

Thyroid Disorders Classification

Final Project Report

Group 23

Stuti Dhebar

Venkati Kavya Kandipalli

857-313-4612 (Tel of Stuti)

857-799-0478 (Tel of Kavya)

dhebar.s@northeastern.edu

kandipalli.v@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Stuti Dhebar

Signature of Student 2: Venkati Kavya Kandipalli

Submission Date: 04/23/2023

Problem Setting

Thyroid disorders occur when the thyroid gland, a small organ located at the front of the neck, produces too few or too many hormones. These hormones play a vital role in maintaining our body functions and an imbalance in these could affect our heart rate, weight, energy levels, bone health, and more. Some causes of thyroid problems are iodine deficiency, genetic disorders, nodules (non-cancerous lumps), inflammation, etc. According to statistics, approximately 200 million people around the world have some type of thyroid problem. In recent times, this condition has been on the rise and if left untreated, it can cause serious issues, including heart disease and nerve damage. Therefore, in order to provide timely treatment and lower medical expenses from the perspective of both the patients and healthcare providers, it is imperative to work on this problem.

Problem Definition

With the advancement in data mining technologies, machine learning can be used to detect thyroid disorders at an early stage leading to early treatment and saving people's lives. Machine learning techniques are being widely used for their ability to analyze large sets of data and identify intrinsic relationships in the dataset better than human beings. So through this project, we aim to work around the problem of thyroid disease identification and classification by developing a supervised machine learning model. With the help of this model, we will be able to understand the following:

- Predict whether an individual has a thyroid condition or is at the risk of having one.
- Determine the type of thyroid condition, such as hyperthyroid or hypothyroid.
- Understand which gender is more prone to be diagnosed with this disorder.

Data Sources

- For this project we have collected the data from UCI Machine Learning Repository which consists of databases, domain theories, and more for working on machine learning problems in different areas such as healthcare, finance, economics, etc.

- The dataset we are using is [UCI Thyroid Disease Data](#), which has been donated by the Garvan Institute of Medical Research in Sydney, Australia.

Data Description

Our data zip file consists of different datasets. For our analysis, we are working on a combination of three datasets with the same features and target variable in each. This combined dataset has approximately 15000 records and 30 columns. Below is a snapshot of the description of all columns along with their data types.

Attribute	Description	Data Type
age	age of the patient	(int)
sex	sex patient identifies	(str)
on_thyroxine	whether patient is on thyroxine	(bool)
query on thyroxine	whether patient is on thyroxine	(bool)
on antithyroid meds	whether the patient is on antithyroid meds	(bool)
sick	whether patient is sick	(bool)
pregnant	whether patient is pregnant	(bool)
thyroid_surgery	whether patient has undergone thyroid surgery	(bool)
I131_treatment	whether patient is undergoing I131 treatment	(bool)
query_hypothyroid	whether the patient believes they have hypothyroid	(bool)
query_hyperthyroid	whether the patient believes they have hyperthyroid	(bool)
lithium	whether patient * lithium	(bool)
goitre	whether patient has goitre	(bool)
tumor	whether patient has tumor	(bool)
hypopituitary	whether patient * hyperpituitary gland	(float)
psych	whether patient * psych	(bool)
TSH_measured	whether TSH was measured in the blood	(bool)
TSH	TSH level in blood from lab work	(float)
T3_measured	whether T3 was measured in the blood	(bool)
T3	T3 level in blood from lab work	(float)
TT4_measured	whether TT4 was measured in the blood	(bool)
TT4	TT4 level in blood from lab work	(float)
T4U_measured	whether T4U was measured in the blood	(bool)
T4U	T4U level in blood from lab work	(float)
FTI_measured	whether FTI was measured in the blood	(bool)
FTI	FTI level in blood from lab work	(float)
TBG_measured	whether TBG was measured in the blood	(bool)
TBG	TBG level in blood from lab work	(float)
referral_source		(str)
target	hyperthyroidism medical diagnosis	(str)
patient_id	unique id of the patient	(str)

Figure 1. Data Attributes & Description

Data Exploration

The next step is data exploration and visualization. Before understanding which models to select and implement, it is important to understand what our dataset looks like and what needs to be done to improve it. We can easily examine enormous amounts of data using various data manipulation and visualization libraries in Python to better comprehend the subsequent steps for our analysis. In this exploratory data analysis, we performed the following:

Described the dataset to statistically analyze values in numerical columns and get count/frequencies for categorical columns.

	Age	Sex	On_Thyroxine	Query_On_Thyroxine	On_Antithyroid_Medication	Sick	Pr	T4U_Measured	T4U	FTI_Measured	FTI	TBG_Measured	TBG	Class
count	7542.000000	7542	7542	7542	7542	7542	7542	7542	7542.000000	7542	7542.000000	7542	7542.000000	7542.000000
unique	NaN	2	2	2	2	2	2	2	NaN	2	NaN	2	NaN	Na
top	NaN	F	f	f	f	f	f	t	NaN	t	NaN	f	NaN	Na
freq	NaN	5148	6695	7419	7450	7262	7450	6866	NaN	6873	NaN	7283	NaN	Na
mean	51.934235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.880593	NaN	100.759996	NaN	0.788299	1.87311
std	18.620319	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.316777	NaN	47.007687	NaN	4.329497	0.39885
min	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	NaN	0.000000	NaN	0.000000	0.00000
25%	37.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.830000	NaN	88.000000	NaN	0.000000	2.00000
50%	55.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.950000	NaN	105.000000	NaN	0.000000	2.00000
75%	67.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.050000	NaN	123.000000	NaN	0.000000	2.00000
max	97.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.120000	NaN	839.000000	NaN	45.000000	2.00000

Figure 2. Summary Statistics of the Dataset

With the help of describe() in Python, we generated a statistical summary of continuous columns such as count, mean, median, percentiles, standard deviation, and more. It also provides a summary for nominal columns, including frequency, most common values, and count of unique values.

Correlation analysis for exploring the relationship between numerical columns

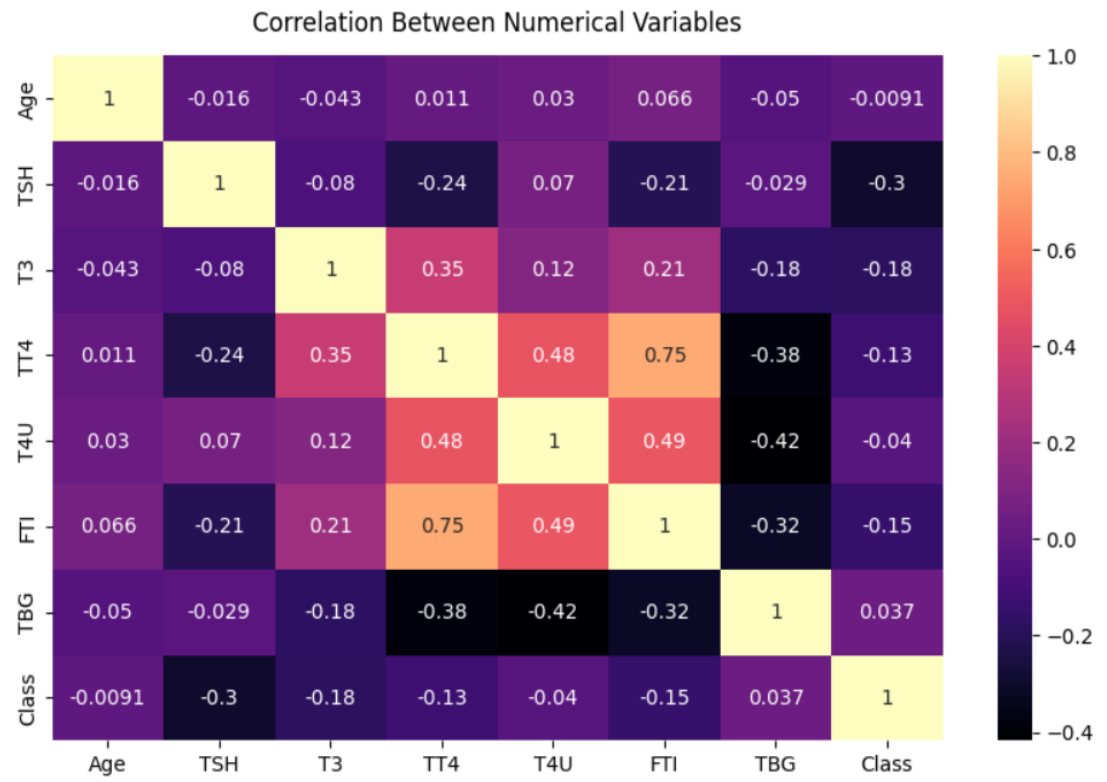


Figure 3. Correlation Analysis

The heatmap showcases the correlation levels among the numerical variables in the thyroid dataset. Based on the chart, it is evident that TT4, T4U, and FTI exhibit a stronger correlation with one another compared to the remaining variables. This indicates that any increase or decrease in the levels of one hormone is likely to cause a corresponding increase or decrease in the levels of the other hormones. Additionally, T3, which is a significant thyroid hormone, shows a positive correlation with TT4 and FTI.

Distribution of instances across each class - hyperthyroid, hypothyroid, and negative

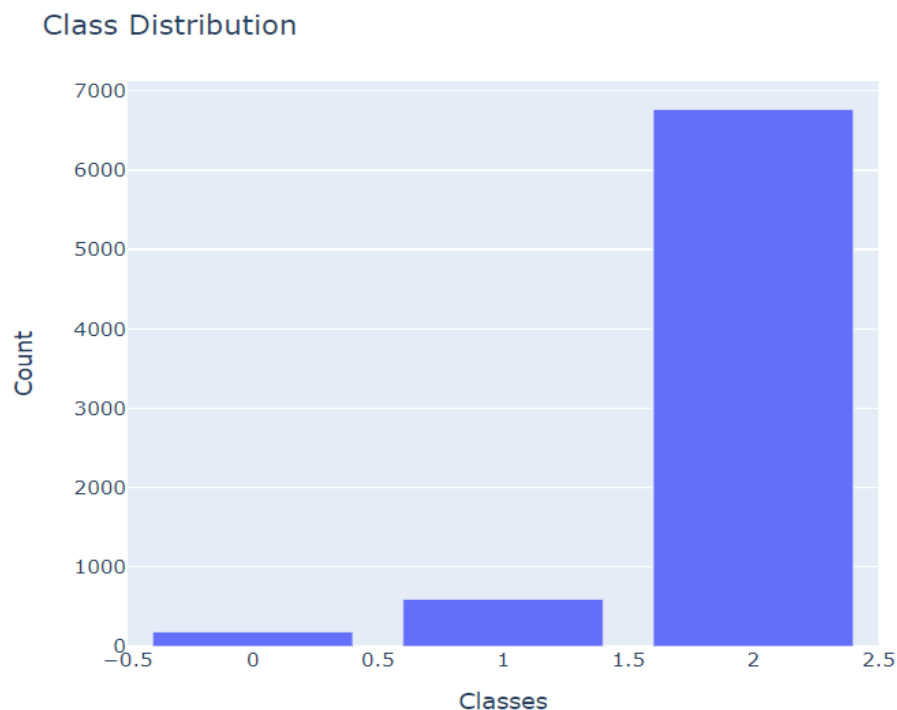


Figure 4. Target Variables Class Distribution

This bar plot shows the number of instances in each class i.e the highest number of instances are for negative (2) class, followed by hypothyroid (1), and then hyperthyroid (0) having the lowest number of instances. From this we understood that our dataset is highly imbalanced and we may need to perform some oversampling techniques such as SMOTE NC or adjust class weights during model implementation so that the model does not become biased towards the class with highest instances during the training phase.

Analysis of value distribution with box plots to check for outliers

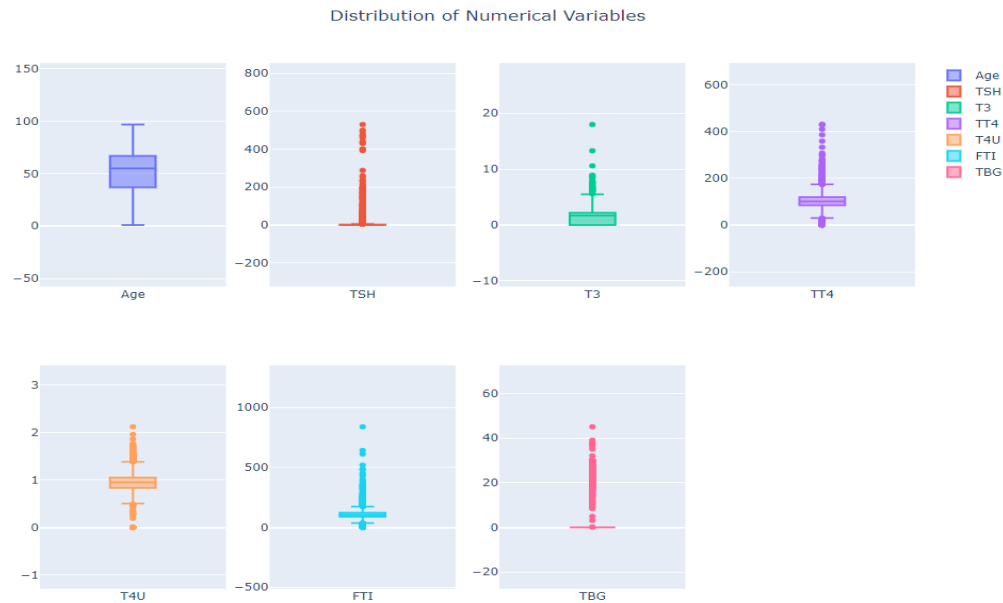


Figure 5. Boxplot Analysis of Numerical Columns

The diagram depicted below illustrates the distribution of several variables that were measured during the thyroid blood test within our dataset. It appears that some patients exhibit unusually high values in critical variables like TSH, T3, TT4, FTI, and TBG, which could suggest an imbalance in their thyroid levels. Interestingly, some patients with values within the normal range still show signs of a thyroid condition. To gain a better understanding of these aberrant results, additional patient information is necessary.

Pie chart showing the percentage of males and females getting affected by thyroid disorders

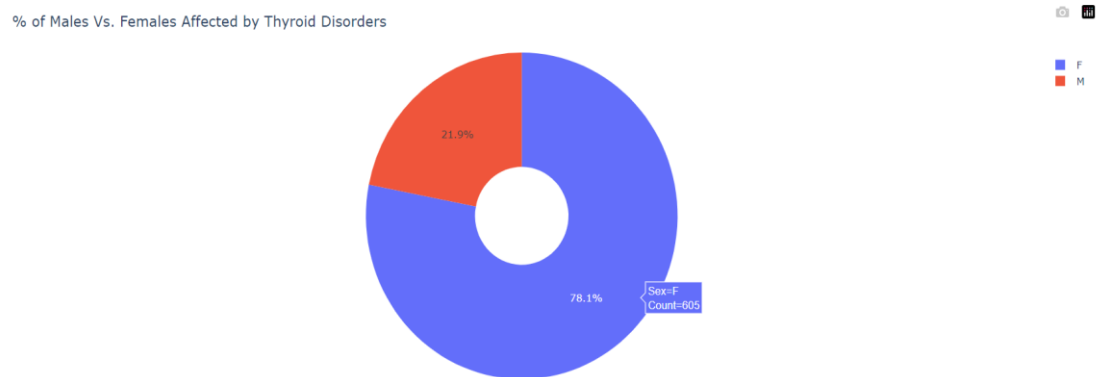


Figure 6. Pie Chart Showing Disease Distribution by Gender

Based on the information presented in the pie chart, we can conclude that a larger number of females in the dataset have thyroid disorders as compared to males. This finding is consistent with information from other sources that suggests women are more susceptible to thyroid conditions than men. One possible explanation for this trend is the interplay between thyroid hormones and hormones that fluctuate during the menstrual cycle. Additionally, autoimmune responses, where the body's immune system attacks its own cells, may contribute to this prevalence. While the exact reasons behind this phenomenon are not well understood, available data supports the notion that it is more prevalent among women than men.

Data Mining Tasks

In this step, we performed various data preprocessing techniques to clean our dataset before passing it to our model. This step is important as it determines the quality of our model's output and hence the data needs to be transformed to its cleanest form so that the model can understand the data and correctly identify instances. We performed the following for data preprocessing:

- Splitting of target variable - The target class consisted of both the classes and the patient IDs. So, we split this column into two separate columns.
- Dropped unnecessary columns - Some columns were not useful at any stage of our analysis such as 'Referral Source' and 'Patient ID', so these columns were dropped.
- Transformed values in target variable - Our target column had subclasses of the main classes of interest (hyperthyroid, hypothyroid, and negative). These subclasses were grouped into the three main classes for our multiclass classification task.
- Checked for discrepancies in the dataset - missing, NaN, duplicate, and outliers:
 - Outliers - In this dataset, there were a lot of outliers. But we choose to keep these as in healthcare datasets, outliers might indicate a more serious problem with the patient. Without much information about why these values are so high or low, we cannot drop them. The 'Age' column also had outliers, like 455 and 65511. We know these cannot be actual age values so we decided to drop such records.

- Missing or ‘?’ values - There were a lot of ‘?’ values in continuous columns. After examining further, we figured that these values were because their related columns were not measured. For instance, in the column ‘TT4_Measured’, if the value is false, then column ‘TT4’ will have ‘?’ as its value. This is because in this patient’s blood sample this value was not measured. Since there was no way to determine these values, we decided to impute these question marks with 0.
- NaN and duplicates - Our dataset did not have any NaN values or duplicate records of patients.
- Label Encoding - The target column was label encoded to convert hyperthyroid class to 0, hypothyroid class to 1, and negative class to 2.
- Data Rescaling and Encoding - This data had both nominal and continuous features. Since some columns were widely distributed, it was important to rescale these columns in a way that the model can interpret. To perform this, we implemented standardization on continuous columns and one-hot encoding on nominal columns. After performing the latter, our dataset’s dimensionality increased in terms of columns. Below is a snippet of how our dataset looks after performing preprocessing.

```
# Getting numerical and categorical columns
num_cols = ['Age', 'TSH', 'T3', 'TT4', 'T4U', 'FTI', 'TBG']
cat_cols = ['Sex', 'On_Thyroxine', 'Query_On_Thyroxine', 'On_Antithyroid_Medication', 'Sick', 'Pregnant', 'Thyroid_Surgery',
            'T131_Treatment', 'Query_Hypothyroid', 'Query_Hyperthyroid', 'Lithium', 'Goitre', 'Tumor', 'Hypopituitary', 'Psych',
            'TSH_Measured', 'T3_measured', 'TT4_Measured', 'T4U_Measured', 'FTI_Measured', 'TBG_Measured']

# Creating a column transformer to implement one hot encoding and normalization
preprocessor = ColumnTransformer(
    transformers = [
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(), cat_cols)
    ])

# Fitting the column transformer on original data
processed_data = preprocessor.fit_transform(thyroid_data)

# Creating a new dataframe with the processed data
processed_df = pd.DataFrame(
    processed_data,
    columns = num_cols + list(preprocessor.named_transformers_['cat'].get_feature_names_out(cat_cols))
)
```

Figure 7. Standardization and One-Hot Encoding Implementation

	Age	TSH	T3	TT4	T4U	FTI	TBG	Sex_F	Sex_M	On_Thyroxine_f	On_Thyroxine_t	Query_On_Thyroxine_f
0	-1.231760	-0.185266	-1.248935	-2.581264	-2.780032	-2.143621	-0.182088	1.0	0.0	1.0	0.0	1.0
1	-1.231760	-0.132967	0.411433	0.714243	-2.780032	-2.143621	-0.182088	1.0	0.0	1.0	0.0	1.0
2	-0.587260	-0.197335	-1.248935	-2.581264	-2.780032	-2.143621	2.358791	1.0	0.0	1.0	0.0	1.0
3	-0.855801	-0.197335	-1.248935	-2.581264	-2.780032	-2.143621	5.823627	1.0	0.0	1.0	0.0	1.0
4	0.433199	-0.197335	-1.248935	-2.581264	-2.780032	-2.143621	5.823627	1.0	0.0	1.0	0.0	1.0

Figure 8. Rescaled and Encoded Features

Data Mining Models/Methods

After processing the records, the next step was to choose the right type of features and check for any oversampling techniques to make the instances equal across all classes. For feature selection, we implemented both Recursive Feature Elimination (RFE) and Random Forest Classifier. After implementing and evaluating the model's performance with both approaches, we decided to keep Random Forest Classifier as our feature selection technique because the features selected by this method led to better model performance. After reading more about why RFC performs better, we understood that this was because RFE is more prone to getting affected by outliers whereas RFC is robust to outliers. As seen in the EDA above, our dataset has many outliers so that explains why RFC performs better.

```
# Performing feature selection to get the most optimal features for our model
rf = RandomForestClassifier()
hyper_rf = rf.fit(processed_df, thyroid_data['class'])
imp = hyper_rf.feature_importances_
sorted_features = sorted(zip(imp, processed_df), reverse=True)

features = []
for importance, feature in sorted_features[:6]:
    features.append(feature)
    print("{}: {}".format(feature, importance))
```

TSH: 0.558398455637358
FTI: 0.1224766955414211
TT4: 0.11705154846260081
T3: 0.06322886330731196
T4U: 0.03748365238170565
Age: 0.024471615459582353

Figure 9. Feature Selection using Random Forest Classification

The next step was to split the dataset into training and testing sets and see if any oversampling techniques are required. For this, we tried a SMOTE (Synthetic Minority Oversampling Technique) technique for nominal and continuous variables - SMOTE NC on the training set. After evaluating the model's performance with and without this technique, we realized that SMOTE was introducing a lot of noise to our dataset and reducing the model's performance on the test data. Therefore, we decided to not use any oversampling technique.

Finally, we selected six algorithms for our multiclass classification task - Naive Bayes (NB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Trees (DT), Logistic Regression (LR), and Multi-Layer Perceptron (MLP). These algorithms were chosen for their ability to perform multiclass classification. Some of these algorithms were implemented with hyperparameter tuning with GridSearchCV (5- fold cross validation). Hyperparameter tuning is done to choose the best parameters for these models. GridSearchCV, by default, uses 5-fold cross validation where it divides the input data into training and testing sets (in this case five times) and evaluates the model's performance every time to choose the best parameters. This approach improved our performance so it was implemented for all algorithms except Naive Bayes.

```
[ ] # Implementing KNN with grid search cv
    param_grid = {'n_neighbors': range(1, 11)}

    knn = KNeighborsClassifier()
    knn_gridsearch = GridSearchCV(knn, param_grid, cv = 5)
    knn_gridsearch.fit(X_train, y_train)
    y_pred = knn_gridsearch.predict(X_test)

    # Evaluating model performance
    knn_acc = accuracy_score(y_test, y_pred)
    knn_pre = precision_score(y_test, y_pred, average = 'weighted')
    knn_rec = recall_score(y_test, y_pred, average = 'weighted')
    knn_f1 = f1_score(y_test, y_pred, average = 'weighted')
```

Figure 10. K-Nearest Neighbors Implementation with GridSearchCV

```
[ ] # Instantiate logistic regression object
lr = LogisticRegression()

# Define hyperparameter grid for grid search
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.01, 0.1, 1, 10, 100]}

# Perform grid search cross-validation
grid_search = GridSearchCV(lr, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get best hyperparameters and fit model with best hyperparameters
best_params = grid_search.best_params_
best_lr = LogisticRegression(penalty=best_params['penalty'], C=best_params['C'])
best_lr.fit(X_train, y_train)

# Predict on test set
y_pred = best_lr.predict(X_test)

# Compute accuracy, precision, recall, and F1 score
lr_acc = accuracy_score(y_test, y_pred)
lr_pre = precision_score(y_test, y_pred, average='weighted')
lr_rec = recall_score(y_test, y_pred, average='weighted')
lr_f1 = f1_score(y_test, y_pred, average='weighted')
```

Figure 11. Logistic Regression Implementation with GridSearchCV

```
# Define the hyperparameter grid for tuning
param_grid = {'C': [0.1, 1, 10], # Values for the regularization parameter C
              'kernel': ['linear', 'rbf'], # Kernels to try: linear and radial basis function (RBF)
              'gamma': ['scale', 'auto'] + [0.1, 1, 10]} # Values for the gamma parameter

# Create an instance of SVM classifier
svm = SVC()

# Create GridSearchCV with SVM classifier and the hyperparameter grid
grid_search = GridSearchCV(svm, param_grid, cv=5) # Use 5-fold cross-validation

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from GridSearchCV
best_params = grid_search.best_params_
best_C = best_params['C']
best_kernel = best_params['kernel']
best_gamma = best_params['gamma']

# Create a new instance of SVM classifier with the best hyperparameters
best_svm = SVC(C=best_C, kernel=best_kernel, gamma=best_gamma)

# Fit the best SVM classifier to the training data
best_svm.fit(X_train, y_train)

# Make predictions on the test data
y_pred = best_svm.predict(X_test)
```

Figure 12. Support Vector Machine Implementation with GridSearchCV

```
[ ] # Instantiating Decision Trees object
dt = DecisionTreeClassifier()

# Defining hyperparameter grid for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


# Performing grid search with cross-validation
grid_search = GridSearchCV(dt, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Extracting best parameters and best model from grid search
best_params = grid_search.best_params_
best_dt = grid_search.best_estimator_

# Predicting new values for target class using best model
y_pred = best_dt.predict(X_test)

# Computing the accuracy of the classifier
dt_acc = accuracy_score(y_test, y_pred)
dt_pre = precision_score(y_test, y_pred, average='weighted')
dt_rec = recall_score(y_test, y_pred, average='weighted')
dt_f1 = f1_score(y_test, y_pred, average='weighted')
```

Figure 13. Decision Tree Implementation with GridSearchCV

```
 # Instantiating naive bayes object
nb = GaussianNB()

# Training model with data for hyperthyroid
nb.fit(X_train, y_train)

# Predicting new values for hyperthyroid
y_pred = nb.predict(X_test)

# Evaluating the classifier
nb_acc = accuracy_score(y_test, y_pred)
nb_pre = precision_score(y_test, y_pred, average = 'weighted')
nb_rec = recall_score(y_test, y_pred, average = 'weighted')
nb_f1 = f1_score(y_test, y_pred, average = 'weighted')
```

Figure 14. Gaussian Naïve Bayes Implementation

```
[ ] # Define MLP Classifier
mlp = MLPClassifier()

# Define hyperparameters and their possible values for tuning
param_grid = {
    'hidden_layer_sizes': [(64,), (32, 16,), (16, 8,)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001, 0.01],
    'batch_size': [32, 64],
    'max_iter': [100, 200],
    'random_state': [42]
}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(mlp, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the MLP Model with the best hyperparameters
best_mlp = MLPClassifier(**best_params)
best_mlp.fit(X_train, y_train)

# Evaluate the MLP Model
y_pred = best_mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

Figure 15. Multi-Layer Perceptron Implementation with GridSearchCV

Performance Evaluation

Finally, the models trained with the training set were evaluated on the testing set. The results were as shown in the table below.

	Algorithm	Accuracy	Precision	Recall	F1 Score
0	K-Nearest Neighbors	0.944857	0.945818	0.944857	0.939589
1	Support Vector Machine	0.973489	0.973926	0.973489	0.973498
2	Decision Trees	0.984624	0.986304	0.984624	0.985246
3	Logistic Regression	0.965536	0.963958	0.965536	0.963846
4	Naive Bayes	0.957052	0.956061	0.957052	0.955141
5	Multi-layer Perceptron	0.979852	0.982013	0.979852	0.980646

Figure 16. Model Performance Evaluation Table

We can see that all our models are performing well across all metrics. But the top three algorithms giving the best performance are Decision Trees, followed by Multi-Layer Perceptron and Support Vector Machine. The performance of these three models is excellent with a high precision, recall, and f1-score of all around 0.97 or 0.98. The models show a good balance between precision (ability to correctly identify true positives) and recall (ability to capture all actual positive instances). The high f1-scores also indicate a good trade-off between precision and recall, making these suitable for our problem where the emphasis is on both precision as well as recall. Overall, these models ability to perform this well could be attributed to their inherent qualities, such as their ability to capture non-linear relationships, adapt to data during training, handle imbalanced datasets, and outliers, and find optimal decision boundaries. Additionally, hyperparameter tuning applied during the training process could have significantly contributed to their performance.

Project Results

In conclusion, we started out by selecting a dataset consisting of blood sample reports measuring thyroid hormone levels and other external factors to perform exploratory data analysis and implement various classification algorithms for multi-class classification. Through EDA, we were able to find out that females are more prone to getting affected by thyroid disorders than males. Next, the records were wrangled to impute missing values, change data types, drop irrelevant records/features, scale numerical variables, and encode nominal variables. Followed by this, the performance of the different algorithms chosen was evaluated using key classification metrics such as precision, recall, f1-score, and accuracy. Based on these results, we found out that the top three performing models were Decision Trees, Multi-layer Perceptron, and Support Vector Machine. These models showed promising results in classifying and predicting the class labels accurately on test data. Finally, the future scope of our project is to read more about the healthcare domain and explore additional ways to improve performance, by refining data imputation techniques and feature and model selection strategies.

Impact of the Project Outcomes

The outcomes of our project on thyroid disorders classification have had a significant impact on our understanding and management of this health condition. Through the analysis of a large dataset

comprising patient records, we successfully developed several classification models that demonstrated high accuracy in predicting thyroid disorder cases. This has potential implications for early diagnosis and intervention, leading to improved patient outcomes. Additionally, the project highlighted the importance of various features in determining the risk of thyroid disorders, such as age, gender, and thyroid-stimulating hormone (TSH) levels. These findings provide valuable insights for healthcare professionals in identifying high-risk individuals and tailoring personalized treatment plans. Overall, the results have the potential to positively impact clinical practice by aiding in early detection, risk assessment, and personalized treatment of thyroid disorders, ultimately benefiting patients, and improving healthcare outcomes.