Ahmed Awadallah

Imputation and Multi Classification

Introduction

Hospitals have a direct interest in the future health of their patients while simultaneously also collecting and recording a lot of data on patients as part of their standard procedure. This provides a unique opportunity to create models to further benefit the patients. For this specific dataset the goal is to predict whether a patient will or will not be readmitted in the near future.

Methods

Our data comes from the DS 7333 Case Study 2. Our objective was to impute the data and then perform multiclass classification using logistic regression.

The dataset was shuffled to remove any chance of an inherited ordering impacting the model.

The features admission_type_id, discharge_disposition_id, and admission_source_id were coded with numerical values. A supplemental dataset supplied mappings from the code to descriptive states relating to the feature. All of these features had their codes mapped to their descriptive counterparts.

The dataset was explored for any possible na like value. Upon an intensive search na values were represented by “?” or “unknown/invalid”. The features that contained possible na variants were weight, race, diag_1, diag_2, diag_3, gender, payer_code, medical_specialty, admission_type_id, discharge_disposition_id, and admission_source_id. Upon inspection of how many NAs each of these features had the following percentages per feature.

- race : 2.234 %
- gender : 0.003 %
- weight : 96.858 %
- admission_type_id : 5.199 %
- discharge_disposition_id : 3.627 %
- admission_source_id : 6.663 %
- payer_code : 39.557 %
- medical_specialty : 49.082 %
- diag_1 : 0.021 %
- diag_2 : 0.352 %
- diag_3 : 1.398 %

Based on this data the following decisions about how to deal with these features were made.

The weight variable would be completely removed as too much of it was NA.

Race, gender, admission_type_id, discharge_disposition_id, admission_source_id, diag_1, diag_2, and diag_3 would all have their NA variants converted to the more suitable type None and then imputed using the highest frequency category within their respective feature. This method was chosen since there was only a small percentage of the row as NA.

Finally, payer_code and medical_specialty would keep their value of “?” as a special category indicating something was different about these rows. This was decided as too much was NA to consider imputing but not an overwhelming amount to have to completely remove the feature. Additionally, I’ve never used this method so I was curious about this approach.

The target feature, readmitted, was encoded so that it could be used in a logistic regression model.

Continuous variables were scaled using standardization to avoid the model biasing towards features with larger numbers.

Categorical features were one hot encoded so that they could be fed into a logistic regression model.

An optimal hyperparameter C was chosen using a custom made search function through a log scale. This hyperparameter was optimized using macro f1 score. Macro f1 since I wanted all the classes to be equally weighted for overall performance.

Using the optimal C a final model was trained and analyzed.

Results

Overall the performance of the model is not terrible but not great. Our final model achieved a macro f1 score of 0.4 with the following classification report.

Classification Report

~~~~~~

	precision	recall	f1-score	support
NO	0.63	0.85	0.72	54864
<30	0.49	0.05	0.10	11357
>30	0.53	0.39	0.45	35545
accuracy			0.60	101766
macro avg	0.55	0.43	0.42	101766
weighted avg	0.58	0.60	0.56	101766

Figure 1: Classification report for all three classes

The accuracy was above 60% which is atleast above randomly guessing the most popular class (the NO class). As a reminder the target for this model was hospital readmittance. “NO” means the patient was not readmitted, “<30” means the patient was readmitted in less than 30 days, and “>30” means the patient was readmitted in over 30 days.

For the “NO” class performance was decent. “NO” also happens to be the most populated class. The additional data may factor into why the model did a better job predicting it. Additionally “NO” is fundamentally different from the “<30” and “>30” classes.

For the “<30” class the performance overall is poor and it is the least populated class. Its poor performance is likely a mix of low data, lack of directly correlated features, and lack of a conceptual difference between “<30” and “>30”.

Lastly the “>30” class had ok performance. This category leaves a lot to be desired but the model is at least able to make some sensical predictions.

Below are the ROC curves for all three models and their AUC metrics. The ROC curve gives further evidence that these models are unfortunately unable to achieve a respectable True Positive Rate without sacrificing too many False Positives. This is further reflected in the AUC metrics being far from the maximal value of 1. An interesting observation to make is “<30” ROC curve is actually better than “>30” which implies its model is better as well. But we know from the classification report that the “>30” performed better.
~~~~~~

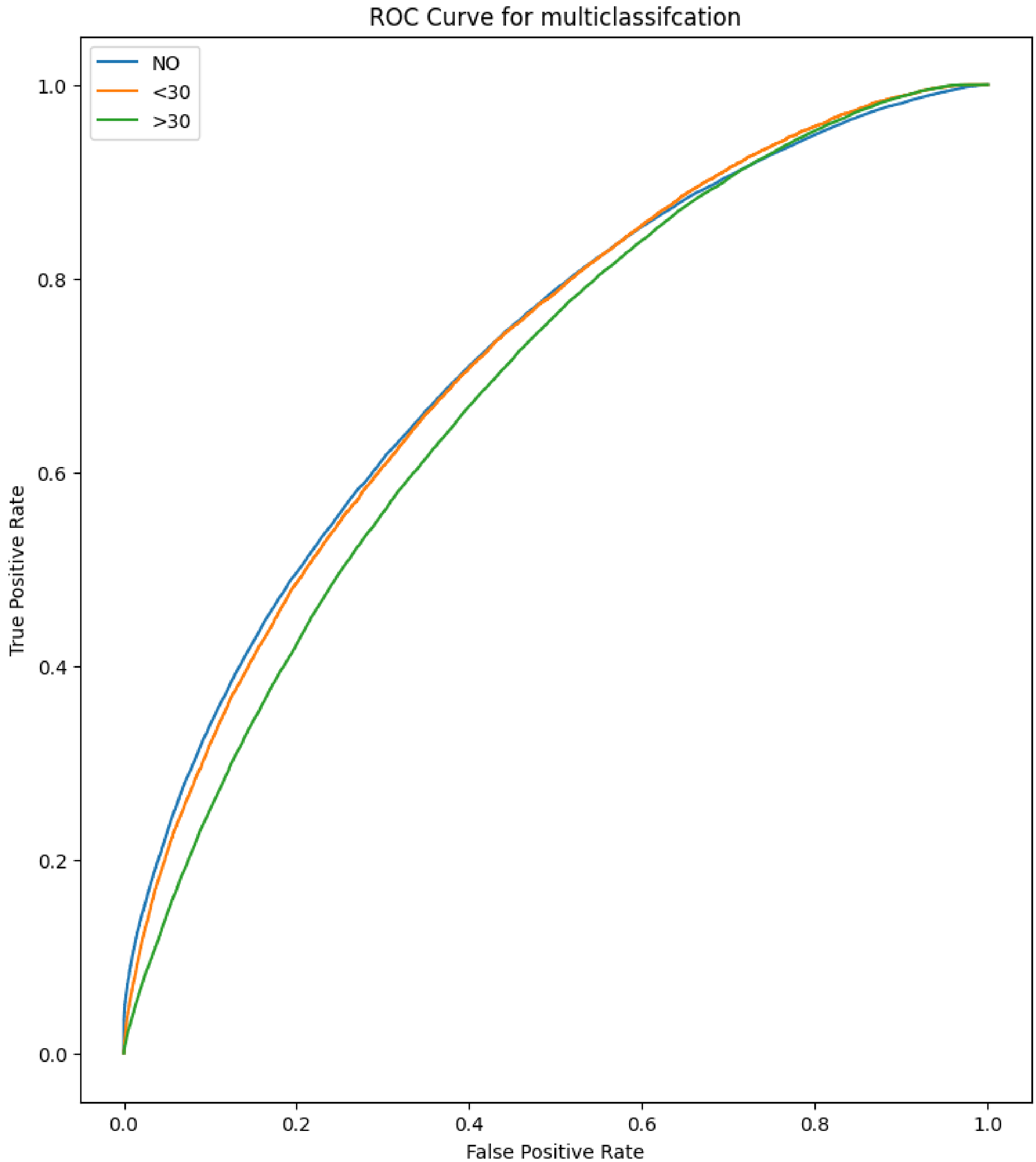

Figure 2: ROC Curve for each classes one versus rest logistic regression model

```
ROC/AUC Metrics
~~~~~~~~~~~~~~~~~~
NO AUC: 0.7199
<30 AUC: 0.7170
>30 AUC: 0.6878
```

Figure 3: AUC for each class

In addition to the above a confusion matrix for our models decision boundary is displayed below.

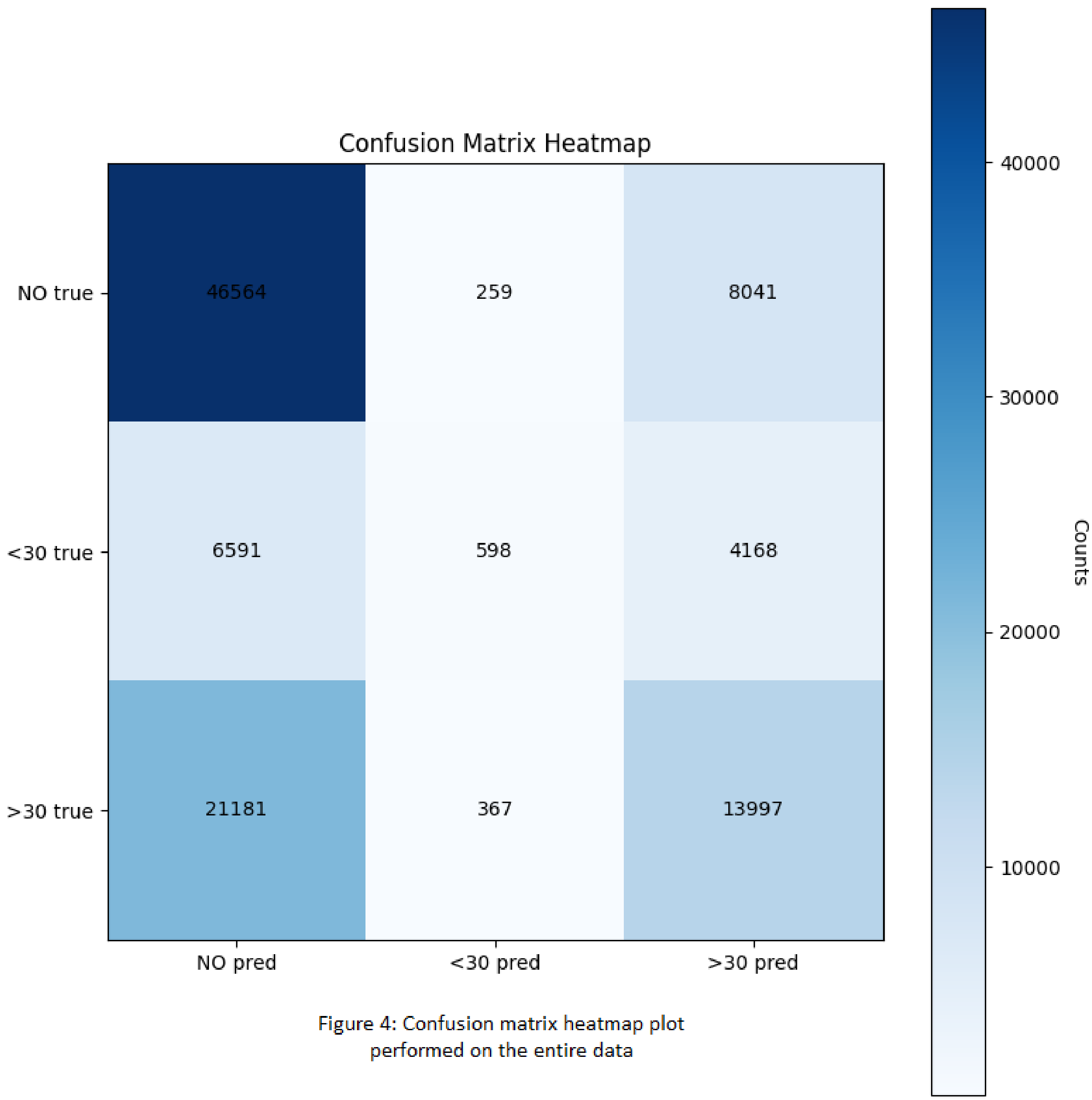

Figure 4: Confusion matrix heatmap plot
performed on the entire data

The “<30” class barely got any predictions (1224) despite having a support of 11357. This was known from the classification report but this further shows that this class was hard to predict..

There is some evidence to suggest that a better model could be made if we changed the target variable of the model to be “NO” and “readmitted”. This would be instead of splitting the readmitted category into two time checkpoints. The reason for this suggestion is that 4168 of our true “<30” were classified as “>30” as well as there was barely any “<30” predictions compared to the overall size of the dataset.

Lastly in the “NO” pred there were more “>30” and “<30” predictions than in the “<30” pred and “>30” pred columns. This displays that there is some difficulty distinguishing between “NO” and the other classes. Perhaps more useful features need to be researched for a future model.

The following is a figure of the top 8 most important weights for each class.

```
Most Relevant Coefs Report
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Class: NO | discharge_disposition_id_Expired: 3.9772
Class: NO | discharge_disposition_id_Hospice / medical facility: 1.4593
Class: NO | diag_2_513: -1.1361
Class: NO | diag_3_250.91: -1.1323
Class: NO | diag_1_526: -1.0922
Class: NO | admission_type_id_Trauma Center: 1.0744
Class: NO | diag_3_V60: -1.0294
Class: NO | diag_3_481: -1.0168
-------------------------------
Class: <30 | discharge_disposition_id_Expired: -2.0111
Class: <30 | diag_1_271: 1.3030
Class: <30 | diag_3_156: 1.2925
Class: <30 | diag_2_136: 1.2416
Class: <30 | diag_1_643: 1.1953
Class: <30 | diag_1_356: 1.1660
Class: <30 | diag_1_82: 1.1329
Class: <30 | diag_1_358: 1.1327
-------------------------------
Class: >30 | discharge_disposition_id_Expired: -1.9661
Class: >30 | discharge_disposition_id_Hospice / medical facility: -1.3181
Class: >30 | diag_3_481: 1.3135
Class: >30 | diag_3_150: -1.2852
Class: >30 | diag_1_583: 1.1994
Class: >30 | diag_2_813: -1.1275
Class: >30 | diag_2_136: -1.0879
Class: >30 | diag_1_199: -1.0714
-------------------------------
```

Figure 5 : Top 8 Weights absolute wieghts for each class.
Negative weight values restored to maintain proper interpretation.

From a glance it appears the various diag codes do have useful information about patient readmittance. A deeper study and analysis of the listed diag code is warranted and may provide additional knowledge about the domain.

For the named weights there are only three unique types. Discharge Disposition Expired, Discharge Disposition Hospice, and Discharge Disposition Trauma Center.

Discharge Disposition Expired refers to patients who are discharged from the hospital because they died within the hospital. To see a high association of this feature with “NO”(3.97) and a negative association with other two classes(-2.01, -1.966) is not surprising. A dead patient isn’t going to be readmitted to a hospital. It’s reassuring to see the model picked this feature up properly but it’s also a feature that is completely trivial to understand without a model.

Discharge Disposition Hospice refers to patients who were discharged into a hospice. A hospice is a place where terminally ill patients are taken care of until they die. Similar to discharge disposition expired we see similar weight associations for the “NO”(positive) and “>30” classes(negative) although it is noticeably missing from the “<30” class. This may indicate hospice patients may in fact bounce between the hospice and being readmitted but only in short time spans.

Discharge Disposition Trauma Center refers to patients who have been discharged into a trauma center. Trauma centers deal with patients who have suffered from trauma type injuries such as falls, car crashes, or firearms. For this feature we only see a positive association with the “NO” class. This association is possibly because traumatic injuries generally lead to two future conditions. Either the patient is able to be patched up and heals their injuries over time or the injuries are too traumatic and they pass away. Both cases would not lead to a hospital readmittance.

Finally, it's interesting to note that “<30” and “>30” largely picked different weights. While the model did perform poorly guessing the “<30” category, the difference in weights provides some evidence that it may be possible. If predicting readmittance of <30 days is important further research into better correlated features may be warranted.

A figure displaying the weights associated with race is shown below.

```
Finding the Weights of the Race Category
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Class: NO | race_Asian: 0.1828
Class: NO | race_Caucasian: -0.0290
Class: NO | race_Hispanic: 0.0794
Class: NO | race_Other: 0.1315
--------------------------------
Class: <30 | race_Asian: 0.0381
Class: <30 | race_Caucasian: 0.0196
Class: <30 | race_Hispanic: 0.0091
Class: <30 | race_Other: -0.0547
--------------------------------
Class: >30 | race_Asian: -0.2209
Class: >30 | race_Caucasian: 0.0094
Class: >30 | race_Hispanic: -0.0885
Class: >30 | race_Other: -0.0768
--------------------------------
```

Figure 3: Weights for each race in each model

Race can be a controversial feature to discuss but we felt it's important to not let cultural stigma prevent us from exploring what reality is telling us. If race is in fact an important factor for predicting patient readmittance there would be a direct benefit with recording this information.

The weights for each model tell us that some races do have some predictive power in hospital readmittance. For example asian has a positive association with not being readmitted while caucasian has near 0 relationship with all three classes. This discrepancy is curious and further research is probably warranted. A hypothesis test on the race should be conducted to see if the weights are in fact non zero as well as if the additional predictive power these features give is significant for a model. Only when both these conditions have been confirmed can we say race has a definitive and useful impact.

Conclusion

The goal of predicting hospital readmittance was achieved but there is a lot of improvement that must be made if the model is to be considered exceptional. Regardless, interesting information was learned about the features embedded within the model and their exploration may provide further insight into patient care.

Code

Appendix A

```python
import pandas as pd
import numpy as np
import math
import sys
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
import time
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn import metrics as mt

# Your case study is to build a classifier using logistic regression to
# predict hospital readmittance. There is missing data that must be imputed.
# Once again, discuss variable importances as part of your submission.


INVALID_INPUT = 1


def try_to_read_input_csvs(inputs):
    try:
        read_input_csvs(inputs)
    except Exception:
        print("This script expects the two csv files associated with Case "
              "Study1. You either input nothing, 1 file path, or one or more "
              "of your inputs contains an invalid path.")
        sys.exit(INVALID_INPUT)


def read_input_csvs(inputs):
    global df1, df2
    df1_path = inputs[1]
    df2_path = inputs[2]
    df1 = pd.read_csv(df1_path)
    df2 = pd.read_csv(df2_path)
```

```python
def get_mapping(range, id_map_df):
    map_dict = {}
    for i in range:
        id = int(id_map_df.loc[i][0])
        descrip = id_map_df.loc[i][1]
        if type(descrip) == float and math.isnan(descrip):
            descrip = None
        map_dict[id] = descrip
    return map_dict
```

```python
def logreg_cv_compare(x, y, print_progress=False, n_jobs=1):
    best_C = 0
    best_f1 = 0
    first = True
    iter_count = 20
    report_freq = int(iter_count/20)
    i = 0
    t1 = time.time()
    for param_C in np.logspace(-6, 0, iter_count):

        model = LogisticRegression(multi_class='multinomial',
                                   solver='lbfgs',
                                   C=param_C,
                                   n_jobs=6,
                                   max_iter=1000)

        f1_cv = cross_val_score(model, x, y, cv=5,
                                scoring='f1_macro')

        f1 = sum(f1_cv)/len(f1_cv)

        if first:
            first = False
            best_C = param_C
            best_f1 = f1
        else:
            if (f1 > best_f1):
                best_C = param_C
                best_f1 = f1

        if ((i+1) % report_freq == 0) and print_progress:
            print(f"Iteration {i+1}. "
                  f"Percent done = {(i+1)/iter_count*100:.4f}%")

        i += 1
    t2 = time.time()
    elapsed_time = t2-t1

    report_str = f"Best f1={best_f1:.4f}, "
    report_str += f"C={best_C}, time={elapsed_time:.4f}s\n"

    return best_C, report_str
```

```python
def get_top_n_coef_multiclass_logreg(feature_list, coef_list, n=5):
    feature_scores_abs = {}
    top_n_per_class = []
    feature_scores = {}
    class_count = coef_list.shape[0]
    for class_id in range(class_count):
        for i in range(len(feature_list)):
            feature = feature_list[i]
            coef = coef_list[class_id, i]
            feature_scores_abs[feature] = abs(coef)
            feature_scores[feature] = coef

        feature_scores_abs = dict(sorted(feature_scores_abs.items(),
                                         key=lambda item: item[1], reverse=True))
        top_n_features = list(feature_scores_abs.keys())[0:n]

        top_feature_scores = {}
        for feature in top_n_features:
            top_feature_scores[feature] = feature_scores[feature]

        top_n_per_class.append(top_feature_scores)

    return top_n_per_class


def print_top_coefs_multiclass_logreg(top_n_coefs, id_targ_map):
    str_report = ""
    for class_id in range(len(top_n_coefs)):
        feature_top_n = top_n_coefs[class_id]
        class_name = id_targ_map[class_id]
        for feature in feature_top_n.keys():
            coef = feature_top_n[feature]
            str_report += f"Class: {class_name} | {feature}: {coef:.4f}\n"
        str_report += "--------------------------------\n"
    return str_report
```

```python
def plot_heatmap(data,
                 xlab,
                 ylab,
                 size=9,
                 title="untitled heatmap",
                 cbar_label="untitled",
                 save_name="untitled.png"):

    fig, ax = plt.subplots(figsize=(size, size))
    im = ax.imshow(data, cmap=plt.cm.Blues)

    for i in range(len(ylab)):
        for j in range(len(xlab)):
            ax.text(j, i, data[i, j],
                    ha="center", va="center", color="black")

    # Show all ticks and label them with the respective list entries
    ax.set_xticks(np.arange(len(xlab)), labels=xlab)
    ax.set_yticks(np.arange(len(ylab)), labels=ylab)

    cbar = ax.figure.colorbar(im, ax=ax)
    cbar.ax.set_ylabel(cbar_label, rotation=-90, va="bottom")

    ax.set_title(title)
    fig.tight_layout()
    plt.savefig(save_name, bbox_inches='tight')
    plt.clf()


if __name__ == '__main__':
    log_file = open("log.txt", "w")
    log_file.write("Case Study 2 Report\n\n")

    inputs = sys.argv
    try_to_read_input_csvs(inputs)
```

```python
    # Find which dataframe contains the cts variables that need to be scaled
    if (df1.shape == (101766, 50)) and (df2.shape == (67, 2)):
        raw_data = df1
        id_map_df = df2
    elif (df2.shape == (101766, 50)) and (df1.shape == (67, 2)):
        raw_data = df2
        id_map_df = df1
    else:
        raise Exception("One or both of dataframes has unexpected shape")

    # Shuffle raw data
    raw_data = raw_data.sample(frac=1)

    # Use metadata to provide useful descriptions for some features
    admission_type_map = get_mapping(range(0, 8), id_map_df)
    discharge_map = get_mapping(range(10, 40), id_map_df)
    admission_source_map = get_mapping(range(42, 67), id_map_df)

    raw_data["admission_type_id"] = raw_data[
        "admission_type_id"].map(admission_type_map)
    raw_data["discharge_disposition_id"] = raw_data[
        "discharge_disposition_id"].map(discharge_map)
    raw_data["admission_source_id"] = raw_data[
        "admission_source_id"].map(admission_source_map)

    # Data Cleaning Decisions
    # Remove weight as a feature as too much of it is missing
    clean_df = raw_data.drop(["weight"], axis=1)
    # Remove the following features as they should have no relationship with
    # target
    features_to_remove = ["patient_nbr", "encounter_id"]
    clean_df = clean_df.drop(features_to_remove, axis=1)
    # Impute desired features
    clean_df["gender"] = clean_df["gender"].replace("Unknown/Invalid", "?")
    features_to_impute = ["race", "diag_1", "diag_2", "diag_3", "gender",
                          "admission_type_id", "admission_source_id",
                          "discharge_disposition_id"]
    # All these categories use "?" to signify na data so replace with None
    for feature in features_to_impute:
        clean_df[feature] = clean_df[feature].replace("?", None)
    imputer = SimpleImputer(missing_values=None, strategy="most_frequent")
    for feature in features_to_impute:
        imputed_feature = imputer.fit_transform(clean_df[[feature]]).ravel()
        clean_df[feature] = imputed_feature
```

```python
    # Encode Target
    readmitted_to_id_map = {"NO": 0, "<30": 1, ">30": 2}
    id_to_readmitted_map = {0: "NO", 1: "<30", 2: ">30"}
    clean_df["readmitted"] = clean_df["readmitted"].map(readmitted_to_id_map)

    # Specifying cts v categorical features
    cts_features = ["time_in_hospital",
                    "num_lab_procedures",
                    "num_procedures",
                    "num_medications",
                    "number_outpatient",
                    "number_emergency",
                    "number_inpatient",
                    "number_diagnoses"]

    categ_features = []
    for feature in list(clean_df):
        if feature not in cts_features:
            categ_features.append(feature)

    # Scaling Cts Data
    scaler = StandardScaler()
    clean_df[cts_features] = scaler.fit_transform(clean_df[cts_features])

    # Split data into train data and target data
    y = clean_df["readmitted"]
    x = clean_df.drop("readmitted", axis=1)

    #  One hot encode
    x = pd.get_dummies(x, drop_first=True)

    # Subset data for demonstration purposes
    subset_count = 500
    print("Data subsetting is ACTIVE, only", subset_count,
          "samples being used to train")
    print("This is just to demonstrate the code behavior quickly")
    x = x.head(subset_count)
    y = y.head(subset_count)
```

```python
    # Search optimal regularization C
    print("Finding optimal model hyperparameters...")
    print("")
    optimal_c, report_str = logreg_cv_compare(
        x,
        y,
        print_progress=True,
        n_jobs=4)
    log_file.write("Optimization Report\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(report_str)
    log_file.write("\n")

    # Final Model
    final_model = LogisticRegression(multi_class='multinomial',
                                     solver='lbfgs',
                                     C=optimal_c,
                                     max_iter=1000,
                                     n_jobs=6)
    final_model.fit(x, y)
    y_pred = final_model.predict(x)

    # Final Model Analysis

    # Top Coefs Per Target Feature Class
    class_top_n = get_top_n_coef_multiclass_logreg(
        list(x),
        final_model.coef_,
        n=8)

    log_file.write("Most Relevant Coefs Report\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(
        print_top_coefs_multiclass_logreg(
            class_top_n,
            id_to_readmitted_map
        )
    )
    log_file.write("\n")
```

```python
    # Print Race Coef
    log_file.write("Finding the Weights of the Race Category\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    races = ['race_Asian',
             'race_Caucasian',
             'race_Hispanic',
             'race_Other',
             'race_AfricanAmerican']
    class_race_score = []
    class_count = final_model.coef_.shape[0]
    for class_id in range(class_count):
        race_feature_score = {}

        for race in races:
            if race in list(x):
                race_ind = list(x).index(race)
                coef = final_model.coef_[class_id, race_ind]

                race_feature_score[race] = coef

        class_race_score.append(race_feature_score)
    log_file.write(
        print_top_coefs_multiclass_logreg(
            class_race_score,
            id_to_readmitted_map
        )
    )
    log_file.write("\n")

    # Classif Report
    log_file.write("Classification Report\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(
        classification_report(
            y,
            y_pred,
            target_names=id_to_readmitted_map.values()
        )
    )
    log_file.write("\n")

    # Conf Matrix Plot
    conf_mat = confusion_matrix(y, y_pred)
```

```python
    base_labels = list(id_to_readmitted_map.values())
    label_true = []
    label_pred = []
    for i in range(len(base_labels)):
        true_label = base_labels[i] + " true"
        pred_label = base_labels[i] + " pred"
        label_pred.append(pred_label)
        label_true.append(true_label)

    plot_heatmap(conf_mat,
                 label_pred,
                 label_true,
                 size=8,
                 title="Confusion Matrix Heatmap",
                 cbar_label="Counts",
                 save_name="conf_heatmap.png")

    # ROC Curve and AUC
    log_file.write("ROC/AUC Metrics\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    y_pred_prob = final_model.predict_proba(x)
    for i in range(y_pred_prob.shape[1]):
        y_label = id_to_readmitted_map[i]
        preds = y_pred_prob[:, i]
        fpr, tpr, thresholds = mt.roc_curve(y, preds, pos_label=i)
        plt.plot(fpr, tpr, label=y_label)
        title_str = 'ROC Curve for multiclassifcation'
        plt.title(title_str)
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        log_file.write(f"{y_label} AUC: {mt.auc(fpr, tpr):.4f}\n")

    plt.legend()
    plt.savefig("roc_plot.png", bbox_inches='tight')
    plt.clf()
    log_file.write("\n")
    log_file.close()
```