DA5401: Contest Report

Nikshay Jain — MM21B044

November 14, 2024

Abstract

This report describes how to create a multi-label classification system, including data preprocessing, neural network model development/optimisation, and output generation. The system receives several label files, converts them to unique labels, and then stores them in a NumPy array using multi-hot encoding. Tensorflow is used to train a neural network model, which is then optimized by reducing early halting and learning rates. Finally, the projected labels for each test sample are recorded as a CSV file.

1 Introduction

Multi-label classification is a machine learning problem in which a single instance might belong to numerous classes. This study describes the design and implementation of a pipeline for preprocessing label input, creating a multi-label classification model, and generating predictions in a structured output file.

2 EDA and Data Preprocessing

The data preprocessing stage entails reading several label files, combining them into a list of unique labels, and encoding them as multi-hot vectors for training. The code for performing this task is provided below.

Listing 1: Data Preprocessing: Parsing and Encoding Labels

```
import numpy as np

def parse_label_file(filename, delimiter=';'):
    with open(filename, 'r') as f:
        lines = f.readlines()
    labels = [line.strip().split(delimiter) for line in lines]
    return labels

def create_label_to_index(labels):
    unique_labels = set(labels)
    label_to_index = {label: i for i, label in enumerate(unique_labels)}
```

```
    return label_to_index

def to_multi_hot(labels, label_to_index):
    vocab_size = len(label_to_index)
    multi_hot = np.zeros(vocab_size)
    for label in labels:
        index = label_to_index[label]
        multi_hot[index] = 1
    return multi_hot

# Loading label files and creating multi-hot encoded labels
labelsfile1 = "/content/drive/MyDrive/dal_endsem/icd_codes_1.txt"
labels1 = parse_label_file(labelsfile1)
labelsfile2 = "/content/drive/MyDrive/dal_endsem/icd_codes_2.txt"
labels2 = parse_label_file(labelsfile2)
labels = labels1 + labels2

all_labels = []
for label in labels:
    all_labels += label
label_to_index = create_label_to_index(all_labels)

multi_hot = [to_multi_hot(label, label_to_index) for label in labels]
y_train = np.array(multi_hot)
```

3 Model Development

Tensorflow (developed by Keras) is used to implement the multi-label classification model. The design includes of dense layers with ReLU activation and dropout regularization, which are followed by a sigmoid-activated output layer. This structure works well for multi-label tasks in which each label has an independent probability. Early stopping and learning rate scheduling are used to increase training efficiency while avoiding overfitting.

Listing 2: Model Development: Neural Network for Multi-Label Classification

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt

# Define model architecture
model = Sequential([
    Dense(1024, activation='relu', input_dim=1024),
    Dropout(0.3),
    Dense(512, activation='relu'),
```

```
    Dropout(0.3),
    Dense(1400, activation='sigmoid')
])

# Compile model
optimizer = Adam(learning_rate=0.001)
model.compile(loss='binary_crossentropy', optimizer=optimizer,
metrics=['accuracy'])

# Callbacks for training
early_stopping = EarlyStopping(monitor='val_loss', patience=8,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=8, min_lr=1e-6)

# Training model
history = model.fit(X_train, y_train, epochs=100, batch_size=64,
validation_split=0.2, callbacks=[early_stopping, lr_scheduler])

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Model Loss over Epochs")
plt.show()
```

The progress of the training process and variation of epochs is plotted to visualize the variation as follows:

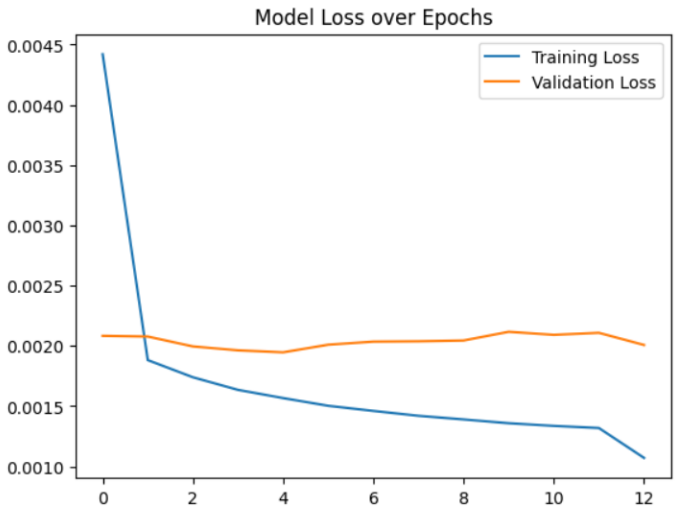

Figure 1: Progression of model training

4 Threshold Optimisation

We now optimise over the threshold of probability as predicted by the model for the final class assignment for the test data set and eventually find the optimum at 0.2 (instead of the default value of 0.5). This is done by cross validation.

Listing 3: Threshold Optimisation

```
# Define a range of thresholds to evaluate
thresholds = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

# Placeholder to store the best threshold and corresponding score
best_threshold = 0.5
best_f2 = 0

# Predict probabilities on the test set
y_pred_prob = model.predict(X_test)

# Loop over each threshold
for threshold in thresholds:
    # Convert probabilities to binary predictions based on the
    current threshold
    y_pred = (y_pred_prob > threshold).astype(int)

    # Evaluate performance using average micro-F2 score
    f2 = fbeta_score(y_test, y_pred, beta=2, average='micro')

    print(f"Threshold: {threshold:.1f}, Micro-F2 Score: {f2:.4f}")

    # Update best threshold if this one is better
    if f2 > best_f2:
        best_f2 = f2
        best_threshold = threshold

print(f"\nBest Threshold: {best_threshold}, Best Micro-F2 Score:
{best_f2:.4f}")

# Use the best threshold to make final predictions
final_y_pred = (y_pred_prob > best_threshold).astype(int)
```

5 Generating Output File

After all prredctions are ready as a nummpy array, its time to bring them to the required format for evaluation. It is facilitated by the below code snippet.

Listing 4: Output Generation: Writing Predicted Labels to CSV

```
def create_index_to_label(label_to_index):
    index_to_label = {v: k for k, v in label_to_index.items()}
    return index_to_label

index_to_label = create_index_to_label(label_to_index)

def create_txt_file(y_pred, index_to_label, filename="predicts.csv"):
    with open(filename, 'w') as f:
        f.write("id,labels\n")
        for i, prediction in enumerate(y_pred):
            labels = [index_to_label[j] for j, value in
            enumerate(prediction) if value == 1]
            labels = sorted(labels)
            f.write(f"{i+1},{';'.join(labels)}\n")

# Predicting and saving to file
y_pred_prob = model.predict(X_test)
threshold = 0.15
y_pred = (y_pred_prob > threshold).astype(int)
create_txt_file(y_pred, index_to_label)
```

6 Conclusion

In this code pipeline, this multi-label classification system efficiently integrates data preprocessing, neural network modelling, and result output. The system makes accurate predictions by employing a neural network model with suitable regularization and optimization approaches, and by arranging label data as multi-hot vectors. The results available are significantly better than simple machine learning classifiers like Random Forest/XGBoost as the complex feature label relations are learnt properly and prepared for deployment or additional analysis as CSV output file.

****