Machine Learning Final Project Report
my github
https://drive.google.com/file/d/169vtSZoEBwEHDyjVxA-4CdmqL8Y9Fzvk/view?usp=sharing- model weight but it's also available in github.

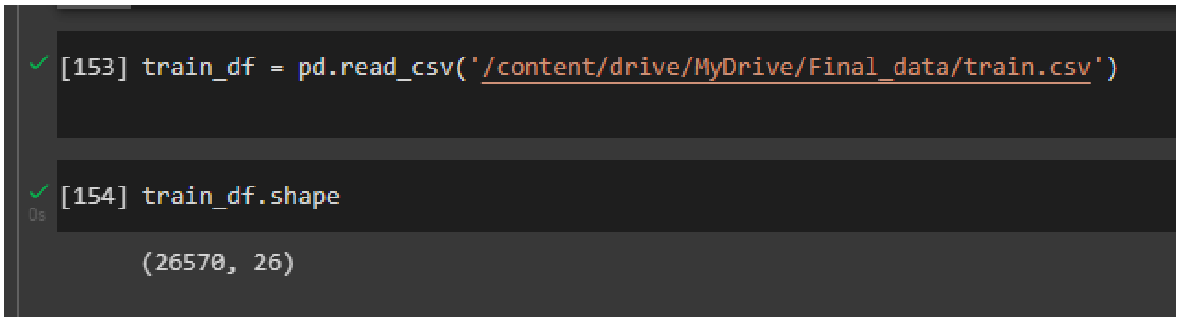
submission.csv
Complete (after deadline) · 20h ago
0.59065 0.58411

Brief Introduction

A company has just completed a large testing study for different product prototypes. And we are asked to use this data to build a model that predicts product failures. Here implemented a machine learning pipeline to train a classification model on the training data set. But before doing this and after reading a couple of discussions in the kaggle community, the trick is in preprocessing the data to get a more higher/accurate result. In my case, I first handle the missing values and categorical values by using KNNImputer and Labelencoder respectively. This is then split into train and validation datasets using k fold cross validation method with 5 numbers of splits as there are 5 product codes. For the model, I used Logistic regression with my custom hyperparameters that I played around with and stuck with the one I did my highest score with.

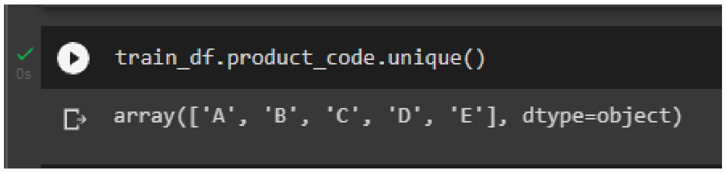
Methodology (Data pre-process, Model architecture, Hyperparameters, ...)

Before pre-processing, we need to understand what the dataset has for us.
As usual we read our dataset using pandas. And I checked the shape of the dataset, 26 columns.



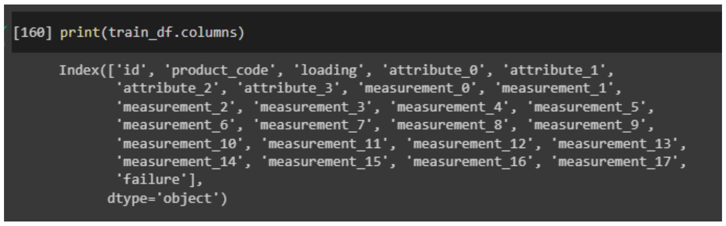
```
[153] train_df = pd.read_csv('/content/drive/MyDrive/Final_data/train.csv')

[154] train_df.shape

      (26570, 26)
```

And below we can see that there are 5 unique product_code. I also printed out the data types for all the columns and they vary from float, integer, and object/string. The other picture are all the columns.



```
train_df.product_code.unique()
```

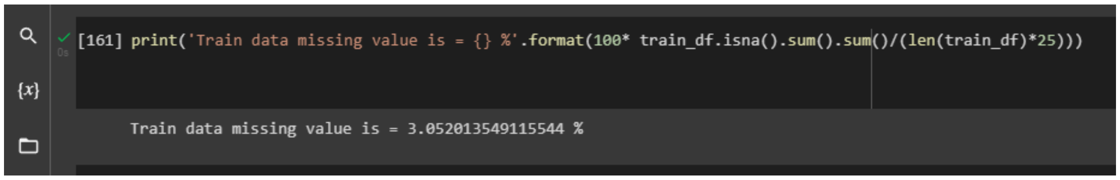
```
array(['A', 'B', 'C', 'D', 'E'], dtype=object)
```



```
[160] print(train_df.columns)
```

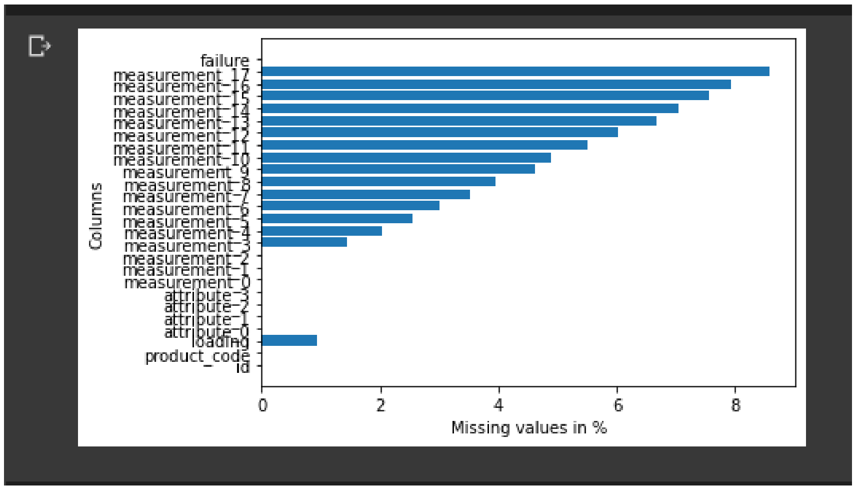
```
Index(['id', 'product_code', 'loading', 'attribute_0', 'attribute_1',
       'attribute_2', 'attribute_3', 'measurement_0', 'measurement_1',
       'measurement_2', 'measurement_3', 'measurement_4', 'measurement_5',
       'measurement_6', 'measurement_7', 'measurement_8', 'measurement_9',
       'measurement_10', 'measurement_11', 'measurement_12', 'measurement_13',
       'measurement_14', 'measurement_15', 'measurement_16', 'measurement_17',
       'failure'],
      dtype='object')
```

Now, we have to check for the missing values. Below we can see that the 3 percent of the train_df dataset are missing. And the graph shows which feature has the highest missing values.

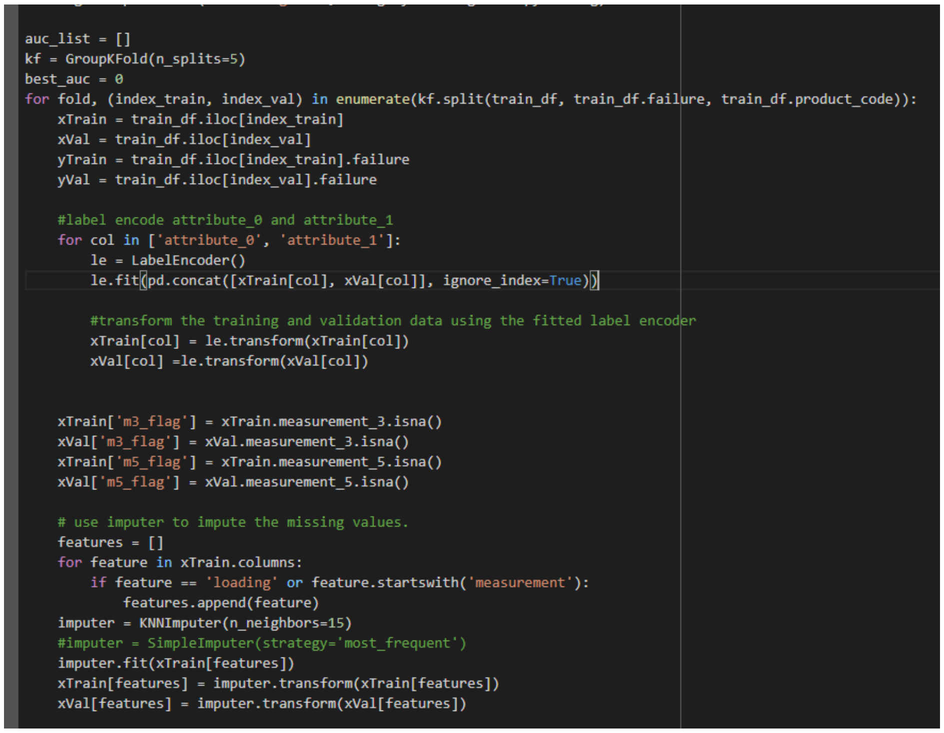


```
[161] print('Train data missing value is = {} %'.format(100* train_df.isna().sum().sum()/(len(train_df)*25)))
```

```
Train data missing value is = 3.052013549115544 %
```

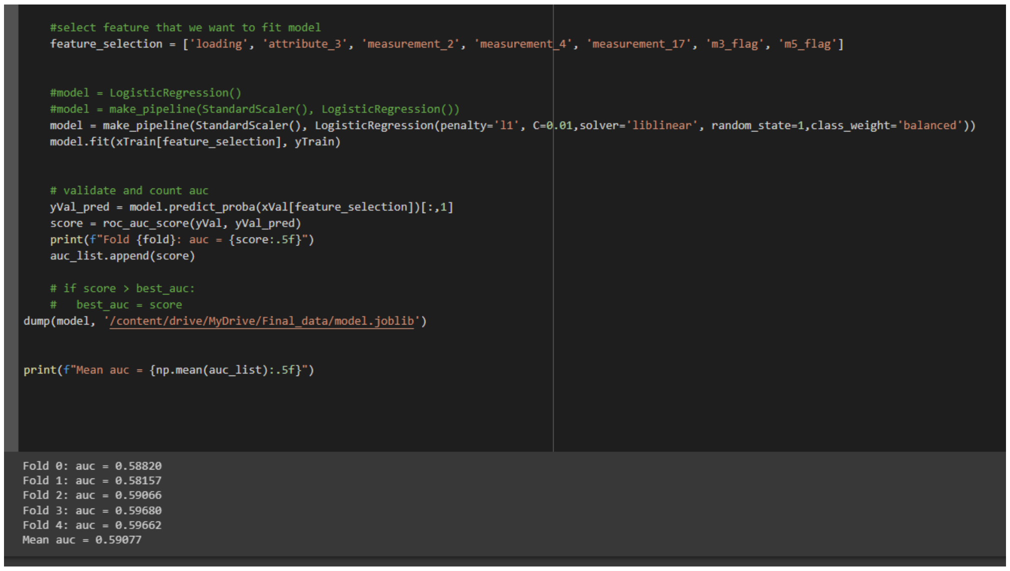


Now that we know some values are missing, we start to pre-process the data.



```
auc_list = []
kf = GroupKFold(n_splits=5)
best_auc = 0
for fold, (index_train, index_val) in enumerate(kf.split(train_df, train_df.failure, train_df.product_code)):
    xTrain = train_df.iloc[index_train]
    xVal = train_df.iloc[index_val]
    yTrain = train_df.iloc[index_train].failure
    yVal = train_df.iloc[index_val].failure

    #label encode attribute_0 and attribute_1
    for col in ['attribute_0', 'attribute_1']:
        le = LabelEncoder()
        le.fit(pd.concat([xTrain[col], xVal[col]], ignore_index=True))

        #transform the training and validation data using the fitted label encoder
        xTrain[col] = le.transform(xTrain[col])
        xVal[col] =le.transform(xVal[col])


    xTrain['m3_flag'] = xTrain.measurement_3.isna()
    xVal['m3_flag'] = xVal.measurement_3.isna()
    xTrain['m5_flag'] = xTrain.measurement_5.isna()
    xVal['m5_flag'] = xVal.measurement_5.isna()

    # use imputer to impute the missing values.
    features = []
    for feature in xTrain.columns:
        if feature == 'loading' or feature.startswith('measurement'):
            features.append(feature)
    imputer = KNNImputer(n_neighbors=15)
    #imputer = SimpleImputer(strategy='most_frequent')
    imputer.fit(xTrain[features])
    xTrain[features] = imputer.transform(xTrain[features])
    xVal[features] = imputer.transform(xVal[features])
```

Now for the pre-processing, as I mentioned earlier in the introduction, I used k fold cross validation, where k is 5 to split train and validation. Then for the categorical values like attribute_0 and attribute_1, we need to do label encoding in order to change these categorical values to numerical values so that they can be processed in the algorithm. Then from what I've read in many discussions which I will reference below, when missing measurement_3 and measurement_5, it deviates from the failure rate. So we have to flag or label these missing values.
Then we use an imputer to impute the missing values, here I used KNNimputer to impute the missing values using the value of the nearest neighbor with n_neighbor being number 15. I will write the comparison below.



```
    #select feature that we want to fit model
    feature_selection = ['loading', 'attribute_3', 'measurement_2', 'measurement_4', 'measurement_17', 'm3_flag', 'm5_flag']


    #model = LogisticRegression()
    #model = make_pipeline(StandardScaler(), LogisticRegression())
    model = make_pipeline(StandardScaler(), LogisticRegression(penalty='l1', C=0.01,solver='liblinear', random_state=1,class_weight='balanced'))
    model.fit(xTrain[feature_selection], yTrain)


    # validate and count auc
    yVal_pred = model.predict_proba(xVal[feature_selection])[:,1]
    score = roc_auc_score(yVal, yVal_pred)
    print(f"Fold {fold}: auc = {score:.5f}")
    auc_list.append(score)

    # if score > best_auc:
    #   best_auc = score
dump(model, '/content/drive/MyDrive/Final_data/model.joblib')


print(f"Mean auc = {np.mean(auc_list):.5f}")
```

```
Fold 0: auc = 0.58820
Fold 1: auc = 0.58157
Fold 2: auc = 0.59066
Fold 3: auc = 0.59680
Fold 4: auc = 0.59662
Mean auc = 0.59077
```

I didn't want to fit all the features into the model, so I picked a few myself that I believe are the most important features to be fit.
Next, I will explain the hyper parameters I used for the Logistic Regression model. I standardized it using scaler and noticed a very huge increase in my auc score.

- penalty = l1, i add l1 penalty term, to prevent overfitting.
- c = 0.01. I used a smaller value to get a stronger regularization.
- solver = ‘liblinear’
- class_weight=’balanced’ , class weight equal.
- random_state = 1

Also, I printed the AUC score to see the performance.
And not to forget, For the test dataset, I used clipping, for measurement_2.

Comparisons of different approaches and Thorough experimental results

For the data pre-process, I tried label encoding and one hot encoding and both seem to be the same in my case.
I also tried Simple Imputer with strategy mean, most_frequent, median, but they seem to result in a worse score than the KNN imputer. And for the KNN imputer, I experimented with a different number of nearest neighbor values, and I went from 3 to 15. And it boosted my score from 0.59019 to 0.59065.
I also tried clipping in the train data, but it seems to be worse, I don't know why.
For Logistic Regression’s hyperparameter, I tried the l2 penalty but it was bad, I used a smaller c value and it resulted better, also adding balanced class weight helped a little.

Summary
Participating in this course and doing this final project has taught me alot about machine learning. I learned how to implement a machine learning pipeline to train and evaluate a binary classification model. This involved preprocessing the data to handle missing values and categorical variables, and using k-fold cross-validation to train and validate the model.I also learned how to use techniques such as label encoding and KNN imputation to prepare the data for modeling, and how each of the parameters of the model affects the score.

Overall, this project was a valuable learning experience and I am grateful to the TA and professor for their guidance and support.

Reference

https://www.kaggle.com/code/ambrosm/tpsaug22-eda-which-makes-sense- for preprocessing
https://www.kaggle.com/code/kartushovdanil/tps-aug-22-advanced-eda-modeling- for the data viewing