黃承豪 109550203

MACHINE LEAERNING FINAL PROJECT REPORT

Screenshot of Private Accuracy :

Link of GitHub :

https://github.com/nathanowen06/ML-2022-NYCU

Link of Best Weight:

https://github.com/nathanowen06/ML-2022-NYCU/blob/main/predictions.joblib

Environment Details

Libray	Version
Python	3.9.0
Numpy	1.22.4
Pandas	1.3.5
Opencv-python	4.4.0.46
tqdm	4.64.0
torch	1.11.0
joblib	1.1.0
sklearn	
mpmath	1.2.1

Implementation Details:

```
xTrainData = pd.read_csv("/kaggle/input/tabular-playground-series-aug-2022/train.csv")
xTrainData = xTrainData.reset_index().set_index('id').drop('index', axis=1)

xTrainData['missingm3'] = xTrainData.measurement_3.isna()
xTrainData['missingm5'] = xTrainData.measurement_5.isna()

trainFeatures = []
for values in xTrainData.columns:
    if values == 'loading' or (values.startswith('measurement') and values != 'measurement_17'):
        trainFeatures.append(values)
chosenImputer = KNNImputer(missing_values=np.nan, n_neighbors=15)
chosenImputer.fit(xTrainData[trainFeatures])
xTrainData[trainFeatures] = chosenImputer.transform(xTrainData[trainFeatures])
```

This code is handling missing values in the xTrainData DataFrame. It first resets the index and sets the 'id' column as the new index, then drops the old 'index' column. It then selects a subset of columns from the DataFrame, specifically 'loading' and columns that start with 'measurement' except 'measurement_17' and stores them in a trainFeatures list. These columns will be used to impute missing values using **KNNImputer** method from sklearn.impute library. It initializes an instance of KNNImputer, it is set to use 15 nearest neighbors and impute the NaN values. The imputer is fitted with the trainFeatures, and the missing values in these columns are imputed. Finally, it assigns the imputed values back to the original DataFrame, effectively handling the missing values.

MACHINE LEAERNING FINAL PROJECT REPORT

```
comparableColumns = ['measurement_3', 'measurement_4', 'measurement_5','measurement_6', 'measurement_7', 'measurement_8', 'measurement_9
xTrainData['measurement17predictions'] = ""
training_list = ['A', 'B', 'C', 'D', 'E']
for result in training_list:
    train_source = xTrainData[(xTrainData['product_code'] == result) & ~pd.isnull(xTrainData.measurement_17)]
    LRofMeasurement17 = LinearRegression().fit(train_source[comparableColumns], train_source['measurement_17'])
    sourceForPrediction = xTrainData[xTrainData['product_code'] == result]
    xTrainData.loc[sourceForPrediction.index, 'measurement17predictions'] = LRofMeasurement17.predict(sourceForPrediction[comparableColur
    xTrainData.loc[sourceForPrediction[pd.isnull(sourceForPrediction.measurement_17)].index, 'measurement_17'] = xTrainData.loc[sourceFor

xTrainData['product_code'] = xTrainData['product_code']
```

This code is predicting values for the 'measurement_17' column based on other related columns, using Linear Regression and then replacing the missing values in 'measurement_17' with the predicted values. It starts by defining a list of column names called comparableColumns which contains all columns that are related to 'measurement_17'. Then, it creates a new column in xTrainData DataFrame called 'measurement17predictions' and initializes it with an empty string for every row.

It then defines a list of 'product_code' called training_list, which contains 'A', 'B', 'C', 'D' and 'E'. In the for loop, the code creates a new variable train_source that holds the rows of xTrainData where the 'product_code' is equal to the current item in the list and 'measurement_17' is not NaN. It then initializes a Linear Regression model and fits it to the 'train_source' dataframe. Then it uses the predicted values to replace the missing values in 'measurement_17' for each product code in the training_list.

In summary, the code is using linear regression to predict 'measurement_17' values based on other related columns, when 'product_code' is 'A', 'B', 'C', 'D', or 'E' and then replacing the missing values in 'measurement_17' with the predicted values for each product code. The new column 'measurement17predictions' is added to store the predicted values for 'measurement_17' in order to know the performance of prediction and compare it to the true values.

```
yTrainData = xTrainData.failure
xTrainData = xTrainData.drop(['failure'],axis=1)
```

This code is separating the target variable 'failure' from the feature set of the xTrainData DataFrame. It first assigns the 'failure' column of the xTrainData to a new variable 'yTrainData', which is the target variable that will be used to train and evaluate the machine learning model. Then, it removes the 'failure' column from the xTrainData DataFrame using the drop method, ensuring that the target variable is not included in the input features during the training of the model. This is a common practice in machine learning where the target variable is usually stored separately from the feature set, thus, xTrainData will no longer include the 'failure' column, and yTrainData will only include the 'failure' column.

MACHINE LEAERNING FINAL PROJECT REPORT

```
PCAObjectives = [
    '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'
]

dropped = ['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']

dropped2 = ['product_code', 'attribute_3', 'attribute_2', 'attribute_1', 'attribute_0']

pcaApplied = PCA(n_components=1)
xTrainData['pcaApplied'] = pcaApplied.fit_transform(xTrainData[PCAObjectives])

# xTrainData['measurement_2'] = xTrainData['measurement_2'].clip(11, None)

xTrainData['area'] = xTrainData['attribute_2'] * xTrainData['attribute_3']

xTrainData = xTrainData.drop(dropped, axis=1)

xTrainData = xTrainData.drop(dropped2, axis=1)

selectionOfFeatures = ['area', 'missingm5', 'missingm3', 'measurement_17', 'measurement_2', 'measurement_1', 'measurement_0', 'loading']
xTrainData[selectionOfFeatures] = StandardScaler().fit_transform(xTrainData[selectionOfFeatures])
```

This code is applying principal component analysis (PCA) on a set of columns and then dropping unnecessary columns from the xTrainData DataFrame. First, it defines a list of column names called PCAObjectives, which contains the names of the columns that will be used for PCA. These columns are related to measurement values, and are the columns used to explain the variation in data.

Next, it creates an instance of the PCA class which is a dimensionality reduction technique that reduces the number of features while preserving as much of the original variation as possible. It fits the PCA model to the xTrainData DataFrame with the specified PCAObjectives, applies the dimensionality reduction, transforms the data and stores the results in a new column named 'pcaApplied'. An additional feature 'area' is computed and unnecessary columns are dropped using the drop method.

In summary, the code applies PCA to a set of columns and reduces the number of features while preserving important information, and also removes unnecessary columns from the DataFrame making it more manageable and easier to work with. The additional feature 'area' is added and standardization is applied to all features in order to provide a more robust model by ensuring all features have zero mean and unit variance.

```
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'solver': ['liblinear'],
              'max_iter': [1000, 10000, 100000]}

kf = KFold(n_splits=5, shuffle=True, random_state=1)

scores = []

for fold, (trainIndex, valueIndex) in enumerate(kf.split(xTrainData, yTrainData)):
    xTrainDataFold, yTrainDataFold = xTrainData.iloc[trainIndex], yTrainData.iloc[trainIndex]
    xValueFold, yValueFold = xTrainData.iloc[valueIndex], yTrainData.iloc[valueIndex]
    grid_search = GridSearchCV(estimator=LogisticRegression(), param_grid=param_grid, scoring='roc_auc', cv=5, n_jobs=-1)
    model = make_pipeline(StandardScaler(), LogisticRegression(penalty='l1', C=0.01, solver='liblinear', random_state=1))
    grid_search.fit(xTrainDataFold, yTrainDataFold)
    model = grid_search.best_estimator_
    y_pred = model.predict_proba(xValueFold)[:, 1]
    score = roc_auc_score(yValueFold, y_pred)
    scores.append(score)
    print(f"Fold {fold}: {score:.5f}")
print(f"Average ROC AUC = {sum(scores) / len(scores):.5f}")
```

黃承豪 109550203

MACHINE LEAERNING FINAL PROJECT REPORT

The code is performing a combination of k-fold cross-validation and grid search to find the best parameters for a logistic regression model and evaluate its performance. The grid search is performed by searching through a defined param_grid of different regularization types, inverse regularization strengths, solvers and maximum iterations. The KFold class is used to split the data into 5 folds, where each fold is used once as the validation set, and k-1 folds are used as the training set. The GridSearchCV class is used to search for the optimal parameters and the performance is evaluated by computing the ROC AUC score and storing it in a scores list. Finally, the code prints the average ROC AUC score after all the iterations to get the overall performance of the model.

This process is pipeline based, where standard scaler is applied to standardize the data and a logistic regression model is applied using the best estimator from grid search and using the fit method to fit the grid search object with the training data. The pipeline uses predict_proba method to predict the probability of each sample belonging to the positive class, and final ROC AUC is computed using roc_auc_score function by passing yValueFold and predicted probability.

```
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'solver': ['liblinear'],
              'max_iter': [1000, 10000, 100000]}
grid_search = GridSearchCV(estimator=LogisticRegression(), param_grid=param_grid, scoring='roc_auc', cv=5, n_jobs=-1)
grid_search.fit(xTrainData, yTrainData)

model = grid_search.best_estimator_
model.fit(xTrainData, yTrainData)
dump(model, 'predictions.joblib')
```

This code is training a logistic regression model using the optimal parameters found by the grid search and then saving the trained model to a file.

So this code, takes advantage of the grid search in order to find the best hyperparameters for the logistic regression model, trains the model using the optimal parameters, and saves the trained model to a file for future use.

MACHINE LEAERNING FINAL PROJECT REPORT

```
xTestData = pd.read_csv("/kaggle/input/tabular-playground-series-aug-2022/test.csv")
xTestData = xTestData.reset_index().set_index('id').drop('index', axis=1)

selectionOfFeatures = ['area', 'missingm5', 'missingm3', 'measurement_17', 'measurement_2', 'measurement_1', 'measurement_0', 'loading']

xTestData['missingm3'] = xTestData.measurement_3.isna()
xTestData['missingm5'] = xTestData.measurement_5.isna()

testFeatures = []
for values in xTestData.columns:
    if values == 'loading' or (values.startswith('measurement') and values != 'measurement_17'):
        testFeatures.append(values)
chosenImputer = KNNImputer(missing_values=np.nan, n_neighbors=15)
chosenImputer.fit(xTestData[testFeatures])
xTestData[testFeatures] = chosenImputer.transform(xTestData[testFeatures])
```

```
comparableColumns = ['measurement_3', 'measurement_4', 'measurement_5','measurement_6', 'measurement_7', 'measurement_8', 'measurement_9
xTestData['measurement17predictions'] = ""
test_list = ['F', 'G', 'H', 'I']
for result in test_list:
    train_source = xTestData[(xTestData['product_code'] == result) & ~pd.isnull(xTestData.measurement_17)]
    LRofMeasurement17 = LinearRegression().fit(train_source[comparableColumns], train_source['measurement_17'])
    sourceForPrediction = xTestData[xTestData['product_code'] == result]
    xTestData.loc[sourceForPrediction.index, 'measurement17predictions'] = LRofMeasurement17.predict(sourceForPrediction[comparableColumn
    xTestData.loc[sourceForPrediction[pd.isnull(sourceForPrediction.measurement_17)].index, 'measurement_17'] = xTestData.loc[sourceForPr

PCAObjectives = [
    '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'
]

pcaApplied = PCA(n_components=1)
xTestData['pcaApplied'] = pcaApplied.fit_transform(xTestData[PCAObjectives])
```

Basically, in the inference code, we just repeat the algorithm done in train code but applying it in a different data, which is xTestData. In the part of the code above, the list is changed to test_list with F, G, H, an I as the components. But, in the inference code, I applied some more methods to increase the accuracy of the model. Inserted a line of code below,

```
xTestData['measurement_2'] = xTestData['measurement_2'].clip(11, None)
```

This line of code is applying a clipping operation on the 'measurement_2' column of the xTestData DataFrame.

In this particular case, the clip method is being used to set all values in the 'measurement_2' column that are less than 11 to 11 and all values that are greater than 11 to be left unchanged. In other words, it's capping the minimum value of 'measurement_2' to 11.

The clip method takes two arguments, the first being the lower bound and the second being the upper bound. Any value below the lower bound will be set to the lower bound value, and any value above the upper bound will be set to the upper bound value.

Clipping the values in this way helps to constrain outliers values, it's common practice in data preprocessing to make sure that the data does not contain any extreme values that may be causing an issue when training the model, in this case, the minimum value of 'measurement_2' is constrained to 11, ensuring that the column contains realistic values.

黃承豪 109550203

MACHINE LEAERNING FINAL PROJECT REPORT

```
xTestData['area'] = xTestData['attribute_2'] * xTestData['attribute_3']

dropped = ['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']
xTestData = xTestData.drop(dropped, axis=1)
dropped2 = ['product_code', 'attribute_3', 'attribute_2', 'attribute_1', 'attribute_0']
xTestData = xTestData.drop(dropped2, axis=1)
xTestData[selectionOfFeatures] = StandardScaler().fit_transform(xTestData[selectionOfFeatures])
```

```
param_grid = {'penalty': ['l1', 'l2'],
              'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'solver': ['liblinear'],
              'max_iter': [1000, 10000, 100000]}
grid_search = GridSearchCV(estimator=LogisticRegression(), param_grid=param_grid, scoring='roc_auc', cv=5, n_jobs=-1)
grid_search.fit(xTrainData, yTrainData)

model = grid_search.best_estimator_
model = load('predictions.joblib')
y_pred = model.predict_proba(xTestData)[:,1]
```

This code is loading a previously trained model from a file, making predictions on new data (xTestData), and storing the predictions in a variable (y_pred). The predict_proba method is used to get the probability of the positive class, allowing to rank the output and take a decision accordingly.

```
sample = pd.read_csv('/kaggle/input/tabular-playground-series-aug-2022/sample_submission.csv', index_col='id')
sample['id'] = xTestData.index
sample['failure'] =  y_pred
sample.to_csv("/kaggle/working/submission.csv",index=False)
```

This is the method I use to output the results into a csv file which is submitted in my github as my best model.

Conclusion:

The methods used in my code are:

1. KNNImputer is used to complete the missing features, since there are some float feature columns that have missing values. I set the nearest neighbor to 5.
2. Both missingmeasurement 3 and missingmeasurement 5 have failure rates that are significantly different from the average failure rate. The model is updated with signs for the missing measurements 3 and 5.
3. The method performs PCA on a set of columns, which decreases the amount of features while maintaining crucial data. It also removes extraneous columns from the DataFrame, which makes it easier to handle and work with.
4. The program uses linear regression to forecast the values of "measurement 17" based on information from other relevant columns to predict values for the 'measurement_17' column based on other related columns.

I chose Logistic Regression as seen from the Kaggle most voted code from AMBROSM, he compared RandomForestClassifier, ExtraTreesClassifier and LogisticRegression, but as I consider the Logistic Regression is the best.

黃承豪 109550203

MACHINE LEAERNING FINAL PROJECT REPORT

My code is performing a combination of k-fold cross-validation and grid search in order to find the best parameters for a logistic regression model and evaluate its performance.

Reference:

Kaggle Solution of AMBROSM:

https://www.kaggle.com/code/ambrosm/tpsaug22-eda-which-makes-sense

Kaggle Solution of AZMINE:

https://www.kaggle.com/code/azminetoushikwasi/classification-comparing-different-algorithms