QBS Competition 2 - Report

(A) what I do for data pre-processing (remove NA or feature engineering……)

(B) explorative data analysis (EDA)

For (A) and (B):

1. I first see whether there are missing data, and I find that there are at least 80% missing data.
2. I check the distribution of 1, 0, and find that there is extremely imbalanced, which 0 accounts for over 99%.

See if there are NA

```
train_categorical.isnull().sum()
```

The distribution of 1, 0

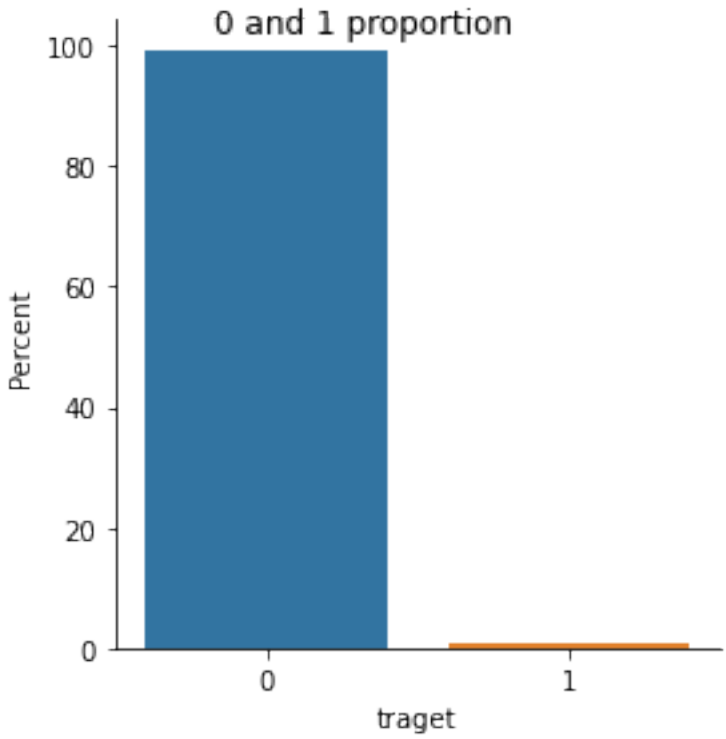
So we could Inow how to set weight class in the model

```
train_y = train_numeric.loc[:,"Response"].astype("int16")
x_train_label_to_draw = pd.DataFrame(dict(traget=train_y))

pic_0_1 = sns.catplot(x="traget", y="traget", data=x_train_label_to_draw, kind="bar",\
                      height=4, aspect=1, estimator=lambda x: len(x) / len(x_train_label_to_draw) * 100)

pic_0_1.set(ylabel="Percent")
pic_0_1.fig.suptitle("0 and 1 proportion")

print("0: {} %".format( round(x_train_label_to_draw.traget.value_counts()[0] / len(x_train_label_to_draw), 4)* 100 ) )
print("1: {} %".format( round(x_train_label_to_draw.traget.value_counts()[1] / len(x_train_label_to_draw), 4)* 100 ) )
```



3. For selecting Important numerical columns, I choose XGboost to find out the important columns.

```
X = train_numeric.iloc[:,0:968]
col_list = X.columns

y = train_numeric.iloc[:,968]

X = X.values
y = np.nan_to_num(y)

clf = XGBClassifier(base_score=0.005)
clf.fit(X, y)

# threshold for a manageable number of features
fig, ax = plt.subplots(nrows=1, ncols=1)
plt.hist(clf.feature_importances_[clf.feature_importances_ > 0])
fig.savefig("/content/output/xgboost_hist.png") ## to see the important feature pic
plt.close(fig)

fig, ax = plt.subplots(figsize=(20,50))
plt.xlabel('xlabel', fontsize=18)
plt.ylabel('ylabel', fontsize=18)
plt.xticks(size=12)
plt.yticks(size=12)
myplt = plot_importance(clf, ax=ax)
fig.savefig("/content/output/xgboost_importantfeatures.png") ## to see the important feature pic
plt.close(fig)

important_indices = np.where(clf.feature_importances_ > 0.005)[0]

important_columns = [col for i, col in enumerate(col_list) if i in important_indices]  # converts important_indices to col names
```

```
important_columns
```

4. To normalize and fill in the NA of numerical columns, I normalize them with “nanmean” and “nanstd” function, which omit the NAs in the columns. And I fill the NAs with mean.

Normalise and Fill numeric NAN

```
selected_train_numeric = train_numeric.loc[:,important_columns]
selected_test_numeric = test_numeric.loc[:,important_columns]

def normalise_and_fill(train_part, test_part):
    mean_nu = np.nanmean(train_part, axis=0)
    std_nu = np.nanstd(train_part, axis=0)
    train_part = (train_part - mean_nu)/std_nu
    test_part = (test_part - mean_nu)/std_nu

    col_list = selected_test_numeric

    select_to_dict = []
    for (col, mean) in zip(col_list, mean_nu):
        select_to_dict.append(col)
        select_to_dict.append(mean)

    it = iter(select_to_dict)
    fill_na_dict = dict(zip(it, it))

    train_part = train_part.fillna(fill_na_dict)
    test_part = test_part.fillna(fill_na_dict)

    return train_part, test_part


selected_train_numeric, selected_test_numeric = normalise_and_fill(selected_train_numeric, selected_test_numeric)

print(selected_train_numeric)
print(selected_test_numeric)
```

5. I drop the similar Date columns, because there are a lot of the same columns in the train_date, and it would cause bad thing to the model due to the multicollinear.

Drop similiar columns

I drop the similiar columns, because when training the model the similiar columns are redundant and could be harmful to the prediction because of the multicollinear

```
[ ]
def getunique(df):
    now_same_data = -1
    uniqueColumnNames = []

    uniqueColumnNames.append(df.columns.values[0])

    # Iterate over all the columns in dataframe
    for x in range(df.shape[1]):
        if (x >= now_same_data):
            col = df.iloc[:, x]
            for y in range(x + 1, df.shape[1]):
                # Select column at yth index.
                otherCol = df.iloc[:, y]
                # Check if two columns at x  y index are equal
                if (not col.equals(otherCol)):
                    uniqueColumnNames.append(df.columns.values[y])
                    now_same_data = y
                    break

    return uniqueColumnNames
```

```
[ ]
uniqueColumnNames = getunique(train_date)
```

6. I fill the NAs in the Date columns.

Fill NA in the Date

```
[ ]
selected_train_date = train_date.loc[:, uniqueColumnNames].fillna(0)
selected_test_date = test_date.loc[:, uniqueColumnNames].fillna(0)
```

```
[ ]
train_x = pd.concat([selected_train_numeric, selected_train_date],axis =1).to_numpy()
test_x =  pd.concat([selected_test_numeric, selected_test_date],axis =1).to_numpy()
train_y = train_numeric.loc[:,"Response"].astype("int16")
train_y = train_y.to_numpy()
```

(C) model development process:

I. DL model draft, parameter initialization, parameter tuning

I only follow the basic model which is show in the textbook

II. DL revised model draft, parameter initialization, parameter tuning

I find that there may be imbalanced in the label of training data, so I try to resample the data by oversampling and under sampling, yet it doesn't perform well.
And I use K-fold to find the best number of epochs.

III. DL finalized model, parameter initialization, parameter tuning

I find that add the hyper parameters would help, such as "dropout" and "kernel_regularizer".

Also, I find that setting the class weight would perform better that resampling, because resampling may cause overfitting in this situation, yet setting the class weight could avoid overfitting in this situation.
Hence, I end up using K-fold, hyper parameters, and setting up the class weight to build and fit my finalized model. And what I actually do is that keep trying to set different class weight to make my model preform better.

My finalized model:

1. Part of K-fold

```
"""from sklearn.utils import class_weight"""

num_portion = 3
num_val_sameples = len(train_x) // num_portion
num_epochs = 10

# four lists to record loss, val_loss, acc, val_acc
all_loss = []
all_val_loss = []
all_acc = []
all_val_acc = []

for i in range(num_portion): # go through every part of validation
    print("processing fold {} ...".format(i+1))

    val_data = train_x[i*num_val_sameples : (i+1)*num_val_sameples]
    val_label = train_y[i*num_val_sameples : (i+1)*num_val_sameples]

    prartial_x_train = np.concatenate( [ train_x[: i*num_val_sameples], train_x[(i+1)*num_val_sameples: ] ], axis=0)

    prartial_x_train_label = np.concatenate( [ train_y[: i*num_val_sameples],\
                                               train_y[(i+1)*num_val_sameples: ] ], axis=0)

    model = build_model()
    class_weights = {0:0.05, 1:10} # set class_weights, so we can avoid that the model tends to predict data as 0

    history = model.fit(prartial_x_train, prartial_x_train_label, validation_data=(val_data, val_label), epochs=num_epochs,\
                        batch_size=1024, verbose=0, class_weight=class_weights)
    ''', class_weight=class_weights'''

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]

    all_loss.append(loss)
    all_val_loss.append(val_loss)
    all_acc.append(acc)
    all_val_acc.append(val_acc)

    print()

print("< ====== K-fold Validation ends ====== >")
```

2. Part of hyperparameters

A function for building up the model, so that when we need to build the model, we don't need to do the redundant work.

```
[ ]  1 from keras import models, layers, regularizers
     2
     3 def build_model(): # a function to build model
     4     model = models.Sequential()
     5
     6     model.add(layers.Dense(80, activation="relu", kernel_regularizer = regularizers.l2(0.001), input_shape=(len(train_x[0]),) ) )
     7     model.add(layers.Dropout(0.5))
     8     model.add(layers.Dense(40, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
     9     model.add(layers.Dropout(0.5))
    10     model.add(layers.Dense(30, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    11     model.add(layers.Dense(25, activation="relu", kernel_regularizer = regularizers.l2(0.001)))
    12     model.add(layers.Dense(1, activation="sigmoid", kernel_regularizer = regularizers.l1_l2(0.001)))
    13
    14     model.compile(optimizer= "rmsprop", loss= "binary_crossentropy", metrics=["accuracy"])
    15
    16     return model
```

(d) The prediction result

The prediction result would be probability. I transform those whose probability < 0.5 into 0, and others into 1. The prediction result [“0” : ”1”] would be extremely imbalanced just like our training data. I think it because that, most of time, the production would not be broken. I get accuracy which is about 0.15, but I think it’s more like guessing because its poor performance.

(e) explaining the model prediction to make people trust and understand whether your predictive machine is reasonable, understandable, and trustable.

Actually, I don’t think my model is that reasonable. Because the accuracy is very low (about 0.15), and that could not be a reasonable model. But I think that the data processing I’ve done and the way I adjust the hyper-parameters are reasonable, such as drop the similar columns and to find the important columns.

(f) Your modeling and data analysis based on the lectures, tutorials, and assigned readings.

The model I build:

The basic and K-fold part is based on the textbook (FC) and tutorials.

The hyper-parameters part is based on the textbook (FC)

The class weight setting is based on the information given by TA and google and a classmate.

The data analysis I do is based on the self-study:

Data visualization and Feature Engineering.

Also, The XGboost classifier to choose the columns is based on google.

Other thought is from the other classmates, I’d discussed with them.

(e) learning progress, reflection, and feedback for the teaching team’s reference

For the learning process:

The most I've learned is that how to read in the large data! I spent about a week to try and asking classmate and TA for help, and ends up readings all of the data within only about 11 GB ram. This is really challenging at first.
I also learn the concept of multicollinear, which would be harmful to the model. At the same time, I try to use XGboost to choose the important columns instead of just using correlation table, which is poor performance and lack of efficiency.

For the feedback:

I consider that the homework is too challenging for us this time, even now, I still have many problems about the data and do not know how to deal with them. I consider that maybe teacher could let us work together in group for the homework. Having person to discuss with would be very helpful.