Objective :

To solve the multilabel classification task on a text dataset represented as GatorTron embeddings and ICD10 medical codes as the target labels.

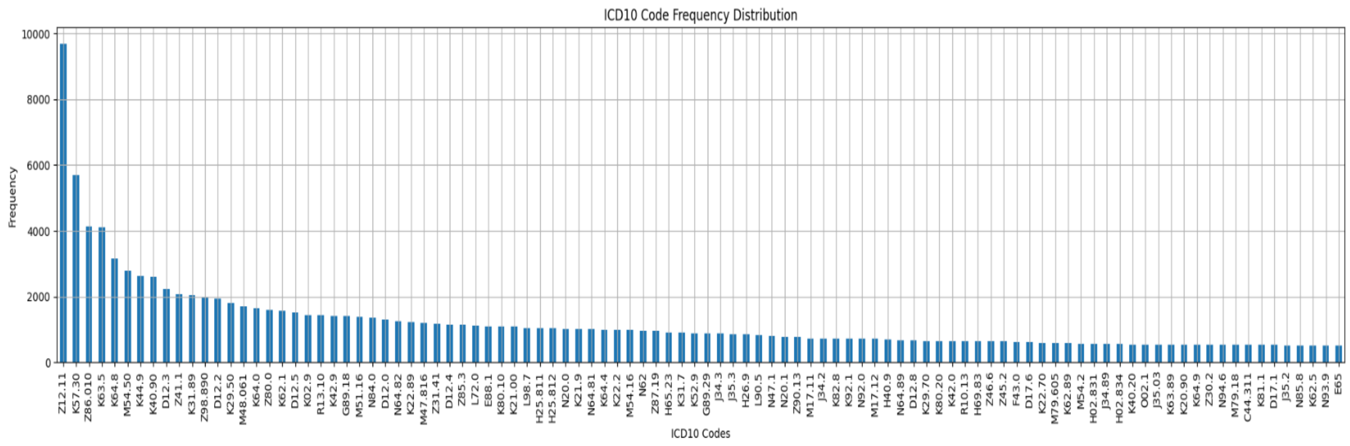
Dataset:

The dataset consists of embeddings of a corpus of medical charts and their respective (multi)labels. Two files embeddings_1.npy and embeddings_2.npy is provided where the embeddings are provided as NumPy matrices of dimension 1024. The labels corresponding to the (embeddings)files are provided as text files.

Methodology:

To analyse the distribution of ICD10 codes within each label associated with the provided embeddings, bar plots are generated for each unique ICD10 code across the labels. This visualization highlights the frequency of each ICD10 code in the dataset.

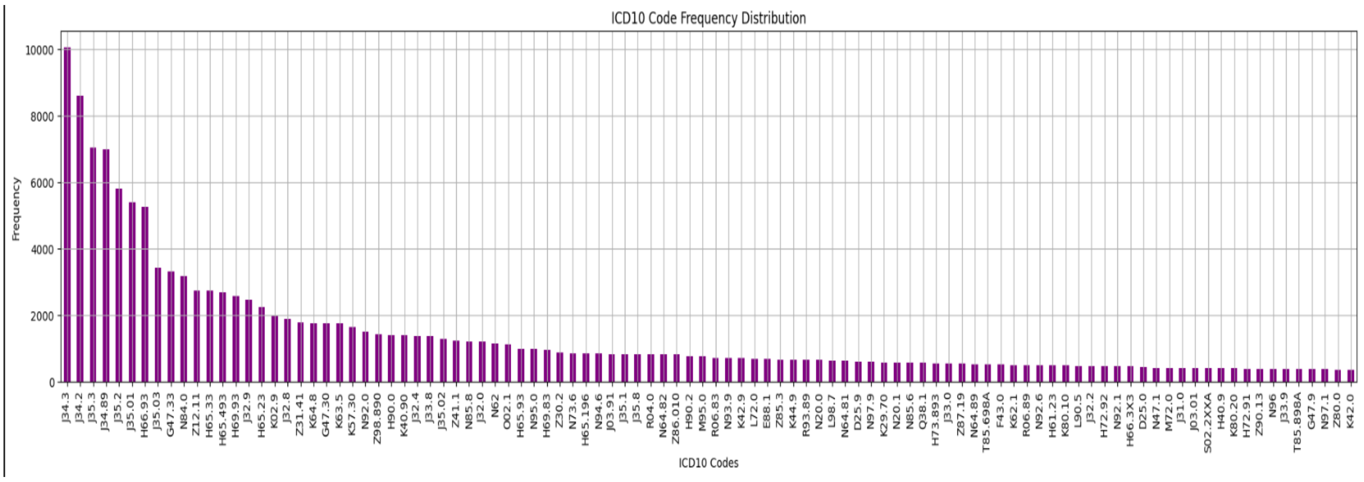
ICD10 codes frequency distribution corresponding to embeddings_1.npy:



No of Unique codes in ICD_Codes_1: 1373

ICD10 codes frequency distribution corresponding to embeddings_2.npy:

No of Unique codes in ICD_Codes_2: 1294



Given In the Problem statement that there exists 1400 unique ICD10 codes. From the plot of the Frequency distribution of labels it is evident that few of the codes are repeated more often as labels to the embeddings and majority of the codes frequency is lesser indicating class imbalance.

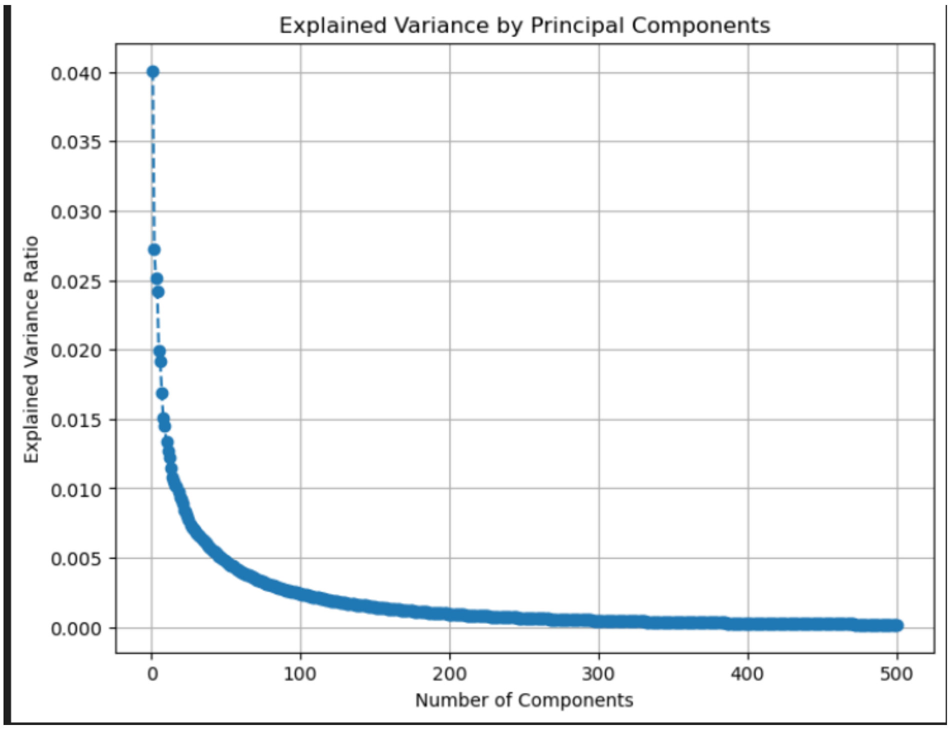
To handle the class imbalance problem, the codes which are underrepresented those embeddings and the corresponding labels are oversampled. A threshold of 100 is set and all those labels whose frequency is less than 100 is subject to oversampling. Around 27000 samples are obtained which is then stacked to the NumPy matrices respectively. The labels are also extended corresponding to the embeddings stacked such that number of rows of NumPy matrices and the number of codes remain constant.

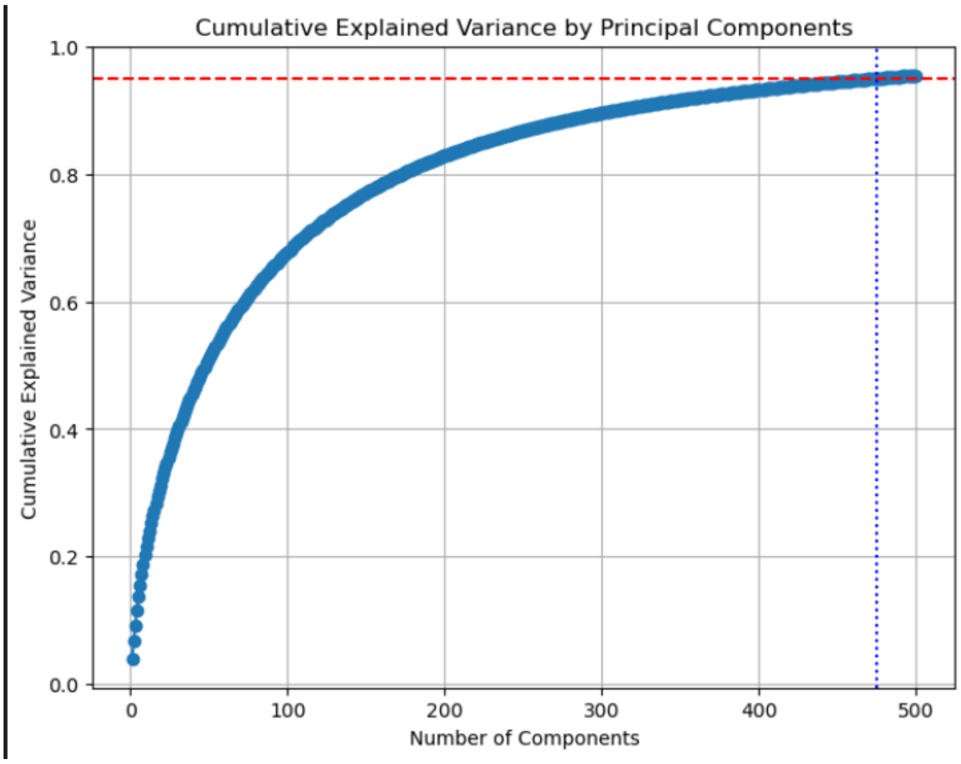
The embeddings and their corresponding labels from both sets, embeddings1.npy with icdcodes1.txt and embeddings2.npy with icdcodes2.txt, are merged to create the final training dataset. Both the embeddings and the corresponding labels are extended by the oversampled data of the underrepresented codes before being merged to obtain the final training data.

The embeddings are of 1024 dimension, this data can be represented in a much lower dimension which captures the same amount of information and variance. To find the appropriate lower dimension PCA is employed.

Using the explained variance and the cumulative explained variance of PCA the appropriate lower dimension is found.

For the given embeddings:





From the plots it is evident that around 500 PCA components is enough such that it captures more than 95% variance and produces less than 5% reconstruction error.

Therefore the embeddings are transformed into 500 PCA components and it is utilized as input features for the model training. The labels are converted into multi hot encoding labels of dimension 1401.

Model's implemented:

1.**Random forest wrapped by Multioutput classifier** module such the model adapts to Multi label classification. It is an ensemble method that uses the bagging (Bootstrap Aggregating) technique to improve classification performance by combining the predictions of multiple decision trees.

Two versions of the model is trained:

1. With 100 trees being trained on the pca components of the embeddings.
   The model achieved a f1 score of **0.105**
2. With 200 trees being trained on the pca components of the embeddings.
   The model achieved a f1 score of **0.114**

This highlights that the model is being overfit and not being able to generalize better on the test/ unseen data. Thus other models is explored.

The parameters are set after performing grid search and evaluating on the performance on the validation set.

2.**MLKNN (Multi label K nearest neighbours)**

It is an adaptation of the traditional k-Nearest Neighbor (kNN) algorithm designed for multi-label classification. Unlike standard kNN, which assigns a single label to each instance, MLKNN can assign multiple labels, making it useful for problems where each data point can belong to more than one class.

The model is trained with class weights being set to uniform and the no of neighbours set as 15. The model achieves an improved f1 score of **0.305** . The model is not able to improve generalization more on the test data.

The parameters are set after performing grid search and evaluating on the performance on the validation set.

3.**Logistic Regression wrapped by one vs rest classifier**

Logistic Regression is a binary classification algorithm that predicts the probability of a class label (1 or 0) for each instance. For Multi label problem one vs rest strategy can be employed to extend the binary classification nature of the logistic regression. In this strategy individual logistic regression model is trained for each class/ label. For each model, that specific label is considered as the "positive" label, while all other labels are treated as the "negative" label. Each Logistic Regression model learns to distinguish a particular label from all others independently. The classifier produces its own binary prediction, so an instance could potentially be classified into multiple classes/ label instances simultaneously.

The logistic regression model is initialized with l2 norm and C as 0.1 (Inverse of regularization strength). It controls the degree of penalty applied to the model for having large coefficients, helping prevent overfitting. The solver utilized is L-BFGS. The model is wrapped by one vs rest classifier module making it suitable to Multi label classification problem.

The model achieves an improved f1 score of **0.451**. This strategy is better compared to the adapted MLKNN approach and the Random forest classifier which is Feature bagging approach towards the Multi label classification problem.

These implementations demonstrate that traditional machine learning algorithms, with a few well chosen adaptations/ strategies, can achieve a macro F1 score above 0.4 in multi-label classification, bypassing the need for deep learning models.

Code :

```
import cupy as cp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patheffects as PathEffects
import matplotlib
from sklearn.utils import resample
from sklearn.preprocessing import MultiLabelBinarizer
#import seaborn as sns
from cuml.neighbors import KNeighborsClassifier  # Importing MLKNN from RAPIDS
from cuml.decomposition import PCA
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import warnings
from keras.layers import StringLookup
warnings.filterwarnings("ignore")

# Load embeddings and codes
embeddings1 = np.load("embeddings_1.npy")
embeddings2 = np.load("embeddings_2.npy")

with open("icd_codes_1.txt", 'r') as file:
   codes1 = [line.strip('\n').split(';') for line in file.readlines()]
with open("icd_codes_2.txt", 'r') as file:
   codes2 = [line.strip('\n').split(';') for line in file.readlines()]

Code1 = pd.Series([label for sublist in codes1 for label in sublist ])
```

```
Code2 = pd.Series([label for sublist in codes2 for label in sublist ])


print("No of Unique codes in ICD_Codes_1:",len(Code1.unique()))

print("No of Unique codes in ICD_Codes_2:",len(Code2.unique()))


# Concatinating both the codes

CODE = pd.concat([Code1, Code2])

print("No of Unique ICD10 Codes:",len(CODE.unique()))


CODE.value_counts()[:100].plot(kind = 'bar', figsize= (25,5), color='green')

plt.title('ICD10 Code Frequency Distribution') # Top 100 ICD10 codes which is frequently
repeated as labels to embeddings

plt.xlabel('ICD10 Codes')

plt.ylabel('Frequency')

plt.grid(True)

plt.xticks(rotation=90)

plt.show()


threshold = 100

underrepresented_class = CODE.value_counts()[CODE.value_counts() < threshold].index


# Extract samples (indices) that contain underrepresented codes

samples_to_oversample = [i for i, code_list in enumerate(codes1) if any(code in
underrepresented_class for code in code_list)]

len(samples_to_oversample)


X_resampled, Y_resampled = resample(embeddings1[samples_to_oversample], [codes1[i]
for i in samples_to_oversample], replace=True, n_samples=27000)


codes1.extend(Y_resampled)
```

```
# Concatenate original and oversampled data

Embeddings1 = np.vstack((embeddings1, X_resampled))

Codes1 = codes1


X_resampled1, Y_resampled1 = resample(embeddings2[samples_to_oversample], [codes2[i]
for i in samples_to_oversample], replace=True, n_samples=27000)

codes2.extend(Y_resampled1)

# Concatenate original and oversampled data

Embeddings2 = np.vstack((embeddings2, X_resampled1))

Codes2 = codes2


vocab = list(CODE.unique())

# Create the StringLookup layer with multi_hot output mode

lookup = StringLookup(vocabulary=vocab, output_mode='multi_hot')


import copy

C = copy.deepcopy(Codes1)

C.extend(Codes2) # contains codes1 followed by codes2 labels

len(C), len(codes2)+len(codes1) # to show both are of same length

Embeddings = np.vstack((Embeddings1, Embeddings2))

len(Embeddings), Embeddings.shape


p = PCA(n_components=500).fit(Embeddings)

explained_variance = p.explained_variance_ratio_


# Plot explained variance

plt.figure(figsize=(8, 6))

plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='--')

plt.title('Explained Variance by Principal Components')

plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.grid()
plt.show()


cumulative_variance = np.cumsum(explained_variance)

plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='-
-')
plt.title('Cumulative Explained Variance by Principal Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='r', linestyle='--')
plt.axvline(x = 475, color = 'b', linestyle='dotted')
plt.grid()
plt.show()

Embeddings_transformed = PCA(n_components=500).fit_transform(Embeddings)
max_length = max(len(sublist) for sublist in C)
Codes_padded = pad_sequences(C, maxlen=max_length, padding='post', value='',
dtype=object)
output = lookup(Codes_padded)

output.numpy().shape, Embeddings_transformed.shape
X_train, X_test, Y_train, Y_test = train_test_split(Embeddings_transformed, output.numpy(),
test_size=0.2, random_state=42)
x_train, x_val, y_train, y_val = train_test_split(X_train, Y_train, test_size=0.2,
random_state=42)

from cuml.multiclass import OneVsRestClassifier
from cuml.linear_model import LogisticRegression

#K = KNeighborsClassifier(weights='uniform', n_neighbors=5)

L = LogisticRegression(C=0.1, max_iter=10000)

K = OneVsRestClassifier(L)


K.fit(Embeddings_transformed, output.numpy())


from sklearn.metrics import f1_score

Y_pred_test = K.predict(X_test)

f1_macro = f1_score(Y_test, Y_pred_test, average='macro', zero_division=0)


print(f"F1 Macro Score for test data: {f1_macro}")



embeddings_test = np.load("test_data.npy")

pca = PCA(n_components=500).fit(Embeddings)  # Fit PCA on training data

Embeddings_test_transformed = pca.transform(embeddings_test)  # Apply same
transformation to test data



Test_pred = K.predict(Embeddings_test_transformed)


V = lookup.get_vocabulary()

def codes_prediction(pred):

    return [V[i] for i in np.where(pred == 1)[0]]



Test_decode = [codes_prediction(i) for i in Test_pred]


def clean_predictions(pred):
    if all(code == '[UNK]' for code in pred):
        return ''
    code_pred = [code for code in pred if code!='[UNK]']
    code_pred.sort()
    return ';'.join(code_pred)

Test_decode_cleaned = [clean_predictions(i) for i in Test_decode]
sub = pd.DataFrame({'id': range(1, len(Test_decode_cleaned)+1),
                'labels': Test_decode_cleaned})
sub.to_csv('submission15.csv', index=False)
```

Submitted by,

Manoj kumar. CM

DA24S018