**Case Study II**

Randy Kim, Kati Schuerger, Will Sherman

September 19, 2022

**1 Introduction**

**Objective**

The objective of this case study was to build a classifier using logistic regression to predict readmission of the patient within 30 days of initial hospitalization. In addition, our analysis sought to identify which variables carry the most importance in predicting hospital readmittance and address imputation of missing values in the provided dataset.

**Dataset**

The dataset was provided by the medical community, and the subject matter concerns details related to patients with diabetes. The goal of healthcare providers was to limit the number of patients that require readmission. The analysis in this case study may be useful in helping to identify patterns that make a patient more or less likely to be readmitted.

Data points include individual demographic, personal physical, and hospital record data. Hospital record data include such features as admission data, discharge data, length of stay, medical specialty, number of labs/procedures/medications, the results of these interventions, and clinical diagnoses.

**2 Methods**

**Normalization**

We used SKLEARN StandardScaler to transform the data to measure features on the same scale. Standard Scaler is a method of standardizing the data, transforming it so that each feature has mean of 0 and a standard deviation of 1.

**Diagnoses**

One feature we decided to work on initially was the diagnoses (*i.e.*, features *diag\_1*, *diag\_2*, and *diag\_3*) from the original dataset. The feature spaces for these were very large: 716, 748, and 789 categories, respectively. Based on data from BioMed Research International[[1]](#footnote-2), we were able to establish these diagnoses as corresponding to values from the International Classification of Diseases, Ninth Revision, Clinical Modification (ICD-9). We opted to transform these numeric and string vectors into grouped categories based on ICD-9 classifications[[2]](#footnote-3).

**Missing data**

Upon initial review of the data, we identified multiple missing values. One reason for some of the missing data was that this information was gathered over 10 years ago, from several different hospitals, and it was known that the data have gaps.

To preserve as much information as possible, additional analysis was conducted to identify possible options for how we might be able to impute some (if not most) of these missing values.

**Imputation of missing values**

One of the primary considerations from the clinician's standpoint will likely be in-house labs and diagnoses. The features for labs (*i.e.*, *num\_lab\_procedures*, *num\_procedures*, *num\_medications*, *number\_outpatient*, *number\_emergency*, *number\_inpatient*) did not include any missing values. However, the diagnoses—particularly secondary and additional secondary diagnoses—did have some missing information. For these missing values we opted to impute using the mode for each of the features.

Of note was that the diagnoses features did not appear to have correlation in their missingness. We expected that this feature set was missing completely at random or that we were unable to identify the relationship between missing values based on the given data.

The weight feature was missing in 96.86% of the collected data. Therefore, we opted to exclude this from the possible feature space, as this may have caused concerns about the confidence of features in the study. We also excluded the patient number as this should not impact modeling on readmission.

Payer code appeared to have values missing not at random. There was a high negative correlation between missing payer codes and encounter ID (Figure 1). The other features that included missing data were race, medical specialty, admission type, discharge disposition, and admission source. For these, mode imputation was chosen again based on underlying distribution of the data.

**Ethical considerations regarding patient race**

When conducting statistical analysis, it is important to be mindful of any possibilities for bias or potential discrimination against (or in favor of) a portion of the population being analyzed.

One such feature in this dataset was race. If this analysis were intended to build a model for setting insurance premiums, sorting individuals by race could certainly be considered as discriminatory or biased. For this case, however, we believe that we do not encounter this difficulty of bias. Here, we are talking about patients and diabetes in relationship to hospital readmission. The medical community is aware that diabetes affects different demographics in different ways. For this reason, race may be a useful feature in helping determine likelihood of the patient to return to the hospital within 30 days.

**Logistic regression and the Sigmoid function**

At a very high-level, logistic regression is a method used to build models where the task is one of categorization, meaning the options for the target variable is a discrete value (for example: yes or no, 0 or 1, color: red or white or blue).

Our target variable is categorical, with two options. Either yes, the patient is likely to be readmitted within 30 days; or no, the patient is not likely to be readmitted within 30 days. Classification will be based on the probability score generated by the model.

Logistic regression makes use of the Sigmoid function, which seeks to determine a probability score that the target variable belongs to one of the category options. The analyst then applies a threshold value to determine the cutoff for what determines which value-option the record belongs to.

**Selecting threshold for category in logistic model**

The go-to cutoff value is 50%, meaning that any observation scoring lower than 50% would belong to class 0, and anything over 50% belongs to class 1.

**Train/Test/Validate split**

Before we began modeling, we randomized the arrangement of the data samples and separated the data into 2 groups: a training dataset (to fit the model), and a validation dataset (holdout group of 20%) that was used to measure model performance on unseen data—evaluating whether the model generalizes well.

**Scoring metric**

The task was to create a logistic regression model for readmission predictions. We used accuracy scores in combination with 10-fold cross-validation to get model scores and compare models. Cross-validation is a way to assess different portions of the data using resampling to identify the model parameters. This was used with regularized logistic regression to prevent over-fitting problem. We iterated through the values of *C* (the penalty term) to generate optimal loss function prior to fitting the regularized logistic regression.

**3 Results**

**Missing values**

We identified several features with missing values. The payer\_code feature had a distinct progression of missingness (Figure 1). The level of correlation between missingness and time series is difficult to quantify and beyond the scope of this particular project.

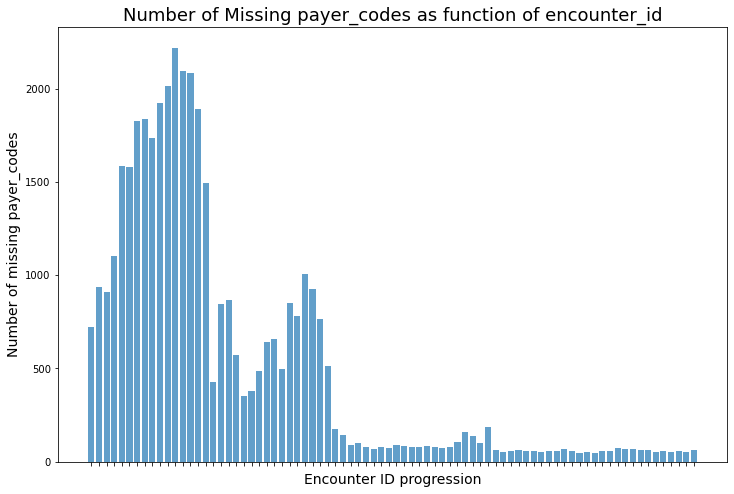


Figure 1. For increasing encounter\_id, the number of missing payer\_code values appears to decrease.

The diagnoses—particularly secondary and additional secondary diagnoses—had missing information (*diag\_1* had 21, *diag\_2* had 358, and *diag\_3* had 1423 missing values). Due to the relatively small amount of missing data, we proceeded to do mode imputation for each of these: *diag\_1* was circulatory disease, *diag\_2* was metabolic disease, and *diag\_3* was diabetes.

The optimization of the L2 penalty coefficient (*C* in this case) was generated by iterating over a 100 sample values in log space between -10 and 10. The optimal value for this hyperparameter was 0.00029.

The imputation for each of the following was also implemented using mode after distribution assessment: *race* as Caucasian, *medical specialty* as Internal Medicine, *admission type* as Emergency, *discharge disposition* as Discharged to home, and admission source

**Feature importance**

Based on the individual demographic, personal physical, and hospital record data that were provided, we pre-selected features of interest prior to model selection. Then, using L2 regularization, we identified the top ten features for the logistic model (Table 1).

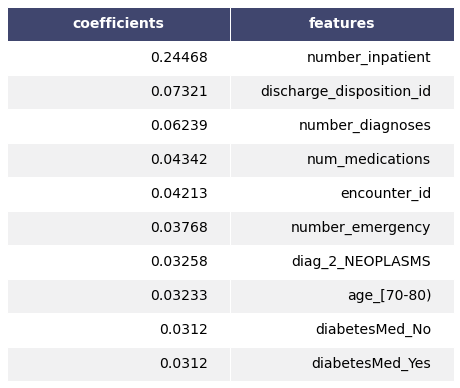


Table 1. Model coefficients by weight.

Table 1. Provides the top 10 coefficients for the Logistic Regression model on diabetes readmission.

The most important feature appeared to be the number of inpatient visits by the patient in the year preceding the current encounter. The discharge disposition also appeared to have a significant impact along with the number of diagnoses and the number of medications the patient was receiving.

Table 2. Initial Accuracy Scores.

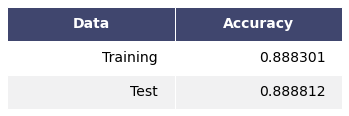


Table 2. General accuracy scoring by the logistic model against the whole training dataset and the holdout (Test) dataset.

The accuracy score of the model against the training data was 88.83% accurate (Table 2). Based on the initial split, the model outperformed this accuracy on the smaller test set with a score of 88.88% accuracy. Assessment against other random seeds (43 randomly selected seeds) showed that the range of the accuracies in test performance were wider with median 88.43% (Table 3).

Table 3. Random Seed Accuracy Scores.

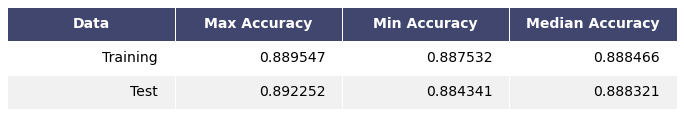


Table 3. Range and behavior of accuracy scores based on random seed splitting of train/test split.

**4 Conclusion**

The regularized logistic regression model performed well for both training and unseen data with 88.83% accuracy score. We could improve this accuracy score by refining the variables such as diabetesMed\_No and diabetesMed\_Yes as these were given as coefficients. We can also try modeling by predictive mean match imputation rather than mode imputation. For example, we could predict missing data of medical\_specialty by admission\_type\_id.

Future studies could try to divide patients by medical specialty or other categorical variables. This would need further investigation to see if there is evidence suggesting that categorical variables and readmission show correlations.

Finally, we would want to meet with the investigators to receive input on variable selection methods for further refinement of the model. Notably, *encounter\_id* is included in the model. In a perfect world, when someone is being discharged, the date of their admission shouldn’t dictate their health outcome (over the span of time this data was collected). However, that is under the assumption that *encounter\_id* is linked to date information and not location information. Regardless, this variable is of particular interest and should be included with caution as there are also ethical concerns if this is location-based data. Namely, predicting readmission on locale if the demographics are different.

Code

# %%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import sklearn

# %%

### Will - read data ###

data\_1 = pd.read\_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 - Quantifying the World\Case Study 2\dataset\_diabetes\diabetic\_data.csv')

### Randy - read data ###

# data\_1 = pd.read\_csv("diabetic\_data.csv")

### Kati - read data ###

# data\_1 =

print(data\_1.shape)

print(data\_1.columns)

data\_1.head(10)

# %%

data\_1.isnull().any()

# %%

print(len(data\_1[data\_1['diag\_1']=='?']))

print(len(data\_1[data\_1['diag\_2']=='?']))

print(len(data\_1[data\_1['diag\_3']=='?']))

# %%

from collections import Counter

diag\_1 = Counter(list(data\_1['diag\_1'])).most\_common(1)[0][0]

diag\_2 = Counter(list(data\_1['diag\_2'])).most\_common(1)[0][0]

diag\_3 = Counter(list(data\_1['diag\_3'])).most\_common(1)[0][0]

data\_1['diag\_1'] = data\_1['diag\_1'].apply(lambda x : diag\_1 if x == '?' else x)

data\_1['diag\_2'] = data\_1['diag\_2'].apply(lambda x : diag\_2 if x == '?' else x)

data\_1['diag\_3'] = data\_1['diag\_3'].apply(lambda x : diag\_3 if x == '?' else x)

# credit for above code goes to Sanchin Ranveer

# https://medium.com/analytics-vidhya/diabetes-130-us-hospitals-for-years-1999-2008-e18d69beea4d

# %%

print(len(data\_1[data\_1['weight']=='?']), 'out of', len(data\_1['weight']))

print('weight values missing at {}%'.format(round(98569/101766\*100, 2)))

# %%

data\_1 = data\_1.drop(['weight'], axis=1)

data\_1 = data\_1.drop(['patient\_nbr'], axis=1)

# %%

data\_1 = data\_1.replace('?', np.NaN)

# %%

data\_1.isnull().any()

# %%

# List of vectors with np.NaN values that need to be imputed

vectors\_w\_NAs = data\_1.columns[data\_1.isnull().any()].tolist()

print(vectors\_w\_NAs)

# %%

print(len(data\_1[data\_1['gender']=='Unknown/Invalid']))

data\_1\_update = data\_1.drop(index = data\_1[data\_1['gender']=='Unknown/Invalid'].index)

# %%

# Missing Not At Random for Payer Code

sorted\_df = data\_1\_update.sort\_values(by=['encounter\_id'])

sorted\_df['bins'] = pd.cut(data\_1\_update['encounter\_id'], 80)

sorted\_df['na\_counts'] = sorted\_df['payer\_code'].isnull()

grouped = sorted\_df.groupby(sorted\_df['bins'], as\_index=False).sum()

x = grouped['bins'].astype(str)

y = grouped['na\_counts']

fig, ax = plt.subplots(figsize=(12,8))

plt.xlabel('Encounter ID progression', fontsize=14)

plt.ylabel('Number of missing payer\_codes', fontsize=14)

plt.title('Number of Missing payer\_codes as function of encounter\_id', fontsize=18)

ax.set\_xticklabels([])

plt.bar(x, y, alpha=.7)

# %%

data\_1\_update['diag\_1'] = data\_1\_update['diag\_1'].str.split('.').str[0]

data\_1\_update['diag\_2'] = data\_1\_update['diag\_2'].str.split('.').str[0]

data\_1\_update['diag\_3'] = data\_1\_update['diag\_3'].str.split('.').str[0]

# %%

def setall(d, keys, value):

for k in keys:

d[k] = value

numeric\_codes = {}

setall(numeric\_codes, range(1, 140), 'INFECTIOUS AND PARASITIC DISEASES')

setall(numeric\_codes, range(140, 240), 'NEOPLASMS')

setall(numeric\_codes, range(240, 280), 'ENDOCRINE, NUTRITIONAL AND METABOLIC DISEASES, AND IMMUNITY DISORDERS')

setall(numeric\_codes, range(280, 290), 'DISEASES OF THE BLOOD AND BLOOD-FORMING ORGANS')

setall(numeric\_codes, range(290, 320), 'MENTAL, BEHAVIORAL AND NEURODEVELOPMENTAL DISORDERS')

setall(numeric\_codes, range(320, 390), 'DISEASES OF THE NERVOUS SYSTEM AND SENSE ORGANS')

setall(numeric\_codes, range(390, 460), 'DISEASES OF THE CIRCULATORY SYSTEM')

setall(numeric\_codes, range(460, 520), 'DISEASES OF THE RESPIRATORY SYSTEM')

setall(numeric\_codes, range(520, 580), 'DISEASES OF THE DIGESTIVE SYSTEM')

setall(numeric\_codes, range(580, 630), 'DISEASES OF THE GENITOURINARY SYSTEM')

setall(numeric\_codes, range(630, 680), 'COMPLICATIONS OF PREGNANCY, CHILDBIRTH, AND THE PUERPERIUM')

setall(numeric\_codes, range(680, 710), 'DISEASES OF THE SKIN AND SUBCUTANEOUS TISSUE')

setall(numeric\_codes, range(710, 740), 'DISEASES OF THE MUSCULOSKELETAL SYSTEM AND CONNECTIVE TISSUE')

setall(numeric\_codes, range(740, 760), 'CONGENITAL ANOMALIES')

setall(numeric\_codes, range(760, 780), 'CERTAIN CONDITIONS ORIGINATING IN THE PERINATAL PERIOD')

setall(numeric\_codes, range(780, 800), 'SYMPTOMS, SIGNS, AND ILL-DEFINED CONDITIONS')

setall(numeric\_codes, range(800, 1000), 'INJURY AND POISONING')

def new\_setall(d, letter, keys, value):

for k in keys:

d[letter + str(k)] = value

extra\_codes={250: 'DIABETES RELATED'}

new\_setall(extra\_codes, 'E', range(800,999), 'OTHER EXTERNAL CAUSES OF INJURY OR POISONING')

new\_setall(extra\_codes, 'V0', range(1,10), 'HEALTH HAZARD - COMMUNICABLE DISEASE')

new\_setall(extra\_codes, 'V', range(10,40), 'FAMILY HISTORY OR REPRODUCTIVE DEVELOPMENT ISSUE')

new\_setall(extra\_codes, 'V', range(40,50), 'NERVE/BRAIN ISSUES, ORGAN TRANSPLANT, MACHINE SUPPORT, OTHER')

new\_setall(extra\_codes, 'V', range(50,60), 'POST-PROCEDURE CARE')

new\_setall(extra\_codes, 'V', range(60,70), 'ECONOMIC, FAMILY, CONVALESCENCE, PALLIATIVE CARE OR FOLLOW-UP EXAM')

new\_setall(extra\_codes, 'V', range(70,90), 'GENETICS, HORMONAL, UNREPORTED DIAGNOSIS, OTHER')

new\_setall(extra\_codes, 'V', range(90,99), 'MULTIPLE GESTATION PLACENTA STATUS')

# diagnoses\_codes = numeric\_codes | extra\_codes

# %%

diagnoses\_codes = {\*\*numeric\_codes, \*\*extra\_codes}

# %%

def transform\_bad\_vals(vector):

for ind, code in vector.iteritems():

try:

code = int(code)

if code not in diagnoses\_codes.keys():

raise ValueError('Incorrect key-value provided: {} {}'.format(ind, code))

else:

vector[ind] = diagnoses\_codes[code]

except:

if code not in diagnoses\_codes.keys():

raise ValueError('Other error: {} {}'.format(ind, code))

else:

vector[ind] = diagnoses\_codes[code]

return vector

# %%

pd.options.mode.chained\_assignment = None # default='warn'

data\_1\_update['diag\_1'] = transform\_bad\_vals(data\_1\_update['diag\_1'])

data\_1\_update['diag\_2'] = transform\_bad\_vals(data\_1\_update['diag\_2'])

data\_1\_update['diag\_3'] = transform\_bad\_vals(data\_1\_update['diag\_3'])

data\_1\_update.loc[:, ['diag\_1','diag\_2','diag\_3']].head()

# %%

set(data\_1\_update['admission\_type\_id'])

# %%

### Will - read data ###

data\_2 = pd.read\_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 - Quantifying the World\Case Study 2\dataset\_diabetes\IDs\_mapping.csv')

### Randy - read data ###

# data\_2 = pd.read\_csv("IDs\_mapping.csv")

### Kati - read data ###

# data\_2 =

pd.set\_option('display.max\_rows', 70)

print(data\_2.shape)

print(data\_2.columns)

data\_2

### Frick this data it's messy \*ugh\* ###

### Original csv has weird formatting, extra spaces

# %%

admission\_type\_id = data\_2.iloc[0:8, :]

discharge\_disposition\_id = data\_2.iloc[10:40, :]

admission\_source\_id = data\_2.iloc[42:, :]

# %%

admission\_type\_id.head(10)

# %%

discharge\_disposition\_id.rename(columns={'admission\_type\_id':'discharge\_disposition\_id'}, inplace=True)

discharge\_disposition\_id.head(10)

# %%

discharge\_disposition\_id.rename(columns={'admission\_type\_id':'discharge\_disposition\_id'}, inplace=True)

discharge\_disposition\_id.head(10)

# %%

admission\_source\_id.rename(columns={'admission\_type\_id':'admission\_source\_id'}, inplace=True)

admission\_source\_id

# %% [markdown]

# ### Still need to finish transforming

# ##### The \*admission\_type\_id\*, \*discharge\_disposition\_id\*, \*admission\_source\_id\* in the original dataframe to categorical values

# ##### (See Above)

# %%

# # Randy

# what = []

# for i in range(0,len(data\_1)):

# try:

# data\_1["admission\_type\_id"].iloc[i] = admission\_type\_id["description"].iloc[int(data\_1["admission\_type\_id"].iloc[i]-1)]

# data\_1["discharge\_disposition\_id"].iloc[i] =discharge\_disposition\_id["description"].iloc[int(data\_1["discharge\_disposition\_id"].iloc[i]-1)]

# data\_1["admission\_source\_id"].iloc[i] = admission\_source\_id["description"].iloc[int(data\_1["admission\_source\_id"].iloc[i]-1)]

# except:

# what.append(i)

# continue

# %%

# Randy

data\_1\_clean = data\_1\_update.copy()

# %%

data\_1\_clean['diag\_1']

# %%

# Randy

colname = list(data\_1\_clean.columns)

mia = []

for i in range(len(data\_1\_clean.columns)):

count = data\_1\_clean[data\_1\_clean.columns[i]].isna().sum()

if count > 0:

mia.append(data\_1\_clean.columns[i])

print("Column '{col}' has {ct} NAs".format(col = colname[i], ct = count))

# %% [markdown]

# ### Still need to finish imputation

# ##### (See Below)

# %%

# Randy

for i in mia:

plt.figure(figsize=(15,5))

data\_1\_clean[i].value\_counts().plot(kind='bar')

plt.title(i)

plt.show()

# %%

data\_1\_clean["race"] = data\_1\_clean["race"].fillna('Caucasian')

data\_1\_clean["admission\_type\_id"] = data\_1\_clean["admission\_type\_id"].fillna('Emergency')

data\_1\_clean["discharge\_disposition\_id"] = data\_1\_clean["discharge\_disposition\_id"].fillna('Discharged to home')

data\_1\_clean["medical\_specialty"] = data\_1\_clean["medical\_specialty"].fillna('InternalMedicine')

data\_1\_clean["payer\_code"] = data\_1\_clean["payer\_code"].fillna('MC')

# %%

# Randy

colname = list(data\_1\_clean.columns)

mia = []

for i in range(len(data\_1\_clean.columns)):

count = data\_1\_clean[data\_1\_clean.columns[i]].isna().sum()

if count > 0:

mia.append(data\_1\_clean.columns[i])

print("Column '{col}' has {ct} NAs".format(col = colname[i], ct = count))

if mia:

print("There are missing values")

else:

print("All gucci")

# %%

data\_1\_clean.head(10)

# %% [markdown]

# ### MODELING (and Setup) ###

# %%

x0\_data = data\_1\_clean.drop(['readmitted'], axis=1)

y0\_data = data\_1\_clean.loc[:, 'readmitted']

column\_cats = [col for col, dt in x0\_data.dtypes.items() if dt == object]

column\_cats

model\_data = pd.get\_dummies(x0\_data, prefix=column\_cats)

print(len(model\_data.columns))

model\_data

# %%

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# %%

column\_nums = [x for x in model\_data.columns if x not in column\_cats]

# column\_nums

model\_data[column\_nums].head(2)

# %%

x\_scaled = model\_data.copy()

x\_scaled[column\_nums] = scaler.fit\_transform(x\_scaled[column\_nums])

# x\_scaled = pd.DataFrame(x\_scaled, columns=x\_data.columns)

x\_scaled[column\_nums].head()

# %%

x\_scaled

# %%

y\_data = y0\_data.copy()

# %%

y\_data.head()

# %%

# LABELS FOR OUTCOMES

y\_data.loc[y\_data == '<30'] = 1

y\_data.loc[y\_data == '>30'] = 0

y\_data.loc[y\_data == 'No'] = 0

y\_data.loc[y\_data == 'NO'] = 0

y\_data = y\_data.astype('int')

# %%

y\_data.head(20)

# %% [markdown]

# ### Actual Modeling ###

# %%

# train/test/split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_scaled, y\_data, test\_size=0.2, random\_state=1)

# %%

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import KFold

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import mean\_squared\_error, accuracy\_score

folder = KFold(shuffle=True, n\_splits=10)

log\_reg = LogisticRegression(penalty = 'l2')

# %%

c\_vals = np.logspace(-10,10,100)

cs\_for\_plots = []

best = -10000000000

best\_c = 'error'

for i in c\_vals:

log\_reg.C = i

out = cross\_val\_score(log\_reg, X\_train, y\_train, scoring='accuracy', cv=folder, n\_jobs=10).mean()

cs\_for\_plots.append([i,out])

if(out > best):

best = out

best\_c = i

print('\tbest accuracy\t\tbest C\n',best, '\t', best\_c)

# %%

model = LogisticRegression(penalty = 'l2', C=best\_c).fit(X\_train, y\_train)

train\_preds = model.predict(X\_train)

train\_score = accuracy\_score(train\_preds, y\_train)

train\_score

# %%

test\_preds = model.predict(X\_test)

test\_score = accuracy\_score(test\_preds, y\_test)

test\_score

# %%

train\_verify = []

test\_verify = []

for state in range(100,745,15):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_scaled, y\_data, test\_size=0.2, random\_state=state)

model = LogisticRegression(penalty = 'l2', C=best\_c).fit(X\_train, y\_train)

train\_preds = model.predict(X\_train)

train\_verify.append(accuracy\_score(train\_preds, y\_train))

test\_preds = model.predict(X\_test)

test\_verify.append(accuracy\_score(test\_preds, y\_test))

# %%

import six

feature\_df = pd.DataFrame({'coefficients':abs(model.coef\_.flatten()),

'features':X\_train.columns})

top10 = feature\_df.sort\_values('coefficients', ascending=False).head(10)

top10 = top10.round({'coefficients':5})

def render\_mpl\_table(data, col\_width=3.0, row\_height=0.625, font\_size=14,

header\_color='#40466e', row\_colors=['#f1f1f2', 'w'], edge\_color='w',

bbox=[0, 0, 1, 1], header\_columns=0,

ax=None, \*\*kwargs):

if ax is None:

size = (np.array(data.shape[::-1]) + np.array([0, 1])) \* np.array([col\_width, row\_height])

fig, ax = plt.subplots(figsize=size)

ax.axis('off')

mpl\_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.columns, \*\*kwargs)

mpl\_table.auto\_set\_font\_size(False)

mpl\_table.set\_fontsize(font\_size)

for k, cell in mpl\_table.\_cells.items():

cell.set\_edgecolor(edge\_color)

if k[0] == 0 or k[1] < header\_columns:

cell.set\_text\_props(weight='bold', color='w')

cell.set\_facecolor(header\_color)

else:

cell.set\_facecolor(row\_colors[k[0]%len(row\_colors) ])

return ax.get\_figure(), ax

fig,ax = render\_mpl\_table(top10, header\_columns=0, col\_width=4.0)

fig.savefig("table\_mpl.png")

# credit to volodymyr: https://stackoverflow.com/questions/19726663/how-to-save-the-pandas-dataframe-series-data-as-a-figure

# %%

L\_features = abs(model.coef\_.flatten())

L\_features = (L\_features > 0.000001).astype(int)

L\_features = L\_features.astype(np.bool)

model\_metrics = pd.DataFrame({'Data':['Training','Test'],

'Accuracy':[train\_score,test\_score]})

# model\_metrics

model\_metrics = model\_metrics.round({'Accuracy':6})

fig,ax = render\_mpl\_table(model\_metrics, header\_columns=0, col\_width=3.0)

fig.savefig("metrics\_tbl.png")

# %%

test\_verify.sort()

len(test\_verify)

# %%

from statistics import median

random\_competition = pd.DataFrame({'Data':['Training','Test'],

'Max Accuracy':[max(train\_verify), max(test\_verify)],

'Min Accuracy':[min(train\_verify), min(test\_verify)],

'Median Accuracy':[median(train\_verify), median(test\_verify)]})

random\_competition = random\_competition.round({'Max Accuracy':6,

'Min Accuracy':6,

'Median Accuracy':6})

fig,ax = render\_mpl\_table(random\_competition, header\_columns=0, col\_width=3.0)

fig.savefig("final\_tbl.png")

1. Strack, DeShazo, *et al.* Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records. BioMed Research International. Volume 2014 | Article ID 781670 | <https://doi.org/10.1155/2014/781670>. [↑](#footnote-ref-2)
2. National Center for Health Statistics. Center for Disease Control and Prevention. <https://www.cdc.gov/nchs/icd/icd9cm.htm>. [↑](#footnote-ref-3)