Homework 2

Data Analysis and Machine Learning with Python

電機四 B09602017 白宗民

Q0. Please choose two datasets for Homework 2, and please tell me why you selected these two datasets.

Answer: I choose 3 datasets for this HW, only without feedback sentiment. Since it's restricted to some specific steps and I do not want to be restricted.

Dataset 1: Maintenance Prediction

Q1. How many unique device IDs are there in this dataset?

```python
# Count unique device IDs
unique_devices = df['device'].nunique()
print(f'Number of unique device IDs: {unique_devices}')
```

```
Number of unique device IDs: 1169
```

Q2. You are asked to do data analysis. What will you find?

```python
# Summary statistics
summary_stats = df.describe()
print(summary_stats)
```

✓ 0.0s Python

```
             failure       metric1        metric2        metric3  \
count  124493.000000  1.244930e+05  124493.000000  124493.000000
mean        0.000851  1.223875e+08     159.493988       9.940977
std         0.029167  7.045934e+07    2179.686488     185.748875
min         0.000000  0.000000e+00       0.000000       0.000000
25%         0.000000  6.128346e+07       0.000000       0.000000
50%         0.000000  1.227971e+08       0.000000       0.000000
75%         0.000000  1.833091e+08       0.000000       0.000000
max         1.000000  2.441405e+08   64968.000000   24929.000000

             metric4        metric5        metric6        metric7  \
count  124493.000000  124493.000000  124493.000000  124493.000000
mean        1.741134      14.222719  260173.031022       0.292531
std        22.908598      15.943082   99151.389285       7.436954
min         0.000000       1.000000       8.000000       0.000000
25%         0.000000       8.000000  221452.000000       0.000000
50%         0.000000      10.000000  249800.000000       0.000000
75%         0.000000      12.000000  310266.000000       0.000000
max      1666.000000      98.000000  689161.000000     832.000000

             metric8        metric9
count  124493.000000  124493.000000
mean        0.292531      13.013953
std         7.436954     275.662324
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max       832.000000   70000.000000
```

```python
# Correlation analysis (excluding non-numeric columns)
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
correlation_matrix = df[numeric_columns].corr()
print(correlation_matrix)
```

✓ 0.0s

```
          failure   metric1   metric2   metric3   metric4   metric5   metric6  \
failure  1.000000  0.001984  0.052901 -0.000949  0.067398  0.002270 -0.000550
metric1  0.001984  1.000000 -0.004253  0.003702  0.001836 -0.003373 -0.001518
metric2  0.052901 -0.004253  1.000000 -0.002617  0.146762 -0.013999 -0.026350
metric3 -0.000949  0.003702 -0.002617  1.000000  0.097452 -0.006697  0.009030
metric4  0.067398  0.001836  0.146762  0.097452  1.000000 -0.009773  0.024870
metric5  0.002270 -0.003373 -0.013999 -0.006697 -0.009773  1.000000 -0.017051
metric6 -0.000550 -0.001518 -0.026350  0.009030  0.024870 -0.017051  1.000000
metric7  0.119055  0.000151  0.141366 -0.001884  0.045631 -0.009384 -0.012207
metric8  0.119055  0.000151  0.141366 -0.001884  0.045631 -0.009384 -0.012207
metric9  0.001066 -0.002256 -0.002049  0.369579  0.024892  0.003591  0.016566
```

We can see that the correlation coefficients are pretty low.

Q3. You are asked to build a prediction model. Which kind of machine learning will be used and why? Supervised learning or Unsupervised learning? Regression, classification, or clustering? Which model will you use?

Logistic Regression: Since it's a kinda simple task I think, just use some simple model would have better performance.

```python
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_

# Initialize and train the Logistic Regression model with class weigh
# log_reg_model = LogisticRegression(random_state=42, max_iter=10000)
log_reg_model = LogisticRegression(random_state=42, class_weight='bal
log_reg_model.fit(X_train_resampled, y_train_resampled)

# Make predictions
log_reg_y_pred = log_reg_model.predict(X_test)

# Evaluate the model
print("Logistic Regression Classification Report (with SMOTE):")
print(classification_report(y_test, log_reg_y_pred))

joblib.dump(log_reg_model, 'log_reg_model.pkl')
# You can apply SMOTE for other models similarly
df
```

Python

```
Logistic Regression Classification Report (with SMOTE):
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     37319
           1       0.01      0.41      0.02        29

    accuracy                           0.97     37348
   macro avg       0.50      0.69      0.50     37348
weighted avg       1.00      0.97      0.98     37348
```

Use **SMOTE** since it's a data imbalance situation

Random Forest: Since I think it's pretty expert-like decision making problem

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     37319
           1       0.00      0.00      0.00        29

    accuracy                           1.00     37348
   macro avg       0.50      0.50      0.50     37348
weighted avg       1.00      1.00      1.00     37348
```

As we can see, the support of label 1 is low → data imbalance
So I use **SMOTE**(Synthetic Minority Oversampling Technique)

```
Random Forest Classification Report (with SMOTE and undersampling):
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     37319
           1       0.02      0.17      0.04        29

    accuracy                           0.99     37348
   macro avg       0.51      0.58      0.52     37348
weighted avg       1.00      0.99      1.00     37348
```

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Define oversampling and undersampling strategies
over = SMOTE(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=0.5)

# Create a pipeline
steps = [('o', over), ('u', under), ('model', RandomForestClassifier(
pipeline = Pipeline(steps=steps)

# Fit the model
pipeline.fit(X_train, y_train)

# Make predictions
y_pred = pipeline.predict(X_test)

# Evaluate the model
print("Random Forest Classification Report (with SMOTE and undersampl
print(classification_report(y_test, y_pred))
df
```

Q4. Can you find the important features, informative features, or coefficient values? (Note: the answer will depend on your selected machine learning model.)

Directly use the code below to get the importances, and we've got coefficient before.

```
feature_importances =
pd.Series(model.feature_importances_,index=X.columns).sort_values(ascending=False)
```

```
metric1    0.339165
metric6    0.264325
metric4    0.090679
metric5    0.087963
metric2    0.085172
metric8    0.051276
metric7    0.040269
metric9    0.029875
metric3    0.011275
```

Q5. There are two data from two devices, please predict the corresponding failure values.

```
new_data = {
    'metric1': [127175526, 4527376],
    'metric2': [4109.433, 0],
    'metric3': [3.90566, 0],
    'metric4': [54.63208, 3],
    'metric5': [15.46226, 24],
    'metric6': [258303.5, 0],
    'metric7': [30.62264, 0],
    'metric8': [30.62264, 0],
    'metric9': [23.08491, 0]
}

new_df = pd.DataFrame(new_data)

# Load the saved model
log_reg_model = joblib.load('log_reg_model.pkl')

# Make predictions for the new data
new_predictions = log_reg_model.predict(new_df)
print("Predictions for new data:", new_predictions)
```

```
Predictions for new data: [1 0]
```

Dataset 2: Insurance Prediction

Q1. Which kind of data selection method will you use to split csv data to training and testing datasets? sequential or random? WHY?

Random (80% train, 20% validation) for sure, my own preference and not to mention that it’s shown that random is generally a better option.

Q2. In class, we learned many model evaluation methods, such as confusion matrix, accuracy score, precision score, recall score, and so on. In addition to the confusion matrix and accuracy score, which must be used in the Q3 and Q4, if you were to choose two evaluation metrics/scores, which two would you choose? Why?

I would directly use the classification report provided by scikit learn, which would give us precision, recall, and f1-score. It's my own preference to use classification report for classification tasks.

Q3. Please use eight classification models taught in the class and find their own best parameters' settings.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score, classification_report
import joblib

# initialize
models = {
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'XGBoost': xgb.XGBClassifier(random_state=42, use_label_encoder=False,
eval_metric='logloss'),
    'SVC': SVC(random_state=42),
    'KNN': KNeighborsClassifier(),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=10000),
    'Gaussian NB': GaussianNB(),
    'Multinomial NB': MultinomialNB()
}
results = {}
best_model_name = None
best_accuracy = 0
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)

    # evaluation matrix
    cm = confusion_matrix(y_val, y_pred)
    accuracy = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred)
    recall = recall_score(y_val, y_pred)

    # print the result
    print(f"Model: {name}")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
```

```python
    print(f"Recall: {recall:.4f}")
    print(classification_report(y_val, y_pred))
    print("="*60)

    results[name] = {
        'Confusion Matrix': cm,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'Classification Report': classification_report(y_val, y_pred,
output_dict=True)
    }

    # check the best model
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model_name = name
        best_model = model
```

There is no way to find the 'best' setting. I just set them with my own preference.

Q4. Which prediction model is the best between these eight classification models? WHY?

```python
# save the result into txt file
with open('model_evaluation_results.txt', 'w') as f:
    for name, result in results.items():
        f.write(f"Model: {name}\n")
        f.write(f"Confusion Matrix:\n{result['Confusion Matrix']}\n")
        f.write(f"Accuracy: {result['Accuracy']:.4f}\n")
        f.write(f"Precision: {result['Precision']:.4f}\n")
        f.write(f"Recall: {result['Recall']:.4f}\n")
        f.write(f"Classification Report:\n{result['Classification Rep
        f.write("="*60 + "\n")

# save the best model
joblib.dump(best_model, 'best_model.joblib')

print(f"Best model: {best_model_name} with accuracy: {best_accuracy:.
print("Model evaluation results saved to 'model_evaluation_results.tx
print("Best model saved to 'best_model.joblib'")
```

✓ 0.0s Python

```
Best model: XGBoost with accuracy: 0.8971
Model evaluation results saved to 'model_evaluation_results.txt'
Best model saved to 'best_model.joblib'
```

As we can see, **XGBoost** is the best model.
XGBoost is particularly strong in classification tasks due to several key features and optimizations: **Gradient Boosting Framework** (sequentially builds and combines weak learners (usually decision trees)), **Regularization**, **Tree Pruning**...

Q5. Insurance_validation.csv is a validation dataset. Please use your best prediction model to get the "Response" and output the results to a csv file.

```
id,Response
57782,0
286811,0
117823,0
213992,0
```

Dataset 3: Store Sale Prediction

Q. Please predict the results of the data in validations.csv, output the results to a csv file, and record and explain all the steps/processes.

Step1: load all of the csv files

Step2: Change the date into datetime

```
# Ensure 'date' columns are in datetime format
dataset['date'] = pd.to_datetime(dataset['date'])
holidays_events['date'] = pd.to_datetime(holidays_events['date'])
oil['date'] = pd.to_datetime(oil['date'])
transactions['date'] = pd.to_datetime(transactions['date'])
validation['date'] = pd.to_datetime(validation['date'])
```

Step3: Merge the dataset and drop the duplicated ones

```
# Merge datasets with suffixes to avoid column name conflicts
dataset = pd.merge(dataset, oil, on='date', how='left', suffixes=('', '_oil'))
dataset = pd.merge(dataset, holidays_events, on='date', how='left', suffixes=('',
'_holiday'))
dataset = pd.merge(dataset, stores, on='store_nbr', how='left', suffixes=('',
'_store'))
dataset = pd.merge(dataset, transactions, on=['date', 'store_nbr'], how='left',
suffixes=('', '_transaction'))

# Handle column conflicts by dropping or renaming columns
columns_to_drop = [
    'dcoilwtico_oil', 'type_holiday', 'locale_holiday', 'locale_name_holiday',
'description_holiday', 'transferred_holiday',
    'type_store', 'locale_store', 'locale_name_store', 'description_store',
'transferred_store',
    'type_transaction', 'locale_transaction', 'locale_name_transaction',
'description_transaction', 'transferred_transaction'
]
```

```
dataset.drop(columns=[col for col in columns_to_drop if col in dataset.columns],
inplace=True)
```

Step4: Do the same thing on validation set

Step5: Define the features and target

```
# Define features and target
features = ['store_nbr', 'family', 'onpromotion', 'dcoilwtico', 'transactions',
'year', 'month', 'day', 'dayofweek']
X = dataset[features]
y = dataset['sales']

# One-hot encoding for categorical variables
X = pd.get_dummies(X, columns=['family'])

# Split the data into training and validation sets
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step6: Training (Random Forest)

```
# Train a Random Forest model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Validate the model
from sklearn.metrics import mean_squared_error
y_pred = model.predict(X_val)
print('Validation RMSE:', mean_squared_error(y_val, y_pred, squared=False))
```

Validation RMSE: 188.73371895227075

Step7: Analysis

```
# Calculate the mean sales in the training set
baseline_sales = np.mean(y_train)

# Predict the validation set using the baseline sales
baseline_predictions = np.full_like(y_val, baseline_sales)

# Calculate the RMSE for the baseline model
baseline_rmse = mean_squared_error(y_val, baseline_predictions, squared=False)
print('Baseline RMSE:', baseline_rmse)
```

Baseline RMSE: 1093.747755347675 → much larger than Validation RMSE

Step7: Save the Prediction csv file