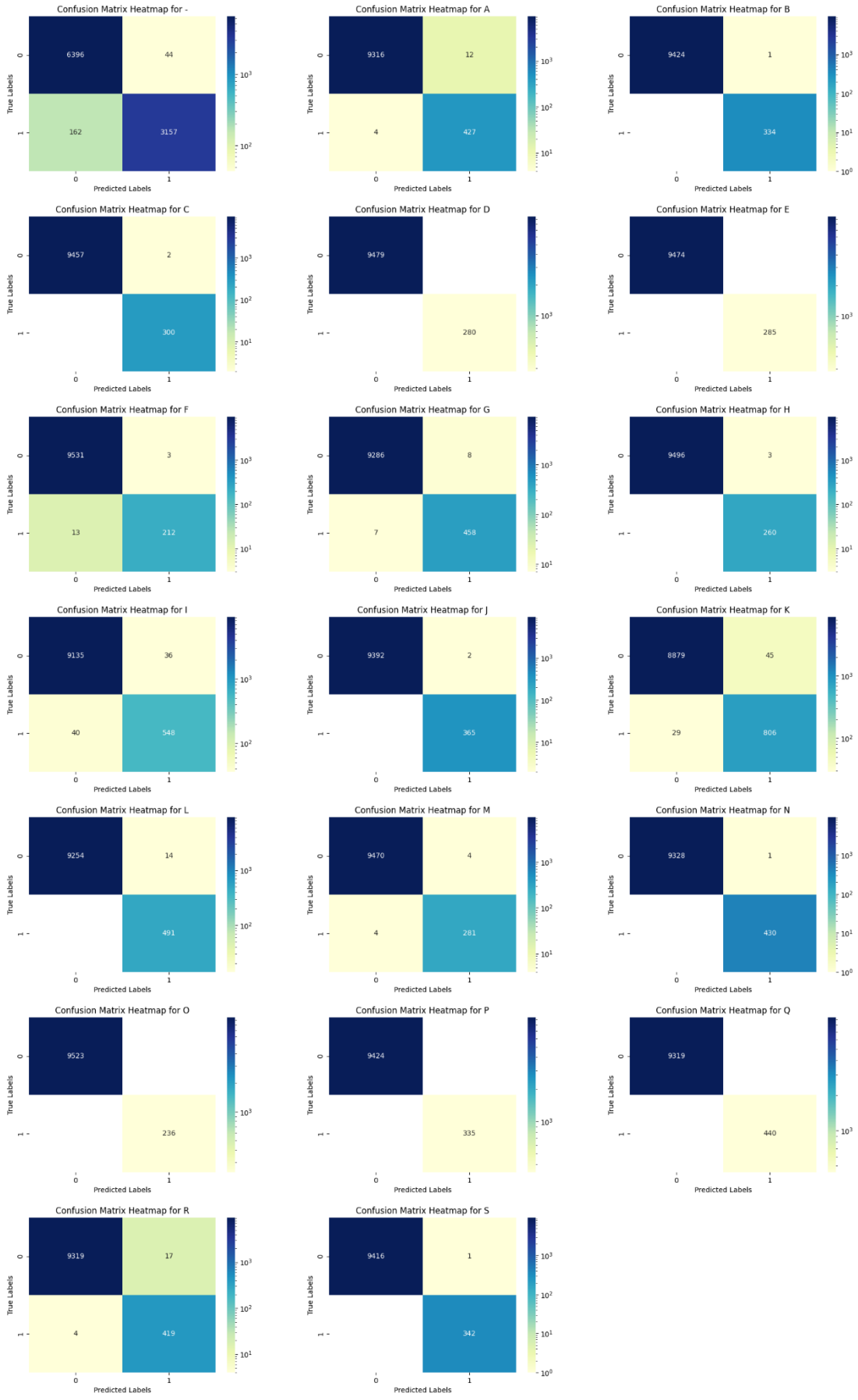

Figure 8: Comprehensive display of confusion matrices of each diagnosis for the testing data.