ware-defect-prediction240159146175

April 1, 2024

[1]:
```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'software-defect-prediction:https%3A%2F%2Fstorage.
 ↪googleapis.com%2Fkaggle-data-sets%2F80237%2F186575%2Fbundle%2Farchive.
 ↪zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-1
 ↪iam.gserviceaccount.
 ↪com%252F20240401%252Fauto%252Fstorage%252Fgoog4_request%26X-Goog-Date%3D20240401T161145Z%26

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'),␣
 ↪target_is_directory=True)
except FileExistsError:
```

```
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'),␣
 ↪target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes␣
 ↪downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path␣
 ↪{destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading software-defect-prediction, 566430 bytes compressed
[==================================================] 566430 bytes downloaded
Downloaded and uncompressed: software-defect-prediction
Data source import complete.
```

Software Defect Prediction

IMPORT LIBRARIES

[2]:
```
import numpy as np
import pandas as pd
!pip install -q fasteda
!pip install -q rich
from rich import print as rich_print
from fasteda import fast_eda
pd.set_option('display.max_columns', None)
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
from scipy.stats import skew
from xgboost import XGBClassifier

from gc import collect
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample

from lightgbm import LGBMClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV,␣
 ↪train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier

sns.set(style="whitegrid")

import warnings
warnings.filterwarnings('ignore')

import lightgbm as lgb
```

```
  Preparing metadata (setup.py) … done
  Building wheel for fasteda (setup.py) … done
```

read data

[5]:
```
df = pd.read_csv('/kaggle/input/s3e23-data-creation/df.csv')
test = pd.read_csv('/kaggle/input/s3e23-data-creation/test.csv')
sub = pd.read_csv('/kaggle/input/playground-series-s3e23/sample_submission.csv')
ids = pd.read_csv(r'/kaggle/input/s3e23-data-creation/ids.csv')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
```

```
<ipython-input-5-78bb066e41b1> in <cell line: 1>()
----> 1 df = pd.read_csv('/kaggle/input/s3e23-data-creation/df.csv')
      2 test = pd.read_csv('/kaggle/input/s3e23-data-creation/test.csv')
      3 sub = pd.read_csv('/kaggle/input/playground-series-s3e23/
 ↪sample_submission.csv')
      4 ids = pd.read_csv(r'/kaggle/input/s3e23-data-creation/ids.csv')

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in␣
 ↪wrapper(*args, **kwargs)
    209                 else:
    210                     kwargs[new_arg_name] = new_arg_value
--> 211             return func(*args, **kwargs)
    212
    213         return cast(F, wrapper)

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in␣
 ↪wrapper(*args, **kwargs)
    329                     stacklevel=find_stack_level(),
    330                 )
--> 331             return func(*args, **kwargs)
    332
    333         # error: "Callable[[VarArg(Any), KwArg(Any)], Any]" has no

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
 ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
 ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates␣
 ↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
 ↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
 ↪quotechar, quoting, doublequote, escapechar, comment, encoding,␣
 ↪encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,␣
 ↪delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    948     kwds.update(kwds_defaults)
    949
--> 950     return _read(filepath_or_buffer, kwds)
    951
    952


/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py in␣
 ↪_read(filepath_or_buffer, kwds)
    603
    604     # Create the parser.
--> 605     parser = TextFileReader(filepath_or_buffer, **kwds)
    606
    607     if chunksize or iterator:

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py in␣
 ↪__init__(self, f, engine, **kwds)
   1440
```

```
   1441         self.handles: IOHandles | None = None
-> 1442         self._engine = self._make_engine(f, self.engine)
   1443
   1444     def close(self) -> None:

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py in␣
 ↪_make_engine(self, f, engine)
   1733                 if "b" not in mode:
   1734                     mode += "b"
-> 1735             self.handles = get_handle(
   1736                 f,
   1737                 mode,

/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in␣
 ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
 ↪errors, storage_options)
    854         if ioargs.encoding and "b" not in ioargs.mode:
    855             # Encoding
--> 856             handle = open(
    857                 handle,
    858                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '/kaggle/input/
 ↪s3e23-data-creation/df.csv'
```

[4]:
```
df.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-c42a15b2c7cf> in <cell line: 1>()
----> 1 df.head()

NameError: name 'df' is not defined
```

[ ]:
```
test.head()
```

[ ]:
```
df.columns
```

feature engine

modeling

[ ]:
```
X = df.drop(columns=['defects','id'])
y = df['defects']
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[ ]:
```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Function for grid search with cross-validation
def perform_grid_search(model, param_grid, X_train, y_train):
    grid_search = GridSearchCV(model, param_grid, cv=cv, scoring='roc_auc',␣
↪n_jobs=-1)
    grid_search.fit(X_train, y_train)
    return grid_search.best_estimator_


def calculate_roc_auc(model, X, y):
    y_pred = model.predict_proba(X)[:, 1]
    roc_auc = roc_auc_score(y, y_pred)
    return roc_auc
```

[ ]:
```
xgb_param_grid = {
    'learning_rate': [0.01],
    'max_depth': [ 7],
    'n_estimators': [50, 200,500,700],
}

# Updated LightGBM parameter grid
lgb_param_grid = {
    'learning_rate': [0.01],
    'num_leaves': [15, 31, 50, 100],
    'n_estimators': [500],
}
```

[ ]:
```
best_xgboost = perform_grid_search(XGBClassifier(random_state=42),␣
↪xgb_param_grid, X_train, y_train)
print("Best XGBoost Parameters:")
print(best_xgboost)
best_lgb = perform_grid_search(lgb.LGBMClassifier(random_state=42),␣
↪lgb_param_grid, X_train, y_train)
print("\nBest LightGBM Parameters:")
print(best_lgb)
```

[ ]:
```
roc_auc_xgboost = calculate_roc_auc(best_xgboost, X_test, y_test)
roc_auc_lgb = calculate_roc_auc(best_lgb, X_test, y_test)

print(f'XGBoost ROC-AUC: {roc_auc_xgboost}')
print(f'LightGBM ROC-AUC: {roc_auc_lgb}')
```

[ ]:
```
ensemble_model = RandomForestClassifier(random_state=42)
ensemble_model.fit(np.column_stack((best_xgboost.predict_proba(X_test)[:, 1],␣
 ↪best_lgb.predict_proba(X_test)[:, 1])), y_test)

ensemble_roc_auc = calculate_roc_auc(ensemble_model, np.
 ↪column_stack((best_xgboost.predict_proba(X_test)[:, 1], best_lgb.
 ↪predict_proba(X_test)[:, 1])), y_test)
print(f'Ensemble ROC-AUC: {ensemble_roc_auc}')
```

[ ]:
```
models = {
    "best_lgb": best_lgb,
    "best_xgboost": best_xgboost,
    "ensemble_model": ensemble_model
}

roc_auc_scores = {
    "ensemble_model": ensemble_roc_auc,
    "best_xgboost": roc_auc_xgboost,
    "best_lgb": roc_auc_lgb
}


# Find the model with the highest ROC AUC score
best_model_name = max(roc_auc_scores, key=roc_auc_scores.get)

# Get the best model object
best_model = models[best_model_name]
best_model_name
```

[ ]:
```
test_main = test.copy()
```

[ ]:
```
test_main['prob_xgboost'] = best_xgboost.predict_proba(test_main)[:, 1]
```

[ ]:
```
test_main['prob_lgb'] = best_lgb.predict_proba(test)[:, 1]
```

[ ]:
```
# Get the probability estimates from the ensemble model
ensemble_probabilities = ensemble_model.predict_proba(np.
 ↪column_stack((best_xgboost.predict_proba(test)[:, 1], best_lgb.
 ↪predict_proba(test)[:, 1])))

test_main['prob_ensemble'] = ensemble_probabilities[:, 1]
test_main.head()
```

[ ]:
```
sub['id'] = ids['id'].values
sub['defects'] = test_main['prob_xgboost']
sub.to_csv('submission_xbg.csv',index=False)
sub['defects'] = test_main['prob_ensemble']
```

```
sub.to_csv('submission_ensemble.csv',index=False)
sub['defects'] = test_main['prob_lgb']
sub.to_csv('submission_lgb.csv',index=False)
```

[ ]: