Report on the Tabular Playground classification project

In the very beginning, I choose to put all the columns in the dataframe, scale all the numbers and throw it to an basic NN model to see how much it shows, and it resulted in a very poor performance around 0.52

submissions2.csv Complete (after deadline) · 2d ago	0.52894	0.50941

Therefore, I decided to do feature selection first.

First of all, the data has a lot of missing values.

```
In [282]: nan_count = data.isna().sum()
          print(nan_count)

id                   0
product_code         0
loading            250
attribute_0          0
attribute_1          0
attribute_2          0
attribute_3          0
measurement_0        0
measurement_1        0
measurement_2        0
measurement_3      381
measurement_4      538
measurement_5      676
measurement_6      796
measurement_7      937
measurement_8     1048
measurement_9     1227
measurement_10    1300
measurement_11    1468
measurement_12    1601
measurement_13    1774
measurement_14    1874
measurement_15    2009
measurement_16    2110
measurement_17    2284
failure              0
dtype: int64
```

So I create columns indicating missing values and see how they correlate to the outcome like this. .

```
In [222]: cor1 = data.iloc[:,22:].corr()
          cor1.iloc[0,:]

Out[222]: failure        1.000000
          m3_missing    -0.015478
          m4_missing     0.008893
          m5_missing     0.016519
          m6_missing     0.000952
          m7_missing    -0.001104
          m8_missing    -0.002275
          m9_missing     0.009699
          m10_missing    0.000260
          m11_missing   -0.000446
          m12_missing    0.006036
          m13_missing   -0.001536
          m14_missing    0.005235
          m15_missing    0.000999
          m16_missing   -0.004288
          m17_missing    0.004398
          Name: failure, dtype: float64
```

It turns out that the absence of measurement_3 and measurement_5 affect the result more than other measurements, so I choose to add these 2 as new features.

Then, I encode the categorial features into integers so it;s easier to preprocess the data and see how they correlate.

```
cleanup_nums1 = {"attribute_0": {"material_5": 0, "material_7": 1}}
cleanup_nums2 = {"product_code": {"A": 0, "B": 1,"C": 2,"D": 3,"E": 4}}
data = data.replace(cleanup_nums1)
data = data.replace(cleanup_nums2)
```

Then, see how it correlates with the data.

```
In [223]: cor2 = data.iloc[:,:23].corr()
          cor2.iloc[22,:]

Out[223]: product_code      -0.007880
          loading            0.129089
          attribute_0        0.014830
          attribute_3       -0.019222
          measurement_0      0.009646
          measurement_1     -0.010810
          measurement_2      0.015808
          measurement_3      0.003577
          measurement_4     -0.010488
          measurement_5      0.018079
          measurement_6      0.014791
          measurement_7      0.016787
          measurement_8      0.017119
          measurement_9     -0.003587
          measurement_10    -0.001515
          measurement_11    -0.004801
          measurement_12     0.004398
          measurement_13    -0.001831
          measurement_14     0.006211
          measurement_15    -0.003544
          measurement_16     0.002237
          measurement_17     0.033905
          failure            1.000000
          Name: failure, dtype: float64
```

It's shown that measurement_17 and loading might be the most significant feature. attribute 0 and 3, measurement_0~2 and4~8 also has some correlation that might be helpful. So, my approach is to consider those features only.

I added another attribute indicating the product from attribute 2 and 3, this trick was referenced from a kaggle discussion here, which assumes that they are related . (https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/342126)

Further digging, I computed the correlation between measurement_17 and other features.

```
Out[226]: measurement_3     0.080902
          measurement_4     0.193045
          measurement_5     0.450024
          measurement_6     0.328136
          measurement_7     0.330328
          measurement_8     0.492839
          measurement_9     0.145465
          Name: measurement_17, dtype: float64
```

measurement 3 to 9 are significant correlated to measurement 17 compare to other features, which means:

1. I only need measurement_17 because it might contain some features other measurements contain and that helps me to do feature selection.
2. I can impute the missing value from those measurements because they are correlated.

After getting the information about measurement_17, I started imputing the missing value. I group the data using product_code, and see how the measurement affects measurement17 in a deeper approach.

```
In [415]: producta=data[data['product_code']==0]
          productb=data[data['product_code']==1]
          productc=data[data['product_code']==2]
          productd=data[data['product_code']==3]
          producte=data[data['product_code']==4]
```

The picture below shows an example, where in product A, measurement_8 has a 0.73 correlation with measurement_17, that's a very very high correlation when doing feature selection. So, for product A, I can use measurement_8 to impute missing value in measurement_17.

```
In [493]: producta.corr().iloc[21,4:-4]

Out[493]: measurement_0    -0.008027
          measurement_1    -0.001799
          measurement_2     0.000132
          measurement_3    -0.013263
          measurement_4     0.141816
          measurement_5     0.564070
          measurement_6     0.275952
          measurement_7     0.242866
          measurement_8     0.737438
          measurement_9     0.016441
          measurement_10    0.006839
          measurement_11   -0.010395
          measurement_12   -0.010791
          measurement_13    0.001045
          measurement_14   -0.002528
          measurement_15    0.014249
          measurement_16   -0.003879
          measurement_17    1.000000
          Name: measurement_17, dtype: float64
```

However, applying this method has another drawback, that is, I found out there are missing values in measurement_8 too. So, when measurement_8 and measurement_17 are missing at the same rows, I need to have a back up plan. Here, I choose measurement_5, which is the second highest correlated measurement, and so on.(usually it only needs one or two, there aren't records with over 3 highest measurements missing)

Here, I use HuberRegressor to fit the selected features, the first one is the most correlated, and the second one is the backup regressor.
Apply the method to all the products and I imputed successfully.

```
train_a = impute_a.dropna(how='any')
hubera_1 = HuberRegressor().fit(train_a.iloc[:,11].values.reshape(-1, 1),train_a.iloc[:,12].values.reshape(-1, 1))
hubera_2 = HuberRegressor().fit(train_a.iloc[:,8].values.reshape(-1, 1),train_a.iloc[:,12].values.reshape(-1, 1))
```

I impute measurement_17 to all the products, and now the only feature with missing value is loading.
For loading, I use KNN to impute the values, I choose the neighbor parameter as 10.

```
imputer = KNNImputer(n_neighbors=10, weights="uniform")
impute_a['loading']=imputer.fit_transform(np.array(impute_a['loading']).reshape(-1,1))
```

Finally, all data is imputed.
I drop all the unnecessary measurements(ex: measurement 3~9),. They've already done their job!

Now, we can see that 9 features and 1 label are left, and none of them contain missing values!

```
loading          0
attribute_0      0
measurement_0    0
measurement_1    0
measurement_2    0
measurement_17   0
failure          0
m3_missing       0
m5_missing       0
attribute_2*3    0
dtype: int64
```

Before I dump it to the model, I have to scale it so here is a scale function, I am using StandardScaler to scale the data.

```
#a fuction to perform scaling before feeding it to the model, here i just use StandardScaler

def scaling(data):
    scaler = StandardScaler()
    select_feature = ['measurement_0', 'measurement_1', 'measurement_2', 'loading', 'measurement_17', 'attribute_2*3']
    scaled = scaler.fit_transform(data[select_feature])
    new = data.copy()
    new[select_feature] = scaled
    assert len(data) == len(new)
    return new
```

Now, it's time to run the model.

Create the final train X and train Y

```
#combine the data after preprocessing to produce the final training data

frames = [impute_a, impute_b, impute_c, impute_d, impute_e]
train = pd.concat(frames)

X = train.drop(['product_code','failure'], axis=1)
Y = train['failure'].astype(int)
```

I'm using a cross-validation approach on a LogisticRegression model.
I choose L2 regularization on this LR model, and apply a model to every fold created.
After fitting the data to the model, I save the model using pickle.

Every model serves the same weight, so here I use 10 fold, I will scale the probability to 1/10 and calculate the sum of them for evaluating.

```
fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
for  id, (tid, vid) in enumerate(fold.split(X, Y)):

    x_train, x_val = X.iloc[tid], X.iloc[vid]
    y_train, y_val = Y.iloc[tid], Y.iloc[vid]

    x_train = scaling(x_train)
    x_val = scaling(x_val)


    model = LogisticRegression(penalty='l2', solver='newton-cg', max_iter=1000, C=0.012)
    model.fit(x_train, y_train)
    filename = 'finalized_model' + str(i) + '.sav'
    pickle.dump(model, open(filename, 'wb'))
    i = i+1

    auc_score += roc_auc_score(y_val, model.predict_proba(x_val)[:,1]) /10
    accuracy += accuracy_score(y_val, model.predict(x_val)) /10
```

Here are the score of the models.

```
(0.5918208636867467, 0.7872036130974782)
```

For inference, I do the same preprocessing like training :

1. Encode the data
2. Select significant features and add new features
3. Impute missing values with the help of : Correlated measurements,HuberRegressor and KNN
4. Drop unnecessary data
5. Scale the final data

After the final testing data is ready, apply the model I just created on them.
Again, the prediction is using balanced weight, every model constructs 1/10 of the probability.

```
predictions = np.zeros(len(X_test))
for i in range(10):
    model = pickle.load(open('finalized_model' + str(i) + '.sav', 'rb'))
    predictions += model.predict_proba(X_test)[:,1] / 10

predictions
```

```
array([0.18599365, 0.1631584 , 0.17514629, ..., 0.14546447, 0.21756607,
       0.16200819])
```

Write the prediction into the submission file.

```
dfsub = pd.read_csv(r'C:\Users\user\Desktop\tabular-playground-series-aug-2022\sample_submission.csv', delimiter=',', usecol
dfsub['failure'] = rankdata(predictions)

os.makedirs('finalprojectml', exist_ok=True)
dfsub.to_csv('finalprojectml/109550092_submissions.csv',index=False)
```

And the result is 0.59114! which is a nice result, a above 0.59 private score.

Submission and Description	Private Score	Public Score	Selected
109550092_submissions.csv Complete (after deadline) · now	0.59114	0.58743	

Finally, I list the library I used here in case there’s problem creating the environment:

numpy
pandas
csv
pickle
os
math
sklearn
scipy

And the link for the github code is here, with both the training and inferencing part.
https://github.com/chiou1203/Project-on-Tabular-Playground-Series-Classification